UNITED TECHNOLOGIES MOSTEK

MICROCOMPUTER COMPONENTS

1982/83 Z80 DESIGNERS GUIDE

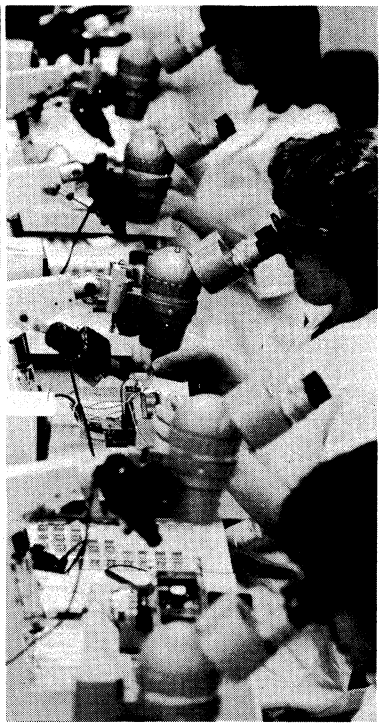
1982/1983
Z80 DESIGNERS GUIDE

PRINTED IN USA June 1982

1982/1983 Z80 DESIGNERS GUIDE

TABLE OF CONTENTS

1982/1983 Z80 DESIGNERS GUIDE



TECHNOLOGY

From its beginning, Mostek has been an innovator. From the developments of the 1K dynamic RAM and the single-chip calculator in 1970 to the current 64K dynamic RAM, Mostek technological breakthroughs have proved the benefits and cost-effectiveness of metal oxide semiconductors. Today, Mostek represents one of the industry's most productive bases of MOS/LSI technology, including Direct-Step-on-Wafer processing and laser implemented redundant circuitry.

The addition of the Microelectronics Research Center in Colorado Springs adds a new dimension to Mostek circuit design capabilities. Using the latest computer-aided design techniques, center engineers will be keeping ahead of the future with new technologies and processes.

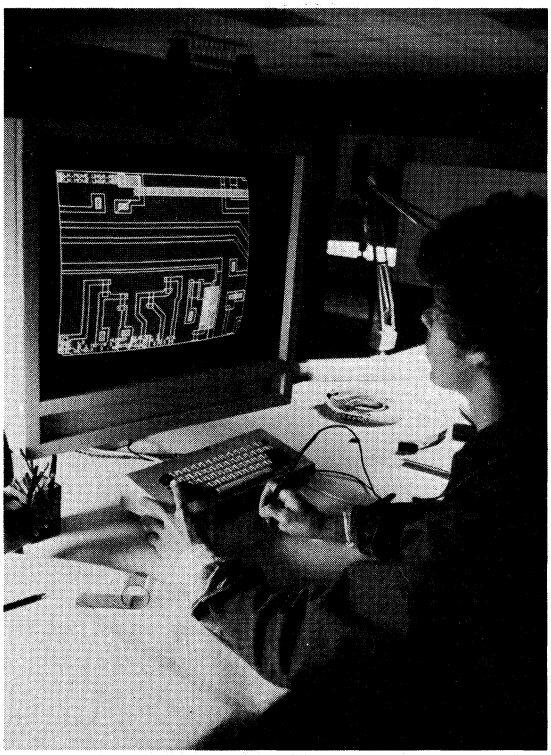
QUALITY

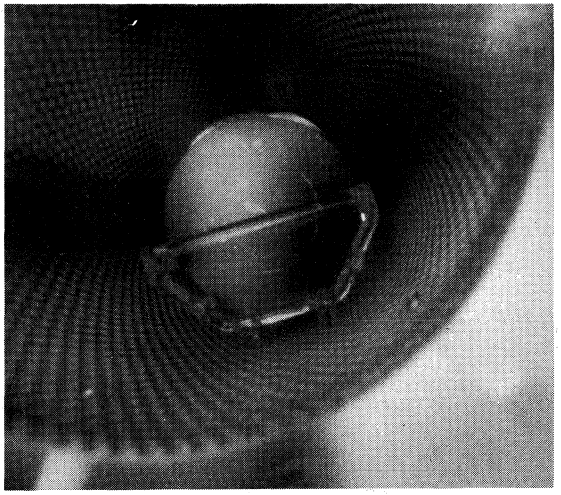
The worth of a product is measured by how well it is designed, manufactured and tested and by how well it works in your system.

In design, production and testing, the Mostek goal is meeting specifications the first time on every product. This goal requires a collective discipline from the company as well as individual efforts. Discipline, coupled with very personal pride, has enabled Mostek to build in quality at every level of production.

PRODUCTION CAPABILITY

The commitment to increasing production capability has made Mostek the world's largest manufacturer of dynamic RAMs. We entered the telecommunications market in 1974 with a tone dialer, and have shipped millions of telecom circuits since then. Millions of our MK3870 single-chip microcomputers are in use throughout the world. Recent construction in Dallas, Ireland and Colorado Springs has added some 50 percent to the Mostek manufacturing capacity.





THE PRODUCTS

Telecommunications Products

Mostek is the leading supplier of tone dialers, pulse dialers, and CODEC devices. As each new generation of telecommunications systems emerges, Mostek is ready with new generation components, including PCM filters, tone decoders, repertory dialers, new integrated tone dialers, and pulse dialers.

These products, many of them using CMOS technology, represent the most modern advancements in telecommunications component design.

Industrial Products

Mostek's line of Industrial Products offers a high degree of versatility per device. This family of components includes various microprocessor-compatible A/D converters, a counter/time-base circuit for the division of clock signals, and combined counter/display decoders. As a result of the low parts count involved, an economical alternative to discrete logic systems is provided.

Memory Products

Through innovations in both circuit design, wafer processing and production, Mostek has become the industry's leading supplier of dynamic RAMs.

Examples of Mostek leadership are families of x1 and x8 high performance static RAMs and our extremely successful 64K ROMs with more codes processed than any other mask-ROM in the industry. Another performance and density milestone is our 256K ROM, the MK38000. In MOS Dynamic RAMs, Mostek led the way as the world's leading supplier of 16K devices.

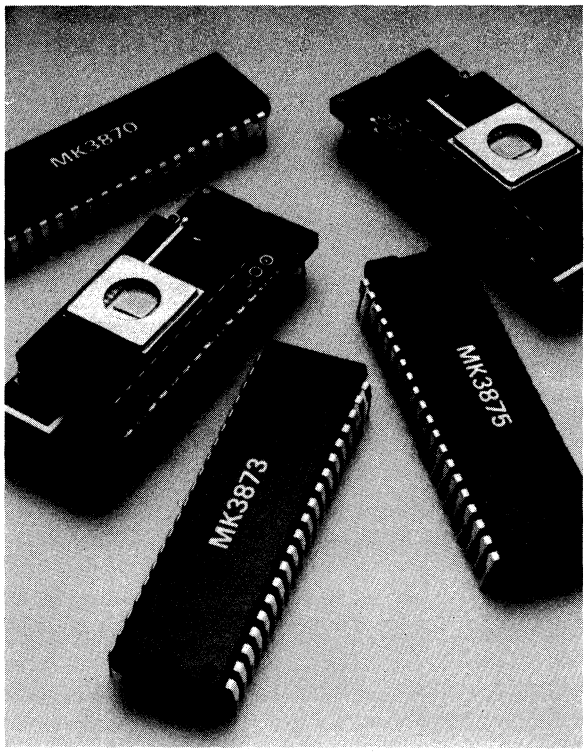
Our MK4564 64K dynamic RAM uses advanced circuit techniques and design to enhance manufacturability to satisfy the demands of a huge marketplace.

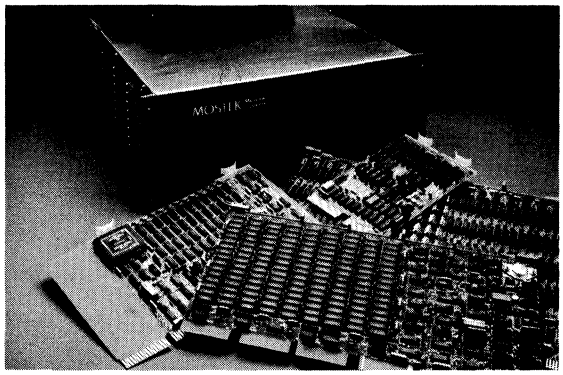
Microcomputer Components

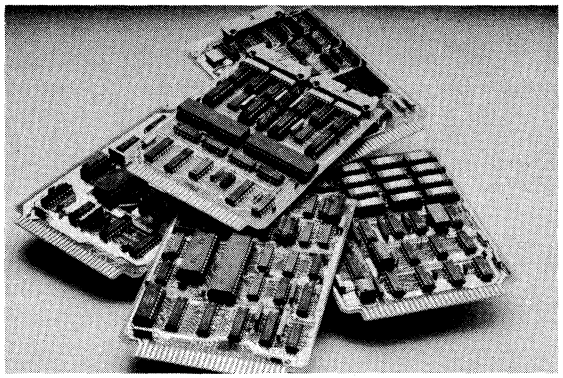
Mostek's microcomputer components cover the entire spectrum of microcomputer applications.

Our MK68000 16-bit microprocessor is designed for high-performance, memory-intensive systems.

Our Z80 is today's industry-standard 8-bit microprocessor. The Mostek 3870 family of single-chip microcomputers offers upgrade options in ROM, RAM, and I/O—all in the same socket. The MK38P70 EPROM piggyback microcomputer emulates the entire family and is ideal for low-volume applications.







Development systems include the RADIUS™ remote development station that lets you use your host minicomputer to develop the applications software. The program is then downloaded into the RADIUS which then lets you perform real-time in-circuit emulation and debug. The Mostek Matrix™ Development System is a stand-alone hardware and software debug and integration system.

Microcomputer Systems

Mostek is the world's leading manufacturer of Z80-based STD BUS system components. A new line of microsystems utilizing the VME BUS and based on the MK68000 will be available soon.

Computer systems include our Matrix line which utilize STD BUS cards to let you custom-design your own system.

Military Products

An extension of the high quality in fabrication and design inherent in Mostek's product line allows many of our ICs to be made available screened to MIL-STD-883. In addition, select parts are qualified to the rigors of MIL-M-38510 and are processed on our QPL certified lines.

The MKB product line begins with the complete Memory Products offering, and extends into microprocessors and gate arrays. Leadless Chip Carrier packaging and prepared customer SCDs address the particular needs of the military community.

Memory Systems

Taking full advantage of our leadership in memory components technology, Mostek Memory Systems offers a broad line of products, all with the performance and reliability to match our industry-standard circuits. Mostek Memory Systems offers add-in memory boards for popular DEC, Data General, and Perkin-Elmer minicomputers.

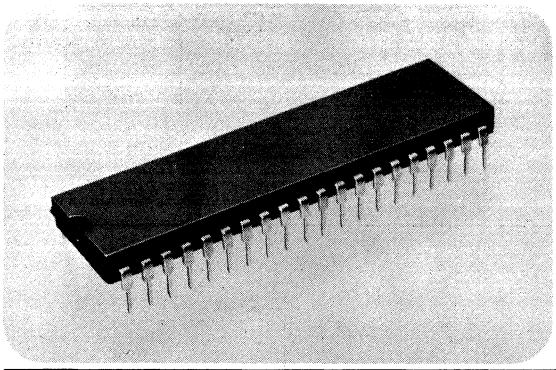
Mostek also offers special purpose and custom memory boards for special applications.

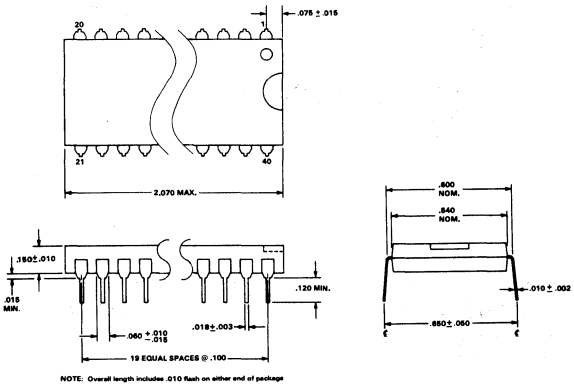
Gate Arrays

Utilizing the technology developed by United Technologies Microelectronic Research Center, Mostek plans to market custom gate array circuits in the second half of 1982.

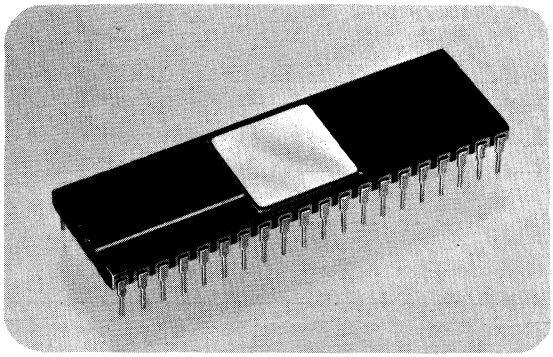
Package Descriptions

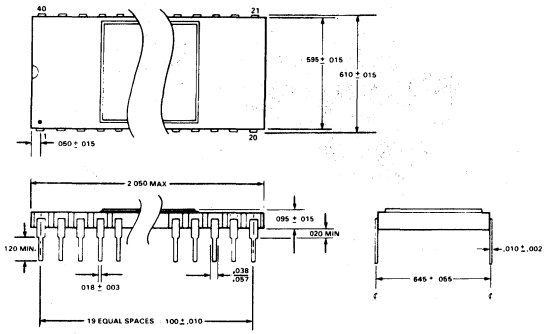
Plastic Dual-In-Line Package (N) 40 Pin



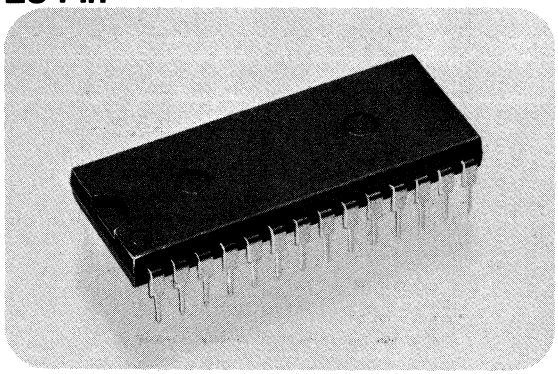


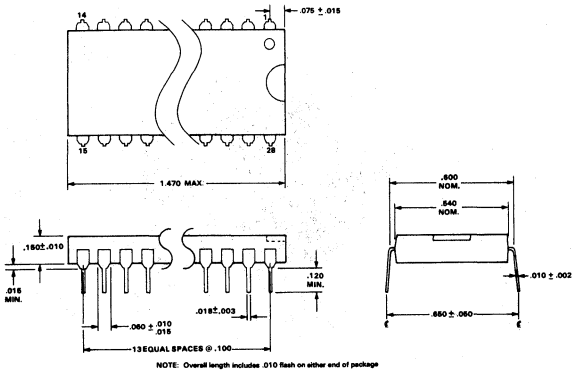
Ceramic Dual-In-Line Package (P) 40 Pin



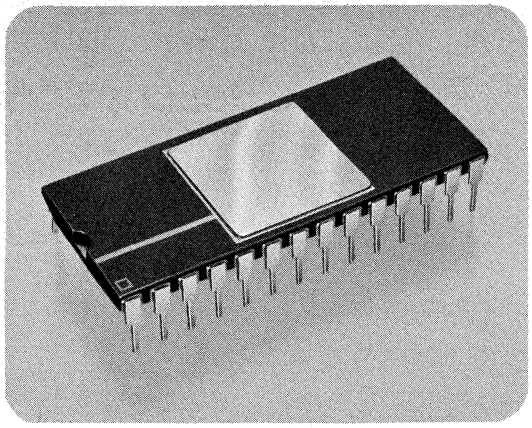


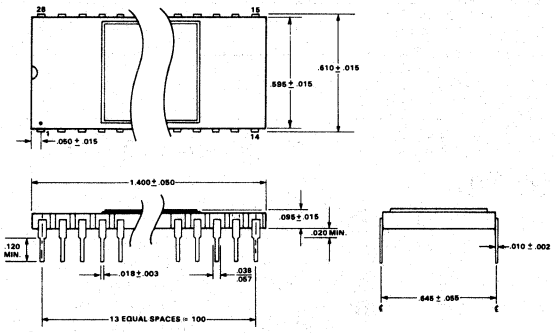
Plastic Dual-In-Line Package (N) 28 Pin



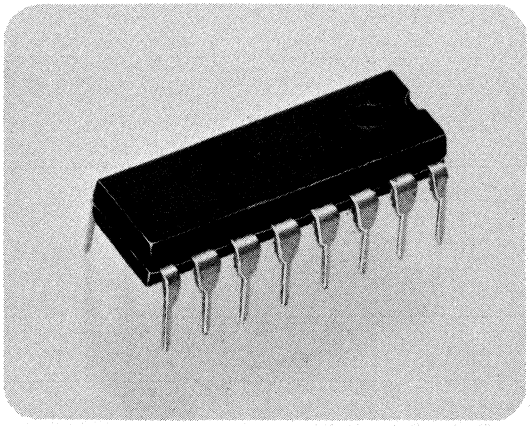


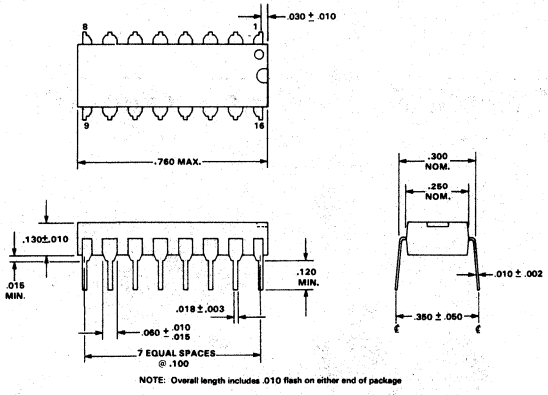
Ceramic Dual-In-Line Package (P)
28 Pin





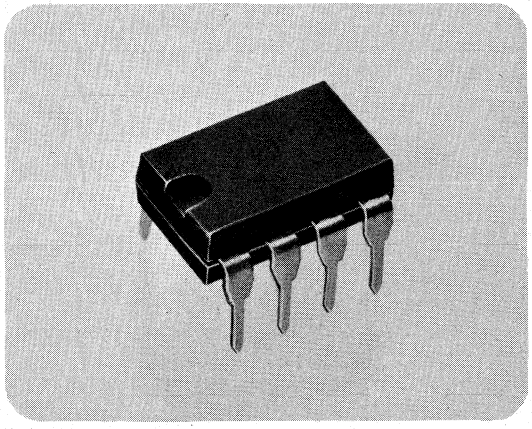
Plastic Dual-In-Line Package (N)
16 Pin

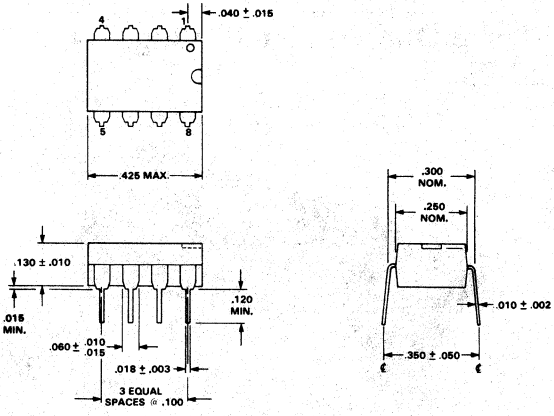




NOTE: Overall length includes .010 flash on either end of package

Plastic Dual-In-Line Package (N)
8 Pin





NOTE: Overall length includes .010 flash on either end of package

ORDERING INFORMATION

Factory orders for parts described in this book should include a four-part number as explained below:

Example: MK4167P-3

1. Dash Number
2. Package
3. Device Number
4. Mostek Prefix

1. Dash Number

   One or two numerical characters defining specific device performance characteristics and operating temperature range.

2. Package

   P - Gold side-brazed ceramic DIP
   N - Epoxy DIP (Plastic)
   K - Tin side-brazed ceramic DIP
   T - Ceramic DIP with transparent lid
   E - Ceramic leadless chip carrier
   D - Dual density RAM-PAC
   F - Flat pack

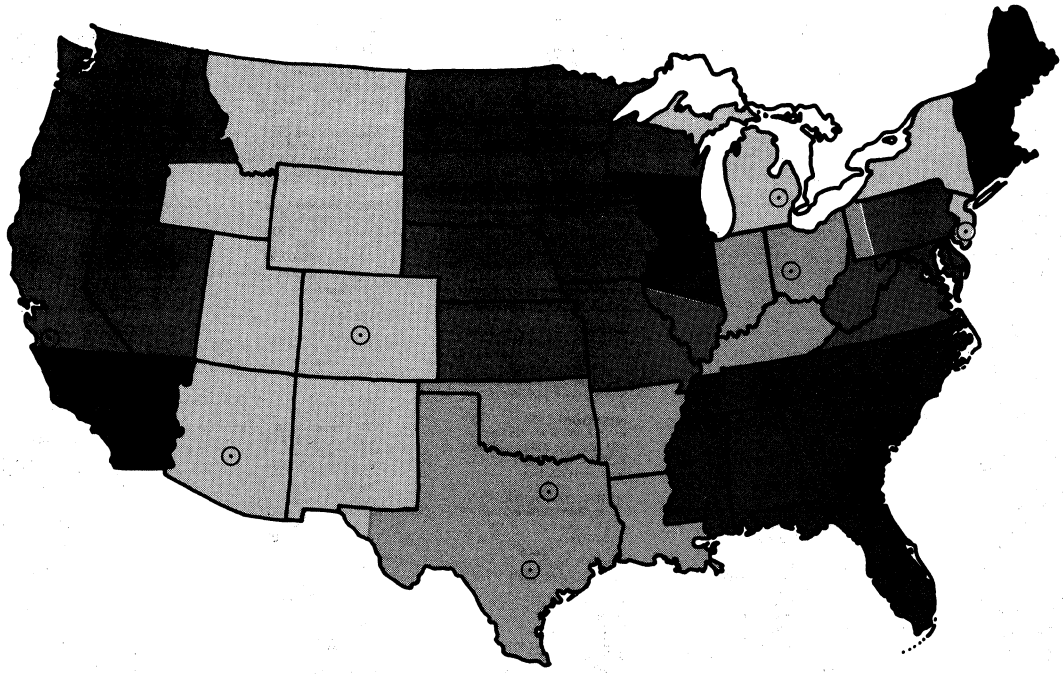
3. Device number

   1XXX or 1XXXX - Shift Register, ROM
   2XXX or 2XXXX - ROM, EPROM
   3XXX or 3XXXX - ROM, EPROM
   38XX - Microcomputer Components
   4XXX or 4XXXX - RAM
   5XXX or 5XXXX - Counters, Telecommunication and Industrial
   7XXX or 7XXXX - Microcomputer Systems

4. Mostek Prefix

   MK - Standard Prefix

U.S. AND CANADIAN SALES OFFICES



CORPORATE HEADQUARTERS

Mostek Corporation
1215 W. Crosby Rd.
P. O. Box 169
Carrollton, Texas 75006

REGIONAL OFFICES

Northeastern Area
Mostek
49 W. Putnam, 3rd Floor
Greenwich, Conn. 06830
203/622-0955
TWX 710-579-2928

Northeast U.S.
Mostek
29 Cummings Park, Suite #426
Woburn, Mass. 01801
617/935-0635
TWX 710-348-0459

Southeastern Area
Mostek
4001B Greentree Executive Campus
Route #73
Marlton, New Jersey 08053
609/596-9200
TWX 710-940-0103

Southeast U.S.
Mostek
13907 N. Dale Mabry Highway
Suite 201
Tampa, Florida 33618
813/962-8338
TWX 810-876-4611

Upstate NY Region
Mostek
4651 Crossroads Park Dr., Suite 201
Liverpool, NY 13088
315/457-2160
TWX 710-945-0255

Chicago Region
Mostek
Two Crossroads of Commerce
Suite 360
Rolling Meadows, Ill. 60008
312/577-9870
TWX 910-291-1207

North Central U.S.
Mostek
6101 Green Valley Dr.
Bloomington, Mn. 55438
612/831-2322
TWX 910-576-2802

Michigan
Mostek
Orchard Hill Place
21333 Haggerty Road
Suite 321
Novi, MI 48050
313/348-8360
TWX 810-242-1471

Central U.S.
Mostek
4100 McEwen Road
Suite 151
Dallas, Texas 75234
214/386-9340
TWX 910-860-5437

Southwest Region
Mostek
4100 McEwen Road
Suite 237
Dallas, Texas 75234
214/386-9141
TWX 910-860-5437

Chevy Chase #4
7715 Chevy Chase Dr., Suite 116
Austin, TX 78752
512/458-5226
TWX 910-874-2007

Western Region
Northern California
Mostek
1762 Technology Drive
Suite 126
San Jose, Calif. 95110
408/287-5080
TWX 910-338-2219

Seattle Region
Mostek
1107 North East 45th St.
Suite 411
Seattle, WA 98105
206/632-0245
TWX 910-444-4030

Southern California
Mostek
18004 Skypark Circle
Suite 140
Irvine, Calif. 92714
714/549-0397
TWX 910-595-2513

Arizona Region
Mostek
2150 East Highland Ave.
Suite 101
Phoenix, AZ 85016
602/954-6260
TWX 910-957-4581

U.S. AND CANADIAN REPRESENTATIVES

ALABAMA
Conley & Associates, Inc.
3322 Memorial Pkwy., S.W.
Suite 17
Huntsville, AL 35801
205/882-0316
TWX 810-726-2159

ARIZONA
Summit Sales
7825 E. Redfield Rd.
Scottsdale, AZ 85260
602/998-4850
TWX 910-950-1283

CALIFORNIA
Harvey King, Inc.
8124 Miramar Road
San Diego, CA 92126
714/566-5252
TWX 910-335-1231

COLORADO
Waugaman Associates*
4800 Van Gordon
Wheat Ridge, CO 80033
303/423-1020
TWX 910-938-0750

CONNECTICUT
New England Technical Sales
240 Pomeroy Ave.
Meriden, CT 06450
203/237-8827
TWX 710-461-1126

FLORIDA
Conley & Associates, Inc.*
P.O. Box 309
235 S. Central
Oviedo, FL 32765
305/365-3283
TWX 810-856-3520

Conley & Associates, Inc.
4021 W. Waters
Suite 2
Tampa, FL 33614
813/885-7658
TWX 810-876-9136

Conley & Associates, Inc.
P.O. Box 700
1612 N.W. 2nd Avenue
Boca Raton, FL 33432
305/395-6108
TWX 510-953-7548

GEORGIA
Conley & Associates, Inc.
3951 Pleasantdale Road
Suite 201
Doraville, GA 30340
414/447-6992
TWX 810-766-0488

ILLINOIS
Carlson Electronic Sales*
600 East Higgins Road
Elk Grove Village, IL 60007
312/956-8240
TWX 910-222-1819

INDIANA
Rich Electronic Marketing*
599 Industrial Drive
Carmel, IN 46032
317/844-8462
TWX 810-260-2631

Rich Electronic Marketing
3448 West Taylor St.
Fort Wayne, IN 46804
219/432-5553
TWX 810-332-1404

IOWA
REP Associates
980 Arica Ave.
Marion, IA 52302
319/393-0231

KANSAS
Rush & West Associates*
107 N. Chester Street
Olathe, KN 66061
913/764-2700
Wichita 316/683-0206
TWX 910-749-6404

KENTUCKY
Rich Electronic Marketing
8819 Roman Court
P. O. Box 91147
Louisville, KY 40291
502/499-7808
TWX 810-535-3757

MARYLAND
Arbotek Associates
3600 St. Johns Lane
Ellicott City, MD 21043
301/461-1323
TWX 710-862-1874

MASSACHUSETTS
New England Technical Sales*
135 Cambridge Street
Burlington, MA 01803
617/272-0434
TWX 710-332-0435

Computer Marketing
241 Crescent St./2nd Floor
Waltham, MA 02154
617/894-7000
710-324-1503

MICHIGAN
Action Components
21333 Haggerty Road
Suite 201
Novi, MI 48050
313/349-3940

MINNESOTA
Cahill, Schmitz & Cahill, Inc.*
315 N. Pierce
St. Paul, MN 55104
612/646-7217
TWX 910-563-3737

Micro Resources, Inc.
2700 Chowen Avenue South
Minneapolis, MN 55416

MISSOURI
Rush & West Associates
481 Melanie Meadows Lane
Ballwin, MO 63011
314/394-7271

NORTH CAROLINA
Conley & Associates, Inc.
4050 Wake Forest Road
Suite 102
Raleigh, NC 27609
919/876-9862
TWX 510-928-1829

NEW JERSEY
Tritek Sales, Inc.
21 E. Euclid Ave.
Haddonfield, NJ 08033
609/429-1551
215/627-0149 (Philadelphia Line)
TWX 710-896-0881

NEW MEXICO
Waugaman Associates
P.O. Box 14894
Albuquerque, NM 87111
or
9004 Menaul NE
Suite 7
Albuquerque, NM 87112
505/294-1437
505/294-1436 (Ans. Service)

NEW YORK
ERA Inc.
354 Veterans Memorial Highway
Commack, NY 11725
516/543-0510
TWX 510-226-1485
(New Jersey Phone #
800/645-5500, 5501)

Precision Sales Corp.
5 Arbustus Ln., MR-97
Binghamton, NY 13901
607/648-3686
607/648-8833

Precision Sales Corp.*
1 Commerce Blvd.
Liverpool, NY 13008
315/451-3480
TWX 710-545-0250

Precision Sales Corp.
3594 Monroe Avenue
Pittsford, NY 14534
716/381-2820

Precision Sales Corp.
Drake Road
Pleasant Valley, NY 12569
914/635-3233

OHIO
The Lyons Corp.
4812 Frederick Rd.
Dayton, Ohio 45414
513/278-0714
TWX 810-459-1754

The Lyons Corp.
4615 N. Streetsboro Rd.
Richfield, Ohio 44286
216/659-9224
TWX 810-427-9103

OREGON
Northwest Marketing Assoc.
9999 S.W. Wilshire St.
Suite 124
Portland OR 97225
503/297-2581
TELEX 910-464-5157

TENNESSEE
Conley & Associates, Inc.
1128 Tusculum Blvd.
Suite D
Greenville, TN 37743
615/639-3139
TWX 810-576-4597

UTAH
Waugaman Associates
10332 South 1540 W
South Jordan, UT 84065
801/254-0570
or
801-254-0572
TWX 910-925-4073

WASHINGTON
Northwest Marketing Assoc.*
12835 Bellevue-Redmond Rd.
Suite 203E
Bellevue, WA 98005
206/455-5846
TWX 910-443-2445

WISCONSIN
Carlson Electronic Sales
Northbrook Executive Ctr.
10701 West North Ave.
Suite 209
Milwaukee, WI 53226
414/476-2790
TWX 910-222-1819

CANADA
Cantec Representatives Inc.*
1573 Laperriere Ave.
Ottawa, Ontario
Canada K1Z 7T3
613/725-3704
TWX 610-562-8967

Cantec Representatives Inc.
15 Charles Street, East
Kitchener, Ontario
Canada N2G2P3
519/744-6341
TWX 610-492-2683 (Toronto)

Cantec Representatives Inc.
8 Strathearn Ave, Unit 18
Brampton, Ontario
Canada L6T4L8
416/791-5922
TWX 610-492-2683

Cantec Representatives Inc.
3639 Sources Blvd.
Suite 116
Dollard Des Ormeaux, Quebec
Canada H9B2K4
514/683-6131
TWX 610-422-3985

*Home Office

U.S. AND CANADIAN DISTRIBUTORS

ARIZONA
Kierulff Electronics
4134 E. Wood St.
Phoenix, AZ 85040
602/243-4101
TWX 910/951-1550

Kierulff Electronics
1806 W. Grant Rd.
Suite 102
Tucson, AZ 85705
602/624-9986
TWX 910/952-1119

CALIFORNIA
Arrow Electronics
4029 Westerly Place
Bldg. 15, Unit 113
Newport Beach, CA 92660
(714) 851-8961
TWX 910/595-2861

Arrow Electronics
19748 Dearborn St.
Chatsworth, CA 91311
(213) 701-7500
TWX 910-493-2086

Arrow Electronics
9511 Ridgehaven Court
San Diego, CA 92123
(714) 565-4800
TWX 910/335-1195

Arrow Electronics
521 Weddell Dr.
Sunnyvale, CA 94086
408/745-6600
TWX 910/339-9371

Kierulff Electronics
2585 Commerce Way
Los Angeles, CA 90040
213/725-0325
TWX 910/580-3106

Kierulff Electronics
3969 E. Bayshore Rd.
Palo Alto, CA 94303
415/968-6292

Kierulff Electronics
8797 Balboa Avenue
San Diego, CA 92123
714/278-2112
TWX 910/335-1182

Kierulff Electronics
14101 Franklin Avenue
Tustin, CA 92680
714/731-5711
TWX 910/595-2599

Schweber Electronics
17811 Gillette Avenue
Irvine, CA 92714
714/556-3880
TWX 910/595-1720

Schweber Electronics
3110 Patrick Henry Dr.
Santa Clara, CA 95050
408/496-0200

Zeus/West, Inc.
1130 Hawk Circle
Anaheim, CA 92807
714/632-6880
TWX 910/591-1961

COLORADO
Arrow Electronics
2121 South Hudson St.
Denver, CO 80222
(303) 758-2100
TWX 910/931-0552

Kierulff Electronics
10890 E. 47th Avenue
Denver, CO 80239
303/371-6500
TWX 910/932-0169

CONNECTICUT
Arrow Electronics
12 Beaumont Rd.
Wallingford, CT 06492
203/265-7741
TWX 710/476-0162

Schweber Electronics
Finance Drive
Commerce Industrial Park
Danbury, CT 06810
203/792-3500
TWX 710/456-9405

FLORIDA
Arrow Electronics
1001 N.W. 62nd St.
Suite 108
Ft. Lauderdale, FL 33309
305/776-7790
TWX 510/955-9456

Arrow Electronics
50 Woodlake Dr.
Palm Bay, FL 32905
305/725-1480
TWX 510-959-6337

Diplomat Southland
2120 Calumet
Clearwater, FL 33515
813/443-4514
TWX 810/866-0436

Kierulff Electronics
3247 Tech Drive
St. Petersburg, FL 33702
813/576-1966
TWX 810/863-5625

GEORGIA
Arrow Electronics
2979 Pacific Drive
Norcross, GA 30071
404/449-8252
TWX 810/766-0439

Schweber Electronics
303 Research Drive, Suite 210
Norcross, GA 30092
404/449-9170

ILLINOIS
Arrow Electronics
492 Lunt Avenue
P. O. Box 94248
Schaumburg, IL 60193
312/893-9420
TWX 910/291-3544

Kierulff Electronics
1536 Landmeier Rd.
Elk Grove Village, IL 60007
312/640-0200
TWX 910/222-0351

Schweber Electronics
904 Cambridge Dr.
Elk Grove Village, IL 60007
312/364-3750

INDIANA
Advent Electronics
8446 Moller
Indianapolis, IN 46268
317/872-4910
TWX 810/341-3228

Arrow Electronics
2718 Rand Road
Indianapolis, IN 46241
317/243-9353
TWX 810/341-3119

Pioneer Electronics
6408 Castleplace Drive
Indianapolis, IN 46250
317/849-7300
TWX 810/260-1794

IOWA
Advent Electronics
682 58th Avenue
Court South West
Cedar Rapids, IA 52404
319/363-0221
TWX 910/525-1337

Arrow Electronics
1930 St. Andrews Dr., NE
Cedar Rapids, IA 52402
(319) 395-7230

MARYLAND
Arrow Electronics
4801 Benson Avenue
Baltimore, MD 21227
301/247-5200
TWX 710/236-9005

Pioneer Electronics
9100 Gaither Road
Gaithersburg, MD 20877
301/948-0710
TWX 710/828-0545

Schweber Electronics
9218 Gaither Rd.
Gaithersburg, MD 20877
301/840-5900
TWX 710/828-9749

MASSACHUSETTS
Arrow Electronics
Arrow Drive
Woburn, MA 01801
617/933-8130
TWX 710/393-6770

Kierulff Electronics
13 Fortune Drive
Billerica, MA 01865
617/935-5134
TWX 710/390-1449

Lionex Corporation
1 North Avenue
Burlington, MA 01803
617/272-1660
TWX 710/332-1387

Schweber Electronics
25 Wiggins Avenue
Bedford, MA 01730
617/275-5100
TWX 710/326-0268

Zeus/New England, Inc.
25 Adams Street
Burlington, MA 01803
617/273-0753
TWX 710/322-0716

MICHIGAN
Arrow Electronics
3810 Varsity Drive
Ann Arbor, MI 48104
313/971-8220
TWX 810/223-6020

Pioneer Electronics
13485 Stamford
Livonia, MI 48150
313/525-1800
TWX 810/242-3271

Schweber Electronics
10260 Hubbard Ave.
Livonia, MI 48150
313/525-8100
TWX 810/242-2983

MINNESOTA
Arrow Electronics
5230 W. 73rd Street
Edina, MN 55435
612/830-1800
TWX 910/576-3125

Kierulff Electronics
7667 Cahill Rd.
Edina, MN 55435
612/941-7500
TWX 910/576-2721

MISSOURI
Arrow Electronics
2380 Schuetz Road
St. Louis, MO 63141
314/567-6888
TWX 910/764-0882

Olive Electronics
9910 Page Blvd.
St. Louis, MO 63132
314/426-4500
TWX 910/763-0720

Semiconductor Spec
3805 N. Oak Trafficway
Kansas City, MO 64116
816/452-3900
TWX 910/771-2114

NEW HAMPSHIRE
Arrow Electronics
1 Perimeter Rd.
Manchester, NH 03103
603/668-6968
TWX 710/220-1684

NEW JERSEY
Arrow Electronics
Pleasant Valley Avenue
Morrestown, NJ 08057
609/235-1900
TWX 710/897-0829

Arrow Electronics
285 Midland Avenue
Saddlebrook, NJ 07662
201/797-5800
TWX 710/988-2206

Kierulff Electronics
3 Edison Place
Fairfield, NJ 07006
201/575-6750
TWX 710/734-4372

Schweber Electronics
18 Madison Road
Fairfield, NJ 07006
201/227-7880
TWX 710/734-4305

U.S. AND CANADIAN DISTRIBUTORS

NEW MEXICO
Arrow Electronics
2460 Alamo Ave. S.E.
Albuquerque, NM 87106
505/243-4566
TWX 910/989-1679

NEW YORK
Add Electronic
7 Adler Drive
E. Syracuse, NY 13057
315/437-0300

Arrow Electronics
900 Broad Hollow Rd.
Farmingdale, L.I., NY 11735
516/694-6800
TWX 510/224-6494

Arrow Electronics
7705 Maltlage Drive
P. O. Box 370
Liverpool, NY 13088
315/652-1000
TWX 710/545-0230

Arrow Electronics
3000 S. Winton Road
Rochester, NY 14623
716/275-0300
TWX 510/253-4766

Arrow Electronics
20 Oser Ave.
Hauppauge, NY 11787
516/231-1000
TWX 510/227-6623

Lionex Corporation
400 Oser Ave.
Hauppauge, NY 11787
516/273-1660
TWX 510/227-1042

Schweber Electronics
3 Town Line Circle
Rochester, NY 14623
716/424-2222

Schweber Electronics
Jericho Turnpike
Westbury, NY 11590
516/334-7474
TWX 510/222-3660

Zeus/Long Island
401 Broad Hollow Rd.
Melville, NY 11746
516/752-9551
TWX 710/567-1248

Zeus Components Components, Inc.
100 Midland Avenue
Port Chester, NY 10573
914/937-7400
TWX 710/567-1248

NORTH CAROLINA
Arrow Electronics
938 Burke St.
Winston Salem, NC 27102
919/725-8711
TWX 510/931-3169

Arrow Electronics
3117 Poplarwood Court
Suite 123, P.O. Box 95163
Raleigh, NC 27625
TWX 919/876-3132

Hammond Electronics
2923 Pacific Avenue
Greensboro, NC 27406
919/275-6391
TWX 510/925-1094

OHIO
Arrow Electronics
7620 McEwen Road
Centerville, OH 45459
513/435-5563
TWX 810/459-1611

Arrow Electronics
6238 Cochran Road
Solon, OH 44139
216/248-3990
TWX 810/427-9409

Pioneer Electronics
4800 East 131st Street
Cleveland, OH 44105
216/587-3600
TWX 810/422-2211

Pioneer Electronics
4433 Interpoint Blvd.
Dayton, OH 45424
513/236-9900
TWX 810/459-1622

Schweber Electronics
23880 Commerce Park Road
Beachwood, OH 44122
216/464-2970
TWX 810/427-9441

OKLAHOMA
Quality Components
9934 East 21st South
Tulsa, OK 74129
918/664-8812

OREGON
Kierulff Electronics
14273 NW Science Park
Portland, OR 97229
503/641-9150
TWX 910/467-8753

PENNSYLVANIA
Arrow Electronics
650 Seco Rd.
Monroeville, PA 15146
412/856-7000

Pioneer Electronics
560 Alpha Drive
Pittsburgh, PA 15238
412/782-2300
TWX 710/795-3122

Pioneer Electronics
261 Gibraltar
Horsham, PA 19044
215/674-4000
TWX 510/665-6778

Schweber Electronics
101 Rock Road
Horsham, PA 19044
215/441-0600

SOUTH CAROLINA
Hammond Electronics
1035 Lowndes Hill Rd.
Greenville, SC 29602
803/233-4121
TWX 810/281-2233

TEXAS
Arrow Electronics
10125 Metropolitan Dr.
Austin, TX 78758
512/835-4180

Arrow Electronics
13715 Gamma Road
Dallas, TX 75240
214/386-7500
TWX 910/860-5377

Arrow Electronics
10700 Corporate Drive
Suite 100
Stafford, TX 77477
713/491-4100
TWX 910/880-4439

Quality Components
4257 Kellway Circle
Addison, TX 75001
214/387-4949
TWX 910/860-5459

Quality Components
2427 Rutland Drive
Austin, TX 78758
512/835-0220
TWX 910/874-1377

Quality Components
6126 Westline
Houston, TX 77036
713/772-7100

Schweber Electronics
10625 Richmond, Suite 100
Houston, TX 77042
713/784-3600
TWX 910/881-4836

Zeus/Dallas, Inc.
14001 Goldmark Dr.
Suite 250
Dallas, TX 75240
214/783-7010
TWX 910/867-9422

UTAH
Arrow Electronics
4980 Amelia Earhart Dr.
Salt Lake City, UT 84116
(801) 539-1135

Kierulff Electronics
2121 South 3600 West
Salt Lake City, UT 84119
801/973-6913

WASHINGTON
Arrow Electronics
14320 NE 21st
Bellevue, WA 98005
(206) 643-4800
TWX 910/444-2017

Kierulff Electronics
1053 Andover Park East
Tukwila, WA 98188
206/575-4420
TWX 910/444-2034

Zeus/West
23701 150th S.E.
Monroe, WA 98279

WISCONSIN
Arrow Electronics
434 Rawson Avenue
Oak Creek, WI 53154
414/764-6600
TWX 910/262-1193

Kierulff Electronics
2212 E. Moreland Blvd.
Waukesha, WI 53186
414/784-8160
TWX 910/265-3653

CANADA
Prelco Electronics
2767 Thames Gate Drive
Mississauga, Ontario
Toronto L4T 1G5
416/678-0401
TWX 610/492-8974

Prelco Electronics
480 Port Royal St. W.
Montreal 357 P.Q. H3L 2B9
514/389-8051
TWX 610/421-3616

Prelco Electronics
1770 Woodward Drive
Ottawa, Ontario K2C 0P8
613/226-3491
Telex 05-34301

R.A.E. Industrial
3455 Gardner Court
Burnaby, B.C. V5G 4J7
604/291-8866
TWX 610/929-3065

Zentronics
141 Catherine Street
Ottawa, Ontario
K2P 1C3
613/238-6411

Zentronics
8 Tilbury Court
Brampton, Ontario
L6T3T4 (Toronto)
416/451-9600
Telex 06-97678

Zentronics
505 Locke St.
St. Laurent, Quebec
H4T IX7
514/735-5361
Telex 058-27535

Zentronics
590 Berry Street
St. James, Manitoba
R2H OR4
204/775-8661

Zentronics
480 "A" Dutton Drive
Waterloo, Ontario
N2L 4C6
519/884-5700

R.A.E. Industrial
11680 170th St.
Edmonton, Alberta T5S 1J7
403/451-4001
Telex 03-72653

Zentronics
550 Cambie St.
Vancouver, B.C. V6B 2N7
604/688-2533
Telex 04507789

Zentronics
3651 21st Street, N.E.
Calgary, Alberta T2E 6T5
403/230-1422

Zentronics
9224 27th Avenue
Edmonton, Alberta T6N 1B2
403/463-3014

Zentronics
30 Sommonds Drive, Unit B
Dartmouth, N.S. B3B1R3
902/463-8411

*Franchised for USA and Canada excluding California for military products

INTERNATIONAL MARKETING OFFICES

EUROPEAN HEAD OFFICE
Mostek International
Av de Tervuren 270-272 Bte 21
B-1150 Brussels/Belgium
02/762.18.80
Telex: 62011

FRANCE
Mostek France s.a.r.l.
30 Rue du Morvan
SILIC 505
F-94623 Rungis Cedex
(1) 687 34.14
Telex: 204049

GERMANY
PLZ 1-5
Mostek GmbH
Friedlandstrasse 1
D-2085 Quickborn
(04106) 2077/78
Telex: 213685

PLZ 6-7
Mostek GmbH
Schurwaldstrasse 15
D-7303 Neuhausen/Filder
(07158) 66.45
Telex: 72.38.86

PLZ 8
Mostek GmbH
Zaunkonigstr. 18
D-8012 Ottobrunn
(089) 95.10.71
Telex: 5216516

ITALY
Mostek Italia SRL
Via F.D. Guerrazzi 27
I 20145 Milano
(02) 318.5337/349.2696
and 34.23.89
Telex: 333601

JAPAN
Mostek Japan KK
Sanyo Bldg. 3F
1-2-7 Kita-Aoyama
Minato-Ku, Tokyo 107
(03) 404-7261
Telex: J23686

SWEDEN
Mostek Scandinavia AB
Spjulvagen 7
S-17561 Järfälla
Sweden
08-36.2820
Telex: 12997

UNITED KINGDOM
Mostek U.K. Ltd.
Masons House,
1-3 Valley Drive
Kingsbury Road
London, N.W.9
01-204 9322
Telex: 25940

FAR EAST
Mostek Asia Ltd.
Kam Chung Commercial Bldg.
19 Hennessy Rd. 11/FL
P.O. Box 10786 Hong Kong
Phone: 5.296.886
Telex: 78072585 MKHK

II

INTERNATIONAL SALES REPRESENTATIVES/DISTRIBUTORS

ARGENTINA
Rayo Electronics S.R.L.
Belgrano 990, Pisos 6y2
1092 Buenos Aires
(38)-1779, 37-9476
Telex - 122153

AUSTRALIA
Amtron Tyree Pty.Ltd.
176 Botany Street
Waterloo, N.S.W. 2017
(61) 69-89.666
Telex - 25643

AUSTRIA
Transistor Vertriebsges, mbH
Auhofstrasse 41 A
A-1130 Vienna
(0222) 82 9451, 83 9404
Telex - 0133738

BRASIL
Cosele, Ltd.
Rua da Consolacao, 867
Conj. 31
01301 Sao Paulo
(55) 11-257.35.35/258.43.25
Telex - 1130869

BELGIUM
Sotronic
14 Rue Pere De Deken
B-1040 Brussels
02 736.10.07
Telex - 25141

DENMARK
Semicap APS
Gammel Kongevej 148
DK-1850 Copenhagen
01-22.15.10
Telex - 15987

FINLAND
Insele Oy
Kumpulantie 1
SF-00520 Helsinki 52
0 735 774
Telex: 122217

FRANCE
Copel
Rue Fourny, Z.I.
B.P. 22, F-78 530 BUC
(1) 956 1018
Telex: 69379

Facen
110 Av de Flandre
F59290 Wasquehal. Nord
(2) 98.92.15

Branch Offices in
Chalon/Saone, Lille,
Nancy, Rouen, Strasbourg

Mecodis
2 Rue Pasteur
F-94380 Bonneuil
(1) 339.20.20
Telex: 250303

P.E.P.
4 Rue Barthelemy
F-92120 Montrouge
(1)-735.33.20
Telex: 204 534

Scaib
80 Rue d'Arcueil
SILIC 137
F-94523 Rungis Cedex
(1)-687.23.12
Telex: 204674

Sorhodis
150-152, Rue A. France
F69100 Villeurbanne
(78) 850044
Telex: 380181

GERMANY
Dr Dohrenberg
Bayreuther Strasse 3
D-1000 Berlin 30
(030) 213.80.43
Telex: 0 184860

Neye Enatechnik GmbH
Schillerstrasse 14
D-2085 Quickborn
(04106) 612-1
Telex: 0 213.590

Branch offices in: Berlin, Hannover,
Dusseldorf, Darmstadt, Stuttgart,
Munchen

Raffel-Electronic GmbH
Lochnerstrasse 1
D-4030 Ratingen 1
(2102) 280.24
Telex: 8585180

Siegfried Ecker
Koenigsberger Strasse 2
D-6120 Michelstadt
(6061) 2233
Telex: 4191630

Matronic GmbH
Lichtenberger Weg 3
D-7400 Tuebingen
(7071) 24331
Telex: 7262879

Dema-Electronic GmbH
Bluetenstrasse 21
D-8000 Munchen 40
(089) 288018/19
Telex: 05-29345

HONG KONG
Cet Limited
1402 Tung Wah Mansion
199-203 Hennessy Road
Wanchai, Hong Kong
(5)-72.93.76
Telex - 85148

ISRAEL
Telsys Ltd.
12, Kehilat Venetsia St.
Tel Aviv. Israel
482126/7/8
Telex: 032392

ITALY
Comprel s.r.l.
V.le Romagna. 1
I-20092 Cinisello B. (MI)
(02) 61.20.641/2/3/4/5
Telex: 332484

Branch offices in
Bologna, Firenze,
Lavagna, Loreto,
Padova, Roma, Torino,
Vicenza, Bari

Emesa S.P.A.
Via L. da Viadana, 9
I-20122 Milano
(02) 869.0616
Telex: 335066

Branch offices in
Torino, Bologna, Roma

JAPAN
Systems Marketing, Inc.
4th Floor, Shindo Blgd.
3-12-5 Uchikanda,
Chiyoda-Ku,
Tokyo, 100
(81) 3-254.27.51
Telex - 25761

Teijin Advanced Products Corp.
1-1 Uchisaiwai-Cho
2-Chome Chiyoda-Ku
Tokyo, 100
(81) 3-506.46.73
Telex - 23548

KOREA
Vine Overseas Trading Corp.
Room 308 Korea Electric
Association Bldg.
11-4 Supyo-Dong Jung-Ku
Seoul
(82) 2-66-1663

THE NETHERLANDS
Nijkerk Elektronika BV
Drentestraat 7
NL - 1083 HK Amsterdam
(020) 428. 933
Telex: 11625

NEW ZEALAND
E.C.S. Div. of Airspares
P.O. Box 1048
Airport Palmerston North
(77)-047
Telex - 3766

NORWAY
Hefro Teknisk A/S
Postboks 6596
Rodelokka, Oslo 5
02-38.02.86
Telex: 16205

PORTUGAL
Digicontrole LDA
Av. de Roma 105
Sexto Esquerdo
1700 Lisboa
19.682.428
Telex: 15084

SINGAPORE
Dynamar International, LTD.
Suite 526, Cuppage Road
Singapore 0922

SOUTH AFRICA
Radiokom
P.O. Box 56310
Pinegowrie
2123,
Transvaal
789-1400
Telex - 8-0838 SA

SPAIN
Comelta S.A.
Emilio Munoz 41, ESC 1
Planta 1 Nave 2
Madrid-17
01-754 3001/3007
Telex: 42007

Branch Office
Diputacion, 79
Entlo 1
Barcelona-15
325 70 62
325 75 75
Telex: 519 34

SWEDEN
TRACO AB, Box 32
S-12221 Enskede
08-13 21 60
Telex: 10 689

Lagercrantz Elektronik AB
Box 48
S-19421 Upplands Vasby
0760 861 20
Telex: 11275

SWITZERLAND
Memotec AG
Gaswerkstrasse, 32
CH-4901 Langenthal
063-28.11.22
Telex: 68636

TAIWAN
Dynamar Taiwan Limited
P.O. Box 67-445
2nd Floor, No. 14, Lane 164
Sung-Chiang Road
Taipei
5418251
Telex - 11064

UNITED KINGDOM
Celdis Limited
37-39 Loverock Road
Reading
Berks RG 31 ED
0734-58.51.71
Telex: 848370

Lock Distribution Ltd.
Neville Street
Chadderton
Oldham
Lancashire
OL9 6LF
061-652.04.31
Telex: 669971

Pronto Electronic Systems Ltd.
466-478 Cranbrook Road,
Gants Hill Ilford
Essex 1G2 6LE
01-554 6222
Telex: 895 4213

VSI Electronics (UK) Ltd.
Roydondury Industrial Park
Horsecroft Rd.
Harlow
Essex CM19 5BY
(0279) 35477
Telex: 81387

Thame Components Ltd.
Thame Park Road
Thame, Oxon OX9 3XD
084 4213146
Telex: 837917

1982/1983 Z80 DESIGNERS GUIDE

MOSTEK®

Z80 MICROCOMPUTER DEVICES

Technical Manual

III

MK3880 CENTRAL PROCESSING UNIT

TABLE OF CONTENTS

1.0 INTRODUCTION

The term "microcomputer" has been used to describe virtually every type of small computing device designed within the last few years. This term has been applied to everything from simple "microprogrammed" controllers constructed out of TTL MSI to low end minicomputers with a portion of the CPU constructed out of TTL LSI "bit slices." However, the major impact of the LSI technology within the last few years has been with MOS LSI. With this technology, it is possible to fabricate complete and very powerful computer systems with only a few MOS LSI components.

The Mostek Z80 family of components is a significant advancement in the state-of-the-art of microcomputers. These components can be configured with any type of standard semiconductor memory to generate computer systems with an extremely wide range of capabilities. For example, as few as two LSI circuits and three standard TTL MSI packages can be combined to form a simple controller. With additional memory and I/O devices, a computer can be constructed with capabilities that only a minicomputer could previously deliver. This wide range of computational power allows standard modules to be constructed by a user that can satisfy the requirements of an extremely wide range of applications.

The major reason for MOS LSI domination of the microcomputer market is the low cost of these few LSI components. For example, MOS LSI microcomputers have already replaced TTL logic in such applications as terminal controllers, peripheral device controllers, traffic signal controllers, point of sale terminals, intelligent terminals and test systems. In fact the MOS LSI microcomputer is finding its way into almost every product that now uses electronics and it is even replacing many mechanical systems such as weight scales and automobile controls.

The MOS LSI microcomputer market is already well established and new products using microcomputer devices are being developed at an extraordinary rate. The Mostek Z80 component set has been designed to fit into this market through the following factors:

1. The Z80 is fully software compatible with the popular 8080A CPU offered from several sources. Existing designs can be easily converted to include the Z80 as a superior alternative.

2. The Z80 component set is superior in both software and hardware capabilities to any other 8-bit microcomputer system on the market. These capabilities provide the user with significantly lower hardware and software development costs while also allowing him to add additional features in his system.

3. A complete development and OEM system product line including full software support is available to enable the user to develop new products easily.

Microcomputer systems are extremely simple to construct using Z80 components. Any such system consists of three parts:

1. CPU (Central Processing Unit)

2. Memory

3. Interface circuits to peripheral devices

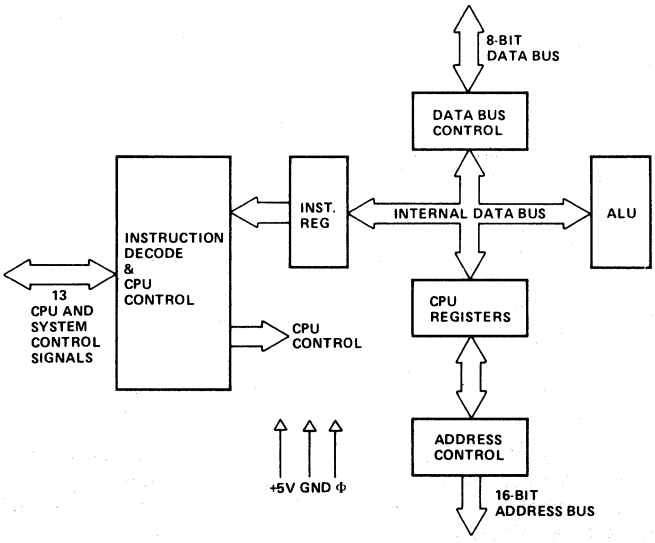
The CPU is the heart of the system. Its function is to obtain instructions from the memory and perform the desired operations. The memory is used to contain instructions and in most cases data that is to be processed. For example, a typical instruction sequence may be to read data from a specific peripheral device, store it in a location in memory, check the parity, and write it out to another peripheral device. Note that the Mostek component set includes the CPU and various general purpose I/O device controllers, as well as a wide range of memory devices. Thus, all required components can be connected together in a very simple manner with virtually no other external logic. The user's effort then becomes primarily one of software development. That is, the user can concentrate on describing his problem and translating it into a series of instructions that can be loaded into the microcomputer memory. Mostek is dedicated to making this step of software generation as simple as possible. A good example of this dedication is our assembly language in which a simple mnemonic is used to represent every instruction that the CPU can perform. This

language is self documenting in such a way that, from the mnemonic, the user can understand exactly what the instruction is doing without constantly checking back to a complex cross listing.

2.0 Z80-CPU ARCHITECTURE

A block diagram of the internal architecture of the Z80-CPU is shown in Figure 2.0-1. The diagram shows all of the major elements in the CPU and it should be referred to throughout the following description.

Z80-CPU BLOCK DIAGRAM
Figure 2.0-1



2.1 CPU REGISTERS

The Z80-CPU contains 208 bits of R/W memory that are accessible to the programmer. Figure 2.0-2 illustrates how this memory is configured into eighteen 8-bit registers and four 16-bit registers. All Z80 registers are implemented using static RAM. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or in pairs as 16-bit registers. There are also two sets of accumulator and flag registers.

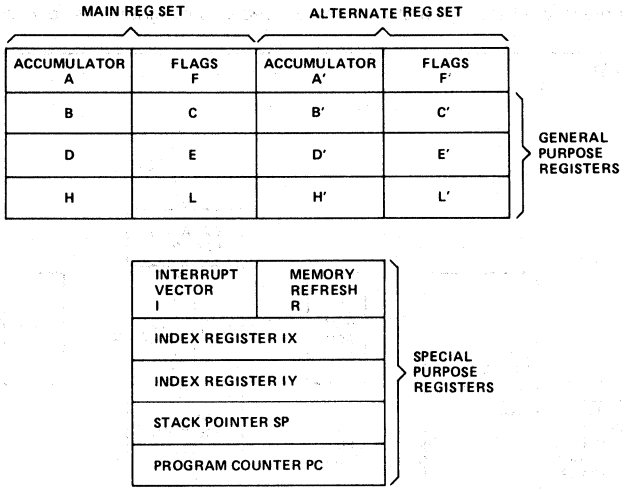
Special Purpose Registers

1. **Program Counter (PC).** The program counter holds the 16-bit address of the current instruction being fetched from memory. The PC is automatically incremented after its contents have been transferred to the address lines. When a program jump occurs, the new value is automatically placed in the PC, overriding the incrementer.

2. **Stack Pointer (SP).** The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in first-out (LIFO) file. Data can be pushed onto the stack from specific CPU registers or popped off the stack into specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the last data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.

3. **Two Index Registers (IX & IY).** The two independent index registers hold a 16-bit base address that is used in indexed addressing modes. In this mode, an index register is used as a base to point to a region in memory from which data is to be stored or retrieved. An additional byte is included in indexed instructions to specify a displacement from this base. This displacement is specified as a two's complement signed integer. This mode of addressing greatly simplifies many types of programs, especially where tables of data are used.

Z80-CPU REGISTER CONFIGURATION
Figure 2.0-2



4. **Interrupt Page Address Register (I).** The Z80-CPU can be operated in a mode where an indirect call to any memory location can be achieved in response to an interrupt. The I Register is used for this purpose to store the high order 8-bits of the indirect address while the interrupting device provides the lower 8-bits of the address. This feature allows interrupt routines to be dynamically located anywhere in memory with absolute minimal access time to the routine.

5. **Memory Refresh Register (R).** The Z80-CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. This 7-bit register is automatically incremented after each instruction fetch. The data in the register is automatically incremented after each instruction fetch. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is totally transparent to the programmer and does not slow down the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer.

Accumulator and Flag Registers

The CPU includes two independent 8-bit accumulators and associated 8-bit flag registers. The accumulator holds the results of 8-bit arithmetic or logical operations while the flag register indicates specific conditions for 8 or 16-bit operations, such as indicating whether or not the result of an operation is equal to zero. The programmer selects with a single exchange instruction the accumulator and flag pair that he wishes to work with so that he may easily work with either pair.

General Purpose Registers

There are two matched sets of general purpose registers, each set containing six 8-bit registers that may be used individually as 8-bit register or as 16-bit register pairs by the programmer. One set is called BC, DE, and HL while the complementary set is called BD', DE' and HL'. At any one time the programmer can select either set of registers to work with through a single exchange command for the entire set. In systems where fast interrupt response is required, one set of general purpose registers and an accumulator/flag register may be reserved for handling this very fast routine. Only a simple exchange command need be executed to go between the routines. This command greatly reduces interrupt service time by eliminating the requirement for saving and retrieving register contents in the external stack during interrupt or subroutine processing. These general purpose registers are used for a wide range of applications by the programmer. They also simplify

programming, especially in ROM based systems where little external read/write memory is available.

2.2 ARITHMETIC & LOGIC UNIT (ALU)

The 8-bit arithmetic and logical instructions of the CPU are executed in the ALU. Internally the ALU communicates with the registers and the external data bus on the internal data bus. The type of functions performed by the ALU includes:

Add	Left or right shifts or rotates (arithmetic and logical)
Subtract	Increment
Logical AND	Decrement
Logical OR	Set bit
Logical Exclusive OR	Reset bit
Compare	Test bit

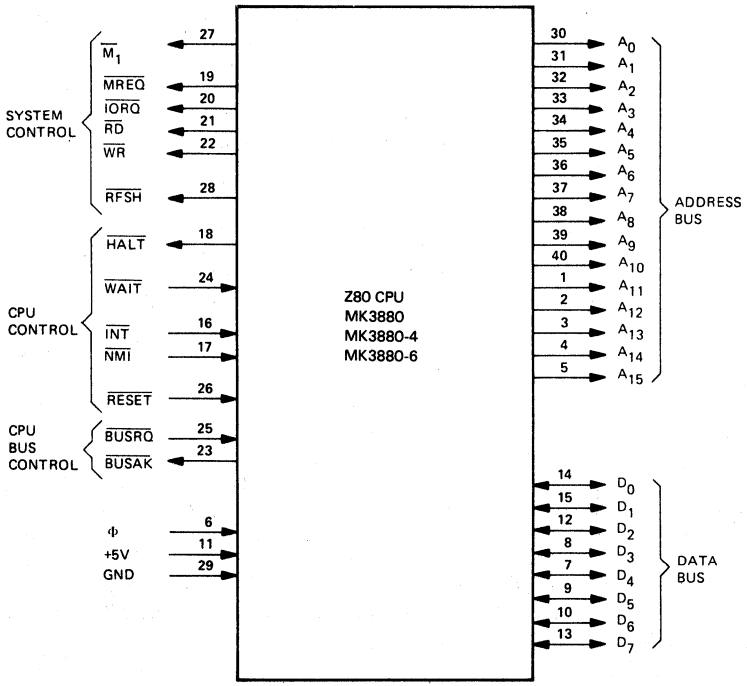
2.3 INSTRUCTION REGISTER AND CPU CONTROL

As each instrution is fetched from memory, it is placed in the instruction register and decoded. The control section performs this function, then generates and supplies all of the control signals necessary to read or write data from or to the registers, controls the ALU and provides all required external control signals.

3.0 Z80-CPU PIN DESCRIPTION

The Z80-CPU is packaged in an industry-standard 40 pin Dual In-Line Package. The I/O pins are shown in Figure 3.0-1 and the function of each is described below.

Z80 PIN CONFIGURATION
Figure 3.0-1



A_0 - A_{15}
(Address Bus)

Tri-state output, active high. A_0-A_{15} constitute a 16-bit address bus. The address bus provides the address for memory (up to 64K bytes) data exchanges and for I/O device data exchanges. I/O addressing uses the 8 lower address bits to allow the user to select up to 256 input or 256 output ports directly. A_0 is the least significant address bit. During refresh time, the lower 7 bits contain a valid refresh address.

D_0 - D_7
(Data Bus)

Tri-state input/output, active high. D_0-D_7 constitute an 8-bit bidirectional data bus. The data bus is used for data exchanges with memory and I/O devices.

$\overline{M_1}$
(Machine Cycle one)

Output, active low. $\overline{M_1}$ indicates that the current machine cycle is the OP code fetch cycle of an instruction execution. Note that during execution of 2-byte op-codes, $\overline{M_1}$ is generated as each op code byte is fetched. These two byte op-codes always begin with CBH, DDH, EDH, or FDH. $\overline{M_1}$ also occurs with $\overline{IORQ}$ to indicate an interrupt acknowledge cycle.

$\overline{MREQ}$
(Memory Request)

Tri-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory read or memory write operation.

$\overline{IORQ}$
(Input/Output Request)

Tri-state output, active low. The $\overline{IORQ}$ signal indicates that the lower half of the address bus holds a valid I/O address for an I/O read or write operation. An $\overline{IORQ}$ signal is also generated with an $\overline{M_1}$ signal when an interrupt is being

acknowledged to indicate that an interrupt response vector can be placed on the data bus. Interrupt Acknowledge operations occur during M_1 time while I/O operations never occur during M_1 time.

$\overline{RD}$
(Memory Read)

Tri-state output, active low. $\overline{RD}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

$\overline{WR}$
(Memory Write)

Tri-state output, active low. $\overline{WR}$ indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.

$\overline{RFSH}$
(Refresh)

Output, active low. $\overline{RFSH}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and current $\overline{MREQ}$ signal should be used to do a refresh read to all dynamic memories. A_7 is a logic zero and the upper 8 bits of the Address Bus contain the I Register.

$\overline{HALT}$
(Halt state)

Output, active low. $\overline{HALT}$ Indicates that the CPU has executed a HALT software instruction and is awaiting either a non maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.

$\overline{WAIT}$*
(Wait)

Input, active low. $\overline{WAIT}$ indicates to the Z80-CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active. This signal allows memory or I/O devices of any speed to be synchronized to the CPU.

$\overline{INT}$
(Interrupt Request)

Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled and if the $\overline{BUSRQ}$ signal is not active. When the CPU accepts the interrupt, an acknowledge signal ($\overline{IORQ}$ during M_1 time) is sent out at the beginning of the next instruction cycle. The CPU can respond to an interrupt in three different modes that are described in detail in section 8.

$\overline{NMI}$

Input, negative edge triggered. The non maskable interrupt request line has a higher priority than $\overline{INT}$ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. $\overline{NMI}$ automatically forces the Z80-CPU to restart to location 0066_H. The program counter is automatically saved in the external stack so that the user can return to the program that was interrupted. Note that continuous WAIT cycles can prevent the current instruction from ending, and that a $\overline{BUSRQ}$ will override a $\overline{NMI}$.

$\overline{RESET}$

Input, active low. $\overline{RESET}$ forces the program counter to zero and initializes the CPU. The CPU initialization will:

1) Disable the interrupt enable flip-flop
2) Set Register I = 00_H
3) Set Register R = 00_H
4) Set Interrupt Mode 0

During reset time, the address bus and data bus go to a high impedance state and all control output signals go to the inactive state. No refresh occurs.

$\overline{BUSRQ}$
(Bus Request)

Input, active low. The bus request signal is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these buses. When $\overline{BUSRQ}$ is activated, the CPU will set these buses to a high impedance state as soon as the current CPU machine cycle is terminated.

$\overline{\text{BUSAK}}$* (Bus Acknowledge)	Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.
Φ	Single phase system clock.

*While the Z80-CPU is in either a $\overline{\text{WAIT}}$ state or a Bus Acknowledge condition, Dynamic Memory Refresh will not occur.

4.0 CPU TIMING

The Z80-CPU executes instructions by stepping through a very precise set of a few basic operations. These include:

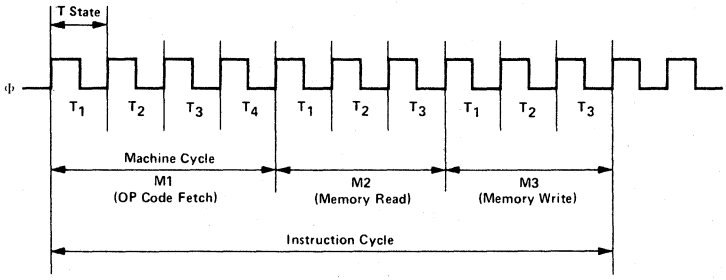
Memory read or write

I/O device read or write

Interrupt acknowledge

All instructions are merely a series of these basic operations. Each of these basic operations can take from three to six clock periods to complete or they can be lengthened to synchronize the CPU to the speed of external devices. The basic clock periods are referred to as T states and the basic operations are referred to as M (for machine) cycles. Figure 4.0-0 illustrates how a typical instruction will be merely a series of specific M and T cycles. Notice that this instruction consists of three machine cycles (M1, M2 and M3). The first machine cycle of any instruction is a fetch cycle which is four, five, or six T states long (unless lengthened by the wait signal which will be fully described in the next section). The fetch cycle (M1) is used to fetch the OP code of the next instruction to be executed. Subsequent machine cycles move data between the CPU and memory or I/O devices, and they may have anywhere from three to five T cycles (again they may be lengthened by wait states to synchronize the external devices to the CPU). The following paragraphs describe the timing which occurs within any of the basic machine cycles. In section 7, the exact timing of each instruction is specified.

BASIC CPU TIMING EXAMPLE
Figure 4.0-0



All CPU timing can be broken down into a few very simple timing diagrams as shown in Figure 4.0-1 through 4.0-7. These diagrams show the following basic operations with and without wait states (wait states are added to synchronize the CPU to slow memory or I/O devices).

4.0-1. Instruction OP code fetch (M1 cycle)
4.0-2. Memory data read or write cycles
4.0-3. I/O read or write cycles
4.0-4. Bus Request/Acknowledge Cycle
4.0-5. Interrupt Request/Acknowledge Cycle
4.0-6. Non maskable Interrupt Request/Acknowledge Cycle
4.0-7. Exit from a HALT instruction

INSTRUCTION FETCH

Figure 4.0-1 shows the timing during an M1 cycle (OP code fetch). Notice that the PC is placed on the address bus at the beginning of the M1 cycle. One half clock time later the $\overline{MREQ}$ signal goes active. At this time the address to the memory has had time to stabilize so that the falling edge of $\overline{MREQ}$ can be used directly as a chip enable clock to dynamic memories. The $\overline{RD}$ line also goes active to indicate that the memory read data should be enabled onto the CPU data bus. The CPU samples the data from the memory on the data bus with the rising edge of the clock of state T3 and this same edge is used by the CPU to turn off the $\overline{RD}$ and $\overline{MREQ}$ signals. Thus the data has already been sampled by the CPU before the $\overline{RD}$ signal becomes inactive. Clock state T3 and T4 of a fetch cycle are used to refresh dynamic memories. (The CPU uses this time to decode and execute the fetched instruction so that no other operation could be performed at this time). During T3 and T4, the lower 7 bits of the address bus contain a memory refresh address and the $\overline{RFSH}$ signal becomes active to indicate that a refresh read of all dynamic memories should be accomplished. Notice that an $\overline{RD}$ signal is not generated during refresh time to prevent data from different memory segments from being gated onto the data bus. The $\overline{MREQ}$ signal during refresh time should be used to perform a refresh read of all memory elements. The refresh signal cannot be used by itself since the refresh address is only guaranteed to be stable during $\overline{MREQ}$ time.

INSTRUCTION OP CODE FETCH
Figure 4.0-1

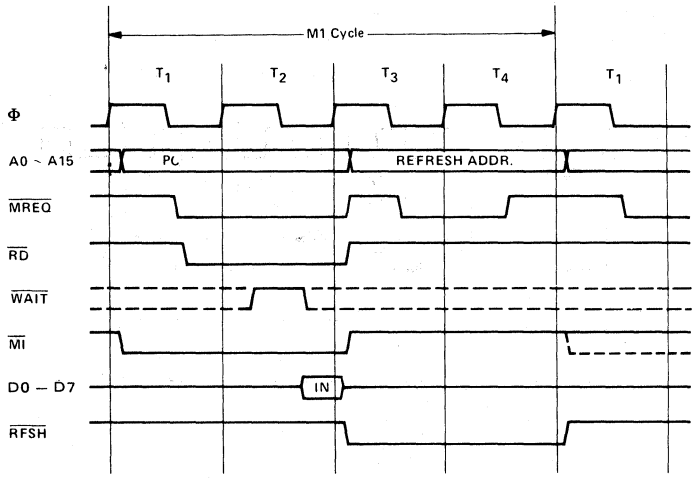
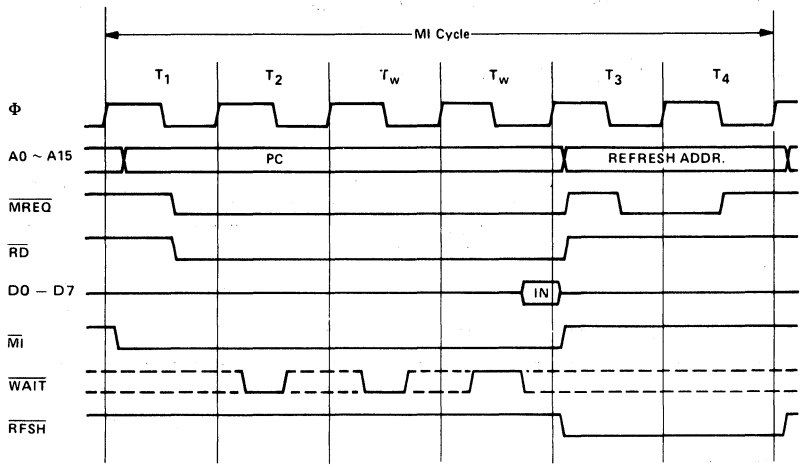


Figure 4.0-1A illustrates how the fetch cycle is delayed if the memory activates the $\overline{WAIT}$ line. During T2 and every subsequent Tw, the CPU samples the $\overline{WAIT}$ line with the falling edge of Φ. If the $\overline{WAIT}$ line is active at this time, another wait state will be entered during the following cycle. Using this technique, the read cycle can be lengthened to match the access time of any type of memory device.

INSTRUCTION OP CODE FETCH WITH WAIT STATES

Figure 4.0-1A



MEMORY READ OR WRITE

Figure 4.0-2 illustrates the timing of memory read or write cycles other than an OP code fetch (M1 cycle). These cycles are generally three clock periods long unless wait states are requested by the memory via the $\overline{WAIT}$ signal. The $\overline{MREQ}$ signal and the $\overline{RD}$ signal are used the same as in the fetch cycle. In the case of a memory write cycle, the $\overline{MREQ}$ also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The $\overline{WR}$ line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory. Furthermore, the $\overline{WR}$ signal goes inactive one half T state before the address and data bus contents are changed so that the overlap requirements for virtually any type of semiconductor memory type will be met.

MEMORY READ OR WRITE CYCLES

Figure 4.0-2

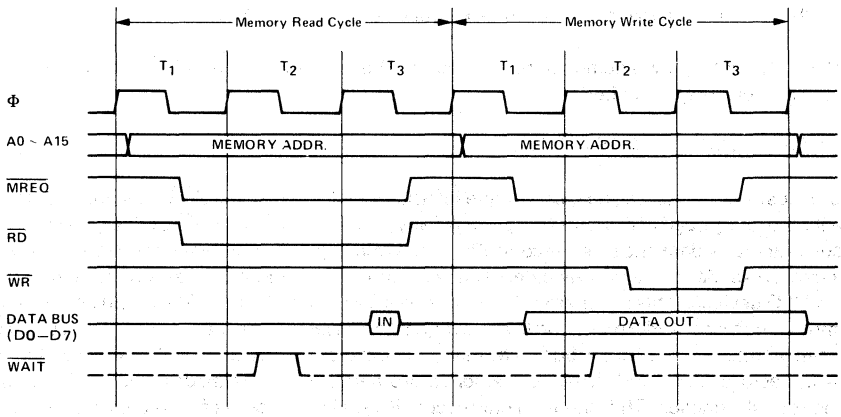
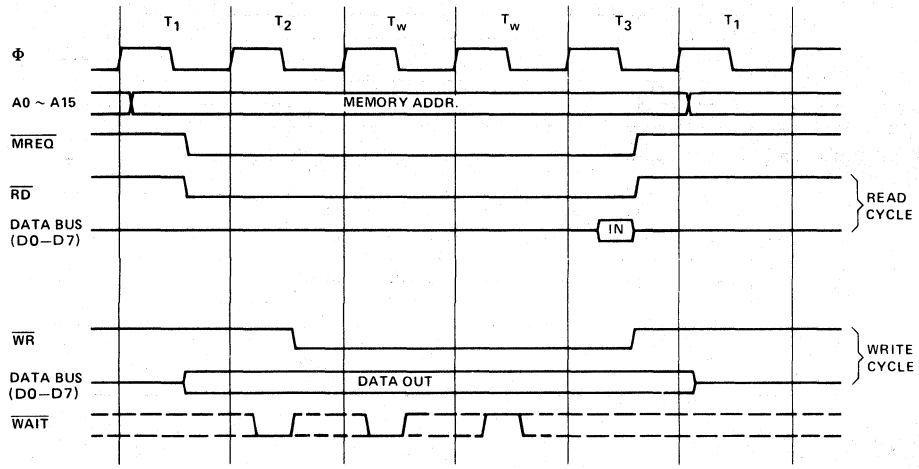


Figure 4.0-2A illustrates how a $\overline{WAIT}$ request signal will lengthen any memory read or write operation. This operation is identical to that previously described for a fetch cycle. Notice in this figure that a separate read and a separate write cycle are shown in the same figure although read and write cycles can never occur simultaneously.

MEMORY READ OR WRITE CYCLES WITH WAIT STATES
Figure 4.0-2A



INPUT OR OUTPUT CYCLES

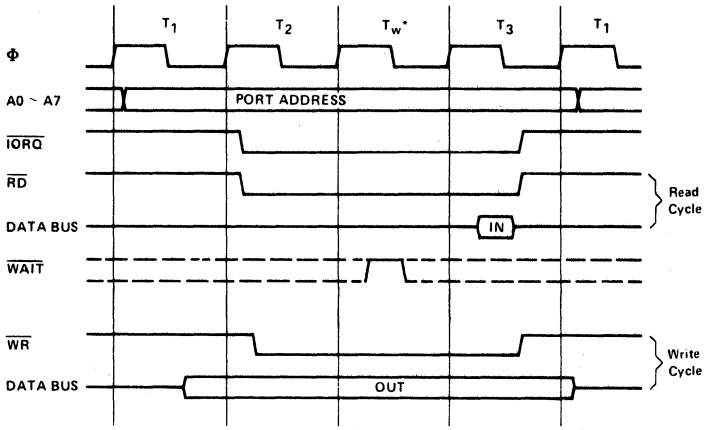
Figure 4.0-3 illustrates an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted. The reason for this is that during I/O operations, the time from when the $\overline{IORQ}$ signal goes active until the CPU must sample the $\overline{WAIT}$ line is very short and, without this extra state, sufficient time does not exist for an I/O port to decode its address and activate the $\overline{WAIT}$ line if a wait is required. Also, without this wait state, it is difficult to design MOS I/O devices that can operate at full CPU speed. During this wait state time, the $\overline{WAIT}$ request signal is sampled. During a read I/O operation, the $\overline{RD}$ line is used to enable the addressed port onto the data bus just as in the case of memory read. For I/O write operation, the $\overline{WR}$ line is used as a clock to the I/O port, again with sufficient overlap timing automatically provided so that the rising edge may be used as a data clock.

Figure 4.0-3A illustrates how additional wait states may be added with the $\overline{WAIT}$ line. The operation is identical to that previously described.

BUS REQUEST/ACKNOWLEDGE CYCLE

Figure 4.0-4 illustrates the timing for a Bus Request/Acknowledge cycle. The $\overline{BUSRQ}$ signal is sampled by the CPU with the rising edge of the last clock period of any machine cycle. If the $\overline{BUSRQ}$ signal is active, the CPU will set its address, data and tri-state control signals to the high impedance state with the rising edge of the next clock pulse. At that time any external device can control the buses to transfer data between memory and I/O devices. (This is generally known as Direct Memory Access [DMA] using cycle stealing). The maximum time for the CPU to respond to a bus request is the length of a machine cycle and the external controller can maintain control of the bus for as many clock cycles as is desired. Note, however, that if very long DMA cycles are used, and dynamic memories are being used, the external controller must also perform the refresh function. This situation only occurs if very large blocks of data are transferred under DMA control. Also note that during a bus request cycle, the CPU cannot be interrupted by either an $\overline{NMI}$ or an $\overline{INT}$ signal.

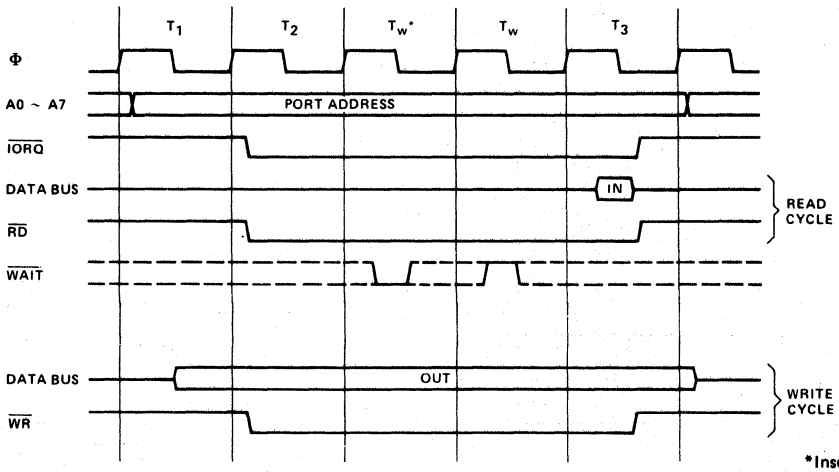
INPUT OR OUTPUT CYCLES
Figure 4.0-3



*Inserted by Z80 CPU

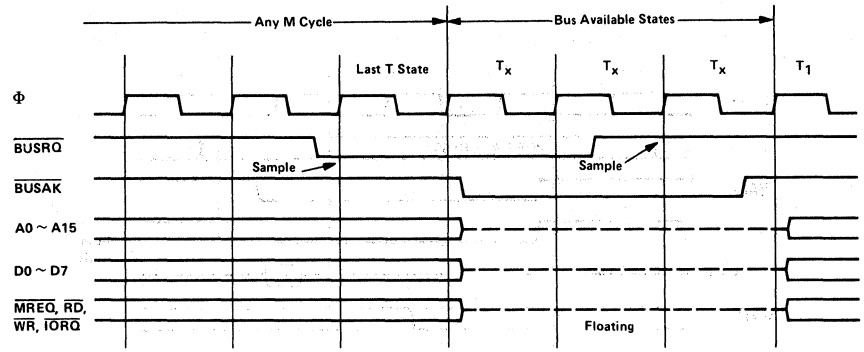
III

INPUT OR OUTPUT CYCLES WITH WAIT STATES
Figure 4.0-3A



*Inserted by Z80 CPU

BUS REQUEST/ACKNOWLEDGE CYCLE
Figure 4.0-4



INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

Figure 4.0-5 illustrates the timing associated with an interrupt cycle. The interrupt signal (INT) is sampled by the CPU with the rising edge of the last clock at the end of any instruction. The signal will not be accepted if the internal CPU software controlled interrupt enable flip-flop is not set or if the BUSRQ signal is active. When the signal is accepted, a special M1 cycle is generated. During this special M1 cycle, the IORQ signal becomes active (instead of the normal MREQ) to indicate that the interrupting device can place an 8-bit vector on the data bus. Notice that two wait states are automatically added to this cycle. These states are added so that a ripple priority interrupt scheme can be easily implemented. The two wait states allow sufficient time for the ripple signals to stabilize and identify which I/O device must insert the response vector. Refer to section 8.0 for details on how the interrupt response vector is utilized by the CPU.

INTERRUPT REQUEST/ACKNOWLEDGE CYCLE
Figure 4.0-5

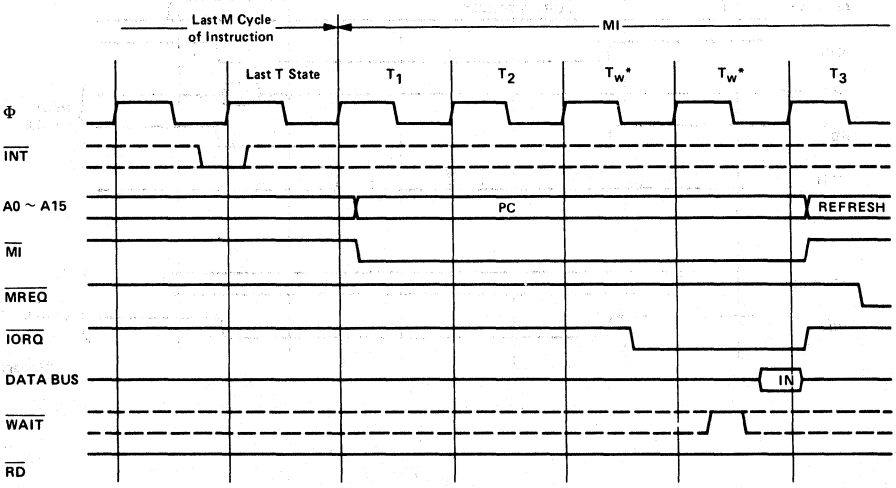
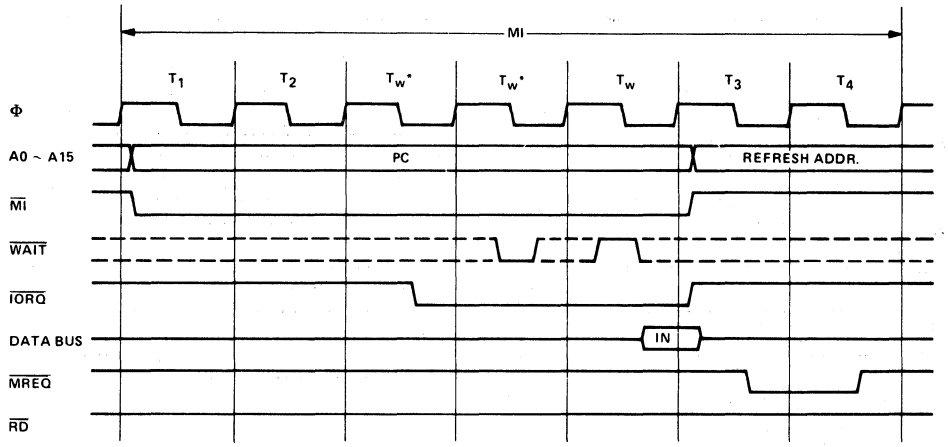


Figure 4.0-5A illustrates how additional wait states can be added to the interrupt response cycle. Again the operation is identical to that previously described.

INTERRUPT REQUEST/ACKNOWLEDGE WITH WAIT STATES
Figure 4.0-5A



Mode 0 shown

III

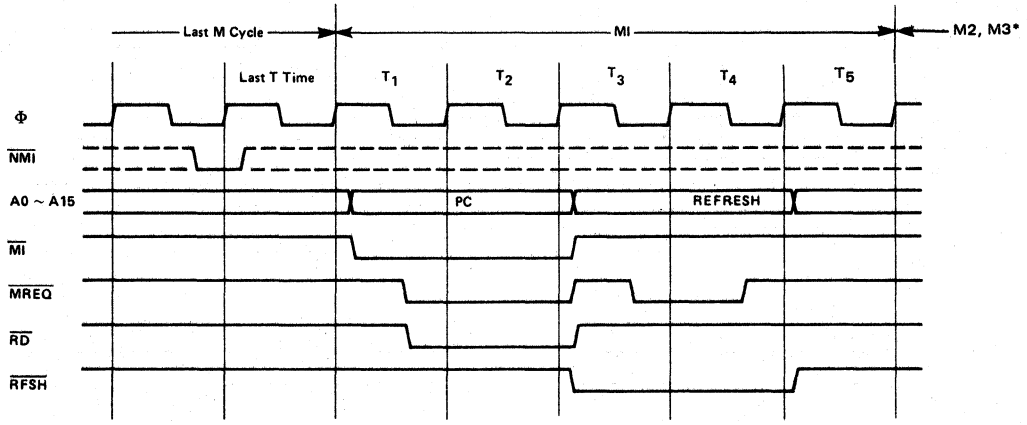
NON MASKABLE INTERRUPT RESPONSE

Figure 4.0-6 illustrates the request/acknowledge cycle for the non-maskable interrupt. A pulse on the NMI input sets an internal NMI latch which is tested by the CPU at the end of every instruction. This NMI latch is sampled at the same time as the interrupt line, but this line has priority over the normal interrupt, and it cannot be disabled under software control. Its usual function is to provide immediate response to important signals such as an impending power failure. The CPU response to a non maskable interrupt is similar to a normal memory read operation, the only difference being that the content of the data bus is ignored while the processor automatically stores the PC in the external stack and jumps to location 0066_H. The service routine for the non maskable interrupt must begin at this location if this interrupt is used.

HALT EXIT

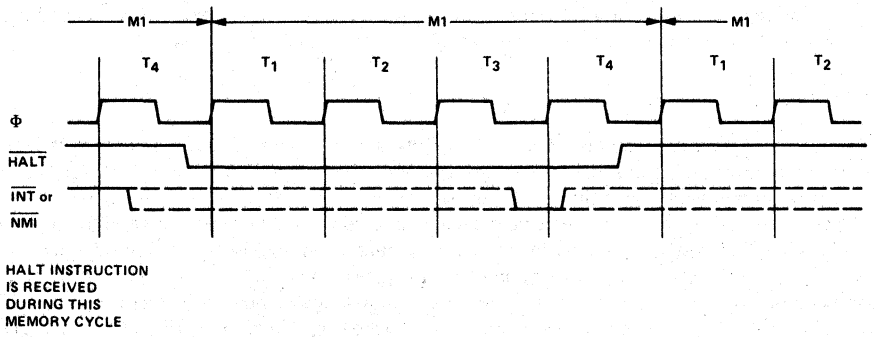
Whenever a software halt instruction is executed, the CPU begins executing NOP's until an interrupt is received (either a non-maskable or a maskable interrupt while the interrupt flip flop is enabled). The two interrupt lines are sampled with the rising clock edge during each T4 state, as shown in Figure 4.0-7. If a non-maskable interrupt has been received or a maskable interrupt has been received and the interrupt enable flip-flop is set, then the halt state will be exited on the next rising clock edge. The following cycle will then be an interrupt acknowledge cycle corresponding to the type of interrupt that was received. If both are received at this time, then the non maskable one will be acknowledged since it was highest priority. The purpose of executing NOP instructions while in the halt state is to keep the memory refresh signals active. Each cycle in the halt state is a normal M1 (fetch) cycle except that the data received from the memory is ignored and an NOP instruction is forced internally to the CPU. The halt acknowledge signal is active during this time to indicate that the processor is in the halt state.

NON MASKABLE INTERRUPT REQUEST OPERATION
Figure 4.0-6



*M2 and M3 are stack write operations

HALT EXIT
Figure 4.0-7



5.0 Z80-CPU INSTRUCTION SET

The Z80-CPU can execute 158 different instruction types including all 78 of the 8080A CPU. The instructions can be broken down into the following major groups:

- Load and Exchange
- Block Transfer and Search
- Arithmetic and Logical
- Rotate and Shift
- Bit Manipulation (set, reset, test)
- Jump, Call and Return
- Input/Output
- Basic CPU Control

5.1 INTRODUCTION TO INSTRUCTION TYPES

III

The load instructions move data internally between CPU registers or between CPU registers and external memory. All of these instructions must specify a source location, from which the data is to be moved, and a destination location. The source location is not altered by a load instruction. Examples of load group instructions include moves between any of the general purpose registers, such as a move of the data to Register B from Register C. This group also includes load immediate to any CPU register or to any external memory location. Other types of load instructions allow transfer between CPU registers and memory locations. The exchange instructions can trade the contents of two registers.

A unique set of block transfer instructions is provided in the Z80. With a single instruction a block of memory of any size can be moved to any other location in memory. This set of block moves is extremely valuable when large strings of data must be processed. The Z80 block search instructions are also valuable for this type of processing. With a single instruction, a block of external memory of any desired length can be searched for any 8-bit character. Once the character is found the instruction automatically terminates. Both the block transfer and the block search instructions can be interrupted during their execution so as not to occupy the CPU for long periods of time.

The arithmetic and logical instructions operate on data stored in the accumulator and other general purpose CPU registers or external memory locations. The results of the operations are placed in the accumulator and the appropriate flags are set according to the result of the operation. An example of an arithmetic operation is adding the accumulator to the contents of an external memory location. The results of the addition are placed in the accumulator. This group also includes 16-bit addition and subtraction between 16-bit CPU registers.

The bit manipulation instructions allow any bit in the accumulator, any general purpose register or any external memory location to be set, reset or tested with a single instruction. For example, the most significant bit of register H can be reset. This group is especially useful in control applications and for controlling software flags in general purpose programming.

The jump, call and return instructions are used to transfer an address between various locations in the user's program. This group uses several different techniques for obtaining the new program counter address from specific external memory locations. A unique type of jump is the restart instruction. This instruction actually contains the new address as a part of the 8-bit OP code. This type of jump is possible since only 8 separate addresses located in page zero of the external memory may be specified. Program jumps may also be achieved by loading register HL, IX or IY directly into the PC, thus allowing the jump address to be a complex function of the routine being executed.

The input/output group of instructions in the Z80 allows for a wide range of transfers between external memory locations or the general purpose CPU registers, and the external I/O devices. In each case, the port number is provided on the lower 8 bits of the address bus during any I/O transaction. One instruction allows this port number to be specified by the second byte of the instruction, while other Z80 instructions allow it to be specified as the content of the C register. One major advantage of using the C register as a pointer to the I/O device is that it allows different I/O

ports to share common software driver routines. This capability is not possible when the address is part of the OP code if the routines are stored in ROM. Another feature of these input instructions is that they set the flag register automatically so that additional operations are not required to determine the state of the input data (for example its parity). The Z80-CPU includes single instructions that can move blocks or data (up to 256 bytes) automatically to or from any I/O port directly to any memory location. In conjunction with the dual set of general purpose registers, these instructions provide for fast I/O block transfer rates. The value of this I/O instruction set is demonstrated by the fact that the Z80-CPU can provide all required floppy disk formatting (i.e., the CPU provides the preamble, address, and data, and enables the CRC codes) on double density floppy disk drives on an interrupt driven basis.

Finally, the basic CPU control instructions allow various options and modes. This group includes instructions such as setting or resetting the interrupt enable flip flop or setting the mode of interrupt response.

5.2 ADDRESSING MODES

Most of the Z80 instructions operate on data stored in the internal CPU registers, the external memory, or in the I/O ports. Addressing refers to how the address of this data is generated in each instruction. This section gives a brief summary of the types of addressing used in the Z80 while subsequent sections detail the type of addressing available for each instruction group.

Immediate. In this mode of addressing, the byte following the OP code in memory contains the actual operand.

OP Code	one or 2 bytes
Operand	

d_7 d_0

An example of this type of instruction would be to load the HL register pair (16-bit register) with 16 bits (2 bytes) of data.

Immediate Extended. This mode is merely an extension of immediate addressing, in that the two bytes following the op codes are the operand.

OP Code	one or 2 bytes
Operand	low order
Operand	high order

An example of this type of instruction would be to load the HL register pair (16-bit register) with 16 bits (2 bytes) of data.

Modified Page Zero Addressing. The Z80 has a special single byte call instruction to any of 8 locations in page zero of memory. This instruction (which is referred to as a restart) sets the PC to an effective address in page zero. The value of this instruction is that it allows a single byte to specify a complete 16-bit address where commonly called subroutines are located, thus saving memory space.

OP Code	one byte

b_7 d_0 Effective address is $(00b_5b_4b_3000)$

Relative Addressing. Relative addressing uses one byte of data following the OP code to specify a displacement from the existing program to which a program jump can occur. This displacement is a signed two's complement number that is added to the address of the OP code of the following instruction.

OP Code	Jump relative (one byte OP code)
Operand	8-bit two's complement displacement added to Address (A+2)

The value of relative addressing is that it allows jumps to nearby locations while only requiring two bytes of memory space. For most programs, relative jumps are by far the most prevalent type of jump owing to the proximity of related program segments. Thus, these instructions can significantly reduce memory space requirements. The signed displacement can range between +127 and −128 from A + 2. This allows for a total displacement of +129 to −126 from the jump relative OP code address. Another major advantage is that it allows for relocatable code.

III

Extended Addressing. Extended Addressing provides for two bytes (16 bits) of address to be included in the instruction. This data can be an address to which a program can jump or it can be an address where an operand is located.

OP Code	one or two bytes
Low Order Address or Low order operand	
High Order Address or High order operand	

Extended addressing is required for a program to jump from any location in memory to any other location, or load and store data in any memory location.

When extended addressing is used to specify the source or destination address of an operand, the notation (nn) will be used to indicate the content of memory at nn, where nn is the 16-bit address specified in the instruction. This means that the two bytes of address nn are used as a pointer to a memory location. The use of the parentheses always means that the value enclosed within them is used as a pointer to a memory location. For example, (1200) refers to the contents of memory at location 1200.

Indexed Addressing. In this type of addressing, the byte of data following the OP code contains a displacement which is added to one of the two index registers (the OP code specifies which index register is used) to form a pointer to memory. The contents of the index register are not altered by this operation.

OP Code	two byte OP code
OP Code	
Displacement	Operand added to index register to form a pointer to memory.

An example of an index instruction would be to load the contents of the memory location (Index Register + Displacement) into the accumulator. The displacement is a signed two's complement number. Indexed addressing greatly simplifies programs using tables of data since the index register can point to the start of any table. Two index registers are provided since, very often, operations require two or more tables. Indexed addressing also allows for relocatable code.

The two index registers in the Z80 are referred to as IX and IY. To indicate indexed addressing, the notation:

(IX+d) or (IY+d)

is used. Here d is the displacement specified after the OP code. The parentheses indicate that this value is used as a pointer to external memory.

Register Addressing. Many of the Z80 OP codes contain bits of information that specify which CPU register is to be used for an operation. An example of register addressing would be to load the data in register B into register C.

Implied Addressing. Implied addressing refers to operations where the OP code automatically implies one or more CPU registers as containing the operands. An example is the set of arithmetic operations where the accumulator is always implied to be the destination of the results.

Register Indirect Addressing. This type of addressing specifies a 16-bit CPU register pair (such as HL) to be used as a pointer to any location in memory. This type of instruction is very powerful and it is used in a wide range of applications.

OP Code	} one or two bytes

An example of this type of instruction would be to load the accumulator with the data in the memory location pointed to by the HL register contents. Indexed addressing is actually a form of register indirect addressing except that a displacement is added with indexed addressing. Register indirect addressing allows for very powerful but simple to implement memory accesses. The block move and search commands in the Z80 are extensions of this type of addressing where automatic register incrementing, decrementing and comparing have been added. The notation for indicating register indirect addressing is to put parentheses around the name of the register that is to be used as the pointer. For example, the symbol

(HL)

specifies that the contents of the HL register are to be used as a pointer to a memory location. Often register indirect addressing is used to specify 16-bit operands. In this case, the register contents point to the lower order portion of the operand while the register contents are automatically incremented to obtain the upper portion of the operand.

Bit Addressing. The Z80 contains a large number of bit set, reset and test instructions. These instructions allow any memory location or CPU register to be specified for a bit operation through one of three previous addressing modes (register, register indirect and indexed), while three bits in the OP code specify which of the eight bits is to be manipulated.

ADDRESSING MODE COMBINATIONS

Many instructions include more than one operand (such as arithmetic instructions or loads). In these cases, two types of addressing may be employed. For example, load can use immediate addressing to specify the source, and register indirect or indexed addressing to specify the source, and register indirect or indexed addressing to specify the destination.

5.3 INSTRUCTION OP CODES

This section describes each of the Z80 instructions and provides tables listing the OP codes for every instruction. In each of these tables, the shaded OP codes are identical to those offered in the 8080A CPU. Also shown is the assembly language mnemonic that is used for each instruction. All instruction OP codes are listed in hexadecimal notation. Single byte OP codes require two hex characters while double byte OP codes require four hex characters. The conversion from hex to binary is repeated here for convenience.

Hex		Binary	Decimal	Hex		Binary	Decimal
0	=	0000 =	0	8	=	1000 =	8
1	=	0001 =	1	9	=	1001 =	9
2	=	0010 =	2	A	=	1010 =	10
3	=	0011 =	3	B	=	1011 =	11
4	=	0100 =	4	C	=	1100 =	12
5	=	0101 =	5	D	=	1101 =	13
6	=	0110 =	6	E	=	1110 =	14
7	=	0111 =	7	F	=	1111 =	15

Z80 instruction mnemonics consist of an OP code and zero, one or two operands. Instructions in which the operand is implied have no operand. Instructions which have only one logical operand or those in which one operand is invariant (such as the Logical OR instruction) are represented by a one operand mnemonic. Instructions which may have two varying operands are represented by two operand mnemonics.

LOAD AND EXCHANGE

Table 5.3-1 defines the OP code for all of the 8-bit load instructions implemented in the Z80-CPU. Also shown in this table is the type of addressing used for each instruction. The source of the data is found on the top horizontal row while the destination is specified by the left hand column. For example, load register C from register B uses the OP code 48H. In all of the tables the OP code is specified in hexadecimal notation and the 48H (=0100 1000 binary) code is fetched by the CPU from the external memory during M1 time, is decoded and then the register transfer is automatically performed by the CPU.

The assembly language mnemonic for this entire group is LD, followed by the destination, followed by the source (LD DEST., SOURCE). Note that several combinations of addressing modes are possible. For example, the source may use register addressing and the destination may be register indirect, as in the case of loading the memory location pointed to by register HL with the contents of register D. The OP code for this operation would be 72. The mnemonic for this load instruction would be as follows: LD (HL), D

The parentheses around the HL mean that the contents of HL are used as a pointer to a memory location. In all Z80 load instruction mnemonics, the destination is always listed first, with the source following. The Z80 assembly language has been defined for ease of programming. Every instruction is self documenting and programs written in Z80 language are easy to maintain.

Note in Table 5.3-1 that some load OP codes that are available in the Z80 use two bytes. This is an efficient method of memory utilization, since 8, 16, 24 or 32 bit instructions are implemented in the Z80. Thus often utilized instructions such as arithmetic or logical operations are only 8-bits which result in better memory utilization than is achieved with fixed instruction sizes such as 16-bits.

All load instructions using indexed addressing for either the source or destination location actually use three bytes of memory with the third byte being the displacement d. For example, a load register E, with the operand pointed to by IX with an offset of +8, would be written: LD E, (IX + 8).

The instruction sequence for this in memory would be:

Address A	DD	OP Code
A+1	5F	
A+2	08	Displacement operand

The two extended addressing instructions are also three byte instructions. For example the instruction to load the accumulator with the operand in memory location 6F32H would be written as:

LD A, (6F 32H)

and its instruction sequence would be:

Address A	3A	OP Code
A+1	32	low order address
A+2	6F	high order address

Notice that the low order portion of the address is always the first operand.

The load immediate instructions for the general purpose 8-bit registers are two-byte instructions. The instruction load register H with the value 36H would be written as:

LD H, 36H

and its sequence would be:

Address A	26	OP Code
A+1	36	Operand

Loading a memory location using indexed addressing for the destination and immediate addressing for the source requires four bytes. For example,

LD (IX - 15), 21H

would appear as:

Address A	DD	OP Code
A+1	36	
A+2	F1	displacement (–15 in signed two's complement)
A+3	21	operand to load

Notice that with any indexed addressing the displacement always follows directly after the OP code.

Table 5.3-2 specifies the 16-bit load operations. This table is very similar to Table 5.3-1. Notice that the extended addressing capability covers all register pairs. Also notice that register indirect operations specifying the stack pointer are the PUSH and POP Instructions. The mnemonics for these instructions are "PUSH" and "POP". These instructions differ from other 16-bit loads in that the stack pointer is automatically decremented and incremented as each byte is pushed onto or popped from the stack respectively. For example, the instruction

PUSH AF

is a single byte instruction with the OP code of F5H. When this instruction is executed the following sequence is generated:

Decrement SP

LD (SP), A

Decrement SP

LD (SP), F

Thus the external stack now appears as follows:

(SP)	F	Top of stack
(SP+1)	A	
•	•	
•	•	

8 BIT LOAD GROUP

Table 5.3-1

			SOURCE															
			IMPLIED		REGISTER							REG INDIRECT			INDEXED		EXT. ADDR.	IMME.
			I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX + d)	(IY + d)	(nn)	n
DESTINATION	REGISTER	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
		B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
		C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
		D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
		E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
		H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
		L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
	REG INDIRECT	(HL)			77	70	71	72	73	74	75							36 n
		(BC)			02													
		(DE)			12													
	INDEXED	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
		(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
	EXT. ADDR	(nn)			32 n n													
	IMPLIED	I			ED 47													
		R			ED 4F													

The POP instruction is the exact reverse of a PUSH. Notice that all PUSH and POP instructions utilize a 16-bit operand and the high order byte is always pushed first and popped last.
That is:

PUSH BC is PUSH B then C
PUSH DE is PUSH D then E
PUSH HL is PUSH H then L
POP HL is POP L then H

The instruction using extended immediate addressing for the source obviously requires 2 bytes of data following the OP code. For example:

LD DE, 0659H

will be:

Address A	11	OP Code
A+1	59	Low order operand to register E
A+2	06	High order operand to register D

In all extended immediate or extended addressing modes, the low order byte always appears first after the OP code.

Table 5.3-3 lists the 16-bit exchange instructions implemented in the Z80. OP code 08H allows the programmer to switch between the two pairs of accumulator flag registers while D9H allows the programmer to switch between the duplicate set or six general purpose registers. These OP codes are only one byte in length to minimize the time necessary to perform the exchange absolutely, so that the duplicate banks can be used to effect very fast interrupt response times.

BLOCK TRANSFER AND SEARCH

Table 5.3-4 lists the extremely powerful block transfer instructions. All of these instructions operate with three registers:

HL points to the source location
DE points to the destination location.
BC is a byte counter.

After the programmer has initialized these three registers, any of these four instructions may be used. The LDI (Load and Increment) instruction moves one byte from the location pointed to by HL to the location pointed to by DE. Register pairs HL and DE are then automatically incremented and are ready to point to the following locations. The byte counter (register pair BC) is also decremented at this time. This instruction is valuable when blocks of data must be moved, but other types of processing are required between each move. The LDIR (load, increment and repeat) instruction is an extension of the LDI instruction. The same load and increment operation is repeated until the byte counter reaches the count of zero. Thus, this single instruction can move any block of data from one location to any other.

Note that since 16-bit registers are used, the size of the block can be up to 64K bytes (1K = 1024) long, and it can be moved from any location in memory to any other location. Furthermore the blocks can be overlapping since there are absolutely no constraints on the data that is used in the three register pair.

The LDD and LDDR instructions are very similar to the LDI and LDIR. The only difference is that register pairs HL and DE are decremented after every move so that a block transfer starts from the highest address of the designated block rather than the lowest.

BIT LOAD GROUP 'LD' 'PUSH' and 'POP'

Table 5.3.-2

DESTINATION		SOURCE: REGISTER AF	BC	DE	HL	SP	IX	IY	IMM. EXT. nn	EXT. ADDR. (nn)	REG. INDIR. (SP)
REGISTER	AF										F1
REGISTER	BC								01 n n	ED 4B n n	C1
REGISTER	DE								11 n n	ED 5B n n	D1
REGISTER	HL								21 n n	2A n n	E1
REGISTER	SP				F9		DD F9	FD F9	31 n n	ED 7B n n	
REGISTER	IX								DD 21 n n	DD 2A n n	DD E1
REGISTER	IY								FD 21 n n	FD 2A n n	FD E1
EXT. ADDR.	(nn)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			
PUSH INSTRUCTIONS → REG. IND.	(SP)	F5	C5	D5	E5		DD E5	FD E5			

NOTE: The Push & Pop Instructions adjust the SP after every execution

POP INSTRUCTIONS

CHANGES 'EX' and 'EXX'

Table 5.3-3

		IMPLIED ADDRESSING: AF'	BC', DE' & HL'	HL	IX	IY
IMPLIED	AF	08				
IMPLIED	BC, DE & HL		D9			
IMPLIED	DE			EB		
REG. INDIR.	(SP)			E3	DD E3	FD E3

III

BLOCK TRANSFER GROUP

Table 5.3-4

DESTINATION		SOURCE: REG. INDIR. (HL)	
REG. INDIR.	(DE)	ED A0	'LDI' – Load (DE)←(HL) Inc HL & DE, Dec BC
		ED A0	'LDIR,' – Load (DE)←(HL) Inc HL & DE, Dec BC, Repeat until BC = 0
		ED B0	'LDD' – Load (DE)←(HL) Dec HL & DE, Dec BC
		ED B8	'LDDR' – Load (DE)←(HL) Dec HL & DE, Dec BC, Repeat until BC = 0

Reg HL points to source
Reg DE points to destination
Reg BC is byte counter

Table 5.3-5 specifies the OP codes for the four block search instructions. The first, CPI (compare and increment) compares the data in the accumulator, with the contents of the memory location pointed to by register HL. The result of the compare instruction is stored in one of the flag bits (see section 6.0 for a detailed explanation of the flag operations) and the HL register pair is then incremented and the byte counter (register pair BC) is decremented.

The instruction CPIR is merely an extension of the CPI instruction in which the compare is repeated until either a match is found or the byte counter (register pair BC) becomes zero. Thus, this single instruction can search the entire memory for any 8-bit character.

The CPD (Compare and Decrement) and CPDR (Compare, Decrement and Repeat) are similar instructions, their only difference being that they decrement HL after every compare instruction so that they search the memory in the opposite direction. (The search is started at the highest location in the memory block).

It should be emphasized again that these block transfer and compare instructions are extremely powerful in string manipulation applications.

ARITHMETIC AND LOGICAL

Table 5.3-6 lists all of the 8-bit arithmetic operations that can be performed with the accumulator; also listed are the increment (INC) and decrement (DEC) instructions. In all of these instructions, except INC and DEC, the specified 8-bit operation is performed between the data in the accumulator and the source data specified in the table. The result of the operation is placed in the accumulator with the exception of compare (CP) that leaves the accumulator unaffected. All of these operations affect the flag register as a result of the specified operation. (Section 6.0 provides all of the details on how the flags are affected by any instruction type). INC and DEC instructions specify a register or a memory location as both source and destination of the result. When the source operand is addressed using the index registers, the displacement must follow directly. With immediate addressing the actual operand will follow directly. For example, the instruction

AND 07H

would appear as:

Address A	E6	OP Code
A+1	07	Operand

BLOCK SEARCH GROUP
Table 5.3-5

SEARCH LOCATION	
REG. INDIR.	
(HL)	
ED A1	'CPI' Inc HL, Dec BC
ED B1	'CPIR', Inc HL, Dec BC repeat until BC = 0 or find match
ED A9	'CPD' Dec HL & BC
ED B9	'CPDR' Dec HL & BC Repeat until BC = 0 or find match

HL points to location in memory to be compared with accumulator contents
BC is byte counter

Assuming that the accumulator contained the value F3H, the result of 03H would be placed in the accumulator:

Acc before operation	1111 0011 = F3H
Operand	0000 0111 = 07H
Result to Acc	0000 0011 = 03H

The Add instruction (ADD) performs a binary add between the data in the source location and the data in the accumulator. The subtract (SUB) does a binary subtraction. When the add with carry is specified (ADC) or the subtract with carry (SBC), then the carry flag is also added or subtracted respectively. The flags and decimal adjust instruction (DAA) in the Z80 (fully described in section 6.0) allow arithmetic operations for:

multiprecision packed BCD numbers

multiprecision signed or unsigned binary numbers

multiprecision two's complement signed numbers

Other instructions in this group are: logical and (AND); logical or (OR); exclusive or (XOR), and compare (CP).

There are five general purpose arithmetic instructions that operate on the accumulator or carry flag. These five are listed in Table 5.3-7. The decimal adjust instruction can adjust for subtraction as well as addition, thus making BCD arithmetic operations simple. Note that to allow for this operation, the flag N is used. This flag is set if the last arithmetic operation was a subtract. The negate accumulator (NEG) instruction forms the two's complement of the number in the accumulator. Finally, notice that a reset carry instruction is not included in the Z80 since this operation can be easily achieved through other instructions such as a logical AND of the accumulator with itself.

Table 5.3-8 lists all of the 16-bit arithmetic operations between 16-bit registers. There are five groups of instructions, including add with carry and subtract with carry. ADC and SBC affect all of the flags. These two groups simplify address calculation operations or other 16-bit arithmetic operations.

8 BIT ARITHMETIC AND LOGIC

Table 5.3-6

SOURCE

	REGISTER ADDRESSING							REG. INDIR.	INDEXED		IMMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADD w CARRY 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUBTRACT 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SUB w CARRY 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARE 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

GENERAL PURPOSE AF OPERATIONS

Table 5.3-7

Decimal Adjust Acc, 'DAA'	27
Complement Acc, 'CPL'	2F
Negate Acc, 'NEG' (2's complement)	ED 44
Complement Carry Flag, 'CCF'	3F
Set Carry Flag, 'SCF'	37

16 BIT ARITHMETIC

Table 5.3-8

DESTINATION		SOURCE: BC	DE	HL	SP	IX	IY
'ADD'	HL	09	19	29	39		
'ADD'	IX	DD 09	DD 19		DD 39	DD 29	
'ADD'	IY	FD 09	FD 19		FD 39		FD 29
ADD WITH CARRY AND SET FLAGS 'ADC'	HL	ED 4A	ED 5A	ED 6A	ED 7A		
SUB WITH CARRY AND SET FLAGS 'SBC'	HL	ED 42	ED 52	ED 62	ED 72		
INCREMENT 'INC.		03	13	23	33	DD 23	FD 23
DECREMENT 'DEC'		0B	1B	2B	3B	DD 2B	FD 2B

III

ROTATE AND SHIFT

A major capability of the Z80 is its ability to rotate or shift data in the accumulator, any general purpose register, or any memory location. All of the rotate and shift OP codes are shown in Table 5.3-9. Also included in the Z80 are arithmetic and logical shift operations. These operations are useful in an extremely wide range of applications including integer multiplication and division. Two BCD digit rotate instructions (RRD and RLD) allow a digit in the accumulator to be rotated with the two digits in a memory location pointed to by register pair HL. (See Figure 5.3-9). These instructions allow for efficient BCD arithmetic.

BIT MANIPULATION

The ability to set, reset, and test individual bits in a register or memory location is needed in almost every program. These bits may be flags in a general purpose software routine, may be indications of external control conditions, or may be data packed into memory locations to make memory utilization more efficient.

The Z80 has the ability to set, reset, or test any bit in the accumulator, any general purpose register or any memory location with a single instruction. Table 5.3-10 lists the 240 instructions that are available for this purpose. Register addressing can specify the accumulator or any general purpose register on which the operation is to be performed. Register indirect and indexed addressing are available to operate on external memory locations. Bit test operations set the zero flag (Z) if the tested bit is a zero. (Refer to section 6.0 for further explanation of flag operation).

JUMP, CALL, AND RETURN

Figure 5.3-11 lists all of the jump, call and return instructions implemented in the Z80 CPU. A jump is a branch in a program where the program counter is loaded with the 16-bit value as specified by one of the three available addressing modes (Immediate Extended, Relative, or Register Indirect). Notice that the jump group has several different conditions that can be specified to be met before the jump will be made. If these conditions are not met, the program merely continues with the next sequential instruction. The conditions are all dependent on the data in the flag register. (Refer to section 6.0 for details on the flag register). The immediate extended addressing is used to jump to any location in the memory. This instruction requires three bytes (two to specify the 16-bit address) with the low order address byte first followed by the high order address byte.

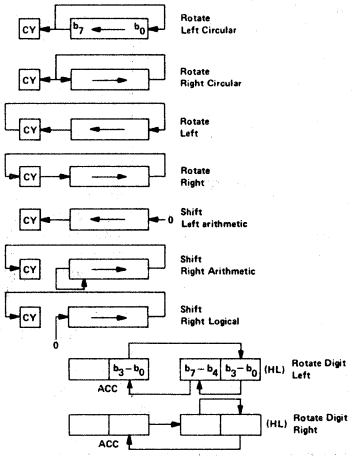
ROTATES AND SHIFTS

Table 5.3-9

Source and Destination

TYPE OF ROTATE OR SHIFT	A	B	C	D	E	H	L	(HL)	(IX + d)	(IY + d)
'RLC'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
'RLD'								ED 6F		
'RRD'								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F



For example an unconditional Jump to memory location 3E32H would be:

Address A	C3	OP Code
A+1	32	Low order address
A+2	3E	High order address

The relative jump instruction uses only two bytes; the second byte is a signed two's complement displacement from the existing PC. This displacement can be in the range of +129 to −126 and is measured from the address of the instruction OP code.

Three types of register indirect jumps are also included. These instructions are implemented by loading the register pair HL or one of the index registers IX or IY directly into the PC. This capability allows for program jumps to be a function of previous calculations.

A call is a special form of a jump where the address of the byte following the call instruction is pushed onto the stack before the jump is made. A return instruction is the reverse of a call because the data on the top of the stack is popped directly into the PC to form a jump address. The call and return instructions allow for simple subroutine and interrupt handling. Two special return instructions have been included in the Z80 family of components. The return from interrupt instruction (RETI) and the return from non-maskable interrupt (RETN) are treated in the CPU as an unconditional return identical to the OP code C9H. The difference is that (RETI) can be used at the end of an interrupt routine and all Z80 peripheral chips will recognize the execution of this instruction for proper control of nested priority interrupt handling. This instruction coupled with the Z80 peripheral devices' implementation simplifies the normal return from nested interrupt. Without this feature, the following software sequence would be necessary to inform the interrupting device that the interrupt routine has been completed:

BIT MANIPULATION GROUP
Table 5.3-10

		REGISTER ADDRESSING							REG. INDIR.	INDEXED	
	BIT	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST 'BIT'	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	GB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB F8	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

III

Disable Interrupt	— prevent interrupt before routine is exited.
LD A,n OUT n, A	— notify peripheral that service routine is complete
Enable Interrupt	
Return	

This seven byte sequence can be replaced with the three byte EI RETI instruction sequence in the Z80. This is important since interrupt service time often must be minimized.

To facilitate program loop control, the instruction DJNZ e can be used advantageously. This two byte, relative jump instruction decrements the B register, and the jump occurs if the B register has not been decremented to zero. The relative displacement is expressed as a signed two's complement number. A simple example of its use might be:

Address	Instruction	Comments
N,N+1	LD B,7	; set B register to count of 7
N+2 to N+9	(Perform a sequence of instructions)	; loop to be performed 7 times
N+10, N+11	DJNZ −10	; to jump from N + 12 to N + 2
N + 12	(Next Instruction)	

JUMP, CALL, AND RETURN GROUP

Table 5.3-11

CONDITION

			UN-COND.	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	REG B≠0
JUMP 'JP'	IMMED. EXT.	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVE	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JUMP 'JP'	REG. INDIR.	(HL)	E9									
JUMP 'JP'		(IX)	DD E9									
JUMP 'JP'		(IY)	FD E9									
'CALL'	IMMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENT B, JUMP IF NON ZERO 'DJNZ'	RELATIVE	PC+e										10 e-2
RETURN 'RET'	REGISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETURN FROM INT 'RETI'	REG. INDIR.	(SP) (SP+1)	ED 4D									
RETURN FROM NON MASKABLE INT 'RETN'	REG. INDIR.	(SP) (SP+1)	ED 45									

NOTE—CERTAIN FLAGS HAVE MORE THAN ONE PURPOSE. REFER TO SECTION 6.0 FOR DETAILS

Table 5.3-12 lists the eight OP codes for the restart instruction. This instruction is a single byte call to any of the eight addresses listed. The simple mnemonic for these eight calls is also shown. The value of this instruction is that frequently used routines can be called with this instruction to minimize memory usage.

RESTART GROUP
Table 5.3-12

CALL ADDRESS	OP CODE	
0000_H	C7	'RST 0'
0008_H	CF	'RST 8'
0010_H	D7	'RST 16'
0018_H	DF	'RST 24'
0020_H	E7	'RST 32'
0028_H	EF	'RST 40'
0030_H	F7	'RST 48'
0038_H	FF	'RST 56'

III

INPUT/OUTPUT

The Z80 has an extensive set of Input and Output instructions, as shown in table 5.3-13 and table 5.3-14. The addressing of the input or output device can be either absolute or register indirect, using the C register. Notice that in the register indirect addressing mode, data can be transferred between the I/O devices and any of the internal registers. In addition eight block transfer instructions have been implemented. These instructions are similar to the memory block transfers except that they use register pair HL for a pointer to the memory source (output commands) or destination (input commands), while register B is used as a byte counter. Register C holds the address of the port for which the input or output command is desired. Since register B is eight bits in length, the I/O block transfer command handles up to 256 bytes.

In the instructions IN A, n and OUT n, A, an I/O device address n appears in the lower half of the address bus (A_0-A_7) while the accumulator content is transferred in the upper half of the address bus. In all register indirect input output instructions, including block I/O transfers, the content of register C is transferred to the lower half of the address bus (device address) while the content of register B is transferred to the upper half of the address bus.

INPUT GROUP

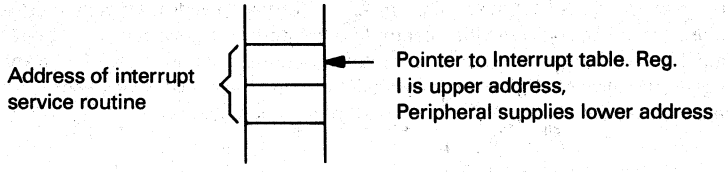
Table 5.3-13

INPUT DESTINATION			PORT ADDRESS IMMED. n	PORT ADDRESS REG. INDIR. (C)
INPUT 'IN'	REG ADDRESSING	A	DB	ED 78
		B		ED 40
		C		ED 48
		D		ED 50
		E		ED 58
		H		ED 60
		L		ED 68
'INI' – INPUT & Inc HL, Dec B	REG, INDIR	(HL)		ED A2
'INIR'– INP, Inc HL, Dec B, REPEAT IF B≠0				ED B2
'IND'– INPUT & Dec HL, Dec B				ED AA
'INDR'– INPUT, Dec HL, Dec B, REPEAT IF B≠0				ED BA

BLOCK INPUT COMMANDS

CPU CONTROL GROUP

The final table, table 5.3-15, illustrates the six general purpose CPU control instructions. The NOP is a do-nothing instruction. The HALT instruction suspends CPU operation until a subsequent interrupt is received, while the DI and EI are used to lock out and enable interrupts. The three interrupt mode commands set the CPU into any of the three available interrupt response modes as follows. If mode zero is set, the interrupting device can insert any instruction on the data bus and allow the CPU to execute it. Mode 1 is a simplified mode where the CPU automatically executes a restart (RST) to location 0038H so that no external hardware is required. (The old PC content is pushed onto the stack). Mode 2 is the most powerful in that it allows for an indirect call to any location in memory. With this mode, the CPU forms a 16-bit memory address where the upper 8-bits are the content of mode, the CPU forms a 16-bit memory address where the upper 8-bits are the content of register I, and the lower 8-bits are supplied by the interrupting device. This address points to the first of two sequential bytes in a table where the address of the service routine is located. The CPU automatically obtains the starting address and performs a CALL to this address.



OUTPUT GROUP
Table 5.3-14

			SOURCE							
			REGISTER							REG. IND.
			A	B	C	D	E	H	L	(HL)
'OUT'	IMMED.	n	D3 n							
	REG. IND.	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69	
'OUTI' – OUTPUT Inc HL, Dec b	REG. IND.	(C)								ED A3
'OTIR' – OUTPUT, Inc HL, Dec B, REPEAT IF B≠0	REG. IND.	(C)								ED B3
'OUTD' – OUTPUT Dec HL & B	REG. IND.	(C)								ED AB
'OTDR' – OUTPUT, Dec HL & B, REPEAT IF B≠0	REG. IND.	(C)								ED BB

BLOCK OUTPUT COMMANDS

PORT DESTINATION ADDRESS

MISCELLANEOUS CPU CONTROL
Table 5.3-15

'NOP'	00	
'HALT'	76	
DISABLE INT '(DI)'	F3	
ENABLE INT '(EI)'	FB	
SET INT MODE 0 'IM0'	ED 46	8080A MODE
SET INT MODE 1 'IM1'	ED 56	CALL TO LOCATION 0038_H
SET INT MODE 2 'IM2'	ED 5E	INDIRECT CALL USING REGISTER I AND 8 BITS FROM INTERRUPTING DEVICE AS A POINTER.

6.0 FLAGS

Each of the two Z80-CPU Flag registers contains six bits of information which are set or reset by various CPU operations. Four of these bits are testable; that is, they are used as conditions for jump, call, or return instructions. For example, a jump may be desired only if a specific bit in the flag register is set. The four testable flag bits are:

1) Carry Flag (C) — This flag is the carry from the highest order bit of the accumulator. For example, the carry flag will be set during an add instruction where a carry from the highest bit of the accumulator is generated. This flag is also set if a borrow is generated during a subtraction instruction. The shift and rotate instructions also affect this bit.

2) Zero Flag (Z) — This flag is set if the result of the operation loaded a zero into the accumulator. Otherwise the flag is reset.

3) Sign Flag (S) — This flag is intended to be used with signed numbers, and it is set if the result of the operation was negative. Since bit 7 (MSB) represents the sign of the number (A negative number has a 1 in bit 7), this flag stores the state of bit 7 in the accumulator.

4) Parity/Overflow Flag (P/V) — This dual purpose flag indicates the parity of the result in the accumulator when logical operations are performed (such as AND A, B) and it represents overflow when signed two's complement arithmetic operations are performed. The Z80 overflow flag indicates that the two's complement number in the accumulator is in error since it has exceeded the maximum possible (+127) or is less than the minimum possible (–128) number that can be represented by two's complement notation. For example consider adding:

```
+120 =      0111 1000
+105 =      0110 1001
-------     ---------
  C = 0     1110 0001 = -95 (wrong) Overflow has occurred
```

Here the result is incorrect. Overflow has occurred and yet there is no carry to indicate an error. For this case, the overflow flag would be set. Also consider the addition of two negative numbers.

```
  -5 =      1111 1011
 -16 =      1111 0000
-------     ---------
  C = 1     1110 1011 = -21 correct
```

Notice that the answer is correct, but the carry is set so that this flag cannot be used as an overflow indicator. In this case, the overflow would not be set.

For logical operations (AND, OR, XOR), this flag is set if the parity of the result is even, and the flag is reset if it is odd.

There are also two non-testable bits in the flag register. Both of these are used for BCD arithmetic. They are:

1) Half carry (H) — This is the BCD carry or borrow result from the least significant four bits of operation. When using the DAA (Decimal Adjust Instruction) this flag is used to correct the result of a previous packed decimal add or subtract.

2) Add/Subtract Flag (N) — Since the algorithm for correcting BCD operations if different for addition or subtraction, this flag is used to specify what type of instruction was executed last so that the DAA operation will be correct for either addition or subtraction.

The Flag register can be accessed by the programmer, and its form is as follows:

D7							D0
S	Z	X	H	X	P/V	N	C

Table 6.0-1 lists how each flag bit is affected by various CPU instructions. In this table, '•' indicates that the instruction does not change the flag; an 'X' means that the flag goes to an indeterminate state; an '0' means that it is reset; a '1' means that it is set, and the symbol ↕ indicates that it is set for reset according to the previous discussion. Note that any instruction not appearing in this table does not affect any of the flags.

Table 6.0-1 includes a few special cases that must be described for clarity. Notice that the block search instruction sets the Z flag if the last compare operation indicated a match between the source and the accumulator data. Also, the parity flag is set if the byte counter (register pair BC) is not equal to zero. This same use of the parity flag is made with the block move instructions. Another special case is during block input or output instructions. Here the Z flag is used to indicate the state of register B which is used as a byte counter. Notice that when the I/O block transfer is complete, the zero flag will be reset to a zero (i.e. B=0), while in the case of a block move command, the parity flag is reset when the operation is complete. A final case occurs when the refresh or I register is loaded into the accumulator, because interrupt enable flip flop is then loaded into the parity flag so that the complete state of the CPU can be saved at any time.

SUMMARY OF FLAG OPERATION

Table 6.0-1

Instruction	D7 S	Z		H		P/V	N	D0 C	Comments
ADD A,s; ADC A,s	‡	‡	X	‡	X	V	0	‡	8-bit add or add with carry
SUB,s; SBCA,s; CP,s; NEG	‡	‡	X	‡	X	V	1	‡	8-bit subtract, subtract with carry, compare and negate accumulator
AND s	‡	‡	X	1	X	P	0	0	Logical operations
OR s; XOR s	‡	‡	X	0	X	P	0	0	Logical operations
INC s	‡	‡	X	‡	X	V	0	•	8-bit increment
DEC s	‡	‡	X	‡	X	V	1	•	8-bit decrement
ADD DD, SS	•	•	X	X	X	•	0	‡	16-bit add
ADC HL, SS	‡	‡	X	X	X	V	0	‡	16-bit add with carry
SBC HL, SS	‡	‡	X	X	X	V	1	‡	16-bit subtract with carry
RLA; RLCA; RRA; RRCA	•	•	X	0	X	•	0	‡	Rotate accumulator
RL s; RLC s; RR s; RRC s; SLA s; SRA s; SRL s	‡	‡	X	0	X	P	0	‡	Rotate and shift locations
RLD; RRD	‡	‡	X	0	X	P	0	•	Rotate digit left and right
DAA	‡	‡	X	‡	X	P	•	‡	Decimal adjust accumulator
CPL	•	•	X	1	X	•	1	•	Complement accumulator
SCF	•	•	X	0	X	•	0	1	Set carry
CCF	•	•	X	X	X	•	0	‡	Complement carry
IN r, (C)	‡	‡	X	0	X	P	0	•	Input register indirect
INI; IND; OUTI; OUTD	X	‡	X	X	X	X	1	X	Block input and output Z = 0 if B ≠ 0 otherwise Z = 1; if bit 7 = 1, N = 1
INIR; INDR; OTIR; OTDR	X	1	X	X	X	X	1	X	Block input and output Z = 0 if B ≠ 0 otherwise Z = 1; if bit 7 = 1, N = 1
LDI; LDD	X	X	X	0	X	‡	0	•	Block transfer instructions P/V = 1 if BC ≠ 0, otherwise P/V = 0
LDIR; LDDR	X	X	X	0	X	0	0	•	Block transfer instructions P/V = 1 if BC ≠ 0, otherwise P/V = 0
CPI; CPIR; CPD; CPDR	‡	‡	‡	X	X	‡	1	•	Block search instructions Z = 1 if A = (HL), otherwise Z = 0 P/V = 1 if BC ≠ 0, otherwise P/V = 0
LD A, I; LD A, R	‡	‡	X	0	X	IFF	0	•	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag
BIT b, s	X	‡	X	1	X	X	0	•	The state of bit b of location s is copied into the Z flag

The following notation is used in this table:

SYMBOL	OPERATION
C	Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result.
Z	Zero flag. Z=1 if the result of the operation is zero.
S	Sign flag. S=1 if the MSB of the result is one.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result, while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V=1 if the result of the operation is even, P/V=0 if result is odd. If P/V holds overflow, P/V= if the result of the operation produced an overflow.
H	Half-carry flag. H=1 if the add or subtract operation produced a carry into or borrow from bit 4 of the accumulator.
N	Add/Subtract flag. N=1 if the previous operation was a subtract. H and N flags are used in conjunction with the decimal adjust instruction (DAA) to correct properly the result into packed BCD format following addition or subtraction using operands with packed BCD format. The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care".
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction.
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range <0, 255>
nn	16-bit value in range <0, 65535>

7.0 SUMMARY OF OP CODES AND EXECUTION TIMES

The following section gives a summary of the Z80 instruction set. The instructions are logically arranged into groups as shown on Tables 7.0-1 through 7.0-11. Each table shows the assembly language mnemonic OP code, the actual OP code, the symbolic operation, the content of the flag register following the execution of each instruction, the number of bytes required for each instruction as well as the number of memory cycles and the total number of T states (external clock periods) required for the fetching and execution of each instruction. Care has been taken to make each table self-explanatory without requiring any cross reference with the text or other tables.

8-BIT LOAD GROUP
Table 7.0-1

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD r, s	r ← s	•	•	X	•	X	•	•	•	01 r s		1	1	4	r, s Reg.
LD r, n	r ← n	•	•	X	•	X	•	•	•	00 r 110		2	2	7	000 B
										← n →					001 C
LD r, (HL)	r ← (HL)	•	•	X	•	X	•	•	•	01 r 110		1	2	7	010 D
LD r, (IX+d)	r ← (IX+d)	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19	011 E
										01 r 110					100 H
										← d →					101 L
LD r, (IY+d)	r ← (IY+d)	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19	111 A
										01 r 110					
										← d →					
LD (HL), r	(HL) ← r	•	•	X	•	X	•	•	•	01 110 r		1	2	7	
LD (IX+d), r	(IX+d) ← r	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19	
										01 110 r					
										← d →					
LD (IY+d), r	(IY+d) ← r	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19	
										01 110 r					
										← d →					
LD (HL), n	(HL) ← n	•	•	X	•	X	•	•	•	00 110 110	36	2	3	10	
										← n →					
LD (IX+d), n	(IX+d) ← n	•	•	X	•	X	•	•	•	11 011 101	DD	4	5	19	
										00 110 110	36				
										← d →					
										← n →					
LD (IY+d), n	(IY+d) ← n	•	•	X	•	X	•	•	•	11 111 101	FD	4	5	19	
										00 110 110	36				
										← d →					
										← n →					
LD A, (BC)	A ← (BC)	•	•	X	•	X	•	•	•	00 001 010	0A	1	2	7	
LD A, (DE)	A ← (DE)	•	•	X	•	X	•	•	•	00 011 010	1A	1	2	7	
LD A, (nn)	A ← (nn)	•	•	X	•	X	•	•	•	00 111 010	3A	3	4	13	
										← n →					
										← n →					
LD (BC), A	(BC) ← A	•	•	X	•	X	•	•	•	00 000 010	02	1	2	7	
LD (DE), A	(DE) ← A	•	•	X	•	X	•	•	•	00 010 010	12	1	2	7	
LD (nn), A	(nn) ← A	•	•	X	•	X	•	•	•	00 110 010	32	3	4	13	
										← n →					
										← n →					
LD A, I	A ← I	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9	
										01 010 111	57				
LD A, R	A ← R	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9	
										01 011 111	5F				
LD I, A	I ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9	
										01 000 111	47				
LD R, A	R ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9	
										01 001 111	4F				

Notes: r, s means any of the registers A, B, C, D, E, H, L
IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

16-BIT LOAD GROUP
Table 7.0-2

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD dd, nn	dd ← nn	•	•	X	•	X	•	•	•	00 dd0 001 ← n → ← n →		3	3	10	dd Pair 00 BC 01 DE
LD IX, nn	IX ← nn	•	•	X	•	X	•	•	•	11 011 101 00 100 001 ← n → ← n →	DD 21	4	4	14	10 HL 11 SP
LD IY, nn	IY ← nn	•	•	X	•	X	•	•	•	11 111 101 00 100 001 ← n → ← n →	FD 21	4	4	14	
LD HL, (nn)	H ← (nn+1) L ← (nn)	•	•	X	•	X	•	•	•	00 101 010 ← n → ← n →	2A	3	5	16	
LD dd, (nn)	dd_H ← (nn+1) dd_L ← (nn)	•	•	X	•	X	•	•	•	11 101 101 01 dd1 011 ← n → ← n →	ED	4	6	20	
LD IX, (nn)	IX_H ← (nn+1) IX_L ← (nn)	•	•	X	•	X	•	•	•	11 011 101 00 101 010 ← n → ← n →	DD 2A	4	6	20	
LD IY, (nn)	IY_H ← (nn+1) IY_L ← (nn)	•	•	X	•	X	•	•	•	11 111 101 00 101 010 ← n → ← n →	FD 2A	4	6	20	
LD (nn), HL	(nn+1) ← H (nn) ← L	•	•	X	•	X	•	•	•	00 100 010 ← n → ← n →	22	3	5	16	
LD (nn), dd	(nn+1) ← dd_H (nn) ← dd_L	•	•	X	•	X	•	•	•	11 101 101 01 dd0 011 ← n → ← n →	ED	4	6	20	
LD (nn), IX	(nn+1) ← IX_H (nn) ← IX_L	•	•	X	•	X	•	•	•	11 011 101 00 100 010 ← n → ← n →	DD 22	4	6	20	
LD (nn), IY	(nn+1) ← IY_H (nn) ← IY_L	•	•	X	•	X	•	•	•	11 111 101 00 100 010 ← n → ← n →	FD 22	4	6	20	
LD SP, HL	SP ← HL	•	•	X	•	X	•	•	•	11 111 001	F9	1	1	6	
LD SP, IX	SP ← IX	•	•	X	•	X	•	•	•	11 011 101 11 111 001	DD F9	2	2	10	
LD SP, IY	SP ← IY	•	•	X	•	X	•	•	•	11 111 101 11 111 001	FD F9	2	2	10	
PUSH qq	(SP-2) ← qq_L (SP-1) ← qq_H	•	•	X	•	X	•	•	•	11 qq0 101		1	3	11	qq Pair 00 BC 01 DE
PUSH IX	(SP-2) ← IX_L (SP-1) ← IX_H	•	•	X	•	X	•	•	•	11 011 101 11 100 101	DD E5	2	4	15	10 HL 11 AF
PUSH IY	(SP-2) ← IY_L (SP-1) ← IY_H	•	•	X	•	X	•	•	•	11 111 101 11 100 101	FD E5	2	4	15	
POP qq	qq_H ← (SP+1) qq_L ← (SP)	•	•	X	•	X	•	•	•	11 qq0 001		1	3	10	
POP IX	IX_H ← (SP+1) IX_L ← (SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 001	DD E1	2	4	14	
POP IY	IY_H ← (SP+1) IY_L ← (SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 001	FD E1	2	4	14	

Notes: dd is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs AF, BC, DE, HL
$(PAIR)_H$, $(PAIR)_L$ refer to high order and low order eight bits of the register pair respectively.
e.g. BC_L = C, AF_H = A

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ flag is affected according to the result of the operation.

III

EXCHANGE GROUP AND BLOCK TRANSFER AND SEARCH GROUP

Table 7.0-3

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE↔HL	•	•	X	•	X	•	•	•	11 101 011	EB	1	1	4	
EX AF, AF'	AF↔AF'	•	•	X	•	X	•	•	•	00 001 000	08	1	1	4	
EXX	(BC↔BC' DE↔DE' HL↔HL')	•	•	X	•	X	•	•	•	11 011 001	D9	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔(SP+1) L ↔(SP)	•	•	X	•	X	•	•	•	11 100 011	E3	1	5	19	
EX (SP), IX	IX_H ↔(SP+1) IX_L ↔(SP)	•	•	X	•	X	•	•	•	11 011 101 11 100 011	DD E3	2	6	23	
EX (SP), IY	IY_H ↔(SP+1) IY_L ↔(SP)	•	•	X	•	X	•	•	•	11 111 101 11 100 011	FD E3	2	6	23	
LDI	(DE)←(HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	X	0	X	① ↕	0	•	11 101 101 10 100 000	ED A0	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE)←(HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 110 000	ED B0	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE)←(HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	X	0	X	① ↕	0	•	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE)←(HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A – (HL) HL ← HL+1 BC ← BC-1	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A – (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A – (HL) HL ← HL-1 BC ← BC-1	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A – (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	↕	② ↕	X	↕	X	① ↕	1	•	11 101 101 10 111 001	ED B9	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

Notes: ① P/V flag is 0 if the result of BC-1 = 0, otherwise P/V = 1
② Z flag is 1 if A = (HL), otherwise Z = 0.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

8-BIT ARITHMETIC AND LOGICAL GROUP

Table 7.0-4

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD A, r	A ← A + r	‡	‡	X	‡	X	V	0	‡	10 [000] r		1	1	4	r Reg.
ADD A, n	A ← A + n	‡	‡	X	‡	X	V	0	‡	11 [000] 110 ← n →		2	2	7	000 B 001 C 010 D
ADD A, (HL)	A ← A+(HL)	‡	‡	X	‡	X	V	0	‡	10 [000] 110		1	2	7	011 E
ADD A, (IX+d)	A←A+(IX+d)	‡	‡	X	‡	X	V	0	‡	11 011 101 10 [000] 110 ← d →	DD	3	5	19	100 H 101 L 111 A
ADD A, (IY+d)	A←A+(IY+d)	‡	‡	X	‡	X	V	0	‡	11 111 101 10 [000] 110 ← d →	FD	3	5	19	
ADC A, s	A←A+s+CY	‡	‡	X	‡	X	V	0	‡	[001]					s is any of r, n,
SUB s	A←A - s	‡	‡	X	‡	X	V	1	‡	[010]					(HL), (IX+d),
SBC A, s	A←A - s - CY	‡	‡	X	‡	X	V	1	‡	[011]					(IY+d) as shown for
AND s	A←A ∧ s	‡	‡	X	1	X	P	0	0	[100]					ADD instruction.
OR s	A←A ∨ s	‡	‡	X	0	X	P	0	0	[110]					The indicated bits
XOR s	A←A ⊕ s	‡	‡	X	0	X	P	0	0	[101]					replace the [000] in
CP s	A - s	‡	‡	X	‡	X	V	1	‡	[111]					the ADD set above.
INC r	r ← r + 1	‡	‡	X	‡	X	V	0	•	00 r [100]		1	1	4	
INC (HL)	(HL)←(HL)+1	‡	‡	X	‡	X	V	0	•	00 110 [100]		1	3	11	
INC (IX+d)	(IX+d) ← (IX+d)+1	‡	‡	X	‡	X	V	0	•	11 011 101 00 110 [100] ← d →	DD	3	6	23	
INC (IY+d)	(IY+d) ← (IY+d)+1	‡	‡	X	‡	X	V	0	•	11 111 101 00 110 [100] ← d →	FD	3	6	23	
DEC s	s ← s - 1	‡	‡	X	‡	X	V	1	•	[101]		1	1	4	s is any of r, (HL), (IX+d), (IY+d) as shown for INC. DEC same format and states as INC. Replace [100] with [101] in OP Code.

Notes: The V symbol in the P/V flag column indicates that the P/V flag contains the overflow of the result of the operation. Similarly the P symbol indicates parity. V = 1 means overflow, V = 0 means not overflow, P = 1 means parity of the result is even, P = 0 means parity of the result is odd.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
‡ = flag is affected according to the result of the operation.

III

GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS
Table 7.0-5

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
DAA	Converts acc, content into packed BCD following add or subtract with packed BCD operands	↕	↕	X	↕	X	P	•	↕	00 100 111	27	1	1	4	Decimal adjust accumulator
CPL	$A \rightarrow \overline{A}$	•	•	X	1	X	•	1	•	00 101 111	2F	1	1	4	Complement accumulator (One's complement)
NEG	$A \rightarrow \overline{A}+1$	↕	↕	X	↕	X	V	1	↕	11 101 101 01 000 100	ED 44	2	2	8	Negate acc, (two's complement)
CCF	$CY \rightarrow \overline{CY}$	•	•	X	X	X	•	0	↕	00 111 111	3F	1	1	4	Complement carry flag
SCF	$CY \rightarrow 1$	•	•	X	0	X	•	0	1	00 110 111	37	1	1	4	Set carry flag
NOP	No operation	•	•	X	•	X	•	•	•	00 000 000	00	1	1	4	
HALT	CPU halted	•	•	X	•	X	•	•	•	01 110 110	76	1	1	4	
DI*	IFF → 0	•	•	X	•	X	•	•	•	11 110 011	F3	1	1	4	
EI*	IFF → 1	•	•	X	•	X	•	•	•	11 111 011	FB	1	1	4	
IM 0	Set interrupt mode 0	•	•	X	•	X	•	•	•	11 101 101 01 000 110	ED 46	2	2	8	
IM 1	Set interrupt mode 1	•	•	X	•	X	•	•	•	11 101 101 01 010 110	ED 56	2	2	8	
IM 2	Set interrupt mode 2	•	•	X	•	X	•	•	•	11 101 101 01 011 110	ED 5E	2	2	8	

Notes: IFF indicates the interrupt enable flip-flop
CY indicates the carry flip-flop.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

*Interrupts are not sampled at the end of EI or DI

16-BIT ARITHMETIC GROUP
Table 7.0-6

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments	
ADD HL, ss	HL ← HL+ss	•	•	X	X	X	•	0	‡	00 ss1 001		1	3	11	ss	Reg.
															00	BC
ADC HL, ss	HL ← HL+ss+CY	‡	‡	X	X	X	V	0	‡	11 101 101	ED	2	4	15	01	DE
										01 ss1 010					10	HL
															11	SP
SBC HL, ss	HL ← HL-ss-CY	‡	‡	X	X	X	V	1	‡	11 101 101	ED	2	4	15		
										01 ss0 010						
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	‡	11 011 101	DD	2	4	15	pp	Reg.
										00 pp1 001					00	BC
															01	DE
															10	IX
															11	SP
ADD IY, rr	IY ← IY + rr	•	•	X	X	X	•	0	‡	11 111 101	FD	2	4	15	rr	Reg.
										00 rr1 001					00	BC
															01	DE
															10	IY
															11	SP
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00 ss0 011		1	1	6		
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10		
										00 100 011	23					
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10		
										00 100 011	23					
DEC ss	ss ← ss - 1	•	•	X	•	X	•	•	•	00 ss1 011		1	1	6		
DEC IX	IX ← IX - 1	•	•	X	•	X	•	•	•	11 011 101	DD	2	2	10		
										00 101 011	2B					
DEC IY	IY ← IY - 1	•	•	X	•	X	•	•	•	11 111 101	FD	2	2	10		
										00 101 011	2B					

Notes: ss is any of the register pairs BC, DE, HL, SP
pp is any of the register pairs BC, DE, IX, SP
rr is any of the register pairs BC, DE, IY, SP.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
‡ = flag is affected according to the result of the operation.

III

ROTATE AND SHIFT GROUP
Table 7.0-7

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
RLCA	CY ← 7 ← 0 ← A	•	•	X	0	X	•	0	‡	00 000 111	07	1	1	4	Rotate left circular accumulator
RLA	CY ← 7 ← 0 ← A	•	•	X	0	X	•	0	‡	00 010 111	17	1	1	4	Rotate left accumulator
RRCA	7 → 0 → CY, A	•	•	X	0	X	•	0	‡	00 001 111	0F	1	1	4	Rotate right circular accumulator
RRA	7 → 0 → CY, A	•	•	X	0	X	•	0	‡	00 011 111	1F	1	1	4	Rotate right accumulator
RLC r		‡	‡	X	0	X	P	0	‡	11 001 011 00 000 r	CB	2	2	8	Rotate left circular register r
RLC (HL)		‡	‡	X	0	X	P	0	‡	11 001 011 00 000 110	CB	2	4	15	r Reg. 000 B 001 C
RLC (IX+d)	CY ← 7 ← 0 r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	11 011 101 11 001 011 ← d → 00 000 110	DD CB	4	6	23	010 D 011 E 100 H 101 L 111 A
RLC (IY+d)		‡	‡	X	0	X	P	0	‡	11 111 101 11 001 011 ← d → 00 000 110	FD CB	4	6	23	
RL s	CY ← 7 ← 0 s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	010					Instruction format and states are as shown for RLC's. To form new Op-Code replace 000 of RLC's with shown code
RRC s	7 → 0 → CY s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	001					
RR s	7 → 0 → CY s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	011					
SLA s	CY ← 7 ← 0 ← 0 s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	100					
SRA s	7 → 0 → CY s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	101					
SRL s	0 → 7 → 0 → CY s ≡ r,(HL),(IX+d),(IY+d)	‡	‡	X	0	X	P	0	‡	111					
RLD	A 7-4 3-0 7-4 3-0 (HL)	‡	‡	X	0	X	P	0	•	11 101 101 01 101 111	ED 6F	2	5	18	Rotate digit left and right between the accumulator and location (HL).
RRD	A 7-4 3-0 7-4 3-0 (HL)	‡	‡	X	0	X	P	0	•	11 101 101 01 100 111	ED 67	2	5	18	The content of the upper half of the accumulator is unaffected

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

BIT SET, RESET AND TEST GROUP
Table 7.0-8

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No.of M Cycles	No.of T States	Comments
BIT b, r	$Z \leftarrow \overline{r_b}$	X	‡	X	1	X	X	0	•	11 001 011 01 b r	CB	2	2	8	
BIT b, (HL)	$Z \leftarrow \overline{(HL)_b}$	X	‡	X	1	X	X	0	•	11 001 011 01 b 110	CB	2	3	12	
BIT b, $(IX+d)_b$	$Z \leftarrow \overline{(IX+d)_b}$	X	‡	X	1	X	X	0	•	11 011 101 11 001 011 ← d → 01 b 110	DD CB	4	5	20	
BIT b, $(IY+d)_b$	$Z \leftarrow \overline{(IY+d)_b}$	X	‡	X	1	X	X	0	•	11 111 101 11 001 011 ← d → 01 b 110	FD CB	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011 $\boxed{11}$ b r	CB	2	2	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011 $\boxed{11}$ b 110	CB	2	4	15	
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 011 101 11 001 011 ← d → $\boxed{11}$ b 110	DD CB	4	6	23	
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 111 101 11 001 011 ← d → $\boxed{11}$ b 110	FD CB	4	6	23	
RES b, s	$s_b \leftarrow 0$ $s \equiv r$, (HL), (IX+d), (IY+d)	•	•	X	•	X	•	•	•	$\boxed{10}$					To form new Op-Code replace $\boxed{11}$ of SET b, s with $\boxed{10}$. Flags and time states for SET instruction

r	Reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

b	Bit Tested
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Notes: The notation s_b indicates bit b (0 to 7) or location s.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

III

JUMP GROUP
Table 7.0-9

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	X	•	X	•	•	•	11 000 011 ← n → ← n →	C3	3	3	10	
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	X	•	X	•	•	•	11 cc 010 ← n → ← n →		3	3	10	cc — Condition 000 NZ non zero 001 Z zero 010 NC non carry 011 C carry 100 PO parity odd 101 PE parity even 110 P sign positive 111 M sign negative
JR e	PC ← PC + e	•	•	X	•	X	•	•	•	00 011 000 ← e-2 →	18	2	3	12	
JR C, e	If C = 0, continue	•	•	X	•	X	•	•	•	00 111 000 ← e-2 →	38	2	2	7	If condition not met
	If C = 1, PC ← PC+e											2	3	12	If condition is met
JR NC, e	If C = 1, continue	•	•	X	•	X	•	•	•	00 110 000 ← e-2 →	30	2	2	7	If condition not met
	If C = 0, PC ← PC+e											2	3	12	If condition is met
JR Z, e	If Z = 0 continue	•	•	X	•	X	•	•	•	00 101 000 ← e-2 →	28	2	2	7	If condition not met
	If Z = 1, PC ← PC+e											2	3	12	If condition is met
JR NZ, e	If Z = 1, continue	•	•	X	•	X	•	•	•	00 100 000 ← e-2 →	20	2	2	7	If condition not met
	If Z = 0, PC ← PC+e											2	3	12	If condition is met
JP (HL)	PC ← HL	•	•	X	•	X	•	•	•	11 101 001	E9	1	1	4	
JP (IX)	PC ← IX	•	•	X	•	X	•	•	•	11 011 101 11 101 001	DD E9	2	2	8	
JP (IY)	PC ← IY	•	•	X	•	X	•	•	•	11 111 101 11 101 001	FD E9	2	2	8	
DJNZ, e	B ← B-1 If B = 0, continue	•	•	X	•	X	•	•	•	00 010 000 ← e-2 →	10	2	2	8	If B = 0
	If B ≠ 0, PC ← PC+e											2	3	13	If B ≠ 0

Notes: e represents the extension in the relative addressing mode.

e is a signed two's complement number in the range <126, 129>

e-2 in the op-code provides an effective address of pc+e as PC is incremented by 2 prior to the addition of e.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

CALL AND RETURN GROUP
Table 7.0-10

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No. of Bytes	No.of M Cycles	No.of T States	Comments
CALL nn	(SP-1) ← PC_H (SP-2) ← PC_L PC ← nn	•	•	X	•	X	•	•	•	11 001 101 ← n → ← n →	CD	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	X	•	X	•	•	•	11 cc 100 ← n → ← n →		3 3	3 5	10 17	If cc is false If cc is true
RET	PC_L ← (SP) PC_H ← (SP+1)	•	•	X	•	X	•	•	•	11 001 001	C9	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	X	•	X	•	•	•	11 cc 000		1 1	1 3	5 11	If cc is false If cc is true
RETI	Return from interrupt	•	•	X	•	X	•	•	•	11 101 101 01 001 101	ED 4D	2	4	14	
RETN[1]	Return from non maskable interrupt	•	•	X	•	X	•	•	•	11 101 101 01 000 101	ED 45	2	4	14	
RST p	(SP-1) ← PC_H (SP-2) ← PC_L PC_H ← 0 PC_L ← p	•	•	X	•	X	•	•	•	11 t 111		1	3	11	

cc	Condition	
000	NZ	non zero
001	Z	zero
010	NC	non carry
011	C	carry
100	PO	parity odd
101	PE	parity even
110	P	sign positive
111	M	sign negative

t	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

[1] RETN loads IFF_2 → IFF_1

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

III

INPUT AND OUTPUT GROUP
Table 7.0-11

Mnemonic	Symbolic Operation	Flags S	Z		H		P/V	N	C	Op-Code 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
IN A, (n)	A ← (n)	•	•	X	•	X	•	•	•	11 011 011 ← n ←	DB	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	↕	↕	X	↕	X	P	0	•	11 101 101 01 r 000	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	X	① ↕	X	X	X	X	1	X	11 101 101 10 100 010	ED A2	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 110 010	ED B2	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	X	① ↕	X	X	X	X	1	X	11 101 101 10 101 010	ED AA	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 111 010	ED BA	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUT (n), A	(n) ← A	•	•	X	•	X	•	•	•	11 010 011	D3	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
OUT (C), r	(C) ← r	•	•	X	•	X	•	•	•	11 101 101 01 r 001	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTI	B ← B - 1 (C) ← (HL) HL ← HL + 1	X	① ↕	X	X	X	X	1	X	11 101 101 10 100 011	ED A3	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTIR	B ← B - 1 (C) ← (HL) HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 110 011	ED B3	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	X	① ↕	X	X	X	X	1	X	11 101 101 10 101 011	ED AB	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	X	1	X	11 101 101 10 111 011	ED BB	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$

Notes: ① If the result of B - 1 is zero the Z flag is set, otherwise it is reset.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

8.0 INTERRUPT RESPONSE

The purpose of an interrupt is to allow peripheral devices to suspend CPU operation in an orderly manner and force the CPU to start a peripheral service routine. Usually this service routine is involved with the exchange of data, or status and control information, between the CPU and the peripheral. Once the service routine is completed, the CPU returns to the operation from which it was interrupted.

INTERRUPT ENABLE — DISABLE

The Z80-CPU has two interrupt inputs, a software maskable interrupt and a non-maskable interrupt. The non-maskable interrupt ($\overline{NMI}$) cannot be disabled by the programmer, and it will be accepted whenever a peripheral device requests it. This interrupt is generally reserved for very important functions that must be serviced whenever they occur, such as with an impending power failure. The maskable interrupt ($\overline{INT}$) can be selectively enabled or disabled by the programmer. This allows the programmer to disable the interrupt during periods where his program has timing constraints that do not allow it to be interrupted. In the Z80-CPU there is an enable flip flop (called IFF) that is set or reset by the programmer using the Enable Interrupt (EI) and Disable Interrupt (DI) instructions. When the IFF is reset, an interrupt cannot be accepted by the CPU.

III

Actually, for purposes that will be subsequently explained, there are two enable flip flops, called IFF_1 and IFF_2.

IFF_1	IFF_2
Actually disables interrupts from being accepted.	Temporary storage location for IFF_1.

The state of IFF_1 is actually used to inhibit interrupts, while IFF_2 is used as a temporary storage location for IFF_1. The purpose of storing the IFF_1 will be subsequently explained.

A reset to the CPU will force both IFF_1 and IFF_2 to the reset state so that interrupts are disabled. They can then be enabled by an EI instruction at any time by the programmer. When an EI instruction is executed, any pending interrupt request will not be accepted until after the instruction following EI has been executed. This single instruction delay is necessary for cases when the following instruction is a return instruction and interrupts must not be allowed until the return has been completed. The EI instructions set both IFF_1 and IFF_2 to the enable state. When an interrupt is accepted by the CPU, both IFF_1 and IFF_2 are automatically reset, inhibiting further interrupts until the programmer wishes to issue a new EI instruction. Note that for all of the previous cases, IFF_1 and IFF_2 are always equal.

The purpose of IFF_2 is to save the status of IFF_1 when a non-maskable interrupt occurs. When a non-maskable interrupt is accepted, IFF_1 is reset to prevent further interrupts until these are reenabled by the programmer. Thus, after a non-maskable interrupt has been accepted, maskable interrupts are disabled, but the previous state of IFF_1 has been saved so that the complete state of the CPU just prior to the non-maskable interrupt can be restored at any time. When a Load Register A with Register I (LD A, I) instruction or a Load Register A with Register R (LD A, R) instruction is executed, the state of IFF_2 is copied into the parity flag where it can be tested or stored.

A second method of restoring the status of IFF_1 is through the execution of a Return From Non-Maskable Interrupt (RETN) Instruction. Since this instruction indicates that the non-maskable interrupt service routine is complete, the contents of IFF_2 are now copied back into IFF_1, so that the status of IFF_1 just prior to the acceptance of the non-maskable interrupt will be restored automatically.

Figure 8.0-1 is a summary of the effect of different instructions on the two enable flip flops.

INTERRUPT ENABLE/DISABLE FLIP FLOPS
Figure 8.0-1

Action	IFF_1	IFF_2	
CPU Reset	0	0	
DI	0	0	
EI	1	1	
LD A, I	•	•	$IFF_2 \rightarrow$ Parity flag
LD A, R	•	•	$IFF_2 \rightarrow$ Parity flag
Accept NMI	0	•	
RETN	IFF_2	•	$IFF_2 \rightarrow IFF_1$
Accept INT	0	0	
RETI	•	•	

"•" indicates no change

CPU RESPONSE

Non-Maskable

A non-maskable interrupt will be accepted at all times by the CPU. When this occurs, the CPU ignores the next instruction that it fetches and instead does a restart to location 0066H. Thus, it behaves exactly as if it had received a restart instruction, but it is to a location that is not one of the 8 software restart locations. A restart is merely a call to a specific address in page 0 memory.

Maskable

The CPU can be programmed to respond to the maskable interrupt in any one of three possible modes.

Mode 0

This mode is identical to the 8080A interrupt response mode. With this mode, the interrupting device can place any instruction on the data bus and the CPU will execute it. Thus, the interrupting device provides the next instruction to be executed instead of the memory. Often, this instruction will be a restart instruction, since the interrupting device only need supply a single byte instruction. Alternatively, any other instruction, such as a 3 byte call to any location in memory, could be executed by issuing a restart to the 3 byte op code.

The number of clock cycles necessary to execute this instruction is 2 more than the normal number for the instruction. This execution occurs since the CPU automatically adds 2 wait states to an interrupt response cycle to allow sufficient time to implement an external daisy chain for priority control. Section 4.0 illustrates the detailed timing for an interrupt response. After the application of RESET, the CPU will automatically enter interrupt Mode 0.

Mode 1

When this mode has been selected by the programmer, the CPU will respond to an interrupt by executing a restart to location 0038H. Thus the response is identical to that for a non-maskable interrupt except that the call location is 0036H instead of 0066H. Another difference is that the number of cycles required to complete the restart instruction is 2 more than normal due to the two added wait states.

Mode 2

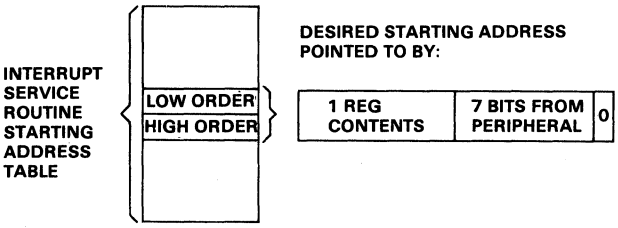
This mode is the most powerful interrupt response mode. With a single 8-bit byte from the user, an indirect call can be made to any memory location.

With this mode, the programmer maintains a table of 16 bit starting addresses for every interrupt service routine. This table may be located anywhere in memory. When an interrupt is accepted, a 16 bit pointer

must be formed to obtain the desired interrupt service routine starting address from the table. The upper 8 bits of this pointer are formed from the contents of the I register. The I register must have been previously loaded with the desired value by the programmer: i.e. LD I, A. Note that the CPU reset clears the I register so that it is initialized to zero. The lower eight bits of the pointer must be supplied by the interrupting device. Actually, only 7 bits are required from the interrupting device, as the least bit must be a zero. This is required since the pointer is used to get two adjacent bytes to form a complete 16 bit service routine starting address, and the addresses must always start in even locations.

INTERRUPT SERVICE ROUTINE STARTING ADDRESS TABLE

Figure 8.0-2



The first byte in the table is the least significant (low order) portion of the address. The programmer must obviously fill this table in with the desired addresses before any interrupts are to be accepted.

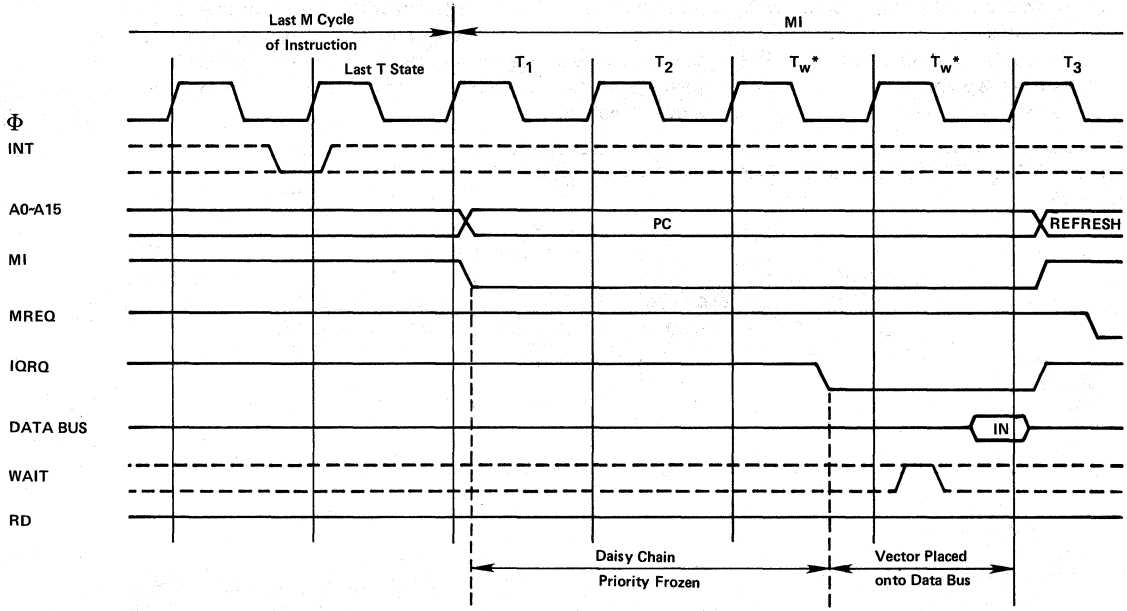
Note that this table can be changed at any time by the programmer (if it is stored in Read/Write Memory) to allow different peripherals to be serviced by different service routines.

Once the interrupting device supplies the lower portion of the pointer, the CPU automatically pushes the program counter onto the stack, obtains the starting address from the table and does a jump to this address. This mode of response requires 19 clock periods to complete (7 to fetch the lower 8 bits from the interrupting device, 6 to save the program counter, and 6 to obtain the jump address.)

Note that the Z80 peripheral devices all include a daisy chain priority interrupt structure that automatically supplies the programmed vector to the CPU during interrupt acknowledge. Refer to the Z80-PIO, Z80-SIO and Z80-CTC manuals for details.

INTERRUPT REQUEST ACKNOWLEDGE CYCLE

Figure 8.0-3



Z80 INTERRUPT ACKNOWLEDGE SUMMARY

1) PERIPHERAL DEVICE REQUESTS INTERRUPT. Any device requesting an interrupt can pull the wired-or line $\overline{INT}$ low.

2) CPU ACKNOWLEDGES INTERRUPT. Priority status is frozen when $\overline{M1}$ goes low during the Interrupt Acknowledge sequence. Propagation delays down the IEI/IEO daisy chain must be settled out when $\overline{IORQ}$ goes low. If IEI is HIGH, an active Peripheral Device will place its Interrupt Vector on the Data Bus when $\overline{IORQ}$ goes low. That Peripheral then releases its hold on $\overline{INT}$ allowing interrupts from a higher priority device. Lower priority devices are inhibited from placing their Vector on the Data Bus or Interrupting because IEO is low on the active device.

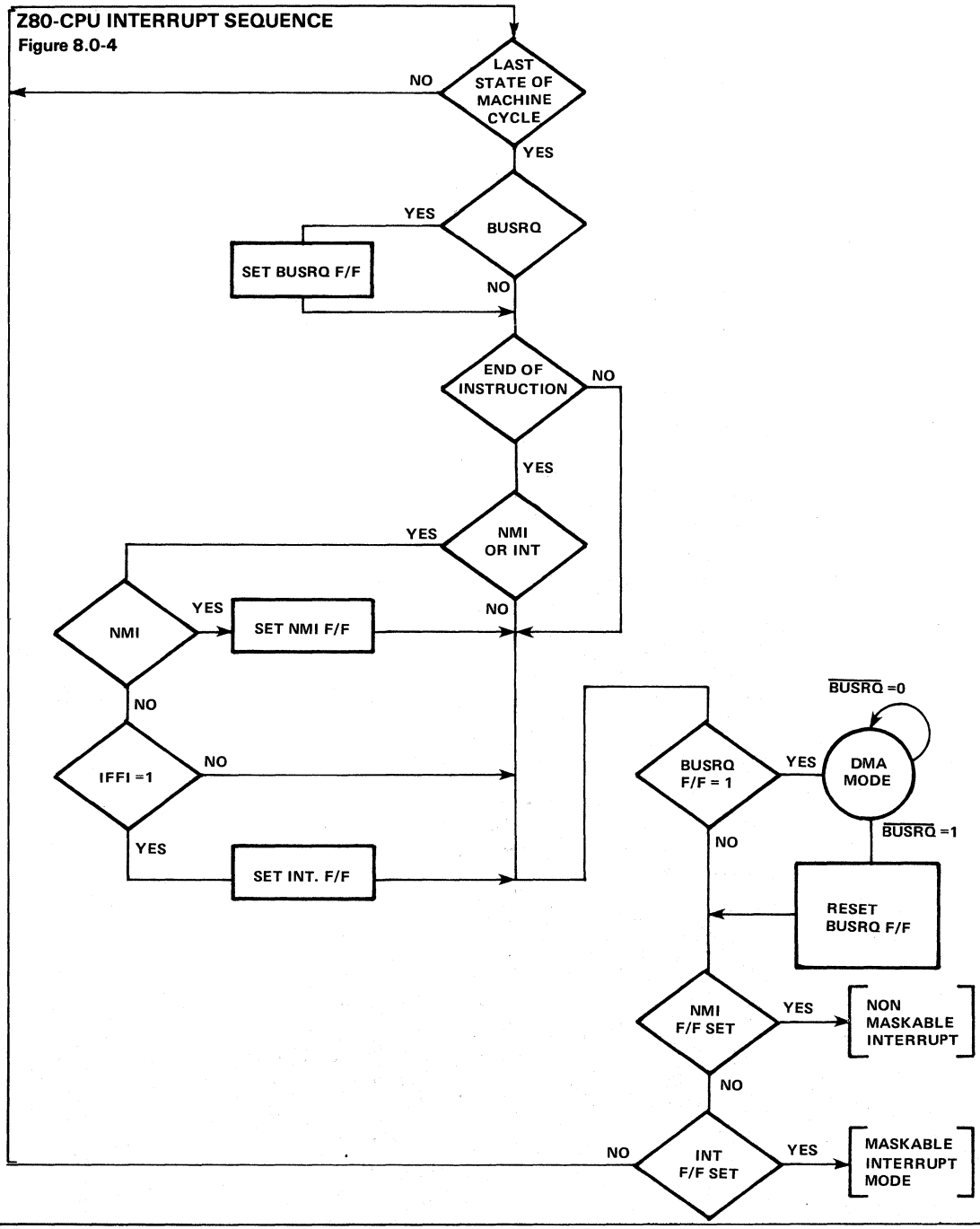
3) INTERRUPT IS CLEARED. An active Peripheral device (IEI=1, IEO=0) monitors OP Code fetches for an RETI (ED 4D) instruction which tells the peripheral that its Interrupt Service Routine is over. The peripheral device then re-activates its internal Interrupt structure as well as raising its IEO line to enable lower priority devices.

INTERRELATIONSHIP OF $\overline{INT}$, $\overline{NMI}$, AND $\overline{BUSRQ}$

The following flow chart details the relationship of three control inputs to the Z80-CPU. Note the following from the flow chart.

1. $\overline{INT}$ and $\overline{NMI}$ are always acted on at the end of an instruction.
2. $\overline{BUSRQ}$ is acted on at the end of a machine cycle.
3. While the CPU is in the DMA MODE, it will not respond to active inputs on $\overline{INT}$ or $\overline{NMI}$.
4. These three inputs are acted on in the following order of priority: a) $\overline{BUSRQ}$ b) $\overline{NMI}$ C) $\overline{INT}$

Z80-CPU INTERRUPT SEQUENCE
Figure 8.0-4



9.0 HARDWARE IMPLEMENTATION EXAMPLES

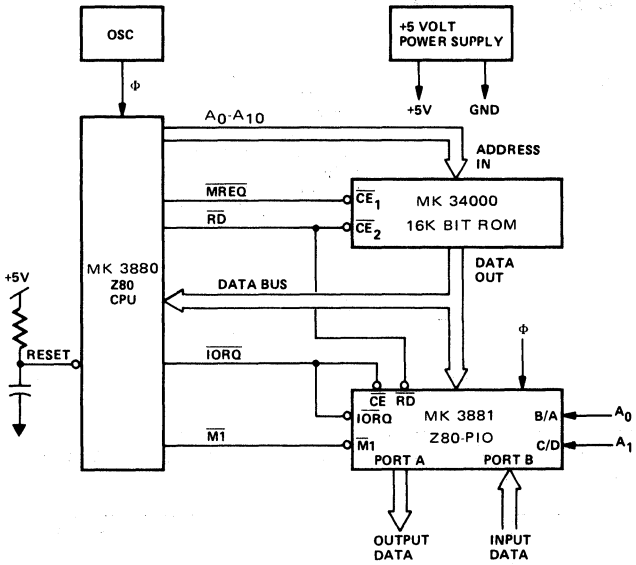
This chapter is intended to serve as a basic introduction to implementing systems with the Z80-CPU.

MINIMUM SYSTEM

Figure 9.0-1 is a diagram of a very simple Z80 system. Any Z80 system must include the following five elements:

1) Five volt power supply
2) Oscillator
3) Memory devices
4) I/O circuits
5) CPU

MINIMUM Z80 COMPUTER SYSTEM
Figure 9.0-1



Since the Z80-CPU only requires a single 5 volt supply, most small systems can be implemented using only this single supply.

The oscillator can be very simple since the only requirement is that it be a 5 volt square wave. For systems not running at full speed, a simple RC oscillator can be used. When the CPU is operated near the highest possible frequency, a crystal oscillator is generally required because the system timing will not tolerate the drift or jitter that an RC network will generate. A crystal oscillator can be made from inverters and a few discrete components or monolithic circuits are widely available.

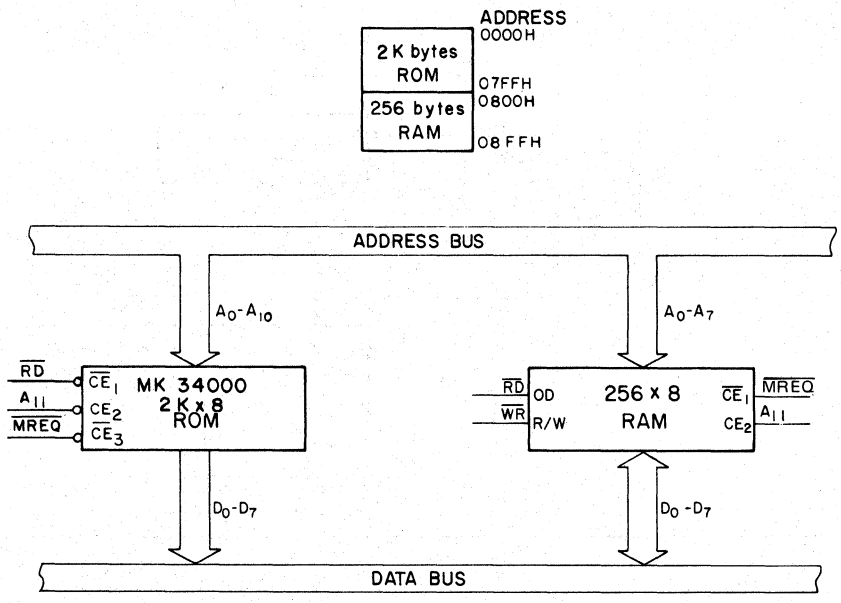
The external memory can be any mixture of standard RAM, ROM, or PROM. In this simple example we have shown a single 16K bit ROM (2K bytes) being utilized as the entire memory system. For this example we have assumed that the Z80 internal register configuration contains sufficient Read/Write storage so that external RAM memory is not required.

Every computer system requires I/O circuits to allow it to interface to the "real world." In this simple example, it is assumed that the output is an 8 bit control vector and the input is an 8 bit status word. The input data could be gated onto the data bus using any standard tri-state driver while the output data could be latched with any type of standard TTL latch. For this example we have used a Z80-PIO for the I/O circuit. This single circuit attaches to the data bus as shown and provides the required 16 bits of TTL compatible I/O. (Refer to the Z80-PIO manual for details on the operation of this circuit.) Notice in this example that with only three LSI circuits, a simple oscillator and a single 5 volt power supply, a powerful computer has been implemented.

ADDING RAM

Most computer systems require some amount of external Read/Write memory for data storage and to implement a "stack". Figure 9.0-2 illustrates how 256 bytes of static memory can be added to the previous example. In this example, the memory space is assumed to be organized as follows:

ROM & RAM IMPLEMENTATION EXAMPLE
Figure 9.0-2

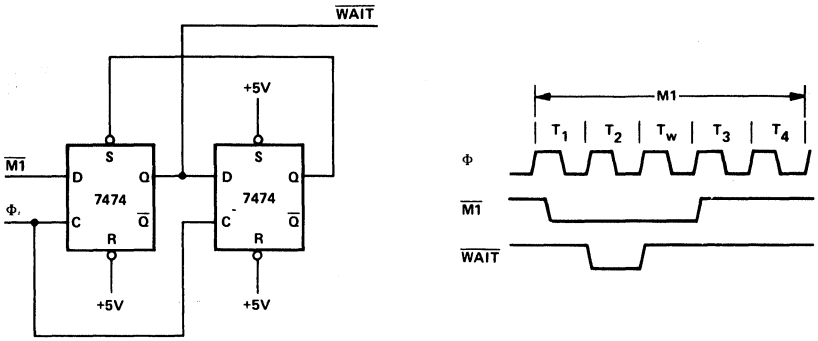


In this diagram the address space is described in hexadecimal notation. For this example, address bit A_{11} separates the ROM space from the RAM space so that it can be used for the chip select function. For larger amounts of external ROM or RAM, a simple TTL decoder will be required to form the chip selects.

MEMORY SPEED CONTROL

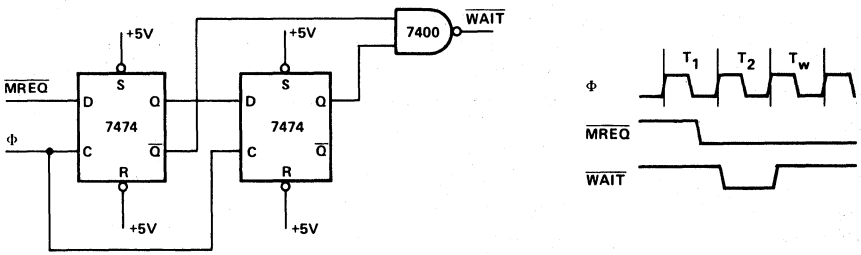
For many applications, it may be desirable to use slow memories to reduce costs. The $\overline{WAIT}$ line on the CPU allows the Z80 to operate with any speed memory. By referring back to section 4 you will notice that the memory access time requirements are most severe during the M1 cycle instruction fetch. All other memory accesses have an additional one half of a clock cycle to be completed. For this reason it may be desirable in some applications to add one wait state to the M1 cycle so that slower memories can be used. Figure 9.0-3 is an example of a simple circuit that will accomplish this task. This circuit can be changed to add a single wait state to any memory access as shown in Figure 9.0-4.

ADDING ONE WAIT STATE TO AN M1 CYCLE
Figure 9.0-3



III

ADDING ONE WAIT STATE TO ANY MEMORY CYCLE
Figure 9.0-4



INTERFACING DYNAMIC MEMORIES

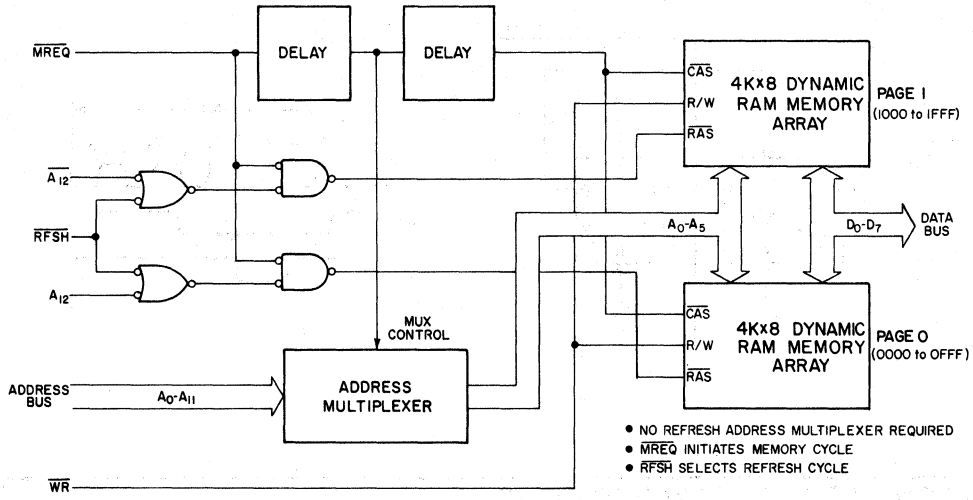
This section is intended only to serve as a brief introduction to interfacing dynamic memories. Each individual dynamic RAM has varying specifications that will require minor modifications to the description given here, and no attempt will be made in this document to give details for any particular RAM.

Figure 9.0-5 illustrates the logic memory to interface 8K bytes of dynamic RAM using 16-pin 4K dynamic memories. This Figure assumes that the RAM's are the only memory in the system so that A_{12} is used to select between the two pages of memory. During refresh time, all memories in the system must be read. The CPU provides the proper refresh address on lines A_0 through A_6. To add additional memory to the system, it is necessary only to replace the two gates that operate on A_{12} with a decoder that operates on all required address bits. For larger systems, buffering for the address and data bus is also generally required.

An application note entitled "Z80 Interfacing Techniques for Dynamic RAM" is available from your Mostek representative which describes dynamic RAM design techniques.

INTERFACING DYNAMIC RAMs

Figure 9.0-5



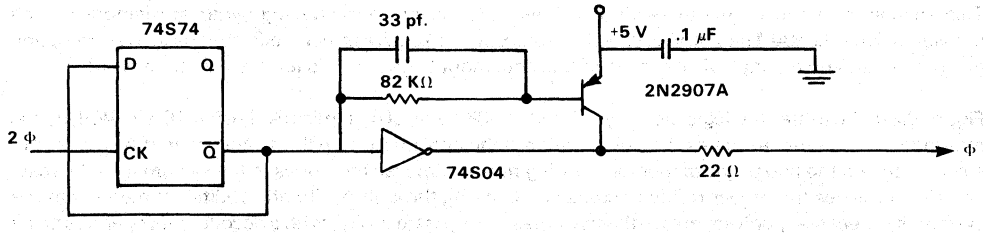
Z80-CPU DESIGN CONSIDERATIONS: CLOCK CIRCUITRY

Proper Z80 clock circuitry design is of paramount importance when designing a Z80 system. Parameters such as clock rise and fall times, min./max. clock high and low times, and max clock over and under shoot should be closely adhered to. Violation of these specs will result in unreliable and unpredictable CPU/peripheral behavior. Several manufacturers offer a wide variety of combination oscillator/drivers housed in 14 pin DIP packages. The following is a suggested source of reliable oscillators/drivers currently available.

Vendor	Function	Part No.
Motorola	Oscillator/Driver	K1160 series
Motorola	Oscillator	K1114
MF Electronics	Oscillator	MF1114
Hybrid House	Driver	HH3006A

Figure 9.0-6 illustrates a schematic recommended for driving the Z80 CPU, as well as other Z80 peripherals. This configuration meets the 30 ns rise and fall time while driving up to a 150 pf. load. Note the divide by two input flip flop to provide a 50 percent duty cycle clock. This stage may be omitted if the oscillator is guaranteed to be within the specifications.

Figure 9.0-6



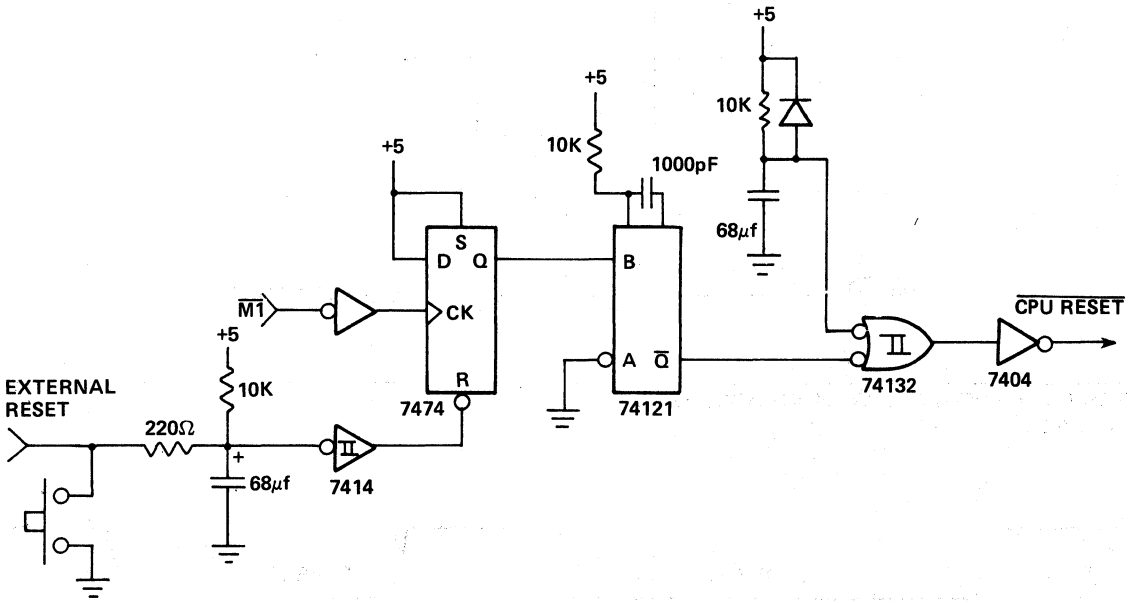
RESET CIRCUITRY

The Z80-CPU has the characteristic that, if the $\overline{\text{RESET}}$ input goes low during T2 or T4 of a cycle, then the MREQ signal will go to an indeterminate state for one T-State approximately 3 T-States later. If there are dynamic memories in the system, this action could cause an aborted or short access of the dynamic RAM,

which could cause destruction of data within the RAM. If the contents of RAM are of no concern after RESET, then this characteristic is no problem, as the CPU always resets properly. If RAM contents must be preserved, then the falling edge of the $\overline{RESET}$ input must be synchronized by the falling edge of $\overline{M1}$.

The circuitry of Figure 9.0-7 does this synchronization as well as providing a one-shot to limit the duration of the CPU $\overline{RESET}$ pulse. The CPU $\overline{RESET}$ signal must be a pulse, even though the EXTERNAL RESET button is held closed in order to avoid suspending the CPU refresh of dynamic RAM for a time long enough to destroy data in the RAM.

MANUAL AND POWER-ON RESET CIRCUIT
Figure 9.0-7

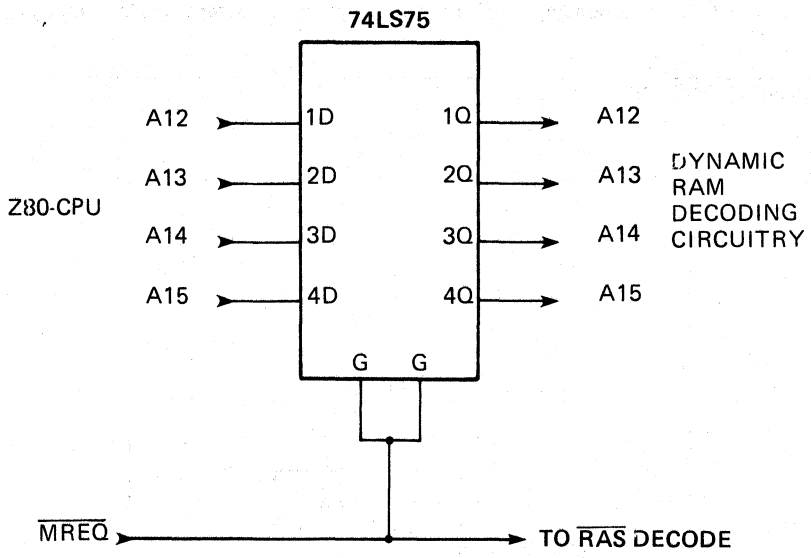


ADDRESS LATCHING

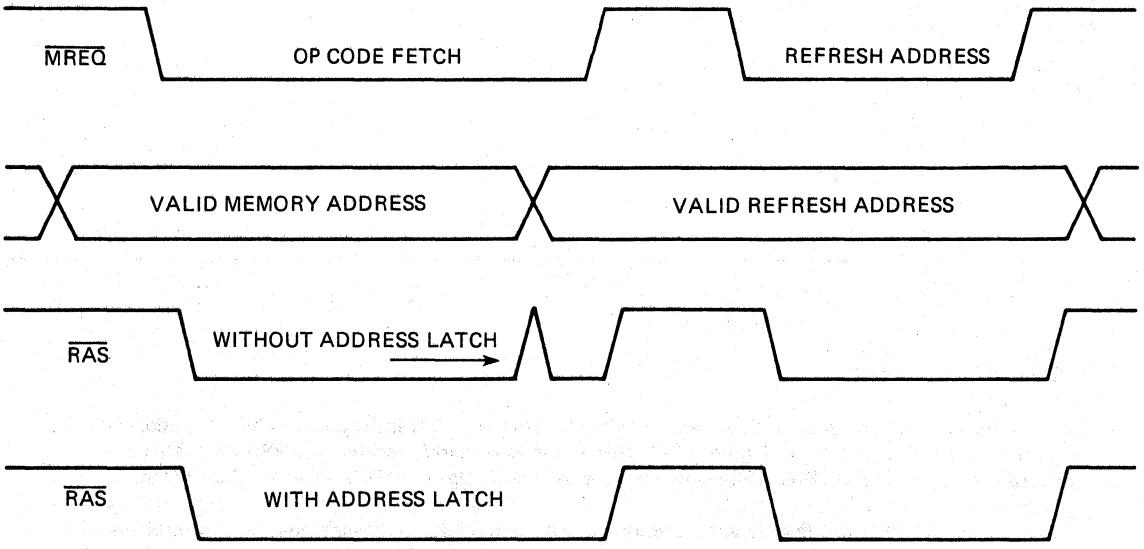
In order to guarantee proper operation of the Z80-CPU with dynamic RAMs the upper 4 bits of the address should be latched as shown in Figure 9.0-8. This action is required because the Z80-CPU does not guarantee that the Address Bus will hold valid before the rising edge of MREQ on an OP Code Fetch.

This action does not directly affect dynamic memories because they latch addresses internally. The problem comes from the address decoder which generates $\overline{RAS}$. If the address lines which drive the decoder are allowed to change while $\overline{MREQ}$ is low, then a "glitch" can occur on the $\overline{RAS}$ line or lines, which may have the effect of destroying one row of data within the dynamic RAM.

ADDRESS LATCH
Figure 9.0-8



RAS TIMING WITH AND WITHOUT ADDRESS LATCH
Figure 9.0-9

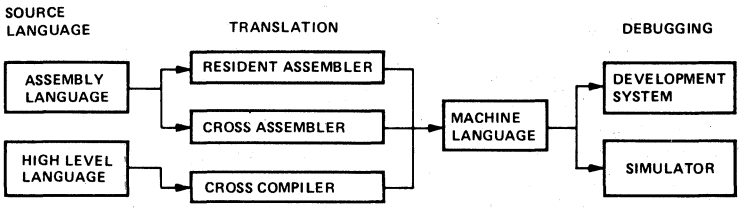


10.0 SOFTWARE IMPLEMENTATION EXAMPLES

10.1 METHODS OF SOFTWARE IMPLEMENTATION

Several different approaches are possible in developing software for the Z80 (Figure 10.1). First of all, Assembly Language or a high level language may be used as the source language. These languages may then be translated into machine language on a commercial time sharing facility using a cross-assembler or cross-compiler, or, in the case of assembly language, the translation can be accomplished on a Z80 Development System using a resident assembler. Finally, the resulting machine code can be debugged either on a time-sharing facility using a Z80 simulator or on a Z80 Development System which uses a Z80-CPU directly.

SOFTWARE GENERATION TECHNIQUES
Figure 10.1



In selecting a source language, the primary factors to be considered are clarity and ease of programming versus code efficiency. A high level language with its machine independent constraints is typically better for formulating and maintaining algorithms, but the resulting machine code is usually somewhat less efficient than what can be written directly in assembly language. These tradeoffs can often be balanced by combining high level language routines, by identifying those portions of a task which must be optimized, and by writing them as assembly language subroutines.

Deciding whether to use a resident or cross assembler is a matter of availability and short-term versus long-term expense. While the initial expenditure for a development system is higher than that for a time-sharing terminal, the cost of an individual assembly using a resident assembler is negligible while the same operation on a time-sharing system is relatively expensive, and in a short time this cost can equal the total cost of a development system.

Debugging on a development system versus a simulator is also a matter of availability and expense combined with operational fidelity and flexibility. As with the assembly process, debugging is less expensive on a development system than on a simulator available through time-sharing. In addition, the fidelity of the operating environment is preserved through real-time execution on a Z80-CPU and by connecting the I/O and memory components which will actually be used in the production system. The only advantage to the use of a simulator is the range of criteria which may be selected for such debugging procedures as tracing and setting breakpoints. This flexibility exists because a software simulation can achieve any degree of complexity in its interpretation of machine instructions while development system procedures have hardware limitations such as the capacity of the real-time storage module, the number of breakpoint registers and the pin configuration of the CPU. Despite such hardware limitations, debugging on a development system is typically more productive than on a simulator because of the direct interaction that is possible between the programmer and the authentic execution of his program.

10.2 SOFTWARE FEATURES OFFERED BY THE Z80-CPU

The Z80 instruction set provides the user with a large and flexible repertoire of operations with which to formulate control of the Z80-CPU.

The primary, auxiliary, and index registers can be used to hold the arguments of arithmetic and logical operations, or to form memory addresses, or as fast-access storage for frequently used data.

Information can be moved directly from register to register; from memory to memory; from memory to registers, or from registers to memory. In addition, register contents and register/memory contents can be exchanged without using temporary storage. In particular, the contents of primary and auxiliary registers can be completely exchanged by executing only two instructions: EX and EXX. This register exchange procedure can be used to separate the set of working registers between different logical procedures or to expand the set of available registers in a single procedure.

Storage and retrieval of data between pairs of registers and memory can be controlled on a last-in first-out basis through PUSH and POP instructions which utilize a special stack pointer register, SP. This stack register is available both to manipulate data and to store and retrieve addresses for subroutine linkage automatically. When a subroutine is called, for example, the address following the CALL instruction is placed on the top of the pushdown stack pointed to by SP. When a subroutine returns to the calling routine, the address on the top of the stack is used to set the program counter for the address of the next instruction. The stack pointer is adjusted automatically to reflect the current "top" stack position during PUSH, POP, CALL and RET instructions. This stack mechanism allows pushdown data stacks and subroutine calls to be nested to any practical depth because the stack area can potentially be as large as memory space.

The sequence of instruction execution can be controlled by six different flags (carry, zero, sign, parity/overflow, add-subtract, half-carry) which reflect the results of arithmetic, logical, shift and compare instructions. After the execution of an instruction which sets a flag, that flag can be used to control a conditional jump or return instruction. These instructions provide logical control following the manipulation of single bit, eight-bit byte (or) sixteen-bit data quantities.

A full set of logical operations, including AND, OR, XOR (exclusive —OR), CPL (NOR) and NEG (two's complement) are available for Boolean operations between the accumulator and 1) all other eight-bit registers, 2) memory locations, or 3) immediate operands.

In addition, a full set of arithmetic and logical shifts in both directions is available and operate on the contents of all eight-bit primary registers or directly on any memory location. The carry flag can be included or simply set by these shift instructions to provide both the testing of shift results and to link register/register or register/memory shift operations.

10.3 EXAMPLES OF USE OF SPECIAL Z80 INSTRUCTIONS

A. Let us assume that a string of data in memory starting at location "DATA" is to be moved into another area of memory starting at location "BUFFER" and that the string length is 737 bytes. This operation can be accomplished as follows:

```
LD      HL, DATA      ;START ADDRESS OF DATA STRING
LD      DE, BUFFER    ;START ADDRESS OF TARGET BUFFER
LD      BC, 737       ;LENGTH OF DATA STRING
LDIR                  ;MOVE STRING — TRANSFER MEMORY
                      ;POINTED TO BY HL INTO MEMORY
                      ;LOCATION POINTED TO BY DE INCREMENT
                      ;HL AND DE, DECREMENT BC PROCESS
                      ;UNTIL BC=0.
```

11 bytes are required for this operation and each byte of data is moved in 21 clock cycles.

B. Assume that a string in memory starting at location "DATA" is to be moved into another area of memory starting at location "BUFFER" until an ASCII $ character (used as string delimiter) is found. Also assume that the maximum string length is 132 characters. The operation can be performed as follows:

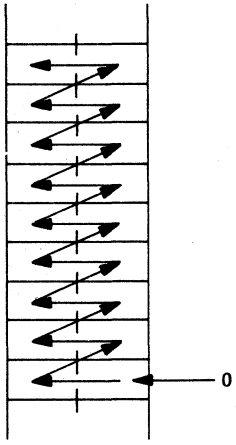
```
        LD   HL, DATA        ;STARTING ADDRESS OF DATA STRING
        LD   DE, BUFFER      ;STARTING ADDRESS OF TARGET BUFFER
        LD   BC, 132         ;MAXIMUM STRING LENGTH
        LD   A, '$'          ;STRING DELIMITER CODE
LOOP: CP   (HL)              ;COMPARE MEMORY CONTENTS WITH DELIMITER
        JR   Z, END—$        ;GOT TO END IF CHARACTERS EQUAL
        LDI                  ;MOVE CHARACTER (HL) TO (DE)
                             ;INCREMENT HL AND DE, DECREMENT BC
        JP   PE, LOOP        ;GO TO "LOOP' IF MORE CHARACTERS
END:                         ;OTHERWISE, FALL THROUGH
                             :NOTE: P/V FLAG IS USED
                             ;TO INDICATE THAT REGISTER BC WAS
                             ;DECREMENTED TO ZERO.
```

19 bytes are required for this operation.

C. Let us assume that a 16-digit decimal number represented in packed BCD format (two BCD digits;/byte) has to be shifted as shown in the Figure 10.2 in order to mechanize BCD multiplication or division. The operation can be accomplished as follows:

```
        LD    HL, DATA       ;ADDRESS OF FIRST BYTE
        LD    B, COUNT       ;SHIFT COUNT
        XOR   A              ;CLEAR ACCUMULATOR
ROTAT: RLD                   ;ROTATE LEFT LOW ORDER DIGIT IN ACC
                             ;WITH DIGITS IN (HL)
        INC   HL             ;ADVANCE MEMORY POINTER
        DJNZ  ROTAT—$        ;DECREMENT B AND GO TO ROTAT IF
                             ;B IS NOT ZERO, OTHERWISE FALL THROUGH
```

BCD DATA SHIFTING
11 bytes are required for this operation.
Figure 10.2



11 bytes are required for this operation.

D. Assume that one number is to be subtracted from another and (a) that they are both in packed BCD format; b) that they are of equal but varying length, and c) that the result is to be stored in a location of the minuend. The operation can be accomplished as follows:

```
        LD      HL, ARG1        ;ADDRESS OF MINUEND
        LD      DE, ARG2        ;ADDRESS OF SUBTRAHEND
        LD      B, LENGTH       ;LENGTH OF TWO ARGUMENTS
        AND     A               ;CLEAR CARRY FLAG
SUBDEC: LD      A, (DE)         ;SUBTRAHEND TO ACC
        SBC     A, (HL)         ;SUBTRACT (HL) FROM ACC
        DAA                     ;ADJUST RESULT TO DECIMAL CODED VALUE
        LD      (HL), A         ;STORE RESULT
        INC     HL              ;ADVANCE MEMORY POINTERS
        INC     DE
        DJNZ    SUBDEC—$        ;DECREMENT B AND GO TO "SUBDEC" IF B
                                ;NOT ZERO, OTHERWISE FALL THROUGH
```

17 bytes are required for this operation.

10.4 EXAMPLES OF PROGRAMMING TASKS

A. The following program sorts an array of numbers each in the range <0,255> into ascending order using a standard exchange sorting algorithm.

```
01/22/76      11:14:37                    BUBBLE LISTING
LOC     OBJ CODE  STMT  SOURCE STATEMENT

                  1     ;  *** STANDARD EXCHANGE (BUBBLE) SORT ROUTINE***
                  2     ;
                  3     ;  AT ENTRY: HL CONTAINS ADDRESS OF DATA
                  4     ;            C CONTAINS NUMBER OF ELEMENTS TO BE SORTED
                  5     ;            (1<C<256)
                  6     ;
                  7     ;  AT EXIT: DATA SORTED IN ASCENDING ORDER
                  8     ;
                  9     ;  USE OF REGISTERS
                  10    ;
                  11    ;  REGISTER    CONTENTS
                  12    ;
                  13    ;  A           TEMPORARY STORAGE FOR CALCULATIONS
                  14    ;  B           COUNTER FOR DATA ARRAY
                  15    ;  C           LENGTH OF DATA ARRAY
                  16    ;  D           FIRST ELEMENT IN COMPARISON
                  17    ;  E           SECOND ELEMENT IN COMPARISON
                  18    ;  H           FLAG TO INDICATE EXCHANGE
                  19    ;  L           UNUSED
                  20    ;  IX          POINTER INTO DATA ARRAY
                  21    ;  IY          UNUSED
                  22    ;
```

```
LOC    OBJ CODE  STMT  SOURCE STATMENT
0000   222600    23    SORT:   LD    (DATA), HL     ;SAVE DATA ADDRESS
0003   CB84      24    LOOP:   RES   FLAG, H        ;INITIALIZE EXCHANGE FLAG
0005   41        25            LD    B,C            ;INITIALIZE LENGTH COUNTER
0006   05        26            DEC   B              ;ADJUST FOR TESTING
0007   DD2A2600  27            LD    IX, (DATA)     ;INITIALIZE ARRAY POINTER
000B   DD7E00    28    NEXT:   LD    A,(IX+0)       ;FIRST ELEMENT IN COMPARISON
000E   57        29            LD    D, A           ;TEMPORARY STORAGE FOR ELEMENT
000F   DD5E01    30            LD    E, (IX+1)      ;SECOND ELEMENT IN COMPARISON
0012   93        31            SUB   E              ;COMPARISON FIRST TO SECOND
0013   3808      32            JR    C, NOEX-$      ;IF FIRST> SECOND, NO JUMP
0015   DD7300    33            LD    (IX), E        ;EXCHANGE ARRAY ELEMENTS
0018   DD7201    34            LD    (IX+1), D
001B   CBC4      35            SET   FLAG H         ;RECORD EXCHANGE OCCURRED
001D   DD23      36    NOEX:   INC   IX             ;POINT TO NEXT DATA ELEMENT
001F   10EA      37            DJNZ  NEXT-$         ;COUNT NUMBER OF COMPARISONS
                                                     ;REPEAT IF MORE DATA PAIRS
0021   CB44      39            BIT   FLAG, H        ;DETERMINE IF EXCHANGE OCCURRED
0023   20DE      40            JR    NZ, LOOP-$     ;CONTINUE IF DATA UNSORTED
0025   C9        41            RET                  ;OTHERWISE, EXIT
               42    ;
0026           43    FLAG:   EQU   0              ;DESIGNATION OF FLAG BIT
0026           44    DATA:   DEFS  2              ;STORAGE FOR DATA ADDRESS
               45            END
```

B. The following program multiplies two unsigned 16-bit integers and leaves the result in the HL register pair.

```
01/22/76      11:32:36          MULTIPLY LISTING
LOC    OBJ CODE    STMT   SOURCE STATEMENT

0000               1    MULT:;  UNSIGNED SIXTEEN BIT INTEGER MULTIPLY.
                   2    ;       ON ENTRANCE: MULTIPLIER IN HL.
                   3    ;                    MULTIPLICAND IN DE.
                   4    ;
                   5    ;       ON EXIT: RESULT IN HL.
                   6    ;
                   7    ;       REGISTERS USES:
                   8    ;
                   9    ;
                  10    ;       H    HIGH ORDER PARTIAL RESULT
                  11    ;       L    LOW ORDER PARTIAL RESULT
                  12    ;       D    HIGH ORDER MULTIPLICAND
                  13    ;       E    LOW ORDER MULTIPLICAND
                  14    ;       B    COUNTER FOR NUMBER OF SHIFTS
                  15    ;       C    HIGH ORDER BITS OF MULTIPLIER
                  16    ;       A    LOW ORDER BITS OF MULTIPLIER
                  17    ;
0000   0610       18            LD    B, 16;         NUMBER OF BITS—INITIALIZE
0002   4A         19            LD    C,D;           MOVE MULTIPLIER
0003   7B         20            LD    A,E;
0004   EB         21            EX    DE,HL;         MOVE MULTIPLICAND
0005   210000     22            LD    HL,0;          CLEAR PARTIAL RESULT
0008   CB39       23    MLOOP:  SRL   C;             SHIFT MULTIPLIER RIGHT
000A   1F         24            RR    A;             LEAST SIGNIFICANT BIT IS
                                                     IN CARRY.
000B   3001       26            JR    NC, NOADD-$    IF NO CARRY, SKIP THE ADD.
```

```
LOC    OBJ CODE   STMT  SOURCE STATMENT
000D   19         27          ADD HL, DE;        ELSE ADD MULTIPLICAND TO
                                                 PARTIAL RESULT.
000E   EB         29   NOADD: EX  DE,HL;         SHIFT MULTIPLICAND LEFT.
000F   29         30          ADD HL,HL;         BY MULTIPLYING IT BY TWO.
0010   EB         31          EX  DE,HL;
0011   10F5       32         DJNZ MLOOP-$;       REPEAT UNTIL NO MORE BITS.
0013   C9         33          RET;
                  34          END;
```

11.0 ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... Specified Operating Range
Storage Temperature ..... -65°C to +150°C
Voltage on Any Pin with Respect to Ground ..... -0.3 V to +7V
Power Dissipation ..... 1.5 W

All ac parameters assume a load capacitance of 50 pF max.

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = 5 V ± 5% unless otherwise specified

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS	TEST CONDITIONS
V_{ILC}	Clock Input Low Voltage	-0.3		0.8	V	
V_{IHC}	Clock Input High Voltage	V_{CC}-.6		V_{CC}+.3	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 1.8 mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -250 μA
I_{CC}	Power Supply Current			150*	mA	
I_{LI}	Input Leakage Current			±10	μA	V_{IN} = 0 to V_{CC}
I_{LO}	Tri-State Output Leakage Current in Float			±10	μA	V_{OUT} = 0.4 V to V_{CC}

*200 mA for -4, -10 or -20 devices

NOTE: All outputs are rated at one standard TTL load.

CAPACITANCE

T_A = 25°C, f = 1 MHz unmeasured pins returned to ground

SYMBOL	PARAMETER	MAX	UNIT
CΦ	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

*Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

AC CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5 V ± 5%, Unless Otherwise Noted

SIGNAL	SYMBOL	PARAMETER	3880 MIN (ns)	3880 MAX (ns)	3880-4 MIN (ns)	3880-4 MAX (ns)	3880-6 MIN (ns)	3880-6 MAX (ns)
Φ	t_c	Clock Period	400	[12]	250	[12]	165	[12]
	$t_w(\Phi H)$	Clock Pulse Width, Clock High	180	(D)	110	(D)	65	(D)
	$t_w(\Phi L)$	Clock Pulse Width, Clock Low	180	2000	110	2000	65	2000
	$t_{r,f}$	Clock Rise and Fall Time		30		30		20
A_{0-15}	$t_{D(AD)}$	Address Output Delay		145		110		90
	$t_{F(AD)}$	Delay to Float		110		90		80
	t_{acm}	Address Stable Prior to $\overline{MREQ}$ (Memory Cycle)	[1]		[13]		[24]	
	t_{aci}	Address Stable Prior to $\overline{IORQ}$, $\overline{RD}$ or $\overline{WR}$ (I/O Cycle)	[2]		[14]		[25]	
	t_{ca}	Address Stable From $\overline{RD}$, $\overline{WR}$, $\overline{IORQ}$ or $\overline{MREQ}$	[3]		[15]		[26]	
	t_{caf}	Address Stable From $\overline{RD}$ or $\overline{WR}$ During Float	[4]		[16]		[27]	
D_{0-7}	$t_{D(D)}$	Data Output Delay		230		150		130
	$t_{F(D)}$	Delay to Float During Write Cycle		90		90		80
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		35		30	
	$t_{S\overline{\Phi}(D)}$	Data Setup Time to Falling Edge at Clock During M2 to M5	60		50		40	
	t_{dcm}	Data Stable Prior to $\overline{WR}$ (memory cycle)	[5]		[17]		[28]	
	t_{dci}	Data Stable Prior to $\overline{WR}$ (I/O Cycle)	[6]		[18]		[29]	
	t_{cdf}	Data Stable from $\overline{WR}$	[7]		[19]		[30]	
	t_H	Input Hold Time	0		0		0	
$\overline{MREQ}$	$t_{DL\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ Low	20	100	20	85	20	70
	$t_{DH\Phi(MR)}$	$\overline{MREQ}$ Delay From Rising Edge of Clock, $\overline{MREQ}$ High		100		85		70
	$t_{DH\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ High		100		85		70
	$t_{w(\overline{MRL})}$	Pulse Width, $\overline{MREQ}$ Low	[8]		[20]		[20]	
	$t_{w(\overline{MRH})}$	Pulse Width, $\overline{MREQ}$ High	[9]		[21]		[21]	
$\overline{IORQ}$	$t_{DL\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ Low		90		75		65
	$t_{DL\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ Low		110		85		70
	$t_{DH\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ High		100		85		70
	$t_{DH\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock Clock, $\overline{IORQ}$ High		110		85		70
$\overline{RD}$	$t_{DL\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ Low		100		85		70
	$t_{DL\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ Low		130		95		80
	$t_{DH\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ High	15	100	15	85	15	70
	$t_{DH\overline{\Phi}(BD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ High		110		85		70
$\overline{WR}$	$t_{DL\Phi(WR)}$	$\overline{WR}$ Delay From Rising Edge of Clock, $\overline{WR}$ Low		80		65		60
	$t_{DL\overline{\Phi}(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ Low		90		80		70
	$t_{DH\overline{\Phi}(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ High		100		80		70
	$t_{w(\overline{WRL})}$	Pulse Width, $\overline{WR}$ Low	[10]		[22]		[22]	

NOTES:

A. Data should be enabled onto the CPU data bus when RD is active. During interrupt acknowledge data should be enabled when M1 and IORQ are both active.

B. The RESET signal must be active for a minimum of 3 clock cycles.

[Cont'd on next page]

SIGNAL	SYMBOL	PARAMETER	3880 MIN (ns)	3880 MAX (ns)	3880-4 MIN (ns)	3880-4 MAX (ns)	3880-6 MIN (ns)	3880-6 MAX (ns)
$\overline{M1}$	$t_{DL(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock $\overline{M1}$ Low		130		100		80
	$t_{DH(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock $\overline{M1}$ High		130		100		80
$\overline{RFSH}$	$t_{DL(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ Low		180		130		110
	$t_{DH(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ High		150		120		100
$\overline{WAIT}$	$t_{S(WT)}$	$\overline{WAIT}$ Setup Time to Falling Edge of Clock	70		70		60	
$\overline{HALT}$	$t_{D(HT)}$	$\overline{HALT}$ Delay Time From Falling Edge of Clock		300		300		260
$\overline{INT}$	$t_{s(IT)}$	$\overline{INT}$ Setup Time to Rising Edge of Clock	80		80		70	
$\overline{NMI}$	$t_{w(NMI)}$	Pulse Width, $\overline{NMI}$ Low	80		80		70	
$\overline{BUSRQ}$	$t_{S(BQ)}$	$\overline{BUSRQ}$ Setup Time to Rising Edge of Clock	80		50		50	
$\overline{BUSAK}$	$t_{DL(BA)}$	$\overline{BUSAK}$ Delay From Rising Edge of Clock, $\overline{BUSAK}$ Low		120		100		90
	$t_{DH(BA)}$	$\overline{BUSAK}$ Delay From Falling Edge of Clock, $\overline{BUSAK}$ High		110		100		90
$\overline{RESET}$	$t_{s(RS)}$	$\overline{RESET}$ Setup Time to Rising Edge of Clock	90		60		60	
	$t_{F(C)}$	Delay to/from Float ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ and $\overline{WR}$)		100		80		70
	t_{mr}	$\overline{M1}$ Stable Prior to $\overline{IORQ}$ (Interrupt Ack.)	[11]		[23]		[31]	

[1] $t_{ACM} = t_w(\Phi H) + t_f - 75$

[2] $t_{aci} = t_c - 80$

[3] $t_{CA} = t_w(\Phi L) + t_r - 40$

[4] $t_{caf} = t_w(\Phi L) + t_r - 60$

[5] $t_{dcm} = t_C - 210$

[6] $t_{dci} = t_w(\Phi L) + t_r - 210$

[7] $t_{cdf} = t_w(\Phi L) + t_r - 80$

[8] $t_w(\overline{MRL}) = t_c - 40$

[9] $t_w(\overline{MRH}) = t_w(\Phi H) + t_f - 30$

[10] $t_w(\overline{WR}) = t_c - 40$

[11] $t_{mr} = 2\,t_c + t_w(\Phi H) + t_f - 80$

[12] $t_c = t_w(\Phi H) + t_w(\Phi L) + t_r + t_f$

[13] $t_{acm} = t_w(\Phi H) + t_f - 65$

[14] $t_{aci} = t_c - 70$

[15] $t_{ca} = t_w(\Phi L) + t_r - 50$

[16] $t_{caf} = t_w(\Phi L) + t_r - 45$

[17] $t_{dcm} = t_c - 170$

[18] $t_{dci} = t_w(\Phi L) + t_r - 170$

[19] $t_{cdf} = t_w(\Phi L) + t_r - 70$

[20] $t_w(\overline{MRL}) = t_c - 30$

[21] $t_w(\overline{MRH}) = t_w(\Phi H) + t_f - 20$

[22] $t_w(\overline{WR}) = t_c - 30$

[23] $t_{mr} = 2t_c + t_w(\Phi H) + t_f - 65$

[24] $t_{ACM} = t_w(\Phi H) + t_f - 50$

[25] $t_{aci} = t_c - 55$

[26] $t_{CA} = t_w(\Phi L) + t_r - 50$

[27] $t_{caf} = t_w(\Phi L) + t_r - 45$

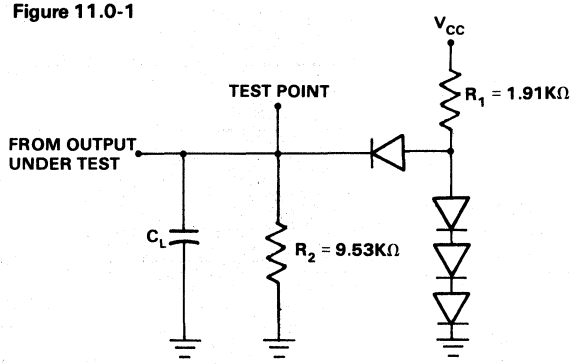
[28] $t_{dcm} = t_c - 140$

[29] $t_{dci} = t_w(\Phi L) + t_r - 140$

[30] $t_{cdf} = t_w(\Phi L) + t_r - 55$

[31] $t_{mr} = 2t_c + t_w(\Phi H) + t_f - 50$

LOAD CIRCUIT FOR OUTPUT
Figure 11.0-1



NOTES (Cont'd.)

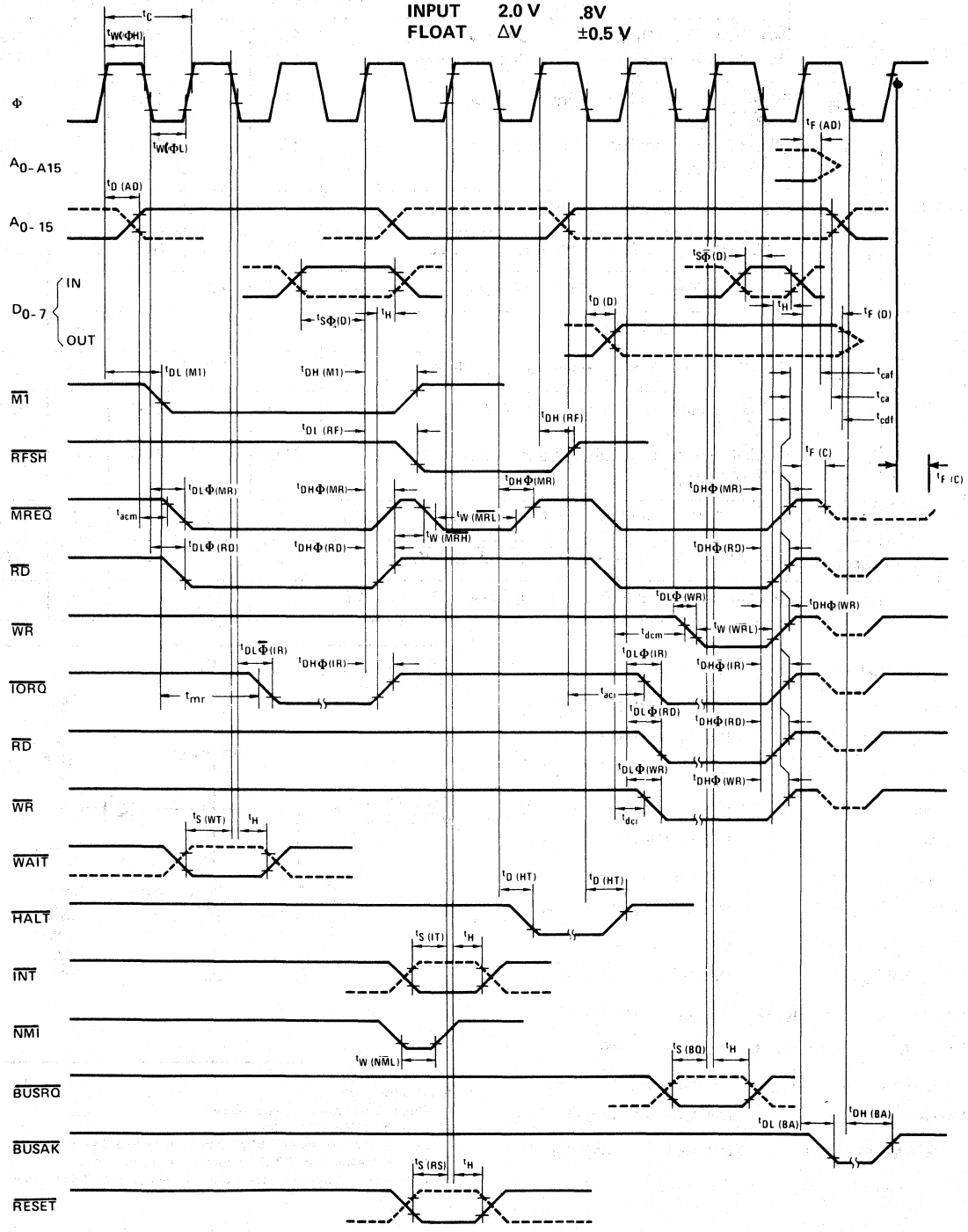
C. Output Delay vs. Load Capacitance
T_A = 70°C V_{CC} = 5 V ± 5%
Add 10 nsec delay for each 50 pF increase in load up to a maximum of 200 pF for the data bus and 100 pF for address and control lines.

D. Although static by design, testing guarantees $t_w(\Phi)$ of 200 μsec maximum.

A.C. TIMING DIAGRAM
Figure 11.0-2

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	V_{CC}–.6	.8V
OUTPUT	2.0 V	.8V
INPUT	2.0 V	.8V
FLOAT	ΔV	±0.5 V



12.0 Z80 INSTRUCTION BREAKDOWN BY MACHINE CYCLE

This section tabulates each Z80 instruction type and breaks each instruction down into its machine cycles and corresponding T States. The different standard machine cycles (OP Code Fetch, Memory Read, Port Read, etc.) are described in Section 4.0 of this manual. This chart will allow the system designer to predict what the Z80 will do on each clock cycle during the execution of a given instruction. The instruction types are listed together by functions and in the same order as the Tables in Section 7.

The best way to learn how to use these tables is to look at a few examples. The first example is to register exchange instructions (LD r, s) where r,s can be any of the following CPU Registers: B,C,D,E,H,L, or A. The instruction breakdown table shows this instruction to have one machine cycle (M1) four T States long (number in parenthesis), which is an OP Code Fetch. Referring to Figure 4.0-1, one sees the standard form for an OP Code Fetch and the state of the CPU bus during these four T-States. Taking the next instruction shown (LD r, n) which loads one of the previous registers with data or immediate value "n" one finds the breakdown to be a four T-State OP Code Fetch followed by a three T-State Operand Data Read. An Operand Data Read takes the form of the Standard Memory Read shown in Figure 4.0-2.

After these two simple examples, a more complex one is in order. The LD r, (IX+d) is the first double byte OP Code shown and executes as follows: First there are two M1 cycles (and related memory refreshes) followed by an Operand Data Read of the displacement "d". Next M3 consists of a five T-State Internal Operation which is the calculation of the Indexed address (IX+d). The last machine cycle (M4) consists of a Memory Read of the data continued in address IX+d and the loading of register "r" with that data.

The LD dd, (nn) instruction loads an internal 16-bit register pair with the contents of the memory location specified in the Operand Bytes of the instruction. This instruction is four bytes long (two bytes of OP Code + two bytes of Operand Address). As shown, there are two M1 cycles to fetch the OP code and then two Machine Cycles to read the Operand Addresses, low order byte first. Machine cycle 4 is a read of memory to obtain the data for the low order register (e.g., C of BC, E of DE, and L of HL) followed by a read of the data for the high order register.

The first instruction to use the Stack Register is the PUSH qq instruction which executes as follows: Machine cycle 1 is extended by one cycle, and the Stack Pointer is decremented in the extra T-State to point to an empty location on the Stack. Machine cycle 2 is a write of the high byte of the referenced register to the address contained in the Stack Pointer. The Stack Pointer is again decremented and a write of the low byte of the referenced register is made to the Stack in Machine Cycle 3. Note that the Stack Pointer is left pointing to the last data referenced on the Stack. The block transfer instruction such as LDI and LDIR are very similar. LDI is 16 T-States long and is composed of a double byte OP Code Fetch (two memory refreshes) followed by a memory read and a memory write. The memory write is 5 T-States long to allow updating of the block length counter —BC. The repetitive form of this instruction (LDIR) has an additional Machine Cycle (M4) of 5 T-States to allow decrementing of the Program Counter by two (PC-2) which results in refetching of the OP Code (LDIR). Each movement of data by this instruction is 21 T-States long (except the last) and the refetching of the OP Codes results in memory refresh occurring as well as the sampling of interrupts and $\overline{\text{BUSRQ}}$.

The $\overline{\text{NMI}}$ Interrupt sequence is 11 T-States long with the first M1 being a dummy OP Code Fetch of 5 T-States long. The Program Counter is not advanced, the OP Code on the data bus is ignored and an internal Restart is done to address 66 H. The following two Machine cycles are a write of the Program Counter to the Stack.

The $\overline{\text{INT}}$ Mode 0 is the 8080A mode and requires the user to place an instruction on the data bus for the CPU to execute. If a RST instruction is used, the CPU stacks the Program Counter and begins execution at the Restart Address. If a CALL instruction is used, the CALL Op Code is placed on the data bus during the INTA cycle (M1). M2 and M3 are normal Memory Read cycles (not INTA cycles) of the CALL addresses (low byte first). Program Counter is stacked in M4 and M5.

Mode 2 is used by the Z80 System Peripherals and operates as follows: During the INTA cycle (M1), a Vector is sent in from the highest priority interrupting device. M2 and M3 are used to Stack the Program Counter. The Vector (low byte) and an internal Interrupt Register (I) form a pointer to a table containing the addresses of Interrupt Service Routines. During M4 and M5, the Service Routines' address is read from this table into the CPU. The next M1 cycle will fetch an OP Code from the address received in M4 and M5.

LEGEND

IO	— Internal CPU Operation		
MR	— Memory Read	ODL	— Operand Data Read of Low Byte
MRH	— Memory Read of High Byte	PR	— Port Read
MRL	— Memory Read of Low Byte	PW	— Port Write
MW	— Memory Write	SRH	— Stack Read of High Byte
MWH	— Memory Write of High Byte	SRL	— Stack Read of Low Byte
MWL	— Memory Write of Low Byte	SWH	— Stack Write of High Byte
OCF	— Op Code Fetch	SWL	— Stack Write of Low Byte
ODH	— Operand Data Read of High Byte	( )	— Number of T-States in that Machine Cycle

Z80 INSTRUCTION BREAKDOWN BY MACHINE CODE

INSTRUCTION TYPE	BYTES	MACHINE CYCLE M1	M2	M3	M4	M5
LD r, s	1	OCF (4)				
LD r, n	2	OCF (4)	OD (3)			
LD r, (HL)	1	OCF (4)	MR (3)			
LD (HL), r		OCF (4)	MW (3)			
LD r, (IX+d)	3	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (3)	
LD (IX+d), r		OCF (4)/OCF (4)	OD (3)	IO (5)	MW (3)	
LD (HL), n	2	OCF (4)	OD (3)	MW (3)		
LD A, (BC/DE)	1	OCF (4)	MR (3)			
LD (BC/DE), A		OCF (4)	MW (3)			
LD A, (nn)	3	OCF (4)	ODL (3)	ODH (3)	MR (3)	
LD (nn) , A		OCF (4)	ODL (3)	ODH (3)	MW (3)	
LD A, I/R	2	OCF (4)/OCF(5)				
LD I/R, A						
LD dd, nn	3	OCF (4)	ODL (3)	ODH (3)		
LD IX, nn	4	OCF (4)/OCF (4)	ODL (3)	ODH (3)		
LD HL, (nn)	3	OCF (4)	ODL (3)	ODH (3)	MRL (3)	MRH (3)
LD (nn), HL		OCF (4)	ODL (3)	ODH (3)	MWL (3)	MWH (3)
LD dd, (nn)	4	OCF (4)/OCF (4)	ODL (3)	ODH (3)	MRL (3)	MRH (3)
LD (nn), dd		OCF (4)/OCF (4)	ODL (3)	ODH (3)	MWL (3)	MWH (3)
LD IX, (nn)		OCF (4)/OCF (4)	ODL (3)	ODH (3)	MRL (3)	MRH (3)
LD (nn), IX		OCF (4)/OCF (4)	ODL (3)	ODH (3)	MWL (3)	MWH (3)
LD SP, HL	1	OCF (6)				
LD SP, IX	2	OCF (4)/OCF (6)				
PUSH qq	1	OCF (5)	SWH (3)	SWL (3)		
		SP-1 →	SP-1 →			
PUSH IX	2	OCF (4)/OCF (5)	SWH (3)	SWL (3)		
		SP-1 →	SP-1 →			
POP qq	1	OCF (4)	SRH (3)	SRL (3)		
			SP+1 →		SP+1 →	
POP IX	2	OCF (4)/OCF (4)	SRH (3)	SRL (3)		
			SP+1 →		SP+1 →	
EX DE, HL	1	OCF (4)				
EX AF, AF'	1	OCF (4)				

		MACHINE CYCLE				
INSTRUCTION TYPE	**BYTES**	**M1**	**M2**	**M3**	**M4**	**M5**
EXX	1	OCF (4)				
EX (SP), HL	1	OCF (4)	SRL (3) SP+1 →	SRH (4)	SWH (3) SP-1 →	SWL (5)
EX (SP), IX	2	OCF (4)/OCF (4)	SRL (3) SP+1 →	SRH (4)	SWH (3) SP-1 →	SWL (5)
LDI LDD CPI CPD	2	OCF (4)/OCF (4)	MR (3)	MW (5)		
LDIR LDDR CPIR CPDR	2	OCF (4)/OCF (4)	MR (3)	MW (5)	IO (5)* *only if BC ≠ 0	
ALU A, r ADD ADC SUB SBC AND OR XOR CP	1	OCF (4)				
ALU A, n	2	OCF (4)	OD (3)			
ALU A, (HL)	1	OCF (4)	MR (3)			
ALU A, (IX+d)	3	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (3)	
DEC INC r	1	OCF (4)				
DEC INC (HL)	1	OCF (4)	MR (4)	MW (3)		
DEC INC (IX+D)	2	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (4)	MW (3)
DAA CPL CCF SCF NOP HALT DI EI	1	OCF (4)				
NEG IMO IM1 IM2	2	OCF (4)/OCF (4)				

INSTRUCTION TYPE	BYTES	MACHINE CYCLE M1	M2	M3	M4	M5
ADD HL, ss	1	OCF (4)	IO (4)	IO (3)		
ADC HL, ss SBC HL, ss ADD IX, pp	2	OCF (4)/OCF (4)	IO (4)	IO (3)		
INC ss DEC ss	1	OCF (6)				
DEC IX INC IX	2	OCF (4)/OCF (6)				
RLCA RLA RRCA RRA	1	OCF (4)				
RLC r RL RRC RR SLA SRA SRL	2	OCF (4)/OCF (4)				
RLC (HL) RL RRC RR SLA SRA SRL	2	OCF (4)/OCF (4)	MR (4)	MW (3)		
RLC (IX+d) RL RRC RR SLA SRA SRL	4	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (4)	MW (3)
RLD RRD	2	OCF (4)/OCF (4)	MR (3)	IO (4)	MW (3)	
BIT b, r SET RES	2	OCF (4)/OCF (4)				

III

		MACHINE CYCLE				
INSTRUCTION TYPE	**BYTES**	**M1**	**M2**	**M3**	**M4**	**M5**
BIT b, (HL)	2	OCF (4)/OCF (4)	MR (4)			
SET b, (HL) RES	2	OCF (4)/OCF (4)	MR (4)	MW (3)		
BIT b, (IX+d)	4	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (4)	
SET b, (IX+d) RES	4	OCF (4)/OCF (4)	OD (3)	IO (5)	MR (4)	MW (3)
JP nn JP cc, nn	3	OCF (4)	ODL (3)	ODH (3)		
JR e	2	OCF (4)	OD (3)	IO (5)		
JR C, e JR NC, e JR Z, e JR NZ, e	2	OCF (4)	OD (3)	IO (5)* * If condition is met		
JP (HL)	1	OCF (4)				
JP (IX)	2	OCF (4)/OCF (4)				
DJNZ, e	2	OCF (5)	OD (3)	IO (5)* * If B≠ 0		
CALL nn CALL cc, nn cc true	3	OCF (4)	ODL (3)	ODH (4) SP-1 →	SWH (3) SP-1 →	SWL (3)
CALL cc, nn cc false	3	OCF (4)	ODL (3)	ODH (3)		
RET	1	OCF (4)	SRL (3) SP+1 →	SRH (3) →	SP+1 →	
RET cc	1	OCF (5)	SRL (3)* * If cc is true SP+1 →	SRH (3)* →	SP+1 →	
RETI RETN	2	OCF (4)/OCF (4)	SRL (3) SP+1 →	SRH (3) →	SP+1 →	
RST p	1	OCF (5) SP-1 →	SWH (3) SP-1 →	SWL (3) →		

		MACHINE CYCLE				
INSTRUCTION TYPE	BYTES	M1	M2	M3	M4	M5
IN A, (n)	2	OCF (4)	OD (3)	PR (4)		
IN r, (c)	2	OCF (4)/OCF (4)	PR (4)			
INI IND	2	OCF (4)/OCF (5)	PR (4)	MW (3)		
INIR INDR	2	OCF (4)/OCF (5)	PR (4)	MW (3)	IO (5)	
OUT (n) , A	2	OCF (4)	OD (3)	PW (4)		
OUT (C), r	2	OCF (4)/OCF (4)	PW (4)			
OUTI OUTD	2	OCF (4)/OCF (5)	MR (3)	PW (4)		
OTIR OTDR	2	OCF (4)/OCF (5)	MR (3)	PW (4)	IO (5)	
<u>INTERRUPTS</u>						
NMI	–	OCF (5) * SP-1 →	SWH (3) SP-1 →	SWL (3)	*Op Code Ignored	
INT						
MODE 0	–	INTA (6) (CALL INSERTED)	ODL (3)	ODH (4) SP-1 →	SWH (3) SP-1 →	SWL (3)
	–	INTA (6) (RST INSERTED) SP-1 →	SWH (3) SP-1 →	SWL (3)		
MODE 1		INTA (7) (RST 38H INTERNAL) SP-1 →	SWH (3) SP-1 →	SWL (3)		
MODE 2	–	INTA (7) (VECTOR SUPPLIED) SP-1 →	SWH (3) SP-1 →	SWL (3)	MRL (3)	MRH (3)

III

13.0 ORDERING INFORMATION

PART NO.	PACKAGE TYPE	MAX CLOCK FREQUENCY	TEMPERATURE RANGE
MK3880N Z80-CPU	Plastic	2.5 MHz	0° to + 70°C
MK3880P Z80-CPU	Ceramic	2.5 MHz	
MK3880N-4 Z80-CPU	Plastic	4.0 MHz	
MK3880P-4 Z80-CPU	Ceramic	4.0 MHz	
MK3880P-10 Z80-CPU	Ceramic	2.5 MHz	–40°C to +85°C

MOSTEK®

Z80 MICROCOMPUTER DEVICES

Technical Manual

III

MK3881 PARALLEL I/O CONTROLLER

TABLE OF CONTENTS

1.0 INTRODUCTION

The Z80 Parallel I/O Circuit is a programmable, two port device which provides a TTL compatible interface between peripheral devices and the Z80-CPU. The CPU can configure the Z80-PIO to interface with a wide range of peripheral devices with no other external logic required. Typical peripheral devices that are fully compatible with the Z80-PIO include most keyboards, paper tape readers and punches, printers, PROM programmers, etc. The Z80-PIO utilizes N channel silicon gate depletion load technology and is packaged in a 40 pin DIP. Major features of the Z80-PIO include:

- Two independent 8 bit bidirectional peripheral interface ports with 'handshake' data transfer control
- Interrupt driven 'handshake' for fast response
- Any one of four distinct modes of operation may be selected for a port including:

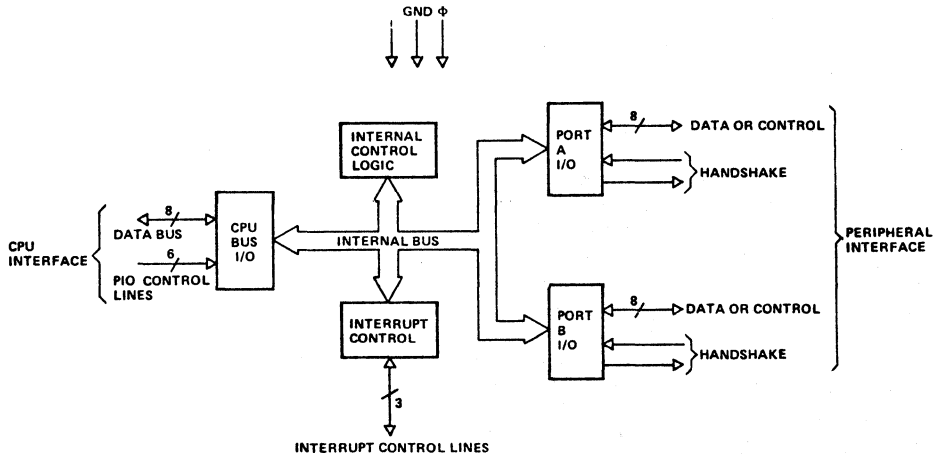
  Byte output
  Byte input
  Byte bidirectional bus (Available on Port A only)
  Bit control mode
  All interrupt controlled handshake
- Daisy chain priority interrupt logic included to provide for automatic interrupt vectoring without external logic
- Eight outputs are capable of driving Darlington transistors
- All inputs and outputs fully TTL compatible
- Single 5 volt supply and single phase clock required

One of the unique features of the Z80-PIO that separates it from other interface controllers is that all data transfer between the peripheral device and the CPU is accomplished under total interrupt control. The interrupt logic of the PIO permits full usage of the efficient interrupt capabilities of the Z80-CPU during I/O transfers. All logic necessary to implement a fully nested interrupt structure is included in the PIO so that additional circuits are not required. Another unique feature of the PIO is that it can be programmed to interrupt the CPU on the occurrence of specified status conditions in the peripheral device. For example, the PIO can be programmed to interrupt if any specified peripheral alarm conditions occur. This interrupt capability reduces the amount of time that the processor must spend in polling peripheral status.

2.0 PIO ARCHITECTURE

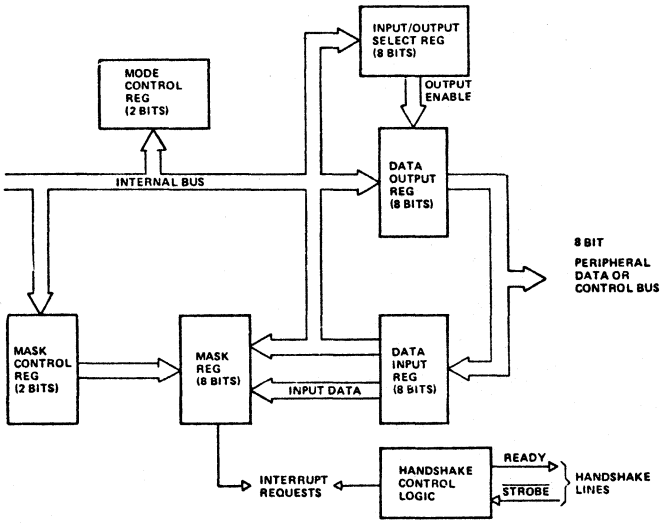
A block diagram of the Z80-PIO is shown in figure 2.0-1. The internal structure of the Z80-PIO consists of a Z80-CPU bus interface, internal control logic, Port A I/O logic, Port B I/O logic, and interrupt control logic. The CPU bus interface logic allows the PIO to interface directly to the Z80-CPU with no other external logic. However, address decoders and/or line buffers may be required for large systems. The internal control logic synchronizes the CPU data bus to the peripheral device interfaces (Port A and Port B). The two I/O ports (A and B) are virtually identical and are used to interface directly to the peripheral devices.

PIO BLOCK DIAGRAM
Figure 2.0-1



The Port I/O logic is composed of 6 registers with "handshake" control logic as shown in figure 2.0-2. The registers include: an 8-bit data input register, an 8-bit data output register, a 2-bit mode control register, an 8-bit mask register, an 8-bit input/output select register and a 2-bit mask control register.

PORT I/O BLOCK DIAGRAM
Figure 2.0-2



The 2-bit mode control register is loaded by the CPU to select the desired operating mode (byte output, byte input, byte bidirectional bus, or bit control mode). All data transfer between the peripheral device and the CPU is achieved through the data input and data output registers. Data may be written into the output register by the CPU or read back to the CPU from the input register at any time. The handshake lines associated with each port are used to control the data transfer between the PIO and peripheral device.

The 8-bit mask register and the 8-bit input/output select register are used only in the bit control mode. In this mode any of the 8 peripheral data control bus pins can be programmed to be an input or an output as specified by the select register. The mask register is used in this mode in conjunction with a special interrupt feature. This feature allows an interrupt to be generated when any or all of the unmasked pins reach a specified state (either high or low). The 2-bit mask control register specifies the active state desired (high or low) and whether the interrupt should be generated when all unmasked pins are active (AND condition) or when any unmasked pin is active (OR condition). This feature reduces the requirement for CPU status checking of the peripheral by allowing an interrupt to be automatically generated on specific peripheral status conditions. For example, in a system with 3 alarm conditions, an interrupt may be generated if any one alarm condition occurs or if all three occur.

The interrupt control logic section handles all CPU interrupt protocol for nested priority interrupt structures. The priority of any device is determined by its physical location in a daisy chain configuration. Two lines are provided in each PIO to form this daisy chain. The device closest to the CPU has the highest priority. Within a PIO, Port A interrupts have higher priority than those of Port B. In the byte input, byte output, or bidirectional modes, an interrupt can be generated whenever a new byte transfer is requested by the peripheral. In the bit control mode an interrupt can be generated when the peripheral status matches a programmed value. The PIO provides for complete control of nested interrupts. That is, lower priority devices may not interrupt higher priority devices that have not had their interrupt service routine completed by the CPU. Higher priority devices may interrupt the servicing of lower priority devices.

When an interrupt is accepted by the CPU in mode 2, the interrupting device must provide an 8-bit interrupt vector for the CPU. This vector is used to form a pointer to a location in the computer memory where the address of the interrupt service routine is located. The 8-bit vector from the interrupting device forms the least significant 8 bits of the indirect pointer while the I Register in the CPU provides the most significant 8 bits of the pointer. Each port (A and B) has an independent interrupt vector. The least significant vector is automatically set to a 0 within the PIO since the pointer must point to two adjacent memory locations for a complete 16-bit address.

The PIO decodes the RETI (Return from interrupt) instruction directly from the CPU data bus so that each PIO in the system knows at all times whether it is being serviced by the CPU interrupt service routine without any other communication with the CPU.

3.0 PIN DESCRIPTION

A diagram of the Z80-PIO pin configuration is shown in figure 3.0-1. This section describes the function of each pin.

D_7-D_0 — Z80-CPU Data Bus (bidirectional, tristate)
This bus is used to transfer all data and commands between the Z80-CPU and the Z80-PIO. D_0 is the least significant bit of the bus.

B/$\overline{A}$ Sel — Port B or A Select (input, active high)
This pin defines which port will be accessed during a data transfer between the Z80-CPU and the Z80-PIO. A low level on this pin selects Port A while a high level selects Port B. Often Address bit A_0 from the CPU will be used for this selection function.

C/$\overline{D}$ Sel — Control or Data Select (input, active high)
This pin defines the type of data transfer to be performed between the CPU and the PIO. A high level on this pin during a CPU write to the PIO causes the Z80 data bus to be interpreted as a command for the port selected by the B/A Select line. A low level on this pin means that the Z80 data bus is being used to transfer data between the CPU and the PIO. Often Address bit A_1 from the CPU will be used for this function.

$\overline{CE}$ — Chip Enable (input, active low)
A low level on this pin enables the PIO to accept command or data inputs from the CPU during a write cycle or to transmit data to the CPU during a read cycle. This signal is generally a decode of four I/O port numbers that encompass port A and B, data and control.

Φ — System Clock (input)
The Z80-PIO uses the standard Z80 system clock to synchronize certain signals internally. This is a single phase clock.

$\overline{M1}$ — Machine Cycle One Signal from CPU (input, active low)
The signal from the CPU is used as a sync pulse to control several internal PIO operations. When $\overline{M1}$ is active and the $\overline{RD}$ signal is active, the Z80-CPU is fetching an instruction from memory. Conversely, when $\overline{M1}$ is active and $\overline{IORQ}$ is active, the CPU is acknowledging an interrupt. In addition, the $\overline{M1}$ signal has two other functions within the Z80-PIO.

1. $\overline{M1}$ synchronizes the PIO interrupt logic.

2. When $\overline{M1}$ occurs without an active $\overline{RD}$ or $\overline{IORQ}$ signal, the PIO logic enters a reset state.

$\overline{IORQ}$ — Input/Output Request from Z80-CPU (input, active low)
The $\overline{IORQ}$ signal is used in conjuction with the B/A Select, C/D Select, $\overline{CE}$, and $\overline{RD}$ signals to transfer commands and data between the Z80-CPU and the Z80-PIO. When $\overline{CE}$, $\overline{RD}$ and $\overline{IORQ}$ are active, the port addressed by B/A will transfer data to the CPU (a read operation). Conversely, when $\overline{CE}$ and $\overline{IORQ}$ are active but $\overline{RD}$ is not active, then the port addressed by B/A will be written into from the CPU with either data or control information as specified by the C/D Select signal. Also if $\overline{IORQ}$ and $\overline{M1}$ are active simultaneously, the CPU is acknowledging an interrupt and the interrupting port will automatically place its interrupt vector on the CPU data bus if it is the highest device requesting an interrupt.

$\overline{RD}$ — Read Cycle Status from the Z80-CPU (input, active low)
If $\overline{RD}$ is active a MEMORY READ or I/O READ operation is in progress. The $\overline{RD}$ signal is used with B/A Select, C/D Select, $\overline{CE}$ and $\overline{IORQ}$ signals to transfer data from the Z80-PIO to the Z80-CPU.

IEI — Interrupt Enable In (input, active high)
This signal is used to form a priority interrupt daisy chain when more than one interrupt driven device is being used. A high level on this pin indicates that no other devices of higher priority are being serviced by a CPU interrupt service routine.

IEO — Interrupt Enable Out (output, active high)
The IEO signal is the other signal required to form a daisy chain priority scheme. It is high only if IEI is high and the CPU is not servicing an interrupt from this PIO. Thus this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU service routine.

$\overline{INT}$ — Interrupt Request (output, open drain, active low)
When INT is active the Z80-PIO is requesting an interrupt from the Z80-CPU.

A_0-A_7 — Port A Bus (bidirectional, tristate)
This 8 bit bus is used to transfer data and/or status or control information between Port A or the Z80-PIO and a peripheral device. A_0 is the least significant bit of the Port A data bus.

$\overline{A\ \ STB}$ — Port A Strobe Pulse from Peripheral Device (input, active low)
The meaning of this signal depends on the mode of operation selected for Port A as follows:

1) Output mode: The positive edge of this strobe is issued by the peripheral to acknowledge the receipt of data made available by the PIO.

2) Input mode: The strobe is issued by the peripheral to load data from the peripheral into the Port A input register. Data is loaded into the PIO when this signal is active.

3) Bidirectional mode: When this signal is active, data from the Port A output register is gated onto Port A bidirectional data bus. The positive edge of the strobe acknowledges the receipt of the data.

4) Control mode: The strobe is inhibited internally.

A RDY — Register A Ready (output, active high)
The meaning of this signal depends on the mode of operation selected for Port A as follows:

1) Output mode: This signal goes active to indicate that the Port A output register has been loaded and the peripheral data bus is stable and ready for transfer to the peripheral device.

2) Input mode: This signal is active when the Port A input register is empty and is ready to accept data from the peripheral device.

3) Bidirectional mode: This signal is active when data is available in Port A output register for transfer to the peripheral device. In this mode data is not placed on the Port A data bus unless $\overline{A\ \ STB}$ is active.

4) Control mode: This signal is disabled and forced to a low state.

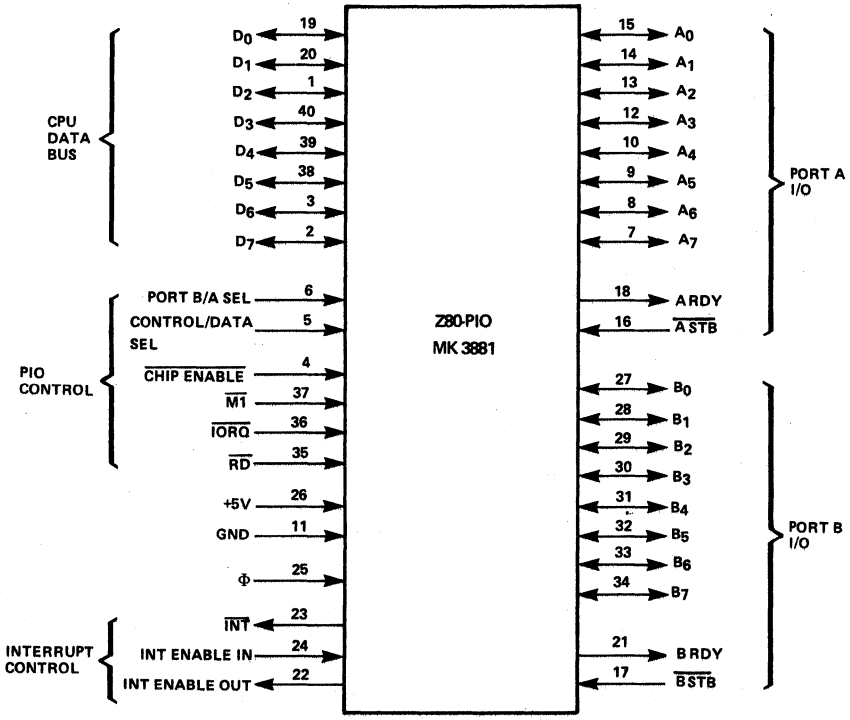
B_0-B_7 — Port B Bus (bidirectional, tristate)
This 8 bit bus is used to transfer data and/or status or control information between Port B of the PIO and a peripheral device. The Port B data bus is capable of supplying 1.5ma @ 1.5 V to drive Darlington transistors. B_0 is the least significant bit of the bus.

$\overline{B\ \ STB}$ — Port B Strobe Pulse from Peripheral Device (input, active low)
The meaning of this signal is similar to that of $\overline{A\ \ STB}$ with the following exception:
In the Port A bidirectional mode this signal strobes data from the peripheral device into the Port A input register.

B RDY — Register B Ready (output, active high)
The meaning of this signal is similar to that of A Ready with the following exception:
In the Port A bidirectional mode this signal is high when the Port A input register is empty and ready to accept data from the peripheral device.

PIO PIN CONFIGURATION

Figure 3.0-1



III

4.0 PROGRAMMING THE PIO

4.1 RESET

The Z80-PIO automatically enters a reset state when power is applied. The reset state performs the following functions:

1) Both port mask registers are reset to inhibit all port data bits.

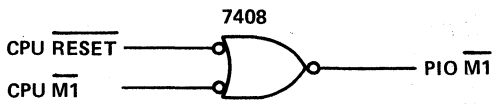
2) Port data bus lines are set to a high impedance state and the Ready "handshake" signals are inactive (low). Mode 1 is automatically selected.

3) The vector address registers are not reset.

4) Both port interrupt enable flip flops are reset.

5) Both port output registers are reset.

In addition to the automatic power on reset, the PIO can be reset by applying an $\overline{M1}$ signal without the presence of a $\overline{RD}$ or $\overline{IORQ}$ signal. If no $\overline{RD}$ or $\overline{IORQ}$ is detected during $\overline{M1}$ the PIO will enter the reset state immediately after the $\overline{M1}$ signal goes inactive. The purpose of this reset is to allow a single external gate to generate a reset without a power down sequence. This approach was required owing to the 40 pin packaging limitation. It is recommended that in breadboard systems with a "Reset" push button that $\overline{M1}$ reset be implemented for the PIO.



A software RESET is possible as described in Section 4.4; however, use of this method during early system debug may not be desirable because of non-functional system hardware (bus buffers or memory for example).

Once the PIO has entered the internal reset state it is held there until the PIO receives a control word from the CPU.

4.2 LOADING THE INTERRUPT VECTOR

The PIO has been designed to operate with the Z80-CPU using the mode 2 interrupt response. This mode requires that an interrupt vector be supplied by the interrupting device. This vector is used by the CPU to form the address for the interrupt service routine of that port. This vector is placed on the Z80 data bus during an interrupt acknowledge cycle by the highest priority device requesting service at that time. (Refer to the Z80-CPU Technical Manual for details on how an interrupt is serviced by the CPU). The desired interrupt vector is loaded into the PIO by writing a control word to the desired port of the PIO with the following format:

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	0

signifies this control word is an interrupt vector

D0 is used in this case as a flag bit which when low causes V7 through V1 to be loaded into the vector register. At interrupt acknowledge time, the vector of the interrupting port will appear on the Z80 data bus exactly as shown in the format above.

4.3 SELECTING AN OPERATING MODE

Port A of the PIO may be operated in any of four distinct modes: Mode 0 (output mode), Mode 1 (input mode), Mode 2 (bidirectional mode), and Mode 3 (control mode). Note that the mode numbers have been selected for mnemonic significance; i.e. 0 = Out, 1 = In, 2 = Bidirectional. Port B can operate in any of these modes except Mode 2.

The mode of operation must be established by writing a control word to the PIO in the following format:

D7	D6	D5	D4	D3	D2	D1	D0
M1	M0	X	X	1	1	1	1

X=unused bit

mode word — signifies mode word to be set

Bits D7 and D6 form the binary code for the desired mode according to the following table:

D7	D6	MODE
0	0	0 (output)
0	1	1 (input)
1	0	2 (bidirectional)
1	1	3 (control)

Bits D5 and D4 are ignored. Bits D3-D0 must be set to 1111 to indicate "Set Mode".

Selecting Mode 0 enables any data written to the port output register by the CPU to be enabled onto the port data bus. The contents of the output register may be changed at any time by the CPU simply by writing a new data word to the port. Also the current contents of the output register may be read back to the Z80-CPU at any time through the execution of an input instruction.

With Mode 0 active, a data write from the CPU causes the Ready handshake line of that port to go high to notify the peripheral that data is available. This signal remains high until a strobe is received from the peripheral. The rising edge of the strobe generates an interrupt (if it has been enabled) and causes the Ready line to go inactive. This very simple handshake is similar to that used in many peripheral devices.

Selecting Mode 1 puts the port into the input mode. To start handshake operation, the CPU merely performs an input read operation from the port. This activates the Ready line to the peripheral to signify that data should be loaded into the empty input register. The peripheral device then strobes data into the port input register using the strobe line. Again, the rising edge of the strobe causes an interrupt request (if it has been enabled) and deactivates the Ready signal. Data may be strobed into the input register regardless of the state of the Ready signal if care is taken to prevent a data overrun condition.

Mode 2 is a bidirectional data transfer mode which uses all four handshake lines. Therefore only Port A may be used for Mode 2 operation. Mode 2 operation uses the Port A handshake signals for

output control and the Port B handshake signals for input control. Thus, both A RDY and B RDY may be active simultaneously. The only operational difference between Mode 0 and the output portion of Mode 2 is that data from the Port A output register is allowed on to the port data bus only when $\overline{A\ STB}$ is active in order to achieve a bidirectional capability.

Mode 3 operation is intended for status and control applications and does not utilize the handshake signals. When Mode 3 is selected, the next control word sent to the PIO must define which of the port data bus lines are to be inputs and which are to be outputs. The format of the control word is shown below:

D7	D6	D5	D4	D3	D2	D1	D0
I/O_7	I/O_6	$IO/_5$	I/O_4	I/O_3	I/O_2	I/O_1	I/O_0

If any bit is set to a one, then the corresponding data bus line will be used as an input. Conversely, if the bit is reset, the line will be used as an output.

During Mode 3 operation the strobe signal is ignored and the Ready line is held low. Data may be written to a port or read from a port by the Z80-CPU at any time during Mode 3 operation. (An exception to this is when Port A is in Mode 2 and Port B is in Mode 3). When reading a port, the data returned to the CPU will be composed of input data from port data bus lines assigned as inputs plus output register data from those lines assigned as outputs.

4.4 SETTING THE INTERRUPT CONTROL WORD

The interrupt control word for each port has the following format:

D7	D6	D5	D4	D3	D2	D1	D0
Enable Interrupt	AND/ OR	High/ Low	Masks follows	0	1	1	1
	used in Mode 3 only			signifies interrupt control word			

If bit D7 = 1, the interrupt enable flip flop of the port is set and the port may generate an interrupt. If bit D7 = 0, the enable flag is reset and interrupts may not be generated. If an interrupt occurs while D7 = 0, then it will be latched internally by the PIO and passed onto the CPU when PIO Interrupts are Re-Enabled (D7 = 1). Bits D6, D5 and D4 are used mainly with Mode 3 operation; however, setting bit D4 of the interrupt control word during any mode of operation will cause a pending interrupt to be reset. These three bits are used to allow for interrupt operation in Mode 3 when any group of the I/O lines go to certain defined states. Bit D6 (AND/OR) defines the logical operation to be performed in port monitoring. If bit D6 = 1, an AND function is specified and if D6 = 0, an OR function is specified. For example, if the AND function is specified, all bits must go to a specified state before an interrupt will be generated while OR function will generate an interrupt if any specified bit goes to the active state.

Bit D5 defines the active polarity of the port data bus line to be monitored. If bit D5 = 1, the port data lines are monitored for a high state, while if D5 = 0, they will be monitored for a low state.

If bit D4 = 1 the next control word sent to the PIO must define a mask as follows:

D7	D6	D5	D4	D3	D2	D1	D0
MB_7	MB_6	MB_5	MB_4	MB_3	MB_2	MB_1	MB_0

Only these port lines whose mask bit is zero will be monitored for generating an interrupt.

The interrupt enable flip flop of a port may be set or reset without modifying the rest of the interrupt control word by using the following command:

Int Enable	X	X	X	0	0	1	1

If an external Asynchronous interrupt could occur while the processor is writing the disable word to the PIO (03H) then a system problem may occur. If interrupts are enabled in the processor, it is possible that the Asynchronous interrupt will occur while the processor is writing the disable word to the PIO. The PIO will generate an INT and the CPU will acknowledge it; however, by this time, the PIO will have received the disable word and deactivated its interrupt structure. The result is that the PIO will not send in its interrupt vector during the interrupt acknowledge cycle because it is disabled and the CPU will fetch an erroneous vector resulting in a program fault. The cure for this problem is to disable interrupts within the CPU with the DI instruction just before the PIO is disabled and then re-enable interrupts with the EI instruction. This action causes the CPU to ignore any faulty interrupts produced by the PIO while it is being disabled. The code sequence would be:

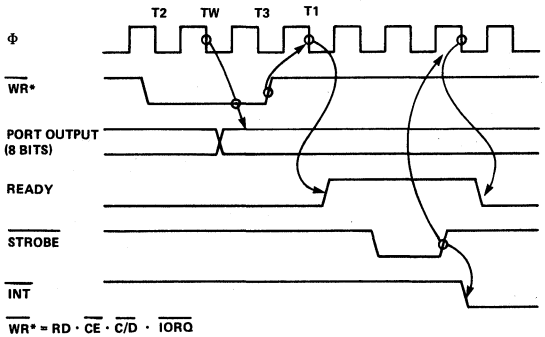
```
.
.
LD A,03H
DI                          ; DISABLE CPU
OUT (PIO),A                 ; DISABLE PIO
EI                          ; ENABLE CPU
.
.
```

5.0 TIMING

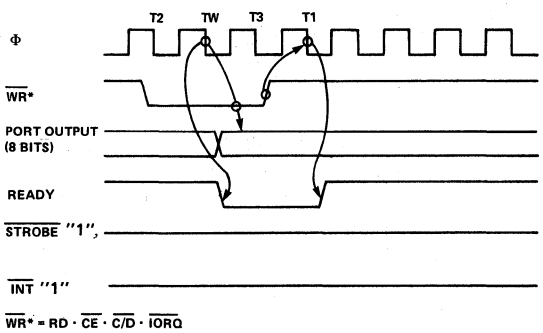
5.1 OUTPUT MODE (MODE 0)

Figure 5.0-1a illustrates the timing associated with Mode 0 operation. An output cycle is always started by the execution of an output instruction by the CPU. A $\overline{WR}$* pulse is generated by the PIO during a CPU I/O write operation and is used to latch the data from the CPU data bus into addressed port's (A or B) output register. The rising edge of the $\overline{WR}$* pulse then raises the READY line after the next falling edge of Φ to indicate that data is available for the peripheral device. In most systems, the rising edge of the READY signal can be used as a latching signal in the peripheral device. The READY signal will remain active until a positive edge is received from the $\overline{STROBE}$ line indicating that the peripheral has taken the data shown in Figure 5.0-1a. If already active, READY will be forced low 1½ Φ cycles after the falling edge of $\overline{IORQ}$ if the port's output register is written into. READY will return high on the first falling edge of Φ after the rising edge of $\overline{IORQ}$ as shown in Figure 5.0-1b. This action guarantees that READY is low while port data is changing and that a positive edge is generated on READY whenever an Output instruction is executed.

MODE 0 (OUTPUT) TIMING
Figure 5.0-1a



MODE 0 (OUTPUT) TIMING
Figure 5.0-1b

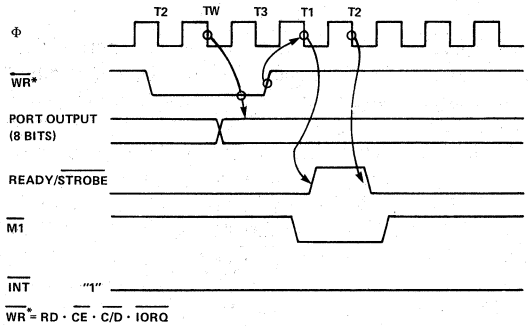


By connecting READY to $\overline{STROBE}$, a positive pulse with a duration of one clock period can be created as shown in Figure 5.0-1c. The positive edge of READY/$\overline{STROBE}$ will not generate an interrupt because the positive portion of $\overline{STROBE}$ is less than the width of $\overline{M1}$ and as such will not generate an interrupt due to the internal logic configuration of the PIO.

If the PIO is not in a reset status (i.e. a control mode has been selected), the output register may be loaded before Mode 0 is selected. This allows port output lines to become active in a user defined state. For example, assume the outputs are desired to become active in a logic one state. The following would be the initialization sequence:

a) PIO RESET
b) Load Interrupt Vector
c) Select Mode 1 (input) (automatic due to RESET)
d) Write FF to Data Port
e) Select Mode 0 (Outputs go to "1s")
f) Enable Interrupt if desired

MODE 0 (OUTPUT) TIMING - READY TIED TO STROBE
Figure 5.0-1c

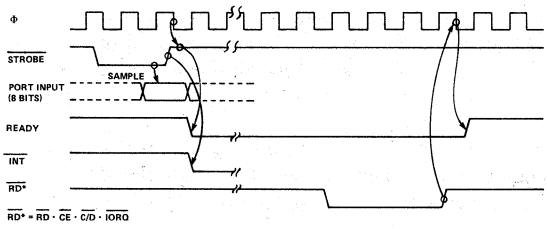


5.2 INPUT MODE (MODE 1)

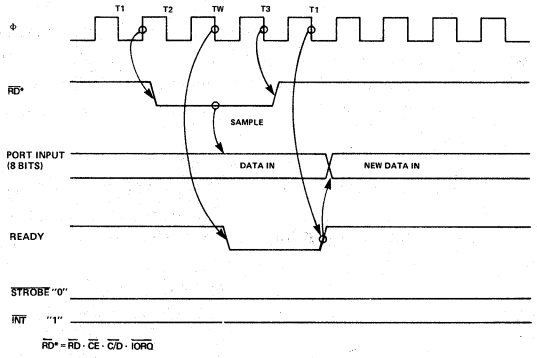
Figure 5.0-2 illustrates the timing of an input cycle. The peripheral initiates this cycle using the STROBE line after the CPU has performed a data read. A low level on this line loads data into the port input register and the rising edge of the STROBE line activates the interrupt request line (INT) if the interrupt enable is set, and this is the highest priority requesting device. The next falling edge of the clock line (Φ) will then reset the READY requesting line to an inactive state signifying that the input register is full and further loading must be inhibited until the CPU reads the data. The CPU will, in the course of its interrupt service routine, read the data from the interrupting port. When this occurs, the positive edge from the CPU RD signal will raise the READY line with the next low going transition of Φ, indicating that new data can be loaded into the PIO.

Since RESET causes READY to go low, a dummy Input instruction may be needed in some systems to cause READY to go high the first time in order to start "handshaking".

MODE 1 (INPUT) TIMING
Figure 5.0-2a



MODE 1 (INPUT) TIMING (NO STROBE INPUT)
Figure 5.0-2b



MODE 1 (INPUT) TIMING (NO STROBE INPUT)

If already active, READY will be forced low one and one-half Φ periods following the falling edge of IORQ during a read of a PIO port as shown in Figure 5.0-2b. If the user strobes data into the PIO only when READY is high, the forced state of READY will prevent input register data from changing while the CPU is reading the PIO. READY will go high again after the rising edge of the IORQ as previously described.

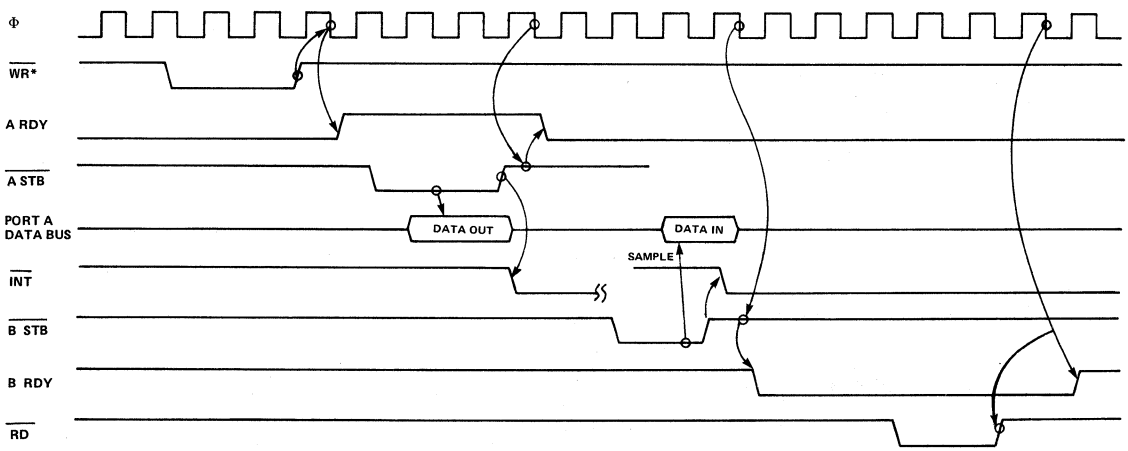
5.3 BIDIRECTIONAL MODE (MODE 2)

This mode is merely a combination of Mode 0 and Mode 1 using all four handshake lines. Since it requires all four lines, it is available only on Port A. When this mode is used on Port A, Port B must be set to the Bit Control Mode. The same interrupt vector will be returned for a Mode 3 interrupt on Port B and an input transfer interrupt during Mode 2 operation of Port A. Ambiguity is avoided if Port B is operated in a polled mode and the Port B mask register is set to inhibit all bits. Furthermore, interrupts from Port B (Mode 3) will not be generated when Port A is programmed for Mode 2, as $\overline{BSTB}$ would have to be active (low) in order to generate interrupts. ($\overline{BTSB}$ is normally high).

Figure 5.0-3 illustrates the timing for this mode. It is almost identical to that previously described for Mode 0 and Mode 1 with the Port A handshake lines used for output control and the Port B lines used for input control. The difference between the two modes is that in Mode 2, data is allowed out onto the bus only when the A $\overline{STROBE}$ is low. The rising edge of this strobe can be used to latch the data into the peripheral since the data will remain stable until after this edge. The input portion of Mode 2 operates identically to Mode 1. Note that both Port A and Port B must have their interrupts enabled to acheive an interrupt driven bidirectional transfer.

III

PORT A, MODE 2 (BIDIRECTIONAL) TIMING
Figure 5.0-3



$\overline{WR}^* = RD \cdot \overline{CE} \cdot \overline{C/D} \cdot \overline{IORQ}$
$\overline{RD}^* = \overline{RD} \cdot \overline{CE} \cdot \overline{C/D} \cdot \overline{IORQ}$

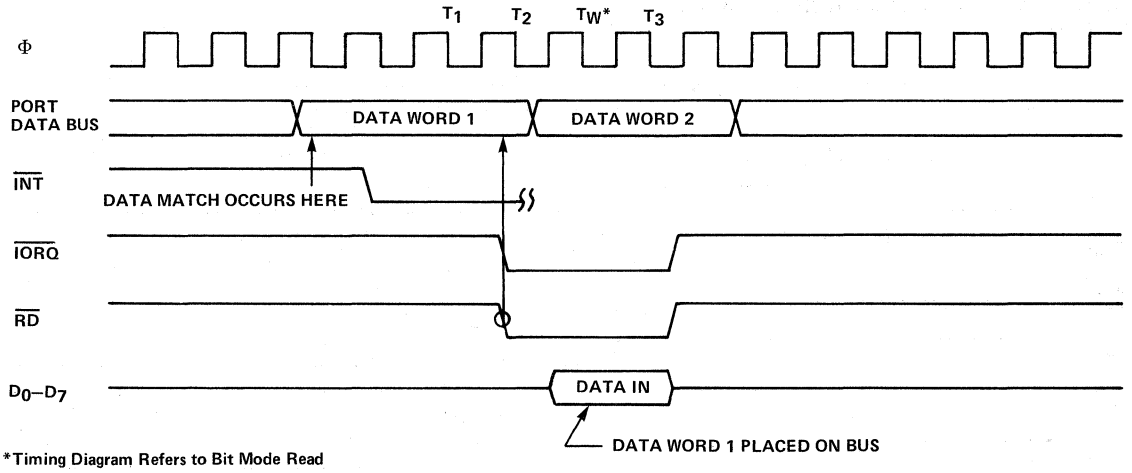
The peripheral must not gate onto a port data bus while $\overline{A\ STB}$ is active. Bus contention is avoided if the peripheral uses $\overline{B\ STB}$ to gate input data onto the bus. The PIO uses the $\overline{B\ STB}$ low level to sample this data. The PIO has been designed with a zero hold time requirement for the data when latching in this mode so that this simple gating structure can be used by the peripheral. That is, the data can be displayed from the bus immediately after the strobe rising edge. Note that if $\overline{A\ STB}$ is low during a read operation of Port A (in response to a $\overline{B\ STB}$ interrupt) the data in the output register will be read by the CPU instead of the correct data in the data input register. The correct data is latched in the input register; it just cannot be read by the CPU while $\overline{A\ STB}$ is low. If the $\overline{A\ STB}$ signal should go low during a CPU Read, it would be blocked from reaching the $\overline{A\ STB}$ input of the PIO while BRDY is low (the CPU read will occur while BRDY is low as the $\overline{RD}$ signal returns BRDY high).

5.4 BIT CONTROL MODE (MODE 3)

The bit control mode does not utilize the handshake signals and a normal port write or port read can be executed at any time. When writing, the data will be latched into output registers with the same timing as Mode 0. A RDY will be forced low whenever Port A is operated in Mode 3. B RDY will be held low whenever Port B is operated in Mode 3 unless Port A is in Mode 2. In the latter case, the state of B RDY will not be affected.

When reading the PIO, the data returned to the CPU will be composed of output register data from those port data lines assigned as outputs and input register data from those port data lines assigned as inputs. The input register will contain data which was present immediately prior to the falling edge of $\overline{RD}$. See Figure 5.0-4.

MODE 3 TIMING
Figure 5.0-4a



*Timing Diagram Refers to Bit Mode Read

An interrupt will be generated if interrupts from the port are enabled and if the data on the port data lines satisfies the logical equation defined by the 8-bit mask control registers. Another interrupt will not be generated until a change occurs in the status of the logical equation. A Mode 3 interrupt will be generated only if the result of a Mode 3 logical operation changes from false to true. For example, assume that the Mode 3 logical equation is an "OR" function. An unmasked port data line becomes active and an interrupt is requested. If a second unmasked port data line becomes active concurrently with the first, a new interrupt will not be requested since a change in the result of the Mode 3 logical operation has not occurred. Note that port pins defined as outputs can contribute to the logical equation if their bit positions are unmasked.

If the result of a logical operation becomes true immediately prior to or during $\overline{M1}$, an interrupt will be requested after the trailing edge of $\overline{M1}$, provided the logical equation remains true after $\overline{M1}$ returns high.

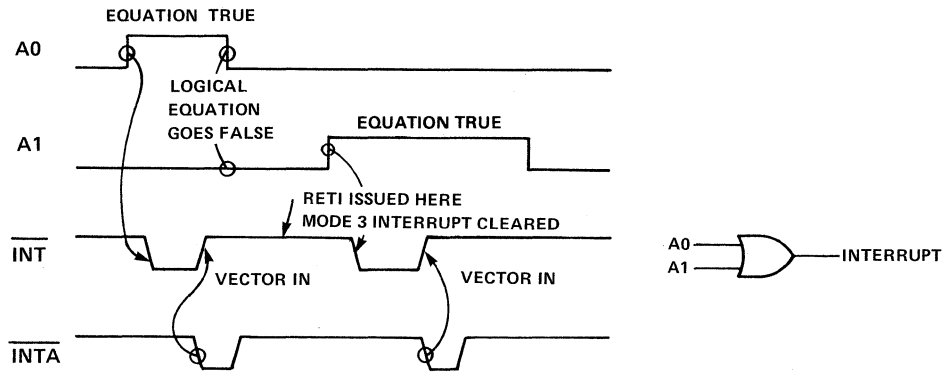
Figure 5.0-4b is an example of Mode 3 interrupts. The port has been placed in Mode 3 with OR logic selected and signals defined high. All but bits A0 and A1 are masked out and are not monitored thereby creating a two input positive logic OR gate. In the timing diagram, A0 is shown going high and creating an interrupt ($\overline{INT}$ goes low) and the CPU responds with an Interrupt Acknowledge cycle ($\overline{INTA}$). The PIO port with its interrupt pending sends in its Vector, and the CPU goes off into the Interrupt Service Routine. A0 is shown going inactive either by itself or perhaps as a result of action taken in the Interrupt Service Routine (making the logical equation false). An arrow is shown at the point in time where the Service Routine issues the RETI instruction which clears the PIO interrupt structure. A1 is next shown going high, making the logical equation true and generating another interrupt. Two important points need to be made from this example:

1) A1 must not go high before A0 goes low or else the logical equation will not go false -a requirement for A1 to be able to generate an interrupt.

2) In order for A1 to generate an interrupt it must be high after the RETI issued by A0's Service Routine clears the PIO's Interrupt structure. In other words, if A 1 were a positive pulse that occurred after A0 went low (to make the equation false) and went low before the RETI had cleared the Interrupt Structure, it would have been missed. The logic equation must become false after the $\overline{INTA}$ for A0's service and then must be true or go true after RETI clears the previous interrupt for another interrupt to occur.

III

MODE 3 EXAMPLE
Figure 5.0-4b



6.0 INTERRUPT SERVICING

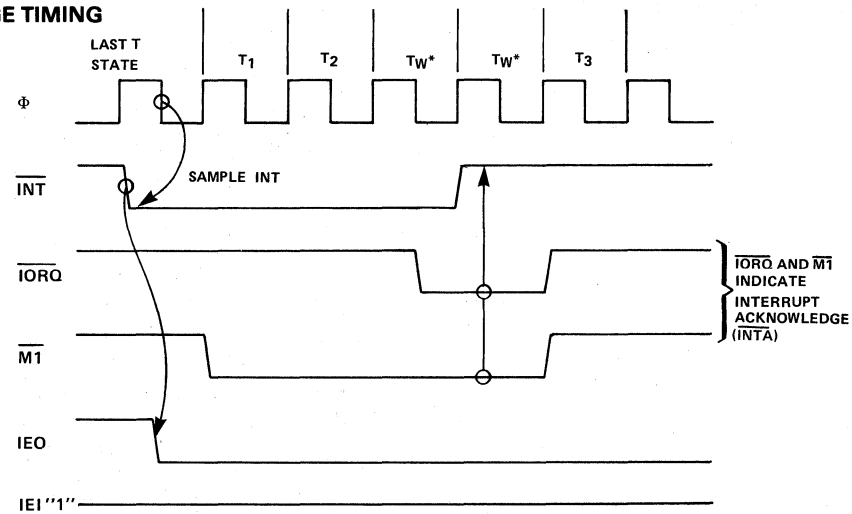
Some time after an interrupt is requested by the PIO, the CPU will send out an interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$). During this time the interrupt logic of the PIO will determine the highest priority port which is requesting an interrupt. (This is simply the device with its Interrupt Enable Output low). To insure that the daisy chain enable lines stabilize, devices are inhibited from changing their interrupt request status when $\overline{M1}$ is active. The highest priority device places the contents of its interrupt vector register onto the Z80 data bus during interrupt acknowledge.

Figure 6.0-1 illustrates the timing associated with interrupt requests. During $\overline{M1}$ time, no new interrupt requests can be generated. This gives time for the Interrupt Enable signals to ripple through up to four PIO circuits. The PIO with IEI high and IEO low during $\overline{INTA}$ will place the 8-bit interrupt vector of the appropriate port on the data bus at this time.

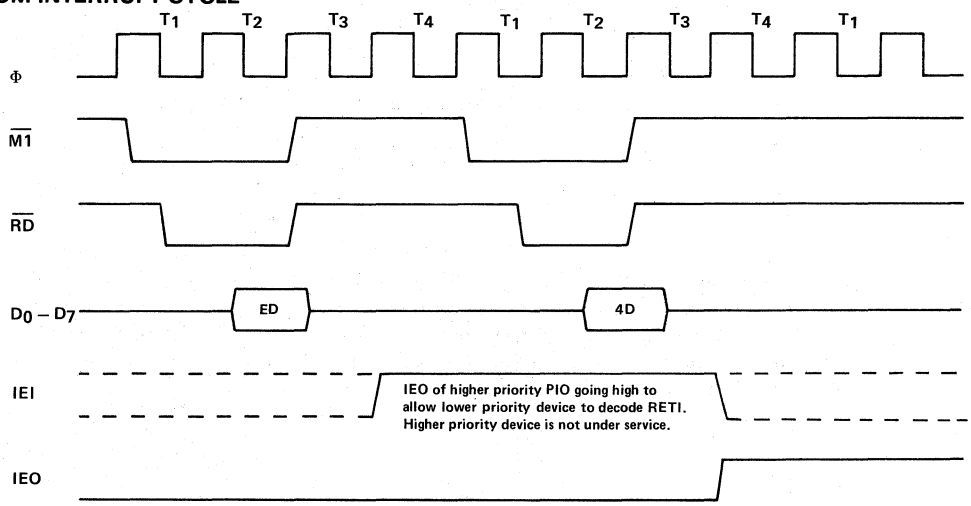
If an interrupt requested by the PIO is acknowledged, the requesting port is 'under service'. IEO of this port will remain low until a return from interrupt instruction (RETI) is executed while IEI of the port is high. If an interrupt request is not acknowledged, IEO will be forced high for one $\overline{M1}$ cycle after the PIO decodes the opcode 'ED'. This action guarantees that the two byte RETI instruction is decoded by the proper PIO port. See Figure 6.0-2.

III

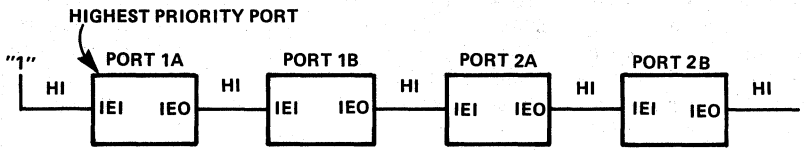
INTERRUPT ACKNOWLEDGE TIMING
Figure 6.0-1



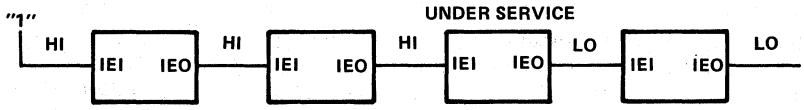
RETURN FROM INTERRUPT CYCLE
Figure 6.0-2



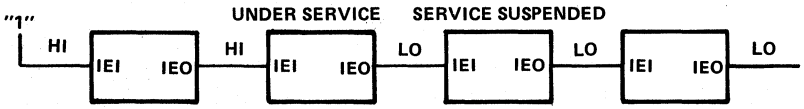
DAISY CHAIN INTERRUPT SERVICING
Figure 6.0-3



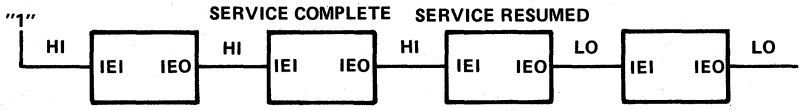
1. PRIORITY INTERRUPT DAISY CHAIN BEFORE ANY INTERRUPT OCCURS.



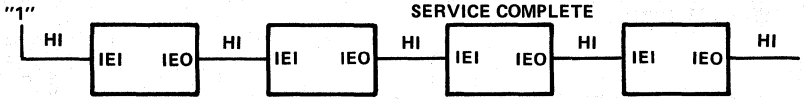
2. PORT 2A REQUESTS AN INTERRUPT AND IS ACKNOWLEDGED.



3. PORT 1B INTERRUPTS, SUSPENDS SERVICING OF PORT 2A.



4. PORT 1B SERVICE ROUTINE COMPLETE, "RETI" ISSUED, PORT 2A SERVICE RESUMED.



5. SECOND "RETI" INSTRUCTION ISSUED ON COMPLETION OF PORT 2A SERVICE ROUTINE.

Figure 6.0-3 illustrates a typical nested interrupt sequence that could occur with four ports connected in the daisy chain. In this sequence Port 2A requests and is granted an interrupt. While this port is being serviced, a higher priority port (1B) requests and is granted an interrupt. The service routine for the higher priority port is completed and RETI instruction is executed to indicate to the port that its routine is complete. At this time the service routine of the lower priority port is completed.

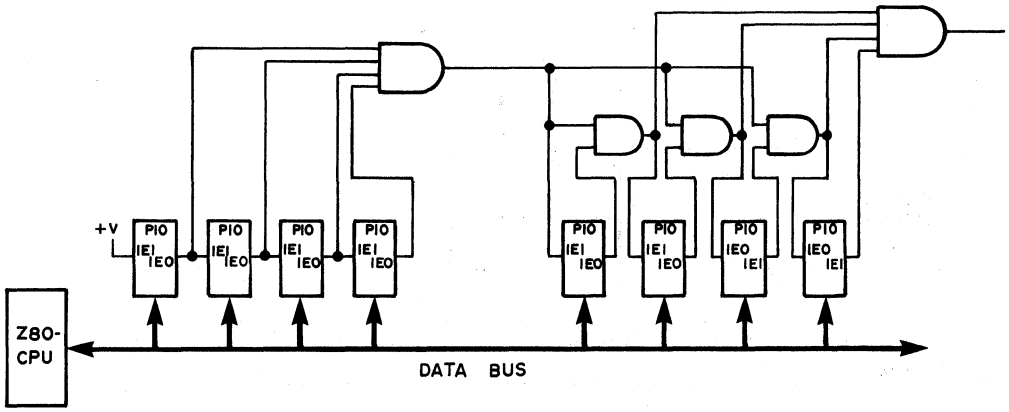
7.0 APPLICATIONS

7.1 EXTENDING THE INTERRUPT DAISY CHAIN

Without any external logic, a maximum of four Z80-PIO devices may be daisy chained into a priority interrupt structure. This limitation is required so that the interrupt enable status (IEO) ripples through the entire chain between the beginning of $\overline{M1}$, and the beginning of $\overline{IORQ}$ during an interrupt acknowledge cycle. Since the interrupt enable status cannot change during $\overline{M1}$, the vector address returned to the CPU is assured to be from the highest priority device which requested an interrupt.

If more than four PIO devices must be accommodated, a "look-ahead" structure may be used as shown in Figure 7.0-1. With this technique more than thirty PIO's may be chained together using standard TTL logic.

A METHOD OF EXTENDING THE INTERRUPT DAISY CHAIN
Figure 7.0-1



7.2 I/O DEVICE INTERFACE

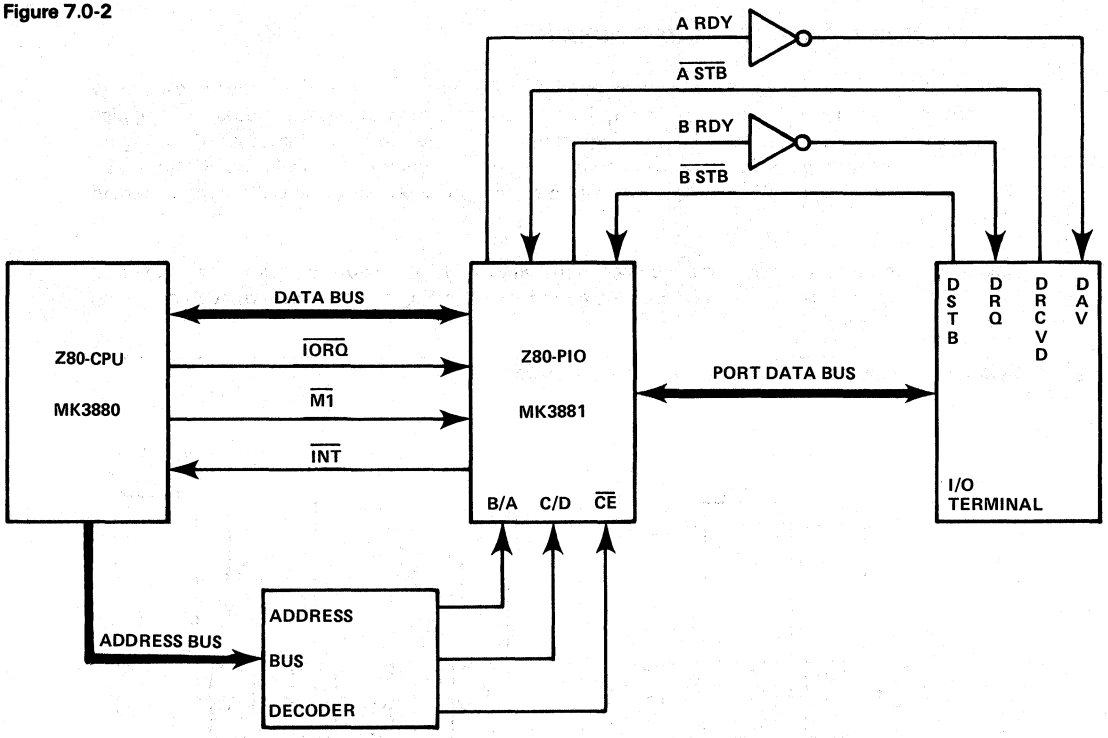
In this example, the Z80-PIO is connected to an I/O terminal device which communicates over an 8-bit parallel bidirectional data bus as illustrated in Figure 7.0-2. Mode 2 operation (bidirectional) is selected by sending the following control word to Port A:

EXAMPLE I/O INTERFACE
Figure 7.0-2

D7	D6	D5	D4	D3	D2	D1	D0
1	0	X	X	1	1	1	1

MODE CONTROL

EXAMPLE I/O INTERFACE
Figure 7.0-2



Next, the proper interrupt vector is loaded (refer to CPU Manual for details on the operation of the interrupt).

V7	V6	V5	V4	V3	V2	V1	0

Interrupts are then enabled by the rising edge of the first $\overline{M1}$ after the interrupt mode word is set unless that $\overline{M1}$ defines an interrupt acknowledge cycle. If a mask follows the interrupt mode word, interrupts are enabled by the rising edge of the first $\overline{M1}$ following the setting of the mask.

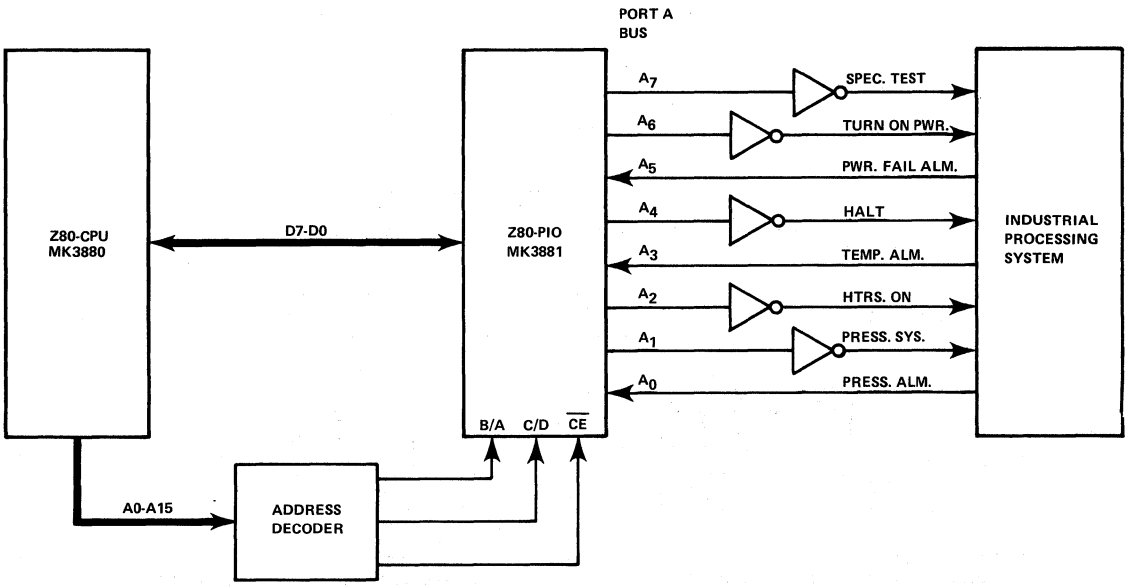
Data can now be transferrred between the peripheral and the CPU. The timing for this transfer is as described in Section 5.0

7.3 CONTROL INTERFACE

A typical control mode application is illustrated in Figure 7.0-3. Suppose an industrial process is to be monitored. The occurrence of any abnormal operating condition is to be reported to a Z80-CPU based control system. The process control and status word have the following format:

D7	D6	D5	D4	D3	D2	D1	D0
Special Test	Turn On Power	Power Failure Alarm	Halt Process-ing	Temp. Alarm	Temp Heaters On	Pressur-ize System	Pressure Alarm

CONTROL MODE APPLICATION
Figure 7.0-3



The PIO may be used as follows. First Port A is set for Mode 3 operation by writing the following control word to Port A.

D7	D6	D5	D4	D3	D2	D1	D0
1	1	X	X	1	1	1	1

Whenever Mode 3 is selected, the next control word sent to the port must be an I/O select word. In this example we wish to select port data lines A5, A3, and A0 as inputs and so the following control word is written:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	0	1	0	0	1

Next the desired interrupt vector must be loaded (refer to the CPU manual for details);

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	V0

An interrupt control word is next sent to the port:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	0	1	1	1
Enable Interrupts	OR Logic	Active High	Mask Follows	Interrupt Control			

The mask word following the interrupt mode word is:

D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	1	0	1	1	0

Selects A5, A3 and A0 to be monitored

Now, if a sensor puts a high level on line A5, A3 or A0, an interrupt request will be generated. The mask word may select any combination of inputs or outputs to cause an interrupt. For example, if the mask word above had been:

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	1	0	1	1	0

then an interrupt request would also occur if bit A7 (special Test) of the output register was set.

Assume that the following port assignments are to be used:

$E0_H$ = Port A Data
$E1_H$ = Port B Data
$E2_H$ = Port A Control
$E3_H$ = Port B Control

All port numbers are in hexadecimal notation. This particular assignment of port numbers is convenient since A_0 of the address bus can be used as the Port B/A Select and A_1 of the address bus can be used as the Control/Data Select. The Chip Enable would be the decode of CPU address bits A_7 through A_2 (111000). Note that if only a few peripheral devices are being used, a Chip Enable decode may not be required since a higher order address bit could be used directly.

8.0 PROGRAMMING SUMMARY

8.1 LOAD INTERRUPT VECTOR

V7	V6	V5	V4	V3	V2	V1	0

8.2 SET MODE

M1	M0	X	X	1	1	1	1

MODE NUMBER	M_1	M_0	MODE
0	0	0	Output
1	0	1	Input
2	1	0	Birdirectional
3	1	1	Bit Control

When selecting Mode 3, the next word to the PIO must set the I/O Register:

I/O_7	I/O_6	I/O_5	I/O_4	I/O_3	I/O_2	I/O_1	I/O_0

I/O = 1 Sets bit to Input
I/O = 0 Sets bit to Output

8.3 SET INTERRUPT CONTROL

Int Enable	AND/ OR	High/ Low	Mask Follows	0	1	1	1

USED IN MODE 3 ONLY

If the "mask follows" bit is high, the next control word written to the PIO must be the mask:

MB_7	MB_6	MB_5	MB_4	MB_3	MB_2	MB_1	MB_0

MB = 0, Monitor bit
MB = 1, Mask bit from being monitored

Also the interrupt enable flip flop of a port may be set or reset without modifying the rest of the interrupt control word by using the following command:

Int Enable	X	X	X	0	0	1	1

9.0 ELECTRICAL SPECIFICATIONS

9.1 ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... Specified operating range
Storage Temperature ..... -65°3C to +150°C
Voltage On Any Pin With Respect to Ground ..... -0.3 V to +7 V
Power Dissipation ..... 0.6 W

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

All ac parameters assume a load capacitance of 100 pF max. Timing references between two output signals assume a load difference of 50 pF max.

III

9.2 D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = 5 V ± 5% unless otherwise specified

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
V_{ILC}	Clock Input Low Voltage	-0.3	0.80	V	
V_{IHC}	Clock Input High Voltage	V_{CC} -.6	V_{CC} +.3	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -250 μA
I_{CC}	Power Supply Current		70*	mA	
I_{LI}	Input Leakage Current		±10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float		10	μA	V_{OUT} = 2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float		-10	μA	V_{OUT} = 0.4 V
I_{LD}	Data Bus Leakage Current in Input Mode		±10	μA	$0 \leq V_{IN} \leq V_{CC}$
I_{OHD}	Darlington Drive Current	-1.5		mA	V_{OH} = 1.5 V Port 8 Only

*150 mA for -4, -10, and -20 devices.

9.3 CAPACITANCE

T_A = 25°C, f = 1 MHz

SYMBOL	PARAMETER	MAX	UNIT	TEST CONDITION
C_Φ	Clock Capacitance	10	pF	Unmeasured Pins
C_{IN}	Input Capacitance	5	pF	Returned to Ground
C_{OUT}	Output Capacitance	10	pF	

9.4 A.C. CHARACTERISTICS MK3881, MK3881-10, MK3881-20, Z80-PIO

T_A = 0°C to 70°C, V_{CC} = +5 V ± 5%, unless otherwise noted

SIGNAL	SYMBOL	PARAMETER	3881 MIN	3881 MAX	3881-4 MIN	3881-4 MAX	UNIT
Φ	t_c	Clock Period	400	[1]	250	[1]	nsec
	$t_{W(\Phi H)}$	Clock Pulse Width, Clock High	170	2000	105	2000	nsec
	$t_{W(\Phi L)}$	Clock Pulse Width, Clock Low	170	2000	105	2000	nsec
	t_r, t_f	Clock Rise and Fall Times		30		30	nsec
	t_h	Any Hold Time for Specified Set-Up Time	0		0		nsec
C/D SEL $\overline{CE}$ ETC.	$t_{S\Phi(CS)}$	Control Signal Set-up Time to Rising Edge of Φ During Read or Write Cycle	280		145		nsec
D_0 - D_7	$t_{DR(D)}$	Data Output Delay from Falling Edge of RD		430		380	nsec
	$t_{S\Phi(D)}$	Data Set-up Time to Rising Edge of Φ During Write or $\overline{M1}$ Cycle	50		50		nsec
	$t_{DI(D)}$	Data Output Delay from Falling Edge of $\overline{IORQ}$ During $\overline{INTA}$ Cycle		340		250	nsec
	$t_{F(D)}$	Delay to Floating Bus (Output Buffer Disable Time)		160		110	nsec
IEI	$t_{S(IEI)}$	IEI Set-Up Time to Falling Edge of $\overline{IORQ}$ During $\overline{INTA}$ cycle	140		140		nsec
IEO	$t_{DH(IO)}$	IEO Delay Time from Rising Edge of IEI		210		160	nsec
	$t_{DL(IO)}$	IEO Delay Time from Falling Edge of IEI		190		130	nsec
	$t_{DM(IO)}$	IEO Delay from Falling Edge of $\overline{M1}$ (Interrupt Occurring Just Prior to $\overline{M1}$) See Note A.		300		190	nsec
$\overline{IORQ}$	$t_{S\Phi(IR)}$	$\overline{IORQ}$ Set-Up Time to Rising Edge of Φ During Read or Write Cycle	250		115		nsec
$\overline{M1}$	$t_{S\Phi(\overline{M1})}$	$\overline{M1}$ Set-Up Time to Rising Edge of Φ During $\overline{INTA}$ or $\overline{M1}$ Cycle. See Note B.	210		90		nsec
$\overline{RD}$	$t_{S\Phi(RD)}$	$\overline{RD}$ Set-Up Time to Rising Edge of Φ During Read or $\overline{M1}$ Cycle	240		115		nsec
A_0 - A_7 B_0 - B_7	$t_{S(PD)}$	Port Data Set-Up Time to Rising Edge of $\overline{STROBE}$ (Mode 1)	260		230		nsec
	$t_{DS(PD)}$	Port Data Output Delay from Falling Edge of $\overline{STROBE}$ (Mode 2)		230		210	nsec
	$t_{F(PD)}$	Delay to Floating Port Data Bus from Rising Edge of $\overline{STROBE}$ (Mode 2)		200		180	nsec
	$t_{DI(PD)}$	Port Data Stable from Rising Edge of $\overline{IORQ}$ During $\overline{WR}$ Cycle (Mode 0)		200		180	nsec
$\overline{ASTB}$ $\overline{BSTB}$	$t_{W(ST)}$	Pulse Width, $\overline{STROBE}$	150	[4]	150	[4]	nsec nsec
$\overline{INT}$	$t_{D(IT)}$	$\overline{INT}$ Delay Time from Rising Edge of $\overline{STROBE}$		490		440	nsec
	$t_{D(IT3)}$	$\overline{INT}$ Delay Time from Data Match During Mode 3 Operation		420		380	nsec
ARDY BRDY	$t_{DH(RY)}$	Ready Response Time from Rising Edge of $\overline{IORQ}$		t_c+ 460		t_c+ 410	nsec
	$t_{DL(RY)}$	Ready Response Time from Rising Edge of $\overline{STROBE}$		t_c+ 400		t_c+ 360	nsec

A. 2.5 $t_c > (N-2)t_{DL(IO)} + t_{DM(IO)} + t_{S(IEI)}$ + TTL Buffer Delay, if any.

B. $\overline{M1}$ must be active for a minimum of 2 clock periods to reset the PIO.

[1] $t_c = t_{W(\Phi H)} + t_{W(\Phi L)} + t_r + t_f$

[2] Increase $t_{DR(D)}$ by 10 nsec for each 50 pF increase in loading up to 200 pF max.

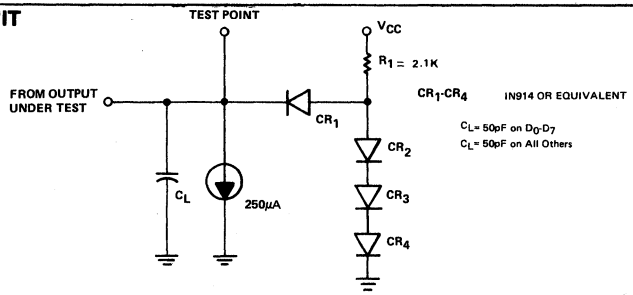
[3] Increase $t_{DI(D)}$ by 10 nsec for each 50 pF increase in loading up to 200 pF max.

[4] For Mode 2: $t_{W(ST)} > t_{S(PD)}$

[5] Increase these values by 2 nsec for each 10 pF increase in loading up to 100 pF max.

OUTPUT LOAD CIRCUIT

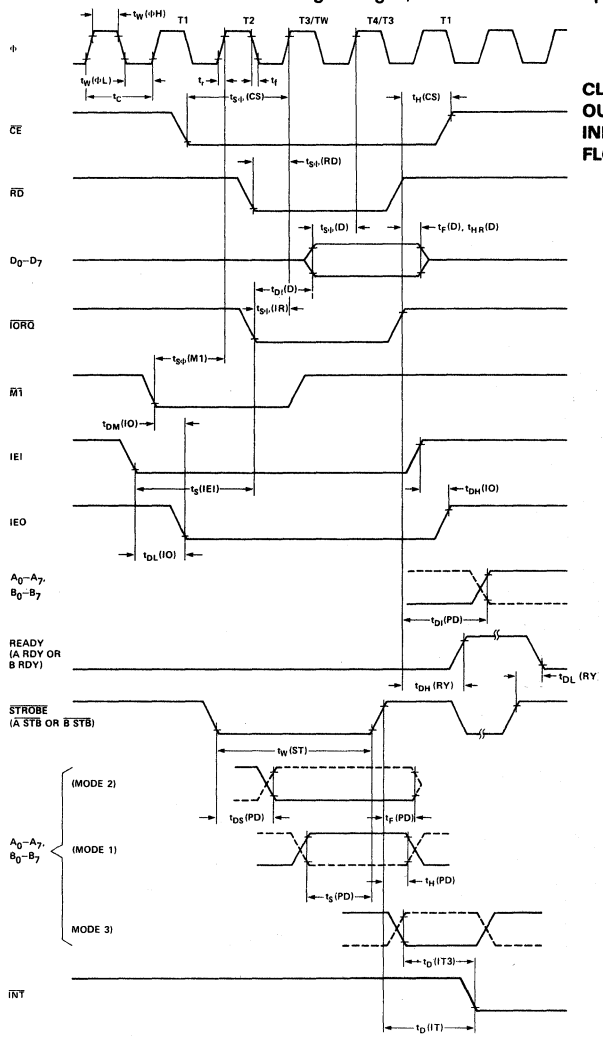
Figure 9.4-1



III

9.5 TIMING DIAGRAM

Timing measurements are made at the following voltages, unless otherwise specified:



	"1"		"0"
CLOCK	4.2 V		0.8 V
OUTPUT	2.0 V		0.8 V
INPUT	2.0 V		0.8 V
FLOAT	Δ V	=	0.5 V

10.0 ORDERING INFORMATION

PART NO.	DESIGNATOR	PACKAGE TYPE	MAX CLOCK FREQUENCY	TEMPERATURE RANGE
MK3881N	Z80-PIO	Plastic	2.5 MHz	0° to 70°C
MK3881P	Z80-PIO	Ceramic	2.5 MHz	
MK3881N-4	Z80A-PIO	Plastic	4.0 Mhz	
MK3881P-4	Z80A-PIO	Ceramic	4.0 MHz	
MK3881P-10	Z80-PIO	Ceramic	4.0 MHz	–40° to +85°C

MOSTEK®

Z80 MICROCOMPUTER DEVICES

Technical Manual

III

MK3882 COUNTER TIMER CIRCUIT

TABLE OF CONTENTS

1.0 INTRODUCTION

The Z80-Counter Timer Circuit (CTC) is a programmable component with four independent channels that provide counting and timing functions for microcomputer systems based on the Z80-CPU. The CPU can configure the CTC channels to operate under various modes and conditions as required to interface with a wide range of devices. In most applications, little or no external logic is required. The Z80-CTC utilizes N-channel silicon gate depletion load technology and is packaged in a 28-pin DIP. The Z80-CTC requires only a single 5 volt supply and a one-phase 5 volt clock. Major features of the Z80-CTC include:

- All inputs and outputs fully TTL compatible.
- Each channel may be selected to operate in either Counter Mode or Timer Mode.
- Used in either mode, a CPU-readable Down Counter indicates number of counts-to-go until zero.
- A Time Constant Register can automatically reload the Down Counter at Count Zero in Counter and Timer Mode.
- Selectable positive or negative trigger initiates time operation in Timer Mode. The same input is monitored for event counts in Counter Mode.
- Three channels have Zero Count/Timeout outputs capable of driving Darlington transistors.
- Interrupts may be programmed to occur on the zero count condition in any channel.
- Daisy chain priority interrupt logic included to provide for automatic interrupt vectoring without external logic.

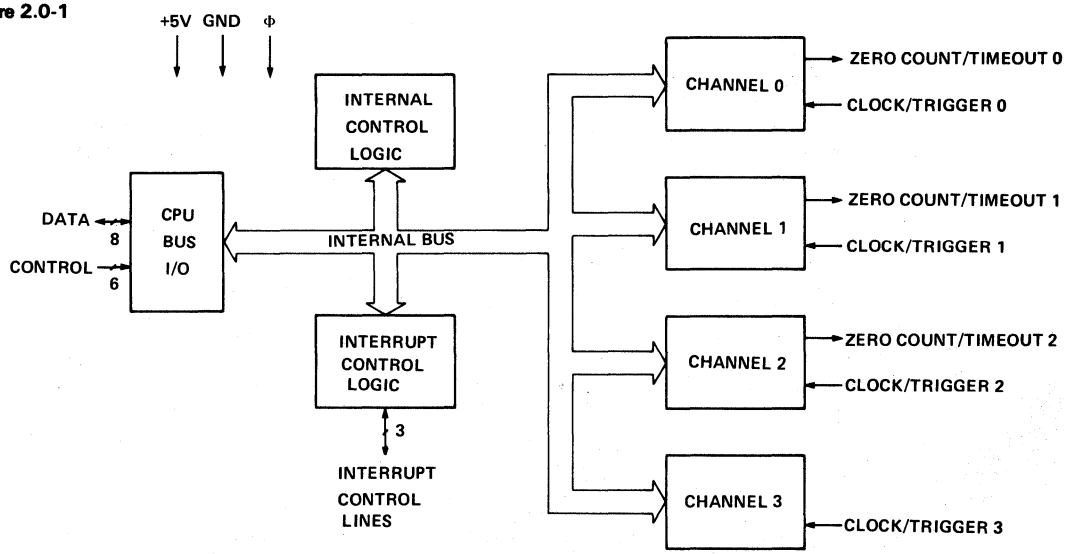
2.0 CTC ARCHITECTURE

2.1 OVERVIEW

A block diagram of the Z80-CTC is shown in Figure 2.0-1. The internal instruction of the Z80-CTC consists of a Z80-CPU bus interface, Internal Control Logic, four sets of Counter/Timer Channel Logic, and Interrupt Control Logic. The four independent counter/timer channels are identified by sequential numbers from 0 to 3. The CTC has the capability of generating a unique interrupt vector for each separate channel (for automatic vectoring to an interrupt service routine). The 4 channels can be connected into four contiguous slots in the standard Z80 priority chain with channel number 0 having the highest priority. The CPU bus interface logic allows the CTC device to interface directly to the CPU with no other external logic. However, port address decoders and/or line buffers may be required for large systems.

Z80-CTC BLOCK DIAGRAM

Figure 2.0-1

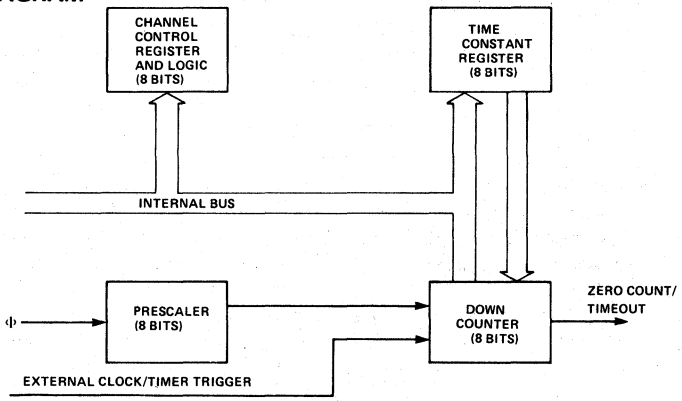


2.2 STRUCTURE OF CHANNEL LOGIC

The structure of one of the four sets of Counter/Timer Channel Logic is shown in Figure 2.0-2. This logic is composed of 2 registers, 2 counters and control logic. The registers are an 8-bit Time Constant Register and an 8-bit Channel Control Register. The counters are an 8-bit CPU-readable Down Counter and an 8-bit Prescaler.

CHANNEL BLOCK DIAGRAM

Figure 2.0-2



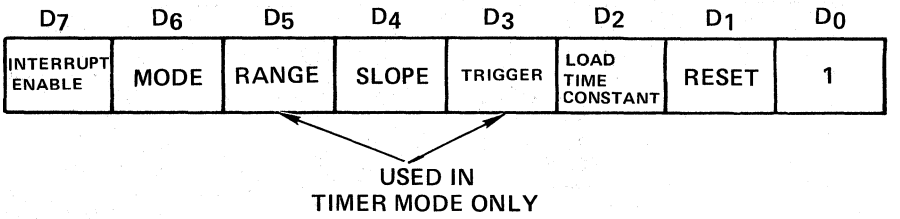
2.2.1 THE CHANNEL CONTROL REGISTER AND LOGIC

The Channel Control Register (8-bit) and Logic is written to by the CPU to select the modes and parameters of the channel. Within the entire CTC device there are four such registers, corresponding to the four Counter/Timer Channels. Which of the four is being written to depends on the encoding of two channel select input pins: CS0 and CS1 (usually attached to A0 and A1 of the CPU address bus). This is illustrated in the truth table below:

	CS1	CS0
Ch0	0	0
Ch1	0	1
Ch2	1	0
Ch3	1	1

In the control word written to program each Channel Control Register, bit 0 is always set, and the other 7 bits are programmed to select alternatives on the channel's operating modes and parameters, as shown in the diagram below. (For a more complete discussion see section 4.0: "CTC Operating Modes" and section 5.0: "CTC Programming.")

CHANNEL CONTROL REGISTER
Figure 2.0-3



2.2.2 THE PRESCALER

Used in the Timer Mode only, the Prescaler is an 8-bit device which can be programmed by the CPU via the Channel Control Register to divide its input, the System Clock (Φ), by 16 or 256. The output of the Prescaler is then fed as an input to clock the Down Counter, which initially, and every time it clocks down to zero, is reloaded automatically with the contents of the Time Constant Register. In effect this again divides the System Clock by an additional factor of the time constant. Every time the Down Counter counts down to zero, its output, Zero Count/Timeout (ZC/TO), is pulsed high.

2.2.3 THE TIME CONSTANT REGISTER

The Time Constant Register is an 8-bit register, used in both Counter Mode and Timer Mode, programmed by the CPU just after the Channel Control Word with an integer time constant value of 1 through 256. This register loads the programmed value into the Down Counter when the CTC is first initialized and reloads the same value into the Down Counter automatically whenever it counts down thereafter to zero. If a new time constant is loaded into the Time Constant Register while a channel is counting or timing, the present down count will be completed before the new time constant is loaded into the Down Counter. (For details of how a time constant is written to a CTC channel, see section 5.0: "CTC Programming.")

2.2.4 THE DOWN COUNTER

The Down Counter is an 8-bit register used in both Counter Mode and Timer Mode loaded initially, and later when it counts down to zero, by the Time Constant Register. The Down Counter is decremented by each external clock edge in the Counter Mode, or in the Timer Mode, by the clock output of the Prescaler. At any time, by performing a simple I/O Read at the port address assigned to the selected CTC channel, the CPU can access the contents of this register and obtain the number of counts-to-zero. Any CTC channel may be programmed to generate an interrupt request sequence each time the zero count is reached.

In channels 0, 1, and 2, when the zero count condition is reached, a signal pulse appears at the corresponding ZC/TO pin. Owing to package pin limitations, however, channel 3 does not have this pin and so may be used only in applications where this output pulse is not required.

2.3 INTERRUPT CONTROL LOGIC

III

The Interrupt Control logic ensures that the CTC acts in accordance with Z80 system interrupt protocol for nested priority interrupting and return from interrupt. The priority of any system device is determined by its physical location in a daisy chain configuration. Two signal lines (IEI and IEO) are provided in CTC devices to form this system daisy chain. The device closest to the CPU has the highest priority; within the CTC, interrupt priority is predetermined by channel number, with channel 0 having highest priority down to channel 3 which has the lowest priority. The purpose of a CTC-generated interrupt, as with any other peripheral device, is to force the CPU to execute an interrupt service routine. According to Z80 system interrupt protocol, lower priority devices or channels may not interrupt higher priority devices or channels that have already interrupted and have not had their interrupt service routines completed. However, high priority devices or channels may interrupt the servicing of lower priority devices or channels.

A CTC channel may be programmed to request an interrupt every time its Down Counter reaches a count of zero. (To utilize this feature requires that the CPU be programmed for interrupt mode 2.) Some time after the interrupt request, the CPU will send out an interrupt acknowledge, and the CTC's Interrupt Control Logic will determine the highest-priority channel which is requesting an interrupt within the CTC device. Then if the CTC's IEI Input is active, indicating that it has priority within the system daisy chain, it will place an 8-bit Interrupt Vector on the system data bus. The high-order 5 bits of this vector will have been written to the CTC earlier as part of the CTC initial programming process; the next two bits will be provided by the CTC's Interrupt Control Logic as a binary code corresponding to the highest-priority channel requesting an interrupt; finally the low-order bit of the vector will always be zero according to a convention described below.

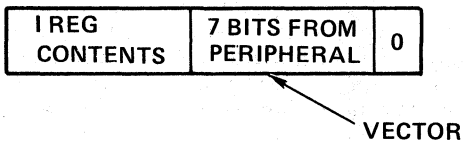
INTERRUPT VECTOR
Figure 2.0-4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	X	X	0

D_2	D_1	
0	0	CHANNEL 0
0	1	CHANNEL 1
1	0	CHANNEL 2
1	1	CHANNEL 3

This interrupt vector is used to form a pointer to a location in memory where the address of the interrupt service routine is stored in a table. The vector represents the least significant 8 bits, while the CPU reads the contents of the I register to provide the most significant 8-bits of the 16-bit pointer. The address in memory pointed to will contain the low-order byte, and the next highest address will contain the high-order byte of an address which in turn contains the first opcode of the interrupt service routine. Thus in mode 2, a single 8-bit vector stored in an interrupting CTC can result in an indirect call to any memory location.

Z80 16-BIT POINTER (INTERRUPT STARTING ADDRESS)
Figure 2.0-5



2.3 INTERRUPT CONTROL LOGIC (Cont'd)

There is a Z80 system convention that all addresses in the interrupt service routine table should have their low-order byte in an even location in memory, and their high-order byte in the next highest location in memory, which will always be odd so that the least significant bit of any interrupt vector will always be even. Hence the least significant bit of any interrupt vector will always be zero.

The RETI instruction is used at the end of any interrupt service routine to initialize the daisy chain enable line IEO for proper control of nested priority interrupt handling. The CTC monitors the system data bus and decodes this instruction when it occurs. Thus the CTC channel control logic will know when the CPU has completed servicing an interrupt, without any further communication with the CPU being necessary.

3.0 CTC PIN DESCRIPTION

A diagram of the Z80-CTC pin configuration is shown in Figure 3.0-1. This section describes the function of each pin.

D7 - D0
Z80-CPU Data Bus (bi-directional, tri-state)

This bus is used to transfer all data and command words between the Z80-CPU and the Z80-CTC. There are 8 bits on this bus, of which D0 is the least significant.

CS1 - CS0
Channel Select (input, active high)

These pins form a 2-bit binary address code for selecting one of the four independent CTC channels for an I/O Write or Read (See truth table below.)

III

	CS1	CS0
Ch0	0	0
Ch1	0	1
Ch2	1	0
Ch3	1	1

$\overline{CE}$
Chip Enable (input, active low)

A low level on this pin enables the CTC to accept control words, Interrupt Vectors, or time constant data words from the Z80 Data Bus during an I/O Write cycle, or to transmit the contents of the Down Counter to the CPU during an I/O Read cycle. In most applications this signal is decoded from the 8 least significant bits of the address bus for any of the four I/O port addresses that are mapped to the four Counter/Timer Channels.

Clock (Φ)
System Clock (input)

This single-phase clock is used by the CTC to synchronize certain signals internally.

$\overline{M1}$
Machine Cycle One Signal from CPU (input, active low)

When $\overline{M1}$ is active and the $\overline{RD}$ signal is active, the CPU is fetching an instruction from memory. When $\overline{M1}$ is active and the $\overline{IORQ}$ signal is active, the CPU is acknowledging an interrupt, alerting the CTC to place an Interrupt Vector on the Z80 Data Bus if it has daisy chain priority and one of its channels has requested an interrupt.

$\overline{IORQ}$
Input/Output Request from CPU (input, active low)

The $\overline{IORQ}$ signal is used in conjunction with the $\overline{CE}$ and $\overline{RD}$ signals to transfer data and Channel Control Words between the Z80-CPU and the CTC. During a CTC Write Cycle, $\overline{IORQ}$ and $\overline{CE}$ must be true and $\overline{RD}$ false. The CTC does not receive a specific write signal, instead generating its own internally from the inverse of a valid $\overline{RD}$ signal. In a CTC Read Cycle, $\overline{IORQ}$, $\overline{CE}$ and $\overline{RD}$ must be active to place the contents of the Down Counter on the Z80 Data Bus. If $\overline{IORQ}$ and $\overline{M1}$ are both true, the CPU is acknowledging an interrupt request, and the highest-priority interrupting channel will place its Interrupt Vector on the Z80 Data Bus.

3.0 CTC PIN DESCRIPTION (CONT'D)

$\overline{RD}$
Read Cycle Status from the CPU (input, active low)

The $\overline{RD}$ signal is used in conjunction with the $\overline{IORQ}$ and $\overline{CE}$ signals to transfer data and Channel Control Words between the Z80-CPU and the CTC. During a CTC Write Cycle, $\overline{IORQ}$ and $\overline{CE}$ must be true and $\overline{RD}$ false. The CTC does not receive a specific write signal, instead generating its own internally from the inverse of a valid $\overline{RD}$ signal. In a CTC Read Cycle, $\overline{IORQ}$, $\overline{CE}$ and $\overline{RD}$ must be active to place the contents of the Down Counter on the Z80 Data Bus.

IEI
Interrupt Enable In (input, active high)

This signal is used to help form a system-wide interrupt daisy chain which establishes priorities when more than one peripheral device in the system has interrupting capability. A high level on this pin indicates that no other interrupting devices of higher priority in the daisy chain are being serviced by the Z80-CPU.

IEO
Interrupt Enable Out (output, active high)

The IEO signal, in conjunction with IEI, is used to form a system-wide interrupt priority daisy chain. IEO is high only if IEI is high and the CPU is not servicing an interrupt from any CTC channel. Thus this signal blocks lower priority devices from interrupting while a higher priority interrupting device is being serviced by the CPU.

$\overline{INT}$
Interrupt Request (output, open drain, active low)

This signal goes true when any CTC channel which has been programmed to enable interrupts has a zero-count condition in its Down Counter.

$\overline{RESET}$
Reset (input, active low)

This signal stops all channels from counting and resets channel interrupt enable bits in all control registers, thereby disabling CTC-generated interrupts. The ZC/TO and $\overline{INT}$ outputs go to their inactive states, IEO reflects IEI, and the CTC's data bus output drivers go to the high impedance state.

CLK/TRG3—CLK/TRG0
External Clock/Timer Trigger (input, user-selectable active high or low)

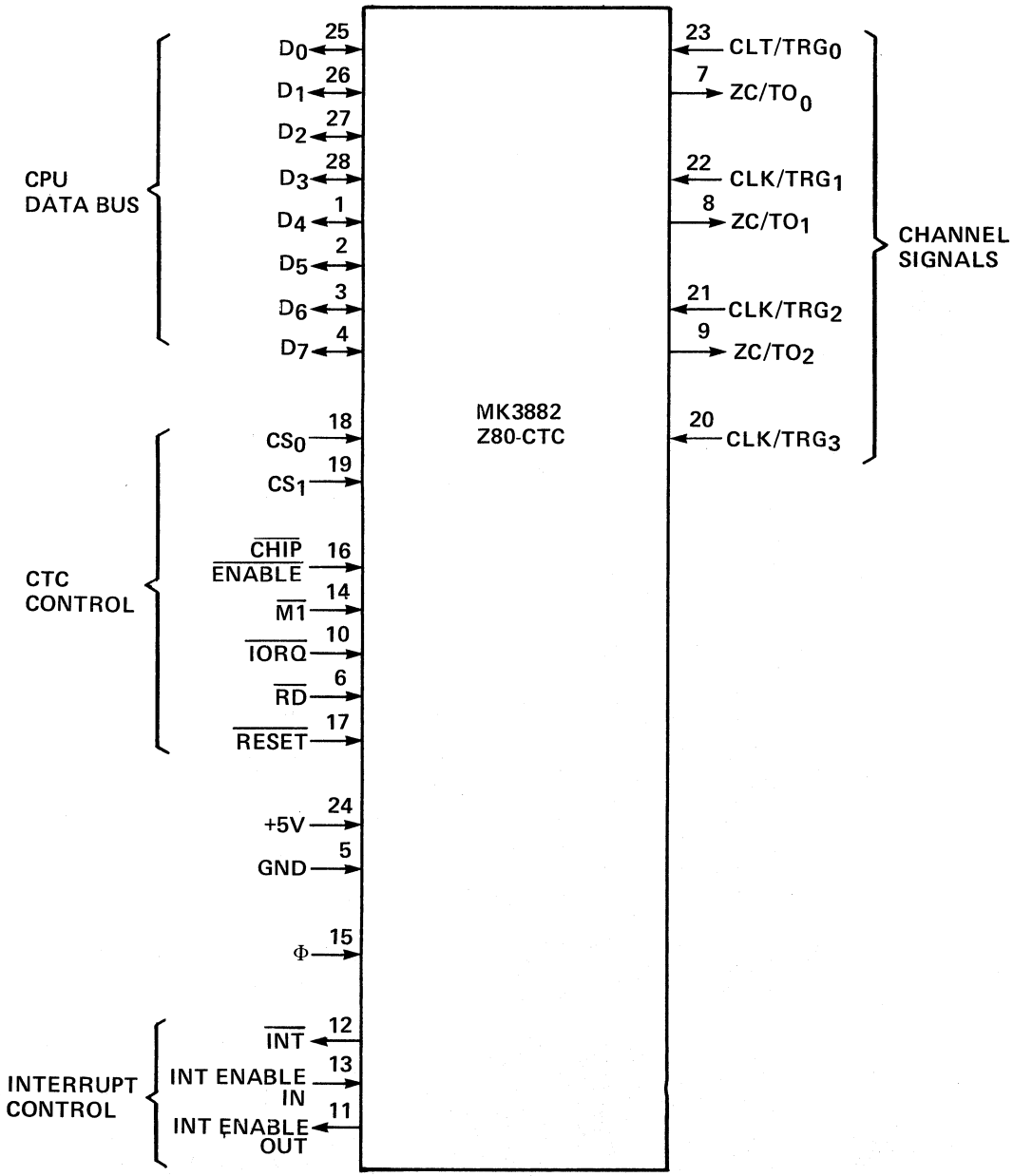
There are CLK/TRG pins, corresponding to the four independent CTC channels. In the Counter Mode, every active edge on this pin decrements the Down Counter. In the Timer Mode, an active edge on this pin initiates the timing function. The user may select the active edge to be either rising or falling.

ZC/TO2—ZC/TO0
Zero Count/Timeout (output, active high)

There are three ZC/TO pins, corresponding to CTC channels 2 through 0. (Due to package pin limitations channel 3 has no ZC/TO pin.) In either Counter Mode or Timer Mode, when the Down Counter decrements to zero, an active high going pulse appears at this pin.

Z80-CTC PIN CONFIGURATION

Figure 3.0-1



4.0 CTC OPERATING MODES

At power-on, the Z80-CTC state is undefined. Asserting RESET puts the CTC in a known state. Before any channel can begin counting or timing, a Channel Control Word and a time constant data word must be written to the appropriate registers of that channel. Further, if any channel has been programmed to enable interrupts, an Interrupt Vector word must be written to the CTC's Interrupt Control Logic. (For further details, refer to section 5.0: "CTC Programming.") When the CPU has written all of these words to the CTC, all active channels will be programmed for immediate operation in either the Counter Mode or the Timer Mode.

4.1 CTC COUNTER MODE

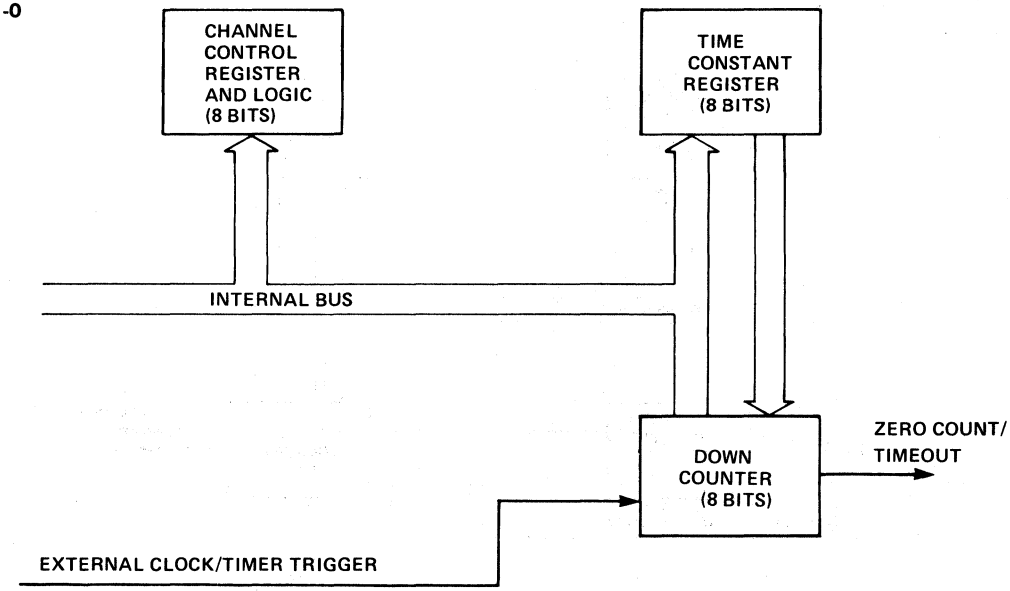
In this mode the CTC counts edges of the CLK/TRG input. The Counter Mode is programmed for a channel when its Channel Control Word is written with bit 6 set. The Channel's External Clock (CLK/TRG) input is monitored for a series of triggering edges; after each, in synchronization with the next rising edge of Φ (the System Clock), the Down Counter (which was initialized with the time constant data word at the start of any sequence of down-counting) is decremented. Although there is no set-up time requirement between the triggering edge of the External Clock and the rising edge of Φ, (Clock), the Down Counter will not be decremented until the following Φ pulse. (See the parameter ts(CK) in section 8.3: "A.C. Characteristics.") A channel's External Clock input is pre-programmed by bit 4 of the Channel Control Word to trigger the decrementing sequence with either a high or a low going edge.

In any of Channels 0, 1, or 2, when the Down Counter is successively decremented from the original time constant until finally it reaches zero, the Zero Count (ZC/TO) output pin for that channel will be pulsed active (high). (However, due to package pin limitations, channel 3 does not have this pin and so may only be used in applications where this output pulse is not required.) Further, if the channel has been so pre-programmed by bit 7 of the Channel Control Word, an interrupt request sequence will be generated. (For more details, see section 7.0: "CTC Interrupt Servicing.")

As the above sequence is proceeding, the zero count condition also results in the automatic reload of the Down Counter with the original time constant data word in the Time Constant Register. There is no interruption in the sequence of continued down-counting. If the Time Constant Register is written to with a new time constant data word while the Down Counter is decrementing, the present count will be completed before the new time constant will be loaded into the Down Counter.

CHANNEL - COUNTER MODE

Figure 4.1-0



4.2 CTC TIMER MODE

In this mode the CTC generates timing intervals that are an integer value of the system clock period. The Timer Mode is programmed for a channel when its Channel Control Word is written with bit 6 reset. The channel then may be used to measure intervals of time based on the System Clock period. The System Clock is fed through two successive counters, the Prescaler and the Down Counter. Depending on the pre-programmed bit 5 in the Channel Control Word, the Prescaler divides the System Clock by a factor of either 16 or 256. The output of the Prescaler is then used as a clock to decrement the Down Counter, which may be pre-programmed with any time constant integer between 1 and 256. As in the Counter Mode, the time constant is automatically reloaded into the Down Counter at each zero-count condition, and counting continues. Also at zero-count, the channel's Time Out (ZC/TO) output (which is the output of the Down Counter) is pulsed, resulting in a uniform pulse train of precise period given by the product.

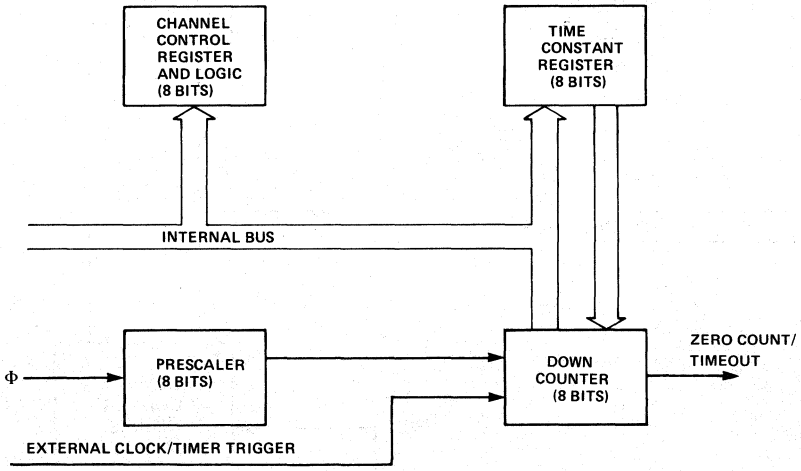
$$t_c * P * TC$$

where tc is the System Clock period, P is the Prescaler factor of 16 or 256 and TC is the pre-programmed time constant.

Bit 3 of the Channel Control Word is pre-programmed to select whether timing will be automatically initiated, or whether it will be initiated with a triggering edge at the channel's Time Trigger (CLK/TRG) input. If bit 3 is reset, the timer automatically begins operation at the start of the CPU cycle following the I/O Write machine cycle that loads the time constant data word to the channel. If bit 3 is set, the timer begins operation on the second succeeding rising edge of Φ after the Timer Trigger edge following the loading of the time constant data word. If no time constant data word is to follow, then the timer begins operation on the second succeeding rising edge of Φ after the Timer Trigger edge following the control word write cycle. Bit 4 of the Channel Control Word is pre-programmed to select whether the Timer Trigger will be sensitive to a rising or falling edge. However, there is no set-up requirement between the active edge of the Timer Trigger and the next rising edge of Φ. If the Timer Trigger edge occurs closer than a specified minimum set-up time to the rising edge of Φ, the Down Counter will not begin decrementing until the following rising edge of Φ. (See parameter ts(TR) in section 8.3: "A.C. Characteristics".)

If bit 7 in the Channel Control Word is set, the zero-count condition in the Down Counter, besides causing a pulse at the channel's Time Out pin, will be used to initiate an interrupt request sequence, (For more details, see section 7.0: "CTC Interrupt Servicing.")

CHANNEL - TIMER MODE
Figure 4.2-0



5.0 CTC PROGRAMMING

Before a Z80-CTC channel can begin counting or timing operations, a Channel Control Word and a Time Constant data word must be written to it by the CPU. These words will be stored in the Channel Control Register and the Time Constant Register of that channel. In addition, if any of the four channels have been programmed with bit 7 of their Channel Control Words to enable interrupts, an Interrupt Vector must be written to the appropriate register in the CTC. Due to automatic features in the Interrupt Control Logic, one pre-programmed Interrupt Vector suffices for all four channels.

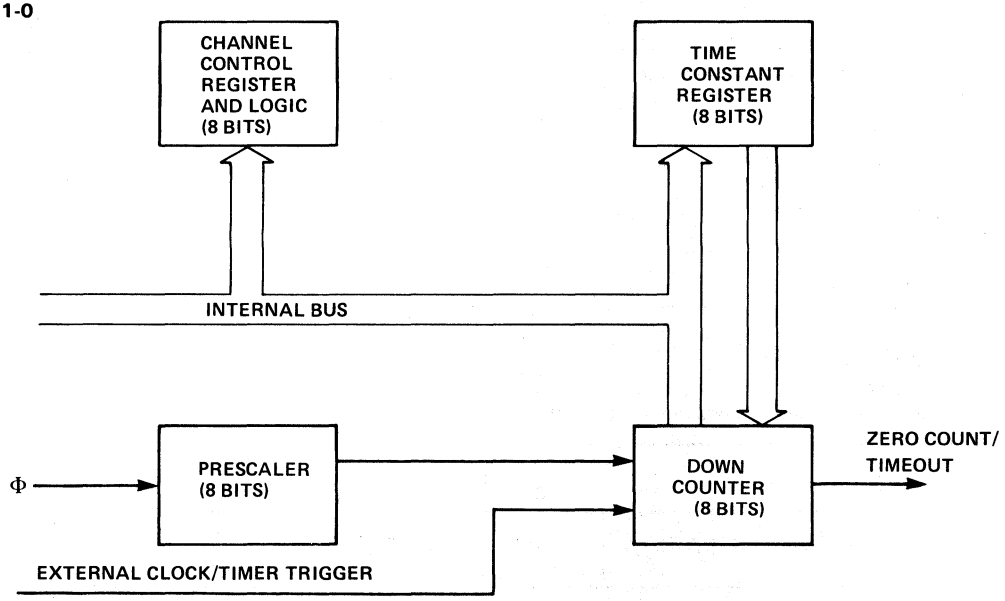
5.1 LOADING THE CHANNEL CONTROL REGISTER

To load a Channel Control Word, the CPU performs a normal I/O Write sequence to the port address corresponding to the desired CTC channel. Two CTC input pins, namely CS0 and CS1, are used to form a 2-bit binary address to select one of four channels within the device. (For a truth table, see section 2.2.1: "The Channel Control Register and Logic".) In many system architectures, these two input pins are connected to Address Bus lines A0 and A1, respectively, so that the four channels in a CTC device will occupy contiguous I/O port addresses. A word written to a CTC channel will be interpreted as a Channel Control Word, and when loaded into the Channel Control Register, its bit 0 is a logic 1. The other seven bits of this word select operating modes and conditions as indicated in the diagram below. Following the diagram, the meaning of each bit will be discussed in detail.

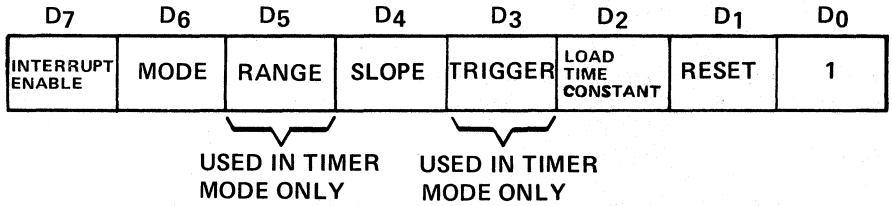
III

CHANNEL BLOCK DIAGRAM

Figure 5.1-0



CHANNEL CONTROL REGISTER



5.1 LOADING THE CHANNEL CONTROL REGISTER (CONT'D)

Bit 7 = 1

The channel is enabled to generate an interrupt request sequence every time the Down Counter reaches a zero-count condition. To set this bit to 1 in any of the four Channel Control Registers necessitates that an Interrupt Vector also be written to the CTC before operation begins. Channel interrupts may be programmed in either Counter Mode or Timer Mode. If an updated Channel Control Word is written to a channel already in operation, with bit 7 set, the interrupt enable selection will not be retroactive to a preceding zero-count condition.

Bit 7 = 0

Channel interrupts disabled. Any pending interrupt by that channel will be cleared.

Bit 6 = 1

Counter Mode selected. The Down Counter is decremented by each triggering edge of the External Clock (CLK/TRG) input. The Prescaler is not used.

Bit 6 = 0

Timer Mode selected. The Prescaler is clocked by the System Clock Φ, and the output of the Prescaler in turn clocks the Down Counter. The output of the Down Counter (the channel's ZC/TO output) is a uniform pulse train of period given by the product.

$$t_c * P * TC$$

where t_c is the period of System Clock Φ, P is the Prescaler factor of 16 or 256, and TC is the time constant data word.

Bit 5 = 1

(Defined for Timer Mode only.) Prescaler factor is 256.

Bit 5 = 0

(Defined for Timer Mode only.) Prescaler factor is 16.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
INTERRUPT ENABLE	MODE	RANGE	SLOPE	TRIGGER	LOAD TIME CONSTANT	RESET	1
		USED IN TIMER MODE ONLY		USED IN TIMER MODE ONLY			

Bit 4 = 1

TIMER MODE - positive edge trigger starts timer operation.
COUNTER MODE - positive edge decrements the down counter.

Bit 4 = 0

TIMER MODE - negative edge trigger starts timer operation.
COUNTER MODE - negative edge decrements the down counter.

5.1 LOADING THE CHANNEL CONTROL REGISTER (CONT'D)

Bit 3 = 1

Timer Mode Only - External trigger is valid for starting timer operation after rising edge of T_2 of the machine cycle following the one that loads the time constant. The Prescaler is decremented 2 clock cycles later if the setup time is met, otherwise 3 clock cycles. Once timer has been started it will free run at the rate determined by the Time Constant register.

Bit 3 = 0

Timer Mode Only - Timer begins operation on the rising edge of T_2 of the machine cycle following the one that loads the time constant.

Bit 2 = 1

The time constant data word for the Time Constant Register will be the next word written to this channel. If an updated Channel Control Word and time constant data word are written to a channel while it is already in operation, the Down Counter will continue decrementing to zero before the new time constant is loaded into it.

Bit 2 = 0

No time constant data word for the Time Constant Register should be expected to follow. To program bit 2 to this state implies that this Channel Control Word is intended to update the status of a channel already in operation, since a channel will not operate without a correctly programmed data word in the Time Constant Register, and a set bit 2 in this Channel Control Word provides the only way of writing to the Time Constant Register.

Bit 1 = 1

Reset channel. Channel stops counting or timing. This is not a stored condition. Upon writing into this bit a reset pulse discontinues current channel operation; however, none of the bits in the channel control register are changed. If both bit 2 = 1 and bit 1 = 1 the channel will resume operation upon loading a time constant.

Bit 1 = 0

Channel continues current operation.

5.2 DISABLING THE CTC'S INTERRUPT STRUCTURE

If an external Asynchronous interrupt should occur while the processor is writing the disable word to the CTC (01H), a system problem may occur. If interrupts are enabled in the processor it is possible that the Asynchronous interrupt will occur while the processor is writing the disable word to the CTC. The CTC will generate an INT and the CPU will acknowledge it; however, by this time, the CTC will have received the disable word and de-activated its interrupt structure. The result is that the CTC will not send in its interrupt vector during the interrupt acknowledge cycle because it is disabled and the CPU will fetch an erroneous vector resulting in a program fault. The cure for this problem is to disable interrupts within the CPU with the DI instruction just before the CTC is disabled and then re-enable interrupts with the EI instruction. This action causes the CPU to ignore any interrupts produced by the CTC while it is being disabled. The code sequence would be:

```
LD A, 01H
DI              ; DISABLE CPU
OUT (CTC),A     ; DISABLE CTC
EI              ; ENABLE CPU
—
—
```

5.3 LOADING THE TIME CONSTANT REGISTER

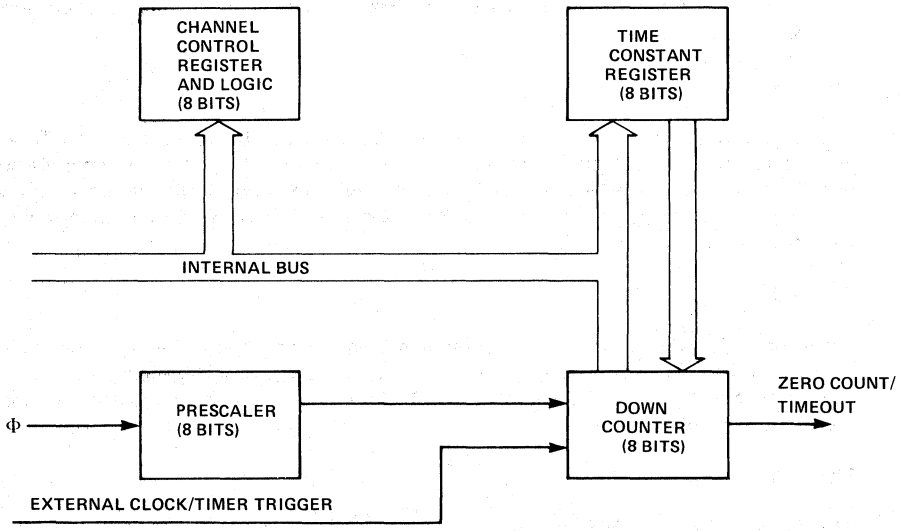
A channel may not begin operation in either Timer Mode or Counter Mode unless a time constant data word is written into the Time Constant Register by the CPU. This data word will be expected on the next I/O Write to this channel following the I/O Write of the Channel Control Word, provided that bit 2 of the Channel Control Word is set. The time constant data word may be an integer value in the range 1-256. If all eight bits in this word are zero, it is interpreted as 256. If a time constant data word is loaded to a channel already in operation, the Down Counter will continue decrementing to zero before the new time constant is loaded from the Time Constant Register to the Down Counter.

TIME CONSTANT REGISTER

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
TC_7	TC_6	TC_5	TC_4	TC_3	TC_2	TC_1	TC_0
MSB							LSB

CHANNEL BLOCK DIAGRAM

Figure 5.3-0

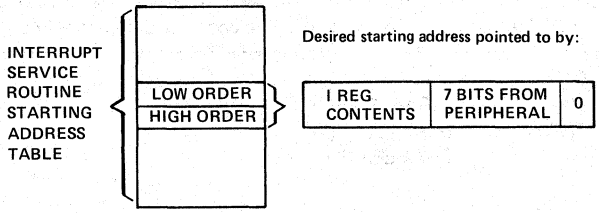


5.4 LOADING THE INTERRUPT VECTOR REGISTER

The Z80-CTC has been designed to operate with the Z80-CPU programmed for mode 2 interrupt response. Under the requirements of this mode, when a CTC channel requests an interrupt and is acknowledged, a 16-bit pointer must be formed to obtain a corresponding interrupt service routine starting address from a table in memory. The upper 8 bits of this pointer are provided by the CPU's I register, and the lower 8 bits of the pointer are provided by the CTC in the form of an Interrupt Vector unique to the particular channel that requested the interrupt. (For further details, see section 7.0: "CTC Interrupt Servicing".)

MODE 2 INTERRUPT OPERATION

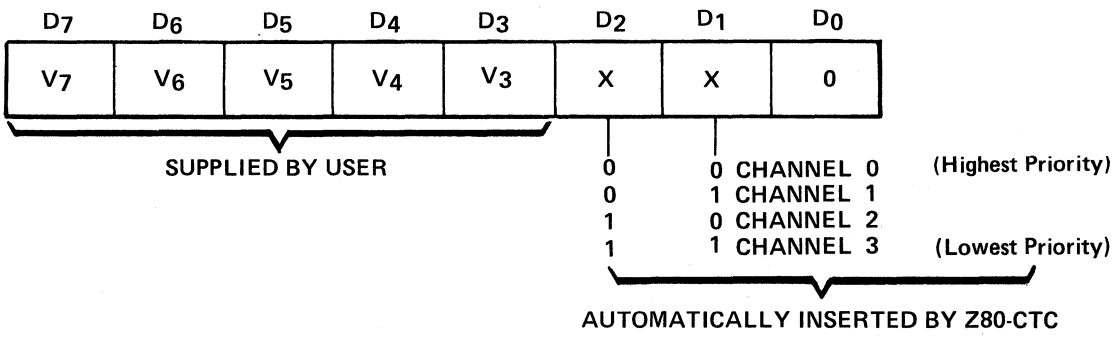
Figure 5.4-0



5.4 LOADING THE INTERRUPT VECTOR REGISTER (CONT'D)

The high order 5 bits of this Interrupt Vector must be written to the CTC in advance as part of the initial programming sequence. To do so, the CPU must write to the I/O port address corresponding to the CTC channel 0, just as it would if a Channel Control Word were being written to that channel, except that bit 0 of the word being written must contain a 0. (As explained above in section 5.1, if bit 0 of a word written to a channel were set to 1, the word would be interpreted as a Channel Control Word, so a 0 in bit 0 signals the CTC to load the incoming word into the Interrupt Vector Register.) Bits 1 and 2, however are not used when loading this vector. At the time when the interrupting channel must place the Interrupt Vector on the Z80 Data Bus, the Interrupt Control Logic of the CTC automatically supplies a binary code in bits 1 and 2 identifying which of the four CTC channels is to be serviced.

INTERRUPT VECTOR REGISTER
Figure 5.4-1



6.0 CTC TIMING

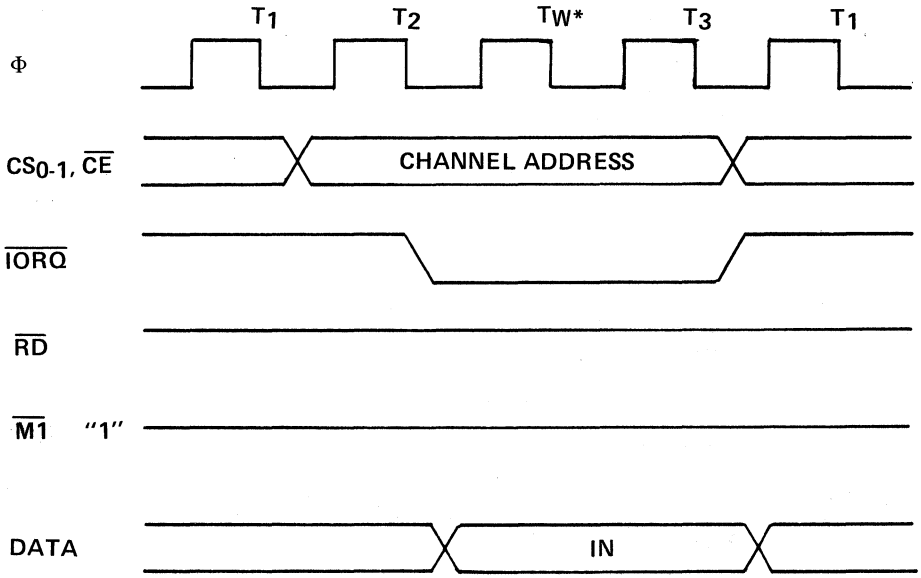
This section illustrates the timing relationships of the relevant CTC pins for the following types of operation: writing a word to the CTC, reading a word from the CTC, counting, and timing. Elsewhere in this manual may be found timing diagrams relating to interrupt servicing (section 7.0) and an A.C. Timing Diagram which quantitatively specifies the timing relationships (section 8.4).

6.1 CTC WRITE CYCLE

Figure 6.1-0 illustrates the timing associated with the CTC Write Cycle. This sequence is applicable to loading either a Channel Control Word, an Interrupt Vector, or a time constant data word.

In the sequence shown, during clock cycle T_1, the Z80-CPU prepares for the Write Cycle with a false (high) signal at CTC input pin $\overline{RD}$ (Read). Since the CTC has no separate Write signal input, it generates its own internally from the false $\overline{RD}$ input. Later, during clock cycle T_2, the Z80-CPU initiates the Write Cycle with true (low) signals at CTC input pins $\overline{IORQ}$ (I/O Request) and $\overline{CE}$ (Chip Enable). (Note: $\overline{M1}$ must be false to distinguish the cycle from an interrupt acknowledge.) Also at this time a 2-bit binary code appears at CTC inputs, CS1 and CS0 (Channel Select 1 and 0), specifying which of the four CTC channels is being written to, and the word being written appears on the Z80 Data Bus. Now everything is ready for the word to be latched into the appropriate CTC internal register in synchronization with the rising edge beginning clock cycle T_3. No additional wait states are allowed.

CTC WRITE CYCLE
Figure 6.1-0

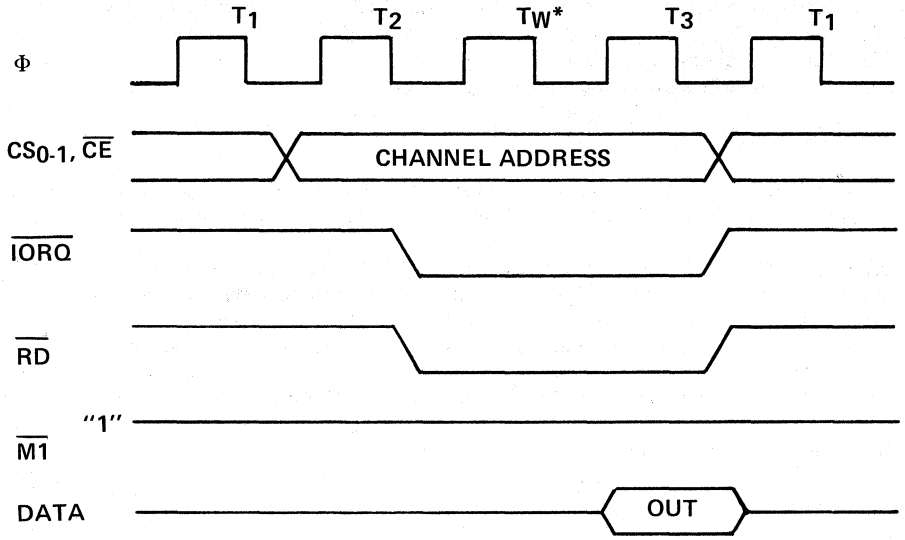


*Automatically inserted by Z80-CPU

6.2 CTC READ CYCLE

Figure 6.2-0 illustrates the timing associated with the CTC Read Cycle. This sequence is used any time the CPU reads the current contents of the Down Counter. During clock cycle T_2, the Z80-CPU initiates the Read Cycle with the true signals at input pins $\overline{RD}$ (Read), $\overline{IORQ}$ (I/O Request), and $\overline{CE}$ (Chip Enable). Also at this time a 2-bit binary code appears at CTC inputs, CS1 and CS0 (Channel Select 1 and 0), specifying which of the four CTC channels is being read from. (Note: $\overline{M1}$ must be false to distinguish the cycle from an interrupt acknowledge.) On the rising edge of the cycle T_3 the valid contents of the Down Counter as of the rising edge of cycle T_2 will be available on the Z80 Data Bus. No additional wait states are allowed.

CTC READ CYCLE
Figure 6.2-0

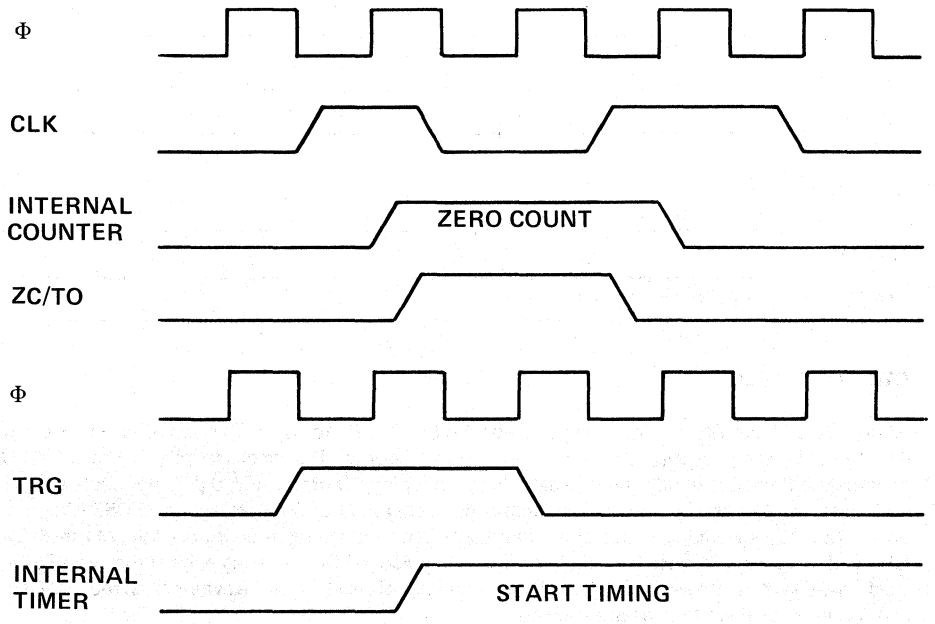


*AUTOMATICALLY INSERTED BY Z80-CPU

6.3 CTC COUNTING AND TIMING

Figure 6.3-0 illustrates the timing diagram for the CTC Counting and Timing Modes.

CTC COUNTING AND TIMING
Figure 6.3-0



6.3 CTC COUNTING AND TIMING (CONT'D)

In the Counter Mode, the edge (rising edge is active in this example) from the external hardware connected to pin CLK/TRG decrements the Down Counter in synchronization with the System Clock Φ. As specified in the A.C. Characteristics (Sections 8.3 and 8.5), this CLK/TRG pulse must have a minimum width and the minimum period must not be less than twice the system clock period. Although there is no set-up requirement between the active edge of the CLK/TRG and the rising edge of Φ, if the CLK/TRG edge occurs closer than a specified mininum time, the decrement of the Down Counter will be delayed one cycle of Φ. Immediately after the decrement of the Down Counter, 1 to 0, the ZC/TO output is pulsed true.

In the Timer Mode, a pulse trigger (user-selectable as either active high or active low) at the CLK/TRG pin enables timing function on the second succeeding rising edge of Φ. As in the Counter Mode, the triggering pulse is detected asynchronously and must have a minimum width. The timing function is initiated in synchronization with Φ, and a minimum set-up time is required between the active edge of the CLK/TRG and the next rising edge of Φ. If the CLK/TRG edge occurs closer than this, the initiation of the timer function will be delayed one cycle of Φ.

7.0 CTC INTERRUPT SERVICING

Each CTC channel may be individually programmed to request an interrupt every time its Down Counter reaches a count of zero. The purpose of a CTC-generated interrupt, as for any other peripheral device, is to force the CPU to execute an interrupt service routine. To utilize this feature the Z80-CPU must be programmed for mode 2 interrupt response. Under the requirements of this mode, when a CTC channel requests an interrupt and is acknowledged, a 16-bit pointer must be formed to obtain a corresponding interrupt service routine starting address from a table in memory. The lower 8 bits of the pointer are provided by the CTC in the form of an Interrupt Vector unique to the particular channel that requested the interrupt. (For further details, refer to Chapter 8.0 of the Z80-CPU Technical Manual.)

The CTC's Interrupt Control Logic ensures that it acts in accordance with Z80 system interrupt protocol for nested priority interrupt and proper return from interrupt. The priority of any system device is determined by its physical location in a daisy chain configuration. Two signal lines (IEI and IEO) are provided in the CTC and all Z80 peripheral devices to form a system daisy chain. The device closest to the CPU has the highest priority; within the CTC, interrupt priority is predetermined by channel number, with channel 0 having highest priority. According to Z80 system interrupt protocol, low priority devices or channels may not interrupt higher priority devices or channels that have already interrupted and not had their interrupt service routines completed. However, high priority devices or channels may interrupt the servicing of lower priority devices or channels. (For further details, see section 2.3: "Interrupt Control Logic".)

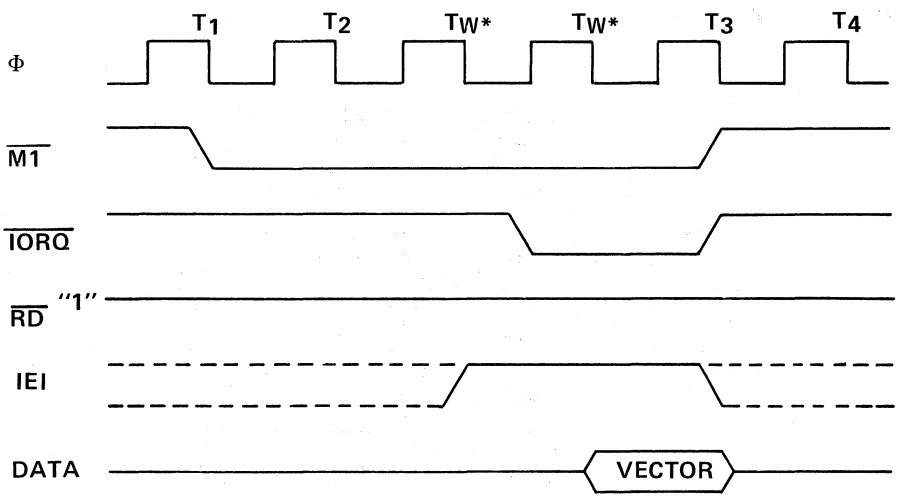
III

Sections 7.1 and 7.2 below describe the nominal timing relationships of the relevant CTC pins for the Interrupt Acknowledge Cycle and the Return from Interrupt Cycle. Section 7.3 below discusses a typical example of daisy chain interrupt servicing.

7.1 INTERRUPT ACKNOWLEDGE CYCLE

Figure 7.1-0 illustrates the timing associated with the Interrupt Acknowledge Cycle. Some time after an interrupt is requested by the CTC, the CPU will send out an interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$). To ensure that the daisy chain enable lines stablize, channels are inhibited from changing their interrupt request status when $\overline{M1}$ is active. $\overline{M1}$ is active about two clock cycles earlier than $\overline{IORQ}$, and $\overline{RD}$ is false to distinguish the cycle from an instruction fetch. During this time the interrupt logic of the CTC will determine the highest priority interrupting channel within the CTC and then will place its Interrupt Vector onto the Data Bus when $\overline{IORQ}$ goes active. Two wait states (T_W*) are automatically inserted at this time to allow the daisy chain to stabilize. Additional wait states may be added.

INTERRUPT ACKNOWLEDGE CYCLE
Figure 7.1-0

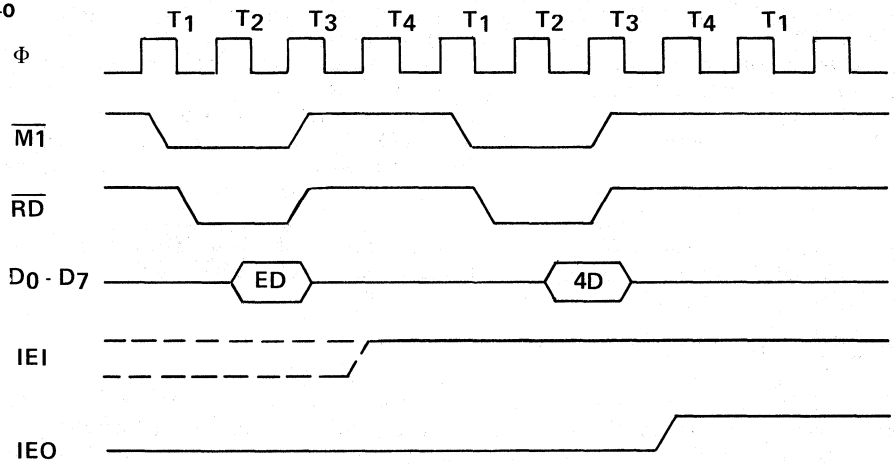


7.2 RETURN FROM INTERRUPT CYCLE

Figure 7.2-0 illustrates the timing associated with the RETI Instruction. This instruction is used at the end of an interrupt service routine to initialize the daisy chain enable lines for proper control of nested priority interrupt handling. The CTC decodes the two-byte RETI code internally and determines whether it is intended for a channel being serviced.

When several Z80 peripheral chips are in the daisy chain, IEI will become active on the chip currently under service when an EDH opcode is decoded. If the following opcode is 4DH, the peripheral being serviced will be re-initialized and its IEO will become active. Additional wait states are allowed.

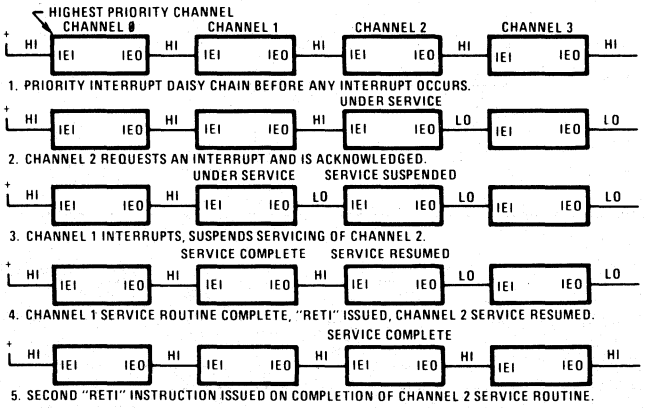
RETURN FROM INTERRUPT CYCLE
Figure 7.2-0



7.3 DAISY CHAIN INTERRUPT SERVICING

Figure 7.3-0 illustrates a typical nested interrupt sequence which may occur in the CTC. In this example, channel 2 interrupts and is granted service. While this channel is being serviced, higher priority channel 1 interrupts and is granted service. The service routine for the higher priority channel is completed, and a RETI instruction (see section 7.2 for further details) is executed to signal the channel that its routine is complete. At this time, the service routine of the lower priority channel 2 is resumed and completed.

DAISY CHAIN INTERRUPT SERVICING
Figure 7.3-0



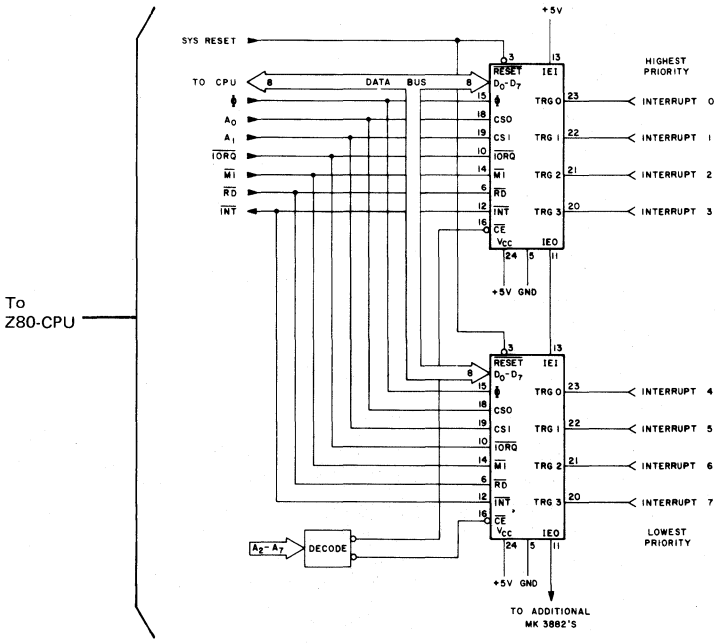
7.4 USING THE CTC AS AN INTERRUPT CONTROLLER

All of the Z80 family parts contain circuitry for prioritizing interrupts and supplying the vector to the CPU. However, in many Z80 based systems interrupts must be processed from devices which do not contain this interrupt circuitry. To handle this requirement the MK3882 CTC can be used, providing prioritized, independently vectored, maskable, edge selectable, count programmable external interrupt inputs. The MK3882 parts may be cascaded, expanding the system to as many as 256 interrupt inputs.

Each MK3882 contains 4 channels with counter inputs able to interrupt upon one or more (up to 256) edge transitions. The active transition may be programmed to be positive or negative. Each of the 4 channels has a programmable vector which is used in powerful Z80 mode 2 interrupt processing. When an interrupt is processed the vector is combined with the CPU I register to determine where the interrupt service routine start address is located. Additionally, priority resolution is handled within the MK3882 when more than one interrupt request is made simultaneously. When more than one MK3882 is used, the prioritizing is done, with the IEI/IEO chain resolving inter-chip priorities. Each channel can be independently "masked" by disabling that channel's local interrupt.

When programming the MK3882 to handle an input as a general purpose interrupt line, the channel is put in the counter mode, with the count set to 1, the active edge specified, and the vector loaded. When the programmed edge occurs, a mode 2 interrupt will be generated by the CTC and the Z80-CPU can vector directly to the service routine for the non-Z80 peripheral device. Note that after the interrupt, the CTC down counter is automatically reloaded with a count of one and the CTC channel begins looking for another active edge after the RETI of the interrupt routine. Therefore, once a particular channel is under service, no active edges will be recognized by that channel until execution of the RETI instruction of the corresponding interrupt routine. Of course, other channels of the CTC can generate interrupts and/or pending interrupts asynchronously, depending on their priority.

CTC AS AN INTERRUPT CONTROLLER
Figure 7.4-0



8.0 ELECTRICAL SPECIFICATIONS

8.1 ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... Specified operating range
Storage Temperature ..... -65°C to +150°C
Voltage On Any Pin With Respect To Ground ..... -0.3 V to +7 V
Power Dissipation ..... 0.8 W

All ac parameters assume a load capacitance of 100 pF max. Timing references between two output signals assume a load difference of 50 pF max.

*Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

III

8.2 D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = 5 V ± 5% unless otherwise specified

SYMBOL	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
V_{ILC}	Clock Input Low Voltage	-0.3	0.80	V	
V_{IHC}	Clock Input High Voltage (1)	V_{CC} -.6	V_{CC} +.3	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -250 μA
I_{CC}	Power Supply Current		120	mA	T_C = 400 nsec**
I_{LI}	Input Leakage Current		±10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float		10	μA	V_{OUT} = 2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float		-10	μA	V_{OUT} = 0.4 V
I_{OHD}	Darlington Drive Current	-1.5		mA	V_{OH} = 1.5 V

**T_C = 250 nsec for MK3882-4

8.3 CAPACITANCE

T_A = 25°C, f = 1 MHz

SYMBOL	PARAMETER	MAX	UNIT	TEST CONDITION
C_Φ	Clock Capacitance	20	pF	Unmeasured Pins
C_{IN}	Input Capacitance	5	pF	Returned to Ground
C_{OUT}	Output Capacitance	10	pF	

8.4 A.C. CHARACTERISTICS MK3882, MK3882-10, Z80-CTC

T_A = 0°C to 70°C, V_{CC} = +5 V ± 5%, unless otherwise noted

SIGNAL	SYMBOL	PARAMETER	3882 MIN	3882 MAX	3882-4 MIN	3882-4 MAX	UNIT	COMMENTS
Φ	t_C	Clock Period	400	(1)	250	(1)	ns	
	t_W (ΦH)	Clock Pulse Width, Clock High	170	2000	105	2000	ns	
	t_W(ΦL)	Clock Pulse Width, Clock Low	170	2000	105	2000		
	t_r, t_f	Clock Rise and Fall Times		30		30	ns	
	t_H	Any Hold Time for Specified Setup Time	0		0		ns	
CS, $\overline{CE}$, etc.	t_SΦ(CS)	Control Signal Setup Time to Rising Edge of Φ During Read or Write Cycle	160		145		ns	
D_0 - D_7	$t_{D\Phi}$(D)	Data Output Delay from Rising Edge of Φ During Read Cycle		240		200	ns	(2)
	t_SΦ(D)	Data Setup Time to Rising Edge of Φ During Write or $\overline{M1}$ Cycle	60		50		ns	
	t_{DI}(D)	Data Output Delay from Falling Edge of $\overline{IORQ}$ During INTA Cycle		340		160	ns	(2)
	t_F(D)	Delay to Floating Bus (Output Buffer Disable Time)		230		110	ns	
IEI	t_S(IEI)	IEI Setup Time to Falling Edge of $\overline{IORQ}$ During INTA Cycle	200		140		ns	
IEO	t_{DH}(IO)	IEO Delay Time from Rising Edge		220		160	ns	(3)
	t_{DL}(IO)	IEO Delay Time from Falling Edge of IEI		190		130	ns	(3)
	t_{DM}(IO)	IEO Delay from Falling Edge of $\overline{M1}$ (Interrupt Occurring just Prior to $\overline{M1}$)		300		190	ns	(3)
$\overline{IORQ}$	t_SΦ(IR)	$\overline{IORQ}$ Setup Time to Rising Edge of Φ During Read or Write Cycle	250		115		ns	
$\overline{M1}$	t_SΦ(M1)	$\overline{M1}$ Setup Time to Rising Edge of Φ During INTA or $\overline{M1}$ Cycle	210		90		ns	
$\overline{RD}$	t_SΦ(RD)	$\overline{RD}$ Setup Time to Rising Edge of Φ During Read or $\overline{M1}$ Cycle	240		115		ns	
$\overline{INT}$	t_DΦ(IT)	$\overline{INT}$ Delay from Rising Edge of Φ		t_C(Φ) + 200		t_C(Φ) + 140		(7)
CLK/ TRG_{0-3}	t_C(CK)	Clock Period	$2t_C$(Φ)		$2t_C$(Φ)			(5)
	t_r, t_f	Clock and Trigger Rise and Fall Times		50		50	ns	
	t_S(CK)	Clock Setup Time to Rising Edge of Φ for Immediate Count	210		130		ns	(5)
	t_S(TR)	Trigger Setup Time to Rising Edge of Φ for Enabling of Prescaler on Following Rising Edge of Φ	210		130		ns	(6)

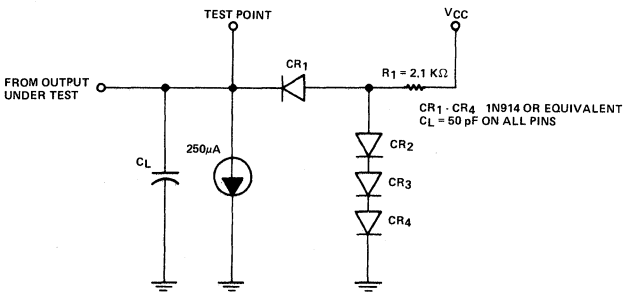
SIGNAI	SYMBOL	PARAMETER	3882		3882-4		UNIT	COMMENTS
			MIN	MAX	MIN	MAX		
CLK/ TRG_{0-3}	$t_W(CTH)$	Clock and Trigger High Pulse Width	200		120		ns	(7)
	$t_W(CTL)$	Clock and Trigger Low Pulse Width	200		120		ns	(7)
ZC/ TO_{0-2}	$t_{DH}(ZC)$	ZC/TO Delay Time from Rising Edge of Φ, ZC/TO High		190		120	ns	(7)
	$t_{DL}(ZC)$	ZC/TO Delay Time from Falling Edge of Φ, ZC/TO Low		190		120	ns	(7)

NOTES:

1. $t_C = t_W(\Phi H) + t_W(\Phi L) + t_r + t_f$.
2. Increase delay by 10 nsec for each 50 pF increase in loading 200 pF maximum for data lines and 100 pF for control lines.
3. Increase delay by 2 nsec for each 10 pF increase in loading, 100 pF maximum.
4. $\overline{RESET}$ must be active for a minimum of 3 clock cycles.
5. Counter mode
6. Timer mode
7. Counter and Timer mode

OUTPUT LOAD CIRCUIT

Figure 2

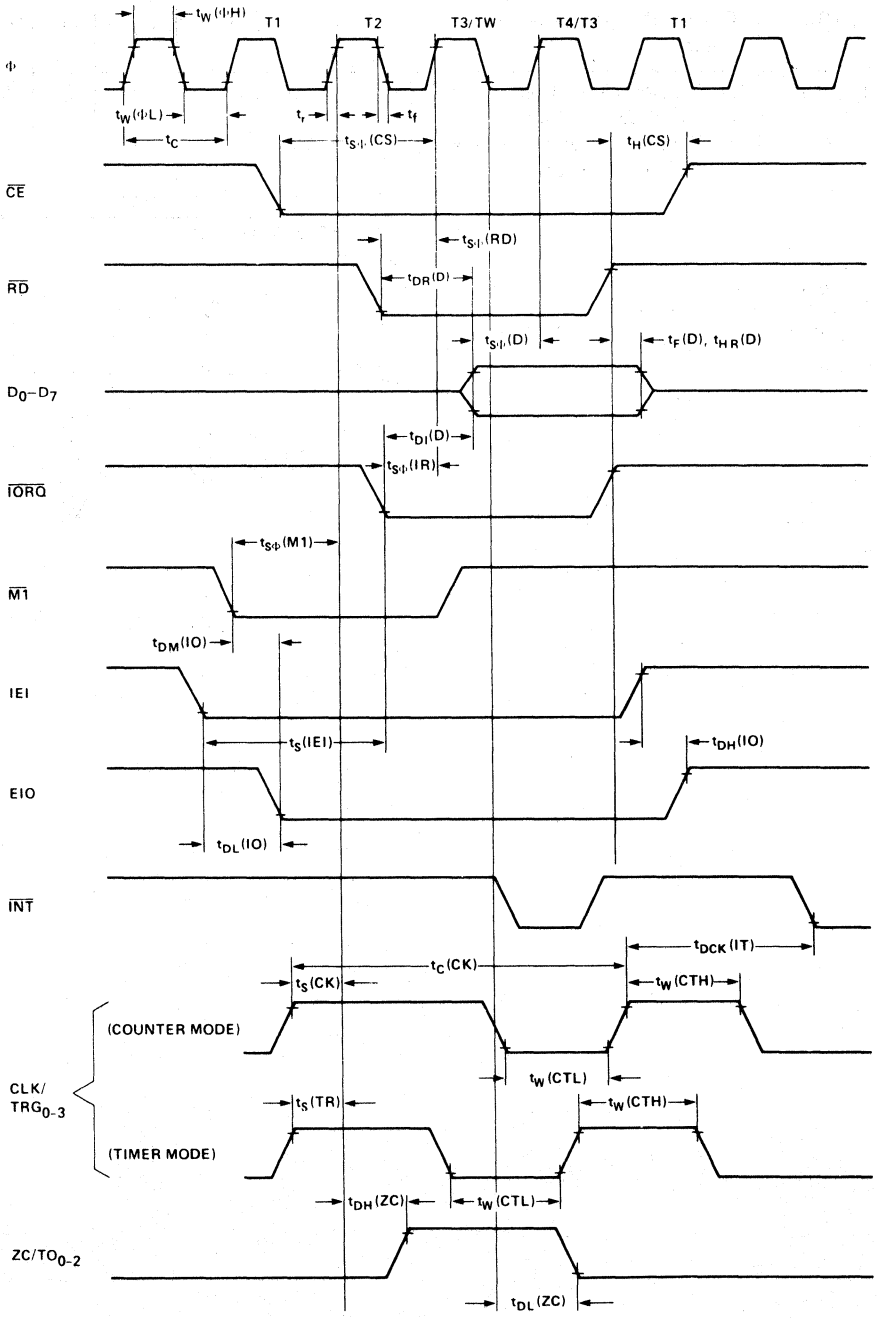


III

8.5 A.C. TIMING DIAGRAM

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	V_{CC} - .6V	.8V
OUTPUT	2.0V	.8V
INPUT	2.0V	.8V
FLOAT	ΔV	+0.5V



9.0 ORDERING INFORMATION

PART NO.	DESIGNATOR	PACKAGE TYPE	MAX CLOCK FREQUENCY	TEMPERATURE RANGE
MK3882N	Z80-CTC	Plastic	2.5 MHz	0° to 70°C
MK3882P	Z80-CTC	Ceramic	2.5 MHz	
MK3882N-4	Z80A-CTC	Plastic	4.0 MHz	
MK3882P-4	Z80A-CTC	Ceramic	4.0 MHz	
MK3882P-10	Z80-CTC	Ceramic	2.5 MHz	–40° to +85°C

Direct Memory Access Controller MK3883

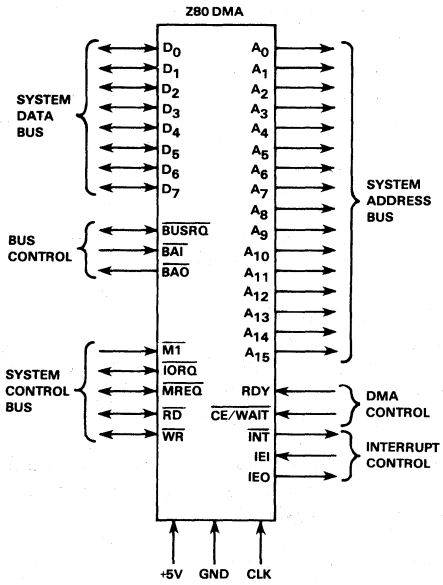
FEATURES

- Transfers, searches and search/transfers in byte-at-a-time, burst or continuous modes. Cycle length and edge timing can be programmed to match the speed of any port.
- Dual port addresses (source and destination) generated for memory-to-I/O, memory-to-memory, or I/O-to-I/O operations. Addresses may be fixed or automatically incremented/decremented.
- Next-operation loading without disturbing current operations via buffered starting-address registers. An entire previous sequence can be repeated automatically.
- Extensive programmability of functions. CPU can read complete channel status.
- Standard Z80 Family bus-request and prioritized interrupt-request daisy chains implemented without external logic. Sophisticated, internally modifiable interrupt vectoring.
- Direct interfacing to system buses without external logic.

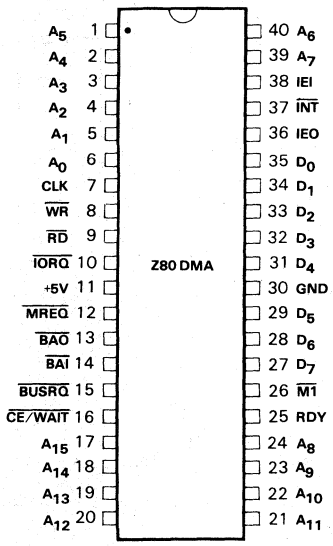
GENERAL DESCRIPTION

The MK3883 Z80 DMA (Direct Memory Access) is a powerful and versatile device for controlling and processing transfers of data. Its basic function of managing CPU-independent transfers between two ports is augmented by an array of features that optimize transfer speed and control with little or no external logic in systems using an 8- or 16-bit data bus and a 16-bit address bus.

PIN FUNCTIONS
Figure 1



PIN ASSIGNMENTS
Figure 2



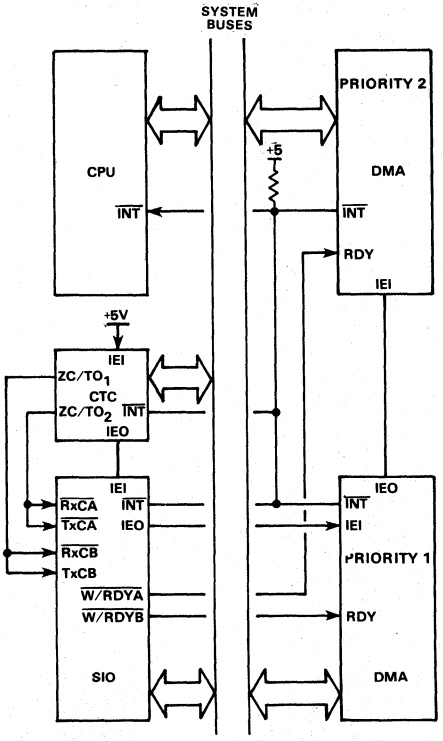
Transfers can be done between any two ports (source and destination), including memory-to-I/O, memory-to-memory, and I/O-to-I/O. Dual port addresses are automatically generated for each transaction and may be either fixed or incrementing/decrementing. In addition, bit-maskable byte searches can be performed either concurrently with transfers or as an operation in itself.

The MK3883 Z80 DMA contains direct interfacing to and independent control of system buses, as well as sophisticated bus and interrupt controls. Many programmable features, including variable cycle timing and auto-restart minimize CPU software overhead. They are especially useful in adapting this special-purpose transfer processor to a broad variety of memory, I/O and CPU environments.

The MK3883 Z80 DMA is an n-channel silicon-gate depletion-load device packaged in a 40-pin plastic, or ceramic DIP. It uses a single +5V power supply and the standard Z80 Family single-phase clock.

Z80 ENVIRONMENT WITH MULTIPLE DMA CONTROLLERS

Figure 3



FUNCTIONAL DESCRIPTION

Classes of Operation

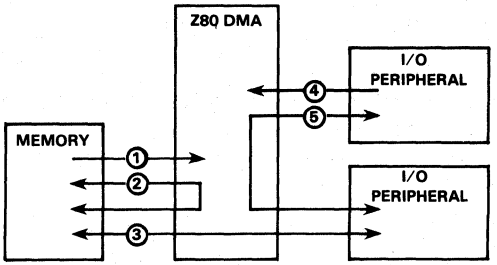
The MK3883 Z80 DMA has three basic classes of operation:

- Transfers of data between two ports (memory or I/O peripheral)
- Searches for a particular 8-bit maskable byte at a single port in memory or an I/O peripheral
- Combined transfers with simultaneous search between two ports

Figure 4 illustrates the basic functions served by these classes of operation.

BASIC FUNCTIONS OF THE Z80 DMA

Figure 4



1. Search memory
2. Transfer memory-to-memory (optional search)
3. Transfer memory-to-I/O (optional search)
4. Search I/O
5. Transfer I/O-to-I/O (optional search)

During a transfer, the DMA assumes control of the system control, address, and data buses. Data is read from one addressable port and written to the other addressable port, byte by byte. The ports may be programmed to be either system main memory or peripheral I/O devices. Thus, a block of data may be written from one peripheral to another, from one area of main memory to another, or from a peripheral to main memory and vice versa.

During a search-only operation, data is read from the source port and compared byte by byte with the DMA-internal register containing a programmable match byte. This match byte may optionally be masked so that only certain bits within the match byte are compared. Search rates up to 1.25M bytes per second can be obtained with the 2.5MHz MK3883 Z80 DMA or 2M bytes per second with the 4MHz MK3883-4 Z80 DMA.

In combined searches and transfers, data is transferred between two ports while simultaneously searching for a bit-maskable byte match.

Data transfers or searches can be programmed to stop or interrupt under various conditions. In addition, CPU-readable status bits can be programmed to reflect the condition.

Modes of Operation

The MK3883 Z80 DMA can be programmed to operate in one of three transfer and/or search modes:

- Byte-at-a-time: data operations are performed one byte at a time. Between each byte operation the system buses are released to the CPU. The buses are requested again for each succeeding byte operation.
- Burst: data operations continue until a port's Ready line to the DMA goes inactive. The DMA then stops and releases the system buses after completing its current byte operation.
- Continuous: data operations continue until the end of the programmed block of data is reached before the system buses are released. If a port's Ready line goes inactive before this occurs, the DMA simply pauses until the Ready line comes active again.

In all modes, once a byte of data is read into the DMA, the operation on the byte will be completed in an orderly fashion, regardless of the state of other signals (including a port's Ready line).

Due to the DMA's high-speed buffered method of reading data, operations on one byte are not completed until the next byte is read in. Consequently, total transfer or search block lengths must be two or more bytes, and those block lengths programmed into the DMA must be one byte less than the desired block length (count is N-1 where N is the block length).

Commands and Status

The Z80 DMA has several writeable control registers and readable status registers available to the CPU. Control bytes can be written to the DMA while the DMA is enabled or disabled, but the act of writing a control byte to the DMA disables the DMA until it is again enabled by a specific command. Status bytes can also be read at any time, but writing the Read Status command or the Read Mask command disables the DMA.

Control bytes to the DMA include those which effect immediate command actions such as enable, disable, reset, load starting-address buffers, continue, clear counters, clear status bits and the like. In addition, many mode-setting control bytes can be written, including mode and class of operation, port configuration, starting addresses, block length, address counting rule, match and match-mask byte, interrupt conditions, interrupt vector, status-affects-vector condition, pulse counting, auto restart, Ready-line and Wait-line rules, and read mask.

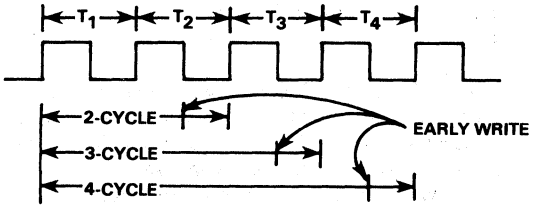
Readable status registers include a general status byte reflecting Ready-line, end-of-block, byte-match and interrupt conditions, as well as Dual-byte registers for the current byte count, Port A address and Port B address.

Variable Cycle

The Z80 DMA has the unique feature of programmable operation-cycle length. This is valuable in tailoring the DMA to the particular requirements of other system components (fast or slow) and maximizes the data-transfer rate. It also eliminates external logic for signal conditioning.

There are two aspects to the variable cycle feature. First, the entire read and write cycles (periods) associated with the source and destination ports can be independently programmed as 2, 3 or 4 T-cycles long (more if Wait cycles are used), thereby increasing or decreasing the speed with which all DMA signals change (Figure 5).

VARIABLE CYCLE LENGTH
Figure 5



Second, the four signals in each port specifically associated with transfers of data (I/O Request, Memory Request, Read and Write) can each have its active trailing edge terminated one-half T-cycle early. This adds a further dimension of flexibility and speed, allowing such things as shorter-than-normal Read or Write signals that go inactive before data starts to change.

Address Generation

Two 16-bit addresses are generated by the Z80 DMA for every transfer or search operation, one address for the source Port A and another for the destination Port B. Each address can be either variable or fixed. Variable addresses can increment or decrement from the programmed starting address. The fixed-address capability eliminates the need for separate enabling wires to I/O ports.

Port addresses are multiplexed onto the system address bus, depending on whether the DMA is reading the source port or writing to the destination port. Two readable address counters (2-bytes each) keep the current address of each port.

Auto Restart

The starting addresses of either port can be reloaded automatically at the end of a block. This option is selected by the Auto Restart control bit. The byte counter is cleared when the addresses are reloaded.

The Auto Restart feature relieves the CPU of software overhead for repetitive operations such as CRT refresh and many others. Moreover, the CPU can write different starting addresses into buffer registers during transfers causing the Auto Restart to begin at a new location.

Interrupts

The MK3883 Z80 DMA can be programmed to interrupt the CPU on four conditions:

- Interrupt on Ready (before requesting bus)
- Interrupt on Match
- Interrupt on End of Block
- Interrupt on Match at End of Block

Any of these interrupts cause an interrupt-pending status bit to be set, and each of them can optionally alter the DMA's interrupt vector. Due to the buffered constraint mentioned under "Modes of Operation," interrupts on Match at End of Block are caused by matches to the byte just prior to the last byte in the block.

The DMA shares the Z80 family's elaborate interrupt scheme, which provides fast interrupt service in real-time applications. In a Z80 CPU environment, the DMA passes its internally modifiable 8-bit interrupt vector to the CPU, which adds an additional eight bits to form the memory address of the interrupt-routine table. This table contains the address of the beginning of the interrupt routine itself.

In this process, CPU control is transferred directly to the interrupt routine, so that the next instruction executed after an interrupt acknowledge is the first instruction of the interrupt routine itself.

Pulse Generation

External devices can keep track of how many bytes have been transferred by using the DMA's pulse output, which provides a signal at 256-byte intervals. The interval sequence may be offset at the beginning by 1 to 255 bytes.

The interrupt line outputs the pulse signal in a manner that prevents misinterpretation by the CPU as an interrupt request, since it only appears when the Bus Request and Bus Acknowledge lines are both active.

PIN DESCRIPTIONS

A_0-A_{15}. System Address Bus (output, 3-state). Addresses generated by the DMA are sent to both source and destination ports (main memory or I/O peripherals) on these lines.

$\overline{BAI}$. Bus Acknowledge In (input, active Low). Signals that the system buses have been released for DMA control. In multiple-DMA configurations, the $\overline{BAI}$ pin of the highest priority DMA is normally connected to the Bus Acknowledge pin of the CPU. Lower-priority DMAs have their $\overline{BAI}$ connected to the $\overline{BAO}$ of a higher-priority DMA.

$\overline{BAO}$. Bus Acknowledge Out (output, active Low). In a multiple-DMA configuration, this pin signals that no other higher-priority DMA has requested the system busses. $\overline{BAI}$ and $\overline{BAO}$ form a daisy chain for multiple-DMA priority resolution over bus control.

$\overline{BUSRQ}$. Bus Request (bidirectional, active Low, open drain). As an output, it sends requests for control of the system address bus, data bus and control bus to the CPU. As an input when multiple DMAs are strung together in a priority daisy chain via $\overline{BAI}$ and $\overline{BAO}$, it senses when another DMA has requested the buses and causes this DMA to refrain from bus requesting until the other DMA is finished. Because it is a bidirectional pin, there cannot be any buffers between this DMA and any other DMA. It can, however, have a unidirectional into the CPU. A pull-up resistor is connected to this pin.

$\overline{CE}$/$\overline{WAIT}$. Chip Enable and Wait (input, active Low). Normally this functions only as a $\overline{CE}$ line, but it can also be programmed to serve a $\overline{WAIT}$ function. As a $\overline{CE}$ line from the CPU, it becomes active when $\overline{WR}$ and $\overline{IORQ}$ are active and the I/O port address on the system address bus is the DMA's address, thereby allowing a transfer of control or command bytes from the CPU to the DMA. As a $\overline{WAIT}$ line from memory or I/O devices, after the DMA has received a bus-request acknowledge from the CPU, it causes wait states to be inserted in the DMA's operation cycles thereby slowing the DMA to a speed that matches the memory or I/O device.

CLK. System clock (input). Standard Z80 single-phase clock at 2.5MHz (MK3883) or 4.0MHz (MK3883-4). For slower system clocks, a TTL gate with a large pullup resistor may be adequate to meet the timing and voltage level specification. For higher-speed systems, use a clock driver with an active pullup to meet the V_{IH} specification and risetime requirements.

D_0-D_7. System Data Bus (bidirectional, 3-state). Commands from the CPU, DMA status, and data from memory or I/O peripherals are transferred on these lines.

IEI. Interrupt Enable In (input, active High). This is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO. Interrupt Enable Out (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this DMA. Thus, this signal blocks lower-priority devices from interrupting while a higher-priority device is being serviced by its CPU interrupt service routine.

$\overline{INT}$. Interrupt Request (output, active Low, open drain). This requests a CPU interrupt. The CPU acknowledges the interrupt by pulling its $\overline{IORQ}$ output Low during an $\overline{M1}$ cycle. It is typically connected to the $\overline{INT}$ pin of the CPU with a pullup resistor and tied to all other $\overline{INT}$ pins in the system.

$\overline{IORQ}$. Input/Output Request (bidirectional, active Low, 3-state). As an input, this indicates that the lower half of the address bus holds a valid I/O port address for transfer of control or status bytes from or to the CPU, respectively; this DMA is the addressed port if its $\overline{CE}$ pin and its $\overline{WR}$ or $\overline{RD}$ pins are simultaneously active. As an output, after the DMA has taken control of the system busses, it indicates that the lower half of the address bus holds a valid port address for another I/O device involved in a DMA transfer of data. When $\overline{IORQ}$ and $\overline{M1}$ are both active simultaneously, an interrupt acknowledge is indicated.

$\overline{M1}$. Machine Cycle One (input, active Low). Indicates that the current CPU machine cycle is an instruction fetch. It is used by the DMA to decode the return-from-interrupt instruction (RETI) (ED-4D) sent by the CPU. During two-byte instruction fetches, $\overline{M1}$ is active as each opcode byte is fetched. An interrupt acknowledge is indicated when both $\overline{M1}$ and $\overline{IORQ}$ are active.

$\overline{MREQ}$. Memory Request (bidirectional, active Low, 3-state). This indicates that the address bus holds a valid address for a memory read or write operation. As an input, it indicates that control or status information from or to memory is to be transferred to the DMA, if the DMA's $\overline{CE}$ and $\overline{WR}$ or $\overline{RD}$ lines are simultaneously active. As an output, after the DMA has taken control of the system buses, it indicates a DMA transfer request from or to memory.

$\overline{RD}$. Read (bidirectional, active Low, 3-state). As an input, this indicates that the CPU wants to read status bytes from the DMA's read registers. As an output, after the DMA has taken control of the system buses, it indicates a DMA-controlled read from a memory or I/O port address.

$\overline{WR}$. Write (bidirectional, active Low, 3-state). As an input, this indicates that the CPU wants to write control or command bytes to the DMA write registers. As an output, after the DMA has taken control of the system busses, it indicates a DMA-controlled write to a memory or I/O port address.

RDY. Ready (input, programmable active Low or High). This is monitored by the DMA to determine when a peripheral device associated with a DMA port is ready for a read or write operation. Depending on the mode of DMA operation (byte, burst or continuous), the RDY line indirectly controls DMA activity by causing the $\overline{BUSRQ}$ line to go Low or High.

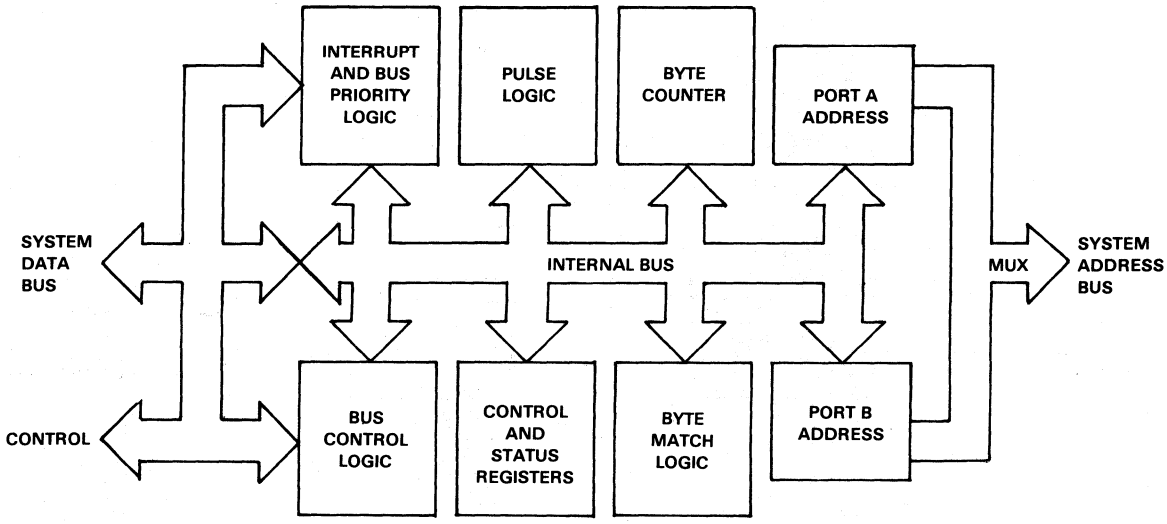
III

INTERNAL STRUCTURE

The internal structure of the MK3883 Z80 DMA includes driver and receiver circuitry for interfacing with an 8-bit system data bus, a 16-bit system address bus, and system control lines (Figure 6). In a Z80 CPU environment, the DMA can be tied directly to the analogous pins on the CPU (Figure 7) with no additional buffering, except for the $\overline{CE/WAIT}$ line.

The DMA's internal data bus interfaces with the system data bus and services all internal logic and registers. Addresses generated from this logic for Ports A and B (source and destination) of the DMA's single transfer channel are multiplexed onto the system address bus.

BLOCK DIAGRAM
Figure 6



Specialized logic circuits in the DMA are dedicated to the various functions of external bus interfacing, internal bus control, byte matching, byte counting, periodic pulse generation, CPU interrupts, bus requests, and address generation. A set of twenty-one writeable control registers and seven readable status registers provide the means by which the CPU governs and monitors the activities of these logic circuits. All registers are eight bits wide, with double-byte information stored in adjacent registers. The two starting-address registers (two bytes each) for Ports A and B are buffered.

The 21 writeable control registers are organized into seven base-register groups, most of which have multiple registers. The base registers in each writeable group contain both control/command bits and pointer bits that can be set to address other registers within the group. The seven readable status registers have no analogous second-level registers.

The registers are designated as follows, according to their base-register groups:

WR0-WR6 - Write Register groups 0 through 6 (7 base registers plus 14 associated registers)

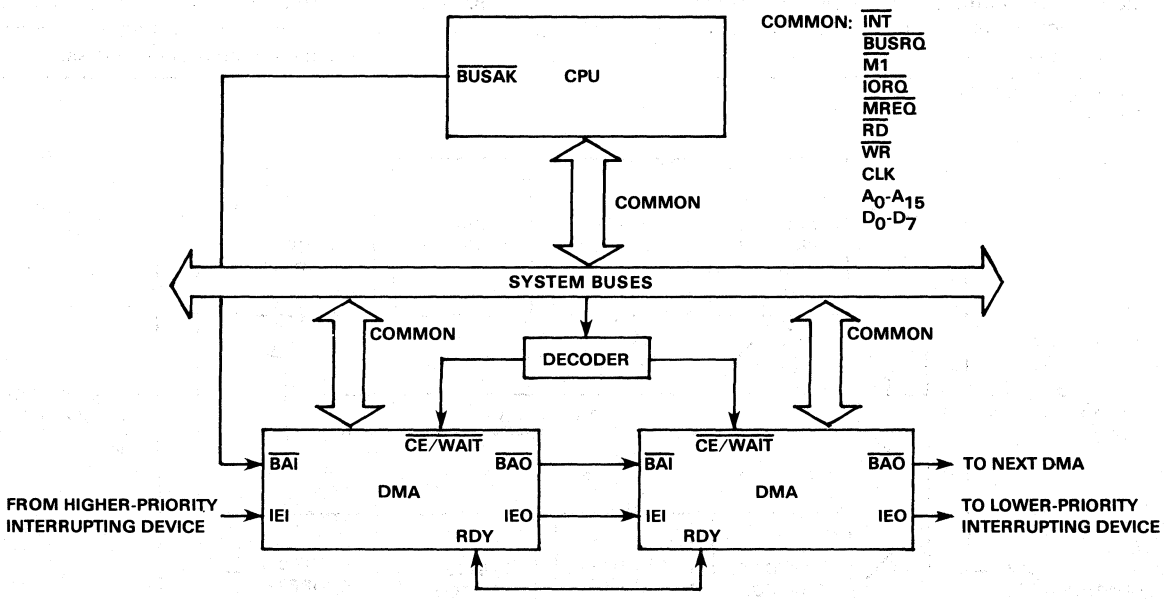
RR0-RR6 - Read Registers 0 through 6

Writing to a register within a write-register group involves first writing to the base register, with the appropriate pointer bits set, then writing to one or more of the other registers within the group. All seven of the readable status registers are accessed sequentially according to a programmable mask contained in one of the writeable registers. The section entitled "Programming" explains this in more detail.

A pipelining scheme is used for reading data in. The programmed block length is the number of bytes compared to the byte counter, which increments at the end of each cycle. In searches, data byte comparisons with the match byte are made during the read cycle of the next byte. Matches are, therefore, discovered only after the next byte is read in.

In multiple-DMA configurations, interrupt-request daisy chains are prioritized by the order in which their IEI and IEO lines are connected. The system bus, however, may not be pre-empted. Any DMA that gains access to the system buses keeps them until it is finished.

MULTIPLE-DMA INTERCONNECTION TO THE Z80 CPU
Figure 7



WRITE REGISTERS

WR0	Base register byte Port A starting address (low byte) Port A starting address (high byte) Block length (low byte) Block length (high byte)
WR1	Base register byte Port A variable-timing byte
WR2	Base register byte Port B variable-timing byte
WR3	Base register byte Mask byte Match byte
WR4	Base register byte Port B starting address (low byte) Port B starting address (high byte) Interrupt control byte Pulse control byte Interrupt vector
WR5	Base register byte
WR6	Base register byte Read mask

READ REGISTERS

RR0	Status byte
RR1	Byte counter (low byte)
RR2	Byte counter (high byte)
RR3	Port A address counter (low byte)
RR4	Port A address counter (high byte)
RR5	Port B address counter (low byte)
RR6	Port B address counter (high byte)

PROGRAMMING

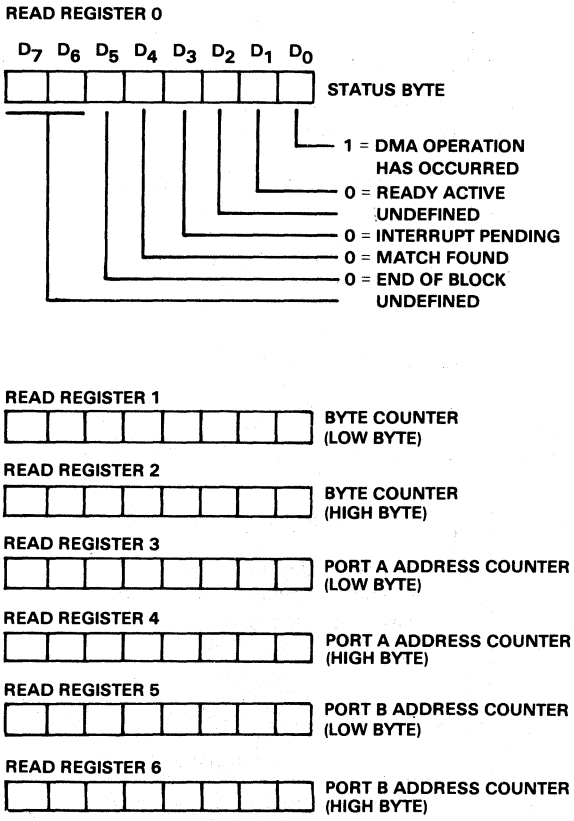
The Z80 DMA has two programmable fundamental states: (1) an enabled state, in which it can gain control of the system buses and direct the transfer of data between ports, and (2) a disabled state, in which it can initiate neither bus requests or data transfers. When the DMA is powered up or reset by any means, it is automatically placed into the disabled state. Program commands can be written to it by the CPU in either state, but this automatically puts the DMA in the disabled state, which is maintained until an enabled command is issued by the CPU. The CPU must program the DMA in advance of any data search or transfer by addressing it as an I/O port and sending a sequence of control bytes using an Output instruction (such as OTIR for the Z80 CPU).

Writing

Control or command bytes are written into one or more of the Write Register groups (WR0-WR6) by first writing to the base register byte in that group. All groups have base registers and most groups have additional associated registers. The associated registers in a group are sequentially accessed by first writing a byte to the base register containing register-group identification and pointer bits (1's) to one or more of that base register's associated registers.

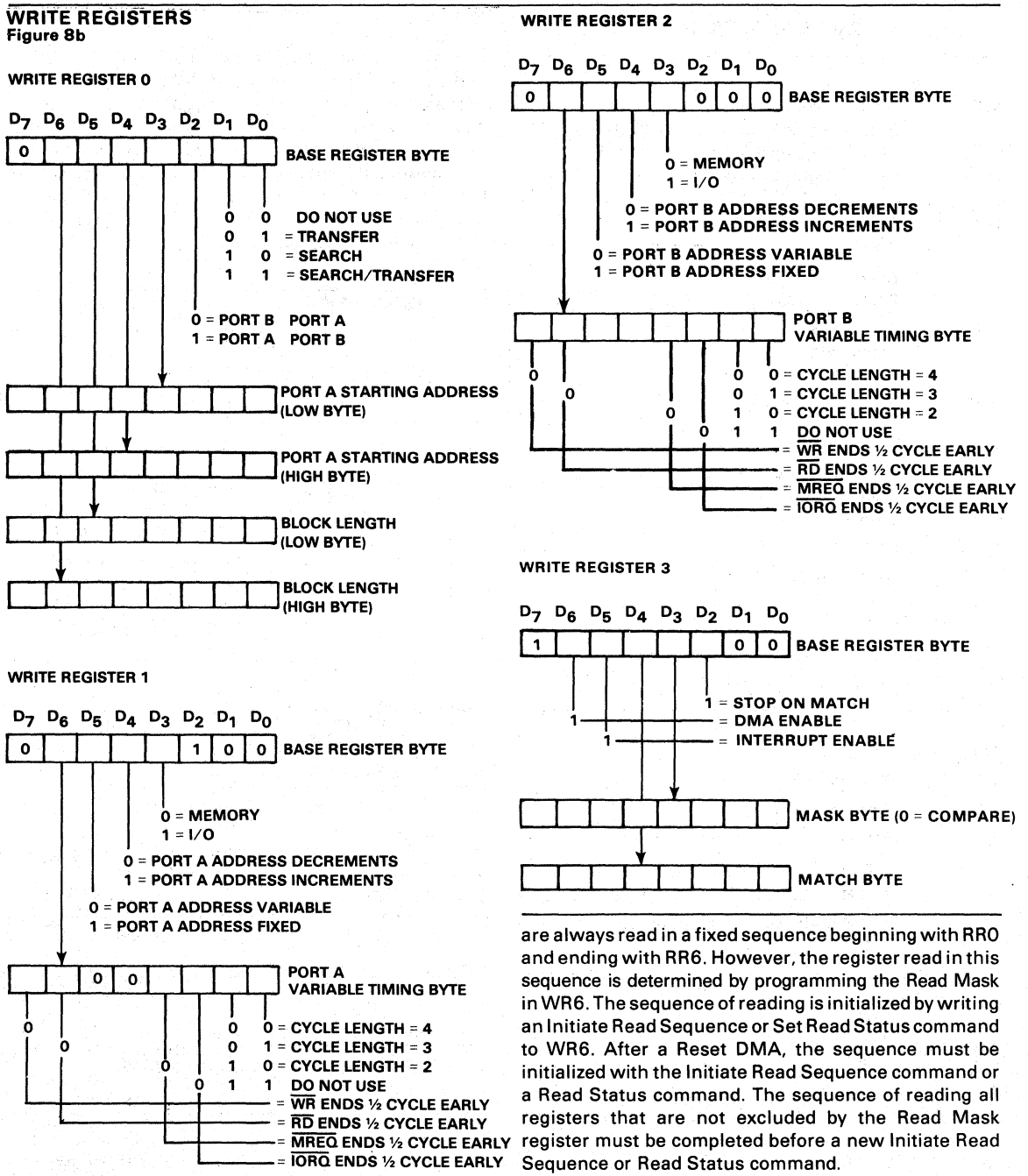
READ REGISTERS

Figure 8a.



This is illustrated in Figure 8. In this figure, the sequence in which associated registers within a group can be written to is shown by the vertical position of the associated registers. For example, if a byte written to the DMA contains the bits that identify WR0 (bits D0, D1 and D7), and also contains 1's in the bit positions that point to the associated "Port A Starting Address (low byte)" and "Port A Starting Address (high byte)" then the next two bytes written to the DMA will be stored in these two registers, in that order.

WRITE REGISTERS
Figure 8b



Reading

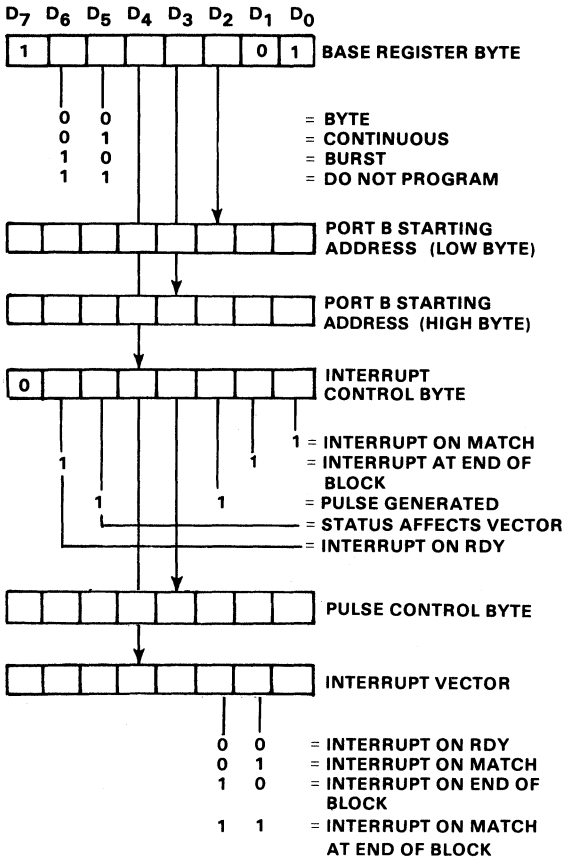
The Read Registers (RR0-RR6) are read by the CPU by addressing the DMA as an I/O port using an Input instruction (such as INIR for the Z80 CPU). The readable bytes contain DMA status, byte-counter values, and port addresses since the last DMA reset. The registers are always read in a fixed sequence beginning with RR0 and ending with RR6. However, the register read in this sequence is determined by programming the Read Mask in WR6. The sequence of reading is initialized by writing an Initiate Read Sequence or Set Read Status command to WR6. After a Reset DMA, the sequence must be initialized with the Initiate Read Sequence command or a Read Status command. The sequence of reading all registers that are not excluded by the Read Mask register must be completed before a new Initiate Read Sequence or Read Status command.

Fixed-Address Programming

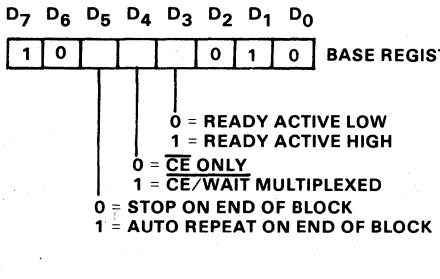
A special circumstance arises when programming a destination port to have a fixed address. The load command in WR6 only loads a fixed address to a port selected as the source, not to a port selected as the destination. Therefore, a fixed destination address must be loaded by temporarily declaring it a fixed-source

WRITE REGISTERS
Figure 8b

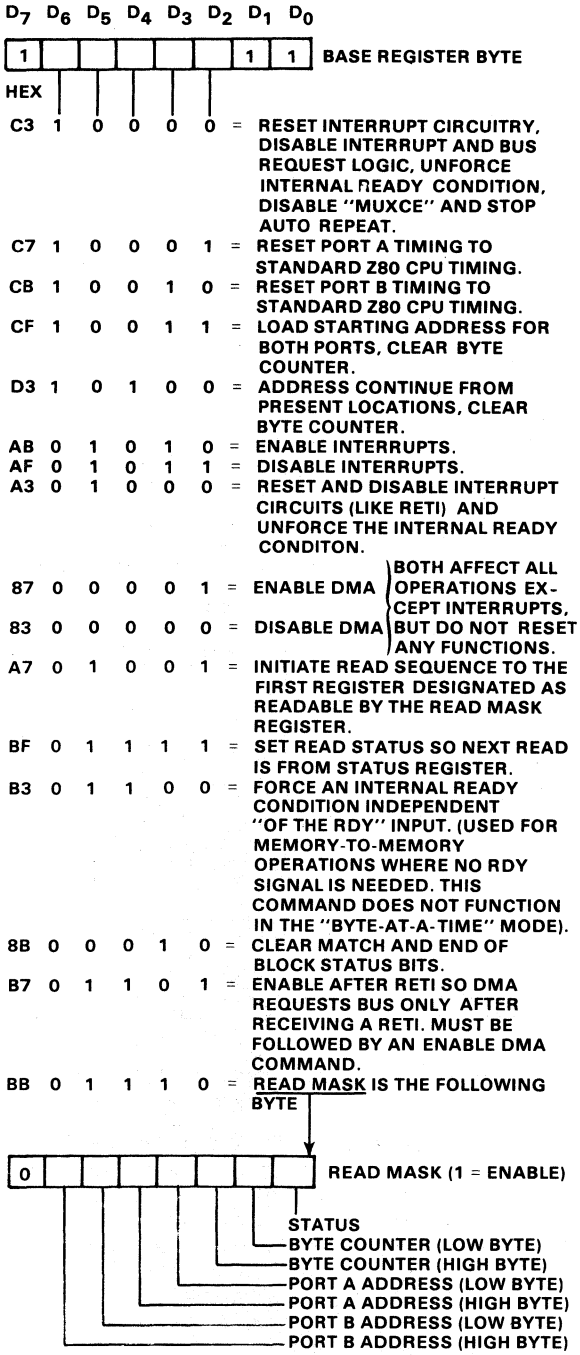
WRITE REGISTER 4



WRITE REGISTER 5



WRITE REGISTER 6



III

SAMPLE DMA PROGRAM
Figure 9

COMMENTS	D_7	D_8	D_5	D_4	D_3	D_2	D_1	D_0	HEX
WR0 sets DMA to receive block length, Port A starting address and temporarily sets Port B as source.	0	1 Block Length Upper Follows	1 Block Length Lower Follows	1 Port A Upper Address Follows	1 Port A Lower Address Follows	0 B→A Temporary for Loading B Address	0 Transfer, No Search	1	79
Port A address (lower)	0	1	0	1	0	0	0	0	50
Port A address (upper)	0	0	0	1	0	0	0	0	10
Block length (lower)	0	0	0	0	0	0	0	0	00
Block length (upper)	0	0	0	1	0	0	0	0	10
WR1 defines Port A as peripheral with fixed address.	0	0 No Timing Follows	0 Address Changes	1 Address Increments	0 Port is Memory	1 This is Port A	0	0	14
WR2 defines Port B as peripheral with fixed address.	0	0 No Timing Follows	1 Fixed Address	0	1 Port is I/O	0 This is Port B	0	0	28
WR4 sets mode to Burst, sets DMA to expect Port B address.	1	1 Burst Mode	0	0 No Interrupt Control Byte Follows	0 No Upper Address	1 Port B Lower Address Follows	0	1	C5
Port B address (lower)	0	0	0	0	0	1	0	1	05
WR5 sets Ready active High	1	0	0 No Auto Restart	0 No Wait States	1 RDY Active High	0	1	0	8A
WR6 loads both Port addresses and resets block counter.*	1	1	0	0 Load	1	1	1	1	CF
WR0 sets Port A as source.*	0	0	0 No Address of Block Length Bytes	0	0	1 A→B	0 Transfer, No Search	1	05
WR6 reloads Port addresses and resets block counter	1	1	0	0 Load	1	1	1	1	CF
WR6 enables DMA to start operation.	1	0	0	0 Enable DMA	0	1	1	1	87

NOTE: The actual number of bytes transferred is one more than specified by the block length.
*These commands are necessary only in the case of a fixed destination address.

address and subsequently declaring the true source as such, thereby implicitly making the other a destination.

The following example illustrates the steps in this procedure, assuming that transfers are to occur from a variable-address source (Port A) to a fixed-address destination (Port B):

1. Temporarily declare Port B as source in WR0.
2. Load Port B address in WR6.
3. Declare Port A as source in WR0.
4. Load Port A address in WR6.
5. Enable DMA in WR6.

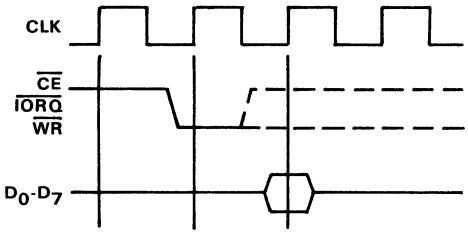
Figure 9 illustrates a program to transfer data from memory (Port A) to a peripheral device (Port B). In this example, the Port A memory starting address is 1050_H and the Port B peripheral fixed address is 05_H. Note that the data flow is 1001_H bytes—one more than specified by the block length. The table of DMA commands may be stored in consecutive memory locations and transferred to the DMA with an output instruction such as the Z80 CPU's OTIR instruction.

INACTIVE STATE TIMING (DMA as CPU Peripheral)

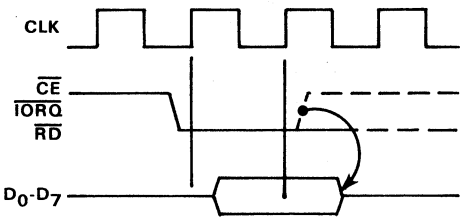
In its inactive state, the DMA is addressed by the CPU as an I/O peripheral for write and read (control and status) operations. Write timing is illustrated in Figure 10.

Reading of the DMA's status byte, byte counter or port address counters is illustrated in Figure 11. These operations require less than three T-cycles. The $\overline{CE}$, $\overline{IORQ}$ and $\overline{RD}$ lines are made active over two rising edges of CLK, and data appears on the bus approximately one T-cycle after they become active.

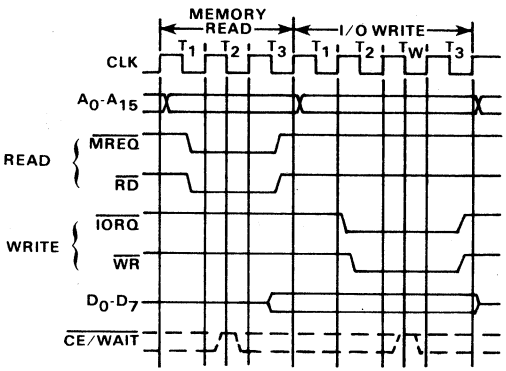
CPU-TO-DMA WRITE CYCLE
Figure 10



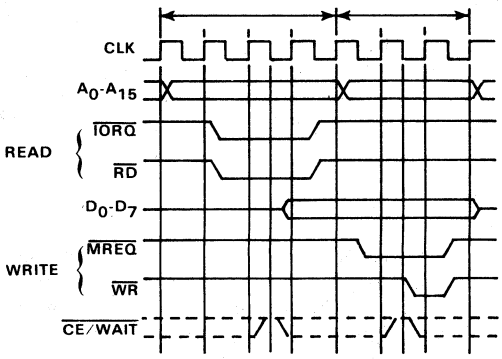
CPU-TO-DMA READ CYCLE
Figure 11



MEMORY-TO-I/O TRANSFER
Figure 12



I/O-TO-MEMORY TRANSFER
Figure 13



ACTIVE STATE TIMING (DMA as Bus Controller)

The DMA is active when it takes control of the system bus and begins transferring data.

Default Read and Write Cycles

By default, and after reset the DMA's timing of read and write operations is exactly the same as the Z80 CPU's timing of read and write cycles for memory and I/O peripherals, with one exception: during a read cycle, data is latched on the falling edge of T_3 and held on the data bus across the boundary between read and write cycles, through the end of the following write cycle.

Figure 12 illustrates the timing for memory-to-I/O port transfers and Figure 13 illustrates I/O-to-memory transfers. Memory-to-memory and I/O-to-I/O transfer timings are simply permutations of these diagrams.

The default timing uses three T-cycles for memory transactions and four T-cycles for I/O transactions, which include one automatically inserted wait cycle between T_2 and T_3. If the $\overline{CE/WAIT}$ line is programmed to act as WAIT line during the DMA's active state, it is sampled on the falling edge of T_2 for memory transactions and the falling edge of T_W for I/O transactions. If $\overline{CE/WAIT}$ is low during this time another T-cycle is added, during which the $\overline{CE/WAIT}$ line will again be sampled. The duration of transactions can thus be indefinitely extended.

Variable Cycle and Edge Timing

The Z80 DMA's default operation-cycle length for the source (read) port and destination (write) port can be independently programmed. This variable-cycle feature allows read or write cycles consisting of two, three or four T-cycles (more if Wait cycles are inserted), thereby increasing or decreasing the speed of all signals generated by the DMA. In addition, the trailing edges of the $\overline{IORQ}$, $\overline{MREQ}$, $\overline{RD}$ and $\overline{WR}$ signals can be independently terminated one-half cycle early. Figure 14 illustrates this.

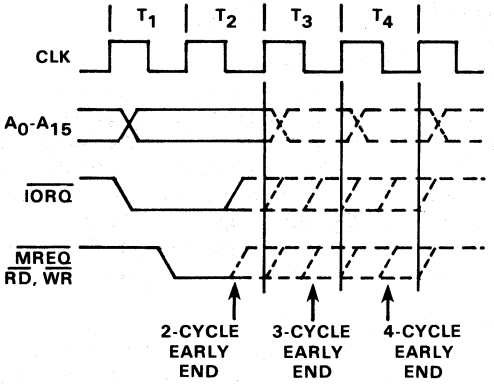
In the variable-cycle mode, unlike default timing, $\overline{IORQ}$ comes active one-half cycle before $\overline{MREQ}$, $\overline{RD}$ and $\overline{WR}$. $\overline{CE/WAIT}$ can be used to extend only the 3 or 4 T-cycle variable cycles. It is sampled at the falling edge of T_2 for 3- or 4-cycle memory cycles, and at the falling edge of T_3 for 4-cycle I/O cycles.

During transfers, data is latched on the clock edge causing the rising edge of $\overline{RD}$ and held until the end of the write cycle.

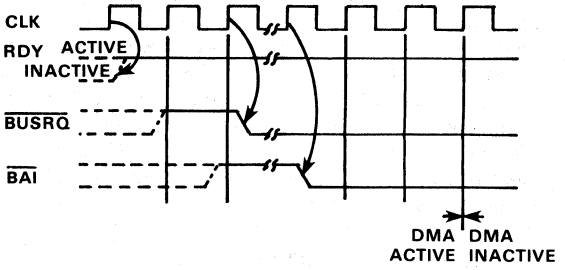
Bus Requests

Figure 15 illustrates the bus request and acceptance timing. The RDY line, which may be programmed active

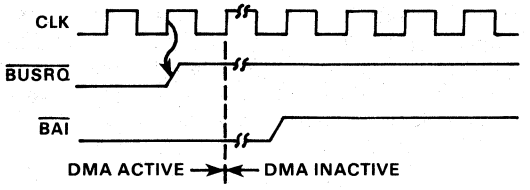
VARIABLE-CYCLE AND EDGE TIMING
Figure 14



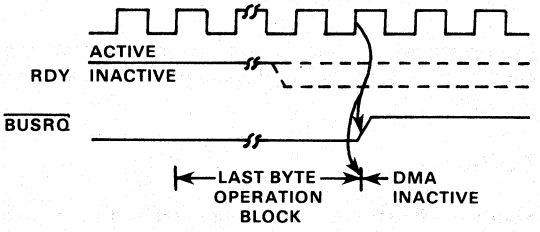
BUS REQUEST AND ACCEPTANCE
Figure 15



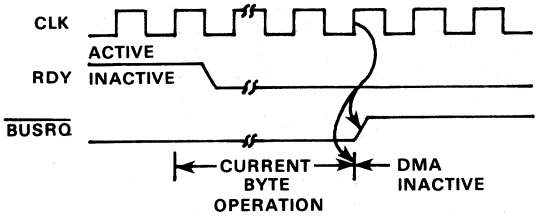
BUS RELEASE (BYTE-AT-A-TIME MODE)
Figure 16



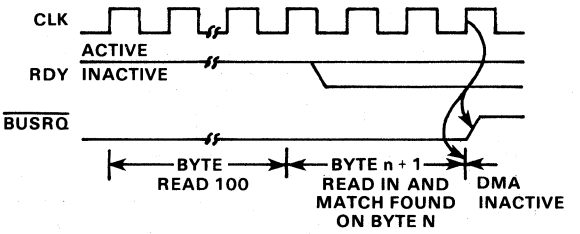
BUS RELEASE (CONTINUOUS MODE)
Figure 17



BUS RELEASE WHEN NOT READY (BURST MODE)
Figure 18



BUS RELEASE ON MATCH (BURST AND CONTINUOUS MODES)
Figure 19



High or Low, is sampled on every rising edge of CLK. If it is found to be active, and the bus is not in use by any other device, the following rising edge of CLK drives $\overline{BUSRQ}$ low. After receiving $\overline{BUSRQ}$, the CPU acknowledges on the $\overline{BAI}$ input either directly or through a multiple-DMA daisy chain. When a low is detected on $\overline{BAI}$ for two consecutive rising edges of CLK, the DMA will begin transferring data on the next rising edge of CLK.

Bus Release Byte-at-a-Time

In Byte-at-a-Time mode, $\overline{BUSRQ}$ is brought high on the rising edge of CLK prior to the end of each read cycle (search-only) or write cycle (transfer and transfer/search) as illustrated in Figure 16. This is done regardless of the state of RDY. There is no possibility of confusion when a Z80 CPU is used since the CPU cannot begin an operation until the following T-cycle. Most other CPUs are not bothered by this either, although note should be taken of it. The next bus request for the next byte will come after both $\overline{BUSRQ}$ and $\overline{BAI}$ have returned high.

Bus Release at End of Block

In Burst and Continuous modes, an end of block causes $\overline{BUSRQ}$ to go High usually on the same rising edge of CLK in which the DMA completes the transfer of the data block (Figure 17). The last byte in the block is transferred even if RDY goes inactive before completion of the last byte transfer.

Bus Release on Not Ready

In Burst Mode, when RDY goes inactive it causes $\overline{BUSRQ}$ to go High on the next rising edge of CLK after the completion of its current byte operation (Figure 18). The action on $\overline{BUSRQ}$ is thus somewhat delayed from action on the RDY line. The DMA always completes its current byte operation in an orderly fashion before releasing the bus.

By contrast, $\overline{BUSRQ}$ is not released in Continuous mode when RDY goes inactive. Instead, the DMA idles after completing the current byte operation, awaiting an active RDY again.

Bus Release on Match

If the DMA is programmed to stop on match in Burst or Continuous modes, a match causes $\overline{BUSRQ}$ to go inactive on the rising edge of CLK after the next byte following the match (Figure 19). Due to the pipelining scheme, matches are determined while the next byte is being read. Matches at End-of-Block are, therefore, actually matches to the byte immediately preceding the last byte in the block.

The RDY line can go inactive after the matching operation begins without affecting this bus-release timing.

Interrupts

Timings for interrupt acknowledge and return from interrupt are the same as timings for these in other Z80 peripherals.

Interrupt on RDY (interrupt before requesting bus) does not directly affect the $\overline{BUSRQ}$ line. Instead, the interrupt service routine must handle this by issuing the following commands to WR6:

1. Enable after Return From Interrupt (RETI) Command —Hex B7
2. Enable DMA—Hex 87
3. A RETI instruction that resets the IUS latch in the Z80 DMA

III

ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Operating Ambient Temperature Under Bias ..... As Specified Under "Ordering Information"
Storage Temperature ..... -65°C to +150°C
Voltage on any pin with respect to ground ..... -0.3V to +7V
Power Dissipation 1.5W

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

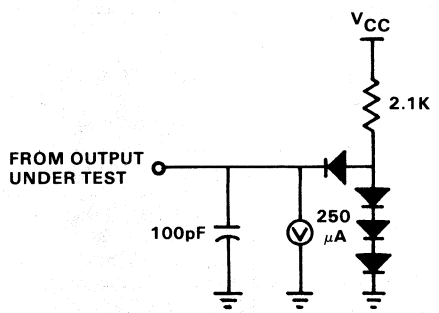
STANDARD TEST CONDITIONS

The characteristics below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin. Standard conditions are as follows:

- $+4.75 \leq V_{CC} \leq +5.25V$
- GND = 0V
- $0°C \leq T_A \leq +70°C$

All AC parameters assume a load capacitance of 100pF max. Timing references between two output signals assume a load difference of 50pF max.

Figure 20



DC CHARACTERISTICS

SYM	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
V_{ILC}	Clock Input Low Voltage	-0.3	0.80	V	
V_{IHC}	Clock Input High Voltage	V_{CC}-.6	5.5	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	

DC CHARACTERISTICS

SYM	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
V_{IH}	Input High Voltage	2.0	5.5	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL}=3.2mA for $\overline{BUSRQ}$ I_{OL}=2.0mA for all others
V_{OH}	Output High Voltage	2.4		V	I_{OH}=250μA
I_{CC}	Power Supply Current MK3883 MK3883-4		 150 200	 mA mA	
I_{LI}	Input Leakage Current		±10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float		10	μA	V_{OUT}=2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float		-10	μA	V_{OUT}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode		±10	μA	$0 \leq V_{IN} \leq V_{CC}$

V_{CC} = 5V ± 5% unless otherwise specified, over specified temperature range.

CAPACITANCE

SYM	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
C	Clock Capacitance		35	pF	Unmeasured Pins
C_{IN}	Input Capacitance		10	pF	Returned to Ground
C_{OUT}	Output Capacitance		10	pF	

f = 1MHz, over specified temperature range

INACTIVE STATE AC CHARACTERISTICS

(See Figure 21)

NO	SYM	PARAMETER	MK3883		MK3883-4		UNIT
			MIN	MAX	MIN	MAX	
1	TcC	Clock Cycle Time	400	4000	250	4000	ns
2	TwCh	Clock Width (High)	170	2000	105	2000	ns
3	TwCl	Clock Width (Low)	170	2000	105	2000	ns
4	TrC	Clock Rise Time		30		30	ns
5	TfC	Clock Fall Time		30		30	ns
6	Th	Hold Time for Any Specified Setup Time	0		0		ns
7	TsC(Cr)	$\overline{IORQ}$, $\overline{WR}$, $\overline{CE}$ ↓ to Clock ↑ Setup	280		145		ns
8	TdDO(RDf)	$\overline{RD}$ to Data Output Delay		500		380	ns
9	TsWM(Cr)	Data In to Clock ↑ Setup ($\overline{WR}$ or $\overline{M1}$)	50		50		ns
10	TdCf(DO)	$\overline{IORQ}$ ↓ to Data Out Delay (INTA Cycle)		340		160	ns

INACTIVE STATE AC CHARACTERISTICS (Continued)

NO	SYM	PARAMETER	MK3883 MIN	MK3883 MAX	MK3883-4 MIN	MK3883-4 MAX	UNIT
11	TdRD(Dz)	$\overline{RD}\uparrow$ to Data Float Delay (output buffer disable)		160		110	ns
12	TslEI(IORQ)	IEI ↓ to $\overline{IORQ}$ ↓ Setup (INTA Cycle)	140		140		ns
13	TdIEOr(IEIr)	IEI ↑ to IEO ↑ Delay		210		160	ns
14	TdIEOf(IEIf)	IEI ↓ to IEO ↓ Delay		190		130	ns
15	TdM1(IEO)	$\overline{M1}\downarrow$ to IEO↓ Delay (interrupt just prior to $\overline{M1}\downarrow$)		300		190	ns
16	TsM1f(Cr)	$\overline{M1}\downarrow$ to Clock ↑ Setup	210		90		ns
17	TsM1r(Cf)	$\overline{M1}\uparrow$ to Clock ↓ Setup	20		0		ns
18	TsRD(C)	$\overline{RD}\downarrow$ to Clock ↑ Setup ($\overline{M1}$ Cycle)	240		115		ns
19	TdI(INT)	Interrupt Cause to $\overline{INT}$ ↓ Delay ($\overline{INT}$ generated only when DMA is inactive)		500		500	ns
20	TdBAIr (BAOr)	$\overline{BAI}\uparrow$ to $\overline{BAO}\uparrow$ Delay		200		150	ns
21	TdBAIf (BAOf)	$\overline{BAI}\downarrow$ to $\overline{BAO}\downarrow$ Delay		200		150	ns
22	TsRDY(Cr)	RDY Active to Clock ↑ Setup	150		100		ns

ACTIVE STATE AC CHARACTERISTICS
(See Figure 22)

NO	SYM	PARAMETER	MK3883 MIN(ns)	MK3883 MAX(ns)	MK3883-4 MIN(ns)	MK3883-4 MAX(ns)
1	TcC	Clock Cycle Time	400		250	
2	TwCh	Clock Width (High)	180	2000	110	2000
3	TwCl	Clock Width (Low)	180	2000	110	2000
4	TrC	Clock Rise Time		30		30
5	TfC	Clock Fall Time		30		30
6	TdA	Address Output Delay		145		110
7	TdC(Az)	Clock ↑ to Address Float Delay		110		90
8	TsA(MREQ)	Address to $\overline{MREQ}$ ↓ Setup (Memory Cycle)	(2)+(5)-75		(2)+(5)-75	
9	TsA(IRW)	Address Stable to $\overline{IORQ}$, $\overline{RD}$, $\overline{WR}$ ↓ Setup (I/O Cycle)	(1)-80		(1)-70	
*10	TdRW(A)	$\overline{RD}$, $\overline{WR}$ ↑ to Addr. Stable Delay	(3)+(4)-40		(3)+(4)-50	
*11	TdRW(Az)	$\overline{RD}$, $\overline{WR}$ ↑ to Addr. Float	(3)+(4)-60		(3)+(4)-45	
12	TdCf(DO)	Clock ↓ to Data Out Delay		230		150

ACTIVE STATE AC CHARACTERISTICS

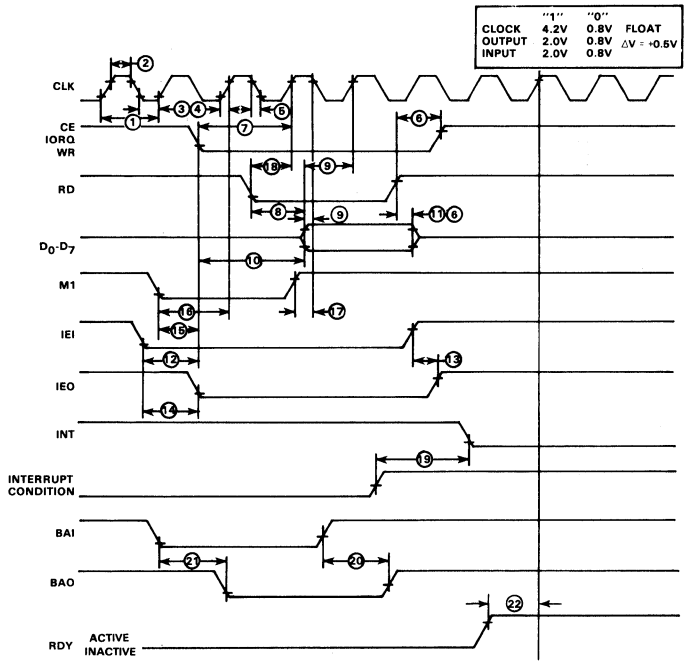
NO	SYM	PARAMETER	MK3883 MIN(ns)	MK3883 MAX(ns)	MK3883-4 MIN(ns)	MK3883-4 MAX(ns)
*13	TdCr(Dz)	Clock↑ to Data Float Delay (Write Cycle)		90		90
14	TsDI(Cr)	Data In to Clock↑ Setup (Read cycle when rising edge ends read)	50		35	
15	TsDI(Cf)	Data In to Clock↓ Setup (Read cycle when falling edge ends read)	60		50	
*16	TsDO(WfM)	Data Out to $\overline{WR}$↓ Setup (Memory Cycle)	(1)-210		(1)-170	
17	TsDO(WfI)	Data Out to $\overline{WR}$↓ Setup (I/O cycle)	100		100	
*18	TdWr(DO)	$\overline{WR}$↑ to Data Out Delay	(3)+(4)-80		(3)+(4)-70	
19	Th	Hold Time for Any Specified Setup Time	0		0	
*20	TdCr(Mf)	Clock↑ to $\overline{MREQ}$↓ Delay		100		85
21	TdCf(Mf)	Clock↓ to $\overline{MREQ}$↓ Delay		100		85
22	TdCr(Mr)	Clock↑ to $\overline{MREQ}$↑ Delay		100		85
23	TdCf(Mr)	Clock↓ to $\overline{MREQ}$↑ Delay		100		85
24	TwM1	$\overline{MREQ}$ Low Pulse Width	(1)-40		(1)-30	
*25	TwMh	$\overline{MREQ}$ High Pulse Width	(2)+(5)-30		(2)+(5)-20	
26	TdCr(If)	Clock↑ to $\overline{IORQ}$↓ Delay		90		75
27	TdCf(If)	Clock↓ to $\overline{IORQ}$↓ Delay		110		85
28	TdCr(Ir)	Clock↑ to $\overline{IORQ}$↑ Delay		100		85
*29	TdCf(Ir)	Clock↓ to $\overline{IORQ}$↑ Delay		110		85
30	TdCr(Rf)	Clock↑ to $\overline{RD}$↓ Delay		100		85
31	TdCf(Rf)	Clock↓ to $\overline{RD}$↓ Delay		130		95
32	TdCr(Rr)	Clock↑ to $\overline{RD}$↑ Delay		100		85
33	TdCf(Rr)	Clock↓ to $\overline{RD}$↑ Delay		110		85
34	TdCr(Wf)	Clock↑ to $\overline{WR}$↓ Delay		80		65
35	TdCf(Wf)	Clock↓ to $\overline{WR}$↓ Delay		90		80
*36	TdCr(Wr)	Clock↑ to $\overline{WR}$↑ Delay		100		80
37	TdCf(Wr)	Clock↓ to $\overline{WR}$↑ Delay		100		80
38	TwWI	$\overline{WR}$ Low Pulse Width	(1)-40		(1)-30	
39	TsWA(Cf)	$\overline{WAIT}$ to Clock↓ Setup	70		70	
40	TdCr(B)	Clock↑ to $\overline{BUSRQ}$ Delay		100		100
41	TdCr(Iz)	Clock↑ to $\overline{IORQ}$, $\overline{MREQ}$, $\overline{RD}$, $\overline{WR}$ Float Delay		100		80

NOTES:

1. Numbers in parentheses are other parameter-numbers in this table; their values should be substituted in equations.
2. All equations imply DMA default (standard) timing.
3. Data must be enabled onto data bus when RD is active.
4. Asterisk(*) before parameter number means the parameter is not illustrated in the AC Timing Diagrams.

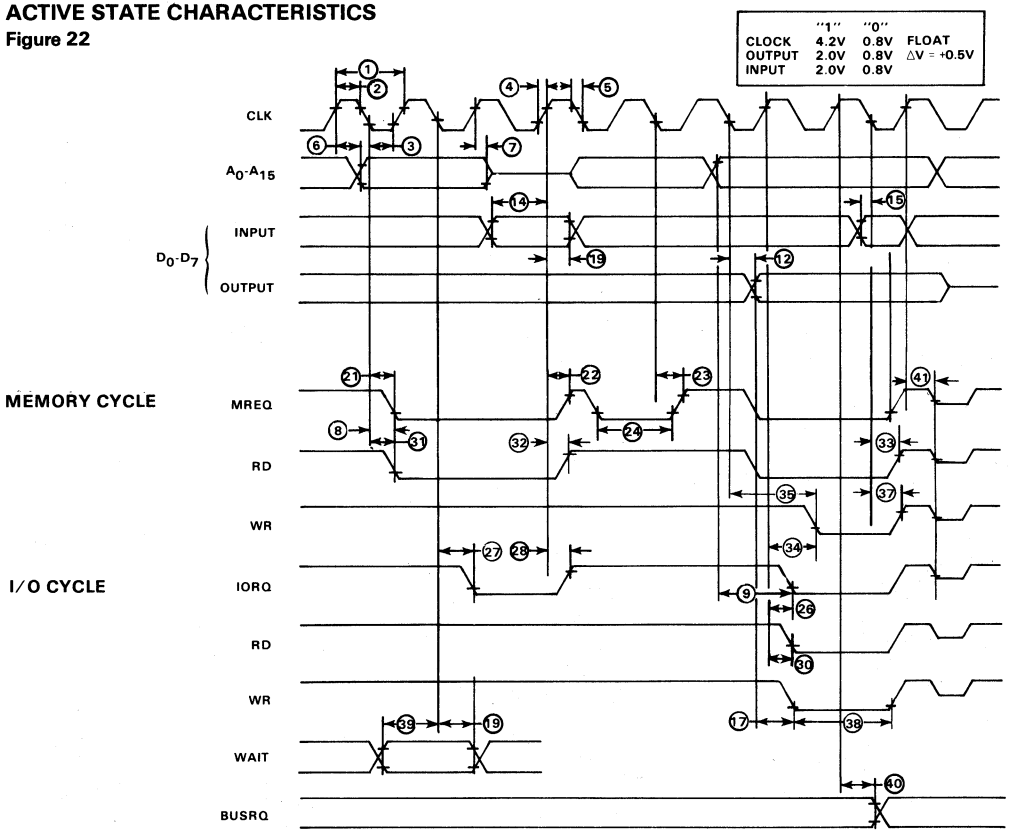
INACTIVE STATE CHARACTERISTICS

Figure 21



ACTIVE STATE CHARACTERISTICS

Figure 22



III

ORDERING INFORMATION

PART NO.	PACKAGE TYPE	MAX CLOCK FREQUENCY	TEMPERATURE RANGE
MK3883N MK3883P	Z80-DMA Plastic Z80-DMA Ceramic	2.5 MHz 2.5 MHz	0°C to +70°C 0°C to +70°C
MK3883N-10 MK3883P-10	Z80-DMA Plastic Z80-DMA Ceramic	2.5 MHz 2.5 MHz	–40°C to +85°C –40°C to +85°C
MK3883N-4 MK3883P-4	Z80A-DMA Plastic Z80A-DMA Ceramic	4 MHz 4 MHz	0°C to +70°C 0°C to +70°C

MK3884/5/7 SERIAL INPUT/OUTPUT

TABLE OF CONTENTS

1.0 GENERAL INFORMATION

1.1 INTRODUCTION

The Mostek Z80-SIO (Serial Input/Output) is a dual-channel, multi-function peripheral component designed to satisfy a wide variety of serial data communications requirements in microcomputer systems. Its basic function is a serial-to-parallel, parallel-to-serial converter/controller, but, within that role, it is configurable by systems software so its "personality" can be optimized for a given serial data communications application.

The Z80-SIO is capable of handling asynchronous and synchronous byte-oriented protocols, such as IBM Bisync, and synchronous bit-oriented protocols, such as HDLC and IBM SDLC. This versatile device can also be used to support virtually any other serial protocol for applications other than data communications (cassette or floppy disk interface, for example).

The Z80-SIO can generate and check CRC codes in any synchronous mode and can be programmed to check data integrity in various modes. The device also has facilities for modem controls in both channels. In applications where these controls are not needed, the modem controls can be used for general-purpose I/O.

1.2 STRUCTURE

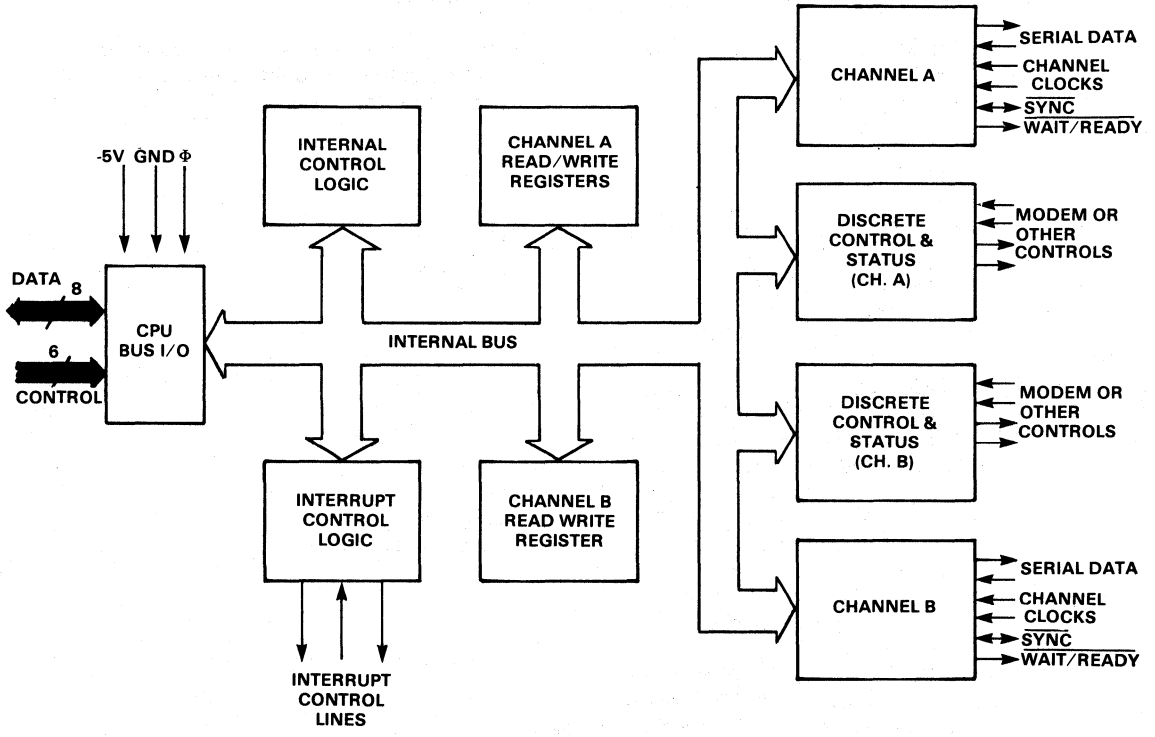
- N-channel silicon-gate depletion-load technology
- 40-pin DIP
- Single 5V power supply
- Single-phase 5V clock
- All inputs and outputs TTL compatible

1.3 FEATURES

- Two independent full-duplex channels
- Data rates in synchronous or isosynchronous modes:
  0-550K bits/second with 2.5 MHz system clock rate
  0-800K bits/second with 4.0 MHz system clock rate
- Receiver data registers quadruply buffered; transmitter doubly buffered.
- Asynchronous features:
  - 5, 6, 7 or 8 bits/character
  - 1, 1 1/2 or 2 stop bits
  - Even, odd or no parity
  - x1, x16, x32 and x64 clock modes
  - Break generation and detection
  - Parity, overrun and framing error detection
  - Binary synchronous features:
    - Internal or external character synchronization
    - One or two sync characters in separate registers
    - Automatic sync character insertion
    - CRC generation and checking
  - HDLC and IBM SDLC-features:
    - Abort sequence generation and detection
    - Automatic zero insertion and deletion
    - Automatic flag insertion between messages
    - Address field recognition
    - I-field residue handling
    - Valid receive messages protected from overrun
    - CRC generation and checking
  - Separate modem control inputs and outputs for both channels
  - CRC-16 or CRC-CCITT block check
  - Daisy-Chain Priority interrupt logic provides automatic interrupt vectoring without external logic
  - Modem status can be monitored

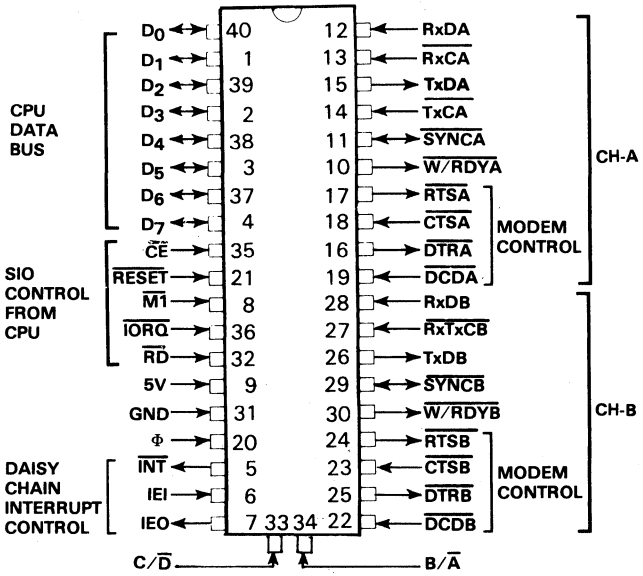
Z80-SIO BLOCK DIAGRAM

Figure 1.0



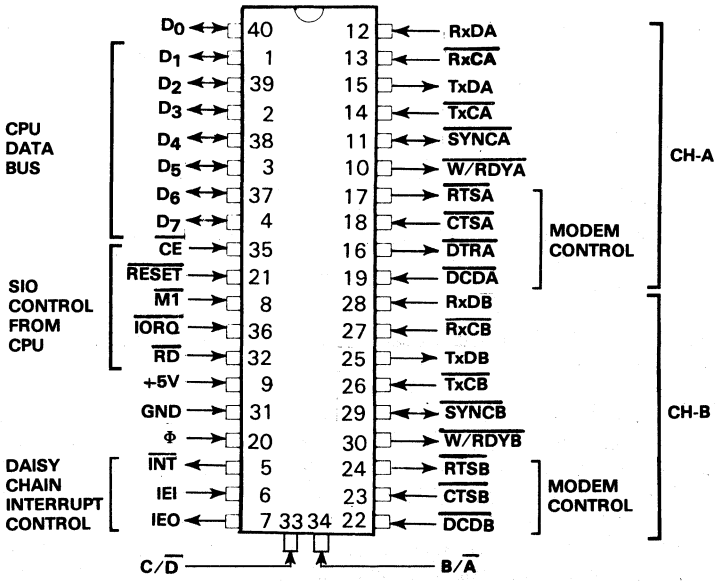
MK3884 PIN CONFIGURATION

Figure 1.1



MK3885 PIN CONFIGURATION

Figure 1.2



1.4 PIN DESCRIPTION

D_0-D_7. System Data Bus (bidirectional, 3-state). The system data bus transfers data and commands between the CPU and the Z80-SIO. D_0 is the least significant bit.

B/$\overline{A}$. Channel A Or B Select (input, High selects Channel B). This input defines which channel is accessed during a data transfer between the CPU and the Z80-SIO. Address bit A_0 from the CPU is often used for the selection function.

C/$\overline{D}$. Control Or Data Select (input, High selects Control). This input defines the type of information transfer performed between the CPU and the Z80-SIO. A High at this input, during a CPU write to the Z80-SIO, causes the information on the data bus to be interpreted as a command for the channel selected by B/$\overline{A}$. A Low at C/$\overline{D}$ means that the information on the data bus is data. Address bit A_1 is often used for this function.

$\overline{CE}$. Chip Enable (input, active Low). A Low level at this input enables the Z80-SIO to accept command or data inputs from the CPU during a write cycle, or to transmit data to the CPU during a read cycle.

Φ. System Clock (input). The Z80-SIO uses the standard Z80A System Clock to synchronize internal signals. This is a single-phase clock.

$\overline{M1}$. Machine Cycle One (input from Z80-CPU, active Low). When $\overline{M1}$ is active and $\overline{RD}$ is also active, the Z80-CPU is fetching an instruction from memory. When $\overline{M1}$ is active, while $\overline{IORQ}$ is active, the Z80-SIO accepts $\overline{M1}$ and $\overline{IORQ}$ as an interrupt acknowledge if the Z80-SIO is the highest priority device that has interrupted the Z80-CPU.

$\overline{IORQ}$. Input/Output Request (input from CPU, active Low). $\overline{IORQ}$ is used in conjunction with B/$\overline{A}$, C/$\overline{D}$, $\overline{CE}$ and $\overline{RD}$ to transfer commands and data between the CPU and the Z80-SIO. When $\overline{CE}$, $\overline{RD}$ and $\overline{IORQ}$ are all active, the channel selected by B/$\overline{A}$ transfers data to the CPU (a read operation). When $\overline{CE}$ and $\overline{IORQ}$ are active, but $\overline{RD}$ is inactive, the channel selected by B/$\overline{A}$ is written to by the CPU with either data or control information, as specified by C/$\overline{D}$. As mentioned previously, if $\overline{IORQ}$ and $\overline{M1}$ are active simultaneously, the CPU is acknowledging an interrupt and the Z80-SIO automatically places its interrupt vector on the CPU data bus, if it is the highest priority device requesting an interrupt.

$\overline{RD}$. Read Cycle Status. (input from CPU, active Low). If $\overline{RD}$ is active, a memory or I/O read operation is in progress. $\overline{RD}$ is used with B/$\overline{A}$, $\overline{CE}$ and $\overline{IORQ}$ to transfer data from the Z80-SIO to the CPU.

$\overline{RESET}$. Reset (input, active Low). A Low $\overline{RESET}$ disables both receivers and transmitters, forces TxDA and TxDB marking, forces the modem controls High and disables all interrupts. The control registers must be rewritten after the Z80-SIO is reset and before data is transmitted or received.

IEI. Interrupt Enable In (input, active High). This signal is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO. Interrupt Enable Out (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this Z80-SIO. Thus, this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

$\overline{INT}$. Interrupt Request (output, open drain, active Low). When the Z80-SIO is requesting an interrupt, it pulls $\overline{INT}$ Low.

$\overline{W/RDYA}$, $\overline{W/RDYB}$. Wait/Ready A, Wait/Ready B (outputs, open drain when programmed for Wait function, driven High and Low when programmed for Ready function). These dual-purpose outputs may be programmed as Ready lines for a DMA controller or as Wait lines that synchronize the CPU to the Z80-SIO data rate. The reset state is open drain.

$\overline{CTSA}$, $\overline{CTSB}$. Clear To Send (inputs, active Low). When programmed as Auto Enables, a Low on these inputs enables the respective transmitter. If not programmed as Auto Enables, these inputs may be programmed as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow-risetime inputs. The Z80-SIO detects pulses on these inputs and interrupts the CPU on both logic level transitions. The Schmitt-trigger inputs do not guarantee a specified noise-level margin.

$\overline{DCDA}$, $\overline{DCDB}$. Data Carrier Detect (inputs, active Low). These signals are similar to the CTS inputs, except they can be used as receiver enables.

RxDA, RxDB. Receive Data (inputs, active High).

TxDA, TxDB. Transmit Data (outputs, active High).

$\overline{RxCA}$, $\overline{RxCB}$*. Receiver Clocks (inputs). See the following section on bonding options. The Receive Clocks may be 1, 16, 32 or 64 times the data rate in asynchronous modes. Receive data is sampled on the rising edge of $\overline{RxC}$.

$\overline{TxCA}$, $\overline{TxCB}$*. Transmitter Clocks (inputs). See section on bonding options. In asynchronous modes, the Transmitter clocks may be 1, 16, 32 or 64 times the data rate. The multiplier for the transmitter and the receiver must be the same. Both the $\overline{TxC}$ and $\overline{RxC}$ inputs are Schmitt-trigger buffered for relaxed rise-and fall-time requirements (no noise margin is specified). TxD changes on the falling edge of $\overline{TxC}$.

$\overline{RTSA}$, $\overline{RTSB}$. Request To Send (outputs, active Low). When the RTS bit is set, the RTS output goes low. When the RTS bit is reset in the Asynchronous mode, the output goes High after the transmitter is empty. In Synchronous modes, the $\overline{RTS}$ pin strictly follows the state of the RTS bit. Both pins can be used as general purpose outputs.

$\overline{DTRA}$, $\overline{DTRB}$. Data Terminal Ready (outputs, active Low). See note on bonding options. These outputs follow the state programmed into the DTR bit. They can also be programmed as general-purpose outputs.

$\overline{SYNC\ A}$, $\overline{SYNC\ B}$. Synchronization (inputs/outputs, active Low). These pins can act either as inputs or outputs. In the Asynchronous Receive mode, they are inputs similar to $\overline{CTS}$ and $\overline{DCD}$. In this mode, the transitions on these lines affect the state of the Sync/Hunt status bits in RRO. In the External Sync mode, these lines also act as inputs. When external synchronization is achieved, $\overline{SYNC}$ must be driven Low on the second rising edge of $\overline{RxC}$ after that rising edge of $\overline{RxC}$, on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the $\overline{SYNC}$ input. Once $\overline{SYNC}$ is forced Low, it is wise to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. Character assembly begins on the rising edge of $\overline{RxC}$ that immediately precedes the falling edge of $\overline{SYNC}$ in the External Sync mode.

In the Internal Synchonrization mode (Monosync and Bisync), these pins act as outputs that are active during the part of the receive clock ($\overline{RxC}$) cycle in which sync characters are recognized. The sync condition is not latched, so these outputs are active each time a sync pattern is recognized, regardless of character boundaries.

1.5 BONDING OPTIONS

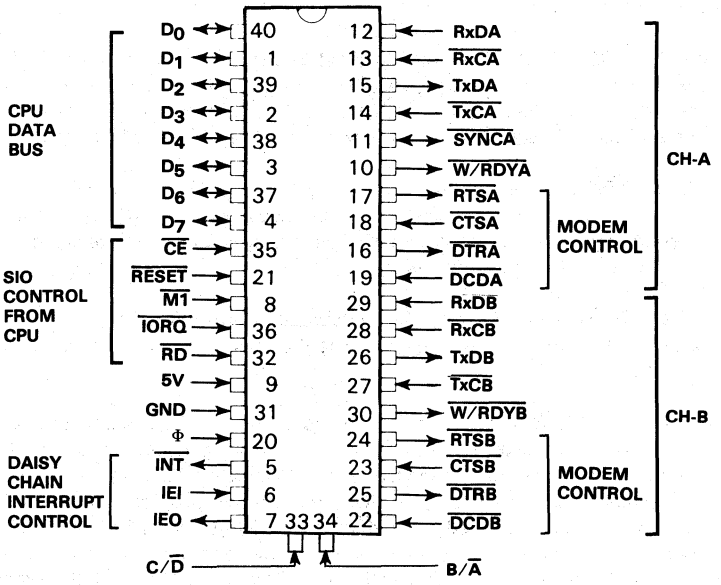
The constraints of a 40-pin package make it impossible to bring out the Receive Clock, Transmit Clock, Data Terminal Ready and Sync signals for both channels. Therefore, Channel B must sacrifice a signal or have two signals bonded together. Since user requirements vary, three bondings options are offered:

MK3884 Z80-SIO has all four signals, but $\overline{TxCB}$ and $\overline{RxCB}$ are bonded together (Fig. 1.1).
MK3885 Z80-SIO sacrifices $\overline{DTRB}$ and keeps $\overline{TxCB}$, $\overline{RxCB}$ and $\overline{SYNCB}$ (Fig. 1.2).
MK3887 Z80-SIO sacrifices $\overline{SYNCB}$ and keeps $\overline{TxCB}$, $\overline{RxCB}$ and $\overline{DTRB}$ (Fig. 1.3).

*These clocks may be directly driven by the Z80-CTC (Counter Timer Circuit) for fully programmable baud rate generation.

MK3887 PIN CONFIGURATION

Figure 1.3



2.0 ARCHITECTURE

2.1 INTRODUCTION

The device internal structure includes a Z80-CPU interface, internal control and interrupt logic and two full-duplex channels. Associated with each channel are read and write registers and discrete control and status logic that provide the interface to modems or other external devices.

The read and write register group includes five 8-bit control registers, two sync-character registers and two status registers. The interrupt vector is written into an additional 8-bit register (Write Register 2) in Channel B that may be read through Read Register 2 in Channel B. The registers for both channels are designated in the text as follows:

WR0-WR7 -- Write Registers 0 through 7
RR0-RR2 -- Read Registers 0 through 2

The bit assignment and functional grouping of each register is configured to simplify and organize the programming process. Table 2.1 illustrates the functions assigned to each read or write register.

III

FUNCTIONAL ASSIGNMENTS OF READ AND WRITE REGISTERS
Table 2.1

REGISTER	FUNCTION
WR0	Register pointers, CRC initialize, initialization commands for the various modes, etc.
WR1	Transmit/Receive interrupt and data transfer mode definition
WR2	Interrupt vector (Channel B only)
WR3	Receive parameters and controls
WR4	Transmit/Receive miscellaneous parameters and modes
WR5	Transmit parameters and controls
WR6	Sync character of SDLC address field
WR7	Sync character of SDLC flag
	(a) Write Register Functions
RR0	Transmit/Receive buffer status, interrupt status and external status
RR1	Special Receive Condition status
RR2	Modified interrupt vector (Channel B only)
	(b) Read Register Functions

The logic for both channels provides formats, synchronization and validation for data transferred to and from the channel interface. The modem control inputs, Clear to Send ($\overline{CTS}$) and Data Carrier Detect ($\overline{DCD}$), are monitored by the discrete control logic under program control. All the modem control signals are general purpose in nature and can be used for functions other than modem control.

For automatic interrupt vectoring, the interrupt control logic determines which channel and which device within the channel has the highest priority. Priority is fixed with Channel A assigned a higher priority than Channel B; Receive, Transmit and External/Status interrupts are prioritized in that order within each channel.

2.2 DATA PATH

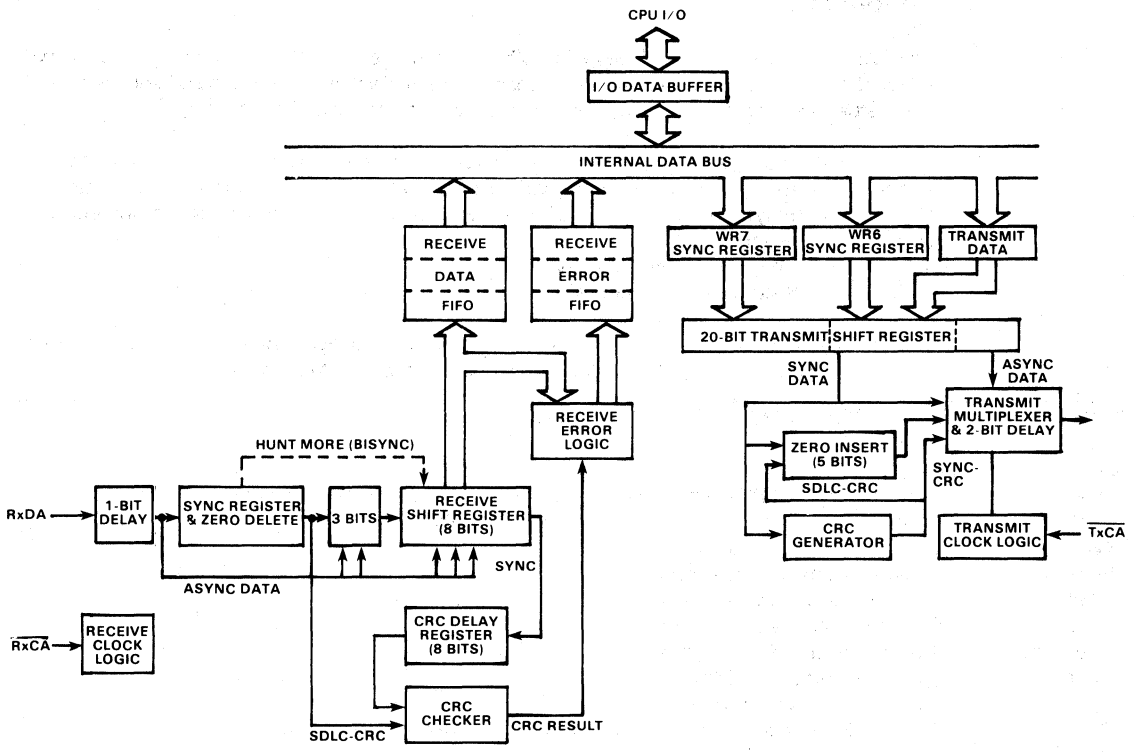
The transmit and receive data paths for each channel are shown in Figure 2.1. The receiver has three 8-bit buffer registers in a FIFO arrangement (to provide a 3-byte delay) in addition to the 8-bit receive shift register. This arrangement creates additional time for the CPU to service an interrupt at the beginning of a block of high-speed data. The receive error FIFO stores parity and framing errors and other types of status information for each of the three bytes in the receive data FIFO.

Incoming data is routed through one of several paths depending on the mode and character length. In the Asynchronous mode, serial data is entered into the 3-bit buffer if it has a character length of seven or eight bits, or is entered into the 8-bit receive shift register if it has a length of five or six bits.

In the Synchronous mode, however, the data path is determined by the phase of the receive process currently in operation. A Synchronous Receive operation begins with the receiver in the Hunt phase, during which the receiver searches the incoming data stream for a bit pattern that matches the

TRANSMIT AND RECEIVE DATA PATH

Figure 2.1



preprogrammed sync characters (or flags in the SDLC mode). If the device is programmed for Monosync Hunt, a match is made with a single sync character stored in WR7. In Bisync Hunt, a match is made with dual sync characters stored in WR6 and WR7.

In either case, the incoming data passes through the receive sync registers and is compared against the programmed sync character in WR6 or WR7. In the Monosync mode, a match between the sync character programmed into WR7 and the character assembled in the receive sync register establishes synchronization.

In the Bisync mode, however, incoming data is shifted to the receive shift register while the next eight bits of the message are assembled in the receive sync register. The match between the assembled character in the receive sync registers with the programmed sync character in WR6, and WR7 establishes synchronization. Once synchronization is established, incoming data bypasses the receive sync register and directly enters the 3-bit buffer.

In the SDLC mode, incoming data first passes through the receive sync register, which continuously monitors the receive data stream and performs zero deletion when indicated. Upon receiving five contiguous I's, the sixth bit is inspected. If the sixth bit is a 0, it is deleted from the data stream. If the sixth bit is a 1, the seventh bit is inspected. If that bit is a 0, a Flag sequence has been received; if it is a 1, an Abort sequence has been received.

The reformatted data enters the 3-bit buffer and is transferred to the receive shift register. Note that the SDLC receive operation also begins in the Hunt phase, during which the Z80-SIO tries to match the assembled character in the receive shift register with the flag pattern in WR7. Once the first flag character is recognized, all subsequent data is routed through the same path, regardless of character length.

Although the same CRC checker is used for both SDLC and synchronous data, the data path taken for each mode is different. In Bisync protocol, a byte-oriented operation requires that the CPU decide to include the data character in CRC. To allow the CPU ample time to make this decision, the Z80-SIO provides an 8-bit delay for sychronous data. In the SDLC mode, no delay is provided since the Z80-SIO contains logic that determines the bytes on which CRC is calculated.

The transmitter has an 8-bit transmit data register that is loaded from the internal data bus and a 20-bit transmit shift register that can be loaded from WR6, WR7 and the transmit data register. WR6 and WR7 contain sync characters in the Monosync or Bisync modes or address field (one character long) and flag, respectively, in the SDLC mode. During Synchronous modes, information contained in WR6 and WR7 is loaded into the transmit shift register at the beginning of the message and, as a time filler, in the middle of the message if a Transmit Underrun condition occurs. In the SDLC mode, the flags are loaded into the transmit shift register at the beginning and end of message.

Asynchronous data in the transmit shift register is formatted with start and stop bits and is shifted out to the transmit multiplexer at the selected clock rate.

Synchronous (Monosync or Bisync) data is shifted out to the transmit multiplexer and also to the CRC generator at the x 1 clock rate.

SDLC/HDLC data is shifted out through the zero insertion logic, which is disabled while the flags are being sent. For all other fields (address, control and frame check) a 0 is inserted following five contiguous 1's in the data stream. The CRC generator result for SDLC data is also routed through the zero insertion logic.

2.3 FUNCTIONAL DESCRIPTION

The functional capabilities of the Z80-SIO can be described from two different points of view: as a data communications device, it transmits and receives serial data, and meets the requirements of various data communications protocols; as a Z80 family peripheral, it interacts with the Z80-CPU and other Z80 peripheral circuits, and shares their data, address and control busses, as well as being a part of the Z80 interrupt structure. As a peripheral to other microprocessors, the Z80-SIO offers valuable features such as non-vectored interrupts, polling, and simple handshake capabilities.

The first part of the following functional description describes the interaction between the CPU and Z80-SIO; the second part introduces its data communications capabilities.

2.3.1 I/O CAPABILITIES

The Z80-SIO offers the choice of Polling, Interrupt (vectored or non-vectored) and Block Transfer modes to transfer data, status, and control information to and from the CPU. The Block Transfer mode can be implemented under CPU or DMA control.

Polling. The Polled mode avoids interrupts. Status registers RR0 and RR1 are updated at appropriate times for each function being performed (for example, CRC Error status valid at the end of the message). All the interrupt modes of the Z80-SIO must be disabled to operate the device in a polled environment.

While in its Polling sequence, the CPU examines the status contained in RR0 for each channel; the RR0 status bits serve as an acknowledge to the Poll inquiry. The two RR0 status bits D_0 and D_2 indicate that a receive or transmit data transfer is needed. The status also indicates Error or other special status conditions (see "Z80-SIO Programming"). The Special Receive Condition status continued in RR1 does not have to be read in a Polling sequence because the status bits in RR1 are accompanied by a Receive Character Available status in RR0.

Interrupts. The Z80-SIO offers an elaborate interrupt scheme to provide fast interrupt response in real-time applications. As mentioned earlier, Channel B registers WR2 and RR2 contain the interrupt vector that points to an interrupt service routine in the memory. To service operations in both channels and to eliminate the necessity of writing a status analysis

routine, the Z80-SIO can modify the interrupt vector in RR2 so it points directly to one of eight interrupt service routines. This is done under program control by setting a program bit (WR1, D_2) in Channel B called "Status Affects Vector." When this bit is set, the interrupt vector in WR2 is modified according to the assigned priority of the various interrupting conditions. The table in the Write Register 1 description (Z80-SIO Programming section) shows the modification details.

Transmit interrupts, Receive interrupts and External/Status interrupts are the main sources of interrupts (Figure 5). Each interrupt source is enabled under program control with Channel A having a higher priority than Channel B, and with Receiver, Transmit and External/Status interrupts prioritized in that order within each channel. When the Transmit interrupt is enabled, the CPU is interrupted by the transmit buffer becoming empty. (This implies that the transmitter must have had a data character written into it so it can become empty.) When enabled, the receiver can interrupt the CPU in one of three ways:

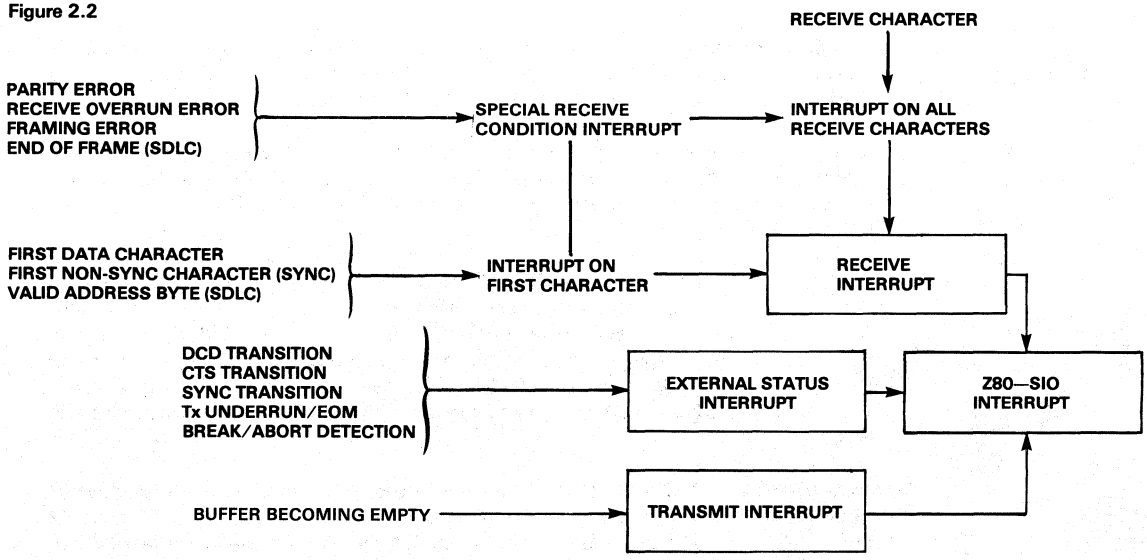
Interrupt on first receive character
Interrupt on all receive characters
Interrupt on a Special Receive condition

Interrupt On First Character is typically used with the Block Transfer mode.

Interrupt On All Receive Characters has the option of modifying the interrupt vector in the event of a parity error. The Special Receive Condition interrupt can occur on a character or message basis (End Of Frame interrupt in SDLC, for example). The Special Receive condition can cause an interrupt only if the Interrupt On First Receive Character or Interrupt On All Receive Characters mode is selected. In Interrupt On First Receive Character, an interrupt can occur from Special Receive conditions (except parity Error) after the first receive character interrupt (example: Receive Overrun interrupt).

The main function of the External/Status interrupt is to monitor the signal transitions of the $\overline{CTS}$, $\overline{DCD}$ and $\overline{SYNC}$ pins; however, an External/Status interrupt is also caused by a Transmit Underrun condition or by the detection of a Break (Asynchronous mode) or Abort (SDLC mode) sequence in the data stream. The interrupt caused by the Break/Abort sequence has a special feature that allows the Z80-SIO to interrupt when the Break/Abort sequence is detected or terminated. The feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the Break/Abort conditon in external logic.

INTERRUPT STRUCTURE
Figure 2.2



CPU/DMA Block Transfer. The Z80-SIO provides a Block Transfer mode to accommodate block transfer functions and DMA controllers (Z80-DMA or other designs). The Block Transfer mode uses the $\overline{\text{WAIT/READY}}$ output in conjunction with the Wait/Ready bits of Write Register 1. The $\overline{\text{WAIT/READY}}$ output can be defined under software control as a WAIT line in the CPU Block Transfer mode or as a READY line in the DMA Block Transfer mode.

To a DMA controller, the Z80-SIO READY output indicates that the Z80-SIO is ready to transfer data to or from memory. To the CPU, the WAIT output indicates that the Z80-SIO is not ready to transfer data, thereby requesting the CPU to extend the I/O cycle. The programming of bits 5, 6 and 7 of Write Register 1 and the logic states of the $\overline{\text{WAIT/READY}}$ line are defined in the Write Register 1 description (Z80-SIO Programming section.)

2.3.2 DATA COMMUNICATIONS CAPABILITIES

In addition to the I/O capabilities previously discussed, the Z80-SIO provides two independent full-duplex channels as well as Asynchronous, Synchronous and SDLC (HDLC) operational modes. These modes facilitate the implementation of commonly used data communications protocols.

The specific features of these modes are described in the following sections. To preserve the independence and completeness of each section, some information common to all modes is repeated.

3.0 ASYNCHRONOUS OPERATION

3.1 INTRODUCTION

To receive or transmit data in the Asynchronous mode, the Z80-SIO must be initialized with the following parameters: character length, clock rate, number of stop bits, even or odd parity, interrupt mode, and receiver or transmitter enable. The parameters are loaded into the appropriate write registers by the system program. WR4 parameters must be issued before WR1, WR3, and WR5 parameters or commands.

If the data is transmitted over a modem or RS232C interface, the $\overline{\text{REQUEST TO SEND}}$ ($\overline{\text{RTS}}$) and $\overline{\text{DATA TERMINAL READY}}$ ($\overline{\text{DTR}}$) outputs must be set along with the Transmit Enable bit. Transmission cannot begin until the Transmit Enable bit is set.

The Auto Enables feature allows the programmer to send the first data character of the message to the Z80-SIO without waiting for $\overline{\text{CTS}}$. If the Auto Enables bit is set, the Z80-SIO will wait for the $\overline{\text{CTS}}$ pin to go Low before it begins data transmission. $\overline{\text{CTS}}$, $\overline{\text{DCD}}$, and $\overline{\text{SYNC}}$ are general-purpose I/O lines that may be used for functions other than their labeled purposes. If $\overline{\text{CTS}}$ is used for another purpose, the Auto Enables Bit must be programmed to 0.

Figure 3.1 illustrates asynchronous message formats. Table 3.1 shows WR3, WR4, and WR5 with bits set to indicate the applicable modes, parameters, and commands in asynchronous modes. WR2 (Channel B only) stores the interrupt vector and WR1 defines the interrupt modes and data transfer modes. WR6 and WR7 are not used in asynchronous modes. Table 3.2 shows the typical program steps that implement a full-duplex receive/transmit operation in either channel.

3.2 ASYNCHRONOUS TRANSMIT

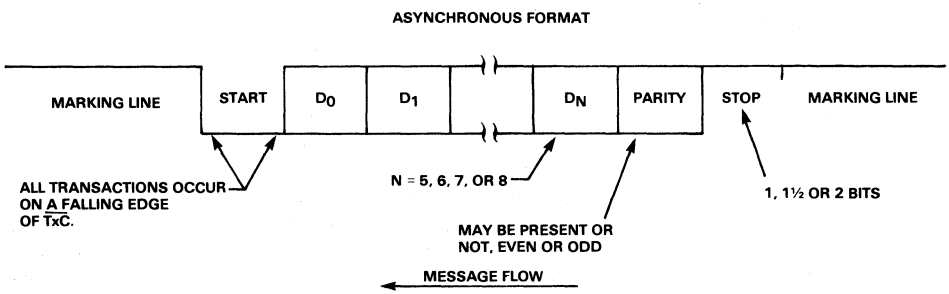
The Transmit Data output (TxD) is held marking (High) when the transmitter has no data to send. Under program control, the Send Break (WR5, D_4) command can be issued to hold TxD spacing (Low) until the command is cleared.

The Z80-SIO automatically adds the start bit, the programmed parity bit (odd, even or no parity) and the programmed number of stop bits to the data character to be transmitted. When the character length is six or seven bits, the unused bits are automatically ignored by the Z80-SIO. If the character length is five bits or less, refer to the table in the Write Register 5 description (Z80-SIO Programming section) for the data format.

Serial data is shifted from TxD at a rate equal to 1, 1/16th, 1/32nd, or 1/64th of the clock rate supplied to the Transmit Clock input ($\overline{\text{TxC}}$). Serial data is shifted out on the falling edge of ($\overline{\text{TxC}}$).

If set, the External/Status Interrupt mode monitors the status of $\overline{\text{DCD}}$, $\overline{\text{CTS}}$ and $\overline{\text{SYNC}}$ throughout the transmission of the message. If these inputs change for a period of time greater than the minimum specified pulse width, the interrupt is generated. In a transmit operation, this feature is used to monitor the modem control signal $\overline{\text{CTS}}$.

ASYNCHRONOUS MESSAGE FORMAT
Figure 3.1



3.3 ASYNCHRONOUS RECEIVE

Asynchronous Receive operation begins when the Receive Enable bit is set. If the Auto Enables option is selected, $\overline{DCD}$ must be Low as well. A Low (spacing) condition on the Receive Data input (RxD) indicates a start bit. If this Low persists for at least one-half of a bit time, the start bit is assumed to be valid and the data input is then sampled at mid-bit time until the entire character is assembled. This method of detecting a start bit improves error rejection when noise spikes exist on an otherwise marking line.

If the x1 clock mode is selected, bit synchronization must be accomplished externally. Receive data is sampled on the rising edge of RxC. The receiver inserts 1's when a character length of other than eight bits is used. If parity is enabled, the parity bit is not stripped from the assembled character for character lengths other than eight bits. For lengths other than eight bits, the receiver assembles a character length of the required number of data bits, plus a parity bit and 1's for any unused bits. For example, the receiver assembles a 5-bit character with the following format: 11 P D_4 D_3 D_2 D_1 D_0.

Since the receiver is buffered by three 8-bit registers in addition to the receive shift register, the CPU has enough time to service an interrupt and to accept the data character assembled by the Z80-SIO. The receiver also has three buffers that store error flags for each data character in the receive buffer. These error flags are loaded at the same time as the data characters.

After a character is received, it is checked for the following error conditions:

When parity is enabled, the Parity Error bit (RR1, D_4) is set whenever the parity bit of the character does not match with the programmed parity. Once this bit is set, it remains set until the Error Reset Command (WR0) is given.

CONTENTS OF WRITE REGISTERS 3, 4 and 5 in ASYNCHRONOUS MODES
Table 3.1

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
WR3	00 = Rx 5 BITS/CHAR 10 = Rx 6 BITS/CHAR 01 = Rx 7 BITS/CHAR 11 = Rx 8 BITS/CHAR		AUTO ENABLES	0	0	0	0	Rx ENABLE
WR 4	00 = x1 CLOCK MODE 01 = x16 CLOCK MODE 10 = x32 CLOCK MODE 11 = x64 CLOCK MODE		0	0	00 = NOT USED 01 = 1 STOP BIT/CHAR 10 = 1½ STOP BITS/CHAR 11 = 2 STOP BITS/CHAR		EVEN-$\overline{ODD}$ PARITY	PARITY ENABLE
WR5	DTR	00 = Tx 5 BITS (OR LESS) CHAR 10 = Tx 6 BITS/CHAR 01 = Tx 7 BITS/CHAR 11 = Tx 8 BITS/CHAR		SEND BREAK	Tx ENABLE	0	RTS	0

The Framing Error bit (RR1, D_6) is set if the character is assembled without any stop bits (that is a Low level detected for a stop bit). Unlike the Parity Error bit, this bit is set (and not latched) only for the character on which it occurred. Detection of framing error adds an additional one-half of a bit time to the character time so the framing error is not interpreted as a new start bit.

ASYNCHRONOUS MODE
Table 3.2

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
	REGISTER:	**INFORMATION LOADED:**	
	WR0	CHANNEL RESET	Reset SIO
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 4, RESET EXTERNAL/STATUS INTERRUPT	
	WR4	ASYNCHRONOUS MODE, PARITY INFORMATION, STOP BITS INFORMATION, CLOCK RATE INFORMATION	Issue Parameters
INITIALIZE	WR0	POINTER 3	
	WR3	RECEIVE ENABLE, AUTO ENABLES, RECEIVE CHARACTER LENGTH	
	WR0	POINTER 5	
	WR5	REQUEST TO SEND, TRANSMIT ENABLE, TRANSMIT CHARACTER LENGTH, DATA TERMINAL READY	Receive and Transmit both fully initialized. Auto Enables will enable Transmitter if $\overline{CTS}$ is active and Receiver if $\overline{DCD}$ is active.
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPT	
	WR1	TRANSMIT INTERRUPT ENABLE, STATUS AFFECTS VECTOR, INTERRUPT ON ALL RECEIVE CHARACTERS, DISABLE WAIT/READY FUNCTION, EXTERNAL INTERRUPT ENABLE	Transmit/Receive interrupt mode selected. External Interrupt monitors the status $\overline{CTS}$. $\overline{DCD}$ and $\overline{SYNC}$ inputs and detects the Break sequence. Status Affects Vector in Channel B only.
	TRANSFER FIRST DATA BYTE TO SIO		This data byte must be transferred or no transmit interrupts will occur.
IDLE MODE		EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	Program is waiting for an interrupt from the SIO.
		Z80 INTERRUPT ACKNOWLEDGE CYCLE TRANSFERS RR2 TO CPU	When the interrupt occurs, the interrupt vector is modified by: 1. Receive Character Available; 2. Transmit Buffer Empty; 3. External/Status change; and 4. Special Receive condition.
		IF A CHARACTER IS RECEIVED: TRANSFER DATA CHARACTER TO CPU UPDATE POINTERS AND PARAMETERS RETURN FROM INTERRUPT	
DATA TRANSFER AND ERROR MONITORING		IF TRANSMITTER BUFFER IS EMPTY: TRANSFER DATA CHARACTER TO SIO UPDATE POINTERS AND PARAMETERS RETURN FROM INTERRUPT	Program control is transferred to one of the eight interrupt service routines.
		IF EXTERNAL STATUS CHANGES: TRANSFER RRO TO CPU PERFORM ERROR ROUTINES (INCLUDE BREAK DETECTION) RETURN FROM INTERRUPT	If used with processors other than the Z80, the modified interrupt vector (RR2) should be returned to the CPU in the Interrupt Acknowledge sequence.

ASYNCHRONOUS MODE
Table 3.2

FUNCTION	TYPICAL PROGRAM STEPS	COMMENTS
	RESISTER: INFORMATION LOADED:	
	IF SPECIAL RECEIVE CONDITION OCCURS: TRANSFER RR1 to CPU DO SPECIAL ERROR (e.g. FRAMING ERROR) RETURN FROM INTERRUPT	
	REDEFINE RECEIVE/TRANSMIT INTERRUPT MODES	When transmit or receive data transfer is complete.
TERMINATION	DISABLE TRANSMIT/RECEIVE MODES	
	UPDATE MODEM CONTROL OUTPUTS (e.g. RTS OFF)	In transmit the All Sent Status bit indicates transmission is complete.

If the CPU fails to read a data character while more than three characters have been received, the Receive Overrun bit (RR1, D_5) is set. When this occurs, the fourth character assembled replaces the third character in the receive buffers. With this arrangement, only the character that has been written over is flagged with the Receive Overrun Error bit. Like Parity Error, this bit can only be reset by the Error Reset command from the CPU. Both the Framing Error and Receive Overrun Error cause an interrupt with the interrupt vector indicating a Special Receive condition (if Status Affects Vector is selected).

Since the Parity Error and Receive Overrun Error flags are latched, the error status that is read reflects an error in the current word in the receive buffer plus any Parity or Overrun Errors received since the last Error Reset command. To keep correspondence between the state of the error buffers and the contents of the receive data buffers, the error status register must be read before the data. This is easily accomplished if vectored interrupts are used, because a special interrupt vector is generated for these conditions.

While the External/Status interrupt is enabled, break detection causes an interrupt and the Break Detected status bit, (RR0, D_7), is set. The Break Detected interrupt should be handled by issuing the Reset External/Status Interrupt command to the Z80-SIO in response to the first Break Detected interrupt that has a Break status of 1 (RR0, D_7). The Z80-SIO monitors the Receive Data input and waits for the Break sequence to terminate, at which point the Z80-SIO interrupts the CPU with the Break status set to 0. The CPU must again issue the Reset External/Status Interrupt command in its interrupt service routine to reinitialize the break detection logic.

The External/Status interrupt also monitors the status of DCD. If the DCD pin becomes inactive for a period greater than the minimum specified pulse width, an interrupt is generated with the DCD status bit (RR0, D_3) set to 1. Note that the DCD input is inverted in the RR0 status register.

If the status is read after the data, the error data for the next word is also included if it has been stacked in the buffer. If operations are performed rapidly enough so the next character is not yet received, the status register remains valid. An exception occurs when the Interrupt On First Character Only mode is selected. A special interrupt in this Mode holds the error data and the character itself (even if read from the buffer) until the Error Reset command is issued. This prevents further data from becoming available in the receiver until the Reset command is issued, and allows CPU intervention on the character with the error even if DMA or block transfer techniques are being used.

If Interrupt On Every Character is selected, the interrupt vector is different if there is an error status in RR1. If a Receiver Overrun occurs, the most recent character received is loaded into the buffer; the character preceding it is lost. When the character that has been written over is read, the Receive Overrun bit is set and the Special Receive Condition vector is returned if Status Affects Vector is enabled.

In a polled environment, the Receive Character Available bit (RR0, D_0) must be monitored so the Z80-CPU can know when to read a character. This bit is automatically reset when the receive buffers are read. To prevent overwriting data in polled operations, the transmit buffer status must be checked before writing into the transmitter. The Transmit Buffer Empty bit is set to 1 whenever the transmit buffer becomes empty.

4.0 SYNCHRONOUS OPERATION

4.1 INTRODUCTION

Before describing synchronous transmission and reception, the three types of character synchronization-Monosync, Bisync, and External Sync-require some explanation. These modes use the x1 clock for both Transmit and Receive operations. Data is sampled on the rising edge of the Receive Clock input ($\overline{RxC}$). Transmitter data transitions occur on the falling edge of the Transmit Clock input ($\overline{TxC}$).

The differences between Monosync, Bisync, and External Sync are in the manner in which initial character synchronization is achieved. The mode of operation must be selected before sync characters are loaded because the registers are used differently in the various modes. Figure 4.1 shows the formats for all three of these synchronous modes.

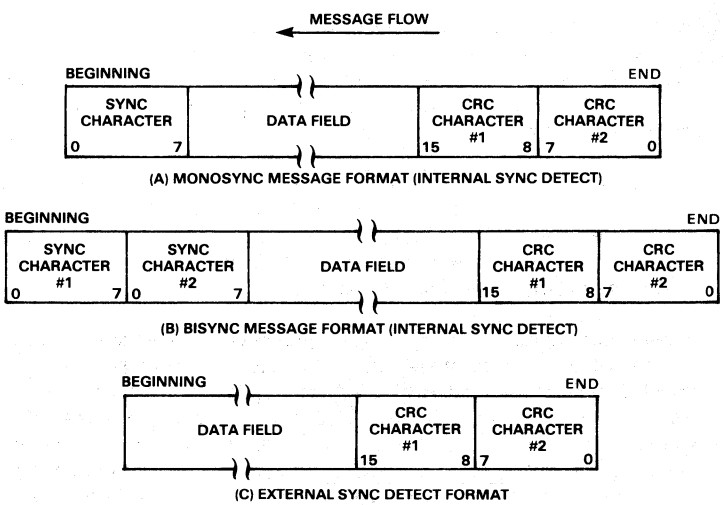
Monosync. In a Receive operation, matching a single sync character (8-bit sync mode) with the programmed sync character stored in WR7 implies character synchronization and enables data transfer.

Bisync. Matching two contiguous sync characters (16-bit sync mode) with the programmed sync characters stored in WR6 and WR7 implies character synchronization. In both the Monosync and Bisync modes, $\overline{SYNC}$ is used as an output and is active for the part of the receive clock that detects the sync character.

External Sync. In this mode, character synchronization is established externally; $\overline{SYNC}$ is an input that indicates that external character synchronization has been achieved. After the sync pattern is detected, the external logic must wait for two full Receive Clock cycle to activate the $\overline{SYNC}$ input. The $\overline{SYNC}$ input must be held Low until character synchronization is lost. Character assembly begins on the rising edge of $\overline{RxC}$ that precedes the falling edge of $\overline{SYNC}$.

In all cases after a reset, the receiver is in the Hunt phase, during which the Z80-SIO looks for character synchronization. The hunt can begin only when the receiver is enabled, and data transfer can begin only when character synchronization has been achieved. If character synchronization is lost, the Hunt phase can be re-entered by writing a control word with the Enter Hunt Phase bit set (WR3, D_4). In the Transmit mode, the transmitter always sends the programmed number of sync bits (8 or 16). In the Monosync mode, the transmitter transmits for WR6; the receiver compares against WR7.

SYNCHRONOUS FORMATS
Figure 4.1



In the Monosync, Bisync, and External Sync modes, assembly of received data continues until the Z80-SIO is reset, or until the receiver is disabled (by command or $\overline{DCD}$ in the Auto Enables mode), or until the CPU sets the Enter Hunt Phase bit.

After initial synchronization has been achieved, the operation of the Monosync, Bisync, and External Sync modes is quite similar. Any differences are specified in the following text.

Table 4.1 shows how WR3, WR4, and WR5 are used in synchronous receive and transmit operations. WR0 points to other registers and issues various commands, WR1 defines the interrupt modes, WR2 stores the interrupt vector and WR6 and WR7 store sync characters. Table 4.2 illustrates the typical program steps that implement a half-duplex Bisync transmit operation.

CONTENTS OF WRITE REGISTERS 3, 4, AND 5 IN SYNCHRONOUS MODES
Table 4.1

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
WR3	00=Rx 5 BITS/CHAR 10=Rx 6 BITS/CHAR 01=Rx 7 BITS/CHAR 11=Rx 8 BITS/CHAR		AUTO ENABLES	ENTER HUNT MODE	Rx CRC ENABLE	0	SYNC CHAR LOAD INHIBIT	RX ENABLE
WR4	0	0	00=8-BIT SYNC CHAR 01=16-BIT SYNC CHAR 10=SDLC MODE 11=EXT SYNC MODE		0 SELECTS SYNC MODES	0	EVEN/$\overline{ODD}$ PARITY	PARITY ENABLE
WR5	DTR	00=Tx 5 BITS (OR LESS)/CHAR 10=Tx 6 BITS/CHAR 01=Tx 7 BITS/CHAR 11=Tx 8 BITS/CHAR		SEND BREAK	Tx ENABLE	1 SELECTS CRC-16	RTS	Tx CRC ENABLE

4.2 SYNCHRONOUS TRANSMIT

4.2.1 INITIALIZATION

The system program must initialize the transmitter with the following parameters: odd or even parity, x1 clock mode, 8- or 16-bit sync character(s), CRC polynomial, Transmitter Enables, Request To Send, Data Terminal Ready, interrupt modes, and transmit character length. WR4 parameters must be issued before WR1, WR3, WR5, WR6, and WR7 parameters or commands.

One of two polynomials CRC-16 ($x^{16} + X^{15} + X^2 + 1$) or SDLC ($X^{16} + X^{12} + X^5 + 1$) may be used with synchronous modes. In either case (SDLC mode not selected), the CRC generator and checker are reset to all 0's. In the transmit initialization process, the CRC generator is initialized by setting the Reset Transmit CRC Generator command bits (WR0). Both the transmitter and the receiver use the same polynomial.

Transmit Interrupt Enable or Wait/Ready Enable can be selected to transfer the data. The External/Status interrupt mode is used to monitor the status of the $\overline{CLEAR\ TO\ SEND}$ input as well as the Transmit Underrun/EOM latch. Optionally, the Auto Enables' feature can be used to enable the transmitter when $\overline{CTS}$ is active monitored so the Z80-CPU can know when to read a character. This bit is automatically reset when the receive buffers are read. To prevent overwriting data in polled operations, the transmit buffer status must be checked before writing into the transmitter. The Transmit Buffer Empty bit is set to 1 whenever the transmit buffer becomes empty.

The first data transfer to the Z80-SIO can begin when the External/Status interrupt occurs (CTS status bit set) or immediately following the Transmit Enable command (if the Auto Enables modes are set).

Transmit data is held marking after reset or if the transmitter is not enabled. Break may be programmed to generate a spacing line that begins as soon as the Send Break bit is set. With the transmitter fully initialized and enabled, the default condition is continuous transmission of the 8- or 16-bit sync character.

4.2.2 DATA TRANSFER AND STATUS MONITORING

In this phase, there are several combinations of interrupts and Wait/Ready.

Data Transfer Using Interrupts. If the Transmit Interrupt Enable bit (WR1, D_1) is set, an interrupt is generated each time the transmit buffer becomes empty. The interrupt can be satisfied either by writing another character into the transmitter or by resetting the Transmitter Interrupt Pending latch with a Reset Transmitter Pending command (WR0, CMD_5). If the interrupt is satisfied with this command and nothing more is written into the transmitter, there can be no further Transmit Buffer Empty interrupts, because it is the process of the buffer becoming empty that causes the interrupts. This situation does cause a Transmit Underrun condition, which is explained in the "Bisync Transmit Underrun" section.

Data Transfer Using WAIT/READY. To the CPU, the activation of $\overline{WAIT}$ indicates that the Z80-SIO is not ready to accept data and that the CPU must extend the output cycle. To a DMA controller, READY indicates that the transmit buffer is empty and that the Z80-SIO is ready to accept the next data character. If the data character is not loaded into the Z80-SIO by the time the transmit shift register is empty, the Z80-SIO enters the Transmit Underrun condition.

Bisync Transmit Underrun. In Bisync protocol, filler characters are inserted to maintain synchronization when the transmitter has no data to send (Transmit Underrun condition). The Z80-SIO has two programmable options for solving this situation: it can insert sync characters or it can send the CRC characters generated so far, followed by sync characters.

These options are under the control of the Reset Transmit Underrun/EOM command in WR0. Following a chip or channel reset, the Transmit Underrun/EOM status bit (RR0, D_6) is in a set condition and allows the insertion of sync characters when there is no data to send. CRC is not calculated on the automatically inserted sync characters. When the CPU detects the end of message, a Reset Transmit Underrun/EOM command can be issued. This allows CRC to be sent when the transmitter has no data. In this case, the Z80-SIO sends CRC, followed by sync characters, to terminate the message.

There is no restriction as to when in the message the Transmit Underrun/EOM bit can be reset. If Reset is issued after the first data character has been loaded, the 16-bit CRC is sent and followed by sync characters the first time the transmitter has no data to send. Because of the Transmit Underrun condition, an External/Status interrupt is generated whenever the Transmit Underrun/EOM bit becomes set.

In the case of sync insertion, an interrupt is generated only after the first automatically inserted sync character has been loaded. The status indicates the Transmit Underrun/EOM bit and the Transmit Buffer Empty bit are set.

In the case of CRC insertion, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset while CRC is being sent. When CRC has been completely sent, the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin (this interrupt occurs because CRC has been sent and sync has been loaded). If no more messages are to be sent, the program can terminate transmission by resetting RTS and disabling the transmitter (WR5, D_3).

Pad characters may be sent by setting the Z80-SIO to 8-bits/transmit character and writing FF to the transmitter while CRC is being sent. Alternatively, the sync characters can be redefined

as pad characters during this time. The following example is included to clarify this point:

The Z80-SIO interrupts with the Transmit Buffer Empty bit set.

The CPU recognizes that the last character (ETX) of the message has already been sent to the Z80-SIO by examining the internal program status.

To force the Z80-SIO to send CRC, the CPU issues the Reset Transmit Underrun/EOM Latch command (WRO) and satisfies the interrupt with the Reset Transmit Interrupt Pending command. (This command prevents the Z80-SIO from requesting more data.) Because of the transmit underrun caused by this command, the Z80-SIO starts sending CRC. The Z80-SIO also causes an External/Status interrupt with the Transmit Underrun/EOM latch set.

The CPU satisfies this interrupt by loading pad characters into the transmit buffer and issuing the Reset External/Status Interrupt command.

With this sequence, CRC is followed by a pad character instead of a sync character. Note that the Z80-SIO will interrupt with a Transmit Buffer Empty interrupt when CRC is completely sent and that the pad character is loaded into the transmit shift register.

From this point on the CPU can send more pad characters or sync characters.

Bisync CRC Generation. Setting the Transmit CRC enable bit (WR5, D_0) initiates CRC accumulation when the program sends the first data character to the Z80-SIO. Although the Z80-SIO automatically transmits up to two sync characters (16-bit sync), it is wise to send a few more sync characters ahead of the message (before enabling Transmit CRC) to ensure synchronization at the receiving end.

The transmit CRC Enable bit can be changed on the fly any time in the message to include or exclude a particular data character from CRC accumulation. The Transmit CRC Enable bit should be in the desired state when the data character is loaded from the transmit data buffer into the transmit shift register. To ensure this bit is in the proper state, the Transmit CRC Enable bit must be issued before sending the data character to the Z80-SIO.

Transmit Transparent Mode. Transparent mode (Bisync protocol) operation is made possible by the ability to change Transmit CRC Enable on the fly and by the additional capability of inserting 16-bit sync characters. Exclusion of the DLE characters from CRC calculation can be achieved by disabling CRC calculation immediately preceding the DLE character transfer to the Z80-SIO.

In the case of a Transmit Underrun condition in the Transparent mode, a pair of DLE-SYN characters are sent. The Z80-SIO can be programmed to send the DLE-SYN sequence by loading a DLE character into WR6 and a sync character into WR7.

Transmit Termination. The Z80-SIO is equipped with a special termination that maintains data integrity and validity. If the transmitter is disabled while a data or sync character is being sent, that character is sent as usual, but is followed by a marking line rather than CRC or sync characters. When the transmitter is disabled, a character in the buffer remains in the buffer. If the transmitter is disabled while CRC is being sent, the 16-bit transmission is completed, but sync is sent instead of CRC.

A programmed break is effective as soon as it is written into the control register; characters in the transmit buffer and shift register are lost.

In all modes, characters are sent with the least significant bits first. This requires right-hand justification of transmitted data if the word length is less than eight bits. If the word length is five bits or less, the special technique described in the Write Register 5 discussion (Z80-SIO Programming section) must be used for the data format. The states of any unused bits in a data character are irrelevant, except when in the Five Bits or Less mode.

If the External/Status Interrupt Enable bit is set, transmitter conditions such as "starting to send CRC characters" "starting to send sync characters," and $\overline{CTS}$ changing state cause interrupts that have a unique vector if Status Affects Vector is set. This interrupt mode may be used during block transfers.

All interrupts may be disabled for operation in a Polled mode or to avoid interrupts at inappropriate times during the execution of a program.

4.3 SYNCHRONOUS RECEIVE

4.3.1 INITIALIZATION

The system program initiates the Synchronous Receive operation with the following parameters: odd or even parity, 8- or 16-bit sync characters, x1 clock mode, CRC polynomial, receive character length, etc. Sync characters must be loaded into registers WR6 and WR7. The receivers can be enabled only after all receive parameters are set. WR4 parameters must be issued before WR1, WR3, WR5, WR6 and WR7 parameters or commands.

After this is done, the receiver is in the Hunt phase. It remains in this phase until character synchronization is achieved. Note that, under program control, all the leading sync characters of the message can be inhibited from loading the receive buffers by setting the Sync Character Load Inhibit bit in WR3.

4.3.2 DATA TRANSFER AND STATUS MONITORING

After character synchronization is achieved, the assembled characters are transferred to the receive data FIFO. The following four interrupt modes are available to transfer the data and its associated status to the CPU.

No Interrupts Enabled. This mode is used for a purely polled operation or for off-line conditions.

Interrupt On First Character Only. This mode is normally used to start a polling loop or a Block Transfer instruction using $\overline{WAIT/READY}$ to synchronize the CPU or the DMA device to the incoming data rate. In this mode, the Z80-SIO interrupts on the first character and thereafter interrupts only if Special Receive conditions are detected. The mode is reinitialized with the Enable Interrupt on Next Receive Character command to allow the next character received to generate an interrupt. Parity errors do not cause interrupts in this mode, but End Of Frame (SDLC mode) and Receive Overrun do.

If External/Status interrupts are enabled, they may interrupt any time $\overline{DCD}$ changes state.

Interrupt On Every Character. Whenever a character enters the receive buffer, an interrupt is generated. Error and Special Receive conditions generate a special vector if Status Affects Vector is selected. Optionally, a Parity Error may be directed not to generate the special interrupt vector.

Special Receive Condition Interrupts. The Special Receive Condition interrupt can occur only if either the Receive Interrupt On First Character Only or Interrupt On Every Receive Character modes is also set. The Special Receive Condition interrupt is caused by the Receive Overrun error condition. Since the Receive Overrun and Parity error status bits are latched, the error status (when read) reflects an error in the current word in the receive buffer in addition to any Parity or Overrun errors received since the last Error Reset command. These status bits can only be reset by the Error reset command.

CRC Error Checking and Termination. A CRC error check on the receive message can be performed on a per character basis under program control. The Receive CRC Enable bit (WR3, D_3) must be set/reset by the program before the next character is transferred from the receive shift register into the receive buffer register. This ensures proper inclusion or exclusion of data characters in the CRC check.

In the Monosync, Bisync, and External Sync modes, the CRC/Framing Error bit (RR1, D_6) contains the comparison result of the CRC checker 16 bit times (eight bits delay and eight shifts for CRC) after the character has been transferred from the receive shift register to the buffer. The result should be zero, indicating an error-free transmission. (Note that the result is valid only at the end of CRC calculation. If the result is examined before this time, it usually indicates an error.) The comparison is made with each transfer and is valid only as long as the character remains in the receive FIFO.

Following is an example of the CRC checking operation when four characters (A,B,C, and D) are received in that order.

Character A loaded into buffer
Character B loaded into buffer

If CRC is disabled before C is in the buffer, CRC is not calculated on B.

Character C loaded into buffer

After C is loaded, the CRC/Framing Error bit shows the result of the comparison through character A.

Character D loaded into buffer

After D is in the buffer, the CRC Error bit shows the result of the comparison through character B whether or not B was included in the CRC calculations.

Owing to the serial nature of CRC calculation, the Receive Clock (RxC) must cycle 16 times (8-bit delay plus 8-bit CRC shift) after the second CRC character has been loaded into the receive buffer, or 20 times (the previous 16 plus 3-bit buffer delay and 1-bit input delay) after the last bit is at the RxD input, before CRC calculation is complete. A faster external clock can be gated into the Receive Clock input to supply the required 16 cycles. The Transmit and Receive Data Path diagram (Figure 4) illustrates the various points of delay in the CRC path.

The typical program steps that implement a half-duplex Bisync Receive mode are illustrated in Table 6. The complete set of command and status bit definitions are explained under "Z80-SIO Programming."

BISYNC TRANSMIT CODE
Table 4.2

FUNCTION	TYPICAL PROGRAM STEPS	COMMENTS
REGISTER:	**INFORMATION LOADED:**	
WR0	CHANNEL RESET, RESET TRANSMIT CRC GENERATOR	Reset SIO, initialize CRC generator
WR0	POINTER 2	
WR2	INTERRUPT VECTOR	Channel B only
WR0	POINTER 3	
WR3	AUTO ENABLES	Transmission begins only after CTS is detected.
WR0	POINTER 4	
WR4	PARITY INFORMATION, SYNC MODES INFORMATION, x1 CLOCK MODE	Issue transmit parameters.
WR0	POINTER 6	
WR6	SYNC CHARACTER 1	
WR0	POINTER 7, RESET EXTERNAL/STATUS INTERRUPTS	

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
INITIALIZE	WR7	SYNC CHARACTER 2	
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPTS	
	WR1	STATUS AFFECTS VECTOR, EXTERNAL INTERRUPT ENABLE, TRANSMIT INTERRUPT ENABLE OR WAIT/READY ENABLE	External Interrupt mode Monitors the status of $\overline{CTS}$ and $\overline{DCD}$ input pins as well as the status of Tx Underrun/EOM latch. Transmit Interrupt Enable interrupts when the Transmit buffer becomes empty; the Wait/Ready mode can be used to transfer data using DMA or CPU Block Transfer.
	WR0	POINTER 5	Status Affects Vector (Channel B only)
	WR5	REQUEST TO SEND, TRANSMIT ENABLE, BISYNC CRC, TRANSMIT CHARACTER LENGTH	Transmit CRC Enable should be set when first non-sync data is sent to Z80-SIO.
		FIRST SYNC BYTE TO SIO	Need several sync characters in the beginning of message. Transmitter is fully initialized.
IDLE MODE		EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	Waiting for interrupt or Wait/Ready output to transfer data.
DATA TRANSFER AND STATUS MONITORING		**WHEN INTERRUPT (WAIT/READY) OCCURS:** INCLUDE/EXCLUDE DATA BYTE FROM CRC ACCUMULATION (IN SIO). TRANSFER DATA BYTE FROM CPU (OR MEMORY) TO SIO. DETECT AND SET APPROPRIATE FLAGS FOR CONTROL CHARACTERS (IN CPU). RESET Tx UNDERRUN/EOM LATCH (WR0) IF LAST CHARACTER OF MESSAGE IS DETECTED. UPDATE POINTERS AND PARAMETERS (CPU). RETURN FROM INTERRUPT.	Interrupt occurs (Wait/Ready becomes active) when first data byte is being sent. Wait mode allows CPU block transfer from memory to SIO; Ready mode allows DMA block transfer from memory to SIO. The DMA chip can be programmed to capture special control characters (by examining only the bits that specify ASCII or EBCDIC control characters) and interrupt CPU.
		IF ERROR CONDITION OR STATUS CHANGE OCCURS: TRANSFER RR0 TO CPU EXECUTE ERROR ROUTINE RETURN FROM INTERRUPT	Tx Underrun/EOM indicates either transmit underrun (sync character being) sent to end of message (CRC-16 being sent)
		REDEFINE INTERRUPT MODES.	
TERMINATION		UPDATE MODEM CONTROL OUTPUTS (E.G., TURN OFF RTS).	Program should gracefully terminate message.
		DISABLE TRANSMIT MODE	

III

BISYNC RECEIVE MODE
Table 4.3

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
	REGISTER:	**INFORMATION LOADED**	
	WR0	CHANNEL RESET, RESET RECEIVE CRC CHECKER	Reset SIO, initialize Receive CRC checker
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 4	
	WR4	PARITY INFORMATION, SYNC MODES INFORMATION, x1 CLOCK MODE	Issue receive parameters.
	WR0	POINTER 5, RESET EXTERNAL STATUS INTERRUPT	
	WR5	BISYNC CRC-16 DATA TERMINAL READY	
	WR0	POINTER 3	
INITIALIZE	WR3	SYNC CHARACTER LOAD INHIBIT, RECEIVE CRC ENABLE, ENTER HUNT MODE, AUTO ENABLES, RECEIVE CHARACTER LENGTH	Sync character load inhibit strips all the loading sync characters at the beginning of the message. Auto Enables enables the receiver to accept data only after the $\overline{DCD}$ input is active
	WR0	POINTER 6	
	WR6	SYNC CHARACTER 1	
	WR0	POINTER 7	
	WR7	SYNC CHARACTER 2	
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPT	
	WR1	STATUS AFFECTS VECTOR, EXTERNAL INTERRUPT ENABLE, RECEIVE INTERRUPT ON FIRST CHARACTER ONLY	In this interrupt mode, only the first non-sync data character is transferred to the CPU. All subsequent data is transferred on a DMA basis; however, Special Receive Condition interrupts will interrupt the CPU. Status Affects Vector used in Channel B only.
	WR0	POINTER 3, ENABLE INTERRUPT ON NEXT RECEIVE CHARACTER	Resetting this interrupt mode provides simple program loopback entry for the next transaction.
	WR3	RECEIVE ENABLE SYNC CHARACTER LOAD INHIBIT, ENTER HUNT MODE, AUTO ENABLE, RECEIVE WORD LENGTH	WR3 is reissued to enable receiver. Receive CRC Enable must be set after receiving SOH or STX character.
IDLE MODE		EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	Receive mode is fully initialized and the system is waiting for interrupt on first character.

FUNCTION	TYPICAL PROGRAM STEPS	COMMENTS
	WHEN INTERRUPT ON FIRST CHARACTER OCCURS, THE CPU DOES THE FOLLOWING: • TRANSFERS DATA BYTE TO CPU • DETECTS AND SETS APPROPRIATE FLAGS FOR CONTROL CHARACTERS (IN CPU) • INCLUDES/EXCLUDES DATA BYTE IN CRC CHECKER • UPDATES POINTERS AND OTHER PARAMETERS • ENABLES WAIT/READY FOR DMA OPERATION • ENABLES DMA CONTROLLER • RETURNS FOR INTERRUPT	During the Hunt mode, the SIO detects two contiguous characters to establish synchronization. The CPU establishes the DMA mode and all subsequent data characters are transferred by the DMA controller. The controller is also programmed to capture special characters (by examining only the bits that specify ASCII or EBCDIC control characters) and interrupt the CPU upon detection. In response, the CPU examines the status or control characters and takes appropriate action (e.g., CRC Enable Update)
DATA TRANSFER AND STATUS MONITORING	**WHEN WAIT/READY BECOMES ACTIVE, THE DMA CONTROLLER DOES THE FOLLOWING:** • TRANSFERS DATA BYTE TO MEMORY • INTERRUPTS CPU IF A SPECIAL CHARACTER IS CAPTURED BY THE DMA CONTROLLER • INTERRUPTS THE CPU IF THE LAST CHARACTER OF THE MESSAGE IS DETECTED	
	FOR MESSAGE TERMINATION, THE CPU DOES THE FOLLOWING: • TRANSFERS RR1 TO THE CPU • SETS ACK/NAK REPLY FLAG BASED ON CRC RESULT • UPDATES POINTERS AND PARAMETERS • RETURNS FROM INTERRUPT	The SIO interrupts the CPU for error condition and the error routine aborts the present message, clears the error condition and repeats the operation.
TERMINATION	REDEFINE INTERRUPT MODES AND SYNC MODES UPDATE MODEM CONTROLS DISABLES RECEIVE MODE	

5.0 SDLC (HDLC OPERATION)

5.1 INTRODUCTION

The Z80-SIO is capable of handling both High-level Synchronous Data Link Control (HDLC) and IBM Synchronous Data Link Control (SDLC) protocols. In the following discussion, only SDLC is referred to because of the high degree of similarity between SDLC and HDLC.

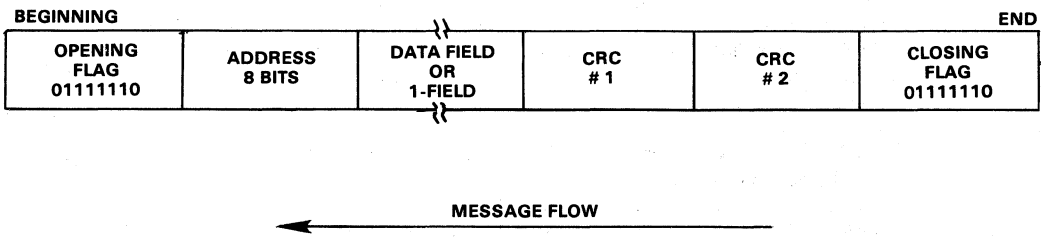
The SDLC mode is considerably different from Synchronous Bisync protocol because it is bit oriented rather than character oriented and, therefore, can naturally handle transparent operation. Bit orientation makes SDLC a flexible protocol in terms of message length and bit patterns. The Z80-SIO has several built-in features to handle variable message length. Detailed information concerning SDLC protocol can be found in literature published on this subject, such as IBM document GA27-3093.

The SDLC message, called the frame (Figure 5.1), is opened and closed by flags that are similar to the sync characters in Bisync protocol. The Z80-SIO handles the transmission and recognition of the flag characters that mark the beginning and end of the frame. Note that the Z80-SIO can receive shared-zero flags, but cannot transmit them. The 8-bit address field of a SDLC frame contains the secondary station address. The Z80-SIO has an Address Search mode that recognizes the secondary station so that it can accept or reject the frame.

Since the control field of the SDLC frame is transparent to the Z80-SIO, it is simply transferred to the CPU. The Z80-SIO handles the Frame Check sequence in a manner that simplifies the program by incorporating features such as initializing the CRC generator to all 1's, resetting the CRC checker when the opening flag is detected in the Receive mode,and sending the Frame Check/Flag sequence in the Transmit mode. Controller hardware is simplified by automatic zero insertion and deletion logic contained in the Z80-SIO.

Table 5.1 shows the contents of WR3,WR4,and WR5 during SDLC Receive and Transmit modes. WR0 points to other registers and issues various commands. WR1 defines the interrupt modes and WR2 stores the interrupt vector. WR7 stores the flag character and WR6 stores the secondary address.

TRANSMIT/RECEIVE SDLC/HDLC MESSAGE FORMAT
Figure 5.1



5.2 SDLC TRANSMIT

5.2.1 INITIALIZATION

Like Synchronous operation, the SDLC Transmit mode must be initialized with the following parameters: SDLC mode, SDLC polynomial, Request to Send, Data Terminal Ready, transmit character length, transmit interrupt modes (or Wait/Ready function), Transmit Enable, Auto Enables and External/Status interrupt.

Selecting the SDLC mode and the SDLC polynomial enables the Z80-SIO to initialize the CRC Generator to all 1's. This is accomplished by issuing the Reset Transmit CRC Generator command (WR0). Refer to the Synchronous Operation section for more details on the interrupt modes.

After reset, or when the transmitter is not enabled, the Transmit Data output is held marking. Break may be programmed to generate a spacing line. With the transmitter fully initialized and enabled, continuous flags are transmitted on the Transmit Data output.

An abort sequence may be sent by issuing the Send Abort command (WR0, CMD_1). This causes at least eight, but less than fourteen, 1's to be sent before the line reverts to continuous flags. It is possible that the Abort sequence (eight 1's) could follow up to five continuous 1 bits (allowed by the zero insertion logic) and, thus, cause up to thirteen 1's to be sent. Any data being transmitted and any data in the transmit buffer is lost when an abort is issued.

When required, an extra 0 is automatically inserted when there are five contiguous 1's in the data stream. This does not apply to flags or aborts.

5.2.2 DATA TRANSFER AND STATUS MONITORING

There are several combinations of interrupts and of Wait/Ready functions in the SLDC mode.

Data Transfer Using Interrupts. If the Transmit Interrupt Enable bit is set, an interrupt is generated each time the buffer becomes empty. The interrupt may be satisfied either by writing another character into the transmitter or by resetting the Transmit Interrupt Pending latch with a Reset Transmitter Pending command (WR0, CMD_5). If the interrupt is satisfied with this command and nothing more is written into the transmitter, there are no further transmitter interrupts. The result is a Transmit Underrun condition. When another character is written and sent out, the transmitter can again become empty and interrupt the CPU. Following the flags in an SDLC operation, the 8-bit address field, control field and information field may be sent to the Z80-SIO using the Transmit Interrupt mode. The Z80-SIO transmits the Frame Check sequence using the Transmit Underrun feature.

When the transmitter is first enabled, it is already empty and obviously cannot then become empty. Therefore, no Transmit Buffer Empty interrupts can occur until after the first data character is written.

Data Transfer Using Wait/Ready. If the Wait/Ready function has been selected, WAIT indicates to the CPU that the Z80-SIO is not ready to accept the data and the CPU must extend the I/O cycle. To a DMA controller, READY indicates that the transmitter buffer is empty and that the Z80-SIO is ready to accept the next character. If the data character is not loaded into the Z80-SIO by the time the transmit shift register is empty, the Z80-SIO enters the Transmit Underrun condition. Address,control,and information fields may be transferred to the Z80-SIO with this mode using the Wait/Ready function. The Z80-SIO transmits the Frame Check sequence using the Transmit Underrun feature.

SDLC Transmit Underrun/End of Message. SDLC-like protocols do not have provisions for fill characters within a message. The Z80-SIO therefore automatically terminates an SDLC frame when the transmit data buffer and output shift register have no more bits to send. It does this by first sending the two bytes of CRC and following these with one or more flags. This technique allows very high-speed transmissions under DMA or CPU block I/O control without requiring the CPU to respond quickly to the end of message situation.

The action that the Z80-SIO takes in the underrun situation depends on the state of the Transmit Underrun/EOM command. Following a reset, the Transmit Underrun/EOM status bit is in the set state and prevents the insertion of CRC characters during the time there is no data to send. Consequently, flag characters are sent. The Z80-SIO begins to send the frame as data is written into the transmit buffer. Between the time the first data byte is written and the end of the message, the Reset Transmit Underrun/EOM command must be issued. Thus the Transmit Underrun/EOM status bit is in the reset state at the end of the message (when underrun occurs), which automatically sends the CRC characters. The sending of the CRC again sets the Transmit/Underrun/EOM status bit.

Although there is no restriction as to when the Transmit Underrun/EOM bit can be reset within a message, it is usually reset after the first data character (secondary address) is sent

CONTENTS OF WRITE REGISTERS, 3, 4 AND 5 IN SDLC MODES
Table 5.1

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
WR3	00=Rx 5 BITS/CHAR 10=Rx 6 BITS/CHAR 01=Rx 7 BITS/CHAR 11=Rx 8 BITS/CHAR		AUTO ENABLES	ENTER HUNT MODE (IF INCOMING DATA NOT NEEDED)	Rx CRC ENABLE	ADDRESS SEARCH MODE	0	Rx ENABLE
WR4	0	0	1 SELECTS SDLC MODE	0	0	0	0	0
WR5	DTR	00=Tx 5 BITS (OR LESS)/CHAR 10=Tx 6 BITS/CHAR 01=Tx 7 BITS/CHAR 11=Tx 8 BITS/CHAR		0	Tx ENABLE	0 SELECTS SDLC CRC	RTS	Tx CRC ENABLE

to the Z80-SIO. Resetting this bit allows CRC and flags to be sent when there are no data to send which gives additional time to the CPU for recognizing the fault and responding with an abort command. By resetting it early in the message, the entire message has the maximum amount of CPU response time in an unintentional transmit underrun situation.

When the External/Status interrupt is set and while CRC is being sent, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset to indicate that the transmit register is full of CRC data. When CRC has been completely sent, the Transmit Buffer Empty status bit it set and an interrupt is generated to indicate to the CPU that another message can begin. This interrupt occurs because CRC has been sent and the flag has been loaded. If no more messages are to be sent, the program can terminate transmission by resetting $\overline{RTS}$, and disabling the transmitter.

In the SDLC mode, it is good practice to reset the Transmit Underrun/EOM status bit immediately after the first character is sent to the Z80-SIO. When the Transmit Underrun is detected, this ensures that the transmission time is filled by CRC characters, giving the CPU enough time to issue the Send Abort command. This also stops the flags from going on the line prematurely and eliminates the possibility of the receiver accepting the frame as valid data. The situation can happen because it is possible that—at the receiving end)—the data pattern immediately preceding the automatic flag insertion could match the CRC checker, giving a false CRC check result. The External/Status interrupt is generated whenever the Transmit Underrun/EOM bit is set because of the Transmit Underrun condition.

The transmit underrun logic provides additional protection against premature flag insertion if the proper response is given to the Z80-SIO by the CPU interrupt service routine. The following example is given to clarify this point:

The Z80-SIO raises an interrupt with the Transmit Buffer Empty status bit set.

The CPU does not respond in time and causes a Transmit Underrun condition.

The Z80-SIO starts sending CRC characters (two bytes).

The CPU eventually satisfies the Transmit Buffer Empty interrupt with a data character that follows the CRC character being transmitted.

The Z80-SIO sets the External/Status interrupt with the Transmit Underrun/EOM status bit set.

The CPU recognizes the Transmit Underrun/EOM status and determines from its internal program status that the interrupt is not for "end of message".

The CPU immediately issues a Send Abort Command (WRO) to the Z80-SIO.

The Z80-SIO sends the Abort sequence by destroying whatever data (CRC, data or flag) is being sent.

This sequence illustrates that the CPU has a protection of 22 minimum and 30 maximum transmit clock cycles.

SDLC CRC Generation. The CRC generator must be reset to all 1's at the beginning of each frame before CRC accumulation can begin. Actual accumulation begins when the program sends the address field (eight bits) to the Z80-SIO. Although the Z80-SIO automatically transmits one flag character following the Transmit Enable, it may be wise to send a few more flag characters ahead of the message to ensure character synchronization at the receiving end. This can be done by externally timing out after enabling the transmitter and before loading the first character.

The Transmit CRC Enable (WR5, D_0) should be enabled prior to sending the address field. In the SDLC mode all the characters between the opening and closing flags are included in CRC accumulation, and CRC generated in the Z80-SIO transmitter is inverted before it is sent on the line.

Transmit Termination. If the transmitter is disabled while a character is being sent, that character (data or flag) is sent in the normal fashion, but is followed by a marking line rather than CRC or flag characters.

A character in the buffer when the transmitter is disabled remains in the buffer; however, a programmed Abort sequence is effective as soon as it is written into the control register. Characters being transmitted, if any, are lost. In the case of CRC, the 16-bit transmission is completed if the transmitter is disabled; however, flags are sent in place of CRC.

In all modes, characters are sent with the least-significant bits first. This requires right-hand justification of data to be transmitted if the word length is less than eight bits. If the word length is five bits or less, the special technique described in the Write Register 5 section ("Z80-SIO Programming" chapter; "Write Registers" section) must be used.

Since the number of bits/character can be changed on the fly, the data field can be filled with any number of bits. When used in conjunction with the Receiver Residue codes, the Z80-SIO can receive a message that has a variable I-field and retransmit it exactly as received with no previous information about the character structure of the I-field (if any). A change in the number of bits does not affect the character in the process of being shifted out. Characters are sent with the number of bits programmed at the time that the character is loaded from the transmit buffer to the transmitter.

If the External/Status Interrupt Enable is set, transmitter conditions such as "starting to send CRC characters," "starting to send flag characters," and $\overline{CTS}$ changing state cause interrupts that have a unique vector if Status Affects Vector is set. All interrupts can be disabled for operation in a polled mode.

Table 5.2 shows the typical program steps that implement the half-duplex SDLC Transmit mode.

SDLC TRANSMIT MODE
Table 5.2

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
	REGISTER:	**INFORMATION LOADED:**	
	WR0	CHANNEL RESET	Reset SIO
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 3	
	WR3	AUTO ENABLES	Transmitter sends data only after $\overline{CTS}$ is detected
	WR0	POINTER 4, RESET EXTERNAL STATUS INTERRUPTS	
	WR4	PARITY INFORMATION, SDLC MODE, x1 CLOCK MODE	
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPTS	
INITIALIZE	WR1	EXTERNAL INTERRUPT ENABLE, STATUS AFFECTS VECTOR, TRANSMIT INTERRUPT ENABLE OR WAIT/READY MODE ENABLE	The External Interrupt Mode monitors the status of the $\overline{CTS}$ and $\overline{DCD}$ inputs, as well as the status of Tx Underrun/EOM latch. Transmit Interrupt interrupts when the Transmit buffer becomes empty; the Wait/Ready mode can be used to transfer data on a DMA or Block Transfer basis. The first interrupt occurs when $\overline{CTS}$ becomes active, at which point flags are transmitted by the Z80-SIO. The first data byte (address field) can be loaded into the Z80-SIO after this interrupt. Flags cannot be sent to the Z80-SIO as data. Status Affects Vector used in Channel B only.
	WR0	POINTER 5	
	WR5	TRANSMIT CRC ENABLE, REQUEST TO SEND, SDLC-CRC, TRANSMIT ENABLE, TRANSMIT WORD LENGTH, DATA TERMINAL READY	Sync mode must be defined before initializing transmit CRC generator.
	WR0	RESET TRANSMIT CRC GENERATOR	Initialize CRC generator to all 1's.
IDLE MODE		EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	Waiting Interrupt or Wait/Ready output to transfer data.
		WHEN INTERRUPT (WAIT/READY) OCCURS, THE CPU DOES THE FOLLOWING: • CHANGES TRANSMIT WORD LENGTH (IF NECESSARY) • TRANSFERS DATA BYTE FROM CPU (MEMORY) TO SIO • RESETS Tx UNDERRUN EOM LATCH WR0	Flags are transmitted by the SIO as soon as Transmit Enable is set and $\overline{CTS}$ becomes active. The $\overline{CTS}$ status change is the first interrupt that occurs and is followed by transmit buffer empty for subsequent transfers.

III

FUNCTION	TYPICAL PROGRAM STEPS	COMMENTS
	IF LAST CHARACTER OF THE I-FIELD IS SENT, THE SIO DOES THE FOLLOWING: • SENDS CRC • SENDS CLOSING FLAG • INTERRUPTS CPU WITH BUFFER EMPTY STATUS	Word length can be changed on the fly for variable I-field length. The data byte can contain address, control or I-field information (never a flag). It is good practice to reset Tx Underrun/EOM latch in the beginning of the message to avoid a false end-of-frame detection at the receiving end; This ensures that when underrun occurs, CRC is transmitted and underrun interrupt (Tx Underrun/EOM latch active) occurs. Note that "Send Abort" can be issued to the SIO in response to any interrupting continuing to abort the transmission.
DATA TRANSFER AND STATUS MONITORING	CPU DOES THE FOLLOWING: • ISSUES RESET Tx INTERRUPT PENDING COMMAND TO THE Z80-SIO • UPDATES NS COUNT • REPEATS THE PROCESS FOR NEXT MESSAGE, ETC. IF VECTOR INDICATES AN ERROR, THE CPU DOES THE FOLLOWING • SENDS ABORT • EXECUTES ERROR ROUTINE • UPDATES PARAMETERS, MODES, ETC. • RETURNS FROM INTERRUPT	
TERMINATION	REDEFINE INTERRUPT MODES UPDATE MODEM CONTROL OUTPUTS DISABLE TRANSMIT MODE	Terminate gracefully

5.3 SDLC RECEIVE

5.3.1 INITIALIZATION

The SDLC Receive mode is initialized by the system with the following parameters: SDLC mode, x1 clock mode, SDLC polynomial, receive word length, etc. The flag characters must also be loaded in WR7 and the secondary address field loaded in WR6. The receiver is enabled only after all the receive parameters have been set. After all this has been done, the receiver is in the Hunt phase and remains in this phase until the first flag is received. While in the SDLC mode, the receiver never re-enters the Hunt phase, unless specifically instructed to do so by the program. The WR4 parameters must be issued prior to the WR1, WR3, WR5, WR6 and WR7 parameters.

Under program control, the receiver can enter the Address Search mode. If the Address Search bit (WR3, D_2) is set, a character following the flag (first non-flag character) is compared against the programmed address in WR6 and the hardwired global address (11111111). If the SDLC frame address field matches either address, data transfer begins.

Since the Z80-SIO is capable of matching only one address character, extended address field recognition must be done by the CPU. In this case, the Z80-SIO simply transfers the additional address bytes to the CPU as if they were data characters. If the CPU determines that the frame does not have the correct address field, it can set the Hunt bit, and the Z80-SIO suspends reception and searches for a new message headed by a flag. Since the control field of the frame is transparent to the Z80-SIO, it is transferred to the CPU as a data character. Extra zeros inserted in the data stream are automatically deleted; flags are not transferred to the CPU.

5.3.2 DATA TRANSFER AND STATUS MONITORING

After receipt of a valid flag, the assembled characters are transferred to the receive data FIFO. The following four interrupt modes are available to transfer this data and its associated status.

No Interrupts Enabled. This mode is used for purely polled operations or for off-line conditions.

Interrupt On First Character Only. This mode is normally used to start a software polling loop or a Block Transfer instruction using $\overline{\text{WAIT}}/\overline{\text{READY}}$ to synchronize the CPU or DMA device to the incoming data rate. In this mode, the Z80-SIO interrupts on the first character and thereafter only interrupts if Special Receive conditions are detected. The mode is reinitialized with the Enable Interrupt On Next Receive Character Command.

The first character received after this command is issued causes an interrupt. If External/Status interrupts are enabled, they may interrupt any time the $\overline{\text{DCD}}$ input changes state. Special Receive conditions such as End Of Frame and Receiver Overrun also cause interrupts. The End of Frame interrupt can be used to exit the Block Transfer mode.

Interrupt On Every Character. An interrupt is generated whenever the receive FIFO contains a character. Error and Special Receive conditions generate a special vector if Status Affects Vector is selected.

Special Receive Condition Interrupts. The Special Receive Condition interrupt is not, as such, a separate interrupt mode. Before the Special Receive condition can cause an interrupt, either Interrupt On First Receive Character Only or Interrupt On Every Character must be selected. The Special Receive Condition interrupt is caused by a Receive Overrun or End of Frame detection. Since the Receive Overrun status bit is latched, the error status read reflects an error in the current word in the receive buffer in addition to any errors received since the last Error Reset command. The Receive Overrun status bit can only be reset by the Error Reset command. The End Of Frame status bit indicates that a valid ending flag has been received and that the CRC Error and Residue codes are also valid.

Character length may be changed on the fly. If the address and control bytes are processed as 8-bit characters, the receiver may be switched to a shorter character length during the time that the first information character is being assembled. This change must be made fast enough so it is effective before the number of bits specified for the character length have been assembled. For example, if the change is to be from the 8-bit control field to a 7-bit information field, the change must be made before the first seven bits of the I-field are assembled.

SDLC Receive CRC Checking. Control of the receive CRC checker is automatic. It is reset by the leading flag and CRC is calculated up to the final flag. The byte that has the End Of Frame bit set is the byte that contains the result of the CRC check. If the CRC/Framing Error bit is not set, the CRC indicates a valid message. A special check sequence is used for the SDLC check because the transmitted CRC check is inverted. The final check must be 0001110100001111. The 2-byte CRC check characters must be read by the CPU and discarded because the Z80-SIO, while using them for CRC checking, treats them as ordinary data.

SDLC Receive Termination. If enabled, a special vector is generated when the closing flag is received. This signals that the byte with the End Of Frame bit set has been received. In addition to the results of the CRC check, RR1 has three bits of Residue code valid at this time. For those cases in which the number of bits in the I-field is not an integral multiple of the character length used, these bits indicate the boundary between the CRC check bits and the I-field bits. For a detailed description of the meaning of these bits, see the description of the residue codes in RR1 under "Z80-SIO Programming".

Any frame can be prematurely aborted by an Abort sequence. Aborts are detected if seven or more 1's occur and cause an External/Status interrupt (if enabled) with the Break/Abort bit in RR0 set. After the Reset External/Status interrupts command has been issued a second interrupt occurs when the continuous 1's condition has been cleared. This can be used to distinguish between the Abort and Idle line conditions.

Unlike the synchronous mode, CRC calculation in SDLC does not have an 8-bit delay since all the characters are included in CRC calculation. When the second CRC character is loaded into the receive buffer, CRC calculation is complete.

Table 5.3 shows the typical steps required to implement a half-duplex SDLC receive mode. The complete set of command and status bit definitions is found in the next section.

SDLC RECEIVE MODE
Table 5.3

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
	REGISTER:	**INFORMATION LOADED**	
	WR0	CHANNEL 2	Reset SIO
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 4	
	WR4	PARITY INFORMATION, SYNC MODE, SDLC MODE, x1 CLOCK MODE	
	WR0	POINTER 5, RESET EXTERNAL/STATUS INTERRUPTS	
	WR5	SDLC-CRC, DATA TERMINAL READY	
	WR0	POINTER 3	
	WR3	RECEIVE CRC ENABLE, ENTER HUNT MODE, AUTO ENABLES RECEIVE CHARACTER LENGTH, ADDRESS SEARCH MODE	"Auto Enables" enables the receiver to accept data only after $\overline{DCD}$ becomes active. Address Search Mode enables SIO to match the message address with the programmed address or the global address.
INITIALIZE	WR0	POINTER 6	
	WR6	SECONDARY ADDRESS FIELD	This address is matched against the message address in an SDLC poll operation.
	WR0	POINTER 7	
	WR7	SDLC FLAG 01111110	This flag detects the start and end of frame in an SDLC operation.
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPTS	In this interrupt mode, only the Address Field (1 character only) is transferred to CPU. All subsequent fields (Control, Information, etc.) are transferred on a DMA basis. Status Affects Vector in Channel B only.
	WR1	STATUS AFFECTS VECTOR, EXTERNAL INTERRUPT ENABLE, RECEIVE INTERRUPT ON FIRST CHARACTER ONLY.	

FUNCTION	TYPICAL PROGRAM STEPS	COMMENTS
WR0 WR3	POINTER 3, ENABLE INTERRUPT ON NEXT RECEIVE CHARACTER RECEIVE ENABLE, RECEIVE CRC ENABLE, ENTER HUNT MODE, AUTO ENABLE, RECEIVER CHARACTER LENGTH, ADDRESS SEARCH MODE	Used to provide simple loop-back entry point for next transaction. WR3 reissued to enable receiver.
IDLE MODE	EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	SDLC Receive Mode is fully initialized and SIO is waiting for the opening flag followed by a matching address field to interrupt the CPU.
	WHEN INTERRUPT ON FIRST CHARACTER OCCURS, THE CPU DOES THE FOLLOWING: • TRANSFERS DATA BYTE (ADDRESS BYTE) TO CPU • DETECTS AND SETS APPROPRIATE FLAG FOR EXTENDED ADDRESS FIELD • UPDATES POINTER AND PARAMETERS • ENABLES DMA CONTROLLER • ENABLES WAIT/READY FUNCTION IN SIO • RETURNS FROM INTERRUPT	During the Hunt phase, the SIO interrupts when the programmed address matches the message address. The CPU establishes the DMA mode and all subsequent data characters are transferred by the DMA controller to memory.
	WHEN THE READY OUTPUT BECOMES ACTIVE, THE DMA CONTROLLER DOES THE FOLLOWING: TRANSFERS THE DATA BYTE TO MEMORY UPDATES THE POINTERS	During the DMA operation, the SIO monitors the $\overline{DCD}$ input and the Abort sequence in the data stream to interrupt the CPU with External Status error. The Special Receive condition interrupt is caused by Receive Overrun error.
DATA TRANSFER AND STATUS MONITORING	WHEN END OF FRAME INTERRUPT OCCURS, THE CPU DOES THE FOLLOWING: • EXITS DMA MODE (DISABLES WAIT/READY) • TRANSFERS RR1 TO THE CPU • CHECKS THE CRC ERROR BIT STATUS AND RESIDUE CODES • UPDATES NR COUNT • ISSUES "ERROR RESET" COMMAND TO SIO	Detection of End of Frame (Flag) causes interrupt and deactiviates the Wait/Ready function. Residue codes indicate the bit structure of the last two bytes of the message, which were transferred to memory under DMA. "Error Reset" is issued to clear the special condition.
	WHEN ABORT SEQUENCE DETECTED INTERRUPT OCCURS, THE CPU DOES THE FOLLOWING: • TRANSFERS RR0 TO THE CPU • EXITS DMA MODE • ISSUES THE RESET EXTERNAL STATUS INTERRUPT COMMAND TO THE SIO • ENTERS THE IDLE MODE	Abort sequence is detected when seven or more 1's are found in the data stream. CPU is waiting for Abort Sequence to terminate. Termination clears the Break/Abort status bit and causes interrupt.

FUNCTION	TYPICAL PROGRAM STEPS	COMMENTS
	WHEN THE SECOND ABORT SEQUENCE INTERRUPT OCCURS, THE CPU DOES THE FOLLOWING: • ISSUES THE RESET EXTERNAL STATUS INTERRUPT COMMAND TO THE SIO.	At this point, the program proceeds to terminate this message.
TERMINATION	REDEFINE INTERRUPT MODES, SYNC MODES AND SDLC MODES; DISABLE RECEIVE MODE	

6.0 Z80-SIO PROGRAMMING

6.1 INTRODUCTION

To program the Z80-SIO, the system program first issues a series of commands that initialize the basic mode of operation and then other commands that qualify conditions within the selected mode. For example, the Asynchronous mode, character length, clock rate, number of stop bits, even or odd parity are first set, then the interrupt mode, and finally, receiver or transmitter enable. The WR4 parameters must be issued before any other parameters are issued in the initialization routine.

Both channels contain command registers that must be programmed via the system program prior to operation. The Channel Select input ($B/\overline{A}$) and the Control/Data input ($C/\overline{D}$) are the command structure addressing controls, and are normally controlled by the CPU address bus. Figures 8.1 - 8.4 illustrate the timing relationships for programming the write registers, and transferring data and status.

$C/\overline{D}$	$B/\overline{A}$	Function
0	0	Channel A Data
0	1	Channel B Data
1	0	Channel A Commands/Status
1	1	Channel B Commands/Status

III

WRITE REGISTERS

The Z80-SIO contains eight registers (WR0-WR7) in each channel that are programmed separately by the system program to configure the functional personality of the channels. With the exception of WR0, programming the write registers requires two bytes. The first byte contains three bits (D_0-D_2) that point to the selected register; the second byte is the actual control word that is written into the register to configure the Z80-SIO.

Note that the programmer has complete freedom, after pointing to the selected register, of either reading to test the read register or writing to initialize the write register. By designing software to initialize the Z80-SIO in a modular and structured fashion, the programmer can use powerful block I/O instructions.

WR0 is a special case in that all the basic commands (CMD_0-CMD_2) can be accessed with a single byte. Reset (internal or external) initializes the pointer bits (D_0-D_2) to point to WR0.

The basic commands (CMD_0-CMD_2) and the CRC controls (CRC_0, CRC_1) are contained in the first byte of any write register access. This maintains maximum flexibility and system control. Each channel contains the following control registers. These registers are addressed as commands (not data).

6.2 WRITE REGISTER 0

WR0 is the command register; however, it is also used for CRC reset codes and to point to the other registers.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CRC Reset Code 1	CRC Reset Code 0	CMD 2	CMD 1	CMD 0	PTR 2	PTR 1	PTR 0

Pointer Bits (D_0-D_2). Bits D_0-D_2 are pointer bits that determine which other write register the next byte is to be written into or which read register the next byte is to be read from. The first byte written into each channel after a reset (either by a Reset command or by the external reset input) goes into WR0. Following a read or write to any register (except WR0), the pointer will point to WR0.

Command Bits (D_3-D_5). Three bits, D_3-D_5, are encoded to issue the seven basic Z80-SIO commands.

COMMAND	CMD_2	CMD_1	CMD_0	
0	0	0	0	Null Command (no effect)
1	0	0	1	Send Abort (SDLC Mode)
2	0	1	0	Reset External/Status Interrupts
3	0	1	1	Channel Reset
4	1	0	0	Enable Interrupt on next Rx Character
5	1	0	1	Reset Transmitter Interrupt Pending
6	1	1	0	Error Reset (latches)
7	1	1	1	Return from Interrupt (Channel A)

Command 0 (Null). The Null command has no effect. Its normal use is to cause the Z80-SIO to do nothing while the pointers are set for the following byte.

Command 1 (Send Abort). This command is used only with the SDLC mode to generate a sequence of eight to thirteen 1's.

Command 2 (Reset External/Status Interrupts). After an External/Status interrupt (a change on a modem line or a break condition, for example), the status bits of RR0 are latched. This command re-enables them and allows interrupts to occur again. Latching the status bits captures short pulses until the CPU has time to read the change.

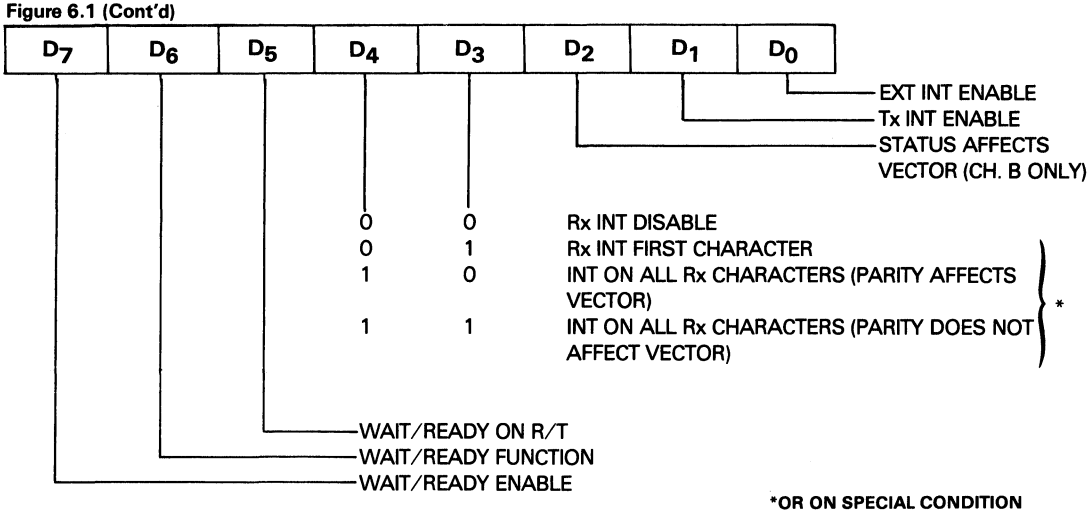
Command 3 (Channel Reset). This command performs the same function as an External Reset, but only on a single channel. Channel A Reset also resets the interrupt prioritization logic. All control registers for the channel must be rewritten after a Channel Reset command.

WRITE REGISTER BIT FUNCTIONS
Figure 6.1

WRITE REGISTER 0

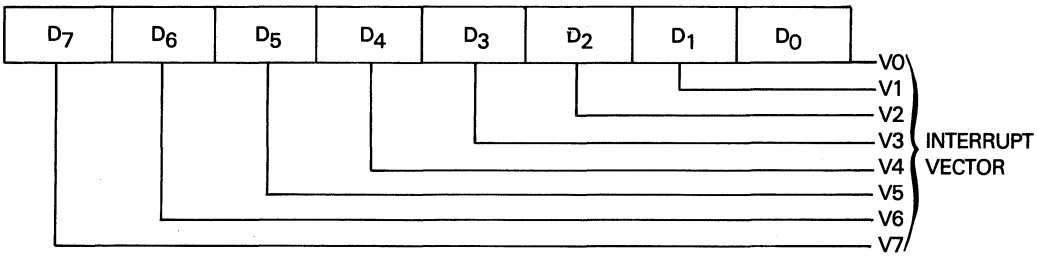
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
					0	0	0	REGISTER 0
					0	0	1	REGISTER 1
					0	1	0	REGISTER 2
					0	1	1	REGISTER 3
					1	0	0	REGISTER 4
					1	0	1	REGISTER 5
					1	1	0	REGISTER 6
					1	1	1	REGISTER 7
		0	0	0	NULL CODE			
		0	0	1	SEND ABORT (SDLC)			
		0	1	0	RESET EXT/STATUS INTERRUPTS			
		0	1	1	CHANNEL RESET			
		1	0	0	ENABLE INT ON NEXT Rx CHARACTER			
		1	0	1	RESET Tx INT PENDING			
		1	1	0	ERROR RESET			
		1	1	1	RETURN FROM INT (CH-A ONLY)			
0	0	NULL CODE						
0	1	RESET Rx CRC CHECKER						
1	0	RESET Tx CRC GENERATOR						
1	1	RESET Tx UNDERRUN/EOM LATCH						

WRITE REGISTER 1
Figure 6.1 (Cont'd)

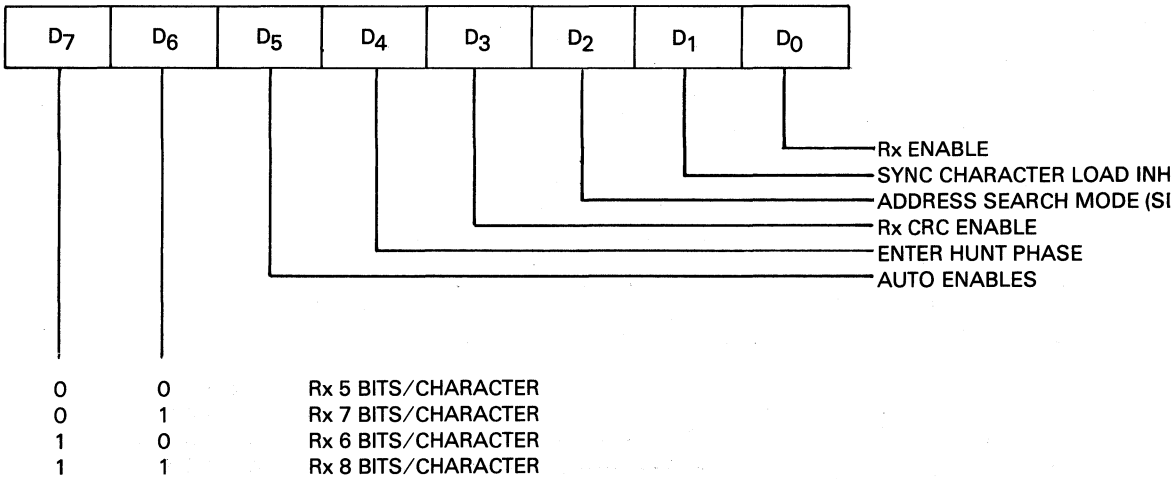


***OR ON SPECIAL CONDITION**

WRITE REGISTER 2 (CHANNEL B ONLY)



WRITE REGISTER 3



WRITE REGISTER 4

Figure 6.1 (Cont'd)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

D_0 — PARITY ENABLE

D_1 — PARITY EVEN ODD

D_3	D_2	
0	0	SYNC MODES ENABLE
0	1	1 STOP BIT/CHARACTER
1	0	1 1/2 STOP BITS CHARACTER
1	1	2 STOP BITS/CHARACTER

D_5	D_4	
0	0	8 BIT SYNC CHARACTER
0	1	16 BIT SYNC CHARACTER
1	0	SDLC MODE (01111110 FLAG)
1	1	EXTERNAL SYNC MODE

D_7	D_6	
0	0	X1 CLOCK MODE
0	1	X16 CLOCK MODE
1	0	X32 CLOCK MODE
1	1	X64 CLOCK MODE

WRITE REGISTER 5

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

D_0 — Tx CRC ENABLE

D_1 — RTS

D_2 — $\overline{\text{SDLC}}$/CRC-16

D_3 — Tx ENABLE

D_4 — SEND BREAK

D_6	D_5	
0	0	Tx 5 BITS (OR LESS)/CHARACTER
0	1	Tx 7 BITS/CHARACTER
1	0	Tx 6 BITS/CHARACTER
1	1	Tx 8 BITS/CHARACTER

D_7 — DTR

WRITE REGISTER 6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

D_0 — SYNC BIT 0

D_1 — SYNC BIT 1

D_2 — SYNC BIT 2

D_3 — SYNC BIT 3

D_4 — SYNC BIT 4

D_5 — SYNC BIT 5

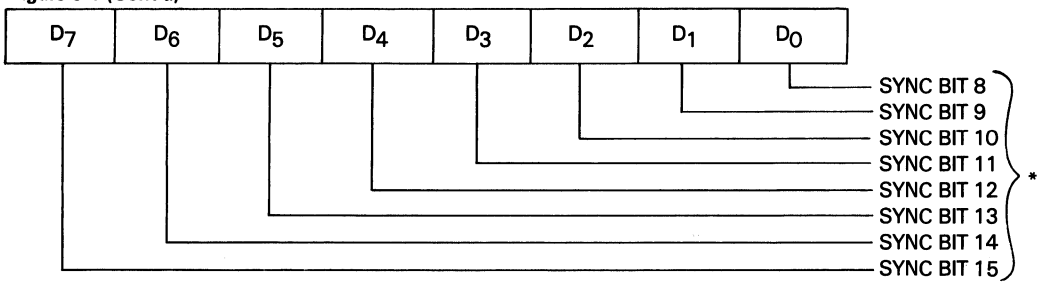
D_6 — SYNC BIT 6

D_7 — SYNC BIT 7

*

***ALSO SDLC ADDRESS FIELD**

WRITE REGISTER 7
Figure 6.1 (Cont'd)



*FOR SDLC, IT MUST BE PROGRAMMED TO "01111110" FOR FLAG RECOGNITION

After a Channel Reset, four extra system clock cycles should be allowed for Z80-SIO reset time before any additional commands or controls are written into that channel. This can normally be the time used by the CPU to fetch the next op code.

Command 4 (Enable Interrupt On Next Character). If the Interrupt On First Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the Z80-SIO for the next message.

Command 5 (Reset Transmitter Interrupt Pending). The transmitter interrupts when the transmit buffer becomes empty if the Transmit Interrupt Enable mode is selected. In those cases where there are no more characters to be sent (at the end of message, for example), issuing this command prevents further transmitter interrupts until after the next character has been loaded into the transmit buffer or until CRC has been completely sent.

Command 6 (Error Reset). This command resets the error latches. Parity and Overrun errors are latched in RR1 until they are reset with this command. With this scheme, parity errors occurring in block transfers can be examined at the end of the block.

Command 7 (Return From Interrupt). This command must be issued in Channel A and is interpreted by the Z80-SIO in exactly the same way it would interpret a RETI command on the data bus. It resets the interrupt-under-service latch of the highest-priority internal device under service and thus allows lower priority devices to interrupt via the daisy chain. This command allows use of the internal daisy chain even in systems with no external daisy chain or RETI command.

CRC Reset Codes 0 and 1 (D_6 and D_7). Together, these bits select one of the three following reset commands:

CRC Reset Code 1	CRC Reset Code 0	
0	0	Null Code (no effect)
0	1	Reset Receive CRC Checker
1	0	Reset Transmit CRC Generator
1	1	Reset Tx Underrun/End Of Message Latch

The Reset Transmit CRC Generator command normally initializes the CRC generator to all 0's. If the SDLC mode is selected, this command initializes the CRC generator to all 1's. The Receive CRC checker is also initialized to all 1's for the SDLC mode.

6.3 WRITE REGISTER 1

WR1 contains the control bits for the various interrupt and Wait/Ready modes.

D_7 Wait/Ready Enable	D_6 Wait Or Ready Function	D_5 Wait/Ready On Receive/Transmit	D_4 Receive Interrupt Mode 1
D_3 Receive Interrupt Mode 0	D_2 Status Affects Vector	D_1 Transmit Interrupt Enable	D_0 External Interrupts Enable

External/Status Interrupt Enable (D_0). The External/Status Interrupt Enable allows interrupts to occur as a result of transitions on the $\overline{DCD}$, $\overline{CTS}$ or $\overline{SYNC}$ inputs, as a result of a Break/Abort detection and termination, or at the beginning of CRC or sync character transmission when the Transmit Underrun/EOM latch becomes set.

Transmitter Interrupt Enable (D_1). If enabled, the interrupts occur whenever the transmitter buffer becomes empty.

Status Affects Vector (D_2). This bit is active in Channel B only. If this bit is not set, the fixed vector programmed in WR2 is returned from an interrupt acknowledge sequence. If this bit is set, the vector returned from an interrupt acknowledge is variable according to the following interrupt conditions:

	V_3	V_2	V_1	
Ch B	0	0	0	Ch B Transmit Buffer Empty
	0	0	1	Ch B External/Status Change
	0	1	0	Ch B Receive Character Available
	0	1	1	Ch B Special Receive Condition*
Ch A	1	0	0	Ch A Transmit Buffer Empty
	1	0	1	Ch A External/Status Change
	1	1	0	Ch A Receive Character Available
	1	1	1	Ch A Special Receive Condition*

*Special Receive Conditions: Parity Error, Rx Overrun Error, Framing Error, End Of Frame (SDLC).

Receive Interrupt Modes 0 and 1 (D_3 and D_4). Together, these two bits specify the various character-available conditions. In Receive Interrupt modes 1, 2 and 3, a Special Receive Condition can cause an interrupt and modify the interrupt vector.

D_4 Receive Interrupt Mode 1	D_3 Receive Interrupt Mode 0	
0	0	0. Receive Interrupts Disabled
0	1	1. Receive Interrupt On First Character Only
1	0	2. Interrupt On All Receive Characters—parity error is a Special Receive condition
1	1	3. Interrupt On All Receive Characters—parity error is not a Special Receive condition

Wait/Ready Function Selection (D_5-D_7). The Wait and Ready functions are selected by controlling D_5, D_6 and D_7. Wait/Ready function is enabled by setting Wait/Ready Enable (WR1, D_7) to 1. The Ready Function is selected by setting D_6 (Wait/Ready function) to 1. If this bit is 1, the $\overline{\text{WAIT/READY}}$ output switches from High to Low when the Z80-SIO is ready to transfer data. The Wait function is selected by setting D_6 to 0. If this bit is 0, the $\overline{\text{WAIT/READY}}$ output is in the open-drain state and goes Low when active.

Both the Wait and Ready functions can be used in either the Transmit or Receive modes, but not both simultaneously. If D_5 (Wait/Ready or Receive/Transmit) is set to 1, the Wait/Ready function responds to the condition of the receive buffer (empty or full). If D_5 is set to 0, the Wait/Ready function responds to the condition of the transmit buffer (empty or full).

The logic states of the $\overline{\text{WAIT/READY}}$ output when active or inactive depend on the combination of modes selected. Following is a summary of these combinations:

	And D_6 = 1	If D_7 = 0	And D_6 = 0
	$\overline{\text{READY}}$ is High		$\overline{\text{WAIT}}$ is floating
	And D_5 = 0	If D_7 = 1	And D_5 = 1
$\overline{\text{READY}}$	Is High when transmit buffer is full.	$\overline{\text{READY}}$	Is High when receive buffer is empty.
$\overline{\text{WAIT}}$	Is Low when transmit buffer is full and an SIO data port is selected.	$\overline{\text{WAIT}}$	Is Low when receive buffer is empty and an SIO data port is selected.
$\overline{\text{READY}}$	Is Low when transmit buffer is empty.	$\overline{\text{READY}}$	Is Low when receive buffer is full.
$\overline{\text{WAIT}}$	Is floating when transmit buffer is empty.	$\overline{\text{WAIT}}$	Is Floating when receive buffer is full.

The $\overline{\text{WAIT}}$ output High-to-Low transition occurs when the delay time $t_DIC(WR)$ after the I/O request. The Low-to-High transition occurs with the delay $t_DH\Phi(WR)$ from the falling edge of Φ. The $\overline{\text{READY}}$ output High-to-Low transition occurs with the delay $t_DL\Phi(WR)$ from the rising edge of Φ. The $\overline{\text{READY}}$ output Low-to-High transition occurs with the delay $t_DIC(WR)$ after $\overline{\text{IORQ}}$ falls.

The Ready function can occur any time the Z80-SIO is not selected. When the $\overline{\text{READY}}$ output becomes active (Low), the DMA controller issues $\overline{\text{IORQ}}$ and the corresponding B/$\overline{\text{A}}$ and C/$\overline{\text{D}}$ inputs to the Z80-SIO to transfer data. The $\overline{\text{READY}}$ output becomes inactive as soon as $\overline{\text{IORQ}}$ and $\overline{\text{CS}}$ become active. Since the Ready function can occur internally in the Z80-SIO whether it is addressed or not, the $\overline{\text{READY}}$ output becomes inactive when any CPU data or command transfer takes place. This does not cause problems because the DMA controller is not enabled when the CPU transfer takes place.

The Wait function—on the other hand—is active only if the CPU attempts to read Z80-SIO data that has not yet been received, which occurs frequently when block transfer instructions are used. The Wait function can also become active (under program control) if the CPU tries to write data while the transmit buffer is still full. The fact that the $\overline{\text{WAIT}}$ output for either channel can become active when the opposite channel is addressed (because the Z80-SIO is addressed) does not affect operation of software loops or block move instructions.

6.4 WRITE REGISTER 2

WR2 is the interrupt vector register; it exists in Channel B only. V_4-V_7 and V_0 are always returned exactly as written; V_1-V_3 are returned as written if the Status Affects Vector (WR1, D_2) control bit is 0. If this bit is 1, they are modified as explained in the previous section.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0

6.5 WRITE REGISTER 3

WR3 contains receiver logic control bits and parameters.

D_7	D_6	D_5	D_4
Receiver Bits/ Char 1	Receiver Bits/ Char 0	Auto Enables	Enter Hunt Phase
D_3	D_2	D_1	D_0
Receiver CRC Enable	Address Search Mode	Sync Char Load Inhibit	Receiver Enable

Receiver Enable (D_0). A 1 programmed into this bit allows receive operations to begin. This bit should be set only after all other receive parameters are set and receiver is completely initialized.

Sync Character Load Inhibit (D_1). Sync characters preceding the message (leading sync characters) are not loaded into the receive buffers if this option is selected. Because CRC calculations are not stopped by sync character stripping, this feature should be enabled only at the beginning of the message.

Address Search Mode (D_2). If SDLC is selected, setting this mode causes messages with addresses not matching the programmed address in WR6 or the global (11111111) address to be rejected. In other words, no receive interrupts can occur in the Address Search mode unless there is an address match.

Receiver CRC Enable (D_3). If this bit is set, CRC calculation starts (or restarts) at the beginning of the last character transferred from the receive shift register to the buffer stack, regardless of the number of characters in the stack. See "SDLC Receive CRC Checking" (SDLC Receive section) and "CRC Error Checking" (Synchronous Receive section) for details regarding when this bit should be set.

Enter Hunt Phase (D_4). The Z80-SIO automatically enters the Hunt phase after a reset; however, it can be re-entered if character synchronization is lost for any reason (Synchronous mode) or if the contents of an incoming message are not needed (SDLC mode). The Hunt phase is re-entered by writing a 1 into bit D_4. This sets the Sync/Hunt bit (D_4) in RR0.

Auto Enables (D_5). If this mode is selected, $\overline{DCD}$ and $\overline{CTS}$ become the receiver and transmitter enables, respectively. If this bit is not set, $\overline{DCD}$ and $\overline{CTS}$ are simply inputs to their corresponding status bits in RR0.

Receiver Bits/Character 1 and 0 (D_7 and D_6). Together, these bits determine the number of serial receive bits assembled to form a character. Both bits may be changed during the time that a character is being assembled, but they must be changed before the number of bits currently programmed is reached.

D_7	D_6	Bits/Character
0	0	5
0	1	7
1	0	6
1	1	8

6.6 WRITE REGISTER 4

WR4 contains the control bits that affect both the receiver and transmitter. In the transmit and receive initialization routine, these bits should be set before issuing WR1, WR3, WR5, WR6, and WR7.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Clock Rate 1	Clock Rate 0	Sync Modes 1	Sync Modes 0	Stop Bits 1	Stop Bits 0	Parity Even/$\overline{\text{Odd}}$	Parity

Parity (D_0). If this bit is set, an additional bit position (in addition to those specified in the bits/character control) is added to transmitted data and is expected in receive data. In the Receive mode, the parity bit received is transferred to the CPU as part of the character, unless 8 bits/character is selected.

Parity Even $\overline{\text{Odd}}$ (D_1). If parity is specified, this bit determines whether it is sent and checked as even or odd (1=even).

Stop Bits 0 and 1 (D_2 and D_3). These bits determine the number of stop bits added to each asynchronous character sent. The receiver always checks for one stop bit. A special mode (00) signifies that a synchronous mode is to be selected.

D_3 Stop Bits 1	D_2 Stop Bits 0	
0	0	Sync modes
0	1	1 stop bit per character
1	0	1½ stop bits per character
1	1	2 stop bits per character

Sync Modes 0 and 1 (D_4 and D_5). These bits select the various options for character synchronization.

Sync Mode 1	Sync Mode 0	
0	0	8-bit programmed sync
0	1	16-bit programmed sync
1	0	SDLC mode (01111110 flag pattern)
1	1	External Sync mode

Clock Rate 0 and 1 (D_6 and D_7). These bits specify the multiplier between the clock ($\overline{\text{TxC}}$ and $\overline{\text{RxC}}$) and data rates. For synchronous modes, the x1 clock rate must be specified. Any rate may be specified for asynchronous modes; however, the same rate must be used for both the receiver and transmitter. The system clock in all modes must be at least 5 times the data rate. If the x1 clock rate is selected, bit synchronization must be accomplished externally.

Clock Rate 1	Clock Rate 0	
0	0	Data Rate x1=Clock Rate
0	1	Data Rate x16=Clock Rate
1	0	Data Rate x32=Clock Rate
1	1	Data Rate x64=Clock Rate

6.7 WRITE REGISTER 5

WR5 contains control bits that affect the operation of transmitter, with the exception of D2, which affects the transmitter and receiver.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
DTR	Tx Bits/ Char 1	Tx Bits/ Char 0	Send Break	Tx Enable	CRC-16/ SDLC	RTS	Tx CRC Enable

Transmit CRC Enable (D_0). This bit determines if CRC is calculated on a particular transmit character. If it is set at the time the character is loaded from the transmit buffer into the transmit shift register, CRC is calculated on the character. CRC is not automatically sent unless this bit is set when the Transmit Underrun condition exists.

Request To Send (D_1). This is the control bit for the $\overline{RTS}$ pin. When the $\overline{RTS}$ bit is set, the $\overline{RTS}$ pin goes Low; when reset, $\overline{RTS}$ goes High. In the Asynchronous mode, $\overline{RTS}$ goes High only after all the bits of the character are transmitted and the transmitter buffer is empty. In Synchronous modes, the pin directly follows the state of the bit.

CRC-16/$\overline{SDLC}$ (D_2). This bit selects the CRC polynomial used by both the transmitter and receiver. When set, the CRC-16 polynomial ($X^{16} + X^{15} + X^2 + 1$) is used; when reset, the SDLC polynomial ($X^{16} + X^{12} + X^5 + 1$) is used. If the SDLC mode is selected, the CRC generator and checker are preset to all 1's and a special check sequence is used. The SDLC CRC polynomial must be selected when the SDLC mode is selected. If the SDLC mode is not selected, the CRC generator and checker are present to all 0's (for both polynomials).

Transmit Enable (D_3). Data is not transmitted until this bit is set and the Transmit Data output is held marking. Data or sync characters in the process of being transmitted are completely sent if this bit is reset after transmission has started. If the transmitter is disabled during the transmission of a CRC character, sync or flag characters are sent instead of CRC.

Send Break (D_4). When set, this bit immediately forces the Transmit Data output to the spacing condition, regardless of any data being transmitted. When reset, TxD returns to marking.

Transmit Bits/Character 0 and 1 (D_5 and D_6). Together, D_6 and D_5 control the number of bits in each byte transferred to the transmit buffer.

D_6 Transmit Bits/ Character 1	D_5 Transmit Bits/ Character 0	Bits/Character
0	0	Five or less
0	1	7
1	0	6
1	1	8

Bits to be sent must be right justified, least-significant bits first. The Five Or Less mode allows transmission of one to five bits per character; however, the CPU should format the data character as shown in the following table.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	1	1	1	0	0	0	D	Sends one data bit
1	1	1	0	0	0	D	D	Sends two data bits
1	1	0	0	0	D	D	D	Sends three data bits
1	0	0	0	D	D	D	D	Sends four data bits
0	0	0	D	D	D	D	D	Sends five data bits

Data Terminal Ready (D_7). This is the control bit for the $\overline{DTR}$ pin. When set, $\overline{DTR}$ is active (Low); when reset, $\overline{DTR}$ is inactive (High).

6.8 WRITE REGISTER 6

This register is programmed to contain the transmit sync character in the Monosync mode, the first eight bits of a 16-bit sync character in the Bisync mode or a transmit sync character in the External Sync mode. In the SDLC mode, it is programmed to contain the secondary address field used to compare against the address field of the SDLC frame.

D_7 Sync 7	D_6 Sync 6	D_5 Sync 5	D_4 Sync 4	D_3 Sync 3	D_2 Sync 2	D_1 Sync 1	D_0 Sync 0

6.9 WRITE REGISTER 7

This register is programmed to contain the receive sync character in the Monosync mode, a second byte (last eight bits) of a 16-bit sync character in the Bisync mode and a flag character (01111110) in the SDLC mode. WR7 is not used in the External Sync mode.

D_7 Sync 15	D_6 Sync 14	D_5 Sync 13	D_4 Sync 12	D_3 Sync 11	D_2 Sync 10	D_1 Sync 9	D_0 Sync 8

7.0 READ REGISTERS

7.1 INTRODUCTION

The Z80-SIO contains three registers, RR0-RR2 (Figure 7.1), that can be read to obtain the status information for each channel (except for RR2-Channel B only). The status information includes error conditions, interrupt vector and standard communications-interface signals.

To read the contents of a selected read register other than RR0, the system program must first write the pointer byte to WR0 in exactly the same way as a write register operation. Then, by executing an input instruction, the contents of the addressed read register can be read by the CPU.

The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring. For example, when the interrupt vector indicates that a Special Receive Condition interrupt has occurred, all the appropriate error bits can be read from a single register (RR1).

7.2 READ REGISTER 0

III

This register contains the status of the receive and transmit buffers, the $\overline{DCD}$, $\overline{CTS}$ and $\overline{SYNC}$ inputs, the Transmit Underrun/EOM latch; and the Break/Abort latch.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Break Abort	Transmit Underrun/ EOM	CTS	Sync/ Hunt	DCD	Transmit Buffer Empty	Interrupt Pending (Ch. A only)	Receive Character Available

Receive Character Available (D_0). This bit is set when at least one character is available in the receive buffer; it is reset when the receive FIFO is completely empty.

Interrupt Pending (D_1). Any interrupting condition in the Z80-SIO causes this bit to be set; however, it is readable only in Channel A. This bit is mainly used in applications that do not have vectored interrupts available. During the interrupt service routine in these applications, this bit indicates if any interrupt conditions are present in all Z80-SIO. This eliminates the need for analyzing all the bits of RR0 in both Channels A and B. Bit D_1 is reset when all the interrupting conditions are satisfied. This bit is always 0 in Channel B.

Transmit Buffer Empty (D_2). This bit is set whenever the transmit buffer becomes empty, except when a CRC character is being sent in a synchronous or SDLC mode. The bit is reset when a character is loaded into the transmit buffer. This bit is in the set condition after a reset.

Data Carrier Detect (D_3). The DCD bit shows the inverted state of the $\overline{DCD}$ input at the time of the last change of any of the five External/Status bits (DCD, $\overline{CTS}$, Sync/Hunt, Break/Abort or Transmit Underrun/EOM). Any transition of the $\overline{DCD}$ input causes the DCD bit to be latched and causes an External/Status interrupt. To read the current state of the DCD bit, this bit must be read immediately following a Reset External/Status Interrupt command.

Sync/Hunt (D_4). Since this bit is controlled differently in the Asynchronous, Synchronous and SDLC modes, its operation is somewhat more complex than that of the other bits and, therefore, requires more explanation.

In Asynchronous modes, the operation of this bit is similar to the DCD status bit, except that Sync/Hunt shows the state of the $\overline{SYNC}$ input. Any High-to-Low transition on the $\overline{SYNC}$ pin sets this bit and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of $\overline{SYNC}$ pin at the time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{SYNC}$ input.

In the External Sync mode, the Sync/Hunt bit operates in a fashion similar to the Asynchronous mode, except the Enter Hunt Mode control bit enables the external sync detection logic. When the External

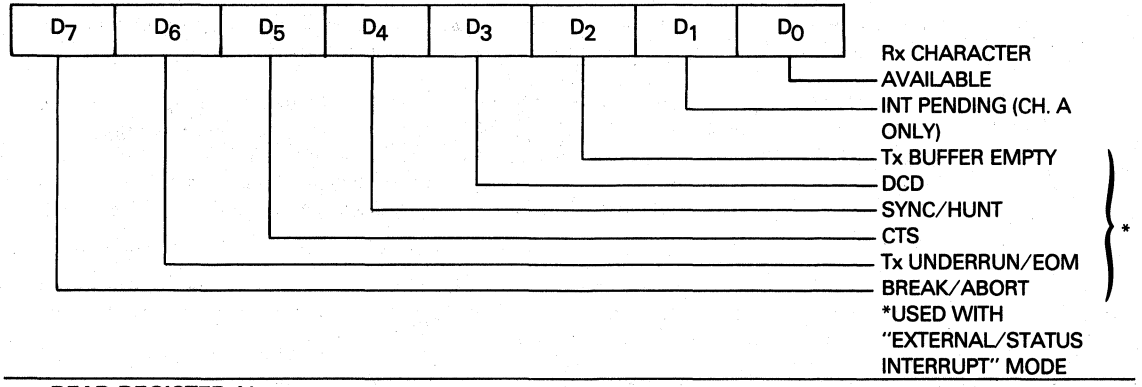
Sync Mode and Enter Hunt Mode bits are set (for example, when the receiver is enabled following a reset), the $\overline{SYNC}$ input must be held High by the external logic until external character synchronization is achieved. A High at the $\overline{SYNC}$ input holds the Sync/Hunt status bit in the reset condition.

When external synchronization is achieved, $\overline{SYNC}$ must be driven Low on the second rising edge of $\overline{RxC}$ on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive clock cycles to activate the $\overline{SYNC}$ input. Once $\overline{SYNC}$ is forced Low, it is a good practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. Refer to Figure 8.6 for timing details. The High-to-Low transition of the $\overline{SYNC}$ input sets the Sync/Hunt bit, which—in turn—sets the External/Status interrupt. The CPU must clear the interrupt by issuing the Reset External/Status Interrupt command.

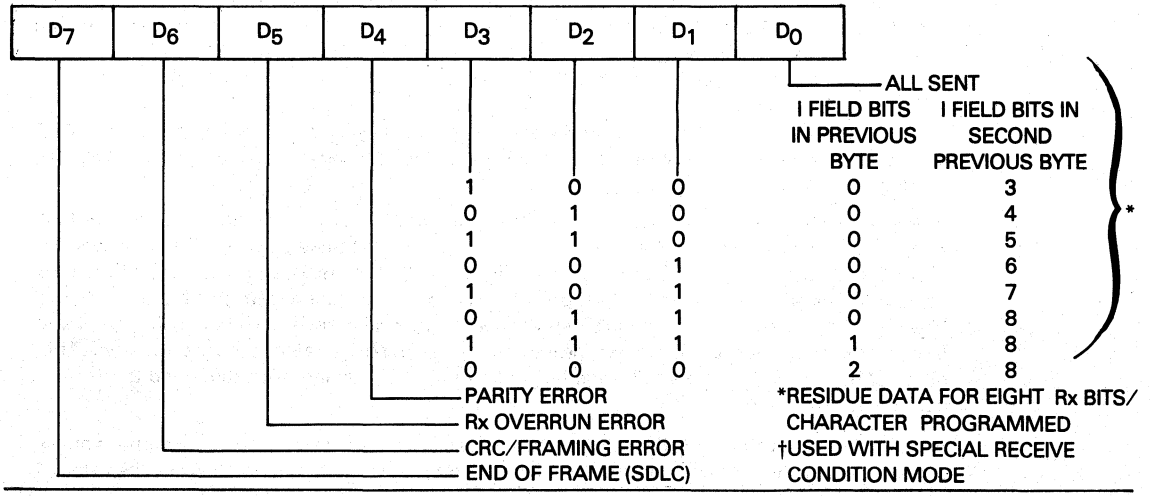
When the $\overline{SYNC}$ input goes High again, another External/Status interrupt is generated that must also be cleared. The Enter Hunt Mode control bit is set whenever character synchronization is lost or the end of message is detected. In this case, the Z80-SIO again looks for a High-to-Low transition on the $\overline{SYNC}$ input and the operation repeats as explained previously. This implies the CPU should also inform the external logic that character synchronization has been lost and that the Z80-SIO is waiting for $\overline{SYNC}$ to become active.

READ REGISTER BIT FUNCTIONS
Figure 7.1

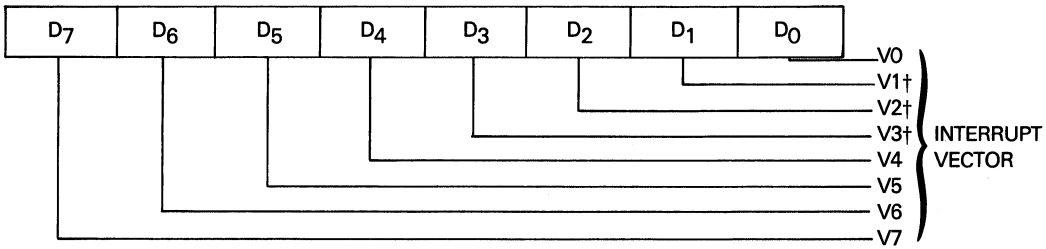
READ REGISTER 0



READ REGISTER 1†



READ REGISTER 2
Figure 7.1 (Cont'd)



†VARIABLE IF "STATUS AFFECTS VECTOR" IS PROGRAMMED

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt Mode bit. The Sync/Hunt bit is reset when the Z80-SIO establishes character synchronization. The High-to-Low transition of the Sync/Hunt bit causes an External/Status interrupt that must be cleared by the CPU issuing the Reset External/Status Interrupt command. This enables the Z80-SIO to detect the next transition of other External/Status bits.

When the CPU detects the end of message of that character and synchronization is lost, it sets the Enter Hunt Mode control bit, which—in turn—sets the Sync/Hunt bit to 1. The Low-to-High transition of the Sync/Hunt bit sets the External/Status interrupt, which must also be cleared by the Reset External/Status Interrupt command. Note that the SYNC pin acts as an output in this mode and goes Low every time a sync pattern is detected in the data stream.

In the SDLC mode, the Sync/Hunt bit is initially set by the Enter Hunt mode bit or when the receiver is disabled. In any case, it is reset to 0 when the opening flag of the first frame is detected by the Z80-SIO. The External/Status interrupt is also generated and should be handled as discussed previously.

Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in the SDLC mode, it does not need to be set when the end of message is detected. The Z80-SIO automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode bit or by disabling the receiver.

Clear to Send (D_5). This bit is similar to the DCD bit, except that it shows the inverted state of the $\overline{CTS}$ pin.

Transmit Underrun/End of Message (D_6). This bit is in a set condition following a reset (internal or external). The only command that can reset this bit is the Reset Transmit Underrun/EOM Latch command (WR0, D_6 and D_7). When the Transmit Underrun condition occurs, this bit is set; its becoming set causes the External/Status interrupt, which must be reset by issuing the Reset External/Status Interrupt command bits (WR0). This status bit plays an important role in conjunction with other control bits in controlling a transmit operation. Refer to "Bisync Transmit Underrun" and "SDLC Transmit Underrun" for additional details.

Break/Abort (D_7). In the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when Break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, CMD_2) to the break detection logic so the Break sequence termination can be recognized.

The Break/Abort bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

In the SDLC Receive mode, this status bit is set by the detection of an Abort sequence (seven or more 1's). The External/Status Interrupt is handled the same way as in the case of a Break. The Break/Abort bit is not used in the Synchronous Receive mode.

7.3 READ REGISTER 1

This register contains the Special Receive condition status bits and Residue codes for the I-field in the SDLC Receive Mode.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
End of Frame (SDLC)	CRC/ Framing Error	Receiver Overrun Error	Parity Error	Residue Code 2	Residue Code 1	Residue Code 0	All Sent

All Sent (D_0). In Asynchronous modes, this bit is set when all the characters have completely cleared the transmitter. Transitions of this bit do not cause interrupts. The bit is always set in Synchronous modes.

Residue Codes 0, 1, and 2 (D_1-D_3). In those cases of the SDLC receive mode where the I-field is not an integral multiple of the character length, these three bits indicate the length of the I-field. These codes are meaningful only for the transfer in which the End Of Frame bit is set (SDLC). For a receive character length of eight bits per character, the codes signify the following:

Residue Code 2	Residue Code 1	Residue Code 0	I-Field Bits In Previous Byte	I-Field Bits In Second Previous Byte
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8
	I-Field bits are right-justified in all cases			

If a receive character length different from eight bits is used for the I-field, a table similar to the previous one may be constructed for each different character length. For no residue (that is, the last character boundary coincides with the boundary of the I-field and CRC field), the Residue codes are:

Bits per Character	Residue Code 2	Residue Code 1	Residue Code 0
8 Bits per Character	0	1	1
7 Bits per Character	0	0	0
6 Bits per Character	0	1	0
5 Bits per Character	0	0	1

Parity Error (D_4). When parity is enabled, this bit is set for those characters whose parity does not match the programmed sense (even/odd). The bit is latched, so once an error occurs, it remains set until the Error Reset command (WR0) is given.

Receive Overrun Error (D_5). This bit indicates that more than three characters have been received without a read from the CPU. Only the character that has been written over is flagged with this error, but when this character is read, the error condition is latched until reset by the Error Reset command. If Status Affects Vector is enabled, the character that has been overrun interrupts with a Special Receive Condition vector.

CRC/Framing Error (D_6). If a Framing Error occurs (asynchronous modes), this bit is set (and not latched) for the receive character in which the Framing error occurred. Detection of a Framing Error adds an additional one-half of a bit time to the character time so the Framing Error is not interpreted as a new start bit. In Synchronous and SDLC modes, this bit indicates the result of comparing the CRC checker to the appropriate check value. This bit is reset by issuing an Error Reset command. The bit is

not latched, so it is always updated when the next character is received. When used for CRC error and status in Synchronous modes, it is usually set since most bit combinations result in a non-zero CRC, except for a correctly completed message.

End of Frame (D_7). This bit is used only with the SDLC mode and indicates that a valid ending flag has been received and that the CRC Error and Residue codes are also valid. This bit can be reset by issuing the Error Reset command. It is also updated by the first character of the following frame.

7.4 READ REGISTER 2 (Ch. B Only)

This register contains the interrupt vector written into WR2 if the Status Affects Vector control bit is not set. If the control bit is set, it contains the modified vector shown in the Status Affects Vector paragraph of the Write Register 1 section. When this register is read, the vector returned is modified by the highest priority interrupting condition at the time of the read. If no interrupts are pending, the vector is modified with V_3=0, V_2=1, and V_1=1. This register may be read only through Channel B.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0
				Variable if Status Affects Vector is enabled			

III

7.5 APPLICATIONS

The flexibility and versatility of the Z80-SIO make it useful for numerous applications, a few of which are included here. These examples show several applications that combine the Z80-SIO with other members of the Z80 family.

Figure 7.2 shows the simple processor-to-processor communication over a direct line. Both remote processors in this system can communicate to the Z80-CPU with different protocols and data rates. Depending on the complexity of the application, other Z80 peripheral circuits (Z80-CTC, for example) may be required. The unused channel of the Z80-SIO can be used to control other peripherals, or they can be connected to other remote processors.

Figure 7.3 illustrates how both channels of a single Z80-SIO are used with modems that have primary and secondary or reverse channel options. Alternatively, two modems without these options can be connected to the Z80-SIO. A suitable baud-rate generator (Z80-CTC) must be used for Asynchronous modems.

Figure 7.4 shows the Z80-SIO in a data concentrator, a relatively complex application that uses two Z80-SIOs to perform a variety of functions. The data concentrator can be used to collect data from many terminals over low-speed lines and transmit it over a single high-speed line after editing and reformatting.

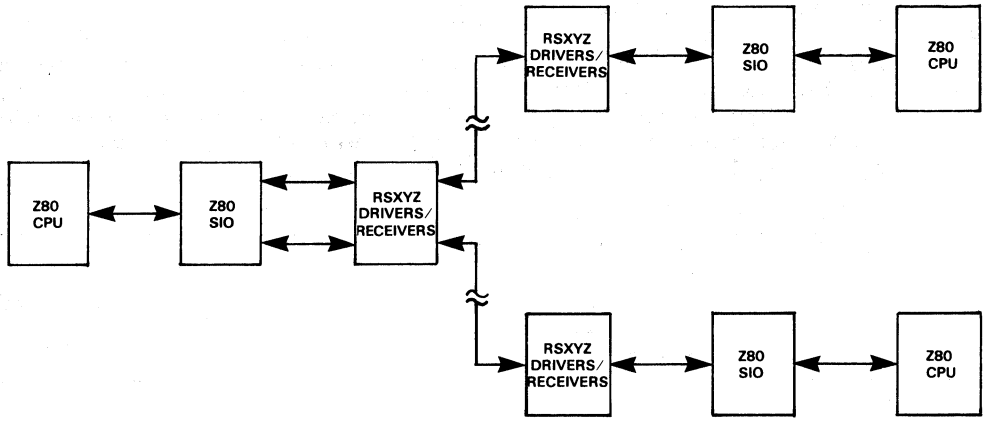
The Z80-DMA controller circuit is used with Z80-SIO #2 to transmit the reformatted data at high speed with the required protocol. The high-speed modem provides the transmit clock for this channel. The Z80-CTC counter-timer circuit supplies the transmit and receive clocks for the low-speed lines and is also used as a time-out counter for various functions.

The Z80-SIO #1 controls local or remote terminals. A single intelligent terminal is shown within the dashed lines. The terminal employs a Z80-SIO to communicate to the data concentrator on one channel while providing the interface to a line printer over its second channel. The intelligent terminal shown could be designed to operate interactively with the operator.

Depending on the software and hardware capabilities built into this system, the data concentrator can employ store-and-forward or hold-and-forward methods for regulating information traffic between slow terminals and the high-speed remote processor. If the high-speed channel is provided with a dial-out option, the channel can be connected to a number of remote processors over a switched line.

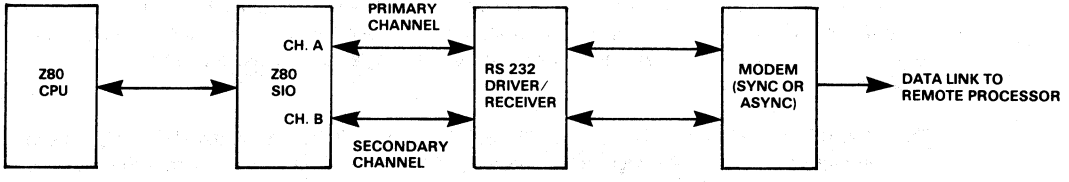
SYNCHRONOUS/ASYNCHRONOUS PROCESSOR-TO-PROCESSOR COMMUNICATION (USING TELEPHONE LINE)

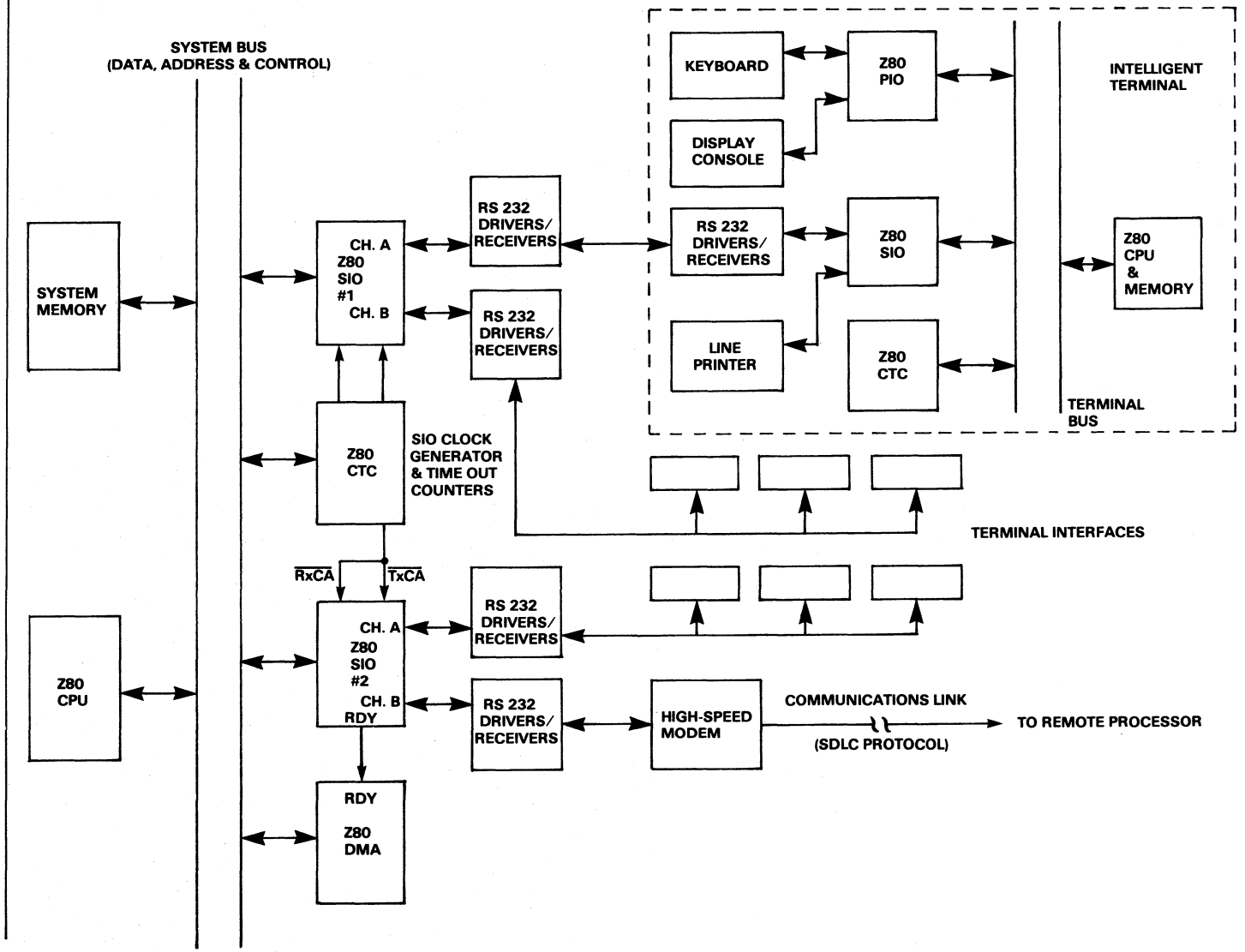
Figure 7.2



BOTH CHANNELS OF A SINGLE Z80-SIO

Figure 7.3





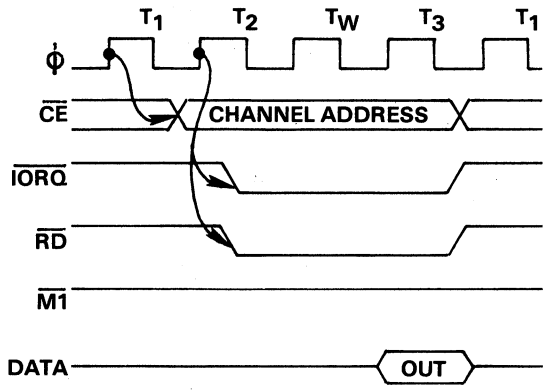
DATA CONCENTRATOR
Figure 7.4

8.0 TIMING

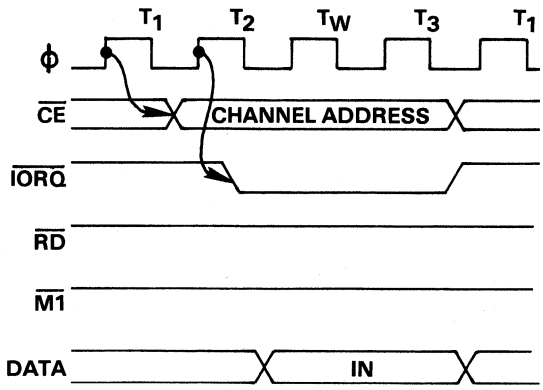
8.1 READ CYCLE

The timing signals generated by a Z80-CPU input instruction to read a Data or Status byte from the Z80-SIO are illustrated in Figure 8.1.

READ CYCLE
Figure 8.1



WRITE CYCLE
Figure 8.2



8.2 INTERRUPT ACKNOWLEDGE CYCLE

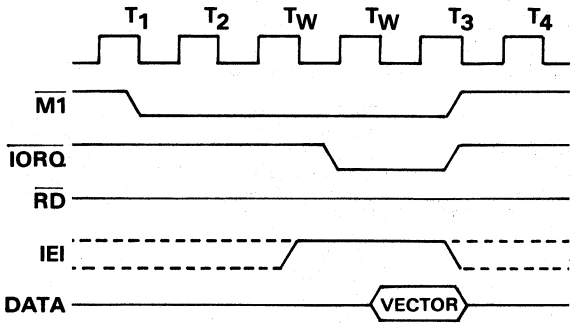
After receiving an Interrupt Request signal ($\overline{INT}$ pulled Low,) the Z80-CPU sends an Interrupt Acknowledge signal ($\overline{M1}$ and $\overline{IORQ}$ both Low). The daisy-chained interrupt circuits determine the highest priority interrupt requestor. The IEI of the highest priority peripheral is terminated High. For any peripheral that has no interrupt pending or under service, IEO=IEI. Any peripheral that does have an interrupt pending or under service forces its IEO Low.

To insure stable conditions in the daisy chain, all-interrupt status signals are prevented from changing while $\overline{M1}$ is Low. When $\overline{IORQ}$ is Low, the highest priority interrupt requestor (the one with IEI High) places its interrupt vector on the data bus and sets its internal interrupt-under-service latch.

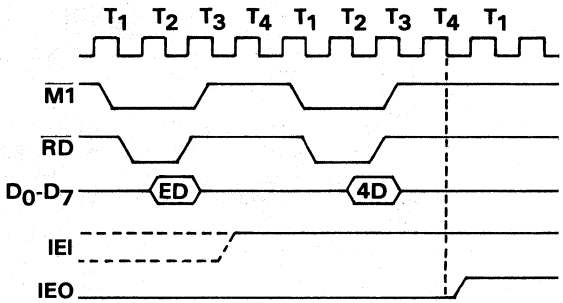
8.3 WRITE CYCLE

Figure 8.2 illustrates the timing and data signals generated by a Z80-CPU output instruction to write a Data or Control byte into the Z80-SIO.

ACKNOWLEDGE CYCLE
Figure 8.3



RETURN FROM INTERRUPT CYCLE



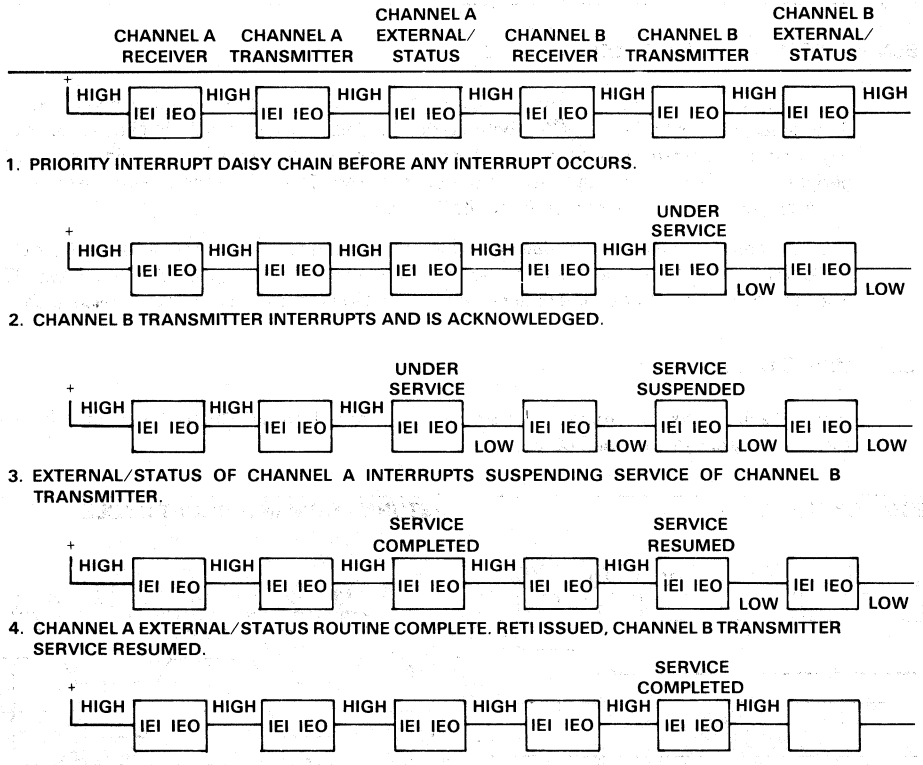
8.4 RETURN FROM INTERRUPT CYCLE

Normally, the Z80-CPU issues a RETI (Return from Interrupt) instruction at the end of an interrupt service routine. RETI is a 2-byte opcode (ED-4D) that resets the interrupt-under-service latch to terminate the interrupt that has just been processed. This is accomplished by manipulating the daisy chain in the following way.

The normal daisy chain operation can be used to detect a pending interrupt; however, it cannot distinguish between an interrupt under service and a pending unacknowledged interrupt of a higher priority. Whenever "ED" is decoded, the daisy chain is modified by forcing High the IEO of any interrupt that has not yet been acknowledged. Thus, the daisy chain identifies the device presently under service as the only one with an IEI High and an IEO Low. If the next opcode byte is "4D", the interrupt-under-service latch is reset.

The ripple time of the interrupt daisy chain (both the High-to-Low and the Low-to-High transitions) limits the number of devices that can be placed in the daisy chain. Ripple time can be improved with carry-look-read, or by extending the interrupt acknowledge cycle. For further information about techniques for increasing the number of daisy-chained devices, refer to Mostek's Application Note on extending the Z80 Interrupt Daisy Chain.

TYPICAL INTERRUPT SEQUENCE
Figure 8.4



1. PRIORITY INTERRUPT DAISY CHAIN BEFORE ANY INTERRUPT OCCURS.
2. CHANNEL B TRANSMITTER INTERRUPTS AND IS ACKNOWLEDGED.
3. EXTERNAL/STATUS OF CHANNEL A INTERRUPTS SUSPENDING SERVICE OF CHANNEL B TRANSMITTER.
4. CHANNEL A EXTERNAL/STATUS ROUTINE COMPLETE. RETI ISSUED, CHANNEL B TRANSMITTER SERVICE RESUMED.
5. CHANNEL B TRANSMITTER SERVICE ROUTINE COMPLETE, SECOND RETI ISSUED.

8.5 DAISY CHAIN INTERRUPT NESTING

Figure 8.4 illustrates the daisy chain configuration of interrupt circuits and their behavior with nested interrupts (an interrupt that is interrupted by another with a higher priority).

Each box in the illustration could be a separate external Z80 peripheral circuit with a user-defined order of interrupt priorities. However, a similar daisy chain structure also exists inside the Z80-SIO, which has six interrupt levels with a fixed order of priorities.

The case illustrated occurs when the transmitter of Channel B interrupts and is granted service. While this interrupt is being serviced, it is interrupted by a higher priority interrupt from Channel A. The second interrupt is serviced and—upon completion—a RETI instruction is executed or a RETI command is written into the Z80-SIO, resetting the interrupt-under-service latch of the Channel A interrupt. At this time, the service routine for Channel B is resumed. When it is completed, another RETI instruction is executed to complete the interrupt service.

9.0 ELECTRICAL SPECIFICATIONS

ABSOLUTE MAXIMUM RATINGS

Voltages on all inputs and outputs with respect to GND ..... -0.3V to +7.0V
Operating Ambient Temperature ..... As Specified in Ordering Information
Storage Temperature ..... -65°C to +150°C

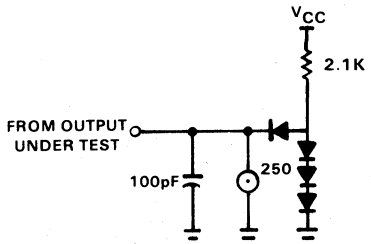
Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The characteristics below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin. Standard conditions are as follows:

- $+4.75V \leq V_{CC} \leq +5.25V$
- GND = 0V
- T_A as specified in Ordering Information

All ac parameters assume a load capacitance of 100 pF max. Timing references between two output signals assume a load difference of 50 pF max.



III

DC CHARACTERISTICS

SYM	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
V_{ILC}	Clock Input Low Voltage	-0.3	+0.80	V	
V_{IHC}	Clock Input High Voltage	V_{CC} -0.6	+5.5	V	
V_{IL}	Input Low Voltage	-0.3	+0.8	V	
V_{IH}	Input High Voltage	+2.0	+5.5	V	
V_{OL}	Output Low Voltage		+0.4	V	I_{OL} = 2.0mA
V_{OH}	Output High Voltage	+2.4		V	I_{OH} = -250 μA
I_{LI}	Input Leakage Current	-10	±10	μA	$0 < V_{IN} < V_{CC}$
I_Z	3-State Output/Data Bus Input Leakage Current	-10	+10	μA	$0 < V_{IN} < V_{CC}$
$I_{L(SY)}$	SYNC Pin Leakage Current	-40	+10	μA	$0 < V_{IN} < V_{CC}$
I_{CC}	Power Supply Current		100	mA	

Overall specified temperature and voltage range.

CAPACITANCE

SYM	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
C	Clock Capacitance		40	pF	Unmeasured
C_{IN}	Input Capacitance		10	pF	pins returned
C_{OUT}	Output Capacitance		10	pF	to ground

Over specified temperature range; f = 1MHz

AC ELECTRICAL CHARACTERISTICS

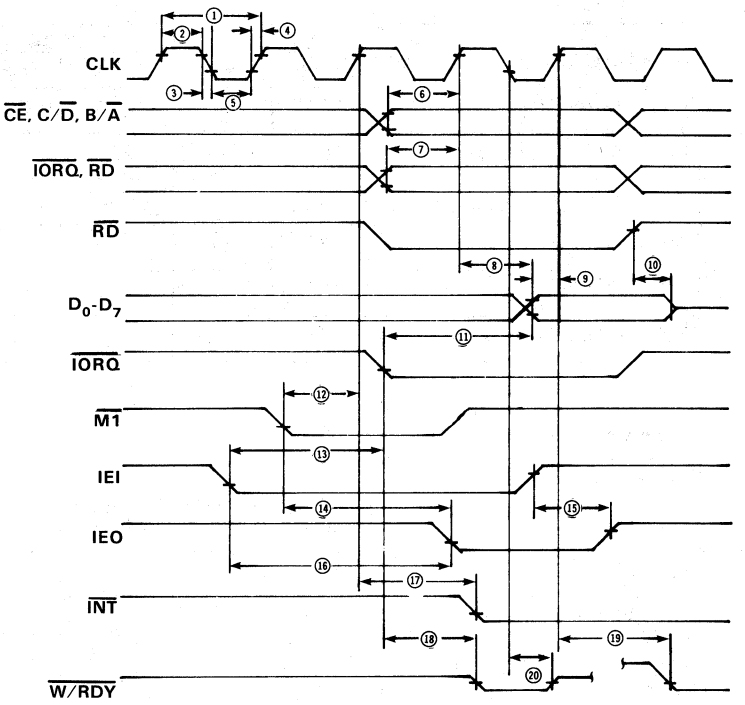
NUMBER	SYM	PARAMETER	MK3884		MK3884-4		UNIT
			MIN	MAX	MIN	MAX	
1	TcC	Clock Cycle Time	400	4000	250	4000	ns
2	TwCh	Clock Width (High)	170	2000	105	2000	ns
3	TfC	Clock Fall Time		30		30	ns
4	TrC	Clock Rise Time		30		30	ns
5	TwC1	Clock Width (Low)	170	2000	105	2000	ns
6	TsAD(C)	$\overline{CE}$, C/$\overline{D}$, B/$\overline{A}$ to Clock ↑ Setup Time	160		145		ns
7	TsCS(C)	$\overline{IORQ}$, $\overline{RD}$ to Clock ↑ Setup Time	240		115		ns
8	TdC(DO)	Clock ↑ to Data Out Delay		240		220	ns
9	TsDI(C)	Data In to Clock ↑ Setup (Write or $\overline{M1}$ Cycle)	50		50		ns
10	TdRD(DOz)	RD ↑ to Data Out Float Delay		230		110	ns
11	TdIO(DOI)	IORQ ↓ to Data Out Delay (INTA Cycle)		340		160	ns
12	TsM1(C)	$\overline{M1}$ to Clock ↑ Setup Time	210		90		ns
13	TsIEI(IO)	IEI to $\overline{IORQ}$ ↓ Setup Time (INTA Cycle)	200		140		ns
14	TdM1(IEO)	$\overline{M1}$ ↓ to IEO ↓ Delay (interrupt before $\overline{M1}$)		300		190	ns
15	TdIEI(IEOr)	IEI ↑ to IEO ↑ Delay (after ED decode)		150		100	ns
16	TdIEI(IEOf)	IEI ↓ to IEO ↓ Delay		150		100	ns
17	TdC(INT)	Clock ↑ to $\overline{INT}$ ↓ Delay		200		200	ns
18	TdIO (W/RWf)	$\overline{IORQ}$ ↓ or $\overline{CE}$ ↓ to $\overline{W/RDY}$ ↓ Delay Wait Mode		300		210	ns
19	TdC (W/RR)	Clock ↑ to $\overline{W/RDY}$ ↓ Delay (Ready Mode)		120		120	ns
20	TdC (W/RWz)	Clock ↓ to $\overline{W/RDY}$ Float Delay (Wait Mode)		150		130	ns
21	Th	Any unspecified Hold when Setup is specified	0		0		ns

AC ELECTRICAL CHARACTERISTICS (continued)

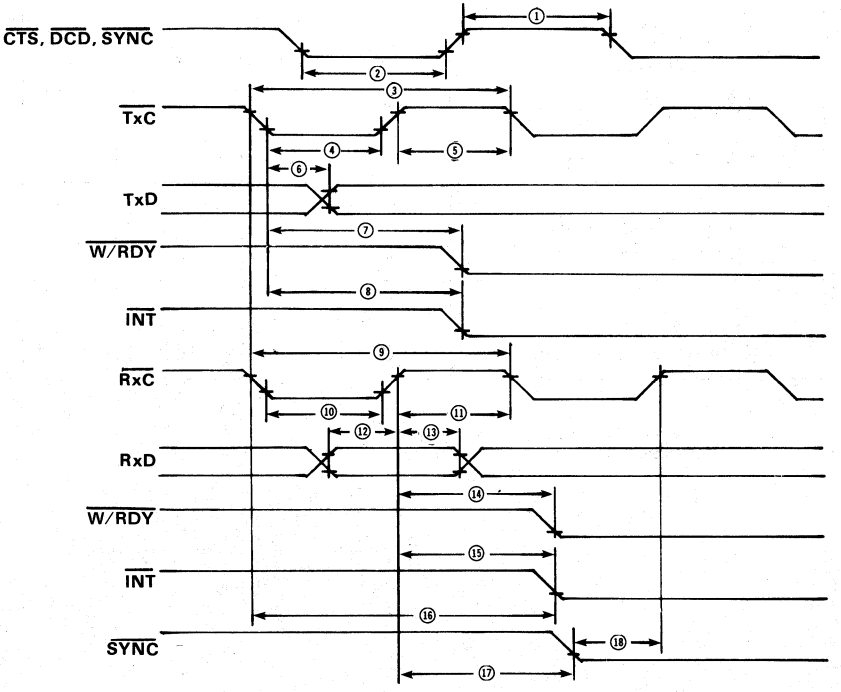
NUMBER	SYM	PARAMETER	MK3884		MK3884-4		UNIT
			MIN	MAX	MIN	MAX	
1	TwPh	Pulse Width (High)	200		200		ns
2	TwPl	Pulse Width (Low)	200		200		ns
3	TcTxC	$\overline{TxC}$ Cycle Time	400	∞	400	∞	ns
4	TwTxCl	$\overline{TxC}$ Width (Low)	180	∞	180	∞	ns
5	TwTxCh	$\overline{TxC}$ Width (High)	180	∞	180	∞	ns
6	TdTxC(TxD)	$\overline{TxC}$ ↓ to TxD Delay (x1 Mode)		400		300	ns
7	TdTxC (W/RRf)	$\overline{TxC}$ ↓ to W/RDY ↓ Delay (Ready Mode)	5	9	5	9	Clk Periods*
8	TdTxC(INT)	$\overline{TxC}$ ↓ to INT ↓ Delay	5	9	5	9	Clk Periods*
9	TcRxC	$\overline{RxC}$ Cycle Time	400	∞	400	∞	ns
10	TwRxCl	$\overline{RxC}$ Width (Low)	180	∞	180	∞	ns
11	TwRxCh	$\overline{RxC}$ Width (High)	180	∞	180	∞	ns
12	TsRxD(RxC)	RxD to $\overline{RxC}$ ↑ Setup Time (x1 Mode)	0		0		ns
13	ThRxD(RxC)	$\overline{RxC}$ ↑ to RxD Hold time (x1 Mode)	140		140		ns
14	TdRxC (W/RRf)	$\overline{RxC}$ ↑ to $\overline{W/RDY}$ ↓ Delay (Ready Mode)	10	13	10	13	Clk Periods*
15	TdRxC(INT)	$\overline{RxC}$ ↑ to $\overline{INT}$ ↓ Delay	10	13	10	13	Clk Periods*
16	TdTxC(INT)	$\overline{TxC}$ ↓ to $\overline{INT}$ ↓ Delay	5	9	5	9	Clk Periods*
17	TdRxC (SYNC)	$\overline{RxC}$ ↑ to $\overline{SYNC}$ ↓ Delay (Output Modes)	4	7	4	7	Clk Periods*
18	TsSYNC (RxC)	$\overline{SYNC}$ ↓ to $\overline{RxC}$ ↑ Setup (External Sync Modes)	-100		-100		ns

In all modes, the System Clock rate must be at least five times the maximum data rate.
RESET must be active a minimum of one complete Clock Cycle.
*System Clock

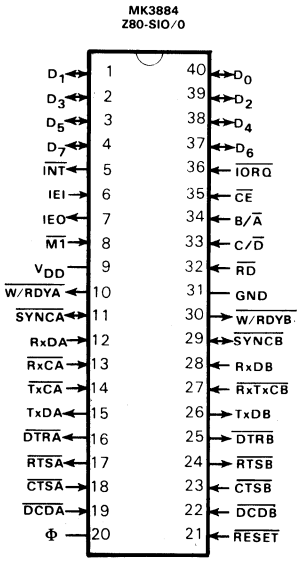
AC ELECTRICAL CHARACTERISTICS
Figure 8.5

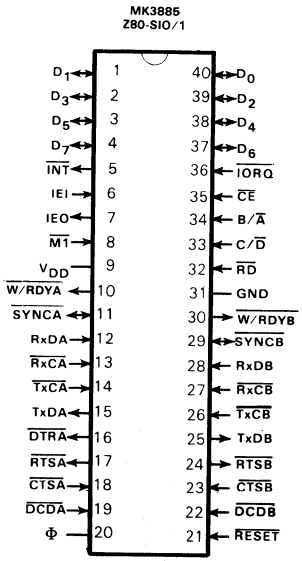


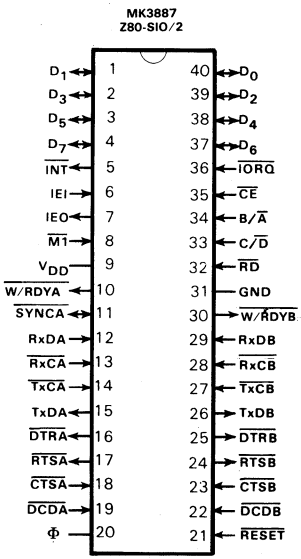
AC ELECTRICAL CHARACTERISTICS
Figure 8.6



PIN ASSIGNMENTS
Figure 8.7







III

10.0 ORDERING INFORMATION

PART NO.	ZILOG EQUIVALENT	PACKAGE TYPE	MAX CLOCK FREQUENCY	TEMPERATURE RANGE
MK3884N	Z80-SIO	Plastic	2.5MHz	0°C to + 70°C
MK3884P	Z80-SIO	Ceramic	2.5MHz	0°C to + 70°C
MK3884N-10	Z80-SIO	Plastic	2.5MHz	-40°C to + 85°C
MK3884P-10	Z80-SIO	Ceramic	2.5MHz	-40°C to + 85°C
MK3884N-4	Z80A-SIO	Plastic	4MHz	0°C to + 70°C
MK3884P-4	Z80A-SIO	Ceramic	4MHz	0°C to + 70°C
MK3885N	Z80-SIO	Plastic	2.5MHz	0°C to + 70°C
MK3885P	Z80-SIO	Ceramic	2.5MHz	0°C to + 70°C
MK3885N-10	Z80-SIO	Plastic	2.5MHz	-40°C to + 85°C
MK3885P-10	Z80-SIO	Ceramic	2.5MHz	-40°C to + 85°C
MK3885N-4	Z80A-SIO	Plastic	4MHz	0°C to + 70°C
MK3885P-4	Z80A-SIO	Ceramic	4MHz	0°C to + 70°C
MK3887N	Z80-SIO	Plastic	2.5MHz	0°C to + 70°C
MK3887P	Z80-SIO	Ceramic	2.5MHz	0°C to + 70°C
MK3887N-10	Z80-SIO	Plastic	2.5MHz	-40°C to + 85°C
MK3887P-10	Z80-SIO	Ceramic	2.5MHz	-40°C to + 85°C
MK3887N-4	Z80-SIO	Plastic	4MHz	0°C to + 70°C
MK3887P-4	Z80-SIO	Ceramic	4MHz	0°C to + 70°C

NOTE: Refer to section on Pin Description for explanation of the differences between the MK3884, MK3885, and MK3887.

SERIAL TIMER INTERRUPT CONTROLLER (STI) MK3801

TABLE OF CONTENTS

1.0 GENERAL DESCRIPTION

1.1 INTRODUCTION

The MK3801 Z80 STI (Serial Timer Interrupt) is a Z80 microprocessor peripheral designed to serve a broad range of applications. By incorporating multiple functions within the Z80 STI, the designer is offered maximum flexibility while keeping the device count to a minimum. The STI integrates four functions within a 40 pin package: Binary Timers, Parallel I/O, Interrupts, and a USART. Given these features, the STI becomes a versatile device which can serve not only a specific design requirement, but a combination thereof. A few examples of these features include:

- Full Duplex Usart with modem controls, DMA Handshake, and baud rate generator
- 8 bit parallel I/O port with timers
- Multifunctional Programmable Timers with Interrupts
- Interrupt Controller

The Interrupt Controller includes 16 prioritized, vectored interrupts which provide maximum speed and efficiency in servicing the various device functions. If interrupts are not desired, each channel may be operated in a polled mode. The STI was designed not only to interface to the Z80 CPU, but also to virtually any microprocessor. Because the STI uses an asynchronous clock, all timing parameters are referenced from the control signals (unlike other Z80 peripherals, which are referenced to the system clock). There is also a special provision for handling interrupts in non-Z80 systems.

III

1.2 FEATURES

Major features of the Z80 STI are:

- Full duplex USART with programmable DMA control signals
- Two binary delay timers
- Two full feature binary timers with:
  * Delay to interrupt mode
  * Pulse width measurement mode
  * Event counter mode
- Eight general purpose lines with:
  * Full bi-directional I/O capability
  * Edge triggered interrupts on either edge
- Full control of each interrupt channel
  * Enable/disable
  * Maskable
  * Automatic end of interrupt mode
  * Software end of interrupt mode

1.3 PIN DESCRIPTION

V_{SS}: Ground
V_{CC}: + 5 volts (± 5%)
$\overline{CE}$: Chip Enable (Input, active low)
$\overline{RD}$: Read Enable (Input, active low)
$\overline{WR}$: Write Enable (Input, active low)
A0-A3: Address Inputs. Used to address one of the internal registers during a read or write operation.
D0-D7: Data Bus (bi-directional)
Used to receive data from or transmit data to one of the internal registers during a read or write operation.
$\overline{RESET}$: Device reset. (Input, active low). When activated, all internal registers (except for timer, USART Data registers, and xmit status register) will be cleared. All timers will be stopped. The USART receiver and transmitter will be turned off. All interrupt channels will be disabled and all pending interrupts will be cleared. The General

DEVICE PINOUT

Figure 1

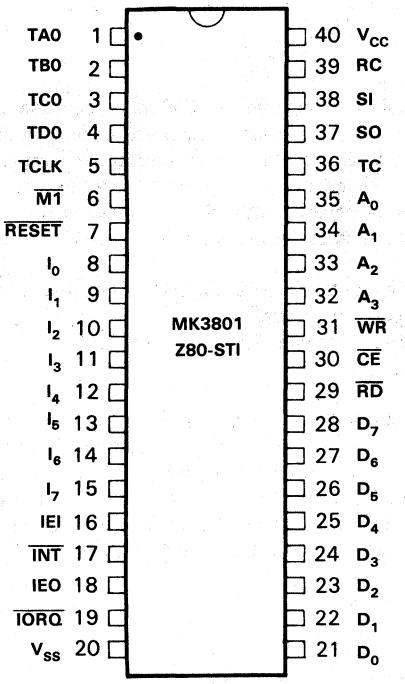


Figure 1 illustrates the pinout for the MK3801. The functions of these individual pins are described below.

Purpose Interrupt / I/O lines will be placed in the tri-state input mode. All timer outputs will be forced to the low (logic "O") state only when TCLK is running.

I_0-I_7: General Purpose Interrupt / I/O lines. These lines may be used as interrupt inputs and/or I/O lines. When used as interrupt inputs, their active edge is programmable. A data direction register is used to define which lines are to be Hi-Z inputs and which lines are to be push-pull outputs.

$\overline{INT}$: Interrupt Request. Used to communicate an interrupt request from the STI to the CPU. /INT is an active low open drain output.

$\overline{IORQ}$: Input/Output Request from Z80-CPU (input, active low). The $\overline{IORQ}$ signal is used in conjunction with $\overline{M1}$ to signal the MK3801 that the CPU is acknowledging its interrupt.

IEI: Interrupt Enable In. (Active high) Used to signal the STI that no higher priority device is requesting interrupt service.

IEO: Interrupt Enable Out. (Active high) Used to signal lower priority peripherals that neither the STI nor another higher priority peripheral is requesting interrupt service.

SO: Serial Output. This is the output of the USART transmitter.

SI: Serial Input. This is the input to the USART receiver.

RC: Receiver Clock. This input controls the serial bit rate of the USART receiver.

TC: Transmitter Clock. This input controls the serial bit rate of the USART transmitter.

TAO-TDO: Timer Outputs. Each of the four timers has an output which can produce a square wave. The output will change states each timer cycle; thus one full period of the timer out signal is equal to two timer cycles.

TCLK: Timer Clock input. All chip accesses are independent of any system clock. Thus only the timers need a frequency reference. That reference can be a system clock or any other clock source.

$\overline{M1}$: Z80 Machine Cycle One (input, active low). Each time this input goes active, interrupt priorities are frozen. If $\overline{IORQ}$ also goes active, and an interrupt is pending, a vector will be passed. Thus the interrupt acknowledge is defined as $\overline{M1}$ $\overline{IORQ}$. $\overline{M1}$ is also used along with $\overline{RD}$ to scan for the ED 4D (RETI) op-code for the automatic end of service feature available with the Z-80.

INTERNAL ORGANIZATION
Figure 2

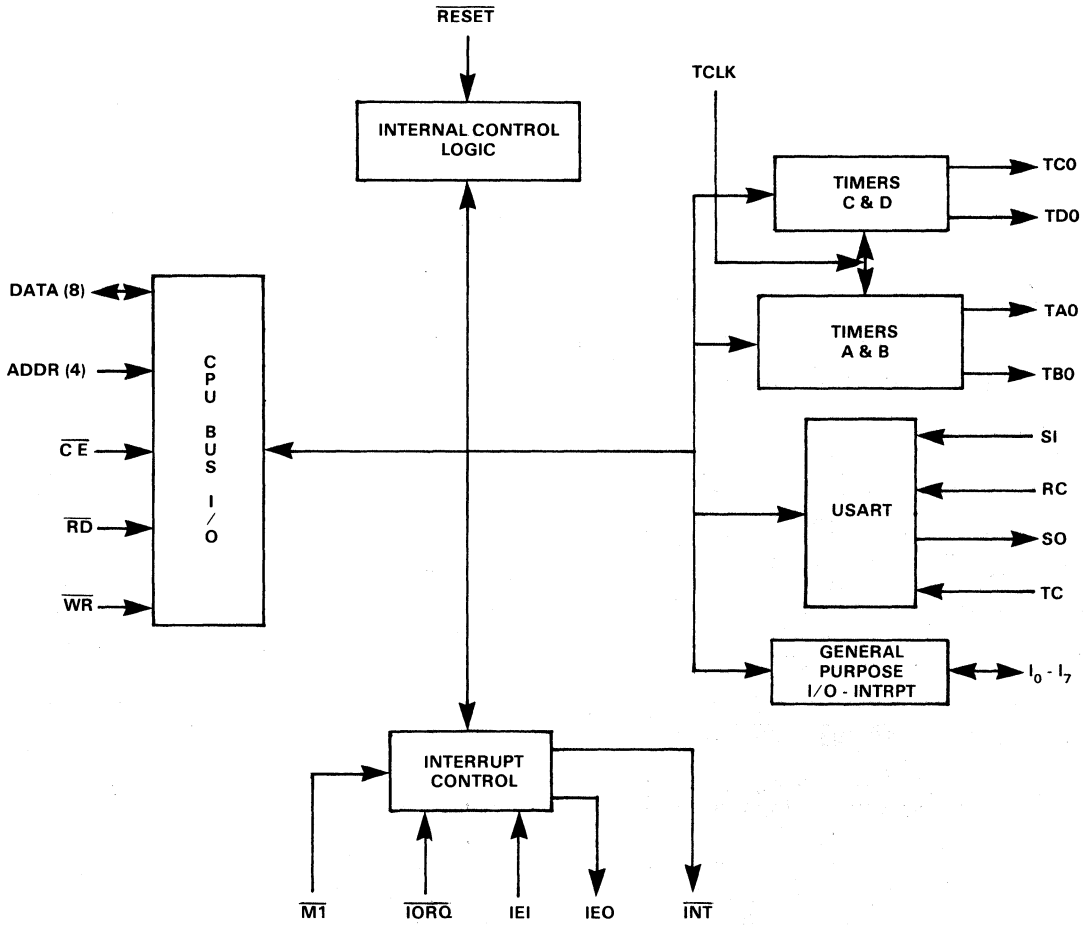


Figure 2 illustrates the MK3801 internal organization, which supports the full set of timing, communications, parallel I/O, and interrupt processing functions available in the device.

1.4 INTERNAL REGISTERS

There are 24 internal registers used to control the operation of the STI. 16 of these registers are directly addressable. Eight registers are indirectly addressable. The first directly addressable register, the Indirect Data Register (IDR) is used in conjunction with the Pointer/Vector register to access the eight indirectly accessible registers. The address of the desired indirectly accessible register must first be loaded into the three lowest order bits of the Pointer/Vector Register. Any subsequent access of the IDR will access the indirect register whose address is contained in the three lowest order bits of the Pointer/Vector Register.

DIRECTLY ACCESSIBLE REGISTERS
Figure 3

ADDRESS	ABBREVIATION	REGISTER NAME
0	IDR	Indirect Data Register
1	GPIP	General Purpose I/O-Interrupt
2	IPRB	Interrupt Pending Register B
3	IPRA	Interrupt Pending Register A
4	ISRB	Interrupt in-Service Register B
5	ISRA	Interrupt in-Service Register A
6	IMRB	Interrupt Mask Register B
7	IMRA	Interrupt Mask Register A
8	PVR	Pointer/Vector Register
9	TABCR	Timers A and B Control Register
A	TBDR	Timer B Data Register
B	TADR	Timer A Data Register
C	UCR	USART Control Register
D	RSR	Receiver Status Register
E	TSR	Transmitter Status Register
F	UDR	USART Data Register

INDIRECTLY ADDRESSABLE REGISTERS
Figure 4

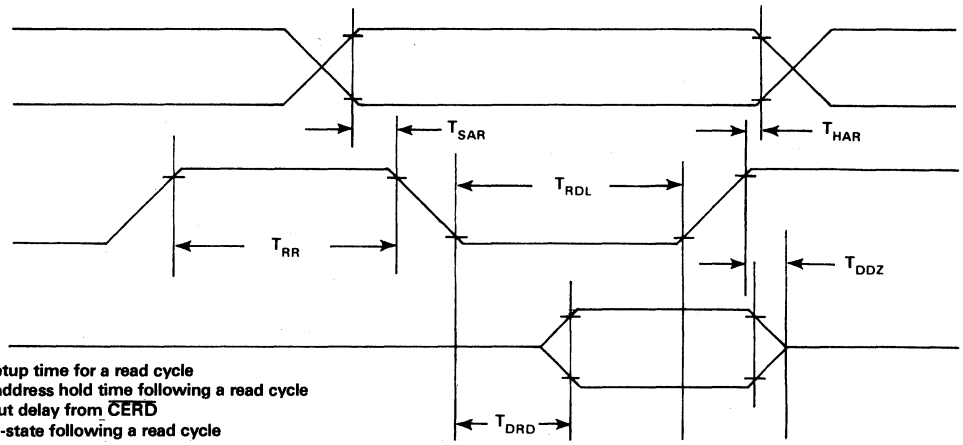
INDIRECT ADDRESS	ABBREVIATION	REGISTER NAME
0	SCR	Sync Character Register
1	TDDR	Timer D Data Register
2	TCDR	Timer C Data Register
3	AER	Active Edge Register
4	IERB	Interrupt Enable Register B
5	IERA	Interrupt Enable Register A
6	DDR	Data Direction Register
7	TCDCR	Timers C and D Control Register

1.5 REGISTER ACCESSES

All register accesses are independent of any system clock. To read a register, both $\overline{CE}$ and $\overline{RD}$ must be active. The internal read control signal is essentially the combination of both $\overline{CE}$ and $\overline{RD}$ active; thus, the read operation will begin when the later of these two signals goes active and will end when the first signal goes inactive. The address bus must be stable prior to the start of the operation and must remain stable until the end of the operation. Unless a read operation, or interrupt acknowledge cycle, is in progress, the data bus (D0-D7) will remain in the tri-state condition.

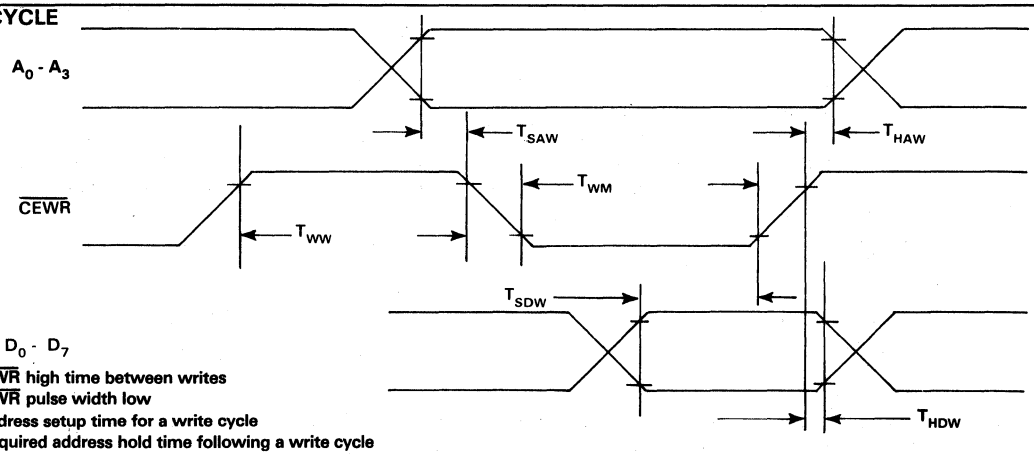
To write a register, both $\overline{CE}$ and $\overline{WR}$ must be active. The address must be stable prior to the start of the operation and must remain stable until the end of the operation. The data must be stable prior to

READ CYCLE
Figure 5



T_{SAR} = Address setup time for a read cycle
T_{HAR} = Required address hold time following a read cycle
T_{DRD} = Data output delay from $\overline{CERD}$
T_{DDZ} = Time to tri-state following a read cycle
T_{RDL} = $\overline{CERD}$ pulse width low (read cycle)
T_{RR} = $\overline{CERD}$ high time between read cycles

WRITE CYCLE
Figure 6



T_{WW} = $\overline{CEWR}$ high time between writes
T_{WM} = $\overline{CEWR}$ pulse width low
T_{SAW} = Address setup time for a write cycle
T_{HAW} = Required address hold time following a write cycle
T_{SDW} = Data setup time prior to the end of a write cycle
T_{HDW} = Required data hold time following a write cycle

the end of the operation and must remain stable until the end of the operation. The data presented on the bus will be latched into the register shortly after either $\overline{WR}$ or $\overline{CS}$ goes inactive.

Note that the control signal $\overline{IORQ}$ is not used internally to enable the device. This requires that $\overline{IORQ}$ be used in decoding $\overline{CE}$. The STI uses $\overline{IORQ}$ for Interrupt Acknowledge only.

1.6 INTERRUPTS

There are sixteen interrupt channels on the STI. Interrupts may be either polled or vectored. Each channel may be individually enabled or disabled by writing a one or a zero in the appropriate bit of IERA or IERB. When disabled, an interrupt channel is completely inactive. Any internal or external action which would normally produce an interrupt on that channel is ignored. Any pending interrupt on that channel will be cleared by disabling that channel. Disabling an interrupt channel has no effect on the corresponding bit in ISRA or ISRB; thus, if the software automatic end of interrupt mode is used and an interrupt is in service on that channel when the channel is disabled, it will remain in service until cleared in the normal manner. IERA and IERB are also readable.

When an interrupt is received on an enable channel, its corresponding bit in the pending register will be set. When that channel is acknowledged it will pass its vector, and the corresponding bit in the pending register will be cleared. IPRA and IPRB are readable; thus by polling IPRA and IPRB, it can be determined whether a channel has a pending interrupt. IPRA and IPRB are writeable and a pending

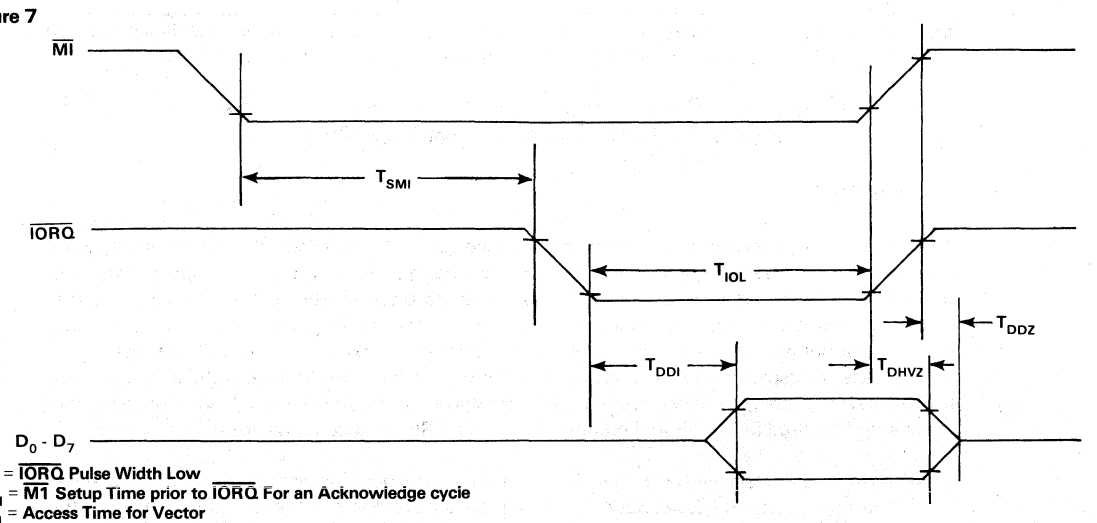
interrupt can be cleared without going through the acknowledge sequence by writing a zero to the appropriate bit. This allows any one bit to be cleared, without altering any other bits, simply by writing all ones except for the bit position to be cleared to IPRA or IPRB. Thus a fully polled interrupt scheme is possible.

The interrupt mask register (IMRA and IMRB) may be used to block a channel temporarily from making an interrupt request. Writing a zero into the corresponding bit of the mask register will still allow the channel to receive an interrupt and latch it into its pending bit (if that channel is enabled), but will prevent that channel from making an interrupt request. If that channel is causing an interrupt request at the time the corresponding bit in the mask register is cleared, the request will cease. If no other channel is making a request, $\overline{INT}$ will go inactive and $\overline{IEO}$ will go high. Note that if $\overline{IEI}$ were low, indicating that a higher priority device were requesting interrupt service, $\overline{INT}$ would already be inactive and $\overline{IEO}$ would remain low. If the mask bit is re-enabled, any pending interrupt is now free to resume its request unless blocked by a higher priority request for service. IMRA and IMRB are also readable.

There are two end of interrupt modes: the automatic end of interrupt mode and the software end of interrupt mode. The mode is selected by writing a one or a zero to the S bit of the Pointer/Vector Register. If the S bit of the PVR is a one, all channels operate in the software end of interrupt mode. If the S bit is a zero, all channels operate in the automatic end of interrupt mode. In the automatic end of interrupt mode, the pending bit is cleared when that channel passes its vector. At that point, no further history of that interrupt remains in the STI. In the software end of interrupt mode, the in-service bit is set and the pending bit is cleared when the channel passes its vector. With the in-service bit set, no lower priority channel is allowed to request an interrupt or to pass its vector during an acknowledge sequence; however, a lower priority channel may still receive an interrupt and latch it into the pending bit. A higher priority channel may still request an interrupt and be acknowledged. The in-service bit of a particular channel may be cleared by writing a zero to the corresponding bit in ISRA or ISRB. Typically, this will be done at the conclusion of the interrupt routine just before the return. Thus no lower priority channel will be allowed to request service until the higher priority channel is complete, while channels of still higher priority will be allowed to request service. The in-service bit can be cleared automatically by an RETI instruction. While the in-service bit is set, a second interrupt on that channel may be received and latched into the pending bit, though no service request will be made in response to the second interrupt until the in-service bit is cleared. ISRA and ISRB may be read at any time. Only a zero may be written into any bit of ISRA and ISRB; thus the in-service bits may be cleared in software but cannot be set in software. This allows any one bit to be cleared, without altering any other bits, simply by writing all ones except for the bit position to be cleared to ISRA or ISRB, as with IPRA and IPRB.

INTERRUPT ACKNOWLEDGE CYCLE

Figure 7



T_{IOL} = $\overline{IORQ}$ Pulse Width Low
T_{SMI} = $\overline{M1}$ Setup Time prior to $\overline{IORQ}$ For an Acknowledge cycle
T_{DDI} = Access Time for Vector
T_{DDZ} = Time to Tri-State Following a Vector
T_{DHVZ} = Data hold time following $\overline{M1}$ $\overline{IORQ}$ during interrupt acknowledge cycle

Each interrupt channel responds with a discrete 8-bit vector when acknowledged. The upper three bits of the vector are set by writing the upper three bits of the PVR. The four next lower order bits (Bit 4-Bit 1) are generated by the interrupting channel, and Bit 0 of the vector is always a zero.

To acknowledge an interrupt, $\overline{M1}$ must first be pulled low. With $\overline{M1}$ low, interrupts will be frozen. $\overline{IORQ}$ must subsequently be pulled low with $\overline{M1}$ remaining low. The vector will now be driven onto the data bus and will remain on the bus as long as both $\overline{M1}$ and $\overline{IORQ}$ remain low, and the bus will go to the tri-state mode shortly after either signal returns to the inactive state.

INTERRUPT CONTROL REGISTER DEFINITIONS

Figure 8

There are sixteen interrupt channels on the STI arranged in the following priority:

PRIORITY	CHANNEL	DESCRIPTION	ALTERNATE USAGE
HIGHEST	1111	General Purpose Interrupt 7(I_7)	
	1110	General Purpose Interrupt 6(I_6)	
	1101	Timer A	
	1100	Receive Buffer Full	
	1011	Receive Error	
	1010	Transmit Buffer Empty	
	1001	Transmit Error	
	1000	Timer B	
	0111	General Purpose Interrupt 5(I_5)	
	0110	General Purpose Interrupt 4(I_4)	TA (PW-Event)
	0101	Timer C	
	0100	Timer D	
	0011	General Purpose Interrupt 3(I_3)	TB (PW-Event)
	0010	General Purpose Interrupt 2(I_2)	
	0001	General Purpose Interrupt 1(I_1)	DMA (TR)TX
LOWEST	0000	General Purpose Interrupt 0(I_0)	DMA (RR)REC

Figure 8 describes the 16 prioritized interrupt channels. As shown, General Purpose Interrupt 7 has the highest priority, while General Purpose Interrupt 0 is assigned the lowest priority. Each of these channels may be reprioritized, in effect, by selectively masking interrupts under software control. The binary numbers under "channel" correspond to the modified bits V4, V3, V2, and V1, respectively, of the Interrupt Vector for each channel.

Each channel has an enable bit contained in IERA or IERB, a pending latch contained in IPRA or IPRB, a mask bit contained in IMRA or IMRB, and an in-service latch contained in ISRA or ISRB. Additionally, the eight General Purpose Interrupts each have an edge bit contained in the Active Edge Register, (AER), a direction bit in the Data Direction Register (DDR), and an I/O port addressable as a bit in the General Purpose Interrupt-I/O Port (GPIP).

The Active Edge Register (AER) allows each of the General Purpose Interrupts to produce an interrupt on either a 1-0 transition or a 0-1 transition. Writing a zero to the appropriate bit of the AER causes the associated input to produce an interrupt on the 1-0 transition, while a 1 causes the interrupt on the 0-1 transition. The edge bit is simply one input to an exclusive-or gate, with the other input coming from the input buffer and the output going to a 1-0 transition detector. Thus, depending upon the state of the input, writing the AER can cause an interrupt-producing transition, which will cause an interrupt on the associated channel if that channel is enabled. One would then normally configure the AER before enabling interrupts via IERA and IERB.

The DDR is used to define I_0-I_7 as inputs or as outputs on a bit by bit basis. Writing a zero into a bit of the DDR causes the corresponding interrupt-I/O pin to be a Hi-Z input. Writing a one into a bit of the DDR causes the corresponding pin(s) to be configured as a push-pull output. When data is written into the GPIP, those pins defined as inputs will remain in the Hi-Z state while those pins defined as outputs will assume the state (high or low) of their corresponding bit in the GPIP. When the GPIP is read, the data read will come directly from the corresponding bit of the GPIP register for all pins defined as output. For the bits defined as inputs, the data will come from the input buffers of the pins.

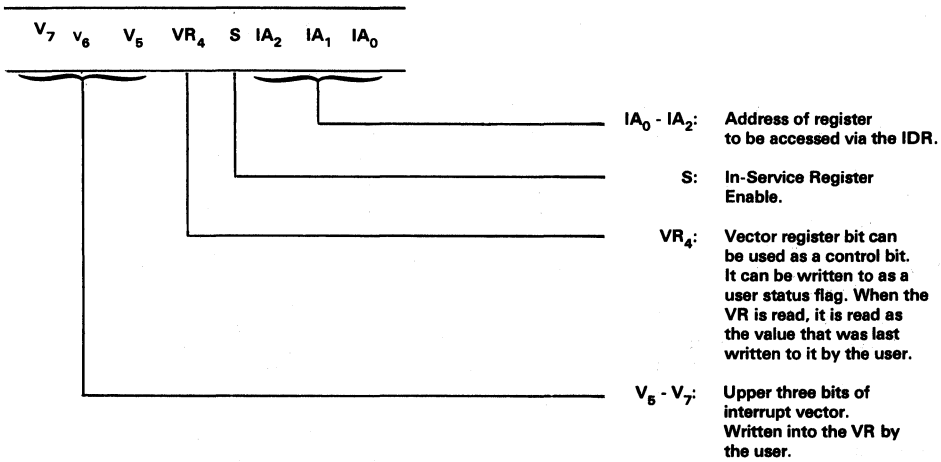
The control bits for each interrupt channel are summarized in the following table:

INTERRUPT CHANNEL CONTROL BITS
Figure 9

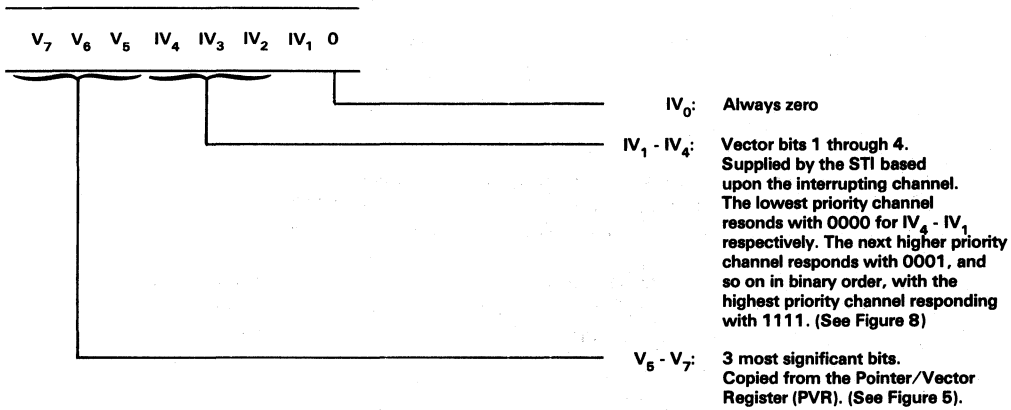
Channel	Enable Bit	Pending Bit	Mask Bit	Service Bit	Active Edge Bit	Data Direction Bit	Port Bit
I_7	$IERA_7$	$IPRA_7$	$IMRA_7$	$ISRA_7$	AER_7	DDR_7	$GPIP_7$
I_6	$IERA_6$	$IPRA_6$	$IMRA_6$	$ISRA_6$	AER_6	DDR_6	$GPIP_6$
Timer A	$IERA_5$	$IPRA_5$	$IMRA_5$	$ISRA_5$			
Receive Buffer Full	$IERA_4$	$IPRA_4$	$IMRA_4$	$ISRA_4$			
Receive Error	$IERA_3$	$IPRA_3$	$IMRA_3$	$ISRA_3$			
Transmit Buffer Empty	$IERA_2$	$IPRA_2$	$IMRA_2$	$ISRA_2$			
Transmit Error	$IERA_1$	$IPRA_1$	$IMRA_1$	$ISRA_1$			
Timer B	$IERA_0$	$IPRA_0$	$IMRA_0$	$ISRA_0$			
I_5	$IERB_7$	$IPRB_7$	$IMRB_7$	$ISRB_7$	AER_5	DDR_5	$GPIP_5$
I_4	$IERB_6$	$IPRB_6$	$IMRB_6$	$ISRB_6$	AER_4	DDR_4	$GPIP_4$
Timer C	$IERB_5$	$IPRB_5$	$IMRB_5$	$ISRB_5$			
Timer D	$IERB_4$	$IPRB_4$	$IMRB_4$	$ISRB_4$			
I_3	$IERB_3$	$IPRB_3$	$IMRB_3$	$ISRB_3$	AER_3	DDR_3	$GPIP_3$
I_2	$IERB_2$	$IPRB_2$	$IMRB_2$	$ISRB_2$	AER_2	DDR_2	$GPIP_2$
I_1	$IERB_1$	$IPRB_1$	$IMRB_1$	$ISRB_1$	AER_1	DDR_1	$GPIP_1$
I_0	$IERB_0$	$IPRB_0$	$IMRB_0$	$ISRB_0$	AER_0	DDR_0	$GPIP_0$

The complete definition of the PVR and of the interrupt vector is summarized below:

POINTER/VECTOR REGISTER (PVR) PORT 08
Figure 10



INTERRUPT VECTOR
Figure 11



1.7 USART

The USART is a full duplex double buffered unit. The USART Data Register (UDR) is used to access both the receive buffer and the transmit buffer. When data is written to the UDR, it is latched into the transmit buffer. When the UDR is read, the data comes from the receive buffer.

There is a USART Control Register (UCR) used to configure certain properties of both the transmitter and the receiver.

USART CONTROL REGISTER (UCR) Port C
Figure 12

UCR_7							UCR_0
1 = ÷ 16 0 = ÷ 1	WL_1	WL_0	ST_1	ST_0	PARITY ENABLED ON 1	1 = EVEN 0 = ODD	DMA CONTROL ENABLE

1 = ÷ 16: 0 = ÷ 1 When this bit is zero, data will be clocked into and out of the receiver and transmitter at the frequency of their respective clocks. When this bit is loaded with a one, data will be clocked into and out of the receiver and transmitter at one sixteenth the frequency of their respective clocks. Additionally, when placed in the divide by sixteen mode, the receiver data transition resynchronization logic will be enabled.

WLO-WLI: Word Length Control. These two bits set the length of the data word (exclusive of start bits, stop bits,and parity bits) as follows:

WL1	WLO	Word Length
0	0	8 bits
0	1	7 bits
1	0	6 bits
1	1	5 bits

STO-ST1: Start/Stop bit control (format control). These two bits set the format as follows:

ST1	STO	Start Bits	Stop Bits	Format
0	0	0	0	SYNC
0	1	1	1	ASYNC
*1	0	1	1 ½	ASYNC
1	1	1	2	ASYNC

*NOTE ÷ 16 only

P: Parity Enabled. When set ("1"), parity will be checked by the receiver, parity will be calculated, and a parity bit will be inserted by the transmitter. When cleared ("0"), no parity check will be made and no parity bit will be inserted for transmission.

The sync character length is the word length plus one when parity is enabled. The extra bit in the sync character is transmitted as the parity bit. However, with a word length of eight, when parity is selected, the parity bit for the sync character is computed and added on by the STI.

E/O: Even-Odd. When set ("1"), even parity will be used if parity is enabled. When cleared ("0"), odd parity will be used if parity is enabled.

DMA: When the bit is set to a one, GPI pin 0 and GPI pin 1 are programmed to be outputs. Pin 0 reflects the status of the receiver buffer full flag. Pin 1 reflects the status of the transmitter buffer empty flag.

Note that the synchronous or asynchronous format may be selected independently of a ÷ 1 or ÷ 16 clock. Thus it is possible to clock data synchronously into the device but still use start and stop bits. In this mode, all normal asynchronous format features still apply. Data will be shifted in after a start bit is encountered, and a stop bit will be checked to determine proper framing. If a transmit underrun condition occurs, the output will be placed in a marking state, etc. It is conversely possible to clock data in asynchronously using a synchronous format. There is data transition detection logic built into the receive clock circuitry which will re-synchronize the internal shift clock on each data transition so that, with sufficiently frequent data transitions, start bits are not required. In this mode, all other common synchronous features function normally.This re-synchronization logic is only active in ÷16 clock mode.

1.7.1 RECEIVER

The receiver section of the USART is configured by the UCR as previously described. The status of the receiver can be determined by reading and writing to the Receiver Status Register (RSR). The RSR is configured as follows:

RECEIVER STATUS REGISTER (RSR) PORT D
Figure 13

RSR_7							RSR_0
BUFFER FULL	OVERRUN ERROR	PARITY ERROR	FRAME ERROR	FOUND/$\overline{\text{SEARCH}}$ OR BREAK DETECT	MATCH/CHARACTER IN PROGRESS	SYNC STRIP ENABLE	RECEIVER ENABLE

III

BF: Buffer Full. This bit is set when the incoming word is transferred to the receive buffer. The bit is cleared when the receive buffer is read by reading the UDR. This bit of the RSR is read only.

OE: Overrun Error. This flag is set if the incoming word is completely received and due to be transferred to the receive buffer, but the last word in the receive buffer has not yet been read. When this condition occurs, the word in the receive buffer is not overwritten by the new word. Note that the status flags always reflect the status of the data word currently in the receive buffer. As such, the OE flag is not actually set until the good word currently in the buffer has been read. The interrupt associated with this error will also not be generated until the old word in the receive buffer has been read.

OE flag is cleared by reading the receiver status register, and new data words cannot be shifted to the receive buffer until this is done.

PE: Parity Error. This flag is set if the word received has a parity error. The flag is set when the received word is transferred from the shift register to the receive buffer if the error condition exists. The flag is cleared when the next word which does not have a parity error is transferred to the receive buffer.

FE: Frame Error. This flag only applies to the asynchronous format. A frame error is defined as a non-zero data word which is not followed by a stop bit. Like the PE flag, the FE flag is set or cleared when a word is transferred to the receive buffer.

F/$\overline{\text{S}}$: Found/$\overline{\text{Search}}$. This combination control bit and flag bit is only used with the synchronous format. It can be set or cleared by writing to this bit of the RSR. When this bit is cleared, the receiver is placed in the search mode. In this mode, a bit by bit comparison of the incoming data to the character in the Sync Character Register (SCR) is made. The word length counter is disabled. When a match is found, this bit will be set automatically, and the word length counter will start as sync has now been achieved. An interrupt will be generated on the receive error channel when the match occurs. The word just shifted in will, of necessity, be equal to the sync character, and it will not be transferred to the receive buffer.

B: Break. This flag is used only when the asynchronous format is selected. This flag will be set when an all zero data word, followed by no stop bit, is received. The flag will stay set until both a non-zero bit is received and the RSR has been read at least once since the flag was set. Break indication will not occur if the receive buffer is full.

M/CIP: Match/Character in Progress. If the synchronous format is selected, this flag is the Match flag. It will be set each time the word transferred to the receive buffer matches the sync character. It will be reset each time the word transferred to the receive buffer does not match the sync character. If the asynchronous format is selected, this flag represents Character in Progress. It will be set upon a start bit detect and cleared at the end of the word.

SS: Sync Strip Enable. If this bit is set to a one, data words that match the sync character will not be loaded into the receive buffer, and no buffer full or match signal will be generated.

RE: Receiver Enable. This control bit is used to enable or disable the receiver. If a zero is written to this bit of the RSR, the receiver will turn off immediately. All flags including the F/$\overline{S}$ bit will be cleared. If a one is written to this bit, normal receiver operation is enabled. The receive clock has to be running before the receiver is enabled.

There are two interrupt channels associated with the receiver. One channel is used for the normal Buffer Full condition, while the other channel is used whenever an error condition occurs. Only one interrupt is generated per word received, but dedicating two channels allows separate vectors: one for the normal condition, and one for an error condition. If the error channel is disabled, an interrupt will be generated via the Buffer Full Channel, whether the word received is normal or in error. Those conditions which produce an interrupt via the error channel are: Overrun, Parity Error, Frame Error, Sync Found, and Break. If a received word has an error associated with it, and the error interrupt channel is enabled, an interrupt will occur on the error channel only.

Each time a word is transferred into the receive buffer, a corresponding set of flags is latched into the RSR. No flags (except CIP) are allowed to change until the data word has been read from the receive buffer. Reading the receive buffer allows a new data word to be transferred to the receive buffer when it is received. Thus one should first read the RSR then read the receive buffer (UDR) to ensure that the flags just read match the data word just read. If done in the reverse order, it is possible that subsequent to reading the data word from the receive buffer, but prior to reading the RSR, a new word may be received and transferred to the receive buffer and, with it, its associated flags latched into the RSR. Thus, when the RSR is read, those flags may actually correspond to a different data word. It is good practice, also, to read the RSR prior to a data read as, when an overrun error occurs, the receiver will not assemble new characters until the RSR has been read.

As previously stated, when overrun occurs, the OE flag will not be set and the associated interrupt will not be generated until the receive buffer has been read. If a break occurs, and the receive buffer has not yet been read, only the B flag will be set (OE will not be set). Again, this flag will not be set until the last valid word has been read from the receive buffer. If the break condition ends and another whole data word is received before the receive buffer is read, both the B and OE flags will be set once the receive buffer is read.

If a break occurs while the OE flag is set, the B flag will also be set.

A break generates an interrupt when the condition occurs and again when the condition ends. If the break condition ends before it is acknowledged by reading the RSR, the end of break interrupt will be generated once the RSR is read.

Anytime the asynchronous format is selected, start bit detection is enabled. New data is not shifted into the shift register until a zero bit is detected. If a ÷16 clock is selected, along with the asynchronous format, false start bit detection is also enabled. Any transition has to be stable for 3 positive going edges of the receive clock to be called a valid transition. For a start bit to be good, a valid 0-1 transition must not occur for 8 positive clock transitions after the initial valid 1-0 transition.

After a good start bit has been detected, valid transitions in the data are checked for continuously. When a valid transition is detected, the counter is forced to state zero, and no more transition checking is started until state four. At state eight, the "previous state" of the transition checking logic is clocked into the receiver.

As a result of this resynchronization logic, it is possible to run with asynchronous clocks without start and stop bits if there are sufficient valid transitions in the data stream. This logic also makes the unit more tolerant of clock skew for normal asynchronous communications than a device which employs only start bit synchronization.

1.7.2 TRANSMITTER

The transmitter section of the USART is configured as to format, word length, etc. by the UCR, as previously described. The status of the transmitter can be determined by reading or writing the Transmitter Status Register (TSR). The TSR is configured as follows:

III

TRANSMITTER STATUS REGISTER (TSR) PORT E
Figure 14

TSR_7							TSR_0
BUFFER EMPTY	UNDERRUN ERROR	AUTO TURNAROUND	END OF TRANSMISSION	BREAK	HIGH	LOW	TRANSMITTER ENABLE

BE: Buffer Empty. This status bit is set when the word in the transmit buffer is transferred to the output shift register and thus the transmit buffer may be reloaded with the following word. The flag is cleared when the transmit buffer is reloaded. The transmit buffer is loaded by writing to the UDR.

UE: This bit is set when the last word has been shifted out of the transmit shift register before a new word has been loaded into the transmit buffer. The bit is cleared by reading the TSR or by disabling the transmitter. It is not necessary to clear this bit before loading the UDR.

AT: This bit causes the receiver to be enabled at the end of the transmission of the last word in the transmitter. The user must turn off the transmitter before the end of the last word.

END: End of transmission. When the transmitter is turned off with a character still in the output shift register, transmission will continue until that character is shifted out. Once it has cleared the output register, the END bit will be set. If no character is being transmitted when the transmitter is disabled, the transmitter will stop at the next rising edge of the shift clock, and END will immediately be set. The END bit is cleared by re-enabling the transmitter.

B: Break. This control bit will cause a break to be transmitted. When a "1" is written to the B bit of the TSR, a break will be transmitted upon completion of the character (if any) currently being transmitted. A break will continue to be transmitted until the B bit is cleared by writing a "0" to this bit of the TSR. At that time, normal transmission will resume. The B bit has no function in the synchronous format. Setting the "B" bit to a one keeps the "BE" bit from being set to a one. So, if there were a word in the buffer at the start of break, it would remain there until the end of break, at which time it would be transmitted (if the transmitter is still enabled). If the buffer were not full at the start of break, it could be written at any time during the break. If the buffer is empty at the end of break, the underrun flag will be set (unless the transmitter is disabled).

H,L: High and Low. These two control bits are used to configure the transmitter output, when the transmitter is disabled, as follows:

H	L	Output State
0	0	Hi-Z
0	1	Low ("0")
1	0	High
1	1	Loop - Connects transmitter output to receiver input, and TC to Receiver Clock (RC and SI are not used; they are bypassed internally). In loop back mode, transmitter output goes high when disabled.

Altering these two bits after XE is low will alter the output state.

XE: Transmitter Enable. This control bit is used to enable or disable the transmitter. When set, the transmitter is enabled. When cleared, the transmitter will be disabled. If disabled, any word currently in the output register will continue to be transmitted until finished. If a break is being transmitted when XE is cleared, the transmitter will turn off at the end of the break character, and no end of break stop bit is transmitted. The transmit clock must be running before the transmitter is enabled. A "one" bit always precedes the first word out of the transmitter after the transmitter is enabled. There is a delay between the time the transmitter enable bit is written and when the transmitter reset goes low; therefore, the H & L bits should be written with the desired state when the transmitter enable bit is written high.

The transmit buffer can be loaded prior to enabling the transmitter. When the transmitter is disabled, any character currently in the process of being transmitted will continue to conclusion, but any character in the transmit buffer will not be transmitted and will remain in the buffer. Thus no buffer empty interrupt will occur nor will the BE flag be set. If the buffer were already empty, the BE flag would be set and would remain set. When the transmitter is disabled with a character in the output register but with no character in the transmit buffer, an Underrun Error will not occur when the character in progress concludes.

Like the receiver section, there are two separate interrupt channels associated with the transmitter. The Buffer Empty condition causes an interrupt via one channel, while the Underrun and END conditions will cause an interrupt via the second channel. When underrun occurs in the synchronous format, the character in the SCR will be transmitted until a new word is loaded into the transmit buffer. In the asynchronous format, a "Mark" will be continuously transmitted when underrun occurs.

Often it is necessary to send a break for some particular period. To aid in timing a break transmission, an END interrupt will be generated at every normal character boundary time during a break transmission.

If the synchronous format is selected, the sync character should be loaded into the Sync Character Register (SCR). This character is compared to the received serial data during a Search, and will be continuously transmitted during an underrun condition.

All flags in the RSR or TSR will continue to function as described whether their associated interrupt channel is disabled or enabled. All interrupt channels are edge triggered and, in many cases, it is the actual output of a flag bit or flag bits which is coupled to the interrupt channel. Thus, if a normal interrupt producing condition occurs while the interrupt channel is disabled, no interrupt would be produced even if the channel was subsequently enabled, because a transition did not occur while the interrupt channel was enabled. That particular

flag bit would have to occur a second time before another "edge" was produced, causing an interrupt to be generated.

Error conditions in the USART are determined by monitoring the Receive Status Register (Port D) and the Transmitter Status Register (Port E). These error conditions are only valid for each word boundary and are not latched. When executing block transfers of data, it is necessary to save any errors so that they can be checked at the end of a block. In order to save error conditions during data transfer, the STI interrupt controller may be used by enabling error interrupts (Port 5, Indirect) for the desired channel (Receive error or Transmit error) and by masking these bits off (Port 7). Once the transfer is complete, the Interrupt Pending Register (Port 3) can be polled to determine the presence of a pending error interrupt, and therefore an error.

Unused bits in the sync character register are zeroed out; therefore, word length should be set up prior to writing the sync word in some cases. Sync word length is the word length plus one when parity is enabled. The user has to determine the parity of the sync word when the word length is not 8 bits. The STI does not add a parity bit to the sync word if the word length is less than 8 bits. The extra bit in the sync word is transmitted as the parity bit. With a word length of eight, and parity selected, the parity bit for the sync word is computed and added on by the STI.

1.8 TIMERS

There are four timers on the STI. Two of the timers (Timer A and Timer B) are full function timers which can perform the basic delay function and can also perform event counting and pulse width measurement. The other two timers (Timer C and Timer D) are delay timers only. One or both of these timers can be used to supply the baud rate clocks for the USART. Each timer has a prescaler which divides the timer clock down before entering the main timer unit.

With the timer stopped, no counting can occur. The timer contents will remain unaltered while the timer is stopped (unless reloaded by writing the Timer Data Register), but any residual count in the prescaler will be lost.

In the delay mode, the prescaler is always active. A count pulse will be applied to the main timer unit each time the prescribed number of timer clock cycles has elapsed. Thus, if the prescaler is programmed to divide by ten, a count pulse will be applied to the main counter every ten cycles of the timer clock.

The counters are initially loaded by writing to the Timer Data Register. Each count pulse will cause the current count to decrement. When the timer has decremented down to "01", the next count pulse will not cause it to decrement to "00". Instead, the next count pulse will cause the timer to be reloaded from its Timer Data Register. Additionally, a "Time Out" pulse will be produced. This Time Out pulse is coupled to the timer interrupt channel, and, if that channel is enabled, an interrupt will be produced. The Time Out pulse is also coupled to the timer output pin and will cause the pin to change states. The output will remain in this new state until the next Time Out pulse occurs. Thus the output will complete one full cycle for each two Time Out pulses.

If, for example, the prescaler were programmed to divide by ten, and the Timer Data Register were loaded with 100 (decimal), the main counter would decrement once for every ten cycles of the timer clock. A Time Out pulse will occur (hence an interrupt if that channel is enabled) every 1000 cycles of the timer clock, and the timer output will complete one full cycle every 2000 cycles of the timer clock.

The counters are 8-bit binary down counters. They may be read at any time by reading their Timer Data Register. The information read is the information last clocked into the timer read register when the $\overline{RD}$ pin had last gone low prior to the current read cycle. When written, data is loaded into the Timer Data Register, and the counter, if the timer is stopped. If the Timer Data Register is written while the timer is running, the new word is not loaded into the timer until it counts through H"01". However, if the timer is written while it is counting through H"01", erroneous data will probably be

written into the timer. This may be circumvented by ensuring that the data register is not written to when the count is H"01".

If the main counter is loaded with "01", a Time Out Pulse will occur every time the prescaler presents a count pulse to the main counter. If loaded with "00", a Time Out pulse will occur after every 256 count pulses.

Changing the prescale value with the timer running can cause the first Time Out pulse to occur at an indeterminate time (no less than one nor more than 200 timer clock cycles), but subsequent Time Out pulses will then occur at the correct interval.

In addition to the delay mode described above, Timers A and B can also function in the pulse width measurement mode or in the event count mode. In either of these two modes, an auxiliary control signal is required. The auxiliary control input for Timer A is I_4 and, for Timer B, I_3 is used. The interrupt channel associated with each input is still fully functional.

The pulse width measurement mode functions much like the delay mode. However, in this mode, the auxiliary control signal acts as an enable to the timer. When the control signal is inactive, the timer will be stopped. When it is active, the prescaler and counter are allowed to run. Thus the width of the active pulse is determined by the number of timer counts which occur while the pulse allows the timer to run. The active state of the control signal is dependent upon that pin's edge bit. If the edge bit associated with the input is a one, it will be active high; thus the timer will be allowed to run when the input is at a high level. If the edge bit is a zero, the input will be active low. As previously stated, the interrupt input associated with the input still functions when the timer is used in the pulse width mode. However, if the timer is programmed for the pulse width measurement mode, the interrupt caused by transitions on the associated input will occur on the opposite transition. For example, if the edge bit associated with the input were a one, an interrupt would normally be generated on the 0-1 transition. If the timer associated with the input is placed in the pulse width measurement mode, the interrupt will occur on the 1-0 transition instead. Because the edge bit is a one, the timer will be allowed to count while the input is high. When the input makes the high to low transition, the timer will stop, and it is at this point that the interrupt will occur (assuming that the channel is enabled). This allows the interrupt to signal the CPU that the pulse being measured has terminated; thus the timer may now be read to determine the pulse width. If the associated timer is re-programmed for another mode, interrupts will again occur on the transition, as normally defined by the edge bit. Note that, like changing the edgebit, placing the timer into or taking it out of the pulse width mode can produce a transition on the signal to the interrupt channel and may cause an interrupt. If measuring consecutive pulses, it is obvious that one must read the contents of the timer and then reinitialize the counter by writing to the timer data register. If the timer data register is written while the pulse is going to the active state, the write operaton will probably result in erroneous data being written into the counter. If the timer is written after the pulse goes active, the timer counts from the previous contents, and when it counts through H"01", the correct value is written into the timer. The pulse width then includes counts from before the timer was reloaded.

In the event count mode, the prescaler is disabled. Each time the control input makes an active transition as defined by the edge bit, a count pulse will be generated, and the main counter will decrement. In all other respects, the timer functions as previously described. Altering the edge bit while the timer is in the event count mode can produce a count pulse. The interrupt channel associated with the input is allowed to function normally but would not normally be enabled as the timer is automatically counting transitions on the input. If the channel were enabled to interrupt, an interrupt would be produced on each transition, and the number of transitions could be counted in software by incrementing a register or word in memory during the interrupt routine without requiring the use of the timer. To count transitions reliably, the input must remain in each state (1 and 0) for a length of time equal to 4 periods of the timer clock; thus signals of a frequency up to one fourth that of the timer clock can be counted.

The manner in which the timer output pins toggle states has previously been described. All timer outputs will be forced low by a device RESET. The output associated with Timers A and B will toggle on each Time Out pulse regardless of the mode the timers are programmed to. In addition, the outputs from Timers A and B can be forced low at any time by writing a "1" to the AR and BR bits respectively, in the TCDCR. The output will be forced low only during the WRITE operation, and at the

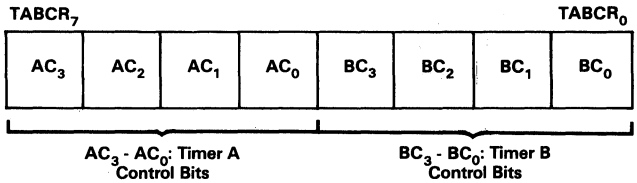
conclusion of the operation, the output will again be free to toggle each time a Time Out pulse occurs.

During reset, the Timer Data Registers and the main counters are not reset. Also, if using the reset option on Timers A or B, one must make sure to keep the other bits in the correct state so as not to affect the operation of Timers C and D.

The Timer A and B Control Register (TABCR) is defined as follows:

TIMER A AND B CONTROL REGISTER (TABCR) Port 9

Figure 15



The four control bits are used to select the timer mode and prescale value as shown below:

CONTROL BIT DEFINITION

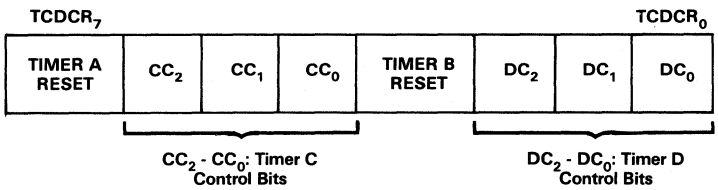
Figure 16

C_3	C_2	C_1	C_0	
0	0	0	0	Timer Stopped
0	0	0	1	Delay Mode, ÷4 Prescale
0	0	1	0	Delay Mode, ÷10 Prescale
0	0	1	1	Delay mode, ÷16 Prescale
0	1	0	0	Delay Mode, ÷50 Prescale
0	1	0	1	Delay Mode, ÷64 Prescale
0	1	1	0	Delay Mode, ÷100 Prescale
0	1	1	1	Delay Mode, ÷200 Prescale
1	0	0	0	Event Count Mode
1	0	0	1	Pulse Width Mode, ÷4 Prescale
1	0	1	0	Pulse Width Mode, ÷10 Prescale
1	0	1	1	Pulse Width Mode, ÷16 Prescale
1	1	0	0	Pulse Width Mode, ÷50 Prescale
1	1	0	1	Pulse Width Mode, ÷64 Prescale
1	1	1	0	Pulse Width Mode, ÷100 Prescale
1	1	1	1	Pulse Width Mode, ÷200 Prescale

The Timer C and D Control Register (TCDCR) is defined as follows:

TIMER C AND D CONTROL REGISTER (TCDCR) Indirect Port 7

Figure 17



Three control bits are used to control each timer, as defined below:

CONTROL BIT DEFINITION
Figure 18

C_2	C_1	C_0	
0	0	0	Timer Stopped
0	0	1	Delay Mode, ÷4 Prescale
0	1	0	Delay Mode, ÷10 Prescale
0	1	1	Delay Mode, ÷16 Prescale
1	0	0	Delay Mode, ÷50 Prescale
1	0	1	Delay Mode, ÷64 Prescale
1	1	0	Delay Mode, ÷100 Prescale
1	1	1	Delay Mode, ÷200 Prescale

MK3801 ELECTRICAL SPECIFICATIONS - PRELIMINARY

ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias ..... -25°C to + 100°C
Storage Temperature ..... -65°C to + 150°C
Voltage on Any Pin with Respect to Ground ..... -.3 V to + 7 V
Power Dissipation ..... 1.5 W

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5 V ± 5% unless otherwise specified.

SYM	PARAMETER	MIN	MAX	UNIT	TEST CONDITION
V_{IH}	Input High Voltage	2.0	V_{CC} + .3	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -120 μA
V_{OL}	Output Low voltage		0.4	V	I_{OL} = 2.0 mA
I_{LL}	Power Supply Current		180	mA	Outputs Open
I_{LI}	Input Leakage Current		±10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float		10	μA	V_{OUT}=2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float		-10	μA	V_{OUT} = 0.4 V

All voltages are referenced to ground.

CAPACITANCE

T_A = 25°C, f = 1 MHz unmeasured pins returned to ground.

SYM	PARAMETER	MAX	UNIT	TEST CONDITION
C_{IN}	Input Capacitance	10	pf	Unmeasured pins returned to ground
C_{OUT}	Tri-state Output Capacitance	10	pf	

A.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = +5 V ± 5% unless otherwise noted.

SIGNAL	SYMBOL	PARAMETER	MK3801-0 MIN	MK3801-0 MAX	MK3801-4 MIN	MK3801-4 MAX	MK3801-6 MIN	MK3801-6 MAX	UNIT	CONDITION
A_0-A_3	T_{SAR} & T_{SAW}	Address setup time prior to falling edge of $\overline{CEWR}$ or $\overline{CERD}$	80		30		15		ns	
	T_{HAR} & T_{HAW}	Address hold time after rising edge of $\overline{CEWR}$ or $\overline{CERD}$	0		0		0		ns	
$\overline{CEWR}$	T_{WL}	$\overline{CEWR}$ pulse width low (write cycle)	360		205		175		ns	Note 1
	T_{WW}	$\overline{CEWR}$ high time between write cycles	580		400		300		ns	
	T_{WRD}	$\overline{CEWR}$ high to $\overline{CERD}$ low	580		400		300		ns	
$\overline{CERD}$	T_{RDL}	$\overline{CERD}$ pulse width low (read cycle)	400		250		215		ns	Note 1
	T_{RR}	$\overline{CERD}$ high time between read cycles	300		200		190		ns	
	T_{M1RD}	Rising $\overline{M1RD}$ to falling $\overline{M1RD}$	225		165		95		ns	
	T_{RDW}	$\overline{CERD}$ high to $\overline{CEWR}$ low	125		100		75			
$\overline{M1}$	T_{SM1}	$\overline{M1}$ setup time prior to falling $\overline{IORQ}$ during interrupt acknowledge	800		500		350		ns	
$\overline{IORQ}$	T_{IOL}	$\overline{IORQ}$ low time	300		185		170		ns	
IEI	T_{SIEI}	Setup to falling $\overline{IORQ}$ during interrupt acknowledge	140		80		65		ns	
	T_{SRD}	Setup prior to end of 4D read on RETI	100		50		40		ns	
D_0-D_7	T_{SDM1}	Data valid prior to rising $\overline{RD}$ ($\overline{M1}$ cycle)	65		50		45		ns	Load 100 pf + 1 TTL load
	T_{HDM1}	Data hold time after rising $\overline{RD}$ ($\overline{M1}$ cycle)	0		0		0		ns	
	T_{DRD}	Data output delay from $\overline{CERD}$		400		250		215	ns	
	T_{SDW}	Data setup time to rising edge of $\overline{CEWR}$	350		280		175		ns	
	T_{HDW}	Data hold time from rising edge of $\overline{CEWR}$	0		0		0		ns	
	T_{DDI}	Data output delay from falling $\overline{IORQ}$ during interrupt acknowledge		300		185		170	ns	

A.C. CHARACTERISTICS (Continued)

SIGNAL	SYMBOL	PARAMETER	MK3801-0 MIN	MK3801-0 MAX	MK3801-4 MIN	MK3801-4 MAX	MK3801-6 MIN	MK3801-6 MAX	UNIT	CONDITION
	T_{DHVZ}	Data hold time following $\overline{M1}$ $\overline{IORQ}$ during interrupt acknowledge cycle.	0		0		0		ns	
	T_{DDZ}	Delay to float		150		100		80	ns	
I_0-I_7	T_{IPW}	Minimum active pulse width	200		100		90		ns	
	T_{ICY}	Minimum time between active edges	200		100		90		ns	
	T_{DIW}	Data valid from rising $\overline{CEWR}$		600		500		400	ns	Load 100 pf + 1 TTL
RR	T_{DRR}	Delay from rising RC		360		240		195	ns	1 TTL
TR	T_{DTR}	Delay from rising TC		450		295		240	ns	
TA0-TD0	T_{DTW}	Timer output low from rising edge of $\overline{CEWR}$ (A & B) (Reset T_{OUT})		600		500		400	ns	Load 100 pf +
	T_{DTI}	T_{OUT} valid from Internal timeout		2 t_{CLK} +400		2 t_{CLK} +300		2 t_{CLK} +250	ns	1 TTL load
TCLK	T_{tCLKL}	Low time	130		95		75		ns	
	T_{tCLKH}	High time	130		95		75		ns	
	T_{tCKC}	Cycle time	300	2500	200	2500	165	2500	ns	
$\overline{RESET}$	T_{RSL}	Low time for part reset	3		2		1.6		μs	
IEO	T_{DIEOH}	IEO delay from rising edge of IEI		200		130		100	ns	Load 100 pf +
	T_{DIEOL}	IEO delay from falling edge of IEI		200		130		100	ns	1 TTL load
	T_{DIEOM}	IEO delay from falling edge of $\overline{M1}$(interrupt occurring just prior to $\overline{M1}$)		270		190		110	ns	
	T_{DIEOA}	Delay to rising IEO from rising $\overline{IORQ}$ during interrupt acknowledge		1000		800		600	ns	
	T_{DIEOR}	Delay to rising IEO from rising edge of RD during ED fetch of RETI		500		400		300	ns	
$\overline{INT}$	T_{DIX}	Delay to falling $\overline{INT}$ from external interrupt active transition		550		380		300	ns	Open drain load 100 pf + 2.1 K resistor

III

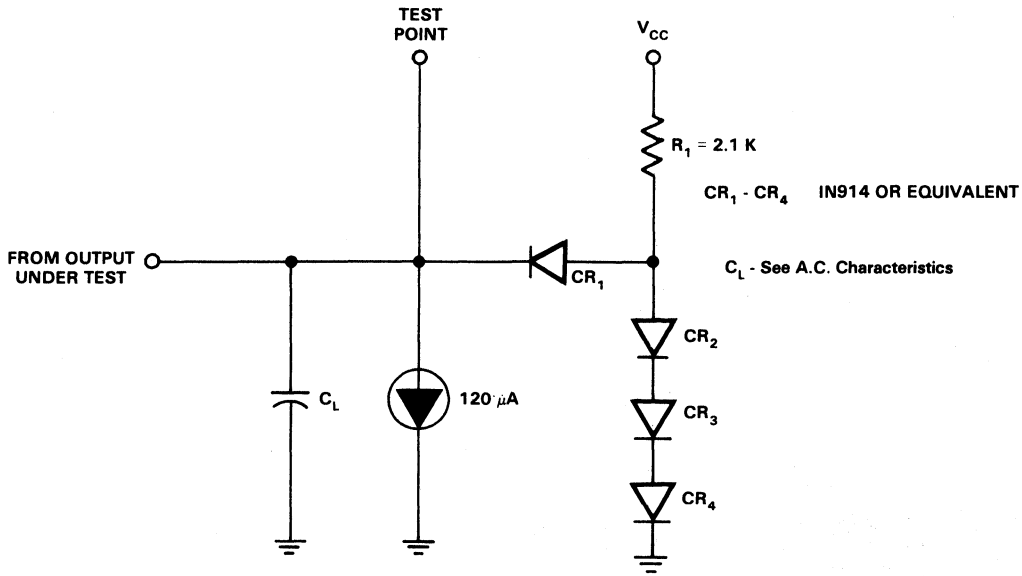
A.C. CHARACTERISTICS (Continued)

SIGNAL	SYMBOL	PARAMETER	MK3801-0 MIN	MK3801-0 MAX	MK3801-4 MIN	MK3801-4 MAX	MK3801-6 MIN	MK3801-6 MAX	UNIT	CONDITION
	T_{DII}	Delay to falling INT from internal interrupt transition		360		280		250	ns	
	T_{DTI}	Transmitter Internal interrupt transition delay from rising or falling edge of TC		560		390		360	ns	
	T_{DRI}	Receiver buffer full internal interrupt transition delay from rising edge of RC		400		300		270	ns	
	T_{DREI}	Receiver error internal interrupt transition delay from falling edge of RC		550		430		400	ns	
SI	T_{SSI}	Serial in set up time to rising edge of RC (Divide by one only)	80		80		55		ns	
	T_{HSI}	Data hold time from rising edge of RC (Divide by one only)	400		350		300		ns	
SO	T_{DSO}	Data valid from falling edge of TC		420		390		345	ns	100 pf + 1 TTL load
TC	T_{TCL}	Low time	650		500		400		ns	
	T_{TCH}	High time	650		500		400		ns	
	T_{TCCY}	Cycle time	1.5		1.05		.85		μs	
RC	T_{RCL}	Low time	650		500		400		ns	
	T_{RCH}	High time	650		500		400		ns	
	T_{RCCY}	Cycle time	1.5		1.05		.85		μs	

NOTES:

1. One wait state must be inserted when used as a 6 MHz memory mapped device.
2. All A.C. measurements are referenced to V_{IL} max., V_{IH} min., V_{08} (0.8 V), or (2.0 V).

OUTPUT LOAD CIRCUIT
Figure 19

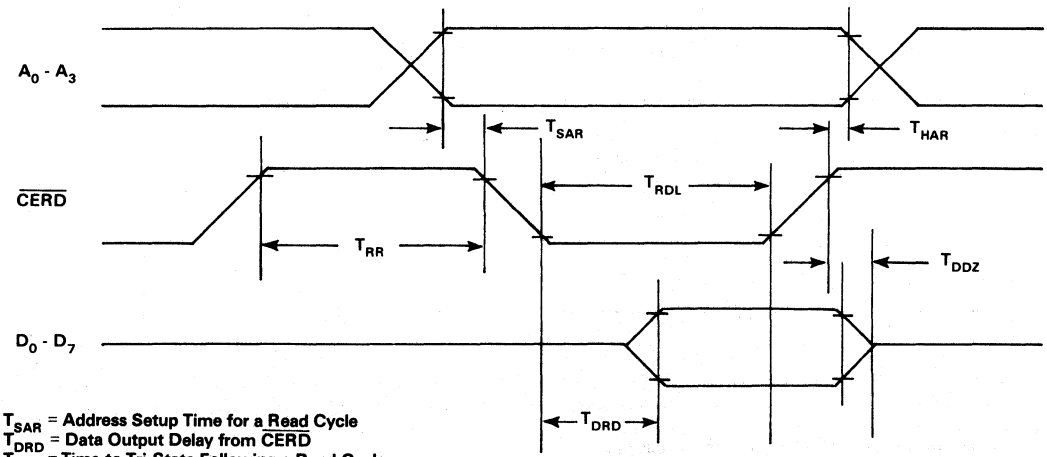


TIMING DIAGRAMS
Figure 20

Timing measurements are made at the following voltages, unless otherwise specified:

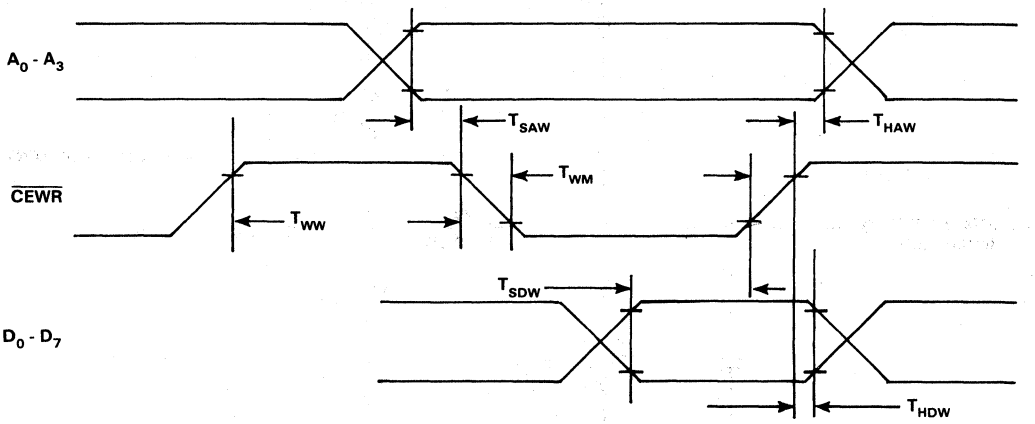
READ CYCLE

	"1"		"0"
OUTPUT	2.0 V		0.8 V
INPUT	2.0 V		0.8 V
FLOAT	Δ V	=	0.5 V



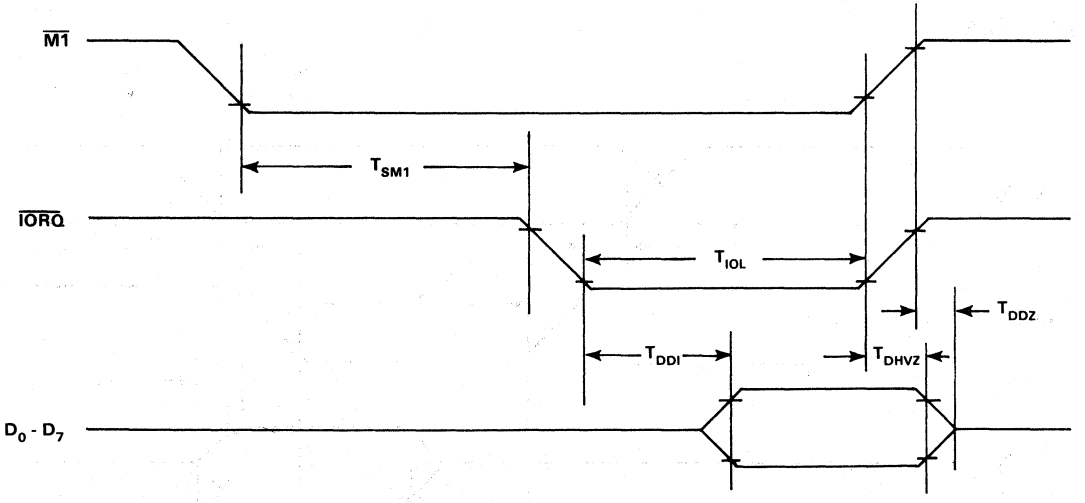
T_{SAR} = Address Setup Time for a Read Cycle
T_{DRD} = Data Output Delay from $\overline{CERD}$
T_{DDZ} = Time to Tri-State Following a Read Cycle
T_{HAR} = Required Address Hold Time Following a Read Cycle

WRITE CYCLE
Figure 21



T_{WW} = $\overline{CEWR}$ High Time Between Writes
T_{SAW} = Address Setup Time for a Write Cycle
T_{SDW} = Data Setup Time Prior to the End of a Write Cycle
T_{HDW} = Required Data Hold Time Following a Write Cycle
T_{HAW} = Required Address Hold Time Following a Write Cycle
T_{WM} = $\overline{CEWR}$ pulse width low

INTERRUPT ACKNOWLEDGE CYCLE
Figure 22



T_{IOL} = $\overline{IORQ}$ Pulse Width Low
T_{SMI} = $\overline{M1}$ Setup Time prior to $\overline{IORQ}$ For an Acknowledge cycle
T_{DDI} = Access Time for Vector
T_{DDZ} = Time to Tri-State Following a Vector
T_{DHVZ} = Data hold time following $\overline{M1}$ $\overline{IORQ}$ during interrupt acknowledge cycle

TIMER A.C. CHARACTERISTICS

Definitions:

Error = Indicated Time Value - Actual Time Value

tpsc = t_{CLK} x Prescale Value

Internal Timer Mode

Single Interval Error (free running) (Note 2) ..... ± 100 ns
Cumulative Internal Error 0
Error Between Two Timer Reads ..... ± (tpsc + 4 t_{CLK})
Start Timer to Stop Timer Error ..... 2 t_{CLK} + 100 ns to –(tpsc + 6t_{CLK} + 100 ns)
Start Timer to Read Timer Error ..... 0 to –(tpsc + 6 t_{CLK} + 400 ns)
Start Timer to Interrupt Request Error (Note 3) ..... –2 t_{CLK} to –(4t_{CLK} +800 ns)

Pulse Width Measurement Mode

Measurement Accuracy (Note 1) ..... 2 t_{CLK} to –(tpsc + 4t_{CLK})
Minimum Pulse Width ..... 4t_{CLK}

Event Counter Mode

Minimum Active Time of I_3, I_4 ..... 4t_{CLK}
Minimum Inactive Time of I_3, I_4 ..... 4t_{CLK}

NOTES:
1. Error may be cumulative if repetitively performed.
2. Error with respect to T_{OUT} or INT if note 3 is true.
3. Assuming it is possible for the timer to make an interrupt request immediately.

ORDERING INFORMATION

PART NO.	DESIGNATOR	PACKAGE TYPE	MAX CLOCK FREQUENCY	TEMPERATURE RANGE
MK3801N-0	Z80-STI	Plastic	2.5 MHz	0 to 70°C
MK3801N-4	Z80-STI	Plastic	4.0 MHz	0 to 70°C
MK3801N-6	Z80-STI	Plastic	6.0 MHz	0 to 70°C

1982/1983 Z80 DESIGNERS GUIDE

MOSTEK®

MICROCOMPUTER COMPONENTS

Technical Manual

IV

MDL FAMILY

1.0 INTRODUCTION

The Micro Data Link (MDL) Family of Serial Peripherals employs a simple serial protocol to allow easy interface to single chip microcomputers or other devices without consuming many package pins. This reduces the pin count of the MDL devices, thus allowing smaller packages or more control functions than could be obtained had a parallel data interface been employed.

While the individual MDL peripherals perform various functions, the interface to the host system (typically, a single chip microcomputer) is common across the family. The details of that interface are discussed herein. The individual data sheets should be consulted for the exact timing specifications of each device.

2.0 MDL SERIAL INTERFACE SIGNALS

The following signals are common to all Mostek MDL devices:

SCLK - Shift Clock (Input)
SIO - Serial Data In/Out (Bidirectional)
CE, $\overline{CE}$ - Chip Enable (Input)
$\overline{DOF}$ - Data Out Flag (Output)

2.1 SCLK

SCLK provides the timing for the bit by bit data transfers to or from the MDL peripheral. Transmitted data changes states on the falling edge of SCLK. Received data is sampled on the rising edge of SCLK.

2.2 SIO

Data is transmitted to and from MDL peripherals over the SIO line. All data transfers are LSB (Least Significant Bit) first.

2.3 CHIP ENABLE

All MDL devices have at least one chip enable input. Chip enable inputs may be either active high (CE) or active low ($\overline{CE}$). When the chip enable inputs is in the inactive state, the serial data transfer logic is forced into a reset state. On devices with more than one chip enable, all chip enable inputs must be active concurrently to allow a data transfer. If any one chip enable is in an inactive state, the serial transfer logic will be placed in the reset (disabled) state. All further references to chip enable inputs will be to CE, the active true type chip enable. To use an active low chip enable ($\overline{CE}$) properly, all waveforms shown for CE need simply be inverted.

2.4 $\overline{DOF}$

The output for the Data Out Flag (active low) is not essential to the MDL Serial Data Link. It is provided on many MDL devices and is useful when buffering of the SIO line is required. $\overline{DOF}$ will go low when the MDL peripheral is transmitting data; otherwise, it will be at a high level.

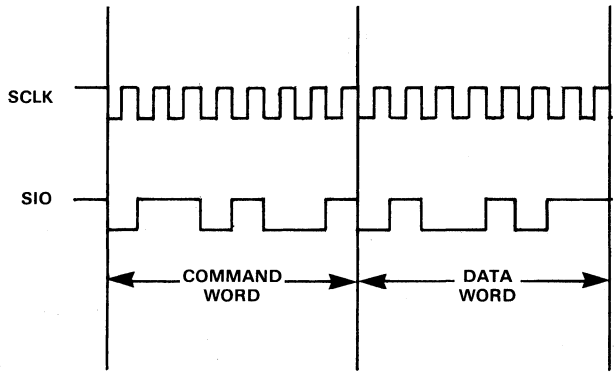
3.0 MDL PROTOCOL

The Micro Data Link protocol is a simple synchronous serial protocol. While the serial interface makes it more practical to locate peripherals some distance away from the CPU than one might if a parallel interface were used, the objective in going to a serial interface was simply to reduce pin count. Thus, the MDL protocol does not employ error detection as is common in serial protocols used for remote data acquisition. It is intentionally designed to be very simple, thus allowing easy communication with a CPU wherein the serial interface is handled by bit manipulation rather than in complex hardware.

3.1 GENERAL FORM

All MDL data transfers are comprised of two 8-bit words. The first word is a command word (CW) which is always transmitted by the CPU. The second word is a data word (DW) which is transmitted by the CPU when writing data to the peripheral or transmitting by the peripheral when the CPU is reading data from the peripheral. The CW and DW are always adjacent (no SCLK cycles between the end of the CW and the start of the DW); thus all transfers take place in 16 SCLK cycles. This is illustrated in Figure 3.1.1

MDL FORMAT
Figure 3.1.1.



3.2 COMMAND WORD

There are two possible commands, read and write. The first bit of the CW (LSB) is a start bit and must always be a "0". The second bit is the Write/$\overline{\text{Read}}$ control bit which must be a "1" for a write (data written to the peripheral from the CPU) or a "0" for a read. The remaining 6 bits of the CW are the address to which data is written or from which data is read. The CW is illustrated in Figure 3.2.1.

COMMAND WORD
Figure 3.2.1

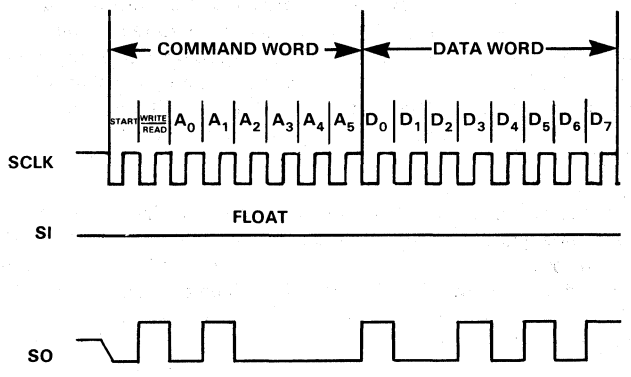
LSB							MSB
START	WRITE / $\overline{\text{READ}}$	A_0	A_1	A_2	A_3	A_4	A_5

3.3 DATA WORD

The DW is simply 8 bits of data written to or read from the peripheral. Figure 3.3.1 and Figure 3.3.2 illustrate the transmission of the DW. The Serial In/Out line (SIO) is separated into a Serial In and a Serial Out for purposes of this illustration. Thus, when Serial In (SI) is active, the CPU is driving SIO. When SI is in the "float" condition (high impedance), the CPU is not driving SIO. Similarly, the peripheral is driving Serial Out when it is shown to be active and not driving SO when SO is shown in the "float" condition. SIO is, of course, the combination of SI and SO.

WRITE OPERATION

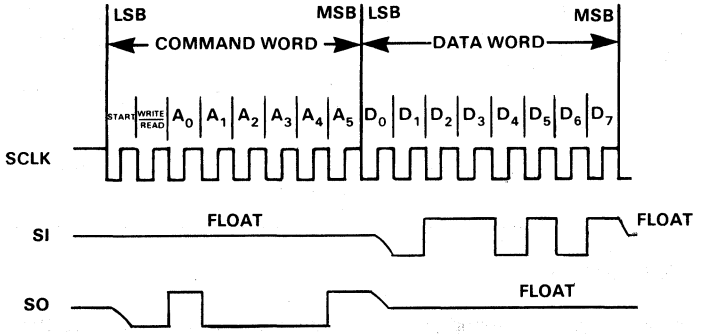
Figure 3.3.1



Shown is a Write to Address 02 Hex of Data A9 Hex.

READ OPERATION

Figure 3.3.2



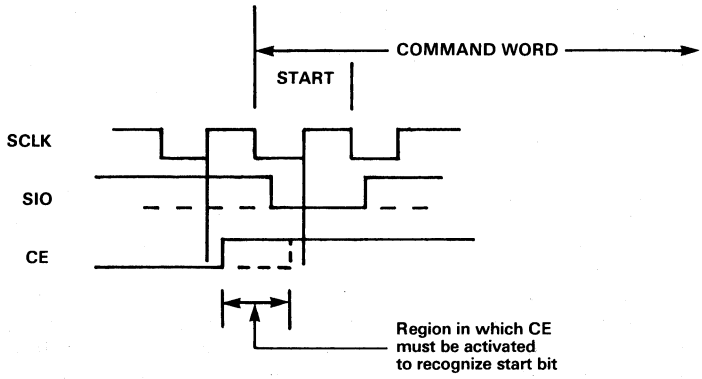
Shown is a Read from Address 21 Hex of Data A6 Hex.

4.0 SELECTION GATING METHODS

4.1 CHIP ENABLE GATING

As previously described, chip enable will hold the serial interface logic of the peripheral in an inactive state as long as CE is inactive. In this state, the peripheral cannot drive SIO, nor does it monitor SIO as an input. Thus, any action occurring on SCLK or SIO is ignored. CE can be used to deactivate a peripheral during the interval between data transfers and SCLK and SIO may continue to be active. CE must go active during data transfers to the peripheral in time for the peripheral to recognize the start bit of the CW. This is illustrated in Figure 4.1.1.

CHIP ENABLE GATING
Figure 4.1.1

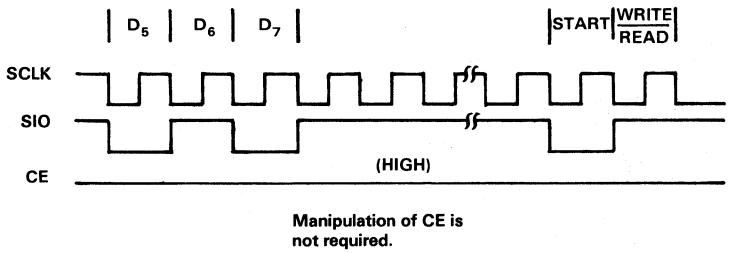


As shown in Figure 4.1.1, CE must not be activated before the previous rising edge of SCLK or else the peripheral could recognize a start bit one cycle too early. If SIO was low during that cycle, CE must be activated before the rising edge of SCLK during the cycle in which the start bit is issued by the CPU.

4.2 START BIT DETECTION

After any data transfer is completed, the peripheral will immediately begin to look for a new start bit if its CE is active. Thus, transfers can occur back to back without any SCLK cycles separating the D_7 bit of the DW and the start bit of the next CW. However, until a start bit is seen by the peripheral, it will take no action. Any dead time between transfers can occur without a problem if the CPU holds SIO high. In this event, CE need not be deactivated nor SCLK stopped.

START BIT DETECTION
Figure 4.2.1



4.3 SCLK MANIPULATION

Dead time between transfers can also be filled by stopping SCLK. This essentially places all transfers back to back. After the rising edge of SCLK in the last bit of the DW, the data will be latched into the peripheral and SIO can then change without altering D_7 (Write Operation). Similarly for Reads, the CPU should have latched in D_7 on the rising edge of SCLK and SCLK could then stop. However, the peripheral will continue to drive SIO until the falling edge of SCLK unless CE is deactivated. While all MDL peripherals sample incoming data on the rising edge of SCLK, they might not actually write the assembled word into the appropriate register until the falling edge of SCLK at the end of the DW (unless CE is deactivated at which time it would complete the Write immediately upon deactivation of CE).

SCLK can also be stopped after the falling edge at the end of the DW. If CE remains active, the peripheral is now looking for a start bit but SIO may make any transitions as long as it is stable at the desired "1" or "0" state ("0" for a start bit) sufficiently before the rising edge of SCLK that will occur once SCLK cycles resume.

4.4 SUMMARY OF SELECTION METHODS

As can be seen from the above discussion, any of three alternatives may be used to prevent false reads or writes between transfers. 1) CE can be deactivated, 2) SIO may be held high, or 3) SCLK may be stopped. Of these methods, perhaps the simplest for CPUs with a hardware serial port (like the MK3873) is to transmit a continual high output between data transfers. However, if the serial port is to be multiplexed between an MDL peripheral (or peripherals) and another device such as an ASCII terminal, deactivating CE when not accessing the MDL device may be more appropriate. (See "CE Timing Considerations.") If the serial interface to the MDL device is performed in software through bit manipulation, stopping the generation of SCLK is perhaps the easiest.

5.0 ADDRESSING

There are six address bits in each CW. These bits define which of 64 possible registers is to be addressed. Many, if not most, MDL peripherals employ less than 64 registers. The MK3821 Serial A/D converter, for example, uses only four registers. For devices with less than 64 registers, the upper address bits are compared to a pre-defined code. If they match this code (called the unit address) then the device will perform the read or write operation. If the unit address of the peripheral is not matched, the peripheral will take no action other than to continue to monitor SCLK cycles to determine when the transfer currently in progress is completed. Upon completion, it will resume looking for a start bit. In the example of the MK3821, A_0 and A_1 are used to determine which of four registers are to be accessed but only if A_2 through A_5 match its unit address. In some MDL devices, the selection of its unit address is "hard-wired" into the device. On other devices like the MK3821, the unit address is user selectable via external strapping options. Thus, 16 MK3821 devices can share the same SCLK, SIO, and CE lines, each being strapped to a unique unit address.

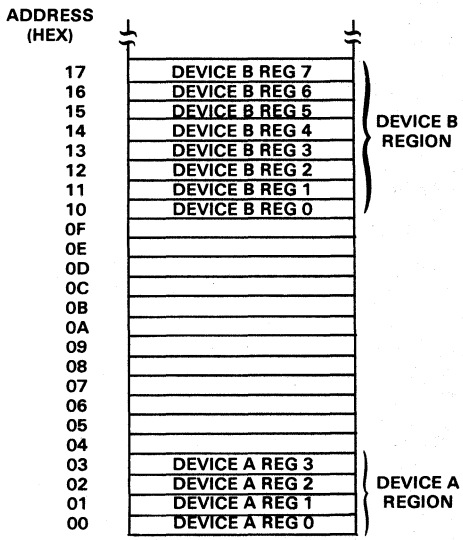
5.1 ADDRESS MAPPING

With devices which allow full unit address selection, the full 64 register map can always be utilized. This is easily accomplished by mapping those devices with the most registers at the lower order addresses. Of course, depending upon the mix of devices, other mappings may also work. Consider two devices. Device A has four internal registers; thus A_2 - A_5 are compared to its unit address. Device B has sixteen internal registers; thus A_0 - A_3 are used to select one of the 16 registers and A_4 and A_5 are compared to its unit address.

If Device A is mapped at unit address 0000 (A_5 - A_2), Device B must be mapped at unit address (UA) 01 (A_5, A_4) or higher. However addresses 04 through 0F hex are an unused hole in the map.

ADDRESS MAPPING

Figure 5.1.1



With Device A starting at address 00 hex, Device B cannot occupy address 04, 05, 06, and so on, because it only has the upper two address bits available for address selection. Thus, its internal register 0 can be mapped to be addressed only at addresses 00, 10, 20, or 30 hex; but address 00 is unavailable because it is occupied by Device A. However, as previously stated, there is always a way to map multiple MDL devices contiguously into the address map.

Note that when uniquely mapped, MDL devices can share the same CE because even when enabled, they will respond only when the address is within their region of the map.

5.2 MULTIPLE REGISTER MAPS

When one desires to access more than the 64 available addresses, this is easily done through CE selection. Multiple devices with up to 64 total internal registers can share the same SCLK, SIO, and CE signals. A second set of devices can share the same SCLK and SIO as the first set but must have a separate CE.

6.0 CHIP ENABLE TIMING CONSIDERATIONS

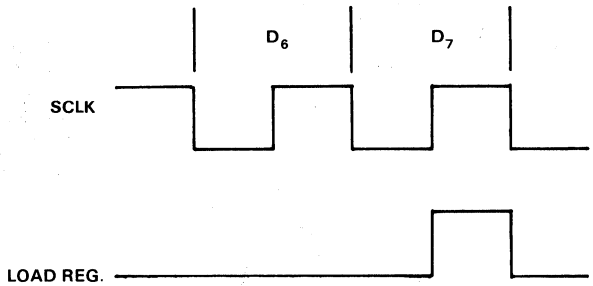
The details of the CE function require some additional discussion. CE basically performs a reset of the serial interface logic. In most cases, this reset occurs immediately after CE is deactivated and will hold the serial interface in this reset state until CE is activated. Upon activating CE, the device will begin to look for a start bit on the next rising SCLK. However, CE must be activated a sufficient time prior to the rising SCLK so that this change in the state of CE has had time to ripple through the internal logic before SCLK rises. If CE is activated while SCLK is already high, the peripheral will not begin to look for a start bit until SCLK first goes low, then returns high. If CE is activated concurrently with a rising SCLK, the resultant action is indeterminate. When using bit manipulation to generate SCLK, SIO, and CE, CE should change states at the same time that SCLK is caused to go low.

6.1 CE DEACTIVATION DURING A TRANSFER

Normally, CE will not be altered during a transfer. If it is deactivated during a transfer, it will abort the transfer. There is potentially one time during a transfer wherein problems might occur. This time is prior to just after the rising edge of SCLK in the last bit (D_7) of the transfer. Primarily, the problem is with write operations. The serial input shift register has been shifting in a data bit on each of the 7 previous rising edges of SCLK. Upon the rising edge of SCLK during the D_7 bit, the last bit is shifted in and the assembled word is loaded into the addressed register in parallel. The timing of the internally generated load register signal is illustrated in Figure 6.1.1.

IV

INTERNALLY GENERATED LOAD REGISTER SIGNAL
Figure 6.1.1

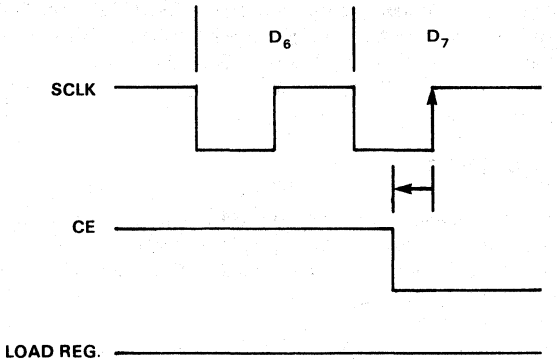


The internal load register signal must be of a sufficient duration. If it is too short, some of the bits of the register may get the new data latched into them but other bits might not. This would depend upon the propagation delay characteristics of the various bits and upon the old and new data. For example, it may be easier to pass a "1" into a given bit which previously contained a "0" than to write a "0" into a bit which was previously a "1". A shortened load register pulse could result in an indeterminate state of the contents of the register.

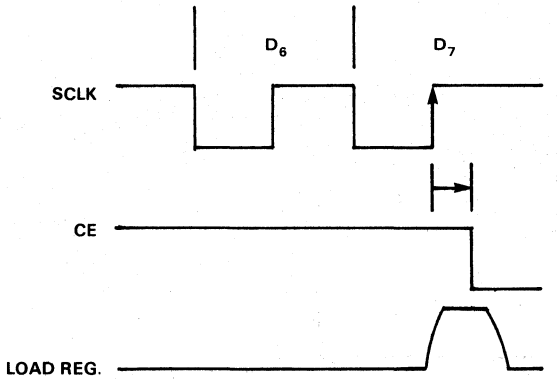
If CE is deactivated sufficiently prior to the critical rising edge of SCLK, the load register signal will not occur. If CE is deactivated sufficiently after the rising edge of SCLK, the load register signal will be deactivated but would have been active long enough to allow a good load. See Figure 6.1.2.

CE DEACTIVATION DURING A WRITE

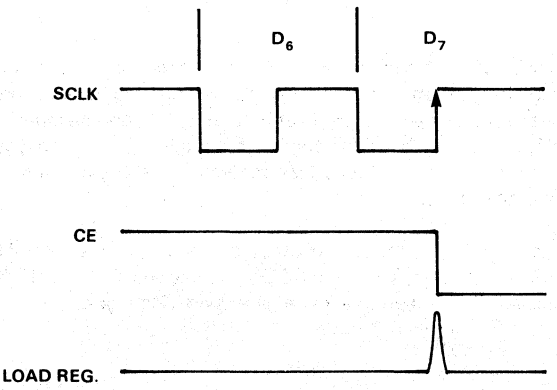
Figure 6.1.2



A. Load Register does not occur because CE is deactivated sufficiently before the rising SCLK.



B. Load Register does occur but lasts a sufficient time before the deactivation of CE causes it to also go to its inactive state.



C. Load Reg. does occur but is terminated too early. This is the problem case.

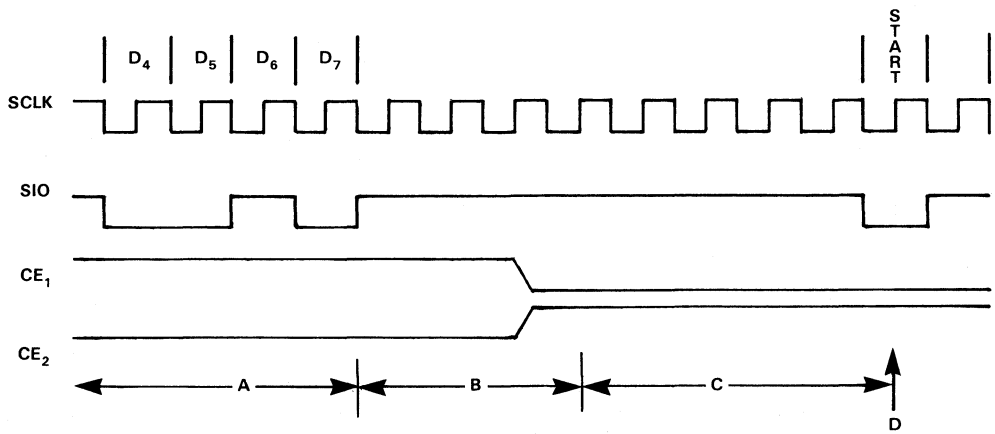
Similar to the write operation, some problems can occur if CE is deactivated during this critical window in D_7 on a read operation. It is not uncommon that status bits, handshake strobes, FIFO address pointers, and so on, are affected when a read occurs. A read complete signal similar in timing to the load register signal would be used to generate the required response to the read. Again, this signal must not be allowed to occur at all or must be allowed to last a sufficient time or a problem could result.

Depending upon the exact nature of the peripheral, the critical window(s) during which CE deactivation may cause a problem may be different than those shown above. Also, depending upon the nature of the peripheral and the system function that it performs, this may not be a significant problem at all. In normal operation, CE would remain active and allow a complete transfer. Only during some extraordinary case such as a power interruption might CE deactivate during a transfer. However, the MK3824 RAM is used specifically to retain key data during CPU power loss in many applications. Thus, it uses some techniques internally to help avoid the truncated load problem. In short, one should be aware of this type of problem and should consult the individual peripheral data sheets for specifics.

6.2 NORMAL CE MANIPULATION

When using a CPU with a high speed serial port, switching CE at the desired time might be difficult. That is, upon being interrupted or having polled status and recognizing that a transfer is complete, one might need to switch one CE to its inactive state and another to its active state before the start of the next transfer. If transfers are to occur back to back, there is less than one bit time to switch CE. An alternative is to transmit a dummy FF hex until the CE switching is accomplished. Anytime that SIO is maintained at a "1" between transfers, CE signals may change without regard to SCLK with no problem. This is illustrated in Figure 6.2.1.

CE MANIPULATION
Figure 6.2.1



A. Unit on CE_1 completes data transfer; unit or units on CE_2 are deactivated.

B. Unit or units on CE_1 look for Start Bit; unit or units on CE_2 are deactivated.

C. Unit or units on CE_1 are deactivated; unit or units on CE_2 are activated and look for Start Bit which is found at D.

7.0 POWER-ON AND RESET

Most peripherals have on-chip power-on-clear circuitry which resets the serial interface logic in a fashion similar to deactivating CE, then activating it. However, this power-on-clear should not be relied upon heavily as it may reset and release the serial interface logic before whatever power-on-clear logic used with the CPU has placed the CPU in a reset state. The peripheral or peripherals may begin to monitor SCLK and SIO to look for a start bit if CE is active before the CPU is capable of correctly supplying them, and bogus transfers may take place or be initiated.

Typically, any complete or partial bogus data transfer will not cause any real problem, but may result in the peripheral being "out of sync" with the CPU. Thus, the CPU might initiate its first transfer but the peripheral might have recognized some power up transients on SCLK and SIO as a start bit and would not correctly interpret the CW. If no bogus transfers can be tolerated and seem to be a possibility, then CE must be held in its inactive state by external logic until the CPU is capable of supplying SCLK, SIO, and perhaps CE properly. Recall that it takes at least 16 cycles of SCLK to complete a transfer, and unless SCLK is being generated by a free-running clock source, it may be highly unlikely that this alone might occur when the CPU is powered up. In general, all that is required is that all peripheral CE lines be deactivated then reactivated prior to the start bit of the first transfer.

7.1 RESET

The MDL system may be reset by deactivating each unit's CE input. For systems with only one MDL device or with separate SI and SO lines (see "Hardware Interface"), the link can also be re-synchronized by transmitting 16 bits of "1" by the CPU. Any transfer in progress must conclude within 16 bit times and the peripheral is guaranteed to be looking for a start bit. However, when two or more units share a common SIO line, if synchronization is lost, one unit can potentially recognize another unit's DW as a CW and in turn its DW (if it interpreted the command as a read it would drive SIO during the DW) could be seen as a new CW by another unit.

7.2 SYSTEM ERROR DETECTION

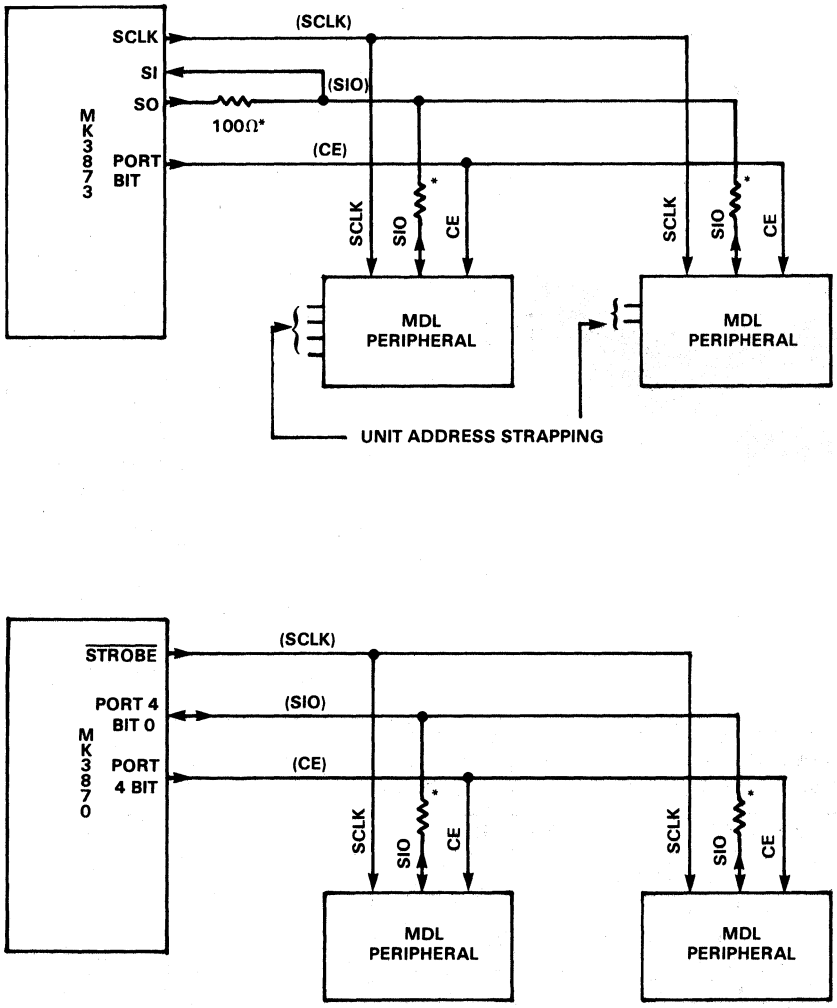
An error should not occur in normal operation. However, power line transients could potentially upset a peripheral or the CPU. Also, a component or connection failure could occur and many systems employ various self tests to monitor their operation. Several alternatives are available for MDL interfaces. In many cases, registers on the peripherals can be read after being written to determine whether the data received back matches that written out. Additionally, the CPU can read the CW while it is being transmitted. Depending upon the hardware used, if one of the peripherals is driving SIO while the CPU is also driving SIO, this can be detected by reading the CW as it is being transmitted. The $\overline{DOF}$ output of a peripheral will go low when a peripheral is driving data out on SIO. This should occur only during the DW in a read operation. Thus, $\overline{DOF}$ of the peripheral or peripherals can be monitored to ensure that it is low only at the proper time.

8.0 HARDWARE INTERFACE

8.1 SIMPLE SYSTEM

Typical interfaces for simple systems are shown in Figure 8.1.1. .

SIMPLE SYSTEM INTERFACES
Figure 8.1.1



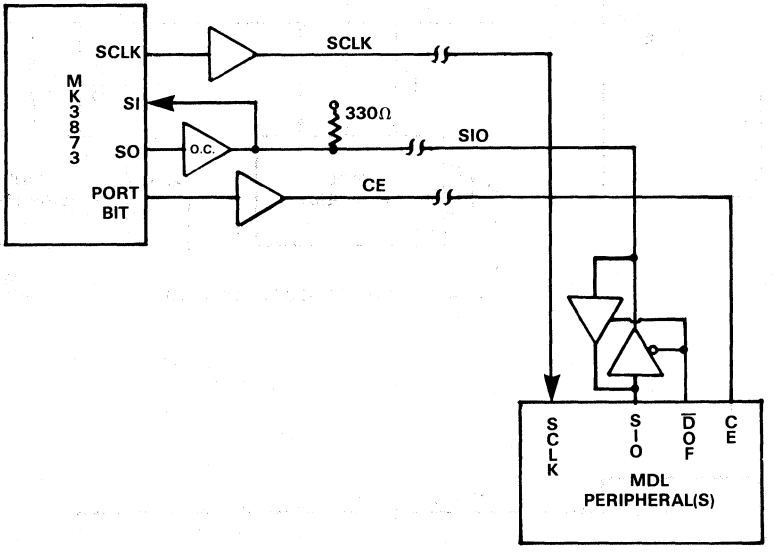
*Optional 100Ω resistors may be inserted where shown to limit current in the event that two or more devices attempt to drive SIO at the same time.

The bulk of MDL applications probably fall into this simple system range. The CE signal shown could just as simply be $\overline{CE}$. If more register addresses are required, multiple port bits can be used for multiple CE signals. In the case where two sets of devices are required, a single enable signal can go to one set and its complement (generated by an inverter) can go to the other set. Some MDL devices have $\overline{CE}$ inputs and some have both $\overline{CE}$ and CE inputs. If mapping permits, those devices with a $\overline{CE}$ input can be placed in one map and devices with CE in another and a single enable signal from the CPU could select one set or the other.

8.2 BUFFERED SIO

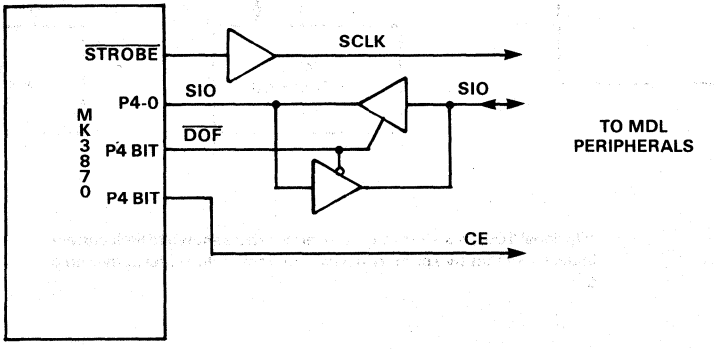
Buffering SIO with the MK3873 Microcomputer requires the generation of a Data Out Flag via a parallel I/O port pin. However, delays of several microseconds would be required in switching this flag and buffer conflicts would occur during that delay. To avoid this, one could use open collector buffering with a passive pull-up on SIO as shown in Figure 8.2.1.

BUFFERED SIO WITH OPEN COLLECTOR BUFFERING
Figure 8.2.1



The MK3870 Microcomputer can easily generate a Data Out Flag in software via bit manipulation along with its generation of SIO and CE. This interface is shown in Figure 8.2.2.

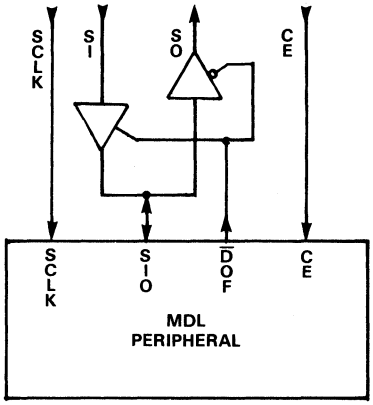
BUFFERED SIO WITH THREE STATE BUFFERING
Figure 8.2.2



8.3 SEPARATE SI AND SO

SIO can be easily separated into an SI and SO line using a bus transceiver controlled by $\overline{DOF}$. The MK3873 Microcomputer provides separate SI and SO so the connection is simple at the CPU end. Note that the peripheral's SI connects to SO of the CPU, and so on. If parallel I/O is used to generate the interface (as with an MK3870), one port pin can be used for SI and another for SO; or a single port pin can be used for SIO and another port pin used to generate a direction control signal, and separate SI and SO created using a bus transceiver.

SEPARATING SIO INTO SI AND SO LINES
Figure 8.3.1



IV

9.0 SOFTWARE

This section is a discussion of various software aspects of handling MDL devices with the MK3873 and MK3870 Microcomputers.

9.1 MK3870 SOFTWARE

(Interface handled via bit manipulation through parallel I/O pins.)

9.1.1 $\overline{\text{STROBE}}$

The $\overline{\text{STROBE}}$ signal associated with I/O Port 4 on 3870 Family devices can be well used in the MDL interface. This signal provides a single low going pulse after each output to Port 4. Thus, it can serve as SCLK for the MDL interface. For a detailed discussion of the 3870 $\overline{\text{STROBE}}$ and the 3870 instruction set, see the 3870 Family Technical Manual.

9.1.2 CONSIDERATION FOR OTHER PORT 4 BITS

Because $\overline{\text{STROBE}}$ pulses after each output to Port 4, it is a good idea to use those pins of Port 4 which are not used in the MDL interface as inputs. As such, they can be read at any time without causing stray $\overline{\text{STROBE}}$ (thus SCLK) transitions. If any of the extra Port 4 pins are to be used as outputs, one should employ an MDL interface which provides a separate CE line for each MDL map whereby all MDL devices may be disabled between data transfers, or one should ensure that SIO remains high between transfers. This will allow outputs to Port 4 to occur between transfers (thus causing an SCLK) without problems. Also, no outputs to Port 4 other than those required for the MDL handling should be made during a transfer. If the extra Port 4 pins are used as inputs, one is free to interrupt a transfer and read the extra Port 4 pins without a problem.

A second problem arises when using the extra Port 4 pins as outputs. During an MDL transfer, many outputs to Port 4 will be performed. It is necessary that those bits of Port 4 not used in the MDL interface but used instead as other output signals be concatenated with the data to be written to Port 4 so that they maintain their desired state. This adds instructions to the MDL interface code. Another point to be considered is that when read, it is the level on the device pin that is fed back into the accumulator, not the state of the output latch. Thus, if any of the extra Port 4 pins are used as outputs in such a way that their output voltage does not reflect their true state, one cannot simply read them back in order to concatenate them with the data to be written to Port 4 during the MDL transfer. This condition occurs most frequently when an output is directly driving the base of an NPN transistor. When a "0" is written to that bit, the device pin will attempt to go high (Port 4 pins being inverted outputs). However, the output voltage will be clamped to about .7 V by the transistor. Had the transistor not been there, the output would have risen to near V_{CC}. If an input of Port 4 is now performed, that bit whose output high level was clamped low by the transistor will be read back as a "1" (again due to the fact that Port 4 pins are inverted signals). Thus a "0" written to that bit would read back as a "1". If that is concatenated with data to be written back out to Port 4, a "1" would be written to that bit causing the pin to go to ground and turning off the transistor when it should have remained on. If this is a possibility, one should keep a copy of all data written to Port 4 in a scratchpad RAM register. When it is necessary to concatenate the extra bits of Port 4 with the data to be written to the MDL interface bits, one could read this scratchpad register instead of the actual Port 4 pins to determine what data to write back to the non-MDL bits of Port 4.

9.1.3 SAMPLE PROGRAM

The following program illustrates how to perform an MDL transfer in software. This example was written to handle a system with two maps of MDL devices. Thus, two CE signals are generated. Also, a $\overline{\text{DOF}}$ signal is generated. This signal is normally low and goes high only when the MK3870 is receiving data (during the DW of a read operation). It employs a copy of Port 4 data in a directly accessible scratchpad register. In this example, it is register 3. Four other scratchpad registers are also used. These registers are assumed to

be in the upper portion of the scratchpad wherein they must be accessed via IS, the indirect scratchpad address register. It is assumed that these four registers lie within a single 8 register segment so that testing for IS roll over and modification of upper IS bits are not required. See the MK3870 Family Technical Manual for more information on the operation of IS.

For the purposes of this example, SIO is generated on Port 4 Bit 0, SCLK is produced by $\overline{\text{STROBE}}$, one CE is produced on Port 4 Bit 1 and the other CE on Port 4 Bit 2, and $\overline{\text{DOF}}$ is produced on Port 4 Bit 3.

The address of the MDL system register to be accessed is placed in the lower 6 bits of the appropriate scratchpad register prior to execution of this code. The seventh bit of that register (Bit 6) is set according to which map is to be accessed. Thus, this bit determines which CE line will be activated. The MSB of that scratchpad register is made a "1" if this is to be a write operation or a "0" for a read. If it is a write, the data to be written is placed in another scratchpad register. The other two scratchpad registers are used for intermediate variables and need not be initialized. Prior to starting execution of this program, IS must be loaded with the address of the scratchpad register which contains the MDL register address.

RESOURCE ALLOCATION

$\overline{\text{P4-0}}$ = SIO
$\overline{\text{P4-1}}$ = CE1
$\overline{\text{P4-2}}$ = CE2
$\overline{\text{P4-3}}$ = $\overline{\text{DOF}}$
$\overline{\text{STROBE}}$ = SCLK

Scratchpad Register 3 = Port 4 data copy (not inverted)

Scratchpad Register S1 = Data
S2 = Intermediate Variable
S3 = MDL Address
S4 = Intermediate Variable

3 Definition

Bits 0-5 = MDL Address
Bit 6 = Map Selection.
"0" Activates CE1,
"1" Activates CE2.
Bit 7 = Command "1" causes a write operation,
"0" causes a read operation.

PRIOR TO EXECUTION

SIO = $\overline{\text{P4-0}}$ = High level on pin (Logic "0" data in port latch)
CE1 = $\overline{\text{P4-1}}$ = Low level on pin
CE2 = $\overline{\text{P4-2}}$ = Low level on pin
$\overline{\text{DOF}}$ = $\overline{\text{P4-3}}$ = Low level on pin
SCLK = $\overline{\text{STROBE}}$ = High level on pin

MSB LSB
Scratchpad Reg R3 = XXXX1110
The upper four bits would reflect the data last written out to those bits of Port 4

Scratchpad Reg S1 = XXXXXXXX for a read operation or data to be written for a write operation.
S2 = XXXXXXXX
S3 =

WRITE / READ	MAP SELECT	ADDRESS

S4 = XXXXXXXX

IS = Pointing to Reg S3.

```
LOC OBJ. CODE          STMT-NR SOURCE-STMT PASS2 IOTWO  IOTWO  IOTWO  REL

                                MK3870 FAMILY MDL SOFTWARE DRIVER
                                   USING PARALLEL I/O PORT 4
                                   AND STROBE  AS SCLK

                        HARDWARE CONFIGURATION:
                        PORT 4-0     SIO
                        PORT 4-1      CE1
                        PORT 4-2      CE2
                        PORT 4-3      DOF
                        STROBE       SCLK

                        ENTRY STATUS:
                        ISAR MUST BE SET TO POINT TO MDL ADDRESS REGISTER
                        DATA REG. MUST CONTAIN DATA TO BE WRITTEN IF WRITE OPERATION
                        PT4IMG SHOULD BE = XXXX1110

                        EXIT STATUS:
                        PT4IMG = XXXX1110
                        DATA = DATA READ FOR READ OPERATION

                        SCRATCHPAD DEFINITION:

          =0003     27 PT4IMG    EQU   3        ;PORT 4 DATA IMAGE (NOT INVERTED)
          =0010     28 DATA      EQU   10H      ;WRITE DATA
          =0011     29 VAR1      EQU   11H      ;INTERMEDIATE VARIABLE
          =0012     30 MDLADR    EQU   12H      ;MDL ADDRESS:
                                                ; BITS 0-5=MDL ADDRESS
                                                ; BIT 6 =MAP SELECTION
                                                ;     0 SELECTS CE1
                                                ;     1 SELECTS CE2
                                                ; BIT 7 =COMMAND
                                                ;     0=READ
                                                ;     1=WRITE
          =0013     38 VAR2      EQU   13H      ;INTERMEDIATE VARIABLE

0000'2080           40 START     LI    80H      ;MASK OFF WRITE/READ BIT
0002 FE             41           NS    D        ; OF MDL ADDRESS REG.
0003 12             42           SR    1        ;MOVE WRITE/READ BIT TO BIT 6
0004 5D             43           LR    I,A      ;STORE IN VAR1 REG.
0005 4C             44           LR    A,S      ;GET MDL ADDRESS
0006 CC             45           AS    S        ;ROTATE MDL ADDRESS LEFT
0007 19             46           LNK            ; MAP SEL=BIT7, WR/RD=BIT0
0008 5C             47           LR    S,A      ;PUT BACK IN MDL ADDRESS REG.
0009 72             48           LIS   2        ;GET CHIP ENABLE PATTERN
000A 8102           49           BP    SETCE    ;SIGN BIT, 0=CE1 1=CE2

                       TO HERE IF CE2 REQUIRED

000C 13             53           SL    1        ;SHIFT CHIP ENA. PAT. FOR CE2
000D'E3             54 SETCE     XS    PT4IMG   ;TOGGLE CHIP ENA. BIT IN PORT IMAGE
```

```
LOC OBJ. CODE        STMT-NR SOURCE-STMT PASS2 IOTWO  IOTWO  IOTWO  REL.

000E 53              55          LR     PT4IMG,A      ;PUT UPDATED IMAGE IN PT4IMG
000F 4C              56          LR     A,S           ;GET MDL ADDRESS
0010 13              57          SL     1             ;SHIFT LEFT TO CREATE COMMAND
                                                          WORD
0011 18              58          COM                  ;INVERT CMD. WORD SINCE PORT
                                                          INVERTS
0012 5D              59          LR     I,A           ;PUT BACK IN MDL ADDR. REG.
                        NOW SET UP LOOP TO SEND COMMAND WORD

0013'78              63 SEND     LIS    8             ;SET LOOP COUNT = 8 BITS
0014 5E              64          LR     D,A           ;PUT LOOP COUNT IN VAR2 REG.
0015'71              65 SENDLP   LIS    1             ;MASK OFF LSB OF
0016 FC              66          NS     S             ; WORD TO BE SENT
0017 E3              67          XS     PT4IMG        ;  MERGE IT WITH PORT 4 IMAGE
0018 B4              68          OUTS   4             ;WRITE TO PORT 4, STROBE IS GENERATED
                                                      ; FOR SCLK
0019 4C              70          LR     A,S           ;GET WORD TO BE SENT
001A 12              71          SR     1             ;MOVE NEXT BIT TO BE SENT INTO LSB
001B 5D              72          LR     I,A           ;PUT BACK INTO REG.
001C 3E              73          DS     D             ;DECREMENT VAR2 REG. = BITS TO BE
                                                          SENT
001D 94F7            74          BNZ    SENDLP        ;IF MORE BITS TO SEND LOOP AGAIN

                        CHECK IF READ OR WRITE OPERATION
                        VAR1 REG. IS 0 FOR A READ AND 40H FOR A WRITE

001F 4E              79          LR     A,D           ;DUMMY LOAD TO MOVE ISAR TO VAR1
                                                      ; LOADED ZERO INTO ACCUM.
0020 CC              81          AS     S             ;THIS SETS ACCUM=VAR1, AND SETS
                                                          STATUS
0021 840B            82          BZ     RECEIV        ;IF ZERO, READ 8 BITS
                                                      ;ELSE, WRITE 8 BITS OF DATA
0023 13              84          SL     1             ;SHIFT VAR1 VALUE LEFT

                        NOTE PRESENT VALUE OF ACCUMULATOR:
                          10000000 IF COMMAND WORD JUST SENT
                          00000000 IF DATA WORD JUST SENT

0024 841F            90          BZ     ENDXFR        ;IF CMD AND DATA SENT, END XFER
                                                      ;ELSE, SEND DATA WORD
0026 5E              92          LR     D,A           ;PUT VALUE OF ACCUM. IN VAR1
0027 4C              93          LR     A,S           ;GET DATA TO BE SENT
0028 6A              94          LISL   MDLADR.AND.7  ;POINT TO MDLADR REG.
0029 18              95          COM                  ;INVERT DATA FOR PORT INVERSION
002A 5D              96          LR     I,A           ;PUT DATA INTO MDLADR REG.
002B 90E7            97          BR     SEND          ;SEND THE DATA

                        TO HERE FOR READ DATA TRANSFER, SET UP TO RECEIVE DATA
```

```
LOC OBJ. CODE
                    STMT-NR SOURCE-STMT PASS2 IOTWO  IOTWO  IOTWO   REL

002D'6B             101 RECEIV   LISL   VAR2. AND. 7 ;POINT TO VAR2
002E 78             102          LIS    8            ;MASK FOR DOF, ALSO BIT COUNT
002F 5E             103          LR     D,A          ;LOAD BIT COUNT INTO VAR2
0030 E3             104          XS     PT4IMG       ;TOGGLE DOF OF PORT 4 IMAGE
0031 53             105          LR     PT4IMG,A     ;PUT BACK IN PORT 4 IMAGE

                        NOW READ DATA FROM PORT 4 BIT 0

0032'43             109 RECVLP   LR     A,PT4IMG     ;GET PORT 4 IMAGE
0033 B4             110          OUTS   4            ;SEND IT TO PORT 4
                                                     ;TO SET DOF LOW AND SEND SCLK
0034 4C             112          LR     A,S          ;GET DATA BEING ASSEMBLED
0035 12             113          SR     1            ;SHIFT RIGHT TO EMPTY MSB
0036 5C             114          LR     S,A          ;PUT DATA BACK
0037 A4             115          INS    4            ;READ PORT 4
0038 15             116          SL     4            ;MOVE BIT 0
0039 13             117          SL     1            ; TO
003A 13             118          SL     1            ;    BIT
003B 13             119          SL     1            ;     7
003C EC             120          XS     S            ;MERGE BIT RECEIVED WITH DATA
003D 5D             121          LR     I,A          ;STORE ASSEMBLED DATA
003E 3E             122          DS     D            ;DECREMENT BIT COUNT
003F 94F2           123          BNZ    RECVLP       ;IF NOT LAST BIT, LOOP AGAIN
0041 4C             124          LR     A,S          ;GET ASSEMBLED DATA
0042 18             125          COM                 ;INVERT RECEIVED DATA
                                                     ;SINCE PORT INVERTED DATA
0043 5C             127          LR     S,A          ;PUT CORRECTED DATA BACK IN REG.
0044'43             128 ENDXFR   LR     A,PT4IMG     ;GET PORT IMAGE
0045 220E           129          OI     0EH          ;SET BITS 1, 2, 3 HIGH TO DISABLE
                                                     ;CHIP ENABLE AND DOF
0047 B4             131          OUTS   4            ;SEND TO PORT
0048 53             132          LR     PT4IMG,A     ; AND UPDATE PORT IMAGE
0049 1C             133          POP                 ;RETURN
```

IV

AFTER EXECUTION

SIO = $\overline{P4\text{-}0}$ = High
CE1 = $\overline{P4\text{-}1}$ = Low
CE2 = $\overline{P4\text{-}2}$ = Low
$\overline{DOF}$ = $\overline{P4\text{-}3}$ = Low
SCLK = $\overline{STROBE}$ = High (one extra SCLK cycle having occurred due to restoring CE and $\overline{DOF}$)

Reg 3 (Port 4 copy) = XXXX1110

Reg S1 = Data just sent if this was a write operation or the same as when execution began if this was a read operation
Reg S2 = 0 0 0 0 0 0 0 0
Reg S3 = Data just read if this was a read operation or all zeros if this was a write operation
Reg S4 = All zero

IS = Pointing to S2 if this was a write operation or pointing to S3 (where the data is) if this was a read.

PROGRAM TIMING

Timings listed are for a 3870 with a 4 MHz time base (4 MHz crystal) unless otherwise shown. One can easily scale this to obtain timings at other frequencies.

The execution time for a write operation is 589 μs if CE1 is activated and 591 μs if CE2 is activated. For a read operation it is 692 μs or 693 μs depending upon whether CE1 or CE2 was activated. To execute repeatedly, Read or Write would add approximately 58 μs to handle the data for each operation and approximately 10 μs to initialize pointers prior to starting. Thus, to write 16 consecutive MDL addresses would take 10 μs plus 16 times of 647 μs. This is a total of 10.362 ms. The effective SCLK rate would then be approximately 24.7 kHz. The data rate would be 16-8 bit words moved in 10.362 ms or approximately 1544 words per second. For 16 consecutive read operations, it would take approximately 11.53 ms or approximately 1387 words per second. The above word rates include the overhead of sending the command word.

With regard to the MDL peripherals, the following timings apply for write operations.
Minimum SCLK High Time = approximately 26 μs
Minimum SCLK Low Time = approximately 4 μs
SIO and CE stable prior to rising edge of SCLK = approximately 5.5μs
SIO held past rising SCLK = approximately 24.5 μs

For read operations, the above timings apply when the 3870 is sending the command word. Additionally, the following requirements occur during the reading of the data word.
SIO stable from falling edge of SCLK = approximately 11.5 μs
SIO hold after rising edge of SCLK = approximately 8.5 μs

The timings are listed as "approximately" because propagation delays were not calculated.

WAVEFORMS PRODUCED FOR A READ OPERATION
Figure 9.1.3.1

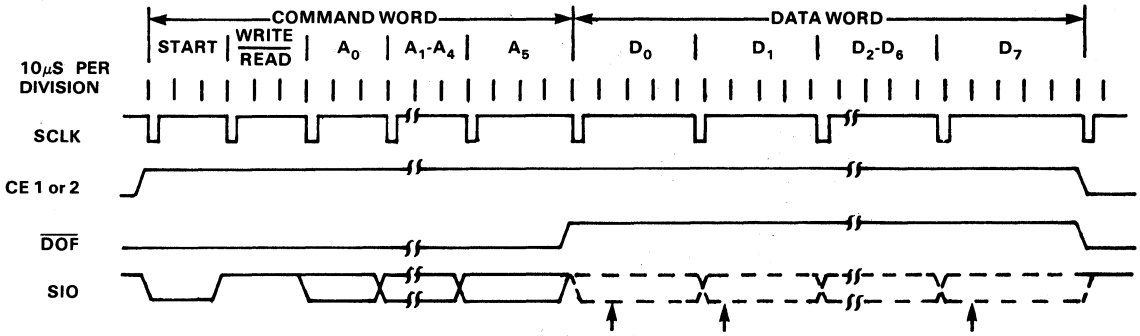


Figure 9.1.3.1 is an example of waveforms produced for a Read operation. All outputs from 3870 (CE1 or CE2, $\overline{DOF}$, and SIO during CW and during DW or Write Operations) actually change shortly before the falling edge of SCLK with this program. The dotted portion of SIO shows when the MDL peripheral is driving it. Peripherals change SIO just after falling SCLK. Arrows show the approximate point when 3870 reads SIO pin. Peripherals sample SIO pin on rising SCLK with some required set up time prior to rising SCLK and some required hold time after rising SCLK.

IV

9.1.4 $\overline{CE}$ SOFTWARE

If it is desired to generate $\overline{CE}$ signals instead of CE signals as shown in the sample program, only minor modification is required. In this case, the state of Bits 1 and 2 in both the Port 4 copy in register 3 and on the port itself would be inverted upon entering the program. The code that activates the desired $\overline{CE}$ signal would not change as it simply causes CE or $\overline{CE}$ to toggle. Thus, whether the signal is normally a 1 or a 0 would not matter. However, to deactivate a $\overline{CE}$, one would have to modify the code from that previously shown. Starting at END, the following code would be used.

```
END LR A, PT4IMG   Get copy of Port 4
    OI '08'        Set Bit 3 to return DOF low in case this was a Read.
    NI 'F8'        Clear Bits 0-2 to restore which ever CE that was activated to a "0".
    OUTS 4         Update Port 4.
    LR PT4IMG, A   Update Port 4 copy.
```

If, for example, CE1 was to be a $\overline{CE}$ and CE2 was non-inverted, the code would be as follows.

```
END LR A, PT4IMG   Get copy of Port 4
    OI 'OC'        Set Bit 3 and Bit 2 to return DOF and CE2 low had they previously been high.
    NI 'FC'        Clears Bits 0 and 1 to restore CE1 high had it previously been low.
    OUTS 4         Update Port 4.
    LR PT4IMG, A   Update Port 4 copy.
```

9.1.5 3870 MDL SAMPLE PROGRAM THAT DOES NOT USE STROBE

While $\overline{STROBE}$ serves well as SCLK of the MDL interface, it also serves well in a variety of other uses. Thus, one might want to use it for another function, requiring SCLK to be generated in software. The following program is identical in assumptions to the one in section 9.1.3 except that Port 0 is used instead of Port 4 and SCLK is generated on Port 0 Bit 4. Also, SCLK now stays low between transfers instead of high so the state of Reg 3 upon entering the program is XXX11110.

```
LOC OBJ. CODE          STMT-NR SOURCE-STMT PASS2 IOONE  IOONE  IOONE  REL

                                MK3870 FAMILY MDL SOFTWARE DRIVER
                                  USING PARALLEL I/O PORTS

                        HARDWARE CONFIGURATION:
                        PORT0-0     SIO
                        PORT0-1     CE1
                        PORT0-2     CE2
                        PORT0-3     DOF
                        PORT0-4     SCLK

                        ENTRY STATUS:
                        ISAR MUST BE SET TO POINT TO MDL ADDRESS REGISTER
                        DATA REG. MUST CONTAIN DATA TO BE WRITTEN IF WRITE OPERATION
                        PT0IMG SHOULD BE = XXX11110

                        EXIT STATUS:
                        PT0IMG = XXX11110
                        DATA = DATA READ FOR READ OPERATION

                        SCRATCHPAD DEFINITION:

          = 0003    26 PT0IMG   EQU   3       ;PORT 0 DATA IMAGE (NOT INVERTED)
          = 0010    27 DATA     EQU   10H     ;WRITE DATA
          = 0011    28 VAR1     EQU   11H     ;INTERMEDIATE VARIABLE
          = 0012    29 MDLADR   EQU   12H     ;MDL ADDRESS:
                                              ; BITS 0-5=MDL ADDRESS
                                              ; BIT 6 =MAP SELECTION
                                              ;     0 SELECTS CE1
                                              ;     1 SELECTS CE2
                                              ; BIT 7 =COMMAND
                                              ;     0=READ
                                              ;     1=WRITE
          = 0013    37 VAR2     EQU   13H     ;INTERMEDIATE VARIABLE

0000'2080           39 START    LI    80H     ;MASK OFF WRITE/READ BIT
0002 FE             40          NS    D       ; OF MDL ADDRESS REG.
0003 12             41          SR    1       ;MOVE WRITE/READ BIT TO BIT 6
0004 5D             42          LR    I,A     ;STORE IN VAR1 REG.
0005 4C             43          LR    A,S     ;GET MDL ADDRESS
0006 CC             44          AS    S       ;ROTATE MDL ADDRESS LEFT
0007 19             45          LNK           ; MAP SEL=BIT7, WR/RD=BIT0
0008 5C             46          LR    S,A     ;PUT BACK IN MDL ADDRESS REG.
0009 72             47          LIS   2       ;GET CHIP ENABLE PATTERN
000A 8102           48          BP    SETCE   ;SIGN BIT, 0=CE1 1=CE2

                        TO HERE IF CE2 REQUIRED

000C 13             52          SL    1       ;SHIFT CHIP ENA. PAT. FOR CE2
000D'E3             53 SETCE    XS    PT0IMG  ;TOGGLE CHIP ENA. BIT IN PORT IMAGE
```

```
LOC OBJ. CODE        STMT-NR SOURCE-STMT PASS2 IOONE  IOONE  IOONE  REL.

000E 53              54          LR     PT0IMG,A     ;PUT UPDATED IMAGE IN PT0IMG
000F 4C              55          LR     A,S          ;GET MDL ADDRESS
0010 13              56          SL     1            ;SHIFT LEFT TO CREATE COMMAND
                                                       WORD
0011 18              57          COM                 ;INVERT CMD. WORD SINCE PORT
                                                       INVERTS
0012 5D              58          LR     I,A          ;PUT BACK IN MDL ADDR. REG.
                        NOW SET UP LOOP TO SEND COMMAND WORD

0013'78              62 SEND     LIS    8            ;SET LOOP COUNT = 8 BITS
0014 5E              63          LR     D,A          ;PUT LOOP COUNT IN VAR2 REG.
0015'71              64 SENDLP   LIS    1            ;MASK OFF LSB OF
0016 FC              65          NS     S            ; WORD TO BE SENT
0017 E3              66          XS     PT0IMG       ;   MERGE IT WITH PORT 0 IMAGE
0018 B0              67          OUTS   0            ;WRITE TO PORT 0, SETS SCLK LOW
0019 21EF            68          NI     0EFH         ;CHANGE BIT 4 BACK TO 0 (SCLK)
001B B0              69          OUTS   0            ;OUTPUT TO PORT 0, SETS SCLK HIGH
001C 4C              70          LR     A,S          ;GET WORD TO BE SENT
001D 12              71          SR     1            ;MOVE NEXT BIT TO BE SENT INTO LSB
001E 5D              72          LR     I,A          ;PUT BACK INTO REG.
001F 3E              73          DS     D            ;DECREMENT VAR2 REG. = BITS TO BE
                                                       SENT
0020 94F4            74          BNZ    SENDLP       ;IF MORE BITS TO SEND LOOP AGAIN

                        CHECK IF READ OR WRITE OPERATION
                        VAR1 REG. IS 0 FOR A READ AND 40H FOR A WRITE

0022 4E              79          LR     A,D          ;DUMMY LOAD TO MOVE ISAR TO VAR1
                                                     ; LOADED ZERO INTO ACCUM.
0023 CC              81          AS     S            ;THIS SETS ACCUM=VAR1, AND SETS
                                                       STATUS
0024 840B            82          BZ     RECEIV       ;IF ZERO, READ 8 BITS
                                                     ;ELSE, WRITE 8 BITS OF DATA
0026 13              84          SL     1            ;SHIFT VAR1 VALUE LEFT

                        NOTE PRESENT VALUE OF ACCUMULATOR:
                          10000000 IF COMMAND WORD JUST SENT
                          00000000 IF DATA WORD JUST SENT

0027 8424            90          BZ     ENDXFR       ;IF CMD AND DATA SENT, END XFER
                                                     ;ELSE, SEND DATA WORD
0029 5E              92          LR     D,A          ;PUT VALUE OF ACCUM. IN VAR1
002A 4C              93          LR     A,S          ;GET DATA TO BE SENT
002B 6A              94          LISL   MDLADR.AND.7 ;POINT TO MDLADR REG.
002C 18              95          COM                 ;INVERT DATA FOR PORT INVERSION
002D 5D              96          LR     I,A          ;PUT DATA INTO MDLADR REG.
002E 90E4            97          BR     SEND         ;SEND THE DATA

                        TO HERE FOR READ DATA TRANSFER, SET UP TO RECEIVE DATA
```

IV

```
LOC OBJ. CODE        STMT-NR SOURCE-STMT PASS2 IOONE  IOONE  IOONE   REL

0030'6B              101 RECEIV   LISL   VAR2. AND. 7 ;POINT TO VAR2
0031 78              102          LIS    8            ;MASK FOR DOF, ALSO BIT COUNT
0032 5E              103          LR     D,A          ;LOAD BIT COUNT INTO VAR2
0033 E3              104          XS     PT0IMG       ;TOGGLE DOF OF PORT 0 IMAGE
0034 53              105          LR     PT0IMG,A     ;PUT BACK IN PORT 0 IMAGE

                         NOW READ DATA FROM PORT 0 BIT 0

0035'43              109 RECVLP   LR     A,PT0IMG     ;GET PORT 0 IMAGE
0036 B0              110          OUTS   0            ;SEND IT TO PORT 0
                                                     ;TO SET DOF LOW AND SEND SCLK
0037 4C              112          LR     A,S          ;GET DATA BEING ASSEMBLED
0038 12              113          SR     1            ;SHIFT RIGHT TO EMPTY MSB
0039 5C              114          LR     S,A          ;PUT DATA BACK
003A A0              115          INS    0            ;READ PORT 0
003B 15              116          SL     4            ;MOVE BIT 0
003C 13              117          SL     1            ; TO
003D 13              118          SL     1            ;    BIT
003E 13              119          SL     1            ;     7
003F EC              120          XS     S            ;MERGE BIT RECEIVED WITH DATA
0040 5D              121          LR     I,A          ;STORE ASSEMBLED DATA
0041 43              122          LR     A,PT0IMG     ;GET IMAGE OF PORT 0
0042 21EF            123          NI     0EFH         ;SET BIT 4 TO A 0
0044 B0              124          OUTS   0            ;OUTPUT TO PORT 0, SCLK GOES HIGH
0045 3E              125          DS     D            ;DECREMENT BIT COUNT
0046 94EE            126          BNZ    RECVLP       ;IF NOT LAST BIT, LOOP AGAIN
0048 4C              127          LR     A,S
0049 4C              128          LR     A,S          ;GET ASSEMBLED DATA
004A 1B              129          COM                 ;INVERT RECEIVED DATA SINCE
004B 5C              130          LR     S,A          ;PUT CORRECT DATA BACK IN REG
                                                     ; PORT INVERTED DATA
004C'43              132 ENDXFR   LR     A,PT0IMG     ;GET PORT IMAGE
004D 220E            133          OI     0EH          ;SET BITS 1, 2, 3 HIGH TO DISABLE
                                                     ; CHIP ENABLE AND DOF
004F B0              135          OUTS   0            ;SEND TO PORT
0050 53              136          LR     PT0IMG,A     ; AND UPDATE PORT IMAGE
0051 1C              137          POP                 ;RETURN


0052'201E            141 RUN      LI     1EH          ;SET PT0IMG
0054 53              142          LR     PT0IMG,A     ;
0055 B0              143          OUTS   0            ; AND PORT 0
0056 2010            144          LI     10H
0058 62              145          LISU   MDLADR. SHR. 3
0059 6A              146          LISL   MDLADR. AND. 7
005A 5C              147          LR     S,A
005B 280000'         148          PI     START
005E'90FF            149 STOP     BR     STOP
```

EXECUTION TIMING

The execution of this program is about the same as the one given in Section 9.1.3 except that the main send loop requires 5 μs more time and the receive loop requires 3 extra microseconds. The final restoring of CE1 or CE2 and $\overline{DOF}$ (for a read) requires 4 μs less time and a seventeenth SCLK cycle does not occur. Another difference is that all outputs change state coincident with the falling edge of SCLK (plus or minus small propagation delay differences).

The total execution time for a write operation is 665 μs if CE1 is selected or 666 μs if CE2 is selected. The total execution time for a read operation is 752 μs or 753 μs depending upon which CE is selected.

Figure 9.1.5.1 graphically illustrates the waveforms involved.

EXECUTION TIMING
Figure 9.1.5.1

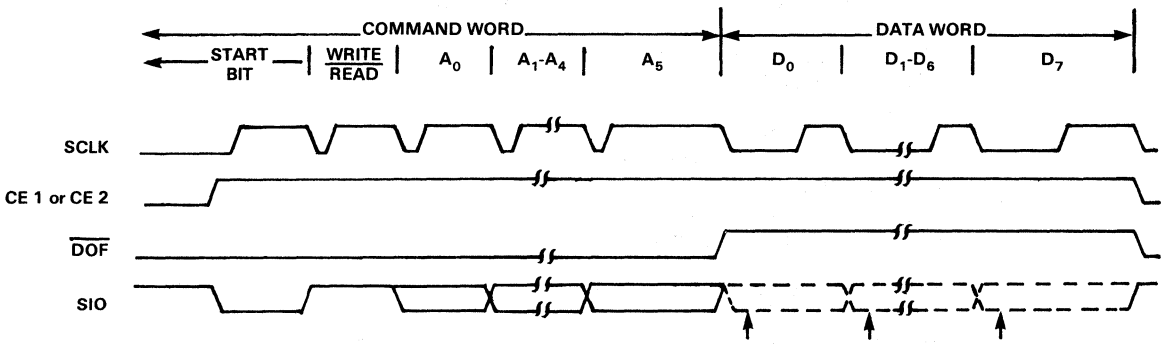


Figure 9.1.5.1 shows the waveforms produced by the program for a read operation. The dotted portion of SIO shows when it would change states when being driven by the MDL peripheral. The arrows show when the CPU samples SIO pin.

9.1.6 POWER-ON CONSIDERATIONS

Principally, the only operation that needs to be performed is to deactivate CE of each peripheral. SCLK need not transition for the peripherals to recognize the deactivation of CE. If using the CE method illustrated in the sample programs, one should place the proper initial condition in the port copy register and output that data to the port. For systems with only one MDL map, CE can be deactivated then reactivated and can remain in the active state throughout normal execution as long as care is taken either not to allow SCLK cycles between accesses or to hold SIO high between accesses.

9.2 MK3873 SOFTWARE FOR MDL INTERFACE

The MK3873 Microcomputer has an on-chip serial port. This port is highly software configurable rather than employing hardware to handle the classical serial interfaces. Thus, while the MDL protocol is not a classical asynchronous or synchronous protocol, it is a simple matter to use the serial port of the 3873 to provide an MDL interface. Other serial ports which are more hardware intensive would require less software support to perform classical serial interfaces, but much more software overhead than the 3873 requires to "trick" them into properly performing the MDL interface.

The general method whereby the 3873 serial port is set up to handle the MDL interface is simple. The port is operated in the synchronous transmit mode. For a write operation, the command word is loaded into the least significant 8 bits of the serial port and the data word is loaded into the most significant 8 bits. The port is set up for a word length of 16 bits. The CW and DW will be shifted out. For a read operation, the CW is loaded into the 8 least significant bits and FF Hex is loaded into the most significant bits.

While data is being shifted out, it is also being shifted in so after all 16 bits have shifted out, the DW from the peripheral will have shifted in and will reside in the 8 most significant bits of the receive buffer. Between MDL transfers, all ones should be loaded into all 16 bits of the port. While SCLK will continue to cycle, SIO will remain high and thus no start bit will occur. The port will continue to retransmit the last word loaded into it. The port can remain in this state indefinitely without CPU intervention.

9.2.1 INITIALIZATION OF THE SERIAL PORT

The serial port should be set up to operate in the synchronous transmit mode with a word length of 16. It will also be set up to generate internally SCLK and buffer it out to the MDL devices. The code to perform the initialization is shown below.

CLR	Load A_{CC} with zeros
COM	Turn 00 into FF
OUTS E	Load MSBs of port with FF
OUTS F	Load LSBs of port with FF
LIS 'B'	Load A_{CC} with 00001011
OUTS C	This sets up 166.66 K bps (with 4 MHz crystal) on SCLK
LI '76'	Get data to set up serial port control
OUTS D	Set up word length = 16, sync, transmit, no interrupt.

9.2.2 SIMPLE SINGLE WORD TRANSFER

Suppose one wants simply to write or read a particular MDL register. The operation can take place in either an interrupt driven or polled mode. Let us first consider the polled mode.

Whether doing a read or a write, it must first be determined when the timing is right to load new data into the serial port. Data must be loaded at a time when an end-of-word (EOW) will not occur. Recall that if initialized as shown in Section 9.2.1, the port is continually transmitting all ones over and over.

```
LOC OBJ. CODE      STMT-NR SOURCE-STMT PASS2 SERONE  SERONE  SERONE   REL
```

MK3873 SERIAL PORT MDL
SOFTWARE DRIVER
SINGLE WORD WRITE

FUNCTION:
THIS ROUTINE PERFORMS A SINGLE WORD WRITE DATA
TRANSFER TO A MDL PERIPHERAL.

ENTRY STATUS:
CMD REG. MUST CONTAIN THE EXACT COMMAND WORD
DATA REG. MUST CONTAIN THE DATA TO BE WRITTEN
CHIPEN REG. MUST CONTAIN THE CHIPEN PATTERN TO
  BE OUTPUT TO THE PORT.
ISAR MUST POINT TO THE CMD REGISTER

EXIT STATUS:
THE COMMAND AND DATA ARE TRANSMITTED AND THE SERIAL
PORT THEN TRANSMITS FF'S.

HARDWARE DEFINITION:
SERIAL PORT SRCLK IS THE MDL SCLK
SERIAL PORT SO AND SI ARE USED FOR THE MDL SIO
PORT 0 BITS 0-3 ARE USED FOR CHIP ENABLES

SCRATCHPAD DEFINITION:

```
        = 0000   31 PT0IMG   EQU   0      ;PORT 0 IMAGE
        = 0010   32 CMD      EQU   10H    ;COMMAND REG.
        = 0011   33 DATA     EQU   11H    ;DATA REG.
        = 0012   34 CHIPEN   EQU   12H    ;CHIP ENABLE PATTERN REG.

0000'70          37 INIT     CLR          ;GET ZEROS
0001 18          38          COM          ;MAKE FF'S
0002 BE          39          OUTS  0EH    ;LOAD FF'S IN SERIAL MSB PORT
0003 BF          40          OUTS  0FH    ;LOAD FF'S IN SERIAL LSB PORT
0004 7B          41          LIS   0BH    ;GET 00001011
0005 BC          42          OUTS  0CH    ;THIS SETS UP SCLK = 166.66 KBPS
                                          ; WITH 4 MHZ CRYSTAL
0006 20E6        44          LI    0E6H   ;SERIAL PORT CONTROL DATA
0008 BD          45          OUTS  0DH    ;SETS 16 BITS,SYNC XMIT,NO INTER.
```

WRITE SINGLE WORD

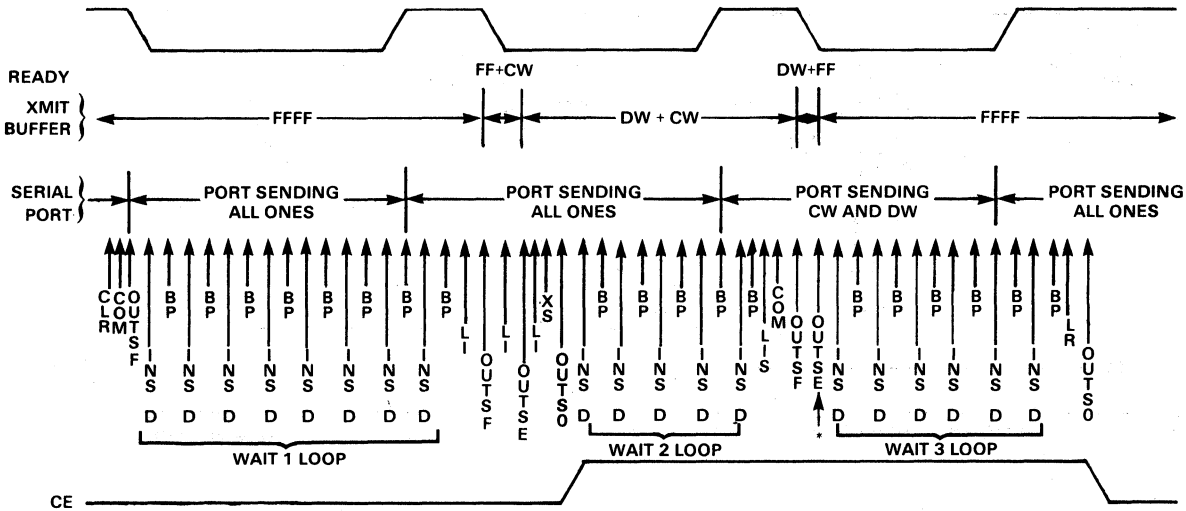
```
0009'70          49 WRITE    CLR          ;GET 0
000A 18          50          COM          ;MAKE FF
000B BF          51          OUTS  0FH    ;CLEAR READY FLAG
000C'AD          52 WAIT1    INS   0DH    ;WAIT1 LOOP
000D'81FE        53          BP    WAIT1  ; UNTIL READY SET
000F'4D          54 GETCW    LR    A,I    ;GET COMMAND WORD
0010 BF          55          OUTS  0FH    ; PUT INTO LOWER BUFFER
```

```
LOC OBJ. CODE        STMT-NR SOURCE-STMT PASS2 SERONE SERONE SERONE  REL.

0011 4D              56          LR     A,I          ;GET DATA WORD
0012 BE              57          OUTS   0EH          ; PUT INTO UPPER BUFFER
0013 4C              58          LR     A,S          ;GET CHIP ENABLE PATTERN
0014 E0              59          XS     PT0IMG       ;TOGGLE CHIP ENABLE BIT
0015 B0              60          OUTS   0            ;SET CE ACTIVE
0016'AD              61 WAIT2    INS    0DH          ;WAIT TILL WORD LOADED INTO
0017 81FE            62          BP     WAIT2        ; SHIFT REGISTER
0019 70              63          LIS    0            ;GET 0
001A 18              64          COM                 ;MAKE IT FF
001B BF              65          OUTS   0FH          ;PUT FF INTO LOWER BUFFER
001C BE              66          OUTS   0EH          ;PUT FF INTO UPPER BUFFER
001D'AD              67 WAIT3    INS    0DH          ;WAIT UNTIL XMIT BUFFER IS
001E 81FE            68          BP     WAIT3        ; LOADED INTO SHIFT REG.
0020 40              69          LR     A,PT0IMG     ;GET IMAGE OF PORT 0
0021 B0              70          OUTS   0            ;SEND TO PORT, SETS CE INACTIVE
0022 1C              71          POP                 ;RETURN
```

The execution time is a function of the serial port Baud rate, and the bit count at the time that execution of this code is encountered. With a 4MHz 3873 time base and maximum internal Baud rate, each bit requires 6 μs. Thus, a full word requires 16 x 6 μs = 96 μs. If the last bit or two of a word was being shifted out at the time that execution started, the maximum execution time would occur.

WORST CASE EXECUTION FOR SINGLE WORD WRITE CODE
Figure 9.2.2.1



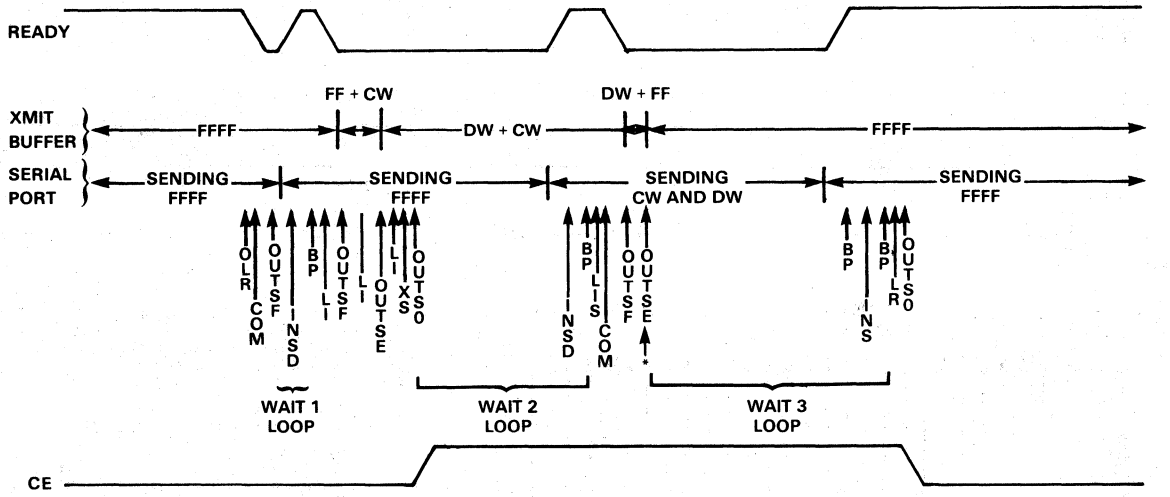
As can be seen from Figure 9.2.2.1, the worst case execution time is three 16 bit word times plus about 30 to 34 μs. This is about 320 μs.

If a CE signal is not switched before and after each transmission as shown, one would not have to wait for the CW and DW to clear the buffer before switching CE and continuing with other code. For a single map system, one could deactivate CE at power-on to reset the serial interface of the peripherals, then reactivate it and leave it continually active. If this was the case, the write code could have concluded with the last OUTS E instruction marked with the asterisk in Figure 9.2.2.1. This would reduce worst case execution time to two word times plus about 40 μs, making the total 232 μs.

Best case execution time would occur if the serial port was near an end of word, but not so

close that the first OUTS F did not occur before the EOW. This is illustrated below in Figure 9.2.2.2.

BEST CASE EXECUTION TIME FOR SINGLE WORD WRITE CODE
Figure 9.2.2.2



Under best case conditions, no time is spent looping in the Wait 1 loop. Thus, execution time is essentially two word times plus about 34 to 38 μs or about 228 μs.

If CE was not switched as shown but left active after the transfer, best case execution would be about one word time plus 4 μs to 46 μs or about 14 μs. Even if there are multiple MDL maps, CE switching is required, and the proper CE could have been selected prior to the transfer and could remain active until just before the next transfer when it could be turned off. The code could be modified between lable CEMSK and the following OUTS 0 as follows.

```
CEMSK   LR A, R     Get Port 0 data copy.
        NI ' '      Deactivate any CE and activate any CE desired.
        OI ' '      Deactivate any CE and activate any desired CE.
        LR R, A     Update Port 0. Copy in register.
        OUTS 0      Switch CE.
```

```
LOC OBJ. CODE        STMT-NR SOURCE-STMT PASS2 SERTWO SERTWO SERTWO  REL
```

MK3873 SERIAL PORT - MDL
SOFTWARE DRIVER
SINGLE WORD WRITE

FUNCTION:
THIS ROUTINE PERFORMS A SINGLE WORD READ OPERATION
WITH A MDL PERIPHERAL

ENTRY STATUS:
ISAR POINTS TO THE COMMAND TO BE SENT
CMD REG. CONTAINS THE COMMAND BYTE
CHIPEN REG. CONTAINS THE CHIP ENABLE PATTERN TO BE
OUTPUT TO THE PORT 0.

EXIT STATUS:
DATA REG. CONTAINS THE DATA BYTE READ

HARDWARE CONFIGURATION:
SERIAL PORT SRCLK IS THE MDL SCLK
SERIAL PORT SO AND SI ARE USED FOR THE MDL SIO
PORT 0 BITS 0-3 ARE USED FOR $\overline{CE1}$ THROUGH $\overline{CE4}$

SCRATCHPAD DEFINITIONS:

```
          = 0000   31 PTOIMG   EQU   0       ;PORT 0 IMAGE
          = 0010   32 CMD      EQU   10H     ;COMMAND REG.
          = 0011   33 DATA     EQU   11H     ;DATA REG.
          = 0012   34 CHIPEN   EQU   12H     ;CHIP ENABLE PATTERN REG.

0000'70            37 INIT     CLR           ;GET ZEROS
0001 18            38          COM           ;MAKE FF'S
0002 BE            39          OUTS  0EH     ;LOAD FF'S IN SERIAL MSB PORT
0003 BF            40          OUTS  0FH     ;LOAD FF'S IN SERIAL LSB PORT
0004 7B            41          LIS   0BH     ;GET 00001011
0005 BC            42          OUTS  0CH     ;THIS SETS UP SCLK = 166.66 KBPS
                                             ; WITH 4 MHZ CRYSTAL
0006 20E6          44          LI    0E6H    ;SERIAL PORT CONTROL DATA
0008 BD            45          OUTS  0DH     ;SETS 16 BITS,SYNC XMIT,NO INTER.

0009 70            47          CLR           ;GET 0
000A 18            48          COM           ;MAKE FF
000B BF            49          OUTS  0FH     ;CLEAR READY FLAG
000C'AD            50 WAIT1    INS   0DH     ;WAIT UNTIL
000D 81FE          51          BP    WAIT1   ; READY FLAG SET
000F'4D            52 GETCW    LR    A,I     ;GET COMMAND
0010 BF            53          OUTS  0FH     ; AND PUT INTO LOWER BUFFER

0011'4D            54 CEMSK    LR    A,I     ;DUMMY LOAD TO INCREMENT ISAR
                                             ;ISAR NOW POINTS TO CHIPEN
```

```
LOC OBJ. CODE        STMT-NR SOURCE-STMT PASS2 SERTWO  SERTWO  SERTWO  REL.

0012 40              56          LR     A,PT0IMG     ;GET PORT IMAGE
0013 EE              57          XS     D            ;MERGE CHIPEN PATTERN
0014 B0              58          OUTS   0            ;OUTPUT TO PORT 0
0015'AD              59 WAIT2    INS    0DH          ;WAIT TILL XMIT BUFFERS
0016 81FE            60          BP     WAIT2        ; ARE LOADED INTO SHIFT REG.
0018 70              61          LIS    0            ;GET 0
0019 18              62          COM                 ;MAKE FF
001A BF              63          OUTS   0FH          ;PUT IN LOWER BUFFER
001B BE              64          OUTS   0EH          ;PUT IN UPPER BUFFER
001C'AD              65 WAIT3    INS    0DH          ;WAIT UNTIL FF'S
001D 81FE            66          BP     WAIT3        ; LOADED INTO SHIFT REG.
001F AE              67          INS    0EH          ;GET DATA READ
0020 5C              68          LR     S,A          ;PUT INTO DATA REG.
0021 40              69          LR     A,PT0IMG     ;GET PORT IMAGE W/O CHIPEN
0022 B0              70          OUTS   0            ;DISABLE CHIP ENABLES
0023 1C              71          POP                 ;RETURN
```

In both the write operation and the read operation shown above, the command word and CE selection was done via immediate data. This could have also come from data elsewhere in the machine as could the data word for the write operation. As can be seen from Figures 9.2.2.1 and 9.2.2.2, time is always spent in the Wait 2 loop. Some of this time could be used to get these pieces of data in some indirect fashion. If this data manipulation required more time than is available, it could be done prior to the first instruction shown in the sample code.

Another option is to execute another code instead of sitting in the Wait 2 or Wait 3 loops. That is, to distribute this code inside some other calculation or data movement routine whereby the instructions themselves dictate that sufficient time (but not too much time) has elapsed. Only the Wait 1 loop is highly variable in duration. This loop causes the code to synchronize its execution with the serial port word times. Once this synchronization has occurred, the times at which serial port reads and writes can occur without problem is known to some reasonable accuracy.

9.2.3 MULTIPLE WORD TRANSFERS IN POLLED MODE

The following program illustrates writing multiple words of data to consecutive registers in a particular MDL device (such as the 3824 serial RAM).

The assumptions are as follows. The words occupy consecutive locations in the scratchpad RAM of the 3873. The location of the first word is in scratchpad register 1 (R1). Thus, if the words were in scratchpad registers 21, 22, 23, . . . register R1 would contain the data '21'. Similarly, the starting MDL address is in register 2 (R2). If R2 contained '34', the data in scratchpad register would be written to MDL register 34, the data in scratchpad register 22 would be written to MDL register 35, and so on. The number of words to be moved out of scratchpad and into MDL registers is in scratchpad register 3 (R3).

It is also assumed that Bit 6 of the MDL pointer (R2) is used to select one of two $\overline{CE}$ lines. Thus, if R2 Bit 6 is a "0", $\overline{CE1}$ will be activated and if it is a "1", $\overline{CE2}$ will be activated. It is also assumed that the wrong $\overline{CE}$ line might be activated at the time that execution of this code begins. $\overline{CE1}$ is generated on Port 0 Bit 6 and $\overline{CE2}$ is generated on Port 0 Bit 5. It is assumed that Port 0 is used only for outputs and that no outputs get clamped so a copy of Port 0 data in scratchpad RAM is not required. Finally, it is assumed that the serial port has been initialized as described in Section 9.2.1. Scratchpad register 4 is used as a working register.

```
LOC OBJ. CODE        STMT-NR SOURCE-STMT PASS2 SERTHR  SERTHR  SERTHR   REL
```

MK3873 SERIAL PORT - MDL
SOFTWARE DRIVER
MULTIPLE WORD TRANSFERS

FUNCTION:
THIS ROUTINE PERFORMS MULTIPLE WORD DATA TRANSFERS WITH A MDL PERIPHERAL VIA THE MK3873 SERIAL PORT

ENTRY STATUS:
POINT REG. CONTAINS THE SCRATCHPAD LOCATION OF THE FIRST WORD TO BE TRANSFERRED. BIT SIX OF POINT IS USED TO SELECT ONE OF TWO $\overline{CE}$ LINES, 0=$\overline{CE1}$, 1=$\overline{CE2}$.
MDLADR REG. IS THE STARTING MDL ADDRESS FOR DATA TO BE WRITTEN TO.
NOWORD REG. IS THE NUMBER OF WORDS TO BE WRITTEN.

EXIT STATUS:
THE NUMBER OF CONSECUTIVE WORDS SPECIFIED ARE TRANSFERRED FROM SCRATCHPAD TO THE MDL STARTING ADDRESS SEQUENTIALLY.

HARDWARE CONFIGURATION:
SERIAL PORT SRCLK IS THE MDL SCLK
SERIAL PORT SO AND SI ARE USED FOR THE MDL SIO
PORT 0 BIT 5 IS $\overline{CE1}$
PORT 0 BIT 6 IS $\overline{CE2}$

SCRATCHPAD DEFINITION:

```
        = 0000   35 POINT   EQU   0          ;POINTER TO DATA LOCATION
        = 0001   36 MDLADR  EQU   1          ;STARTING MDL ADDRESS
        = 0002   37 NOWORD  EQU   2          ;NO. OF WORDS TO BE TRANSFERRED
        = 0004   38 WRKREG  EQU   4          ;WORKING REGISTER

0000'70          41 INIT    CLR              ;GET ZEROS
0001 18          42         COM              ;MAKE FF'S
0002 BE          43         OUTS  0EH        ;LOAD FF'S IN SERIAL MSB PORT
0003 BF          44         OUTS  0FH        ;LOAD FF'S IN SERIAL LSB PORT
0004 7B          45         LIS   0BH        ;GET 00001011
0005 BC          46         OUTS  0CH        ;THIS SETS UP SCLK = 166.66 KBPS
                                              ; WITH 4 MHZ CRYSTAL
0006 20E6        48         LI    0E6H       ;SERIAL PORT CONTROL DATA
0008 BD          49         OUTS  0DH        ;SETS 16 BITS,SYNC XMIT,NO INTER.

0009'41          51 START   LR    A,MDLADR   ;GET MDL ADDRESS
000A 13          52         SL    1          ;SHIFT LEFT, PUTS CE IN BIT 7
000B 1F          53         INC              ;PUTS A 1 IN BIT 0
```

```
LOC OBJ. CODE        STMT-NR SOURCE-STMT PASS2 SERTHR  SERTHR SERTHR   REL.

000C 51              54          LR     MDLADR,A    ;PUT BACK INTO MDL ADDRESS REG.
000D C1              55          AS     MDLADR      ;THIS SHIFTS LEFT, AND SETS STATUS
000E 51              56          LR     MDLADR,A    ;NOW IN CW FORMAT PUT INTO MDLADR
000F 2040            57          LI     40H         ;GET PATTERN TO TOGGLE CE1

0011 9202            58          BNC    SAVE        ;IF CE BIT OF MDLADR WAS 0 DO SAVE
0013 12              59          SR     1           ;IF CE BIT OF MDLADR WAS 1, SHIFT
                                                      ; PATTERN TO TOGGLE CE2
0014'54              61 SAVE     LR     WRKREG,A    ;SAVE CHIP ENABLE PATTERN
0015 A0              62          INS    0           ;READ PORT 0
0016 219F            63          NI     9FH         ;SET BITS 5 AND 6 TO 0
0018 E4              64          XS     WRKREG      ;TOGGLE CHIP ENABLE SELECTED TO A 1
0019 B0              65          OUTS   0           ;ENABLE CHIP
001A 70              66          CLR                ;GET 0
001B 18              67          COM                ;MAKE IT FF
001C BF              68          OUTS   0FH         ;CLEAR READY FLAG
001D'AD              69 MNLOOP   INS    0DH         ;WAIT TILL READY
001E 81FE            70          BP     MNLOOP      ; FLAG SET
0020 41              71          LR     A,MDLADR    ;GET COMMAND WORD
0021 BF              72          OUTS   0FH         ;PUT INTO LOWER BUFFER
0022 2404            73          AI     4           ;INCREMENT MDL ADDRESS
0024 51              74          LR     MDLADR,A    ;STORE NEXT COMMAND BACK IN
                                                       MDLADR
0025 40              75          LR     A,POINT     ;GET POINTER TO DATA
0026 0B              76          LR     IS,A        ;PUT IN ISAR
0027 1F              77          INC                ;INCREMENT POINTER
0028 50              78          LR     POINT,A     ;PUT BACK FOR NEXT WORD
0029 4C              79          LR     A,S         ;GET DATA WORD FOR WRITE
002A BE              80          OUTS   0EH         ;PUT IN UPPER BUFFER
002B 32              81          DS     NOWORD      ;DECREMENT NO. OF WORDS
002C'94F0            82 ENDLOOP  BNZ    MNLOOP      ;IF NOT END OF WORDS, DO AGAIN
002E'AD              83 WAIT1    INS    0DH         ;WAIT UNTIL CW=DW LOADED
002F 81FE            84          BP     WAIT1       ; INTO SHIFT REG.
0031 70              85          CLR                ;GET 0
0032 18              86          COM                ;MAKE IT FF
0033 BE              87          OUTS   0EH         ;PUT INTO UPPER BUFFER
0034 BF              88          OUTS   0FH         ;PUT INTO LOWER BUFFER
0035'AD              89 WAIT2    INS    0DH         ;WAIT UNTIL BUFFERS LOADED
0036 81FE            90          BP     WAIT2       ; INTO SHIFT REG.
0038 A0              91          INS    0           ;READ PORT 0
0039 219F            92          NI     9FH         ;CLEAR CE BITS
003B B0              93          OUTS   0           ;SET CE BITS FALSE IN PORT
003C 1C              94          POP                ;RETURN
```

IV

The main loop, MNLOOP, through ENDLOOP requires 59 μs to execute. It must execute in one word time (96 μs) with a little room to spare which it easily does. The set up portion requires about 54 μs and the ending portion requires about 30 μs past the end of the last CW + DW shifted out. It might require as much as two word times from the time that MNLOOP is first encountered until the first CW + DW is transmitted. Thus, the total execution time is about (N + 2) WT + 84 μs where N is the number of words and WT is the word time. At the 166.6 KHz SCLK rate, WT is 96 μs. To send 16 words would then require about (18 x 96 μs) + 84 μs or about 1.812 ms. This, when compared to the 10.36 ms required when generating the interface through manipulation of parallel I/O pins, shows the advantage of the serial port in terms of speed. However, the parallel I/O version could be optimized somewhat and perhaps reduced by 10% or so. The program just presented executes the main required code faster than the serial port can do its required function. So optimizing it for speed of execution would not materially affect the results.

A multiple word read routine can be generated in similar fashion. This will not be presented in detail in this document. It would also appear that there is sufficient time margin to allow for a Write/Read test upon entering MNLOOP with a branch to either a write loop or a read loop, thus allowing for a composite program which can do either operation. This would allow some of the code to be shared and might save instruction space.

9.2.4 INTERRUPT DRIVEN TRANSFERS

The serial port of the 3873 can interrupt the CPU each time it receives or transmits a full character. Thus, it is not necessary to sit in a wait loop polling the ready flag to determine when the transfer of each word has completed. At maximum Baud rate, the time spent in the wait loops is fairly short (about 40μs or so with a 4 MHz time base), and is about equal to the overhead involved in acknowledging an interrupt, saving existing status, and accumulator contents, performing the desired operation, and returning to the main program. Little throughput advantage may be realized if an interrupt driven routine is used instead of a polled routine. However, at lower Baud rates a considerable throughput advantage might result. Additionally, the interrupt driven method does allow the processor to continue to execute other codes while an MDL transfer is in progress. Thus, it can continue to monitor and respond to other events while a long transfer (either due to a slow Baud rate or due to the number of words involved in the transfer) is in progress.

A sample program is shown below. It is assumed that two $\overline{CE}$ lines are generated on Port 0 Bits 0 and 1. The program is written to handle either multi-word Reads or Writes of successive MDL addresses. The data to be written to or the data read from the MDL device occupies successive words in the scratchpad. Prior to execution, the following initialization is assumed. The serial port is in the transmit mode (word length 16) and is sending all ones. A correct command word is placed in scratchpad register 1 (R1). From this CW the starting MDL address is derived. The starting address of the data block in the scratchpad is placed in register 2 (R2). The number of words to be transferred is placed in scratchpad register 3. Scratchpad register 4 (R4) is loaded with any 2's complement negative number for a write operation or is loaded with '04' Hex for a read. Registers 5 and 6 as well as register 9 are used to save the contents of the accumulator, IS, and status register when the serial port interrupt causes execution of the main program to stop and execution of this interrupt handling routing to begin. It is also assumed that a copy of the data last written to Port 0 is in register 7.

After setting up the appropriate registers, the main program should handle bringing the correct $\overline{CE}$ line low, then serial port interrupts should be enabled. After that, the main program can continue with another desired task. The transfer will proceed under interrupt control with execution of the main program being suspended from time to time to allow this program to handle the data being read or written. After the transfer of the desired number of words is complete, this program will de-activate the activated $\overline{CE}$ line and also update the copy of Port 0 data in R7 to reflect this. The main program can use this as a flag to determine when the transfer is done.

MK3870 INTERRUPT DRIVEN
MDL TRANSFERS

FUNCTION:
THIS ROUTINE WILL PERFORM MULTI-WORD READS
OR WRITES OF SUCCESSIVE MDL ADDRESSES.

ENTRY STATUS:
THE DATA TO BE WRITTEN TO OR THE DATA READ
FROM THE MDL DEVICE OCCUPIES SUCCESSIVE
BYTES IN THE SCRATCHPAD. THE SERIAL PORT
HAS BEEN INITIALIZED IN THE TRANSMIT MODE
WITH A 16 BIT WORD LENGTH.
THE FOLLOWING REGISTERS CONTAIN:

REGISTER	CONTENTS
1	CORRECT COMMAND WORD
2	POINTER TO DATA BLOCK
3	NUMBER OF WORDS
4 (WRITE)	2'S COMPLIMENT NEG#
4 (READ)	'04'HEX

ALSO, THE MAIN PROGRAM IS EQUIPPED TO HANDLE
CHIP ENABLES.

EXIT STATUS:
AN ENTIRE DATA BLOCK IS READ FROM AN MDL
DEVICE OR WRITTEN TO AN MDL DEVICE.

HARDWARE DEFINITIONS:
SERIAL PORT SRCLK IS THE MDL SCLK
SERIAL PORT SO AND SI ARE USED FOR THE MDL SIO

SCRATCHPAD DEFINITIONS:

```
          = 0000    41 CW       EQU   1          ;COMMAND WORD
          = 0002    42 POINT    EQU   2          ;DATA POINTER
          = 0003    43 NOWORD   EQU   3          ;# OF DATA WORDS
          = 0004    44 RD - WR  EQU   4          ;READ/WRITE REG
          = 0005    45 SAVE -1  EQU   5          ;ACCUMULATOR SAVE
          = 0006    46 SAVE - 2 EQU   6          ;ISAR SAVE
          = 0007    47 PT0IMG   EQU   7          ;PORT 0 IMAGE

0000 55             50          LR    SAVE 1, A  ;SAVE ACC
0001 0A             51          LR    A, IS      ;GET IS
0002 56             52          LR    SAVE 2, A  ;SAVE IS
0003 1E             53          LR    J, W       ;STORE STATUS REG
0004 41             54          LR    A, CW      ;GET COMMAND WORD
0005 BF             55          OUTS  0FH        ;PUT IN XMIT BUF
0006 2404           56          AI    04H        ;INCREMENT ADDRESS
                                                        ;IN CW
```

```
LOC OBJ. CODE          STMT-NR SOURCE-STMT PASS2 PAUL  PAUL PAUL  REL.

0008 51                58          LR     CW, A          ;STORE NEW CW
0009 70                59          CLR                   ;CLEAR ACC
000A C4                60          AS     RD-WR          ;ADD RD-WR TO ACC
                                                                ;ACC NOW = TO RD-WR
                                                                ;BUT STATUS IS SET.
000B 9113              63          BM     WRITE          ;IF NEG GO TO WRITE
000D'12                64 READ     SR     1              ;SHIFT RIGHT
000F'9407              65          RNZ    SKIP           ;ON FIRST AND SECOND
                                                                ;TIMES THROUGH, RE-
                                                                ;SULT OF SHIFT WILL
                                                                ;BE '02' & '01' SO
                                                                ;WILL SKIP READING
0010 42                70          LR     A, POINT       ;GET POINTER
0011 0B                71          LR     IS,A           ;PUT IN IS
0012 1F                72          INC                   ;INC POINTER
0013 52                73          LR     POINT, A       ;STORE NEW POINTER
0014 AF                74          INS    0EH            ;READ DATA
0015 5C                75          LR     S,A            ;PUT DATA IN REG
                                                                ;POINTED TO BY IS
0016'33                77 SKIP     DS     NOWORD         ;DECREMENT WORD#
0017 8120              78          BP     RESTORE        ;IF NEG RETURN
0019 43                79          LR     A,NOWORD       ;GET WORD COUNT
                                                                ;IT MUST BE FF OR FE
001A 1F                81          INC                   ;INC, ACC MUST BE
                                                                ;00 OR FF
001B 9410              83          BNZ    END            ;IF ACC IS FF BRANCH
001D 9017              84          BR     PREEND         ;IF ACC WAS NOT 00
                                                                ;THEN POINTER WAS FF
                                                                ;THUS LAST CW IS NOW
                                                                ;IN SHIFTER. CW NOW
                                                                ;IN XMIT BUFFER IS
                                                                ;BOGUS.
001F'42                90 WRITE    LR     A,POINT        ;GET POINTER
0020 0B                91          LR     IS,A           ;STORE POINTER
0021 1F                92          INC                   ;INC POINTER
0022 52                93          LR     POINT,A        ;STORE N POINTER
0023 4C                94          LR     A,S            ;GET DATA POINTED
                                                                ;TO BY IS
0024 BE                96          OUTS   0EH            ;DATA TO XMIT BUF
0025 33                97          DS     NOWORD         ;DEC WORD COUNT
0026 8111              98          BP     RESTORE        ;IF POS, OR 0 RET
0028 43                99          LR     A,NOWORD       ;GET WORD COUNT
0029 1F                100         INC                   ;IF 'FF' NOW '00'
002A 840A              101         BZ     PREEND         ;IF POINTER WAS
                                                                ;'FF' GO TO PREEND
                                                                ;LAST WORD IN SFTR
002C'47                104 END     LR     A,PTOIMG       ;WHEN SHIFTER IS
                                                                    ;SENDING 'FF' IT
                                                                ;SAFE TO
                                                                ;TURN OFF CE's
                                                                ;AND END TRANSFER
                                                                ;GET PORT DATA
002D 21FC              110         NI     0FCH           ;MAKE 0 & 1 BITS = 0
002F B0                111         OUTS   0              ;DISABLES CE's
0030 57                112         LR     PTOIMG,A       ;STORE PORT COPY
0031 2076              113         LI     76H            ;DATA FOR SERIAL
                                                                ;PORT CONTROL REG.
```

```
LOC OBJ. CODE          STMT-NR SOURCE-STMT PASS2 PAUL  PAUL PAUL   REL.

0033 BD                115             OUTS  0DH          ;DISABLE INT. ONLY
0034 70                116             CLR                ;CLEAR ACC
0035'18                117     PREEND  COM                ;ACC TO 'FF'
0036 BE                118             OUTS  0EH          ;LOAD XMIT BUFFER
0037 BF                119             OUTS  0FH          ;WITH 'FFFF'
0038'46                120     RESTORE LR    A,SAVE 2     ;PREPARE TO RETURN
                                                                 ;TO MAIN PROGRAM
                                                                 ;RESTORE ISAR
0039 0B                123             LR    IS,A
003A 1D                124             LR    W,J          ;RESTORE STATUS
003B 45                125             LR    A,SAVE 1     ;RESTORE ACC
003C 1B                126             EI                 ;ENABLE INTRPTS
003D 1C                127             POP                ;RETURN
```

IV

The flow for a Read transfer is as follows:

1st Interrupt

When the first interrupt occurs, the serial port is shifting out all ones and all ones are in the XMIT buffer. Any data in the receive buffer is not meaningful. The first CW is placed in the XMIT buffer and the address field in the CW register is incremented to make it the correct next CW. The word counter is decremented. Control is returned to the main program.

2nd Interrupt

The serial port is now shifting out the first CW. There is still no valid data in the receive buffer. The next CW is placed in the XMIT buffer and the CW register is updated. The word counter is decremented. Control is returned.

3rd-Nth Interrupts

The first through N-2 commands have been sent. The data in the receive buffer is valid data and is stored in the scratchpad with the pointer being incremented each time. The N-1 CW is in the shifter and the Nth CW is placed in the XMIT buffer. The word counter is decremented to initial value-N. Control is returned to the main program.

N + 1 Interrupt

For a transfer of N words, the N = 1 interrupt will occur when CWn-1 clears the shifter and CWn is moved to the shifter from the XMIT buffer. The receive buffer will contain DWn-1. It is read and placed in the scratchpad. The scratchpad pointer is incremented to initial value + N-1. The word counter is decremented from '00' to 'FF'. Before control is returned to the main program, the XMIT buffer is loaded with all ones.

N + 2 Interrupt

The last CW has now shifted out and the shifter is now sending all ones. The last DW is in the receive buffer so it is moved to the scratchpad. The serial port interrupt is disabled and $\overline{CE}$ is deactivated. Control is then returned to the main program.

The flow of this program for a Write transfer is as follows.

1st Interrupt

When the first interrupt occurs, the serial port is shifting out all ones and also has all ones in its transmit buffer. The first CW is loaded into the lower XMIT buffer and the CW in R1 is incremented to point to the next MDL address. The first DW is read from the address in R2 and loaded into the upper XMIT buffer. The address in R2 is incremented. The word count in R3 is decremented. Control is returned to the main program.

2nd-Nth Interrupt

When each successive interrupt occurs, the serial port has just moved the previous (N-1) CW and DW into the shifter. A new (nth) CW is read from R1 and placed in the XMIT buffer along with the Nth data word. The pointer is incremented to value = N and the word counter is now at words - N.

N + 1 Interrupt

Assuming N words are to be written, when the N + 1 interrupt occurs, the last CW and DW are now in the shifter (but not yet completely out the door!) Initially a false CW and DW (the N + 1 words) are loaded into XMIT buffer but are corrected to be all ones before returning. The pointer is now at value + N + 1 and the word counter is at 'FF'.

N + 2 Interrupt

At this interrupt, the last CW and DW have cleared the shifter. The shifter is now shifting out all ones. At this point the $\overline{CE}$ line is deactivated, the interrupt is turned off and the serial port is left in a mode to continue to shift out all ones indefinitely.

In general, this routine must execute within one word time. In addition, there must be time for the main program to execute at least one instruction and time for the interrupt acknowledge sequence. In fact, there must be time for the main program to execute its longest sequence of protected instructions plus the next instruction. Thus, if there are, for example, two protected instructions in a row, there must be time for both of these to execute plus time for the next instruction to execute. About 15 μs should be allowed for the interrupt acknowledge. The time allowed for main program execution depends upon the code to be executed but would probably be in the 10 to 30 μs range or longer if subroutine calls or other interrupts are involved. The worst case pass through the MDL routine shown above is 133 μs for the last read interrupt and 128 μs for the last write interrupt, but it is not necessary that these two passes execute within a word time because no subsequenally serial port interrupt will be immediately encountered.

The actual time constraining pass through the MDL routine is the next to last pass for the Read or Write which (with a 4 MHz time base) are 106 μs and 105 μs, respectively. Thus, word time of the serial port must exceed this 106 μs plus the near 15 μs interrupt acknowledge plus the worst case execution time within the main program wherein acknowledging the serial port interrupt is not allowed. This total time might, for example, be 160 μs. Thus the word time must equal or exceed this 160 μs. At the maximum internally generated Baud rate (with a 4 MHz time base) the word time is 96 μs. Thus, this general purpose routine could not function properly. However, at one half the maximum Baud rate it would probably function. At this Baud rate (83.3 K Baud with a 4 MHz time base) the word time would be 192 μs. The average execution time per interrupt for a 16 word transfer would be 82.3 μs for a Write or 84.5 μs for a Read. In either case, 18 interrupts (N = 2) must be handled to accomplish a 16 (N) word transfer. When the 15 μs interrupt acknowledge time is added, this brings the average interrupt handling time to roughly 100 μs. With 192 μs between interrupts, this means that about 52% of the time would be spent in handling interrupts (thus producing an MDL transfer) and 48% of the time is used for execution of the main program. During the 18 word times required for the 16 word transfer, processor throughput is reduced to 48% of normal, but execution of other tasks can still proceed though at a reduced rate.

Note that the routine presented ways very general in nature. For a given specific system, its execution time could be reduced. Thus, it could be possible to operate the serial port in the interrupt driven environment even at the higher Baud rate and still allow some throughput of the main program. An example of a specific case might be a read or write only routine. For a device like an MDL display driver, reading any registers might not occur at all. Write operations could be restricted to perhaps two or three specific registers (display data) for this device in normal execution. Thus, a lot of the instructions which manipulate loop counts, pointers, etc. could be eliminated as well as the testing done to differentiate whether a read operation or a write operation is to occur.

1982/1983 Z80 DESIGNERS GUIDE

MOSTEK®

ADD SERIAL COMMUNICATION CAPABILITY TO THE 8086/8088 FAMILY USING THE Z80 SIO

Application Note

INTRODUCTION

Since its introduction, the SIO (MK3884/5/7) has been widely recognized as one of the most powerful serial communications devices commercially available. The SIO is a dual-channel, multifunction peripheral device capable of handling a wide variety of serial data communications requirements in microcomputer systems. The system designer can configure the SIO for the personality of almost any system, as each channel is software programmable. The communications' power of the SIO does not need to be limited to those applications where a Z80 CPU is being used. To explain more fully how it can be used with other processors is the basis of this application note. The information presented here can be used to adapt the use of the SIO not only to the 8086/8088 family but also to other microprocessors. All references made to the 8086 hereafter apply also to the 8088, as the hardware and software implementation is identical for each.

DESCRIPTION

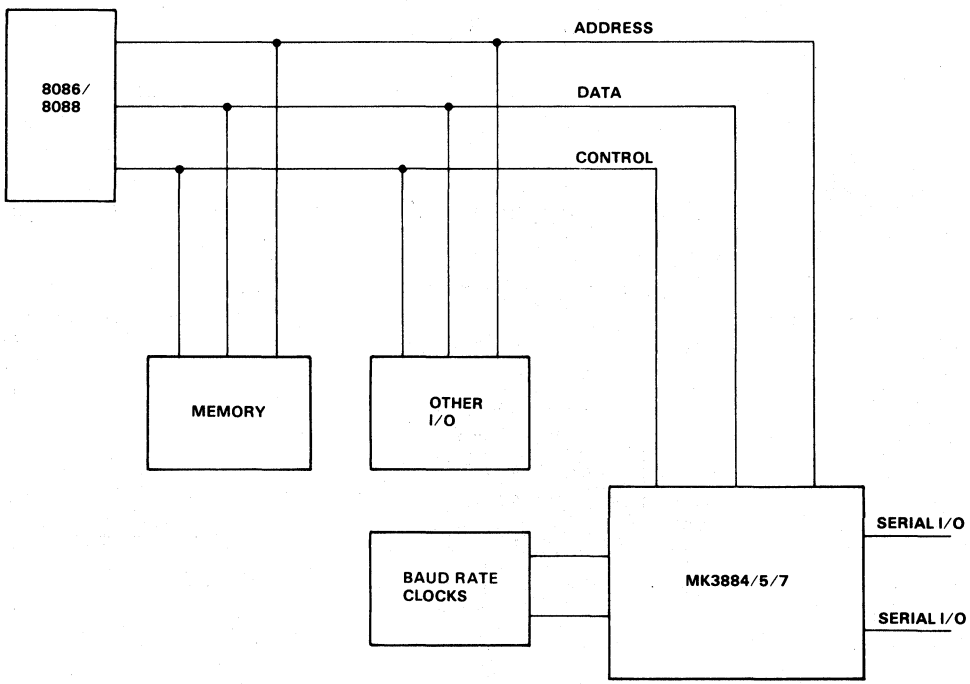
The MK3884/5/7 (SIO) is a dual full duplex USART device that includes special logic to allow it to handle special purpose serial protocols such as SDLC and HDLC. Three bonding options are available to the user, allowing alternate signals to be brought out to some pins for a particular application. For example, in some cases the receive and transmit clocks for the second channel are both needed because they will be at different frequencies. A user desiring this can use the MK3887 but the $\overline{\text{SYNC}}$ signal must be sacrificed. Other alternative signal selections are brought out on the MK3884 and MK3885. This flexibility and the fully dual nature of the SIO give it particular appeal in designing serial communications equipment.

The system that will be considered is a very simplified 8086 based minimum mode system which will use the SIO as a serial communications port interfaced to a standard ASCII terminal. It is assumed that the device will be memory mapped to avoid the use of the limited I/O instructions of

V

SYSTEM BLOCK DIAGRAM

Figure 1



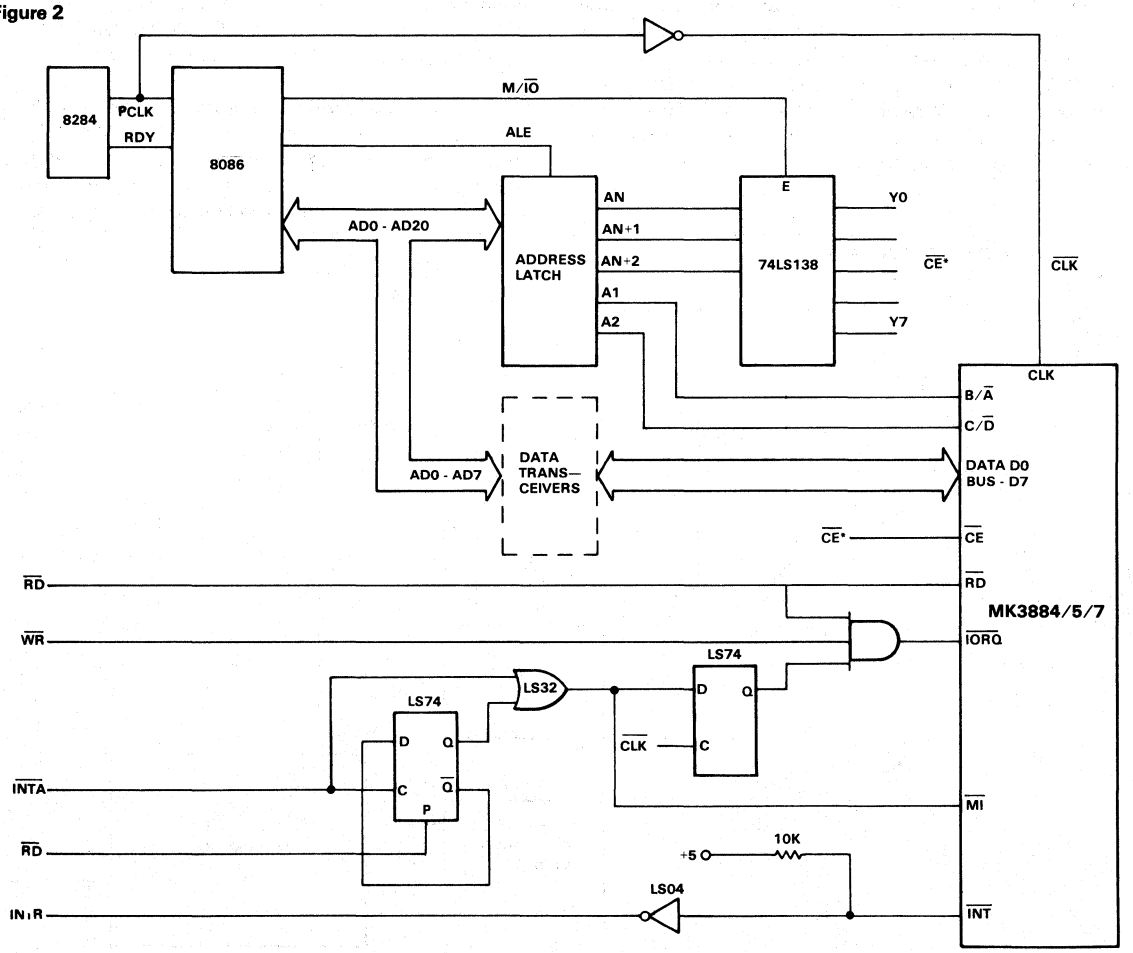
the 8086 family. Interrupts will be used for the receive side of channel A to signal the processor that another character has been received. Figure 1 shows a simplified block diagram of the system.
There are several control signals used by the SIO which are peculiar to the Z80 family of devices and, as such, must be given special consideration in using this device with another processor.

1. $\overline{A}$/B and C/$\overline{D}$ (inputs)
2. $\overline{CE}$ (input)
3. $\overline{M1}$ (input-needed for interrupts)
4. $\overline{IORQ}$ (input)
5. $\overline{RD}$ (input)
6. $\overline{INT}$ (output)

The most readily obtainable signals are $\overline{A}$/B and C/$\overline{D}$ corresponding to A1 and A2 of the address bus, respectively. Note that the address bus must first be latched by the ALE signal from the 8086. (NOTE: A0 is not an address but a bank select signal.) For the sake of programming, we will consider the SIO as a memory mapped device as suggested earlier. We will also want to use the interrupt capability of the device and that will be covered later. Since the address signals are latched out of the processor first, there is not a timing problem with the interface to the SIO if the chip enable signal is decoded well before any data is available. The $\overline{CE}$ signal is derived from address decoding the upper address bits. This can be done using a BIPOLAR PROM or a decoder such as the 74LS138 as shown in Figure 2. It is important to note that the $\overline{CE}$ signal will be valid only during a memory cycle through the use of the M/$\overline{IO}$ line from the 8086. In a maximum 8086 system, this must be done with the signals from the Bus Controller Device.

$\overline{M1}$ is used by the SIO only during the interrupt acknowledge cycle and the interrupt return cycle. To handle interrupts with the 8086, the $\overline{INT}$ signal is inverted between the SIO and the processor to form the INTR signal with its logic true high. When the 8086 detects an interrupt, it issues an $\overline{INTA}$ (interrupt acknowledge) signal twice: first in conjunction with LOCK to lock out any possible outside bus requests and then a second time to initiate the interrupt

8086 MINIMUM SYSTEM USING THE MK3884/5/7
Figure 2



vector response. This second $\overline{INTA}$ is detected by excluding the first $\overline{INTA}$ using a toggle flip-flop which is then applied to the SIO as $\overline{M1}$. This $\overline{M1}$ is delayed using a flip-flop and gated with $\overline{CE}$ to form the necessary $\overline{IORQ}$ signal. The combination of the $\overline{M1}$ followed by the $\overline{IORQ}$ signals the SIO that an interrupt acknowledge cycle is being performed. At that point the SIO will place its programmed interrupt vector on the data bus for use by the processor. At the end of the interrupt service the SIO would normally be reset by the detection of the two byte opcode ED and 4D. With a non-Z80 processor, the SIO can be reset by the software interrupt reset command. This will be discussed later.

The $\overline{IORQ}$ signal is used to signal the SIO that an I/O instruction is being performed. Since we have assumed that the device will be memory mapped, the $\overline{IORQ}$ signal can be simply the $\overline{RD}$ or $\overline{WR}$ signal gated with the delayed $\overline{M1}$ signal discussed previously. If the $\overline{CE}$ signal is not fully decoded, a problem can occur if other devices are similarly accessed. Therefore, it is suggested that full decoding be used to allow unrestricted use of the remainder of the memory space.

The $\overline{RD}$ (read) signal comes directly from the 8086 and is applied to the SIO as such. This is to ensure that the device timing is maintained.

The pull-up resistor and inverter on the $\overline{INT}$ (interrupt) line are necessary to ensure that the INTR signal to the 8086 is logic false (low) for all conditions except an actual interrupt where the SIO pulls $\overline{INT}$ low. If more than one SIO were being used, the $\overline{INT}$ signals from all the devices would be wire-ored together before going to the inverter.

In most designs the introduction of the logic delays in the control signals will not adversely affect the timing necessary for proper operation of the SIO with the 8086. However, if the SIO is to be on a peripheral card or isolated from the processor by data, address, and control buffers, then the timing interrelations for the SIO as called out on the data sheets must be carefully examined to insure that proper set up and delay times are maintained. This is especially critical in the interrupt acknowledge cycle where the SIO is attempting to pass a vector back to the processor. Some caution, therefore, must be taken in respect to buffer direction & tri-state control.

SOFTWARE

Software needed to drive the SIO is most easily implemented as a block of command and data bytes that are transferred to the SIO using a memory block move type of operation. This is the primary reason for memory mapping the device. The actual programming of the SIO will be covered in another application note, but it is important to point out that the sequence of initialization commands and control commands for interrupt servicing is critical. Interrupt service routines are similar to other 8086 type services with the exception that the service routine for the SIO must have a Write Register 0, Command 7, as its last instruction to reset the internal interrupt hardware of the SIO so that another interrupt can be generated.

OPERATION

The maximum clock frequency of the SIO is 4 MHz, and for test purposes the clock speed was 3 MHz. This clock must be synchronized to the clock of the 8086 and inverted to ensure proper data transfers. This will limit the 8086 to a clock of 4 MHz, unless special speed-up/slow-down hardware is designed to synchronize the data transfers when differing clock rates are used. Under normal operation of the SIO and 8086 at 4 MHz, the SIO will not have a problem keeping pace with the processor because of its ability to transfer at very high data rates. Caution should be taken if the SIO clock is not 50% duty cycle to ensure that the clock high (min.) and clock low (min.) signals are not violated. If synchronous data transfers at very high clock rates are to be done with the SIO, it may be necessary to use a direct memory access controller to load the transmit buffer of the SIO to ensure that it does not become prematurely empty. If the transmit buffer does become empty before the end of the transmission, the block of data being transmitted may be terminated before all the data has been transferred. Successful operation of the SIO in very high speed systems has demonstrated that the SIO is capable of the most sophisticated data transfers with the least interface to the processor of a system.

PROGRAM EXAMPLE

Table 1 illustrates an echo program showing the initialization and transfer of data between the 8086 CPU, the Z80 SIO, and a terminal. Proper initialization includes first resetting B and loading the interrupt vector '10'H to the SIO. In the event of an interrupt, '10'H will be read by the 8086, multiplied by 4 internally, and will hence provide an interrupt look up table located at '40'H as indicated in Table 1. As shown, channel A is configured for data communications at 9600 baud, no parity, one stop bit, and 8 bits / character. Reference should be made to the Mostek MK3884 technical manual for proper SIO initialization procedures. Worthy of note are the command strings incorporated for SIO initialization. This technique is analogous to the efficient Z80 OTIR instruction. Also of interest and necessity is the software return from interrupt. When the Z80 SIO is in an interrupt driven environment, it must see an RETI instruction (ED 4D) on its data bus in order to reset the internal interrupt logic. In non-Z80 CPU environments, however, the interrupt reset may be effected by writing a '38'H to the appropriate command/status register of the SIO. Program execution loops until interrupted by the data terminal, causing an interrupt to the starting address, '002B'H in this example.

Table 1

SIO86

```
LINE    SOURCE

1        ************************************************************
2       * THIS PROCEDURE (SUBROUTINE) ASSUMES THE SEGMENT REGISTERS *
3       * AND STACK POINTER ARE SET UP.                             *
4       ************************************************************
5       INT_VECTS       SEGMENT AT 0
6                       ORG     40 H
7                       DD      ECHO
8       INT_VECTS       ENDS
9       ;
10      ;
11      CODENAME                SEGMENT
12                      ASSUME  CS:CODENAME,DS:CODENAME, SS:CODENAME,ES:CODENAME
13      ;
14      ;   THE SIO IS MEMORY MAPPED AT LOCATIONS 1000H-1006H
15      ;
16      SIO_ADATA       EQU     1000H               ;ADDRESS OF SIO CHANNEL A DATA
17      SIO_BDATA       EQU     SIO_ADATA+2         ;ADDRESS OF SIO CHANNEL B DATA
18      SIO_ACS         EQU     SIO_ADATA+4         ;ADDRESS OF SIO CHANNEL A
19      ;                                                COMMAND/STATUS
20      SIO_BCS         EQU     SIO_ADATA+6         ;ADDRESS OF SIO CHANNEL B
21      ;                                                COMMAND/STATUS
22      ;
23      ;  B COMMAND STRING TO INITIALIZE CHANNEL B
24      ;
25      B_COMM          DB      18,02H,10H


26      ;               18H     CHANNEL B RESET,WRITE REGISTER 0
27      ;               02H     SET POINTER TO WRITE REGISTER 2
28      ;               10H     SIO INTERRUPT VECTOR
29      ;
30      ;  A COMMAND STRING TO INITIALIZE CHANNEL A
31      ;
32      A_COM           DB      18H,04H,44H,01H,18H,03H,0C1H,05H,68H



33      ;               18H     CHANNEL A RESET,WRITE REGISTER 0
34      ;               04H     SET POINTER TO WRITE REGISTER 4
35      ;               44H     16X CLOCK MODE,1 STOP BIT,NO PARITY
36      ;               01H     SET POINTER TO WRITE REGISTER 1
37      ;               18H     INTERRUPT ON ALL Rx CHARACTERS
38      ;               03H     SET POINTER TO WRITE REGISTER 3
39      ;               C1H     Rx 8 BITS/CHARACTER,Rx ENABLE
```

```
LINE    SOURCE

40      ;               05H     SET POINTER TO WRITE REGISTER 5
41      ;               68H     Tx 8 BITS/CHARACTER,Tx ENABLE
42      ;
43      ;   START OF SIO INITIALIZATION PROCEDURE
44      ;
45      SIO_INIT        PROC    NEAR
46      START:          MOV     BX,SIO_BCS          ;CHANNEL B COMMAND ADDRESS
47                      MOV     SI,OFFSET B_COM     ;ADDRESS OF COMMAND STRING
48                      MOV     CX,LENGTH B_COM     ;STRING LENGTH IN COUNT REGISTER
49                      CLD                         ;CLEAR DIRECTION FLAG,AUTO-INC.
50      CBLD:           LODS    B_COM               ;    CHANNEL B
51                      MOV     [BX],AL             ;    INITIALIZATION
52                      LOOP    CBLD                ;    LOOP UNTIL CX=0
53                      MOV     BX,SIO_ACS          ;CHANNEL A COMMAND ADDRESS
54                      MOV     SI,OFFSET A_COM     ;ADDRESS OF COMMAND STRING
55                      MOV     CX,LENGTH A_COM     ;STRING LENGTH IN COUNT REGISTER
56      CALD:           LODS    A_COM               ;    CHANNEL A
57                      MOV     [BX],AL             ;    INITIALIZATION
58                      LOOP    CALD                ;    LOOP UNTIL CX=0
59                      STI                         ;SET INTERRUPT FLAG
60                      RET                         ;RETURN TO CALLING PROGRAM
61      SIO_INIT        ENDP
62      ;
63      ;   END OF INITIALIZATION OF SIO
64      ;
65      ;   INTERRUPT SERVICE ROUTINE TO ECHO BACK CHARACTERS TO THE TERMINAL
66      ;
67      SIO_ISR         PROC    NEAR
68      ECHO:           MOV     BX,SIO_ADATA        ;CHANNEL A DATA ADDRESS
69                      MOV     AL,[BX]             ;GET CHARACTER
70                      MOV     [BX],AL             ;ECHO TO TERMINAL
71                      MOV     BX,SIO_ACS          ;CHANNEL A C/S ADDRESS
72                      MOV     BYTE PTR [BX],38H          ;SOFTWARE RETURN
73      ;                                                   FROM INTERRUPT
74                      IRET                        ;RETURN TO INTERRUPTED PROGRAM
75      SIO_ISR         ENDP
76      ;
77      ;   END OF INTERRUPT SERVICE ROUTINE, CHARACTER RECEIVED
78      ;   FROM TERMINAL IN AL REGISTER.
79      ;
80      CODENAME                ENDS
81                      END
```

V

MOSTEK®

Z80 INTERFACING TECHNIQUES FOR DYNAMIC RAM

Application Note

INTRODUCTION

Since the introduction of second generation microprocessors, there has been a steady increase in the need for larger RAM memory for microcomputer systems. This need for larger RAM memory is due in part to the availability of higher level languages such as PL/M, PL/Z, FORTRAN, BASIC, and COBOL. Until now, when faced with the need to add memory to a microcomputer system, most designers have chosen static memories such as the 2102 1Kx1 or possibly one of the new 4Kx1 static memories. However, as most mini or mainframe memory designers have learned, 16-pin dynamic memories are often the best overall choice for reliability, low power, performance, and board density. This same philosophy is true for a microcomputer system. Why then have microcomputer designers been reluctant to use dynamic memory in their system? The most important reason is that second generation microprocessors such as the 8080 and 6800 do not provide the necessary signals to interface dynamic memories easily into a microcomputer system.

Today, with the introduction of the Z80, a true third generation microprocessor, not only can a microcomputer designer increase system throughput by the use of more powerful instructions, but he can also easily interface either static or dynamic memories into the microcomputer system. This application note provides specific examples of how to interface 16-pin dynamic memories to the Z80.

OPERATION OF 16-PIN DYNAMIC MEMORIES

The 16-pin dynamic memory concept, pioneered by MOSTEK, uses a unique address multiplexing technique which allows memories as large as 16, 384 bits x 1 to be packaged in a 16-pin package. For example the MK4027 (4,096x1 dynamic MOS RAM) and the MK4116 (16,384x1 dynamic MOS RAM) both use address multiplexing to load the address bits into memory. The MK4027 needs 12 address bits to select 1 out of 4,096 locations, while the MK4116 requires 14 bits to select 1 out of 16,384. The internal memories of the MK4027 and MK4116 can be thought of as a matrix. The MK4027 matrix can be thought of as 64x64, and the MK4116 as 128x128. To select a particular location, a row and column address is supplied to the memory. For the MK4027, address bits A_0-A_5 are the row address, and bits A_6-A_{11} are the column addresses. For the MK4116, address bits A_0-A_6 are the row address, and A_7-A_{13} are the column address. The row and column addresses are strobed into the memory by two negative going clocks called Row Address Strobe ($\overline{RAS}$) and Column Address Strobe ($\overline{CAS}$). By the use of $\overline{RAS}$ and $\overline{CAS}$, the address bits are latched into the memory for access to the desired memory location.

Dynamic memories store their data in the form of a charge on a small capacitor. In order for the dynamic memory to retain valid data, this charge must be periodically restored. The process by which data is restored in a dynamic memory is known as refreshing. A refresh cycle is performed on a row of data each time a read or write cycle is performed on any bit within the given row. A row consists of 64 locations for the MK4027 and 128 locations for the MK4116. The refresh period for the MK4027 and the MK4116 is 2ms which means that the memory will retain a row of data for 2ms without a refresh. Therefore, to refresh all rows within 2ms, a refresh cycle must be executed every 32μs (2ms÷64) for the MK4027, and 16μs (2ms÷ 128) for the MK4116.

To ensure that every row within a given memory is refreshed within the specified time, a refresh row address counter must be implemented either in external hardware or as an internal CPU function as in the Z80. (Discussed in more detail under Z80 Refresh Control and Timing.) The refresh row address counter should be incremented each time that a refresh cycle is executed. When a refresh is performed, all RAMs in the system should be loaded with the refresh row address. For the MK4027 and the MK4116, a refresh cycle consists of loading the refresh row address on the address lines and then generating a $\overline{RAS}$ for all RAMs in the system. This is known as a $\overline{RAS}$ only refresh. The row that was addressed will be refreshed in each memory. The $\overline{RAS}$ only refresh prevents a conflict between the outputs of all the RAMs by disabling the output on the MK4116, and maintaining the output state from the previous memory cycle on the MK4027.

Z80 TIMING AND MEMORY CONTROL SIGNALS

The Z80 was designed to make the job of interfacing

V

to dynamic memories easier. One of the reasons the Z80 makes dynamic memory interfacing easier is because of the number of memory control signals that are available to the designer. The Z80 control signals associated with memory operations are:

MEMORY REQUEST ($\overline{MREQ}$) - Memory request signal indicating that the address bus holds a valid memory address for a memory read, memory write, or memory refresh cycle.

READ ($\overline{RD}$) - Read signal indicating that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

WRITE ($\overline{WR}$) - Write signal indicating that the CPU data bus hold valid data to be stored in the addressed memory or I/O device.

REFRESH ($\overline{RFSH}$) - Refresh signal indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the current MREQ signal should be used to generate a refresh cycle for all dynamic memories in the system.

Figures 1a, 1b, and 1c show the timing relationships of the control signals, address bus, data bus, and system clock Φ. By using these timing diagrams, a set of equations can be derived to show the worst case access times needed for dynamic memories with the Z80 operating at 2.5MHz.

The access time needed for the op code fetch cycle and the memory read cycle can be computed by equations 1 and 2.

(1) $t_{ACCESS\ OP\ CODE} = 3(t_c/2) - t_{DL\overline{\Phi}(MR)} - t_{S\Phi(D)}$

where: t_c = Clock period

$t_{DL\overline{\Phi}(MR)}$ = $\overline{MREQ}$ delay from falling edge of clock.

$t_{s\Phi(D)}$ = Data setup time to rising edge of clock during op code fetch cycle.

let: t_C = 400ns; $t_{DL\overline{\Phi}(MR)}$ = 100ns; $t_{s\Phi}$ = 50ns

then: $t_{ACCESS\ OP\ CODE}$ = 450ns

(2) $t_{ACCESS\ MEMORY\ READ} = 4(t_c/2) - t_{DL\overline{\Phi}(MR)} - t_{S\overline{\Phi}(D)}$

where: t_C = Clock period

$t_{DL\overline{\Phi}(MR)}$ = $\overline{MREQ}$ delay from falling edge of clock

$t_{S\overline{\Phi}(D)}$ = Data Setup time to falling edge of clock

let: t_C = 400ns; $t_{DL\ (MR)}$ = 100ns; $t_{S\ (D)\ \overline{\Phi}}$ = 60ns

then: $t_{ACCESS\ MEMORY\ READ}$ = 640ns

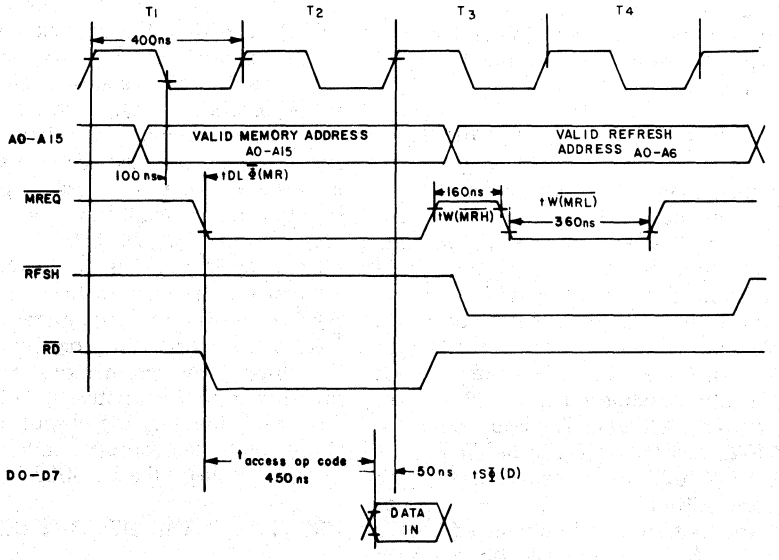
The access times computed in equations 1 and 2 are overall worst case access times required by the CPU. The overall access times must include all TTL buffer delays and the access time for the memory device. For example, a typical dynamic memory design would have the following characteristics (see Figure 2):

The example in Figure 2 shows an overall access time of 336ns. This would more than satisfy the 450ns required for the op code fetch and the 640ns required for a memory read.

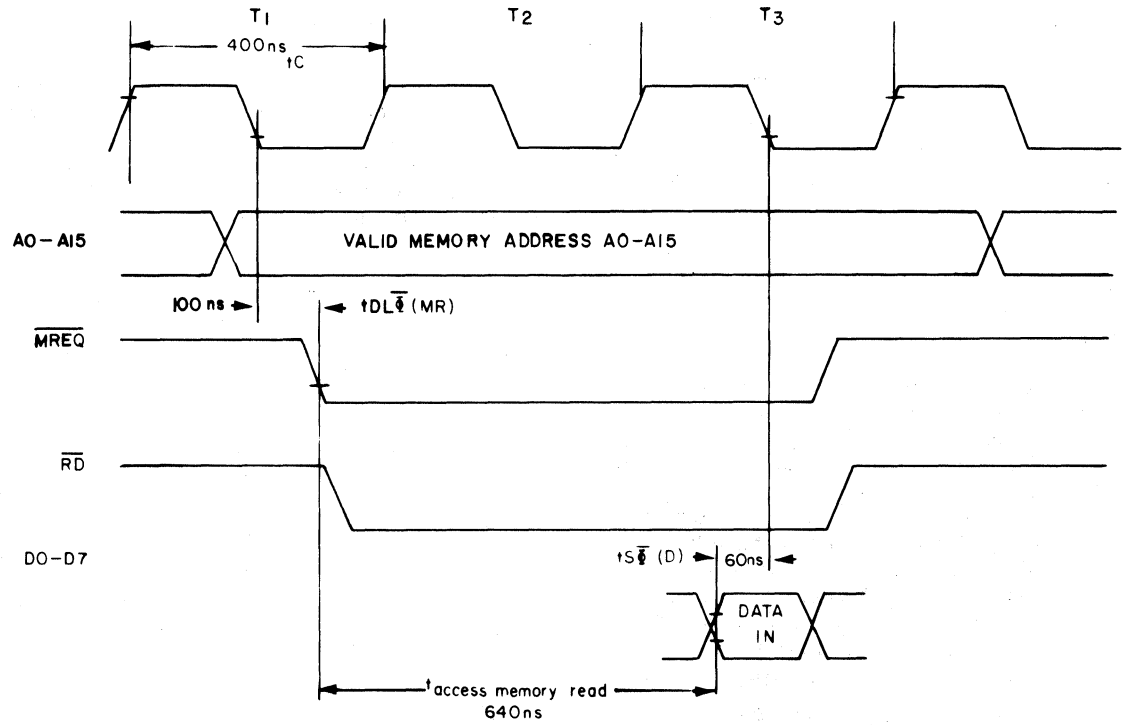
CPU $\overline{MREQ}$ buffer delay	12ns (8T97)
Memory gating and timing delays	40ns
Memory device access time	250ns (MK4027/4116-4)
Memory data bus buffer delay	17ns (8T28)
CPU data bus buffer delay	17ns (8T28)
	336ns

OP CODE FETCH TIMING

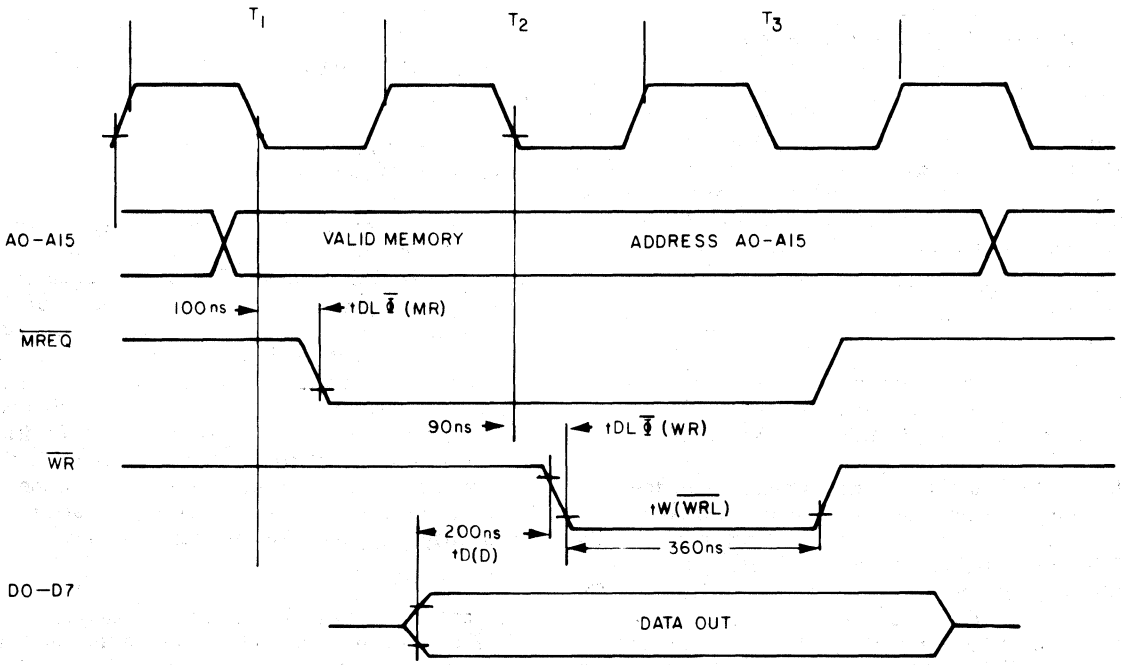
Figure 1a



MEMORY READ TIMING
Figure 1b



MEMORY WRITE TIMING
Figure 1c



V

Z80 REFRESH CONTROL AND TIMING

One of the most important features provided by the Z80 for interfacing to dynamic memories is the execution of a refresh cycle every time an op code fetch cycle is performed. By placing the refresh cycle in the op code fetch, the Z80 does not have to allocate time in the form of "wait states" or by "stretching" the clock to perform the refresh cycle. In other words, the refresh cycle is "totally transparent" to the CPU and does not decrease the system throughput (see Figure 1a). The refresh cycle is transparent to the CPU because, once the op code has been fetched from memory during states T_1 and T_2, the memory would normally be idle during states T_3 and T_4.

Therefore, by placing the refresh in the T_3 and T_4 states of the op code fetch, no time is lost for refreshing dynamic memory. The critical timing parameters involving the Z80 and dynamic memories during the refresh cycle are: $t_{W(MRH)}$ and $t_{W(MRL)}$. The parameter known as $t_{W(MRH)}$ refers to the time that $\overline{MREQ}$ is high during the op code fetch between the fetch of the op code and the refresh cycle. This time is known as "precharge" for dynamic memories and is necessary to allow certain internal nodes of the RAM to be charged-up for another memory cycle. The equation for the minimum $t_{W(MRH)}$ time period is:

(3) $t_{W(MRH)} = t_{W(\Phi H)} + t_f - 30$

where: $t_{W(\Phi H)}$ is clock pulse width high

t_f is clock fall time

let: $t_{W(\Phi H)} = 180ns$; $t_f = 10ns$

then: $t_{W(MRH)} = 160ns$ (min)

A $t_{W(MRH)}$ of 160ns is more than adequate to meet the worst case precharge times for most dynamic RAMs. For example, the MK4027-4 and the MK4116-4 require a 120ns precharge.The other refresh cycle parameter of importance to dynamic RAMs is $t_{W(MRL)}$, (the time that $\overline{MREQ}$ is low during the refresh cycle). This time is important because $\overline{MREQ}$ is used to generate $\overline{RAS}$ directly. The equation for the minimum time period is:

(4) $t_{W(MRL)} = t_c - 40$

where: t_c is the clock period

let: $t_c = 400ns$

then: $t_{W(MRL)} = 360ns$

A 360ns $t_{W(MRL)}$ exceeds the 250ns min $\overline{RAS}$ time required for the MK4027-4 and the MK4116-4.

By controlling the refresh internally with the Z80, the designer must be aware of one limitation. The limitation is that to refresh memory properly, the Z80 CPU must be able to execute op codes since the refresh cycle occurs during the op code fetch. The following conditions cause the execution of op codes to be inhibited, and will destroy the contents of dynamic memory.

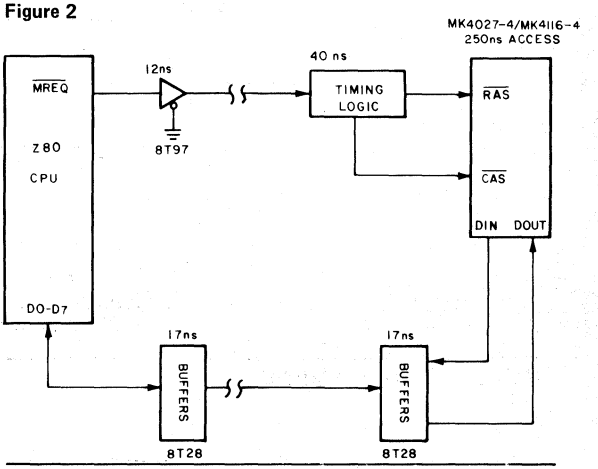
(1) Prolonged reset > 1ms
(2) Prolonged wait state operation > 1ms
(3) Prolonged bus acknowledge (DMA) > 1ms
(4) Φ clock of < 1.216 MHz for 16K RAMs
< .608 MHz for 4K RAMs

The clocks' rate in number 4 is based on the Z80 continually executing the worst case instruction which is an EX (SP), HL that executes in 19 T states. Therefore, by operating the Z80 at or above these clocks' frequencies, the user is ensured that the dynamic memories in the system will be refreshed properly.

Remember to refresh memory properly, the Z80 must be able to execute op codes!

DELAY FOR A TYPICAL MEMORY SYSTEM

Figure 2



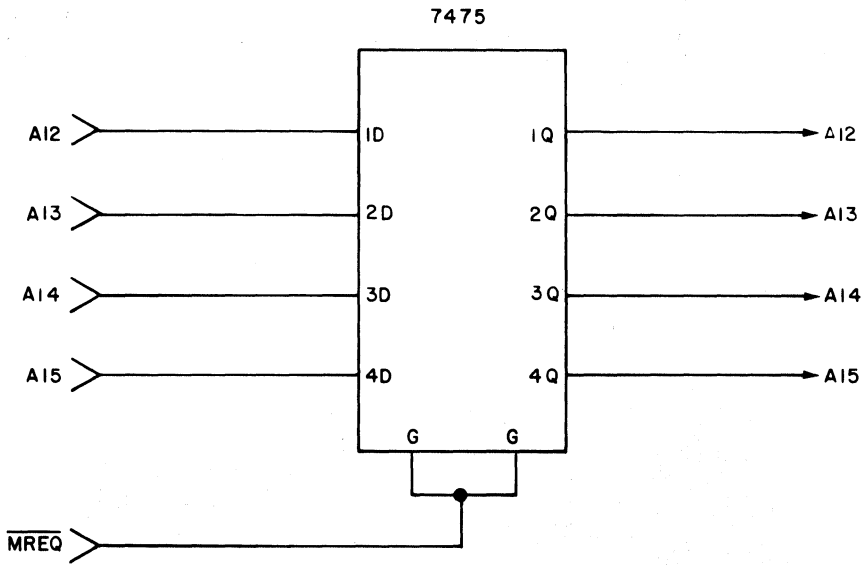
SUPPORT CIRCUITS FOR DYNAMIC MEMORY INTERFACE

Two support circuits are necessary to ensure reliable operation of dynamic memory with the Z80.

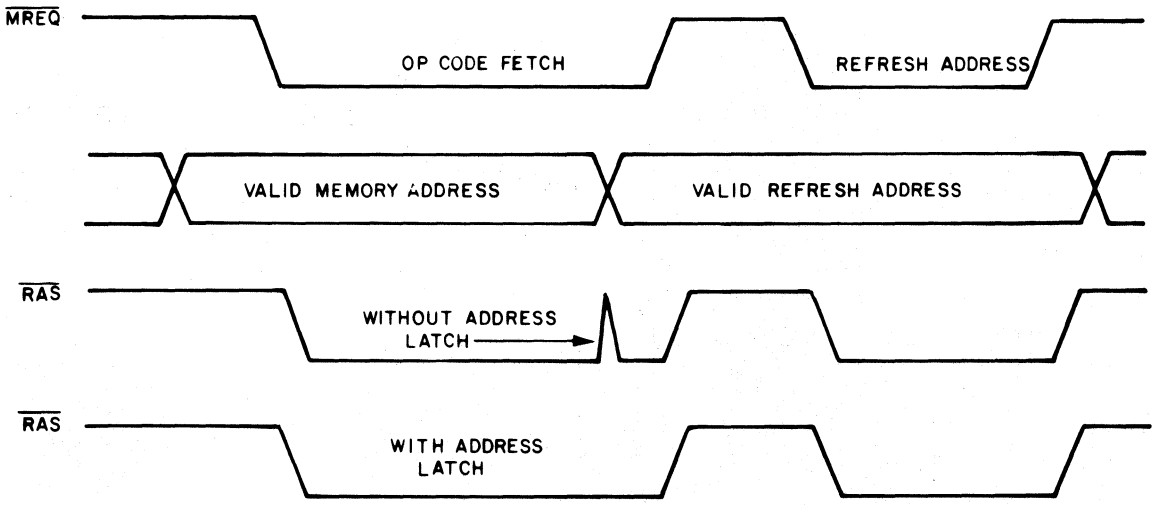
The first of these circuits is an address latch shown in Figure 3. The latch is used to hold addresses A_{12}-A_{15} while $\overline{MREQ}$ is active. This action is necessary because the Z80 does not ensure the validity of the address bus at the end of the op code fetch (see Figure 4). This action does not directly affect dynamic memories because they latch addresses internally. The problem comes from the address decoder which generates $\overline{RAS}$. If the address lines which drive the decoder are allowed to change while $\overline{MREQ}$ is low, then a "glitch" can occur on the $\overline{RAS}$ line or lines (if more than one row of RAMs are used) which may have the effect of destroying one row of data.

The second support circuit is used to generate a power on and short manual reset pulse. Recall from the discussion under Z80 Timing and Memory Con-

ADDRESS LATCH
Figure 3



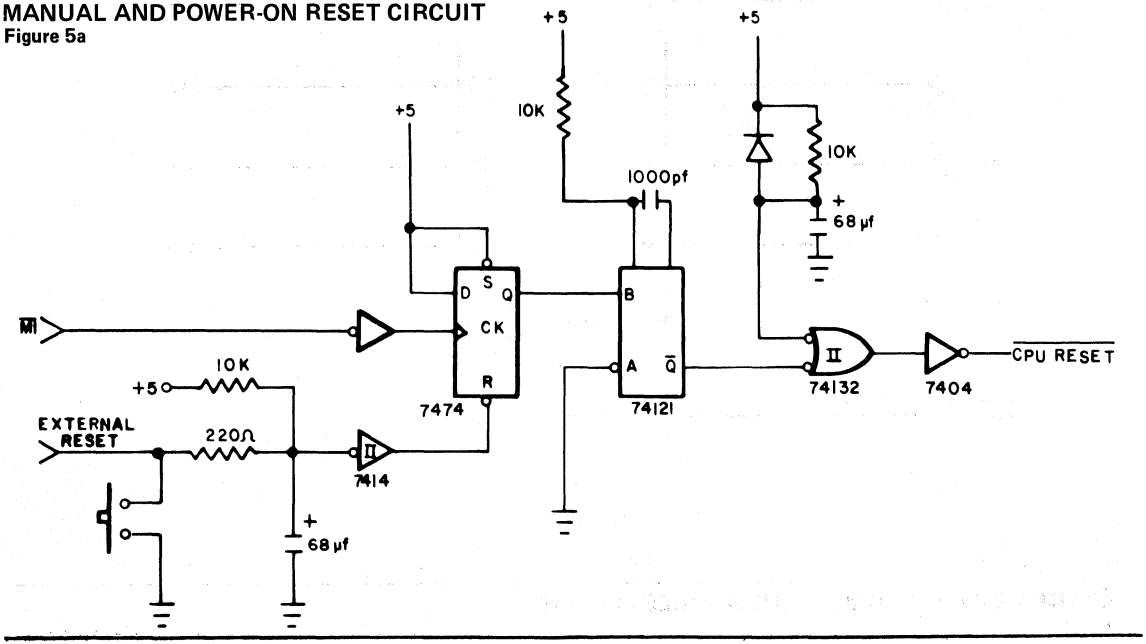
$\overline{\text{RAS}}$ TIMING WITH AND WITHOUT ADDRESS LATCH
Figure 4



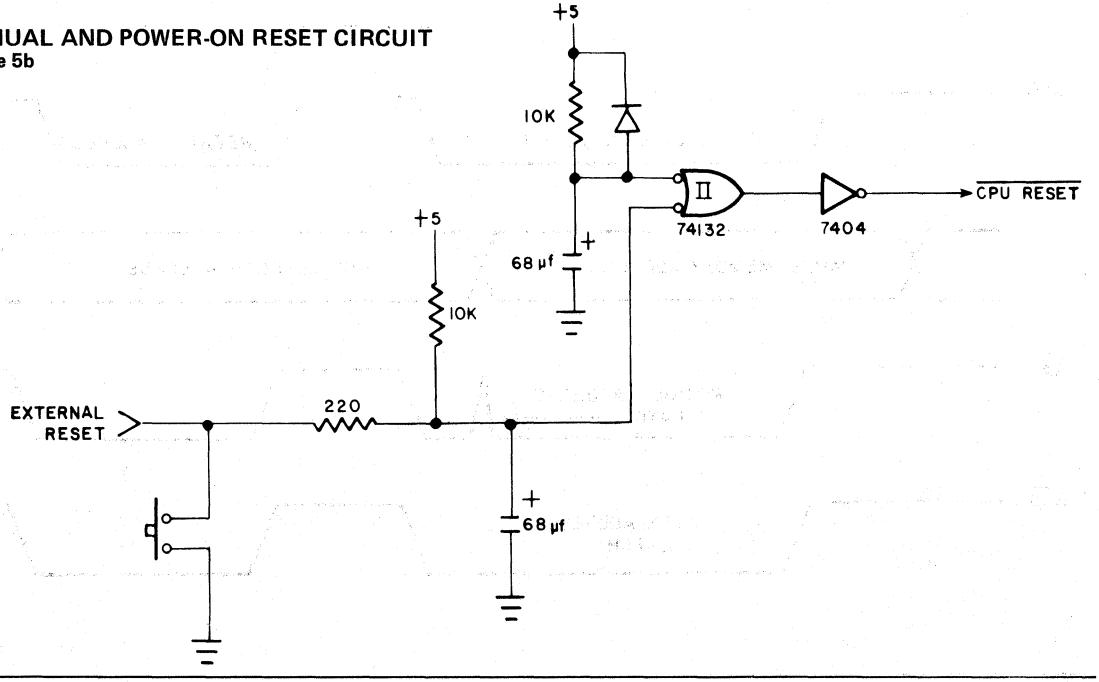
trol Signals that one of the conditions that will cause dynamic memory to be destroyed is a reset pulse of duration greater than 1ms. The circuit shown in Figure 5a can be used to generate a short reset pulse from either a push button or an external source. Additionally the manual reset is synchronized to the start of an M1 cycle so that the reset will not fall during the middle of a memory cycle. Along with the manual reset, the circuit will also generate a power on reset.

If it is not necessary that the contents of the dynamic memory be preserved, then the reset circuit shown in Figure 5b may be used to generate a manual or power on reset.

MANUAL AND POWER-ON RESET CIRCUIT
Figure 5a

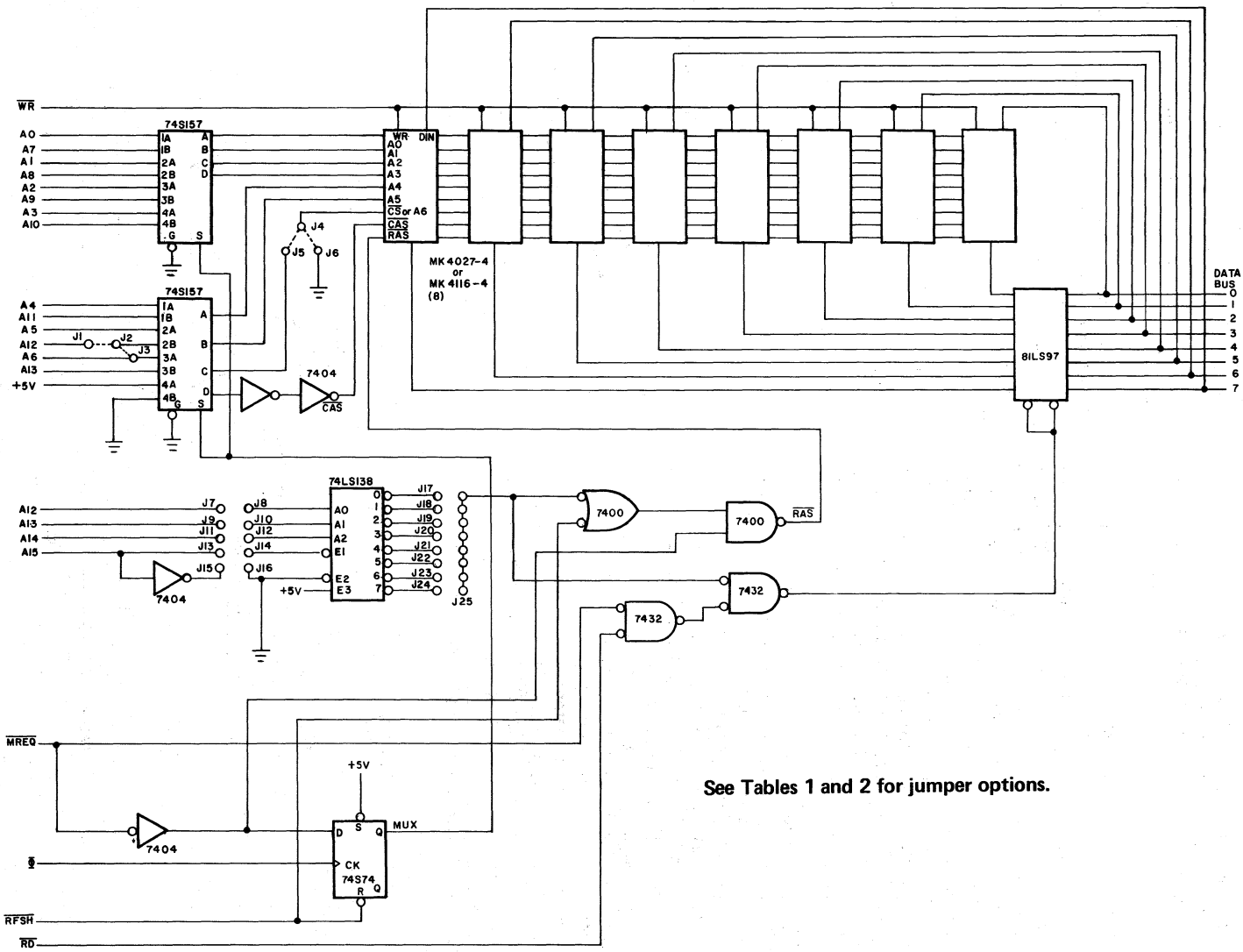


MANUAL AND POWER-ON RESET CIRCUIT
Figure 5b



DESIGN EXAMPLE NO. 1 SCHEMATIC DIAGRAM

Figure 6



See Tables 1 and 2 for jumper options.

DESIGN EXAMPLES FOR INTERFACING THE Z80 TO DYNAMIC MEMORY

To illustrate the interface between the Z80 and dynamic memory, two design examples are presented. Example number 1 is for a 4K/16Kx8 memory and the example number 2 is a 16K/64Kx8 memory.

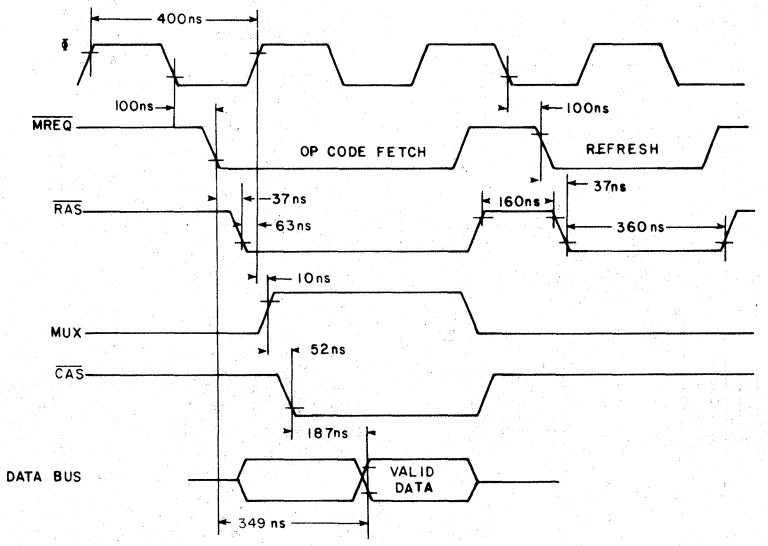
Design Example Number 1: 4K/16Kx8 Memory

This design example describes a 4K/16Kx8 memory that is best suited for a small single board Z80 based microcomputer system. The memory devices used in the example are the MK4027 (4,096x1 MOS Dynamic RAM) and the MK4116 (16,384x1 MOS Dynamic RAM). A very important feature of this design is the ease in which the memory can be expanded from a 4Kx8 to a 16Kx8 memory. This is made possible by the use of jumper options which configure the memory for either the MK4027 or the MK4116. See Table 1 and 2 for jumper options.

Figure 6 shows the schematic diagram for the 4K/16Kx8 memory. A timing diagram for the Z80 control signals and memory control signals is shown in Figure 7. The operation of the circuit may be described as follows: $\overline{RAS}$ is generated by NANDing MREQ with RFSH + ADDRESS DECODE. $\overline{RFSH}$ is generated directly from the Z80 while address decode comes from the 74LSI38 decoder. Address decode indicates that the address on the bus falls within the memory boundaries of the memory. If an op code fetch or memory read is being executed the 81LS97 output buffer will be enabled at approximately the same time as $\overline{RAS}$ is generated for the memory array. The output buffer is enabled only during an op code fetch or memory read when $\overline{ADDRESS\ DECODE}$, $\overline{MREQ}$, and $\overline{RD}$ are all low. The switch multiplexer signal (MUX) is generated on the rising edge of Φ after $\overline{MREQ}$ has gone low during an op code fetch, memory read or memory write. After MUX is generated and the address multiplexers switch from the row address to column address, $\overline{CAS}$ will be generated. $\overline{CAS}$ comes from one of the outputs of the multiplexer and is delayed by two gate delays to ensure that the proper column address set-up time will be achieved. Once $\overline{RAS}$ and $\overline{CAS}$ have been generated for the memory array, the memory will then access the desired location for a read or write operation.

7404	22ns	Generate $\overline{RAS}$ from $\overline{MREQ}$
7400	15ns	
	63ns	$\overline{RAS}$ to rising edge of Φ
74S74	10ns	Φ to MUX
74S157	15ns	Generate $\overline{CAS}$ from MUX
7404	22ns	
7404	15ns	
t_{CAC}	165ns	$\overline{CAS}$ access time
81LS97	22ns	Output buffer delay
	349ns	Worst case access

DESIGN EXAMPLE NO. 1 MEMORY TIMING
Figure 7



The worst case access time required by the CPU for the op code fetch is 450ns (from equation 1); therefore, the circuit exceeds the required access time by 101ns (worst case).

The circuit shown in Figure 6 provides excellent performance when used as a small on board memory. The memory size should be held at eight devices because there is not sufficient timing margin to allow the interface circuit to drive a larger memory array.

Design Example Number 2: 16Kx8 Memory

This design example describes a 16K/64Kx8 memory which is best suited for a Z80 based microcomputer system where a large amount of RAM is desired. The memory devices used in this example are the same as for the first example, the MK4027 and the MK4116. Again as with the first example, the memory may be expanded from a 16Kx8 to a 64Kx8 by reconfiguring jumpers. See Table 3 and 4 for jumper options.

Figure 8 shows the schematic diagram for the 16K/64K memory. A timing diagram is shown in Figure 9. The operation of the circuit can be described as follows: $\overline{RAS}$ is generated by NANDing MREQ with ADDRESS DECODE (from the two 74LSI38s) + RFSH. Only one row of RAMs will receive a $\overline{RAS}$ during an op code fetch, memory read or memory write. However, an $\overline{RAS}$ will be generated for all rows within the array during a refresh cycle. $\overline{MREQ}$ is inverted and fed into a TTL compatible delay line to generate MUX and $\overline{CAS}$. (This particular approach differs from the method used in example number 1 in that all memory timing is referenced to MREQ, whereas the circuit in example number 1 bases its memory timing from both $\overline{MREQ}$ and the clock. Both methods offer good results; however, the TTL delay line approach offers the best control over the memory timing.) MUX is generated 65ns later and is used to switch the 74157 multiplexers from the row to the column address. The 65ns delay was chosen to allow adequate margin for the row address hold time t_{RAH}. At 110ns, $\overline{CAS}$ is generated from the delay line and NANDed with RFSH, which inhibits a $\overline{CAS}$ during refresh cycle. After $\overline{CAS}$ is applied to the memory, the desired location is then accessed. A worst case access timing analysis for the circuit shown in Figure 8 can be computed as follows:

74LS14	22ns	Generate $\overline{RAS}$ from $\overline{MREQ}$
74LS00	15ns	
delay line	50ns	MUX from $\overline{RAS}$
delay line	45ns	$\overline{CAS}$ delay from MUX
7400	20ns	
t_{CAC}	165ns	Access time from $\overline{CAS}$
8833	30ns	Output buffer delay
	347ns	

The required access time from the CPU is 450ns (from equation 1). This leaves 103ns of margin for additional CPU buffers on the control and address lines. This particular circuit offers excellent results for an application which requires a large amount of RAM memory. As mentioned earlier, the memory timing used in this example offers the best control over the memory timing and would be ideally suited for an application which required direct memory access (DMA).

V

4K x 8 CONFIGURATION(MK4027) JUMPER
Table 1

CONNECT: J13 to J14

ADDRESS	CONNECT
0000-0FFF	J17 to J25
1000-1FFF	J18 to J25
2000-2FFF	J19 to J25
3000-3FFF	J20 to J25
4000-4FFF	J21 to J25
5000-5FFF	J22 to J25
6000-6FFF	J23 to J25
7000-7FFF	J24 to J25

Connect: J2 to J3
J4 to J6
J7 to J8
J9 to J10
J11 to J12

CONNECT: J14 to J15

ADDRESS	CONNECT
8000-8FFF	J17 to J25
9000-9FFF	J18 to J25
A000-AFFF	J19 to J25
B000-BFFF	J20 to J25
C000-CFFF	J21 to J25
D000-DFFF	J22 to J25
E000-EFFF	J23 to J25
F000-FFFF	J24 to J25

16K x 8 CONFIGURATION (MK4116) JUMPER CONNECTIONS
Table 2

CONNECT:
J1 to J2
J4 to J5
J8 to J11
J10 to J13
J12 to J16
J14 to J16

ADDRESS	CONNECT
0-3FFF	J17 to J25
4000-7FFF	J18 to J25
8000-BFFF	J19 to J25
C000-FFFF	J20 to J25

16K x 8 CONFIGURATION (MK4027)

Table 3

CONNECT: J1 to J3
J5 to J6
J7 to J8
J9 to J10
J11 to J12
J13 to J14

ADDRESS:	0-3FFF	ADDRESS:	4000-7FFF	ADDRESS:	8000-BFFF	ADDRESS:	C000-FFFF
CONNECT:	J24 to J25	CONNECT:	J16 to J17	CONNECT:	J40 to J41	CONNECT:	J32 to J33
	J26 to J27		J18 to J19		J42 to J43		J34 to J35
	J28 to J29		J20 to J21		J44 to J43		J36 to J37
	J30 to J31		J22 to J23		J46 to J47		J38 to J39

64K x 8 CONFIGURATION(MK4116)

Table 4

CONNECT: J1 to J2
J4 to J5
J8 to J11
J10 to J13
J12 to J15
J14 to J15

ADDRESS: 0-FFFF
CONNECT: J32 to J33
J34 to J35
J36 to J37
J38 to J39

SYSTEM PERFORMANCE CHARACTERISTICS

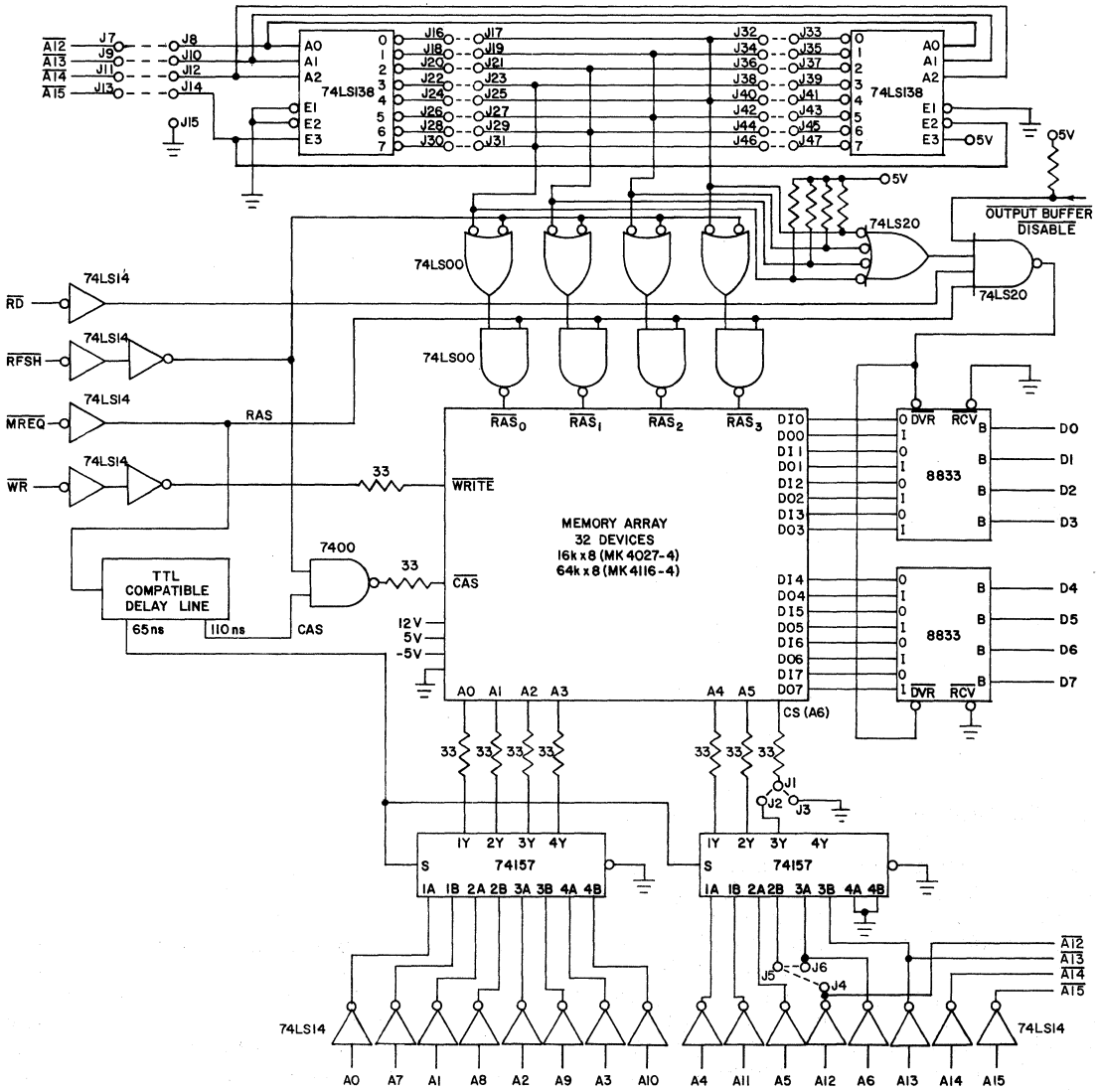
Table 5

The system characteristics for the preceeding design examples are shown in Table 5.

EXAMPLE #	MEMORY CAPACITY	MEMORY ACCESS	POWER REQUIREMENTS
1	4K/16Kx8	349ns max.	+12V @ 0.0250 A max. +5V @ 0.422 A max.* -5V @ 0.030 A max.
2	16K/64Kx8	347ns max.	+12V @ 0.600 A max. +5V @ 0.550 A max. * -5V @ 0.030 A max.

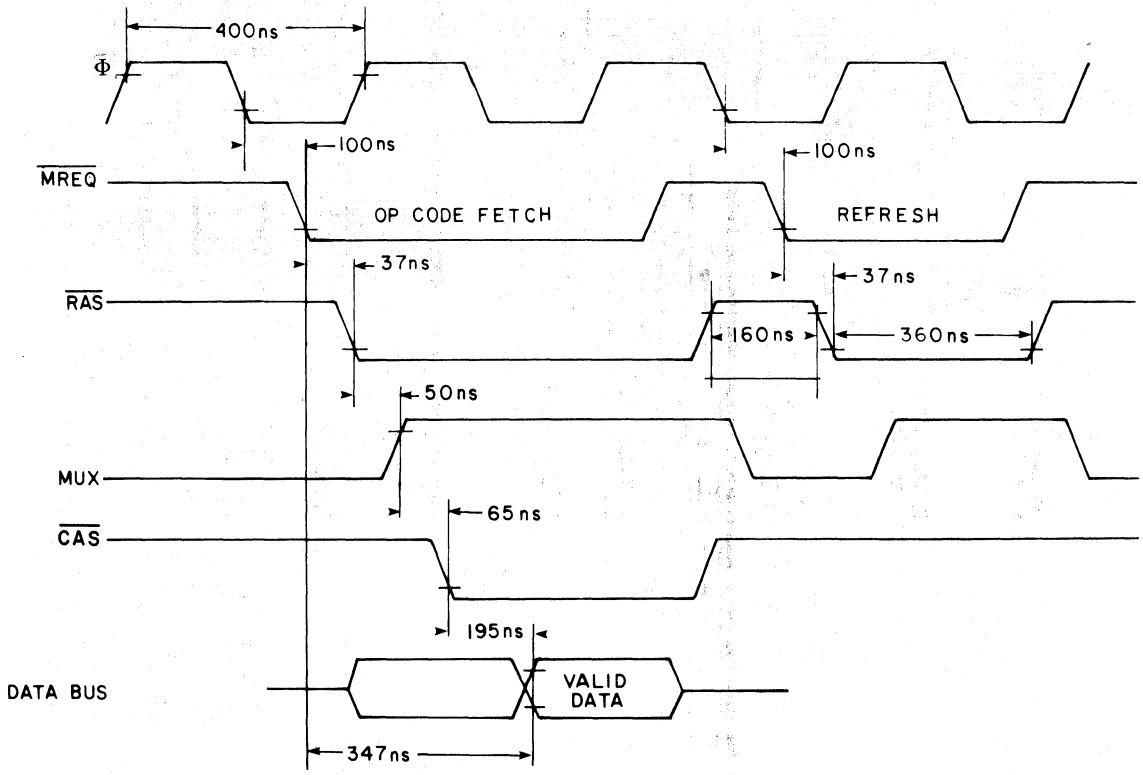
*All power requirements are max.; operating temperature 0°C to 70°C ambient, max +12V current computed with Z80 executing continuous op code fetch cycles from RAM at 1.6 μs intervals.

DESIGN EXAMPLE NO. 2 SCHEMATIC DIAGRAM
Figure 8



FOR JUMPER OPTIONS SEE TABLES 3 AND 4

DESIGN EXAMPLE NO. 2 MEMORY TIMING
Figure 9



PRINTED CIRCUIT LAYOUT

One of the most important parts of a dynamic memory design is the printed circuit layout. Figure 10 illustrates a recommended layout for 32 devices. A very important factor in the P.C. layout is the power distribution. Proper power distribution on the V_{DD} and V_{BB} supply lines is necessary because of the transient current characteristics which dynamic memories exhibit. To achieve proper power distribution, V_{DD}, V_{BB}, V_{CC} and ground should be laid out in a grid to help minimize the power distribution impedance. Along with good power distribution, adequate capacitive bypassing for each device in the memory array is necessary. In addition to the individual by-passing capacitors, it is recommended that each supply (V_{BB}, V_{CC} and V_{DD}) be bypassed with an electrolytic capacitor 20μF.

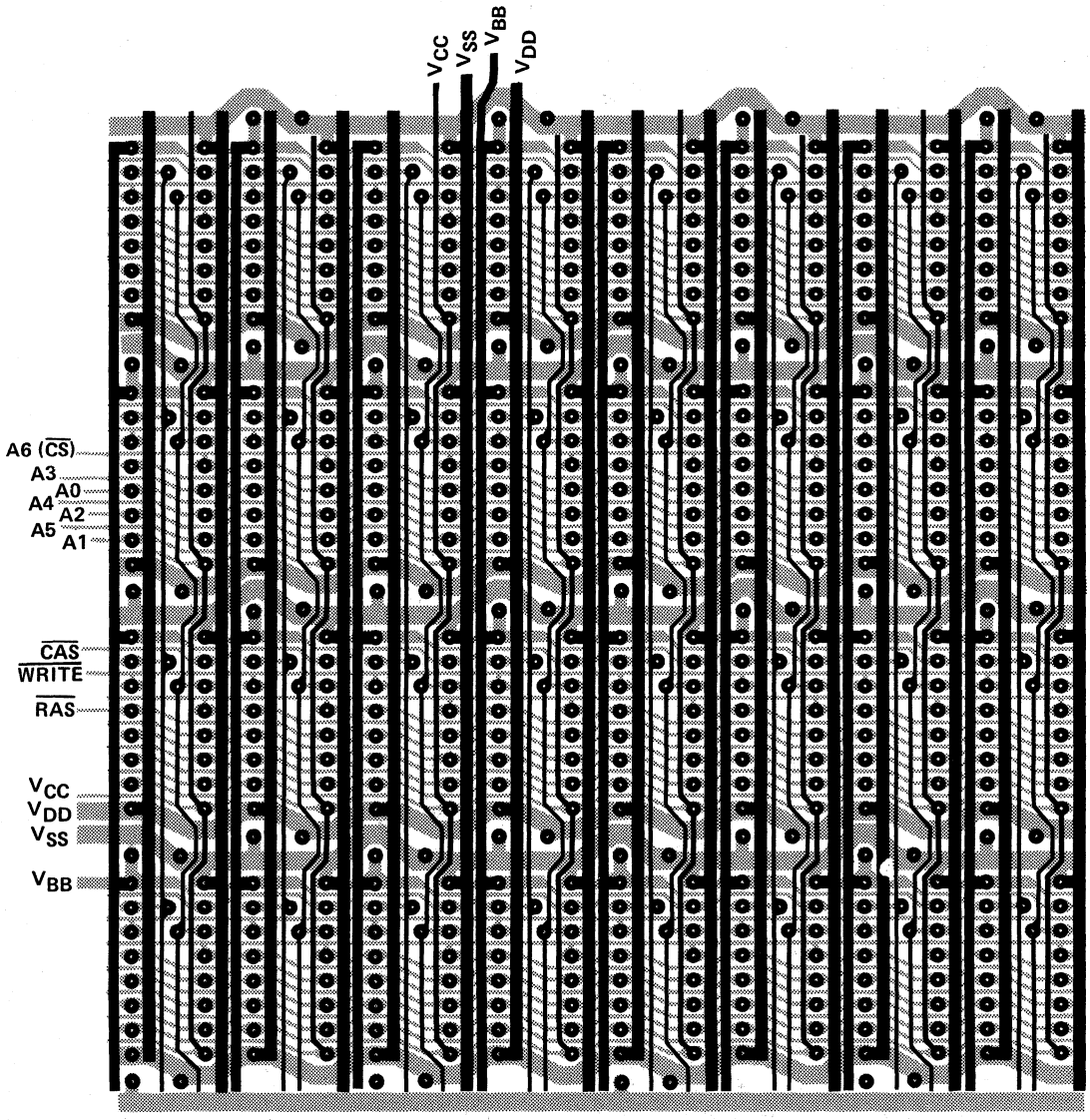
By using good power distribution techniques and using the recommended number of bypassing capacitors, the designer can minimize the amount of noise in the memory array. Other layout considerations are the placement of signal lines. Lines such as address, chip select, column address strobe, and write should be bussed together as rows; then, all rows should be bussed together at one end of the array. Interconnection between rows should be avoided. Row address strobe lines should be bussed together as a row, then connected to the appropriate $\overline{RAS}$ driver. TTL drivers for the memory array signals should be located as close as possible to the array to help minimize signal noise.

For a large memory array such as the one shown in design example number 2, series terminating resistors should be used to minimize the amount of negative undershoot. These resistors should be used on the address lines, $\overline{CAS}$ and $\overline{WRITE}$, and have values between 20 Ω to a 33 Ω .

The layout for a 32 device array can be put in a 5" x 5" area on a two sided printed circuit board.

SUGGESTED P. C. LAYOUT FOR MK4027 or MK4116

Figure 10



V

4MHz Z80 DYNAMIC MEMORY INTERFACE CONSIDERATIONS

A 4MHz Z80 is available for the microcomputer designer who needs higher system throughput. Considerations which must be faced by the designer when interfacing the 4MHz Z80 to dynamic memory are the need for memories with faster access times and for providing minimum RAM precharge time. The access times required for dynamic memory interfaced to a 4MHz Z80 can be computed from equations 1 and 2 under Z80 Timing and Memory Control Signals.

Access time for op code fetch for 4MHz Z80,
let: $t_C = 250ns$; $t_{DL\overline{\Phi}(MR)} = 75ns$; $t_{s\Phi(D)} = 35ns$
then: $t_{ACCESS\ OP\ CODE} = 265ns$

Access time for memory read for 4MHz Z80,
let: $t_C = 250ns$; $t_{DL\overline{\Phi}(MR)} = 75ns$; $t_{S\overline{\Phi}(D)} = 50ns$
then: $t_{ACCESS\ MEMORY\ READ} = 375ns$

The problem of faster access times can be solved by using 200ns memories such as the MK4027-3 or MK4116-3. Depending on the number of buffer delays in the system, the designer may have to use 150ns memories such as the MK4027-2 or MK4116-2. The most critical problem that exists when interfacing dynamic memory to the 4MHz Z80 is the RAM precharge time (trp). This parameter is called $t_{W(MRH)}$ on the Z80 and can be computed by the following equation.

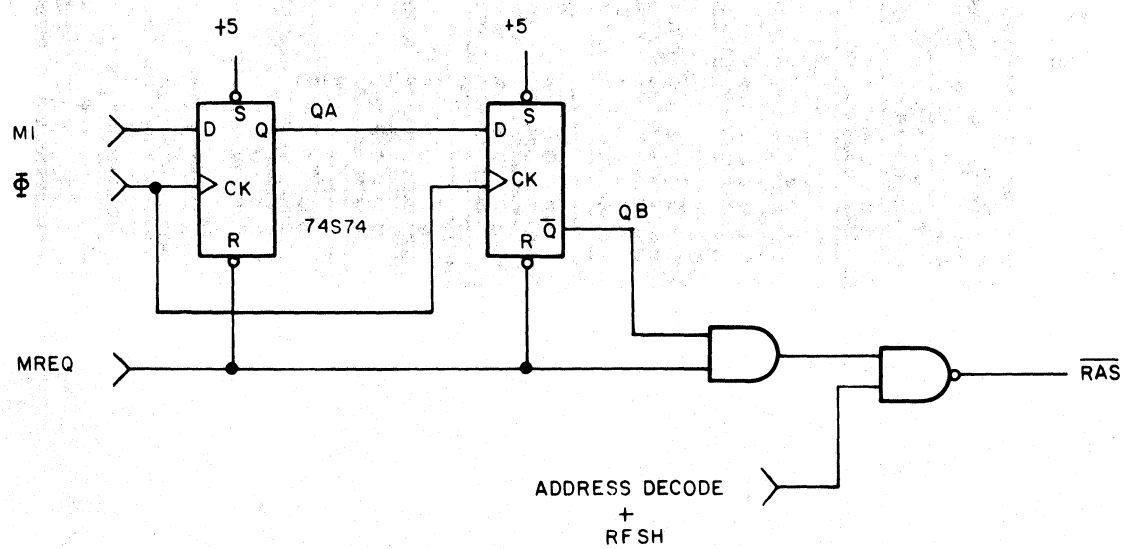
(4) $t_{W(RH)} = t_{W(\Phi H)} + t_f - 20ns$
let: $t_{W(\Phi H)} = 110ns$; $t_f = 5ns$
then: $t_{W(MRH)} = 95ns$

A $t_{W(MRH)}$ of 95ns will not meet the minimum precharge time of the MK4027-2 or MK4116-2 which is 100ns. The MK4027-3 and MK4116-3 require a 120ns precharge. Figure 11 shows a circuit that will lengthen the $t_{W(MRH)}$ pulse from 95ns to a minimum of 126ns while only inserting one gate delay into the access timing chain. Figure 12 shows the timing for the circuit of Figure 11. The operation of the circuit in Figure 11 can be explained as follows: the D flip flops are held in a reset condition until MREQ goes to its active state. After MREQ goes active, on the next positive clock edge, the D input of U1 and U2 will be transferred to the outputs of the flip flops. Output QA will go high if M1 was high when Φ clocked U1. Output QB will go low on the next positive going clock edge, which will cause the output of U3 to go low and force the output of U4, which is $\overline{RAS}$, high. The flip flops will be reset when MREQ goes inactive.

The circuit shown in Figure 11 will give a minimum of 126ns precharge for dynamic memories, with the Z80 operating at 4MHz. The 126ns $t_{W(MRH)}$ is computed as follows.

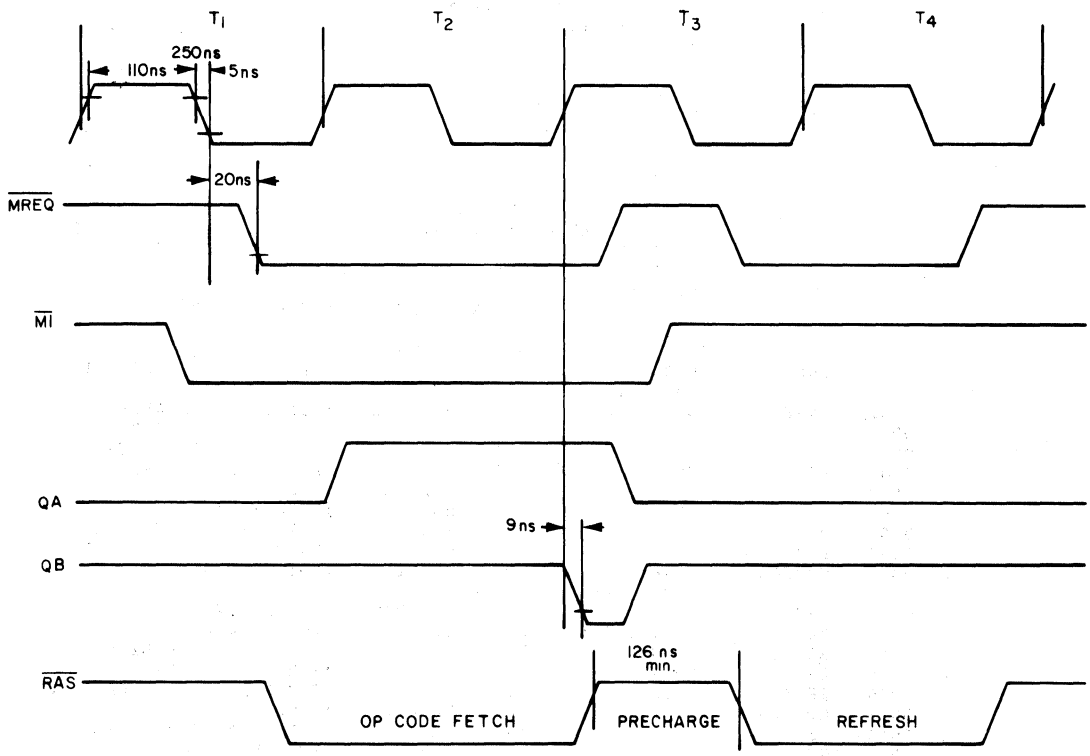
110ns	$t_{W(\Phi H)}$ - clock pulse width high (min)
5ns	t_F - clock full time (min)
20ns	$t_{DL\overline{\Phi}(MR)}$ - MREQ delay (min)
-9ns	74S74 delay (min)
126ns	$t_{W(MRH)}$ modified (min)

4MHz Z80 PRECHARGE EXTENDER FOR DYNAMIC MEMORIES
Figure 11



TIMING DIAGRAM FOR 4MHz Z80 PRECHARGE EXTENDER

Figure 12



APPENDIX

MEMORY TEST ROUTINE

This section is intended to give the microcomputer designer a memory diagnostic suitable for testing memory systems such as the ones shown in Section VI.

The routine is a modified address storage test with an incrementing pattern. A complete test requires 256_{10} passes, which will execute in less than 4 minutes for a 16Kx8 memory. If an error occurs, the program will store the pattern in location '2C'H and the address of the error at locations '2D'H and '2E'H.

The program is set up to test memory starting at location '2F'H up to the end of the block of memory defined by the bytes located at 'OC'H and 'OD'H. The test may be set up to start at any location by modifying locations '03'H - '04'H and '11'H - '12'H with the starting address that is desired.

```
                                MXRTS  LISTING                    PAGE    0001
LOC    OBJ CODE    STMT SOURCE STATEMENT

                   0001 ;TRANSLATED FROM DEC 1976 INTERFACE MAGAZINE
                   0002 ;
                   0003 ;THIS IS A MODIFIED ADDRESS STORAGE TEST WITH AN
                   0004 ;INCREMENTING PATTERN
                   0005 ;
                   0006 ;256 PASSES MUST BE EXECUTED BEFORE THE MEMORY IS
                   0007 ;COMPLETELY TESTED.
                   0008 ;
                   0009 ;IF AN ERROR OCCURS, THE PATTERN WILL BE STORED
                   0010 ;AT LOCATION '002C'H AND THE ADDRESS OF THE
                   0011 ;ERROR LOCATION WILL BE STORED AT '002D'H AND
                   0012 ;'002E'H.
                   0013 ;
```

MEMORY TEST ROUTINE (Cont'd.)

```
                    0014 ;THE CONTENTS OF LOCATIONS '000C'H AND '001D'H
                    0015 ;SHOULD BE SELECTED ACCORDING TO THE FOLLOWING
                    0016 ;MEMORY SIZE TO BE TESTED
                    0017 ;
                    0018 ;TOP OF MEMORY TO
                    0019 ;BE TESTED                          VALUE OF EPAGE
                    0020 ;
                    0021 ;      4K                                  '10'H
                    0022 ;      8K                                  '20'H
                    0023 ;     16K                                  '40'H
                    0024 ;     32K                                  '80'H
                    0025 ;     48K                                  'C0'H
                    0026 ;     64K                                  'FF'H
                    0027 ;
                    0028 ;THE PROGRAM IS SET UP TO START TESTING AT
                    0029 ;LOCATION '002F'H. THE STARTING ADDRESS FOR THE
                    0030 ;TEST CAN BE MODIFIED BY CHANGING LOCATIONS
                    0031 ;'0003-0004'H AND '0011-0012'H.
                    0032 ;
                    0033 ;TEST TIME FOR A 16K X 8 MEMORY IS APPROX. 4 MIN
                    0034 ;
0000                0035          ORG   0000H
0000   0600         0036          LD    B,0        ;CLEAR B PATRN MODIFIER
                    0037 ;LOAD UP MEMORY
0002   212F00       0038 LOOP:    LD    HL,START   ;GET STARTING ADDR
0005   7D           0039 FILL:    LD    A,L        ;LOW BYTE TO ACCM
0006   AC           0040          XOR   H          ;XOR WITH HIGH BYTE
0007   A8           0041          XOR   B          ;XOR WITH PATTERN
0008   77           0042          LD    (HL),A     ;STORE IN ADDR
0009   23           0043          INC   HL         ;INCREMENT ADDR
000A   7C           0044          LD    A,H        ;LOAD HIGH BYTE OF ADDR
000B   FE10         0045          CP    EPAGE      ;COMPARE WITH STOP ADDR
000D   C20500       0046          JP    NZ,FILL    ;NOT DONE,GO BACK
                    0047 ;READ AND CHECK TEST DATE
0010   212F00       0048          LD    HL,START   ;GET STARTING ADDR
0013   7D           0049 TEST:    LD    A,L        ;LOAD LOW BYTE
0014   AC           0050          XOR   H          ;XOR WITH HIGH BYTE
0015   A8           0051          XOR   B          ;XOR WITH MODIFIER
0016   BE           0052          CP    (HL)       ;COMPARE WITH MEMORY LOC
0017   C22500       0053          JP    NZ,FXIT    ;ERROR EXIT
001A   23           0054          INC   HL         ;UPDATE MEMORY ADDRESS
001B   7C           0055          LD    A,H        ;LOAD HIGH BYTE
001C   FE10         0056          CP    EPAGE      ;COMPARE WITH STOP ADDR
001E   C21300       0057          JP    NZ,TEST    ;LOOP BACK
0021   04           0058          INC   B          ;UPDATE MODIFIER
```



```
 LOC   OBJ CODE   STMT SOURCE STATEMENT

0022   C30200      0059          JP    LOOP       ;RST WITH NEW MODIFIER
                   0060 ;ERROR EXIT
0025   222D00      0061 FXIT:    LD    (BYTE),HL  ;SAVE ERROR ADDRESS
0028   322C00      0062          LD    (PATRN),A  ;SAVE BAD PATTERN
002B   76          0063          HALT             ;FLAG OPERATOR
002C               0064 PATRN:   DEFS  1
002D               0065 BYTE:    DEFS  2
002F   2F00        0066 START:   DEFW  $
                   0068 EPAGE:   EQU   10H        ;SET UP FOR 4K TEST
                   0069          END
```

MOSTEK®

APPLYING THE Z80 SIO IN ASYNCHRONOUS DATA COMMUNICATIONS

Application Note

Serial asynchronous data links, probably the most prevalent mode of data communications in existence today, require versatile, easy to interface communications devices. The Z80 SIO is just such a device. Although it is just as equally suited in virtually all serial protocol environments, no compromises were made in asynchronous applications. The Z80 SIO operating features include:

- ☐ Data communications' rates of up to 800K bits/s
- ☐ Three FIFO receive data buffers per channel
- ☐ Full duplex operation
- ☐ Break generation and detection
- ☐ Parity, overrun, and framing error detection
- ☐ Polled or interrupt driven

The most salient of the SIO's many features is its capability to operate using prioritized vector interrupts, offering unparalleled speed and efficiency in maximizing data throughput. Although the SIO can be operated in polled as well as interrupt modes, the latter will be emphasized in the following discussion owing to its inherent power and versatility.

In order to understand the use of the SIO in serial data communications better, a look at the internal organization of the chip would be helpful. Figure 1 depicts the functional logic of one of the SIO channels. As shown, there are a total of 11 registers accessible by the programmer. "Write Registers" (WR0-WR7), as they are referred to, are used to configure the SIO to the desired type of protocol and include such information as data rates, parity information, word length, etc. In addition, three "Read Registers" (RR0-RR2) are provided for monitoring data flow and error conditions. For a detailed description of these registers, as well as the entire MK3884, reference should be made to the MK3884 Z80 SIO Technical Manual. In the receiver section, notice that data flows from the receive shift register to the three receive buffers.

These registers are configured in a first in, first out (FIFO) arrangement, thus providing the data link with additional overrun protection. Associated with each receive buffer is a corresponding error buffer, enabling the programmer to poll the various Read Registers and ascertain error conditions corresponding to the data. This concept is illustrated in Figure 2. Receive Buffer 3 contains data, has no associated errors, and will be read next by the CPU, as it is at the top of the stack. Receive Buffer 2 has a parity error associated with its data word, indicated by the "1" in the parity column. Similarly, Receive Buffer 1 has an associated overrun error condition, indicating that it has been overwritten at least once. This type of FIFO arrangement allows the programmer three full receive word-times to read the SIO before losing any data, which is extremely advantageous when the programmer must perform numerous house-keeping functions. The SIO is also capable of full duplex operation, illustrated in Figure 2 by separate data paths for the transmitter and receiver. Notice the separate transmit and receive clock inputs for situations requiring different clock rates. A $\overline{SYNC}$ input is provided as a general purpose input in asynchronous communications, and is used to establish synchronization in monosync and bisync communications. Finally, all standard modem control signals are present for handshaking including $\overline{DCD}$, $\overline{DTR}$, $\overline{CTS}$, and $\overline{RTS}$.

The SIO interfaces easily with the Z80 CPU, and generally requires little, if any modification of control signals when used with other CPU's. Figures 3A and 3B show the typical interconnections and addressing techniques between the SIO and CPU. Note that the C/$\overline{D}$ (control data) and B/$\overline{A}$ (Channel B/A) pins may be connected to A0 and A1 of the address bus, respectively. Figure 3B further illustrates SIO addressing, where even numbered addresses decode the channel (A or B) and define a data operation. Conversely, odd numbered addresses define a control operation to the addressed port.

There are also two clock considerations that deserve attention. 1) The SIO is a synchronous device, whose clock (Φ) must be identical to that of the CPU clock. 2) Although the SIO is capable of high data rates, care should be taken to ensure that the system clock (Φ) is at least 5 times the data rate, as specified in the data book.

THE ASYNCHRONOUS MODEL

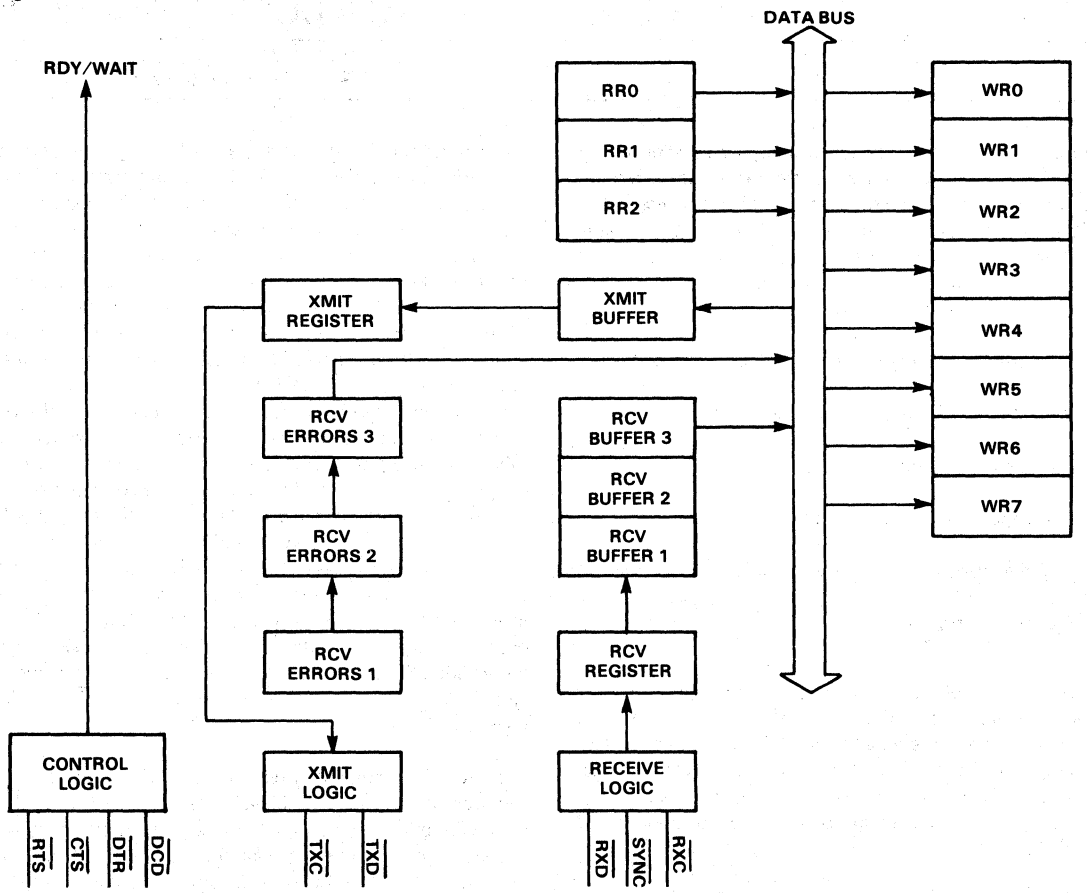
In discussing the use of the Z80 SIO in Async communications, the illustration in Figure 4 depicts the method in which the SIO receiver logic assembles, and

V

transmitter logic sends, serial data asynchronously. The data stream consists of one start bit, a variable length data word (selectable 5-8 bits), an optional parity bit, and selectable stop bits (1, 1½ or 2). Note that the data word is sent low order bit first and must be right justified if less than 8 data bits are contained in the data field.

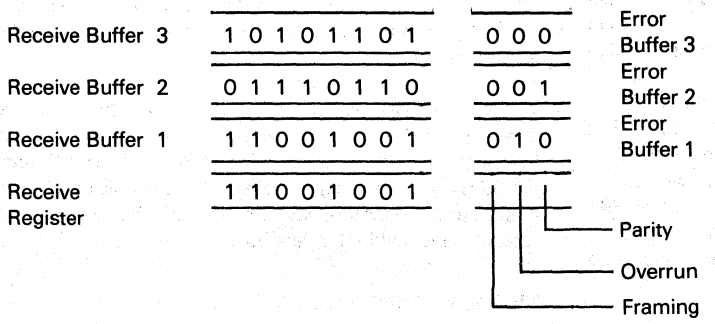
FUNCTIONAL LOGIC OF AN SIO CHANNEL

Figure 1

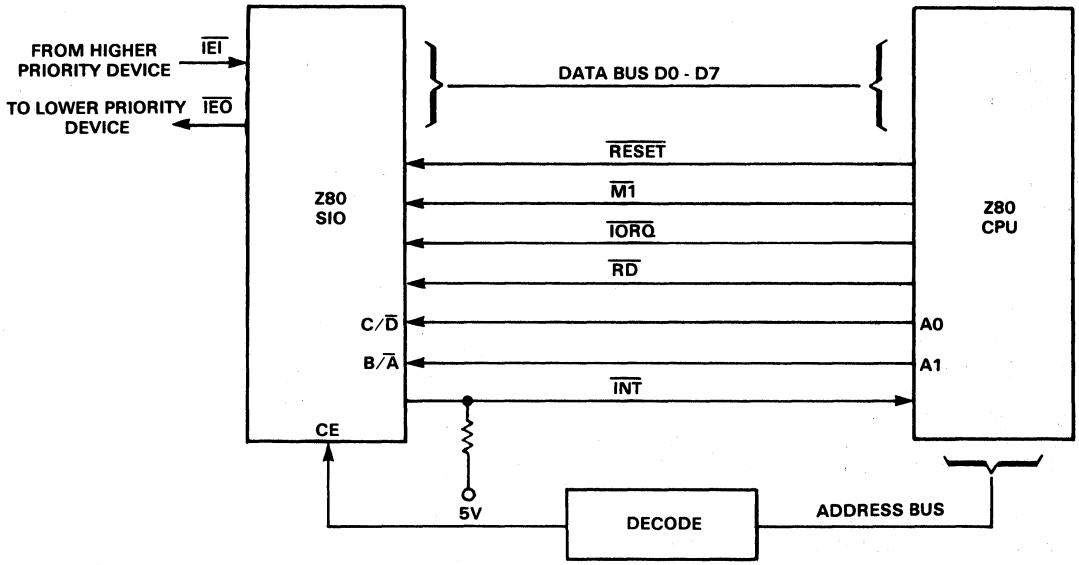


SIO RECEIVE FIFO DATA/ERROR BUFFERS

Figure 2



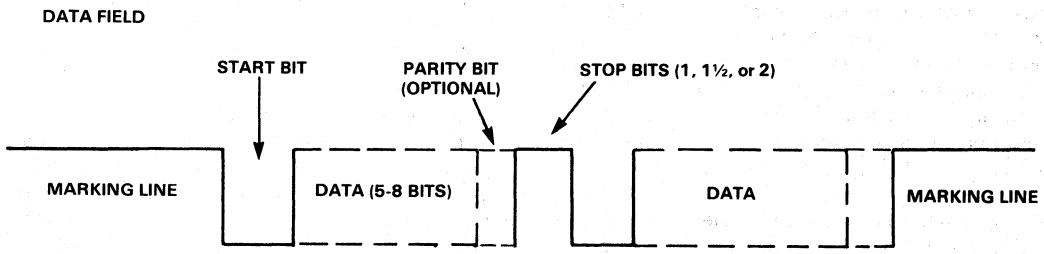
Z80 CIO - CPU INTERCONNECTIONS
Figure 3A



Z80 SIO-CPU INTERCONNECTIONS
Figure 3B

Address Label	$\overline{CE}$	B/$\overline{A}$	C/$\overline{D}$	
SIO ADD	0	0	0	Channel A XMIT/RCV ADDRESS
SIO ADD +1	0	0	1	Channel A READ/WRITE ADDRESS
SIO ADD +2	0	1	0	Channel B XMIT/RCV ADDRESS
SIO ADD +3	0	1	1	Channel B READ/WRITE ADDRESS

ASYNCHRONOUS FORMAT
Figure 4



ASYNCHRONOUS PROGRAMMING

The design of a serial data communications link utilizing the SIO will comprise three basic software modules:

- Initialization
- Data transfer
- Error detection and recovery

The program flow for initialization is illustrated by the flow diagram in Figure 5. This procedure consists of writing a string of control bytes to the SIO using the pointer register, WR0. Each control byte is preceded by the pointer register byte, which tells the SIO which of 8 write registers is to be addressed. Initialization is executed in this alternating pointer register/control-data fashion until the SIO is configured as desired, as shown in Table 1.

Although SIO initialization is not necessarily order dependent, the logical order as depicted in Figure 5 and Table 1 is highly recommended.

Also, a word of caution concerning the use of WR0 is appropriate at this time. As WR0 has a dual function - that of a pointer to other control registers and commands (i.e., reset channel, error reset, etc.) to the SIO - it is possible to perform both of these functions simultaneously. This is not recomended because following each command, the internal register pointer resets to zero, thus preventing the ensuing control word from loading properly. Each command issued should address WR0, as pointed out in the example.

Data transfer and error handling methods are presented in their simplest form in Table 1. The first eight bytes of code initialize the SIO, which consists of initializing the CPU internal registers B, C, and HL with the table length, port address, and table address, respectively. Notice how efficiently the use of the OTIR instruction transfers the entire block of data to the SIO. Although channel A is the active channel being used in this example, channel B must also be accessed, as shown. This is because the WR2 and the status affects vector bit are active in channel B only.

Another instruction of interest is the "EI" instruction, both because of its existence and placement. Whenever the CPU acknowledges an interrupt, interrupts within the CPU are disabled and remain so until an "EI" instruction is executed. Hence, the placement of "EI" in the program example forms a non-nested interrupt structure. Conversely, placing "EI" at the beginning of a subroutine would constitute nested interrupts, as other devices could now cause interrupts.

The interrupts themselves may be initiated by the SIO in many different ways. The transmitter and receiver interrupts are initiated when the transmit buffer empty and receiver character available bits are set. In the case of receive errors, interrupt requests are made (if programmed to do so) when any of several special error conditions exist. As shown in the program initialization (Table 1), the special effects vector is enabled, allowing the SIO to modify the returned vector, indicating either a) transmit buffer full, b) receive-character available, c) External/Status change, or d) special receive conditions. This powerful feature further reduces programming overhead and thus allows greater efficiency and data throughput. Also at the programmers discretion is the ability to initiate data transfer automatically by monitoring the modem control signals. This is effected by the SIO detecting $\overline{DCD}$ and $\overline{CTS}$ in an active state which, in turn, enables the receiver and transmitter, respectively. Also, if External interrupts are enabled as they are in the example, interrupts are generated upon transition of $\overline{DCD}$ or $\overline{CTS}$. This feature is useful in initiating line turn-around and detecting break conditions. Once External/Status Interrupts have been acknowledged, they must be reset by writing to register 0 of the appropriate channel. Note also in the initialization procedure that immediately following a chip or channel reset, the "Reset External/Status Interrupts" command should be executed to prevent possible spurious interrupts.

Should the programmer choose not to operate in an interrupt mode, all of the aforementioned conditions would have to be polled by reading the appropriate read register (RR0-RR2). When operating in this mode, the proper sequence of checking the SIO for receive characters would be:

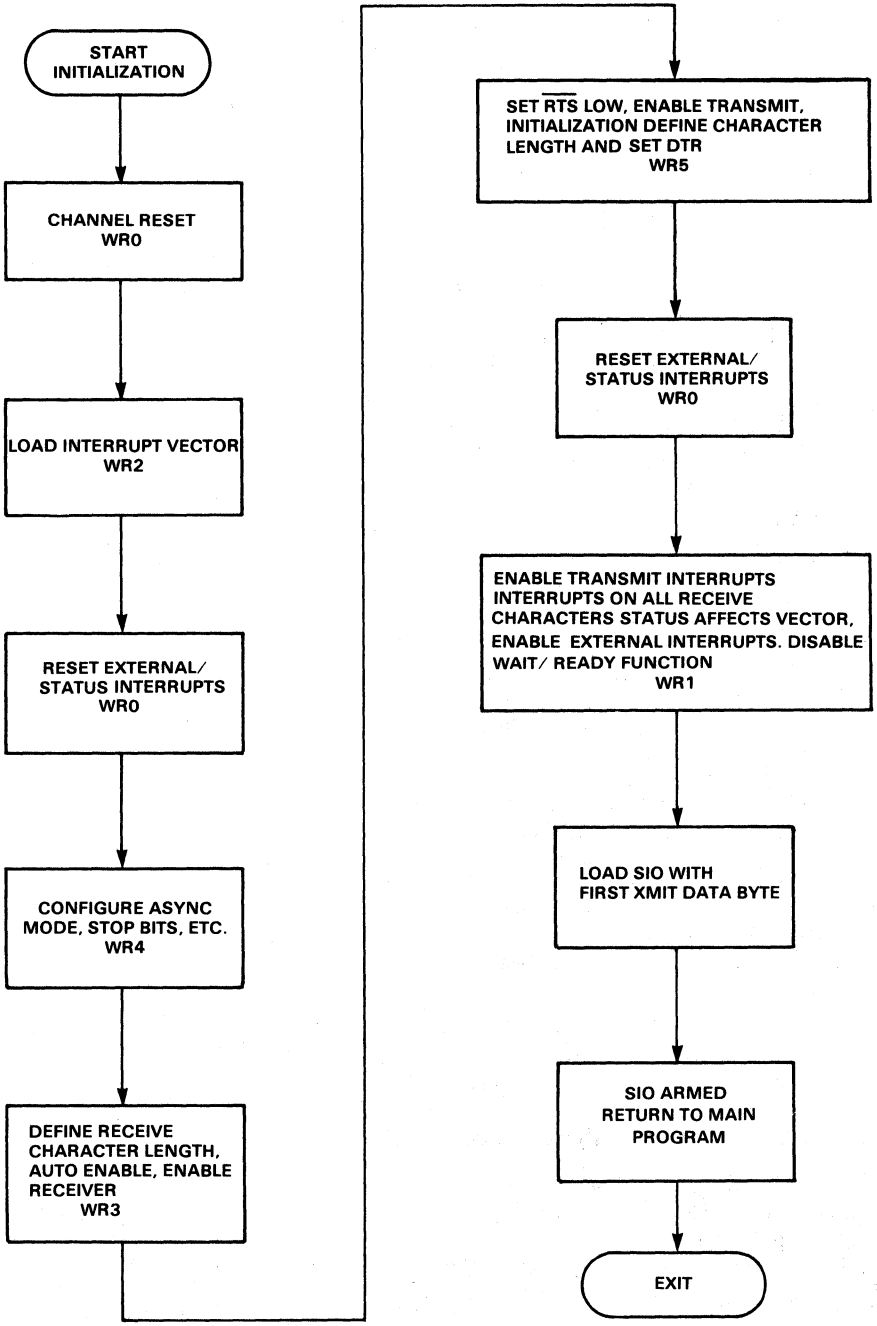
1. Read RR0; determine if a character is available.
2. If so, interrogate RR1 to ascertain error status.
3. Read the DATA.

The status of errors should be checked before reading the data to preserve the proper error to data word correspondence. An example of reading the Read Registers is given in Table 1 under Error Handler. As illustrated, RR1 is accessed by first performing a "write" to WR0 which points to register 1, and then by performing a "READ" operation. Once the status byte is in the accumulator, each of the pertinent error bits are interrogated using the "bit" instruction. Associated with each type of error is its error routine which takes the appropriate recovery action. When interrupts are used, as in this example, care should be taken within each Error Routine to perform an Error Reset command, thus allowing future error interrupts to occur.

The preceding example should equip the user with a "guide" for programming the SIO, not only in asynchronous communications, but synchronous and SDLC/HDLC as well. Of course, the latter protocols deserve special attention, and are covered in detail in the MK3884 Technical Manual. As demonstrated, the SIO, when taken advantage of, can be an extremely powerful device in any data communications' link.

LOGICAL FLOW OF SIO INITIALIZATION

Figure 5



V

TYPICAL PROGRAM EXAMPLE

Table 1

```
LABEL   SOURCE STATEMENT  COMMENTS

UNIT    LD B, LENG B      ; LENGTH of Table, CH B
        LD C, SIOCTL +2   ; Port Address, CH B
        LD HL, CTLTB      ; TABLE Address, CH B
        OTIR              ; Initialized CH B
        LD B, LENG A      ; Length of Table, CH A
        LD C, SIOCTL      ; Port Address, CH A
        LD HL, CTLTA      ; Table Address, CH A
        OTIR              ; Initialize CH A

CTRLTA  DEFB '18' H       ; WR0, RESET CH A
         DEFB '10' H      ; WR0, Reset External/Status
                            Interrupts
        DEFB '04' H       ; Pointer to WR4
        DEFB '40' H       ; X16 CLK, ODD Parity, 2 stop
                            bits
        DEFB '03' H       ; Pointer to WR3
        DEFB '61' H       ; 7 bits/char, receive and
                            auto enable
        DEFB '05' H       ; Pointer to WR5
        DEFB 'AA' H       ; Set RTS, DTR; 7 bits/char.,
                            enable Xmit
        DEFB '10' H       ; WR0; reset EXT/STATUS
                            INT.
        DEFB '01' H       ; Pointer to WR1
        DEFB '17' H       ; Enable external and
                            transmit interrupts,
                            status affects vector,
                            interrupt on all RCV
                            characters.

CTLTB   DEFB '18' H       ; WR0, Reset CH B
        DEFB '10' H       ; WR0, Reset External/Status
                            Interrupts
        DEFB '02' H       ; Pointer to WR2
        DEFB '00' H       ; Load Interrupt Vector
        DEFB '01' H       ; Pointer to WR1
        DEFB '14' H       ; Status affects vector
```

TRANSMIT DATA HANDLER

```
LABEL   SOURCE STATEMENT  COMMENTS

XMTINT  EX AF,AF'         ; Save Registers
        LD A,(T BUF)      ; Load Character
        OUT (SIODAT),A    ; Ship it Out
        EX AF,AF'         ; Restore Registers
        EI                ; Re-enable interrupts
        RETI
```

RECEIVE DATA HANDLER

```
RCVINT  EX AF,AF'         ; Save Registers
        IN A,(SIODAT)     ; Read Character
        LD (RBUF),A       ; Save in Memory
        EX AF,AF'         ; Restore Registers
        EI
        RETI
```

ERROR HANDLER

```
INTERR  EX AF,AF'         ; Save Registers
        LD A, '01' H      ; Set Pointer to
        OUT (SIOCTL),A    ; Reg. 1
        IN A,(SIODAT)     ;
        BIT 6,A           ;
        JR Z, FMER        ; Framing Error
        BIT 5,A           ;
        JR Z,ORER         ; Overrun Error
        JP PAER           ; Parity Error
        EX AF,AF'         ; Restore Registers
        EI
        RETI
```

MOSTEK®

USING THE MK3807 VCU IN A MICROPROCESSOR ENVIRONMENT

Application Note

INTRODUCTION

MK3807, the programmable CRT Video Control Unit (VCU), is a user programmable 40-pin n-channel MOS/LSI chip containing the logic functions required to generate all the timing signals for the formatting and presentation of interlaced or non-interlaced video data on a standard or non-standard CRT monitor.

In all the formatting, such as horizontal, vertical, and composite sync, characters per data row and per frame are totally user programmable. The data row counter has been designed to facilitate scrolling.

Programming is accomplished by loading seven 8 bit control registers directly off an 8 bit bidirectional data bus. Four register address lines and a chip enable line provide complete microprocessor compatibility for program controlled set up. The device can also be "self loaded" via an external PROM tied on the data bus. (See Figure 1).

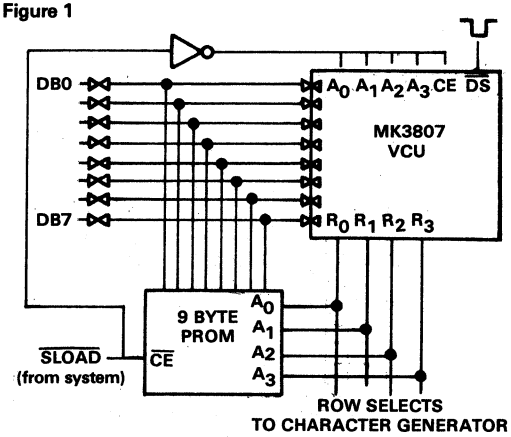
In addition to the seven control registers, two additional registers are provided to store the cursor character and row addresses for generation of the cursor video signal. The contents of these two registers can be read out onto the bus for update by the program or used by the microprocessor as two memory locations. (See Figure 2).

PROGRAM REGISTERS

The VCU contains 9 working registers (7 control registers and 2 data location registers).

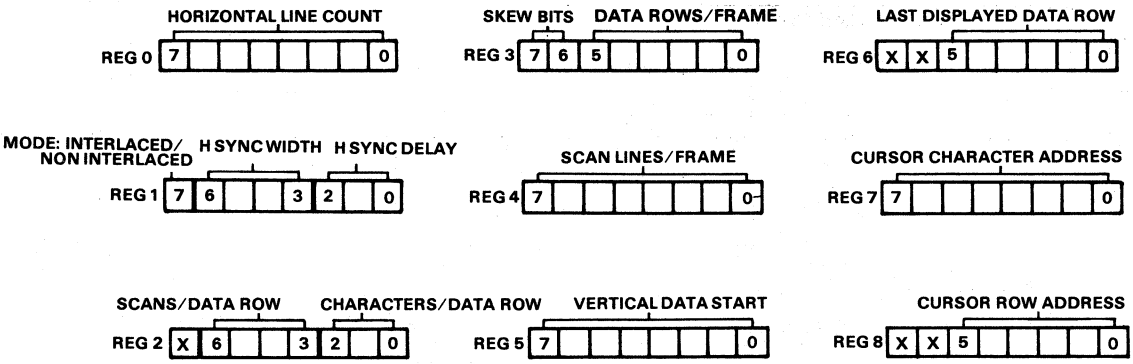
SELF LOADING SCHEME FOR VCU SET-UP

Figure 1



BIT ASSIGNMENT

Figure 2



V

REGISTER 0

This 8 bit register contains the number of character times for 1 horizontal period of the TV raster scan. For example, using American Standard Television (63.5 μs per line) at a character time of 500 ns, the value for this register would be 63.5 divided by .5 = 127. The number in this register is normally 1.25 times the number of characters per line displayed on this screen. The value loaded into this register is the binary equivalent of 126 (127-1). Since character times are counted from zero instead of one, the value loaded into this register is one less than the actual number of character times. (Refer to Figure 3 for timing diagrams).

REGISTER 1

This register contains 3 fields of information. The most significant bit (7) is the interlace bit. If this bit is set to a 1, Interlace mode is indicated; if set to a 0, Non-Interlace mode is indicated. The next 4 bits (6-3) define the number of character times for the width of the horizontal sync pulse. For example, using American Standard Television (4.5 μs) and a character time of 500 ns indicates that it would require 9 character times; therefore the binary equivalent 9 would be loaded in these bits. The least significant 3 bits (2-0) are used to specify the horizontal sync delay. This is commonly called the Front Porch and is the period between the end of active video to the beginning of the horizontal sync pulse. The value here is not critical and can be used to position the video horizontally on the screen.

REGISTER 2

This register contains both the number of characters to be displayed per line as well as the number of scans per character. Bit 7 is not used (B7 = X). Bits 6 through 3 define the number of scans per character. For example, using a 7 X 9 dot matrix character generator, the normal number of scans might be 12. Therefore, using 12 scans per character, the binary equivalent of eleven (12-1) is inserted into this field. The least significant 3 bits (2-0) contain a 3 bit code which defines the number of characters per line. The VCU is pre-programmed for 20, 32, 40, 64, 72, 80, 96, and 132 characters per line. The 3 bit binary number used in this field determines the particular format, for example, 80 characters being the 6th value would be coded as a binary 5 (101).

CHARACTERS/DATA ROW

DB2	DB1	DB0	
0	0	0	= 20
0	0	1	= 32
0	1	0	= 40
0	1	1	= 64
1	0	0	= 72
1	0	1	= 80
1	1	0	= 96
1	1	1	= 132

REGISTER 3

This register contains both the propagation delay compensation field (skew bits) as well as the data row fields. Bits 7 and 6 are used to adjust the blanking, cursor position and sync delay so as to compensate for either 0, 1 or 2 character time propagation delays of the character generator and the frame buffer RAM.

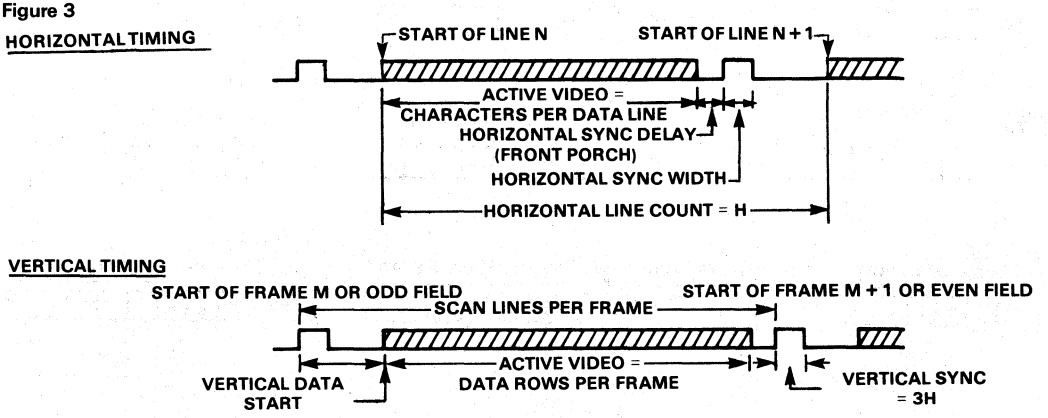
SKEW BITS

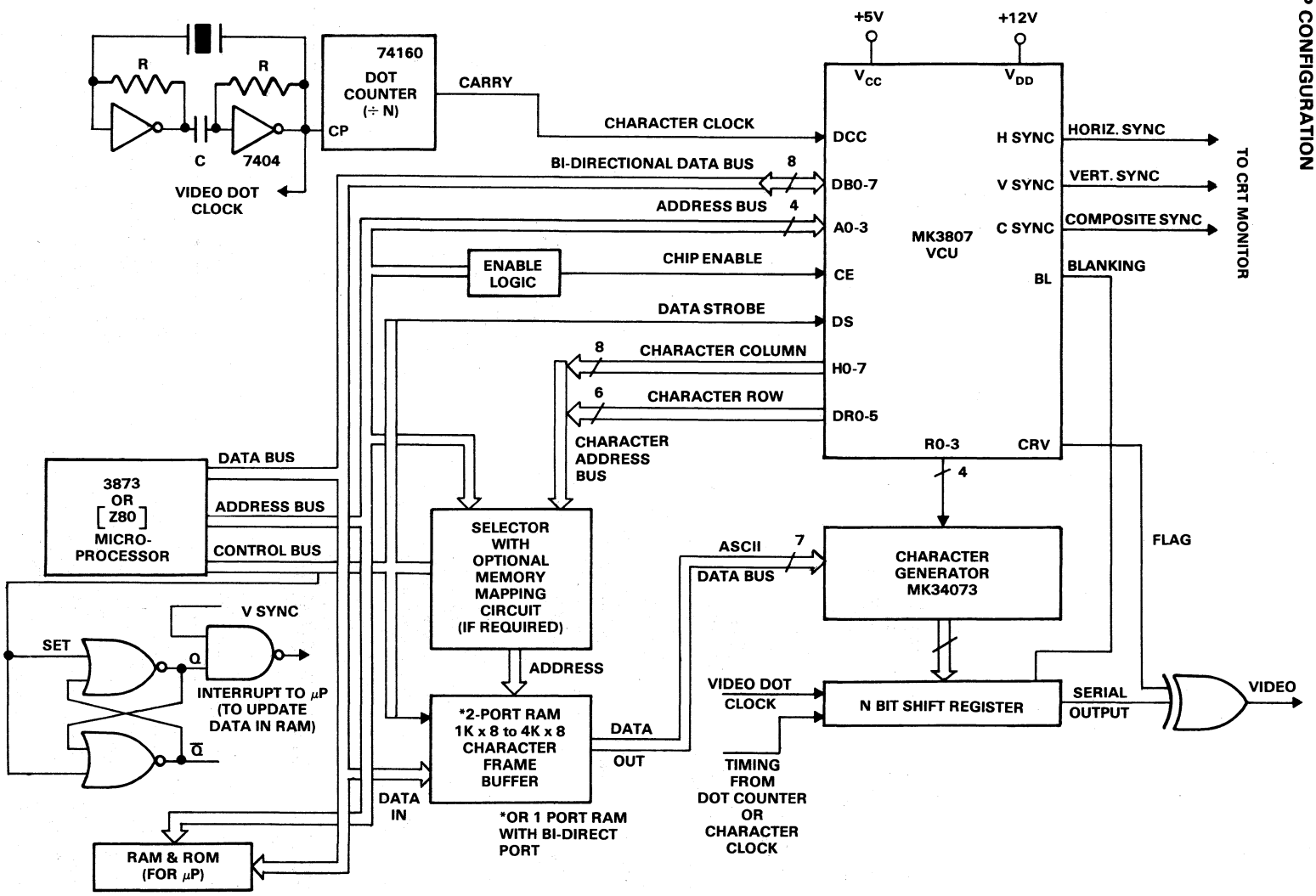
DB7	DB6	Sync/Blank Delay (Character Times)	Cursor Delay
0	0	0	0
1	0	1	0
0	1	2	1
1	1	2	2

The 6 least significant bits (5-0) define the number of data rows to be displayed on the screen. The number of rows begins at 000000 (single row) and continues to 111111 (64 rows).

HORIZONTAL AND VERTICAL TIMING

Figure 3





VCU μP CONFIGURATION
Figure 4

REGISTER 4

This 8 bit register defines the number of raster lines in the field (frame). Care should be taken when programming this register to make sure that the product of the scans per data row times the number of data rows is less than the number of raster scans. There are 2 methods of programming this register. In the interlaced mode, subtract 513 from the number of raster lines desired and divide by 2. For example, for 525 scans, the register should contain the number 6. In the non-interlaced mode, subtract the number 256 from the desired number of raster lines and divide by 2. For example, for 262 raster lines, the value is 3.

REGISTER 5

This register defines the number of raster lines between the beginning of the vertical sync pulse and the start of the first data row being displayed. Typically, values of 20 or 21 lines are used. Higher values can be used to position data lower on the screen to a maximum 255. This is called Vertical Data Start and is the sum of Vertical Sync and Vertical Scan Delay.

REGISTER 6

The least significant 6 bits (5-0) of this register define the last data row to be displayed on the screen. Bits 7 and 6 are not used. This feature is useful for both scrolling and positioning of data. For example, if the display were set for 24 data rows, normally row 0 would be on top of the screen and row 23 would be at the bottom. If the scroll register (register 6) contained the number 15, then row 15 would be at the bottom and row 16 would be at the top of the screen. Row 23 and row 0 would be contiguous in the middle of the screen.

REGISTER 7

This 8 bit register contains the character number at which the cursor is to be addressed. For example, if the last character of an 80 character per line display were to be cursored, the binary equivalent of 79 would be in this register.

REGISTER 8

The least significant 6 bits (5-0) of this register define the data row for the cursor, similar to Register 7.

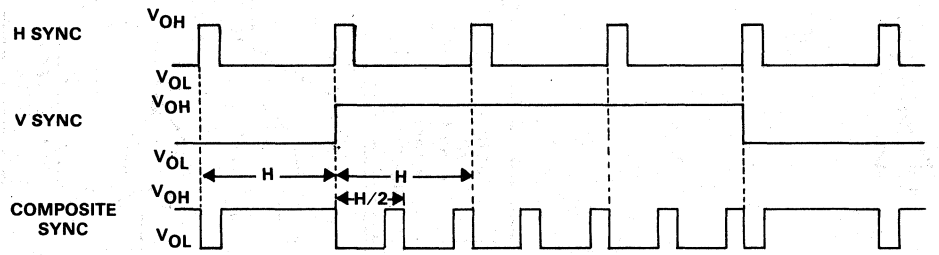
BASIC DISPLAY CONFIGURATION

Figure 4 shows the basic configuration for a Bus Oriented, microprocessor based, CRT display system utilizing Mostek's MK3807, the Programmable CRT Video Control Unit (VCU). Either a standard or a non-standard CRT monitor may be used. The user programmable VCU provides Horizontal Sync, Vertical Sync, and Composite Sync with serrations, to the monitor's sync deflection circuitry. (Figure 5 shows the composite sync timing). A serial output character generator provides video dot clock frequency data to the Z axis video input of the monitor.

In addition to the VCU, character generator, and shift register, the display system requires a crystal oscillator and a dot counter, typically consisting of two gates of a 7404 and a crystal as well as a 74160 (or equivalent) dot counter. The dot counter divisor (N) is set for the number of horizontal bits in the character plus the number of dots desired for spacing (i.e., for a 7 bit wide character + 2 dots of spacing N = 9). The carry output of the dot counter pulses once per character (character clock) and is fed into the MK3807 DCC (pin 12) input. This enables the VCU to keep track of the character positions as well as generate the entire video timing chain. At the same time the output of the oscillator is fed into the video dot clock input of the shift register of the Video Signal Generator.

An 8 bit bidirectional Data Bus (DB0-DB7), a 4 bit Address Bus (A0-A3), a Chip Enable, and a Data Strobe are used in programming the VCU. These buses connect to the microprocessor Data Bus and Address Bus. The VCU appears to the microprocessor as 16 memory or I/O locations. Page logic (high order address bit decoder) connects the Address Bus to the Chip Enable (CE) thereby determining where in the microprocessor memory space the VCU will be located. The Data Strobe (DS) signal is connected to the microprocessor Control Bus. This signal is used to read or write via the Data Bus, as well as to activate control functions.

COMPOSITE SYNC TIMING DIAGRAM
Figure 5



The VCU raster scan counter outputs (R0-R3) are connected directly to the raster line address inputs of the character generator. This 4 bit address indicates which raster line of the selected character is to be parallel loaded into the shift register. The bit pattern, along with the additional blank spaces, is then shifted out of the video output at the video dot clock rate. The blanking signal can be connected to retrace blanking logic to provide both horizontal and vertical blanking of the video signal to the CRT monitor. The load/shift signals for character generator logic can be derived from the outputs of the dot counter (74160) or taken directly from the character clock (DCC, pin 12 of 3807).

HOW TO USE ROW-COLUMN ADDRESSING

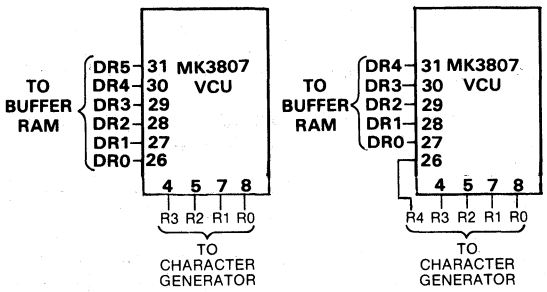
The VCU outputs the character position via the character counter outputs (H0-H7) and the data row counter outputs (DR0-DR5). These outputs define the character column and row location. They are used to address a character frame buffer RAM in which the frame image is stored. Since the VCU keeps counting horizontal addresses (H0-H7) during both horizontal and vertical blanking, dynamic RAMs may be refreshed.

Many advantages are realized using Row-Column (X-Y) Addressing. Among these are:

Oversize Characters

Character fonts with heights greater than 16 dots (raster lines) can be achieved. This is done by using the LSB of the row counter (DR0) as the MSB of the raster scan counter (R4), and then moving the remaining bits of the row counter down one bit (DR1 becomes DR0, etc.). This is achieved by connecting the pins of the VCU in a different configuration. No additional components are required. This is shown in Figure 6. In addition, the VCU must be programmed for twice the desired number of data rows; thus using the above configuration (Figure 6), 32 rows of data with up to 32 lines per character (or 16 rows of data with up to 64 lines per character) can be accomplished.

USING THE VCU WITH CHARACTER FONTS OF HEIGHTS GREATER THAN 16 DOTS (LINES)
Figure 6



A. 64 ROWS OF 16 LINES B. 32 ROWS OF 32 LINES

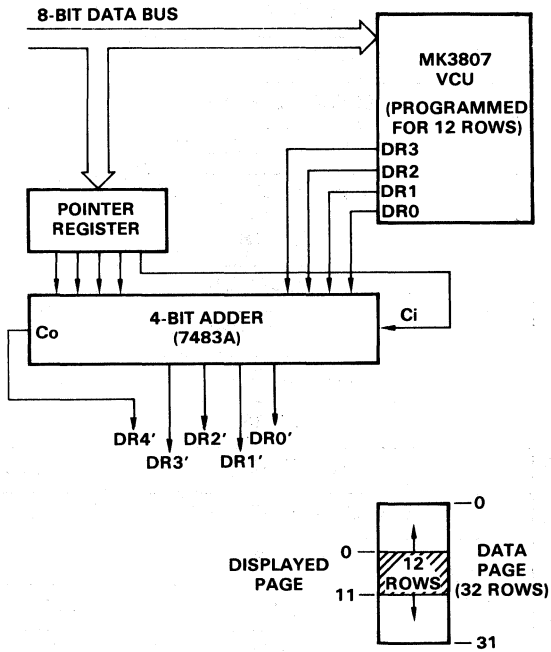
Page Scrolling

Scrolling a smaller page through a larger page (1K in 4K) can be done on a row by row basis. If the DR0-DR5 lines are offset by a pointer register, the smaller page can be moved up or down inside the larger page by the offset number of rows. This is shown in Figure 7. In this example, if the pointer register contains zero, the VCU will address the first 12 lines of the 32 line page. When the pointer register contains ten, the VCU will address rows 10 to 21. Thus, by loading the pointer register (from the microprocessor data bus), the display can scroll row by row through the data base.

Software Addressing

Most programmers use X — Y (row-column) addressing when writing software for CRT terminals. This makes it easier to blank the bottom line when scrolling, changing cursor positions, etc. Therefore, by having row-column addressing in the VCU, the address bus of the microprocessor can also have the preferred row-column addressing, and the two buses can be mapped together as shown in Figure 8. Without this feature, a software algorithm would have to convert a row-column address to binary address every time the microprocessor wanted to access the frame buffer. This algorithm usually requires a 16 bit multiplication. Thus the VCU, by utilizing row-column addressing, can save significant overhead and program execution time.

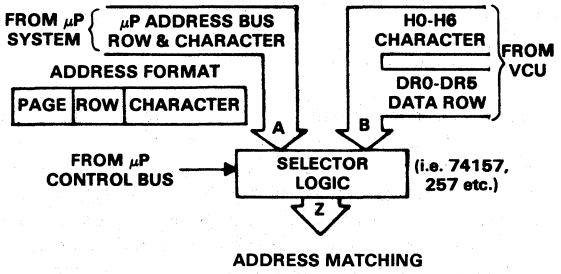
SCROLLING A 12 ROW PAGE THRU A 32 ROW PAGE
Figure 7



MEMORY MULTIPLEXING

The character column and character row outputs combine to form the character address bus. This bus, along with the microprocessor address bus, is connected to a 2 X 1 selector which addresses the character frame buffer RAM. Figure 8 shows the selector and the mapping for the various formats of the standard VCU. Numerous methods are available to build 2 X 1 selectors. One low-cost technique uses three 74157 or equivalent (74LS157 or 257, 9322, etc.) quad 2 X 1 selector chips. Figure 8 tabulates the mapping on to the microprocessor address bus into the selector with the DR and H lines of the VCU. The output of the selector (Z), is decomposed into two fields, row (Y) and column or character (X). Refer to Table 1.

ADDRESS BUS MAPPING
Figure 8



Memory Addressing

When the number of characters per row is non-binary, i.e. 80, addressing the frame buffer RAM is wasteful of memory. To solve this problem and still retain the advantages of row-column addressing, an address mapping is performed. The output of the selector (Z) is connected to another 74157 quad 2 X 1 selector chip or equivalent. Figures 6A, B, and C show the connection for 12 rows (1K), 24 rows (2K), and 48 rows (4K) of 80 characters. Figure 5 shows the mapping technique. The first 64 characters are mapped directly and the next 16 characters (H6 = 1) are mapped in a higher part of the RAM. The microprocessor address (row and column), is overlaid onto the VCU address bus (row and column) via the selector. The output of the selector maps into the frame buffer. Thus, every character is addressed by its row and column from both the microprocessor and the VCU. The

ADDRESS BUS MAPPING
Table 1

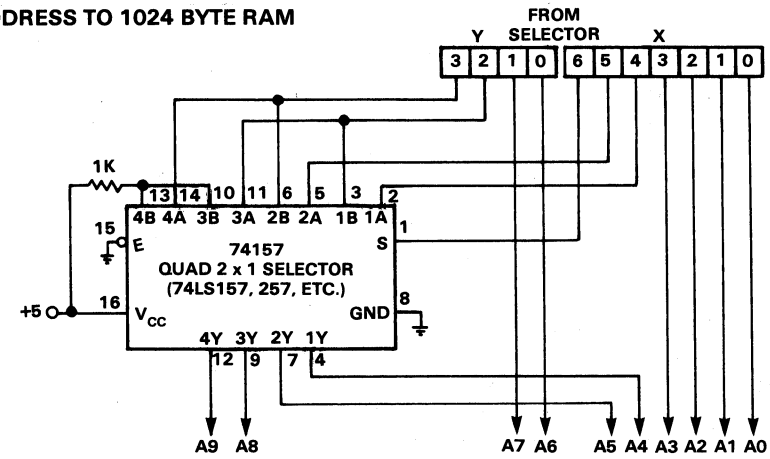
	SELECTOR													
μP ADDRESS BUS (UNUSED BITS ARE FOR PAGE LOCATION)	INPUT (A)	AB12	AB11	AB10	AB9	AB8	AB7	AB6	AB5	AB4	AB3	AB2	AB1	AB0
20 & 32 CHARACTERS/LINE														
FUNCTIONS		ROW								CHARACTER				
VCU OUTPUTS	INPUT (B)			DR5	DR4	DR3	DR2	DR1	DR0	H4	H3	H2	H1	H0
SELECTOR OUTPUTS	OUTPUT (Z)			Y5	Y4	Y3	Y2	Y1	Y0	X4	X3	X2	X1	X0
40 & 64 CHARACTERS/LINE														
FUNCTIONS		ROW							CHARACTER					
VCU OUTPUTS	INPUT (B)		DR5	DR4	DR3	DR2	DR1	DR0	H5	H4	H3	H2	H1	H0
SELECTOR OUTPUTS	OUTPUT (Z)		Y5	Y4	Y3	Y2	Y1	Y0	X5	X4	X3	X2	X1	X0
72, 80 & 96 CHARACTERS/LINE														
FUNCTIONS		ROW						CHARACTER						
VCU OUTPUTS	INPUT (B)	DR5	DR4	DR3	DR2	DR1	DR0	H6	H5	H4	H3	H2	H1	H0
SELECTOR OUTPUTS	OUTPUT (Z)	Y5	Y4	Y3	Y2	Y1	Y0	X6	X5	X4	X3	X2	X1	X0
132 CHARACTERS/LINE														
FUNCTIONS		ROW					CHARACTER							
VCU OUTPUTS	INPUT (B)	DR4	DR3	DR2	DR1	DR0	H7	H6	H5	H4	H3	H2	H1	H0
SELECTOR OUTPUTS	OUTPUT (Z)	Y4	Y3	Y2	Y1	Y0	X7	X6	X5	X4	X3	X2	X1	X0

MEMORY MAPPING CIRCUITS FOR 72 OR 80 CHARACTERS/LINE

Figure 9

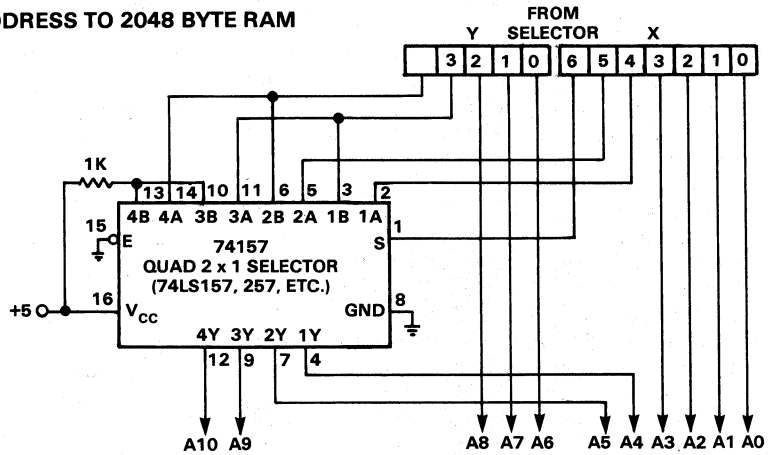
10 BIT BINARY ADDRESS TO 1024 BYTE RAM
12 LINES INTO 1K

Figure 9A



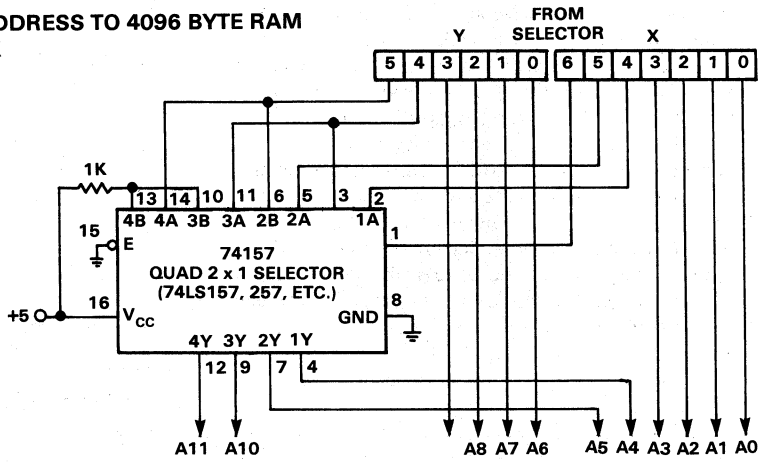
11 BIT BINARY ADDRESS TO 2048 BYTE RAM
24 LINES INTO 2K

Figure 9B



12 BIT BINARY ADDRESS TO 4096 BYTE RAM
48 LINES INTO 4K

Figure 9C



V

same memory location will be accessed whether the identical address originates from the microprocessor or VCU address bus.

OPERATION

The character frame buffer RAM is initially loaded via the microprocessor data and address buses (see Figure 1). After the microprocessor has loaded the character frame buffer RAM with a complete page, the selector flip-flop is switched (via the microprocessor control bus) so that the RAM is addressed by the character address bus of the VCU. In this mode the VCU operates independent of the microprocessor by addressing the character frame buffer RAM which sends the ASCII data to the CRT character generator. The selected character is then further decomposed by the raster scan counter (R0-R3), from the VCU, and loaded into the character generator shift register. This bit pattern is then serially shifted out at the video dot clock frequency and the data can be encoded so as to compose the video signal.

One possible way to change the data in the frame buffer (which is in microprocessor address space but physically separate) is: whenever the data in the character frame buffer is to be changed or updated, the microprocessor (via the control bus) sets an external flip-flop. The output of this flip-flop is ANDed with the vertical sync signal from the VCU. When this occurs an interrupt is generated to the microprocessor. This alerts the microprocessor to the fact that the vertical blanking interval has begun; it then switches the address selector (via control bus) so that the character frame buffer is now addressed by the microprocessor instead of the VCU. Since the system is in the vertical blanking interval, the screen is blank at this time. Using the American standard of 63.5 μs. per horizontal line and a typical value of 21 horizontal lines for the blanking interval gives the system 1.33 ms. in which the microprocessor can change data in the character frame buffer. If this time is not sufficient, the 1.33 ms. window will appear every 1/60 of a second, allowing the microprocessor to change part of the RAM data each time.

After the microprocessor has completed its updating of the character frame buffer RAM, it resets the external flip-flop (via the control bus) and switches the selector back to the character address bus of the VCU. Then the microprocessor goes about its normal system operation without being interrupted or having its throughput slowed down. This is because the VCU refreshes the CRT independently with the character frame buffer RAM, supplying the data, while the microprocessor operates at full speed with its own RAM and ROM. This method is more efficient for microprocessor throughput and control as opposed to having to DMA (cycle steal) or interrupt the processor continually, thereby reducing its throughput.

SYNC-LOCK

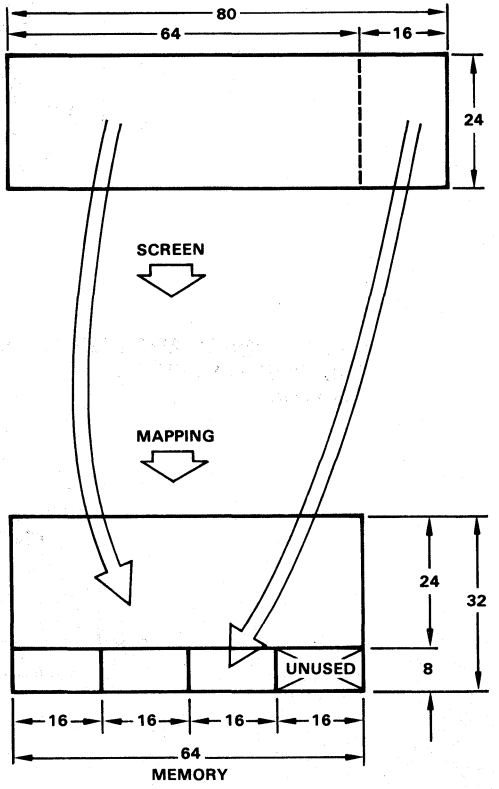
Some applications require adding alphanumeric characters (text) or graphics to the same screen as closed circuit or external (off-the-air) video. Figure 11 illustrates a simple technique of externally synchronizing the VCU using 2 chips (7474 and 7402 or equivalent). The external video can come from a closed circuit television system, off-the-air television, or some other video display system. The technique involves stopping the character clock (DCC) when the VCU sync occurs and restarting it when the external sync occurs. In this way, the VCU will be synchronized to the external video. One requirement for the reliable operation of this system is that the VCU horizontal and vertical sync rates must be programmed to be slightly faster than the external sync rate (i.e., the horizontal line counter register of the VCU must be programmed to be less than 63.5 μs., which is the American TV horizontal rate).

HOW TO PROGRAM THE MK3807 VCU

In order to pick the correct video dot clock frequency and to program the registers in the VCU, it is first necessary to determine several key parameters. Among these parameters are: the vertical refresh rate; the number of horizontal raster lines per frame; the number of characters per line, and the format of the characters.

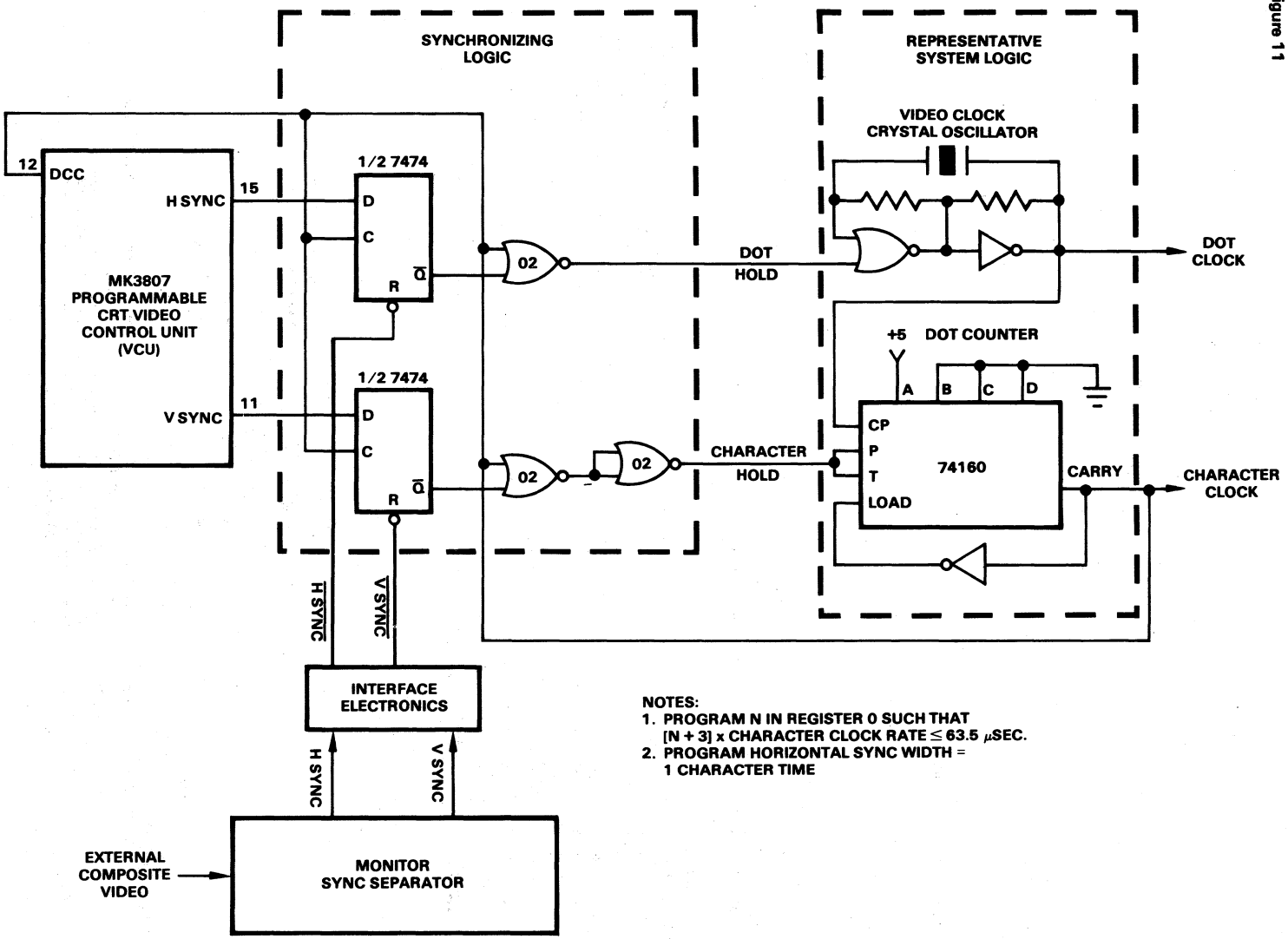
Tables 2A, B list work sheets which give the designer an

ADDRESS COMPRESSION SCHEME FOR 80 CHARACTERS/LINE
Figure 10



MK3807 EXTERNAL VIDEO SYNCHRONIZING CIRCUIT

Figure 11



NOTES:
1. PROGRAM N IN REGISTER 0 SUCH THAT [N + 3] x CHARACTER CLOCK RATE ≤ 63.5 μSEC.
2. PROGRAM HORIZONTAL SYNC WIDTH = 1 CHARACTER TIME

orderly method of determining the frequencies and register contents from the above parameters. In order to demonstrate its use, typical examples will be shown.

EXAMPLE FOR 80 CHARACTERS BY 24 ROWS

A 7 X 9 character matrix is chosen as it is the most popular for the display of both upper and lower case characters. Also, a non-interlaced system is chosen. The character block of 9 X 12 allows for a 2 dot space between characters and a 3 line space between data rows. The impact of the character block size on the horizontal frequency and the video clock rate will be shown below. A frame refresh rate of 60Hz is chosen for this example. These numbers can be modified for 50Hz systems.

This system will have 24 rows of data and 80 characters per data row. Thus, there are (24 X 12) 288 active scan lines.

The monitor chosen for this example is capable of accepting a composite video signal or separate TTL horizontal and vertical sync pulses. The sum of the horizontal sync delay (front porch), horizontal sync pulse, and horizontal scan delay (back porch) is the horizontal blanking interval. This interval is required as a window in the horizontal scan period to allow retrace. The retrace time is internal to the CRT monitor; this time is a function of monitor horizontal scan components. This time, at a minimum, is the time it takes the display to return from the right to the left hand side of the display. The retrace time is less than the horizontal blanking interval. The horizontal blanking interval is normally about 20% of the total horizontal scanning period. See Figure 12 for horizontal and vertical timing, and Figure 13 for derived register bit assignments.

In an 80 character per data row system, this would give 20 character times for the sum of the Front Porch, Horizontal Sync Pulse, and Back Porch. In the example of table 2C, a sum of 22 character time is used to illustrate that some flexibility exists in the choice of these parameters.

The vertical scanning frequency can be obtained by counting the total number of horizontal lines. The total number of scan lines generated for a vertical field equals the number of data rows times the number of lines per character, plus the vertical sync delay, plus the vertical sync pulse, plus the vertical scan delay.

Vertical sync delay is the number of scan lines delay before vertical sync. Vertical sync pulse width should be expressed in scan line units. The VCU is fixed at the standard vertical sync width of 3 horizontal scan lines (3H). Scan line delay is the delay between vertical sync and the display information in scan line units. The sum of the vertical sync and the 2 delays in the vertical blanking interval is normally 5% to 8% of the total number of scan lines.

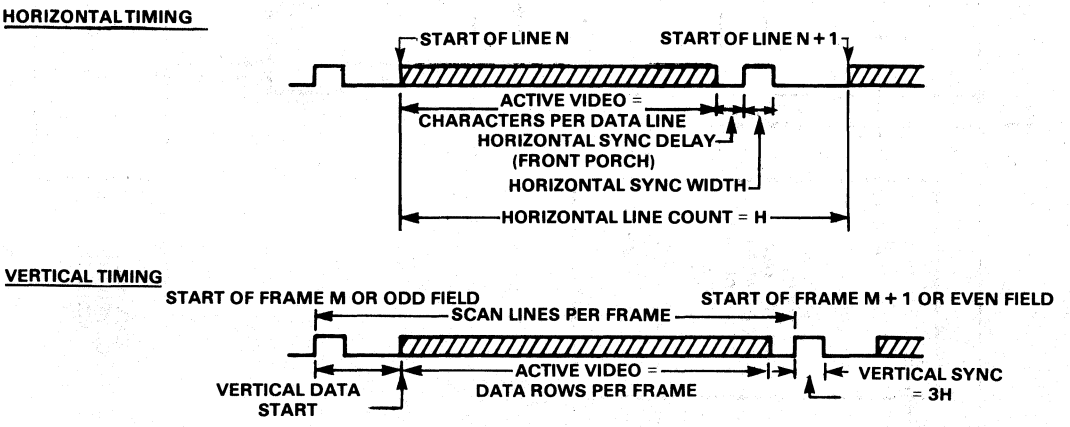
The vertical period (for 60Hz vertical refresh rate) can be calculated as: 1 divided by 60Hz = 16.67 ms.

Thus, the vertical blanking period (at 8%) equals 1.3 ms. In the example of table 2C, the sum of the "Front Porch, Vertical Sync Pulse, and Back Porch" is 22 scan lines long. Again, some flexibility exists in the choice of these parameters.

Adding the displayed lines (24 X 12 = 288) plus the vertical blanking interval (0 + 3 + 19 = 22), 310 horizontal scan lines are required. These 310 lines must be repeated 60 times a second (every 16.67 ms.). Thus 18,600 horizontal scan lines per second is the horizontal frequency. It can now be seen that any further increase in the number of scan lines per data character block will cause a direct increase in the horizontal frequency, possibly to a point beyond the monitor's specification.

HORIZONTAL AND VERTICAL TIMING

Figure 12



MK3807 VCU WORK SHEET
Table 2A

1. H CHARACTER MATRIX (No. of Dots): ..... _____
2. V CHARACTER MATRIX (No. of Horiz. Scan Lines): ..... _____
3. H CHARACTER BLOCK (Step 1 + Desired Horiz. Spacing = No. in Dots): ..... _____
4. V CHARACTER BLOCK (Step 2 + Desired Vertical Spacing = No. in Horiz. Scan Lines): ..... _____
5. VERTICAL FRAME (REFRESH) RATE (Freq. in Hz): ..... _____
6. DESIRED NO. OF CHARACTER ROWS: ..... _____
7. TOTAL NO. OF ACTIVE "VIDEO DISPLAY" SCAN LINES (Step 4 x Step 6 = No. in Horiz. Scan Lines): ..... _____
8. VERT. SYNC DELAY (No. in Horiz. Scan Lines): ..... _____
9. VERT. SYNC (No. in Horiz. Scan Lines; T = _____ μs*): ..... _____
10. VERT. SCAN DELAY (No. in Horiz. Scan Lines; T = _____ ms*): ..... _____
11. TOTAL VERTICAL FRAME (Add steps 7 thru 10 = No. in Horiz. Scan Lines): ..... _____
12. HORIZONTAL SCAN LINE RATE (Step 5 x step 11 = Freq. in KHz): ..... _____
13. DESIRED NO. OF CHARACTERS PER HORIZ. ROW: ..... _____
14. HORIZ. SYNC DELAY (No. in Character Time Units; T = _____ μs**): ..... _____
15. HORIZ. SYNC (No. in Character Time Units; T = _____ μs**): ..... _____
16. HORIZ. SCAN DELAY (No. in Character Time Units; T = _____ μs**): ..... _____
17. TOTAL CHARACTER TIME UNITS IN (1) HORIZ. SCAN LINE (Add Steps 13 thru 16): ..... _____
18. CHARACTER RATE (Step 12 x Step 17 = Freq. in MHz): ..... _____
19. CLOCK (DOT) RATE (Step 3 x Step 18 = Freq. in MHz): ..... _____

*Vertical Interval
**Horizontal Interval

V

MK3807 VCU WORK SHEET
Table 2B

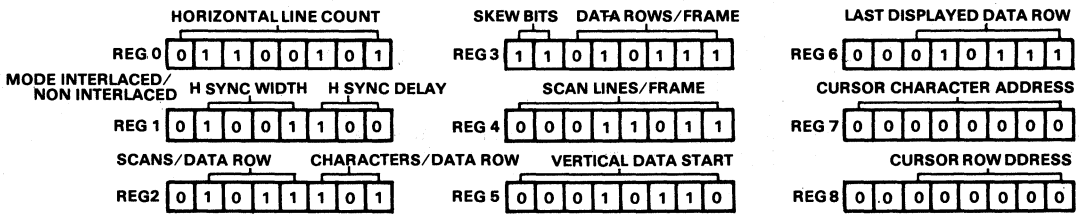
REG. #	ADDRESS A3—A0	FUNCTION	BIT ASSIGNMENT	HEX.	DEC.
0	0000	HORIZ. LINE COUNT _____	[ ][ ][ ][ ][ ][ ][ ][ ]	____	____
1	0001	INTERLACE _____ H SYNC WIDTH _____ H SYNC DELAY _____	[ ][ ][ ][ ][ ][ ][ ][ ]	____	____
2	0010	SCANS/DATA ROW _____ CHARACTERS/ROW _____	[X][ ][ ][ ][ ][ ][ ][ ]	____	____
3	0011	SKEW CHARACTERS _____ DATA ROWS _____	[ ][ ][ ][ ][ ][ ][ ][ ]	____	____
4	0100	SCANS/FRAME _____ X = _____	[ ][ ][ ][ ][ ][ ][ ][ ]	____	____
5	0101	VERTICAL DATA START = 3 + VERTICAL SCAN DELAY: SCAN DELAY _____ DATA START _____	[ ][ ][ ][ ][ ][ ][ ][ ]	____	____
6	0110	LAST DISPLAYED DATA ROW (= DATA ROWS)	[X][X][ ][ ][ ][ ][ ][ ]	____	____

MK3807 VCU WORK SHEET
Table 2C

1. H CHARACTER MATRIX (No. of Dots): 7
2. V CHARACTER MATRIX (No. of Horiz. Scan Lines): 9
3. H CHARACTER BLOCK (Step 1 + Desired Horiz. Spacing = No. in Dots): 9
4. V CHARACTER BLOCK (Step 2 + Desired Vertical Spacing = No. in Horiz. Scan Lines): 12
5. VERTICAL FRAME (REFRESH) RATE (Freq. in Hz): 60
6. DESIRED NO. OF CHARACTER ROWS: 24
7. TOTAL NO. OF ACTIVE "VIDEO DISPLAY SCAN LINES" (Step 4 x Step 6 = No. in Horiz. Scan Lines): 288
8. VERT. SYNC DELAY (No. in Horiz. Scan Lines): 0
9. VERT. SYNC (No. in Horiz. Scan Lines; T = 161.29 μs*): 3
10. VERT. SCAN DELAY (No. in Horiz. Scan Lines; T = 1.02 ms*): 19
11. TOTAL VERTICAL FRAME (Add steps 7 thru 10 = No. in Horiz. Scan Lines): 310
12. HORIZONTAL SCAN LINE RATE (Step 5 x step 11 = Freq. in KHz): 18.6
13. DESIRED NO. OF CHARACTERS PER HORIZ. ROW: 80
14. HORIZ. SYNC DELAY (No. in Character Time Units; T = 2.11 μs**): 4
15. HORIZ. SYNC (No. in Character Time Units; T = 4.74 μs**): 9
16. HORIZ. SCAN DELAY (No. in Character Time Units; T = 4.74 μs**): 9
17. TOTAL CHARACTER TIME UNITS IN (1) HORIZ. SCAN LINE (Add Steps 13 thru 16): 102
18. CHARACTER RATE (Step 12 x Step 17 = Freq. in MHz): 1.8972
19. CLOCK (DOT) RATE (Step 3 x Step 18 = Freq. in MHz): 17.0748

*Vertical Interval
**Horizontal Interval

BIT ASSIGNMENT
Figure 13



XTAL Frequency

At a frequency of 18.6 kHz, a scan line takes 53.76 μs. In this time, 102 characters (80 displayed + 22 blanked) have to be accessed. Thus the character time is 527.06 ns (53.76 μs/102). Since each character is 9 dots in this example (7 character and 2 blank), the dot period is 58.56 ns (527.06 ns/9). The inverse of the dot period is the video dot clock XTAL frequency. For this example, the video dot clock XTAL is 1/58.56 ns = 17.0748 MHz (53.76 μs/102). Since each character is 9 dots in this example (7 character and 2 blank), the dot period increases in the video clock rate, possibly to a point beyond the monitor's specification.

A more detailed example, using 40 character by 12 row format, follows.

Having chosen the display format and display monitor, the actual settings for the VCU registers can now be established. See Table 2C.

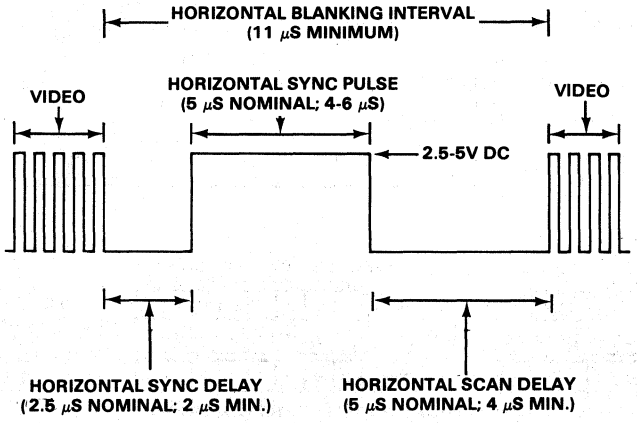
EXAMPLE FOR 40 CHARACTER BY 12 ROWS

Using the VCU worksheet (Table 2A), steps 1 and 2 determine the character matrix. In this example, a 7 X 9 dot matrix will be used, thus in step 1, 7 dots are used horizontally and in step 2, 9 scan lines are used vertically. This defines the character size (other character sizes might be 5 X 7 etc.). Steps 3 and 4 determine the character block size. The character block is composed of the character matrix along with both the horizontal and vertical blank spaces between characters. Step 3 shows the H character block for this example to be 7 dots from step 1 plus 2 additional dots for blank space, giving a total of 9. Step 4 shows the vertical height (V character block) being 9 lines from step 2, plus 3 additional raster lines for vertical spacing, giving a total of 12. The next parameter is the vertical frame refresh rate and this example uses the American Standard of 60Hz (in this example the non-interlace mode will also be used).

As this example uses twelve rows of data, step 6 indicates 12. Step 7 determines the number of active video display raster scan lines. This is determined by taking the number of raster scan lines from step 4 and multiplying that by the number of data rows in step 6, thus giving us the number of displayed horizontal scan lines. In this example, multiply 12 raster lines per data row by 12 data rows to give 144 active video raster scan lines.

The next portion of this example is dependent upon the characteristics of the video monitor being used. For the purposes of this example, a standard sync driven video monitor using RS-170 non-interlace sync is used. In accordance with the standard for this monitor, the vertical sync pulse width will be between 180 and 200 μs. with 190 μs. as the nominal value. In addition, the vertical blanking interval, which is made up of the vertical sync pulse and the 2 delays , is defined as being 1 ms. minimum. The same monitor specification defines the horizontal sync pulse width as being between 4 and 6 μs. with 5 μs. as the nominal horizontal sync pulse width. In addition, the horizontal sync delay or front porch is defined as 2.5 μs.

MONITOR HORIZONTAL TIMING
Figure 14



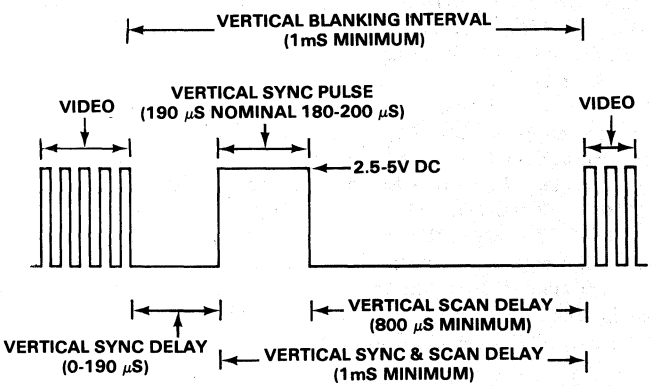
nominally with a 2 μs. minimum. At the same time, the horizontal blanking interval, which is composed of the front porch, horizontal sync pulse, and the back porch is defined as 11 μs. minimum. See Figures 14 and 15.

The monitor characteristics determine the values for steps 9 and 10. Step 9 lists the vertical sync pulse width. The VCU has a fixed vertical sync pulse width of 3 horizontal raster scan lines (3H). Later, the period of a horizontal raster scan line will be determined and it will be verified that this meets the RS-170 specification. Enough time must be allowed for vertical retrace and some blanking at the top of the screen. This is indicated in step 10 as the vertical scan delay. The VCU can be programmed for a vertical scan delay between 0 and 255 raster scan lines to allow utilization of various types of monitors, as well as to position the data vertically on the screen. For purposes of this example, a vertical scan delay of 19 raster lines is chosen. After the horizontal period is determined, it can be verified that these values comply with the specification. Step 11 is the total number of raster lines per frame or, in other words, the number of raster lines per vertical refresh time. Normally, this will be determined by adding to the number of displayed scan lines, the vertical sync pulse width, the vertical scan delay, and the vertical sync delay which has not yet been determined. However, in this case, since the example uses a standard monitor, it is possible to work backwards. Therefore, for step 11 we will enter 262 raster lines per frame (a typical number of raster lines/field of a standard monitor). Now work backwards to step 8 and determine the vertical sync delay. This is the number of raster lines between the last displayed video raster line and the beginning of vertical sync. Subtracting 144, 19, and 3 from 262 leaves 96; thus for step 8, 96 horizontal lines is the vertical sync delay. We have now determined the vertical timing waveform for this example. The next part of the example is to determine the horizontal scan line rate or how many raster lines per second will be displayed. This is determined by multiplying the vertical frame refresh rate from step 5—in this case 60 frames per second—by the total number of raster lines per frame from step 11, in this case 262. The product will be 15,720 raster lines per second. This is the horizontal scan rate. The horizontal period is determined by taking the inverse of horizontal scan rate, 1 divided by 15,720 Hz, which is 63.6132 μs. This is the time of 1 horizontal raster line. This information is now used to go back and check that the specifications in steps 9 and 10 are met. Step 9 lists 3 horizontal lines as the vertical sync pulse width. 3 X 63.6132 μs. yields 190.84 μs. This is the nominal value specified for the monitor. Step 10 lists the vertical scan delay as 19 raster lines which multiplied by 63.61 μs. yields 1.21 ms; thus the values picked for the above parameters meet the specification for the monitor.

In step 13 the desired number of active display characters per horizontal data row is listed. 40 characters per row have been chosen. Steps 14, 15 and 16 are now selected using the horizontal period and the monitor specifications. Step 14 is the horizontal sync delay or front porch, in this case 2 character times. The period of a character will be determined later in this example, which will be used to verify that this parameter meets the RS-170 specification given earlier. In step 15 the horizontal sync width is chosen to be 4 character times, and in step 16 the horizontal scan

V

MONITOR VERTICAL TIMING
Figure 15



delay is also chosen to be 4 character times. Step 17 is the total number of character times per horizontal scan line and this sum is determined by adding steps 13 through 16; thus we add 40+2+4+4=50 character times per horizontal scan line. In step 18 the character rate is determined by multiplying the horizontal line rate of step 12 by the total character units per horizontal line; thus, 15,720 X 50 = 786,000 characters per second. The character period is the inverse of the character rate; thus 1 over 786,000 yields a character period of 1.272 μs. This information is used to verify steps 14, 15, and 16. In step 14 the horizontal sync delay was chosen as 2 character units. 2 times 1.272 μs yields 2.54 μs. Step 15, the horizontal sync width was 4 character units. 4 times 1.272 μs. yields 5.089 μs. and similarly, in step 16, four character units also are 5.089 μs. These three values are in agreement with the specification for the monitor. The next step is to determine the video dot clock frequency. It is determined by multiplying the number of dots per character from step 3 by the character rate in step 18: 9 X 786 KHz = 7.074 MHz. Thus, the crystal frequency required for this example is 7.074 MHz and the dot clock counter divisor N is 9 (from step 3).

Register Programming

Register 0 (Horizontal Line Count) determines the total number of character units per horizontal line. From step 17 we have determined that there would be 50 character units per line. This register is loaded with (N — 1), the decimal number 49.

Register 1 contains 3 fields. The first field is the most significant bit, and this determines the interlaced or non-interlaced mode of operation. This example uses the non-interlaced mode; therefore, bit 7 is loaded with a 0. The next field is the horizontal sync pulse width, and this field is bits 6 through 3. Step 15 determines that the horizontal sync width is 4 character times. Therefore the binary equivalent of 4 is loaded into these bits. Thus bits 6 through 3 are loaded with 0100. The third field is the horizontal sync delay. Step 14 determines that this is 2 character time units. Therefore, bits 2 through 0 are loaded with 010.

Register 2 contains 2 fields, with the most significant bit unused. Bits 6 through 3 determine the scans per data row. In this example from step 4, there will be 12 raster lines per data row, and, from the VCU data sheet note, this is an N + 1 register. Therefore the decimal number eleven is loaded into bits 6 through 3. The second field is characters per data row, bits 2 through 0. In this example, 40 active characters per data row were chosen. The VCU data sheet specifies that 010 in this field will give 40 characters per data row; thus bits 2 through 0 are loaded with 010.

Register 3 also contains 2 fields. The first field, bits 7 and 6, are the skew bits. These bits allow the hardware designer to

MK3807 VCU WORK SHEET

1. H CHARACTER MATRIX (No. of Dots): 7
2. V CHARACTER MATRIX (No. of Horiz. Scan Lines): 9
3. H CHARACTER BLOCK (Step 1 + Desired Horiz. Spacing = No. in Dots): 9
4. V CHARACTER BLOCK (Step 2 + Desired Vertical Spacing = No. in Horiz. Scan Lines): 12
5. VERTICAL FRAME (REFRESH) RATE (Freq. in Hz): 60
6. DESIRED NO. OF CHARACTER ROWS: 12
7. TOTAL NO. OF ACTIVE "VIDEO DISPLAY SCAN LINES" (Step 4 x Step 6 = No. in Horiz. Scan Lines): 144
8. VERT. SYNC DELAY (No. in Horiz. Scan Lines): 96
9. VERT. SYNC (No. in Horiz. Scan Lines; T = 190.84 μs*): 3
10. VERT. SCAN DELAY (No. in Horiz. Scan Lines; T = 1.21 ms*): 19
11. TOTAL VERTICAL FRAME (Add steps 7 thru 10 = No. in Horiz. Scan Lines): 262
12. HORIZONTAL SCAN LINE RATE (Step 5 x step 11 = Freq. in KHz): 15.72
13. DESIRED NO. OF CHARACTERS PER HORIZ. ROW: 40
14. HORIZ. SYNC DELAY (No. in Character Time Units; T = 2.54 μs**): 2
15. HORIZ. SYNC (No. in Character Time Units; T = 5.09 μs**): 4
16. HORIZ. SCAN DELAY (No. in Character Time Units; T = 5.09 μs**): 4
17. TOTAL CHARACTER TIME UNITS IN (1) HORIZ. SCAN LINE (Add Steps 13 thru 16): 50
18. CHARACTER RATE (Step 12 x Step 17 = Freq. in MHz): ..... .786
19. CLOCK (DOT) RATE (Step 3 x Step 18 = Freq. in MHz): 7.074

*Vertical Interval

**Horizontal Interval

use a slower buffer RAM memory and allow compensation for slower character generator access times. In the example shown, as well as most typical applications, these bits are set for 2 character time delays; therefore bit 7 and bit 6 will both contain a 1. The other field is data rows per frame, bits 5 through 0. In Step 6 there are 12 data rows per frame, and the VCU data sheet specifies that this is an N + 1 register. Thus the decimal number eleven is loaded in bits 5 through 0.

Register 4 determines the number of horizontal raster lines per frame. From this example, step 11, specifies that there are 262 raster lines per frame. The VCU data sheet specifies that there are two modes of loading this register. In the non-interlace mode (this example) the equation 2X + 256 is equal to 262. Thus, X is equal to 3. The decimal number 3 is loaded into register 4.

Register 5 is the vertical start of data. From steps 9 and 10 in the example, the vertical data start is 22 raster lines; thus the decimal number 22 is loaded into register 5.

Register 6 is the last displayed data row. This register, which is used for multi-line scrolling and for initialization purposes is set to the same data as in register 3, the same data rows per frame. Thus, the decimal number eleven is loaded into register 6.

The following will illustrate the use of register 6 for multi-line scrolling:

> Using 12 rows of data with row 0 on top of the screen and row 11 on the bottom and as programmed in register 6 with eleven, this will be the case. Now, if another number is programmed into register 6, such as 5, data row 5 will be on the bottom of the screen, while data row 6 will be on the top followed by data row 7, 8, through to 11, which will then be followed by row 0 through 5.

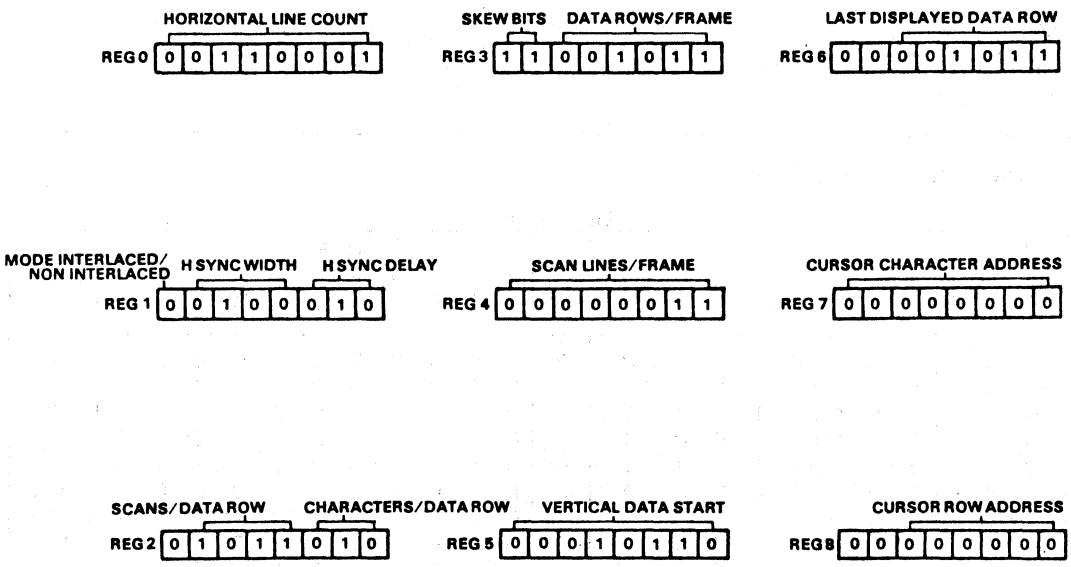
Register 7 is the cursor character address. It is initialized to 0; thus it is now set to the beginning of the data row.

Register 8 is also initialized to 0. This is the cursor row address and is set to the top data row. The 2 cursor addresses (X-Y) coincide at the upper left hand corner of the screen. See the VCU work sheet on page 16.

The above is only a typical example of how to determine the frequencies, program the frequencies, and program the registers of the VCU. This is shown for illustrative purposes only and designers/programmers should determine these values for their specific CRT requirements.

V

BIT ASSIGNMENT CHART



APPLICATION NOTES

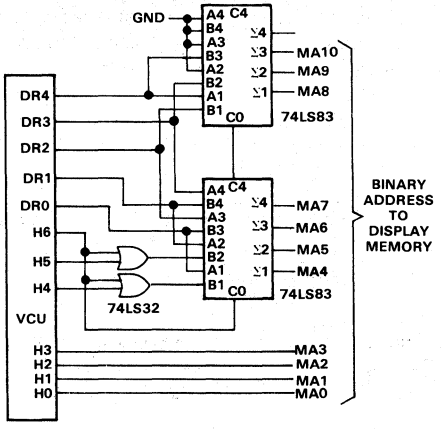
Conversion of Row Column to Binary Address

With only slightly more complicated circuitry than required by memory mapping, the row column addressing outputs of the VCU may be readily changed to binary address outputs. For data formats that use 48 or 80 visible characters per data row, this can be done by the addition of two 74LS83's and a 74LS32 (or equivalent) in some formats or by the addition of the one 256x8 PROM. Figure 16 below shows the implementation for an 80 character by 24 data row display using the adders. Figure 17 is an implementation using a bipolar PROM.

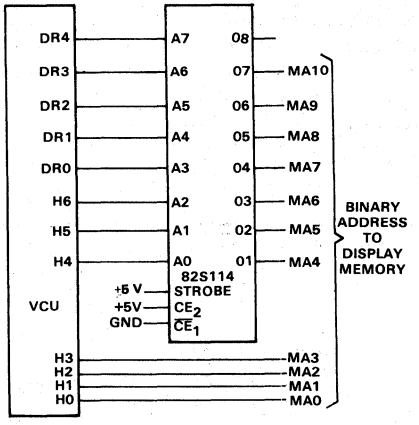
In essence the adders are used to add groups of 16. Since there are 5 groups of 16 in each data row of 80 characters, the adders effectively multiply the data row count (DR0-DR4) by 5 to obtain the starting binary address for each row. This is done by adding DR0-DR4 to itself shifted two positions to the left. Within each data row, H6, H5, and H4 are used to add from 0 to 4 groups of 16. The PROM configuration is merely a table look-up implementation of the adder configuration.

The PROM configuration can be programmed to provide binary addresses for any number of groups of 16 characters per data row (i.e., 48, 80, 96, 112, 144, 160). Table 3 shows some typical mapping for an 80x24 display.

80x24 DISPLAY WITH BINARY ADDRESS USING 74LS83 ADDERS
Figure 16



80x24 DISPLAY WITH BINARY ADDRESS USING 256x8 PROM
Figure 17



TYPICAL MAPPING OF 80x24 DISPLAY
Table 3

ADDRESS TABLE

DR4	DR3	DR2	DR1	DR0	H6	H5	H4	H3	H2	H1	H0	ROW	COL	MA10	MA9	MA8	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0	ADDR. (BIN)
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	16	0	0	0	0	0	0	1	0	0	0	0	16
0	0	0	0	0	1	0	0	1	1	1	1	0	79	0	0	0	0	1	0	0	1	1	1	1	79
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	80
0	0	0	0	1	1	0	0	1	1	1	1	1	79	0	0	0	1	0	0	1	1	1	1	1	159
0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	0	1	0	1	0	0	0	0	0	160
0	0	0	1	0	1	0	0	1	1	1	1	2	79	0	0	0	1	1	1	0	1	1	1	1	239
0	0	0	1	1	0	0	0	0	0	0	0	3	0	0	0	0	1	1	1	1	0	0	0	0	240
1	0	1	1	1	0	0	0	0	0	0	0	23	0	1	1	1	0	0	1	1	0	0	0	0	1840
1	0	1	1	1	1	0	0	1	1	1	1	23	79	1	1	1	0	1	1	1	1	1	1	1	1919

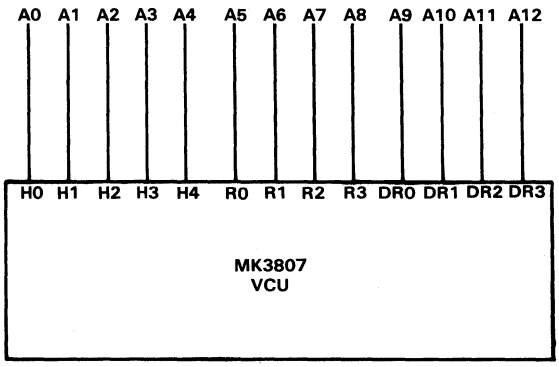
USING THE VCU FOR A 256 X 256 DOT GRAPHIC DISPLAY

The VCU can be used for dot matrix graphic displays as well as alphanumeric displays. The following is an example of a 256 x 256 dot matrix graphic display using the raster line counter outputs (R0-R3) as part of the RAM addressing.

For this example the character width (the dot counter divisor) should be 8 dots. The VCU should be programmed (See Figure 18) for:

Characters per data row = 32
Scans per data row = 16
Data rows per frame = 16

USING THE VCU FOR A 256 x 256 DOT GRAPHIC DISPLAY
Figure 18



USING THE VCU FOR MORE THAN 128 CHARACTERS PER ROW AND MORE THAN 32 ROWS

Due to pin limitation, the most significant character count output of the VCU is multiplexed with the most significant bit of the data row counter. When the horizontal line count is greater than 128, this output (H7/DR5) automatically becomes H7. On the surface, this creates a limitation of no more than 32 data rows.

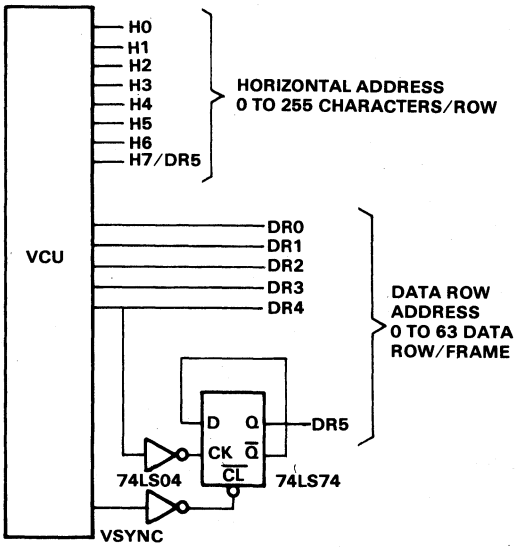
In actual fact, the row column addressing of the VCU permits the display of more than 128 characters per row and more than 32 rows per frame with only two inverters and one D-type flip flop. In the following example, the display format will be 132 characters per row by 35 data rows.

The horizontal row address will appear on outputs H0 to H7. Data row outputs DR0 to DR4 will provide five of the six bits required for the data row addressing. The circuit shown in Figure 19 will generate the required sixth row address bit.

There are many other applications of the VCU other than the alphanumeric CRT terminal as shown above.

Because of the speed and flexibility of the device, it can be used to generate television pictures (with gray scale and color), facsimile, slow-scan TV, frame storage, scan conversion, etc. Since the VCU generates composite sync (with serrations), the serial video can be combined with the composite sync to produce composite video (RS-170).

USING THE VCU FOR MORE THAN 128 CHARACTERS PER ROW AND MORE THAN 32 ROWS
Figure 19



V

MOSTEK®

USE OF THE MK3805 CLOCK/RAM

Application Note

INTRODUCTION

Many microprocessor applications require a real time clock and/or memory that can be battery powered with very low power drain. A typical application might be an automobile trip computer, where the clock could provide the time of day and the memory would be used to retain vital information when the ignition switch is off. The interfacing technique needs to be kept as simple as possible so as to minimize the required overhead in software, and it should minimize the number of pins required in order that other I/O requirements can be efficiently accommodated.

FEATURES

Mostek's CLOCK/RAM microcomputer peripheral chip satisfies all of these requirements. The device, designated MK3805, contains a real-time clock/calendar, 24 bytes of static RAM, and an on-chip oscillator, and communicates serially with the microcomputer via a simple interface protocol. The MK3805 is fabricated using CMOS technology, thus ensuring very low power consumption.

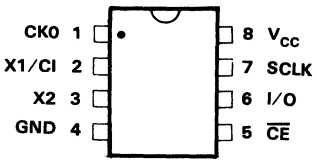
The real-time clock/calendar provides all timekeeping functions. It contains registers for seconds, minutes, hours, day, date, month, and year. The end of the month date is automatically adjusted for months with less than 31 days. The clock operates in either the 24 hour or 12 hour format with an AM/PM indicator. Since the MK3805 is designed to interface to a microcomputer, the alarm function is easily accommodated in the microcomputer, should it be required.

The on-chip oscillator provides the clock source for the clock/calendar. It incorporates a programmable divider so that a wide variety of crystal frequencies can be accommodated. The oscillator also has an output available that is designed to serve as the clock generator for the microcomputer. A separately programmable divider provides several different output frequencies for any given crystal frequency. This feature can eliminate the need for a separate crystal or external oscillator for the microcomputer, thereby reducing system cost.

Interfacing the CLOCK/RAM with a microcomputer is greatly simplified using asynchronous serial communication. Only 3 lines are required to communicate with the CLOCK/RAM: (1) $\overline{CE}$ (chip enable), (2) I/O (data line), and (3) SCLK (shift register clock). Data can be transferred to and from the CLOCK/RAM one byte at a time, or in a burst of up to 24 bytes.

PINOUT DIAGRAM

Figure 1



PIN	NAME	DESCRIPTION
1	CKO	System clock (output).
2	X1/CI	Crystal or external clock (input).
3	X2	Crystal (input).
4	GND	Ground.
5	$\overline{CE}$	Chip enable (input, active low).
6	I/O	Data I/O (input/output).
7	SCLK	Shift register clock (input).
8	V_{CC}	Positive supply voltage.

PINOUT DESCRIPTION

Figure 1 is a pinout diagram of the MK3805. It is packaged in an 8-pin DIP to conserve PC board space. A brief description of the function of each pin is listed.

TECHNICAL DESCRIPTION

Figure 2 is a block diagram of the CLOCK/RAM chip. The main components are the oscillator and divider, the real time clock/calendar, the static RAM, the command register and logic, the control register and logic, and the serial shift register.

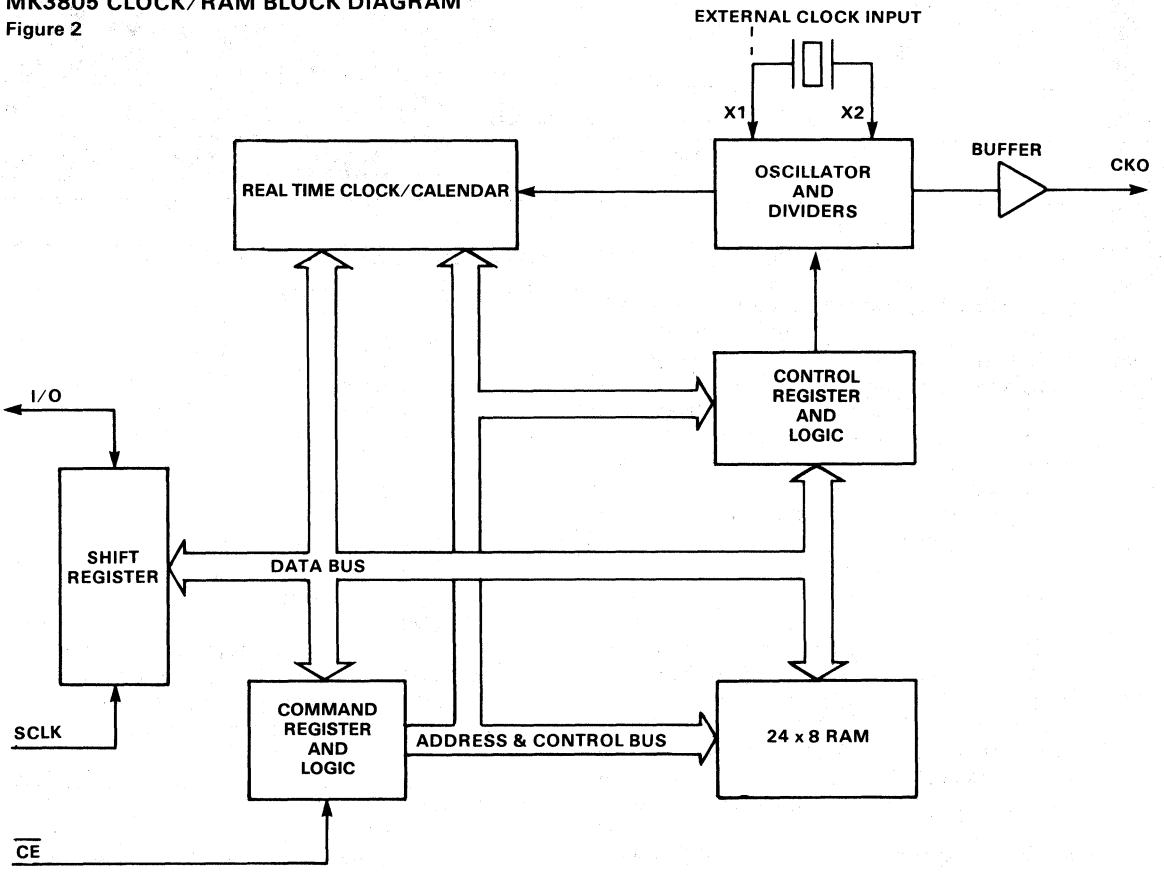
The shift register is used to communicate with the outside world. Data on the I/O line is either input or output on each shift register clock pulse when the chip is enabled. If the chip is in the input mode, the data on the I/O line is input to the shift register on the rising edge of SCLK. If the chip is in the output mode, data is shifted out onto the I/O line on the falling edge of SCLK.

The command register receives the first byte input by the shift register after $\overline{CE}$ goes true (low). This byte must be the command byte and will direct further operations within the CLOCK/RAM. The command specifies whether subsequent transfers will be read or written, and what register or RAM location will be involved.

V

MK3805 CLOCK/RAM BLOCK DIAGRAM

Figure 2



The control register has bits defined which control the divider for the internal real-time clock and the external system clock. One bit serves as the write protect control flag, preventing accidental write operations during power-up or power-down situations.

The real-time clock/calendar is accessed via seven registers. These registers contain seconds, minutes, hours, day, date, month, and year information. Certain bits within these registers also control a run/stop function, 12/24 hour clock mode, and indicate AM or PM (12 hour mode only). These registers can be accessed either randomly in byte mode, or sequentially in burst mode.

The static RAM is organized as 24 bytes of 8-bits each. They can be accessed either randomly in byte mode, or sequentially in burst mode.

The reader should refer to the MK3805 data sheet for operating specifications and detailed timing information.

DATA TRANSFERS

Data transfer is accomplished under control of the $\overline{CE}$ and SCLK inputs by an external microcomputer. Each transfer consists of a single byte (COMMAND) input followed by a single or multiple byte input or output (as defined by the command byte).

The general format for the command byte is shown in Figure 3. The most significant bit (bit 7) must be a logical 1; bit 6 specifies a clock function if logical 0, or a RAM function if logical 1. Bits 1-5 specify the clock register(s) or RAM location(s) to be accessed. The least significant bit (bit 0) specifies a write operation if a logical 0 or a read operation if a logical 1.

In the clock burst mode, all clock, calendar, and control registers are transferred beginning with register 0 (seconds) and ending with register 7 (control). Unless terminated early, this burst mode requires that $\overline{CE}$ be true and 72 SCLK cycles be supplied. This mode may be terminated at any time by taking $\overline{CE}$ false. This mode is specified by setting all address bits in the command byte to a logical 1.

In the RAM burst mode, all RAM locations are transferred beginning with location 0 and ending with location 23 (017H). Unless terminated early, this burst mode transfer

Figure 3

COMMAND, REGISTER, DATA FORMAT SUMMARY

I. GENERAL COMMAND FORMAT:

7	6	5	4	3	2	1	0
1	R/$\overline{C}$	A_4	A_3	A_2	A_1	A_0	R/$\overline{W}$

II. CLOCK COMMAND FORMAT:

	7	6	5	4	3	2	1	0
SEC	1	0	0	0	0	0	0	R/$\overline{W}$
MIN	1	0	0	0	0	0	1	R/$\overline{W}$
HR	1	0	0	0	0	1	0	R/$\overline{W}$
DATE	1	0	0	0	0	1	1	R/$\overline{W}$
MONTH	1	0	0	0	1	0	0	R/$\overline{W}$
DAY	1	0	0	0	1	0	1	R/$\overline{W}$
YEAR	1	0	0	0	1	1	0	R/$\overline{W}$
CONTROL	1	0	0	0	1	1	1	R/$\overline{W}$
CLOCK BURST	1	0	1	1	1	1	1	R/$\overline{W}$

III. RAM COMMAND FORMAT:

RAM 0	1	1	0	0	0	0	0	R/$\overline{W}$
• • •								
RAM 23	1	1	1	0	1	1	1	R/$\overline{W}$
RAM BURST	1	1	1	1	1	1	1	R/$\overline{W}$

IV. CLOCK PROGRAMMING MODEL:

	7	6	5	4	3	2	1	0
00-59	STOP	10 SEC			SEC			
00-59	10 MIN				MIN			
01-12 00-23	12/24	0	10 A/P	HR	HR			
01-28/29 01-30 01-31	T_1	0	10 DATE		DATE			
01-12	0	0	0	10 M	MONTH			
01-07	T_2	0	0	0	0	DAY		
0-99	10 YEAR				YEAR			
	WP	C_1	C_0	X_4	X_3	X_2	X_1	X_0

NOTES:

WP	Write protect.
X_4 - X_0	Program dividers for real time clock.
C_1 - C_0	Program dividers for clock output.
T_2 - T_1	Test bits (normally set to 0).

V

requires that $\overline{CE}$ be true and 200 SCLK cycles be supplied. This mode may be terminated at any time by taking $\overline{CE}$ false. This mode is specified by setting all address bits in the command byte to a logical 1.

Refer to Figure 3 for a summary of the command, register, and data formats.

POWER-ON STATES

When the MK3805 is first powered up, all eight clock registers come up to a pre-defined state. These are listed below. The RAM locations contain unspecified data.

Clock:

Seconds	00	
Minutes	00	
Hours	00	
Date	01	
Month	01	
Day	01	
Year	00	
Halt	1	(clock stopped)
12/24 Hour	0	(24 hour mode)

Control:

Write Protect	1	(protect on)
C0 & C1	01	(CKO = crystal frequency /2)
X3 & X4	00	(crystal frequency is binary: 2^h)
X0, X1 & X2	000	(divide by 2^{23})

SERIAL TIMING

The timing sequence for data transfer with the CLOCK/RAM is started when $\overline{CE}$ goes low (see Figure 4). After $\overline{CE}$ goes low, the next 8 SCLK cycles will input the command byte of the proper format. If the most significant bit (bit 7) is a logical 0, the command byte will be ignored, as will all SCLK cycles until $\overline{CE}$ goes high and returns low to signify the start of a new transfer. Command bits are input on the rising edge of SCLK.

Input data will be input on the rising edge of the next 8 SCLK cycles (per byte if burst mode is specified). Additional SCLK cycles will be ignored, should they inadvertently occur.

Output data will be output on the falling edge of the next 8 SCLK cycles (per byte if burst mode is specified). Additional SCLK cycles will retransmit the information, thereby permitting continuous transmission of clock information for certain applications.

A data transfer will terminate if $\overline{CE}$ goes high, and the transfer must be reinitiated by the proper command when $\overline{CE}$ goes low again. The I/O pin will be in the high impedance state when $\overline{CE}$ is high.

DESIGN EXAMPLE

As a demonstration of the software and hardware interfacing for the CLOCK/RAM chip, the design of a demonstration model used for electronic shows is given here. The hardware used included a standard CRT terminal, an MK38P73 single chip microcomputer, the MK3805 CLOCK/RAM chip, and some miscellaneous parts to interface to the CRT. Refer to Figure 5 for a schematic of the circuit used. Note how simple the design is. The MK3805 interfaces directly to the MK38P73 via 3 pins, and it provides the clock input to the MK38P73 via a fourth pin.

HARDWARE DESCRIPTION

The MK38P73 is an 8-bit single-chip microcomputer with 4 parallel ports, a serial port, 128 bytes of RAM, and 2K bytes of EPROM (in the form of a piggy back 2716). Because the serial communications with the CLOCK/RAM use a simple shift register type interface, the serial port of the 38P73 is not used here. It remains free for serial communications with the CRT.

The MK3805 is interfaced to the microcomputer via port 4. This is done to take advantage of the $\overline{STB}$ line associated with that port. The $\overline{STB}$ line goes low for a short time after each output to port 4 instruction is executed. This normally would be used to strobe data into an output device attached to the port. In this example, the $\overline{STB}$ line provides the SCLK pulse to the CLOCK/RAM shift register to clock data into and out of the chip. By using this line, toggling another port bit to strobe data in and out is not required. Such an interface to other microcomputers is straightforward.

The CLOCK/RAM chip also provides the clock source for the microcomputer. By selecting a crystal frequency of 3.6864 MHz and setting the CKO divider to divide by 1, the serial port on the MK38P73 operates at standard Baud rates (9600, 4800, 2400, 1200, etc.).

The 75150 and 1489 chips convert the TTL level signals output by the microcomputer to RS-232 levels in order that the circuit can be interfaced to a standard CRT.

SOFTWARE DESCRIPTION

The heart of the software is the subroutine labeled 'CLKRAM'. This subroutine provides all the necessary software interfacing to the CLOCK/RAM.

Before calling the subroutine, the necessary parameters must be set up in the proper registers. The ISAR is used as a pointer to where the data is to be read from or written to in the MK38P73 RAM area.

The scratchpad register 'CMD' must contain the command to be sent to the CLOCK/RAM. (See the description of the command given earlier.)

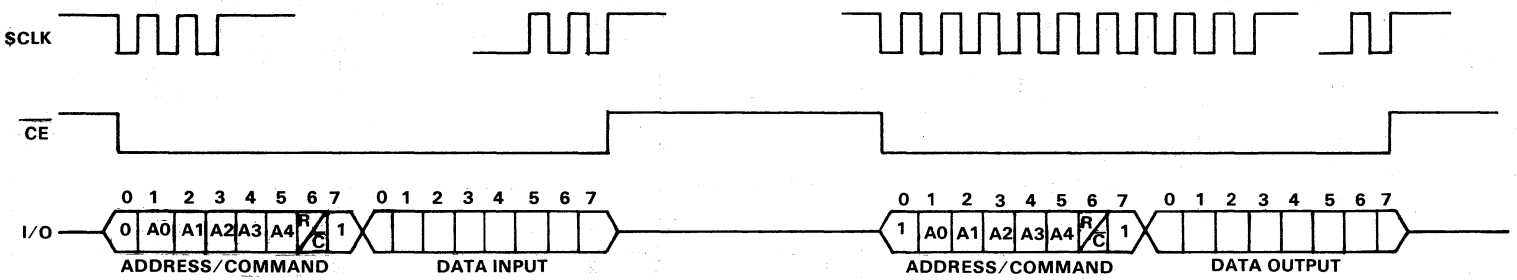
The bit pattern for enabling the CLOCK/RAM must be

MK3805 RAM DATA TRANSFER SUMMARY

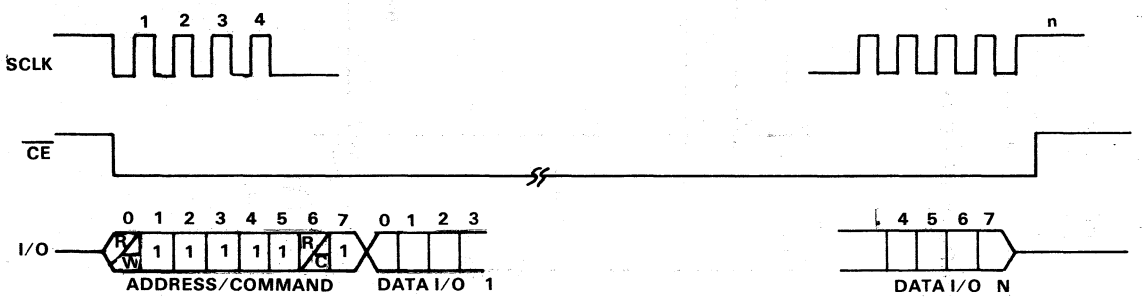
Figure 4

Notes: 1) Data input sampled on rising edge of clock.
2) Data output changes on falling edge of clock.
3) Rising edge of CE terminates operation and resets command register.

I. SINGLE BYTE TRANSFER



II. BURST MODE TRANSFER



FUNCTION	N	n
CLOCK	8	72
RAM	24	200

stored in the scratchpad register 'CHIPEN'. This bit pattern should contain a logic 1 in the bit position that corresponds to the port 4 line tied to the CLOCK/RAM $\overline{CE}$ pin. All other bits should be 0. This technique allows multiple serial microcomputer peripheral chips to be tied together with common I/O and SCLK lines, with a separate port line for each device $\overline{CE}$.

The subroutine also provides an option for using the port 4 pins not used by the CLOCK/RAM interface for any other purpose. To accomplish this option, a copy of whatever is written to port 4 by other routines must be kept in the scratchpad register 'PT4IMG'. This option is not used in this example.

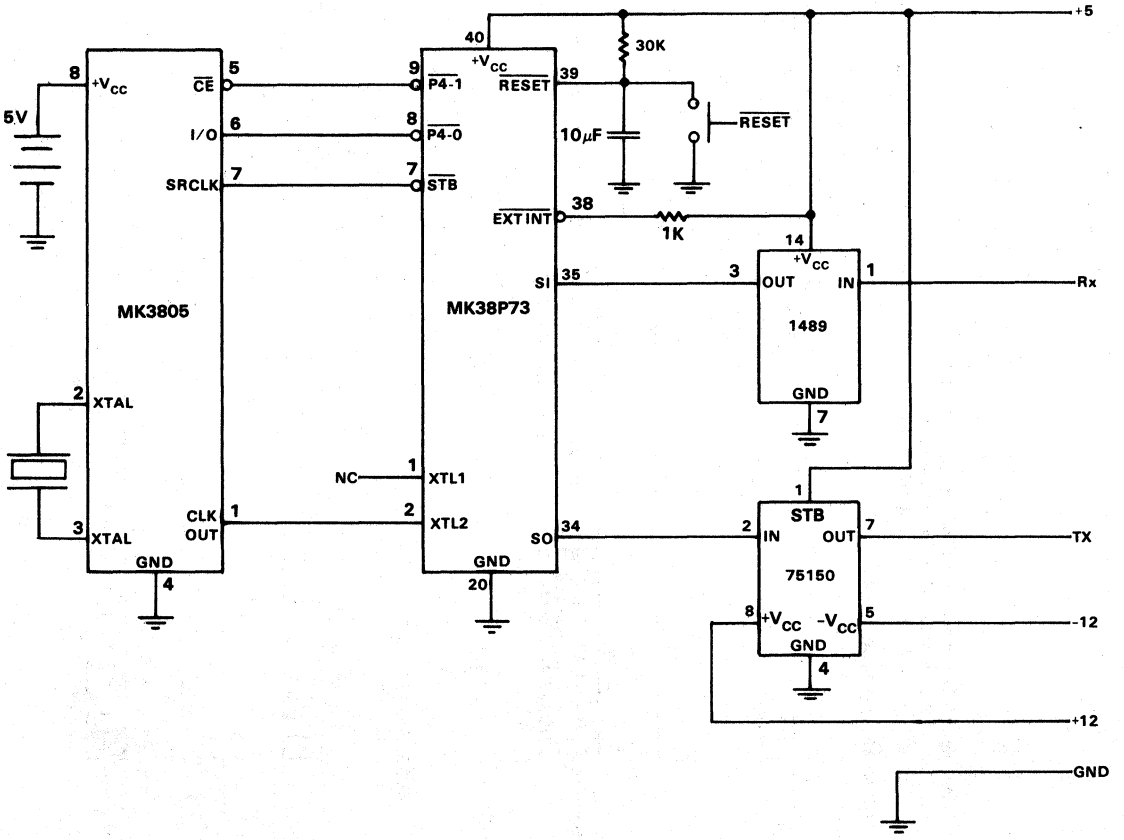
The main demonstration routine (listing 1) is quite basic. Its purpose is to print the features of the CLOCK/RAM on the CRT, then read the clock and display its contents once every second. A reentry point is provided in order that the clock/calendar settings may be changed after power up. (See the flowchart in Figure 6.)

When power is applied to the microcomputer, it resets and begins execution of the program at location 0000H. The code at this point initializes the system and checks for valid CLOCK/RAM data. This condition is indicated by the state of the write protect bit in the control byte. If the bit is set to a logical 1, then the CLOCK/RAM has also just been powered up. This indicates that the registers contain invalid data and should be initialized before continuing. If the bit is reset to a logical 0, the CLOCK/RAM did not just power up, and the data in its registers should be valid.

After the clock data is verified, the routine prints a message consisting of CLOCK/RAM features. The timer is then set to interrupt once every 1/36 second so that the time, etc., may be updated on the CRT screen. The routine then just waits for an interrupt from the timer or the keyboard.

When a timer interrupt occurs, the service routine checks to see if 1 second has elapsed since the last service. If not, it resets the timer and returns to the wait for interrupt state. If 1 second has gone by, the routine proceeds to erase the

SCHEMATIC OF DEMONSTRATION CIRCUIT
Figure 5



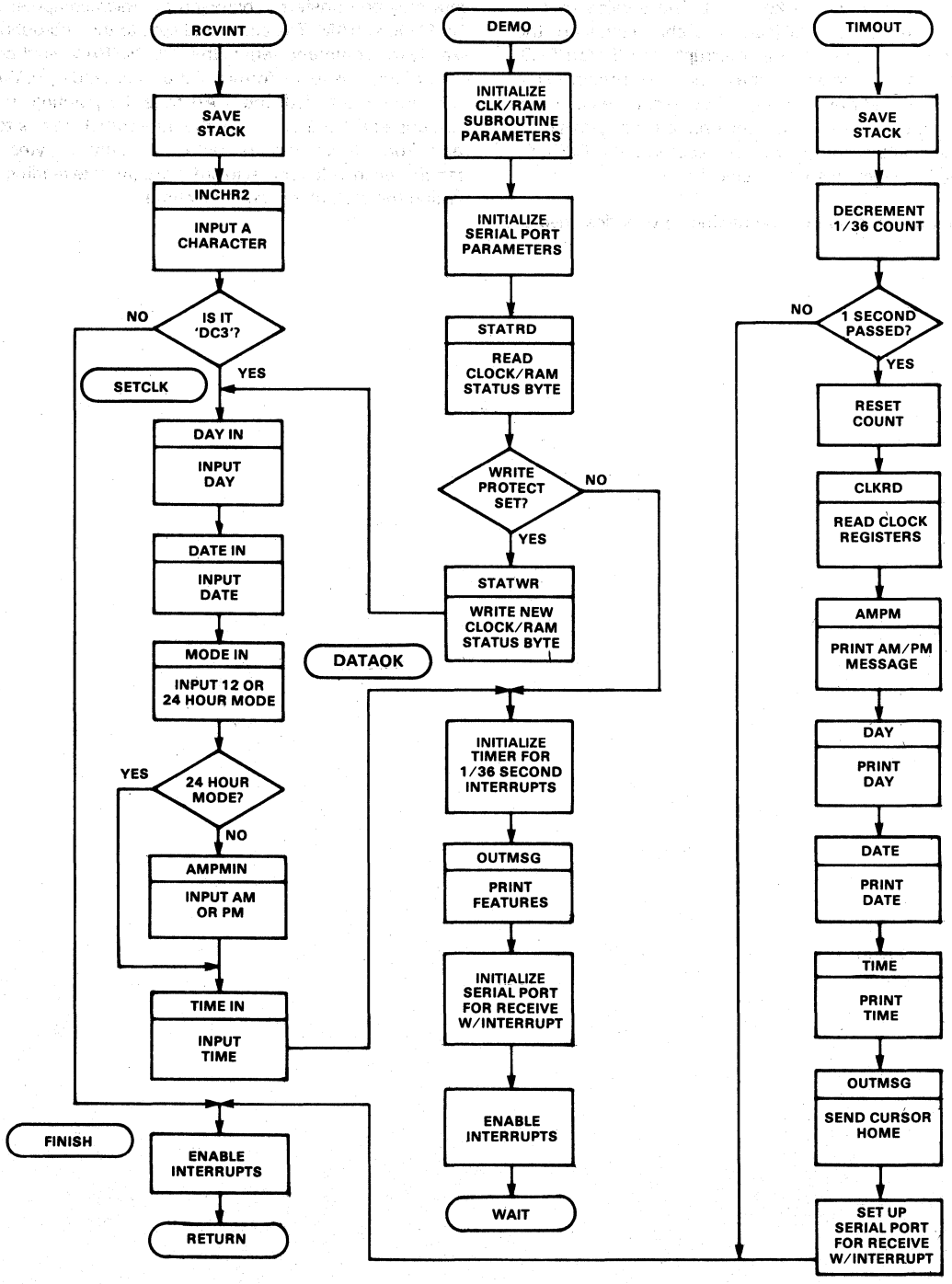
time, etc., from the top of the screen and will print new data obtained from the CLOCK/RAM. The timer is then reset and returns to the wait for interrupt state.

When a receiver interrupt occurs, the serial port contains a valid character from the keyboard. The service routine checks to see if it is a 'DC3' (control-S) character. If not, the routine returns to the 'wait for' interrupt state. If it is a 'DC3' character, the routine goes to the clock set entry point of the main routine and the user is allowed to set the clock and calendar values. The main routine entered in this fashion is executed similarly to a power on reset withthe CLOCK/RAM write protect bit set to a logical 1.

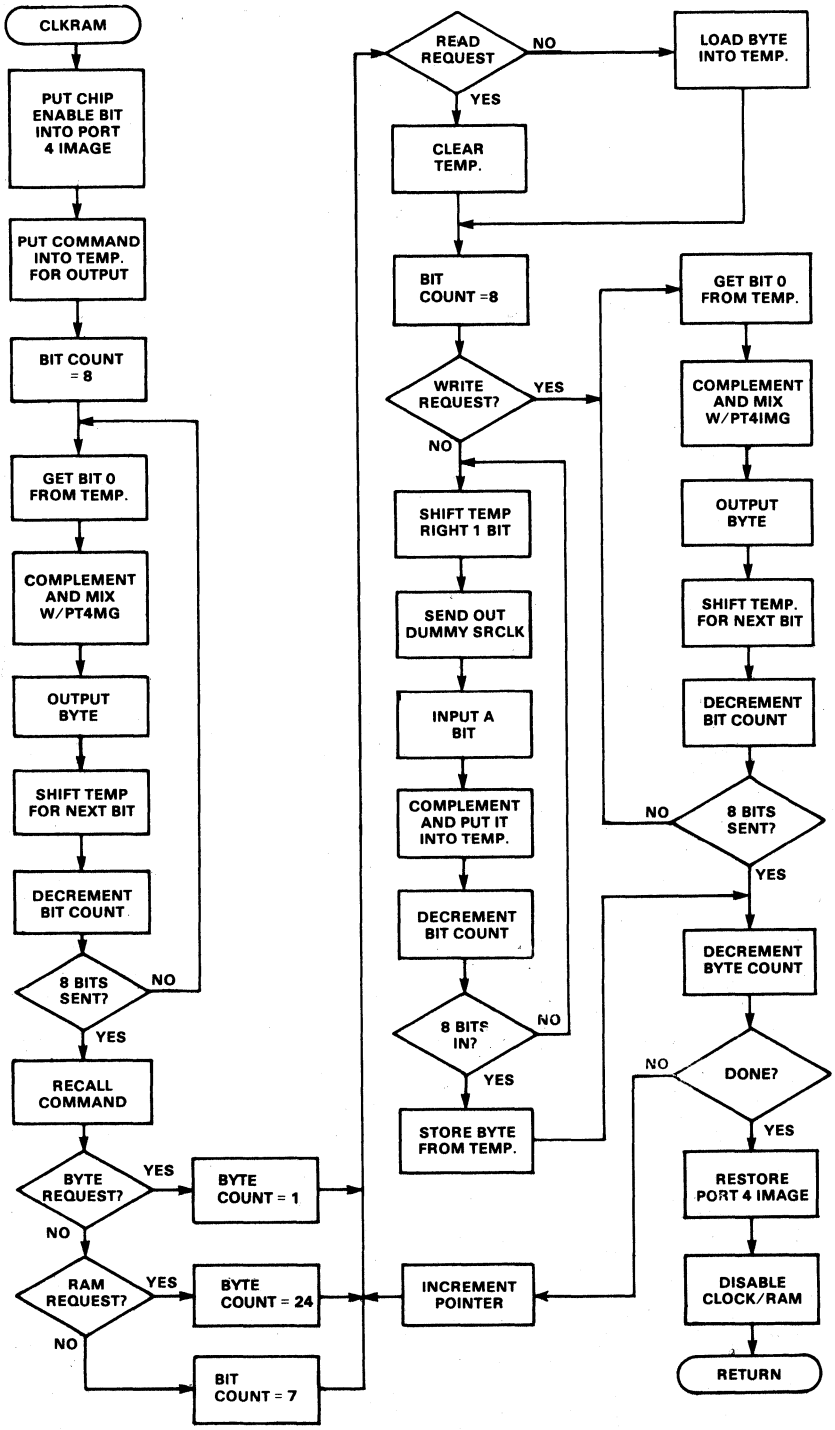
The CLOCK/RAM subroutine (listing 2) was designed to send the command to the CLOCK/RAM chip and then to transfer the number of data bytes specified by the command.

As seen in the flowchart (Figure 7), either 1, 7, or 24 bytes of data may be transferred between the microcomputer and the CLOCK/RAM. The command sent to the subroutine is exactly the command sent to the CLOCK/RAM, so there is no confusion as to the format of the command byte. When this routine is called, the ISAR must be pointing to the scratchpad RAM area where the data transferred is to be read from or written to. Note that only 7 bytes are transferred in a clock burst in order to eliminate reading and writing the control register every time.

MAIN ROUTINE FLOWCHART
Figure 6



CLKRAM SUBROUTINE FLOWCHART
Figure 7



V

LISTING 1 - DEMO PROGRAM

```
CLOCK/RAM DEMONSTRATION MODULE     F8/3870 MACRO CROSS ASSM.  V2.2
LOC  OBJ.CODE      STMT-NR-SOURCE-STMT PASS2 DEMO    DEMO    DEMO    ABS

                        1          TITLE CLOCK/RAM DEMONSTRATION MODULE
                        2          NAME DEMO
                        3          PSECT ABS
                        4          GLOBAL CLKRAM
                          *
                          * THIS MODULE MUST BE LINKED WITH THE CLOCK/RAM MODULE
                          * TO CREATE A WORKING PROGRAM.
                          *
                          *
                          ********************************
                          *                              *
                          * DEMO FOR MK3805 CLOCK/RAM CHIP *
                          *                              *
                          ********************************
```

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ************************************
                          *                                  *
                          * SCRATCH PAD REGISTER DEFINITIONS *
                          *                                  *
                          ************************************
                          *
                          * GLOBAL REGISTERS. THESE REGISTERS MUST BE THE SAME
                          * AS IN THE CLOCK/RAM MODULE.
                          *
=0000                  25 PT4IMG  EQU  00H          ;PORT 4 IMAGE STORAGE
=0001                  26 CHIPEN  EQU  01H          ;CHIP ENABLE STORAGE
=0002                  27 CMD     EQU  02H          ;COMMAND STORAGE
                          *
                          * LOCAL REGISTERS. THESE REGISTERS DO NOT NEED TO BE
                          * MADE KNOWN TO THE CLOCK/RAM MODULE.
                          *
=0003                  32 TEMP    EQU  03H          ;TEMPERARY STORAGE
=0004                  33 CNTSAV  EQU  04H          ;DIGIT COUNT SAVE
=0005                  34 DCOUNT  EQU  05H          ;DIGIT COUNTER
=0006                  35 TIMCNT  EQU  06H          ;TIMER COUNTER
=0007                  36 CTRL    EQU  07H          ;CLOCK/RAM CONTROL STORAGE
=0010                  37 SECOND  EQU  10H          ;SECOND BUFFER
=0011                  38 MINUTE  EQU  11H          ;MINUTE BUFFER
=0012                  39 HOUR    EQU  12H          ;HOUR BUFFER
=0013                  40 DAY     EQU  13H          ;DAY BUFFER
=0014                  41 DATE    EQU  14H          ;DATE BUFFER
=0015                  42 MONTH   EQU  15H          ;MONTH BUFFER
=0016                  43 YEAR    EQU  16H          ;YEAR BUFFER
                          *
                          ********************
                          *                  *
                          * PORT DEFINITIONS *
                          *                  *
                          ********************
                          *
=0004                  51 CRDATA  EQU  04H          ;CLOCK/RAM DATA PORT
=0006                  52 TICTRL  EQU  06H          ;TIMER, INTERUPT CTRL PORT
=0007                  53 TIMER   EQU  07H          ;TIMER PORT
=000C                  54 RXCTRL  EQU  0CH          ;SERIAL CONTROL PORT
=000D                  55 RXSTAT  EQU  0DH          ;SERIAL STATUS PORT
=000E                  56 MSBYTE  EQU  0EH          ;SERIAL MSB PORT
=000F                  57 LSBYTE  EQU  0FH          ;SERIAL LSB PORT
                          *
                          *********************
                          *                   *
                          * ASCII DEFINITIONS *
                          *                   *
                          *********************
                          *
=0004                  65 EOT     EQU  04H          ;END OF TEXT
=000A                  66 LF      EQU  0AH          ;LINE FEED
=000C                  67 FF      EQU  0CH          ;FORM FEED
=000D                  68 CR      EQU  0DH          ;CARIAGE RETURN
=0013                  69 DC3     EQU  13H          ;DEVICE CONTROL 3 (^S)
=001B                  70 ESC     EQU  1BH          ;ESCAPE
```

V

```
                        **************
                        *            *
                        * CONSTANTS  *
                        *            *
                        **************
                        *
                        * DAYS OF THE WEEK
                        *
=0001            80 SUN      EQU  1          ;SUNDAY IS DAY 1
=0002            81 MON      EQU  2          ;MONDAY IS DAY 2
=0003            82 TUES     EQU  3          ;TUESDAY IS DAY 3
=0004            83 WED      EQU  4          ;WEDNESDAY IS DAY 4
=0005            84 THURS    EQU  5          ;THURSDAY IS DAY 5
=0006            85 FRI      EQU  6          ;FRIDAY IS DAY 6
=0007            86 SAT      EQU  7          ;SATURDAY IS DAY 7
                        *
                        * MONTHS OF THE YEAR
                        *
=0001            90 JAN      EQU  1          ;JANUARY IS MONTH 1
=0002            91 FEB      EQU  2          ;FEBRUARY IS MONTH 2
=0003            92 MARCH    EQU  3          ;MARCH IS MONTH 3
=0004            93 APRIL    EQU  4          ;APRIL IS MONTH 4
=0005            94 MAY      EQU  5          ;MAY IS MONTH 5
=0006            95 JUNE     EQU  6          ;JUNE IS MONTH 6
=0007            96 JULY     EQU  7          ;JULY IS MONTH 7
=0008            97 AUG      EQU  8          ;AUGUST IS MONTH 8
=0009            98 SEPT     EQU  9          ;SEPTEMBER IS MONTH 9
=000A            99 OCT      EQU  10         ;OCTOBER IS MONTH 10
=000B           100 NOV      EQU  11         ;NOVEMBER IS MONTH 11
=000C           101 DEC      EQU  12         ;DECEMBER IS MONTH 12
                        *
                        * COUNTER VALUES
                        *
=0000           105 ZERO     EQU  0          ;COUNT IS 0
=0001           106 ONE      EQU  1          ;COUNT IS 1
=0002           107 TWO      EQU  2          ;COUNT IS 2
=0003           108 THREE    EQU  3          ;COUNT IS 3
=0004           109 FOUR     EQU  4          ;COUNT IS 4
=0005           110 FIVE     EQU  5          ;COUNT IS 5
=0006           111 SIX      EQU  6          ;COUNT IS 6
=0007           112 SEVEN    EQU  7          ;COUNT IS 7
=0008           113 EIGHT    EQU  8          ;COUNT IS 8
=0009           114 NINE     EQU  9          ;COUNT IS 9
=000A           115 TEN      EQU  10         ;COUNT IS 10
=0010           116 TENBCD   EQU  10H        ;BCD VALUE OF 10
                        *
                        * BCD MASKS
                        *
=000F           120 LSD      EQU  0FH        ;MASK FOR ONE'S DIGIT
=00F0           121 MSD      EQU  0F0H       ;MASK FOR TEN'S DIGIT
                        *
                        * LEAP YEAR MASKS
                        *
=0013           125 LEAP1    EQU  13H        ;MASK TO CHECK FOR ?
=0012           126 LEAP2    EQU  12H        ;MASK TO CHECK FOR ?
                        *
                        * ISAR MASK
```

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                           *
=003F              130 ISMASK  EQU  3FH          ;MASK TO 6 BITS
                           *
                           * CLOCK/CALENDAR MASKS
                           *
=0080              134 HALT    EQU  80H          ;HALT FLAG IS BIT 7 OF SECOND
                                                 S
=0070              135 SECMSD  EQU  70H          ;SECONDS TEN'S DIGIT
=000F              136 SECLSD  EQU  0FH          ;SECONDS ONE'S DIGIT
=0070              137 MINMSD  EQU  70H          ;MINUTES TEN'S DIGIT
=000F              138 MINLSD  EQU  0FH          ;MINUTES ONE'S DIGIT
=0080              139 MODE    EQU  80H          ;12/24 HOUR MODE IS BIT 7 OF
                                                 HOURS
=0020              140 AMPM    EQU  20H          ;AM/PM FLAG IS BIT 5 OF HOURS
=0030              141 HR2MSD  EQU  30H          ;24 HOUR MODE TEN'S DIGIT
=0010              142 HR1MSD  EQU  10H          ;12 HOUR MODE TEN'S DIGIT
=000F              143 HRLSD   EQU  0FH          ;HOURS ONE'S DIGIT
=0007              144 DAYLSD  EQU  07H          ;DAY MASK
=0030              145 DATMSD  EQU  30H          ;DATE TEN'S DIGIT
=000F              146 DATLSD  EQU  0FH          ;DATE ONE'S DIGIT
=0010              147 MNMSD   EQU  10H          ;MONTH TEN'S DIGIT
=000F              148 MNLSD   EQU  0FH          ;MONTH ONE'S DIGIT
=00F0              149 YRMSD   EQU  0F0H         ;YEARS TEN'S DIGIT
=000F              150 YRLSD   EQU  0FH          ;YEARS ONE'S DIGIT
                           *
                           * TIMER VALUES
                           *
=0024              154 MAXCNT  EQU  36           ;TIMER MAXIMUM COUNT
=00EA              155 TMCTRL  EQU  0EAH         ;TIMER CONTROL BYTE
                           *
                           * CHIP ENABLE BITS
                           *
=0001              159 DATA    EQU  01H          ;DATA BIT IS BIT 0
=0002              160 CE1     EQU  02H          ;CHIP ENABLE BIT IS BIT 1
                           *
                           * PARITY FOR TRANSMITTER
                           *
=00FE              164 PARITY  EQU  0FEH         ;PARITY (BIT 0) IS 'SPACE'
                           *
                           * SERIAL PORT VALUES
                           *
=000B              168 BAUD    EQU  0BH          ;BAUD RATE = 9600
=00A2              169 XMIT    EQU  0A2H         ;TRANSMIT COMMAND
=00B0              170 RCV     EQU  0B0H         ;RECEIVE COMMAND
=00B1              171 RCVI    EQU  0B1H         ;RECEIVE W/INTERUPT
                           *
                           * CLOCK/RAM VALUES
                           *
=0000              175 CRCTRL  EQU  00H          ;CLK/RAM CONTROL BYTE
=0002              176 CRCHIP  EQU  02H          ;CLK/RAM CHIP ENABLE BYTE
=008F              177 RDSTAT  EQU  8FH          ;READ CLK/RAM STATUS
=008E              178 WRSTAT  EQU  8EH          ;WRITE CLK/RAM STATUS
=00BF              179 RDCLK   EQU  0BFH         ;READ CLOCK REGISTERS
=00BE              180 WRCLK   EQU  0BEH         ;WRITE CLOCK REGISTERS
```

V

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ******************
                          *                *
                          * INITIALIZATION *
                          *                *
                          ******************
                          *
                          * FUNCTION:
                          * THIS IS THE START OF THE DEMO PROGRAM. WHEN THE
                          * MICROCOMPUTER RESETS DUE TO POWER UP OR A HARDWARE
                          * (PUSH BUTTON) RESET, THIS CODE IS ENTERED. THE
                          * INITIALIZATION CONSISTS OF CLEARING ALL SCRATCH PAD
                          * REGISTERS, SETTING UP THE CHIP ENABLE PARAMETER,
                          * SETTING THE SERIAL PORT BAUD RATE AND PARITY,
                          * AND CHECKING IF THE CLOCK DATA IS VALID. IF IT IS
                          * NOT VALID, THE ROUTINE CONTINUES ON TO SET THE CLOCK.
                          * OTHERWISE, THE DATA IS ASSUMED OK.
                          *
                          * ENTRY STATUS:
                          * THE CPU HAS BEEN RESET.
                          *
                          * EXIT STATUS:
                          * IF THE CLOCK DATA IS VALID, THEN THE ROUTINE EXITS
                          * TO THE DATA OK ROUTINE. OTHERWISE, THE ROUTINE
                          * EXITS TO THE SET CLOCK ROUTINE.
                          *
                          * CLEAR SCRATCH PAD
                          *
0000                  209         ORG  0000H
0000 70               210         CLR              ;CLEAR ALL SCRATCH PAD
0001 0B               211 INIT    LR   IS,A        ;PUT POINTER INTO ISAR
0002 70               212         CLR              ;CLEAR THAT LOCATION
0003 5C               213         LR   S,A         ;
0004 0A               214         LR   A,IS        ;BUMP POINTER
0005 1F               215         INC              ;BUMP POINTER
0006 213F             216         NI   ISMASK      ;MASK TO 6 BITS
0008 94F8             217         BNZ  INIT        ;GO IF NOT DONE
                          *
                          * SET UP CLOCK/RAM SUBROUTINE PARAMETERS.
                          *
000A 2002             221         LI   CRCHIP      ;SET CLK/RAM CHIP ENABLE
000C 51               222         LR   CHIPEN,A    ;
                          *
                          * INITALIZE SERIAL PORT PARAMETERS.
                          *
000D 200B             226         LI   BAUD        ;SET SERIAL BAUD RATE
000F BC               227         OUTS RXCTRL      ;
0010 20FE             228         LI   PARITY      ;SET PARITY TO 'SPACE'
0012 BE               229         OUTS MSBYTE      ;
                          *
                          * CHECK IF CLOCK/RAM HAS JUST BEEN POWERED UP. IF SO,
                          * INITIALIZE AND SET THE CLOCK. IF NOT, THEN THE CLOCK
                          * DATA SHOULD BE VALID.
                          *
0013 2802AE           235         PI   STATRD      ;READ CLK/RAM STATUS
0016 47               236         LR   A,CTRL      ;CHECK WRITE PROTECT BIT
0017 F7               237         NS   CTRL        ;
0018 8169             238         BP   DATAOK      ;BRANCH IF DATA GOOD
```

```
CLOCK/RAM DEMONSTRATION MODULE     F8/3870 MACRO CROSS ASSM.  V2.2
LOC   OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO    DEMO    DEMO    ABS

                            *
                            * CLOCK/RAM JUST POWERED UP, SO INITALIZE IT.
                            *
001A 2802B6            242          PI    STATWR       ;WRITE CLK/RAM STATUS
001D 29006C            243          JMP   SETCLK       ;SET CLOCK
```

V

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ***********************************
                          *                                 *
                          * TIMER INTERRUPT SERVICE ROUTINE *
                          *                                 *
                          ***********************************
                          *
                          * FUNCTION:
                          * THE TIMER INTERRUPT SERVICE ROUTINE IS ENTERED EVERY
                          * TIME THE HARDWARE TIMER TIMES OUT (APPROXIMATELY
                          * EVERY 1/36 SECONDS.) THE TIMER COUNTER IS
                          * DECREMENTED TO DETERMINE IF 1 SECOND HAS PASSED
                          * SINCE THE LAST SCREEN UPDATE. IF NOT, THE ROUTINE
                          * TERMINATES. IF SO, NEW DATA IS READ FROM THE CLOCK/
                          * RAM AND THE SCREEN IS UPDATED.
                          *
                          * ENTRY STATUS:
                          * THE TIMER HAS TIMED OUT.
                          *
                          * EXIT STATUS:
                          * IF 1 SECOND HAS NOT PASSED, THEN THE COUNTER IS
                          * DECREMENTED. OTHERWISE, THE COUNTER IS RESET AND
                          * THE NEW TIME IS READ FFROM THE CLOCK/RAM AND
                          * PRINTED.
                          *
0020                  269         ORG  0020H
0020 08               270         LR   K,P          ;SAVE STACK
0021 00               271         LR   A,KU         ;
0022 06               272         LR   QU,A         ;
0023 01               273         LR   A,KL         ;
0024 07               274         LR   QL,A         ;
                          *
                          * CHECK IF 1 SECOND HAS PASSED SINCE LAST INTERRUPT.
                          *
0025 36               278         DS   TIMCNT       ;DECREMENT COUNT
0026 941C             279         BNZ  FINISH       ;BRANCH IF NOT ZERO
                          *
                          * IT HAS, SO RESET COUNTER, READ NEW CLOCK DATA AND
                          * DISPLAY IT.
                          *
0028 2024             284         LI   MAXCNT       ;RESET COUNT
002A 56               285         LR   TIMCNT,A
002B 2802C1           286         PI   CLKRD        ;READ CLOCK REGISTERS
002E 2801A4           287         PI   AMPMOT       ;PRINT AM/PM MESSAGE
0031 2801C0           288         PI   DAYOT        ;PRINT DAY
0034 2801CC           289         PI   DATEOT       ;PRINT DATE
0037 2801F7           290         PI   TIMEOT       ;PRINT TIME
003A 2A05EB           291         DCI  HOME         ;SEND CURSOR HOME
003D 28029F           292         PI   OUTMSG
                          *
                          * PUT SERIAL PORT BACK IN RECEIVE MODE AND RETURN
                          * FROM INTERRUPT.
                          *
0040 20B1             297         LI   RCVI         ;ENABLE RCV INTERUPT
0042 BD               298         OUTS RXSTAT
0043 1B               299 FINISH  EI                ;ENABLE INTERUPTS
0044 0D               300         LR   P0,G         ;RETURN
```

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ***************************************
                          *                                     *
                          * RECEIVER INTERRUPT SERVICE ROUTINE  *
                          *                                     *
                          ***************************************
                          *
                          * FUNCTION:
                          * THE RECEIVER INTERRUPT SERVICE ROUTINE IS ENTERED
                          * EVERY TIME A CHARACTER IS RECEIVED IN THE SERIAL
                          * PORT. THE CHARACTER IS CHECKED FOR 'DC3' (CONTROL
                          * S). IF NOT A 'DC3', THEN THE ROUTINE IS TERMINATED.
                          * OTHERWISE, THE USER IS ALLOWED TO SET THE CLOCK
                          * VALUES.
                          *
                          * ENTRY STATUS:
                          * A CHARACTER HAS BEEN RECEIVED FROM THE KEYBOARD.
                          *
                          * EXIT STATUS:
                          * IF THE CHARACTER WAS NOT A 'DC3', THEN A RETURN
                          * FROM INTERRUPT IS DONE. OTHERWISE, THE ROUTINE
                          * EXITS TO THE SET CLOCK ROUTINE.
                          *
0060                  324         ORG  0060H
0060 08               325         LR   K,P          ;SAVE STACK
0061 00               326         LR   A,KU         ;
0062 06               327         LR   QU,A         ;
0063 01               328         LR   A,KL         ;
0064 07               329         LR   QL,A         ;
                          *
                          * CHECK FOR 'DC3' FROM KEYBOARD. SET THE CLOCK IF
                          * THIS KEY FOUND.
                          *
0065 280287           334         PI   INCHR2       ;GET CHARACTER
0068 2513             335         CI   DC3          ;CHECK FOR 'DC3'
006A 94D8             336         BNZ  FINISH       ;BRANCH IF NOT
                          *
                          * WAS 'DC3', SO FALL THROUGH TO SET CLOCK.
```

V

```
LOC   OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                            *****************
                            *               *
                            * SET THE CLOCK *
                            *               *
                            *****************
                            *
                            * FUNCTION:
                            * THIS ROUTINE ALLOWS THE USER TO SET THE CLOCK AND
                            * CALENDAR SETTINGS.
                            *
                            * ENTRY STATUS:
                            * EITHER THE CLOCK DATA WAS INVALID AT POWER UP OR
                            * THE USER ENTERED A 'DC3' FROM THE KEYBOARD.
                            *
                            * EXIT STATUS:
                            * ALL CLOCK/CALENDAR SETTINGS ARE SET.
                            *
006C 68                 357 SETCLK  LISL SECOND.AND.7 ;POINT TO CLOCK BUFFER
006D 62                 358         LISU SECOND.SHR.3 ;
006E 2800AB             359         PI   DAYIN        ;SET DAY OF WEEK
0071 280126             360         PI   DATEIN       ;SET DATE IN CALENDAR
0074 280096             361         PI   MODEIN       ;SET 12/24 HOUR MODE
0077 8104               362         BP   SET1         ;BRANCH IS 24 HOUR MODE
0079 2800BC             363         PI   AMPMIN       ;SET AM/PM FLAG
007C 2800D0             364 SET1    PI   TIMEIN       ;SET TIME IN CLOCK
007F 2802C9             365         PI   CLKWR        ;WRITE DATA TO CLOCK
                            *
                            * CLOCK NOW SET, SO FALL THROUGH TO START INTERRUPTS.
```

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          *************************
                          *                       *
                          * SET UP FOR INTERRUPTS *
                          *                       *
                          *************************
                          *
                          * FUNCTION:
                          * THIS ROUTINE INITIALIZES THE TIMER AND SERIAL PORT
                          * AND ENABLES INTERRUPTS.
                          *
                          * ENTRY STATUS:
                          * EITHER THE DATA WAS VALID AT POWER UP, OR THE CLOCK
                          * HAS JUST BEEN SET.
                          *
                          * EXIT STATUS:
                          * THE TIMER AND RECEIVER INTERRUPTS ARE THE ONLY EXIT.
                          *
0082 70              386 DATAOK  CLR                 ;CLEAR TIMER
0083 B7              387         OUTS TIMER          ;
0084 2024            388         LI   MAXCNT         ;SET COUNTER
0086 56              389         LR   TIMCNT,A       ;
0087 20EA            390         LI   TMCTRL         ;SET TIMER CONTROL
0089 B6              391         OUTS TICTRL         ;
008A 2A02DF          392         DCI  SIGNON         ;PRINT FEATURES
008D 28029F          393         PI   OUTMSG         ;
0090 20B1            394         LI   RCVI           ;ENABLE RCV INTERRUPT
0092 BD              395         OUTS RXSTAT         ;
0093 1B              396         EI                  ;ENABLE INTERRUPTS
0094 90FF            397 STOP    BR   STOP           ;WAIT FOR INTERRUPT
```

V

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ************************************
                          *                                  *
                          * 12/24 HOUR MODE INPUT SUBROUTINE *
                          *                                  *
                          ************************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE ASKS THE USER IF THE MODE IS TO BE
                          * 12 OR 24 HOUR FORMAT. THE ANSWER IS ACQUIRED, AND
                          * THE PROPER MODE IS SET.
                          *
                          * ENTRY STATUS:
                          * NONE.
                          *
                          * EXIT STATUS:
                          * THE MODE IS SET FOR 12 OR 24 HOUR OPERATION.
                          *
0096 08              416 MODEIN  LR   K,P          ;SAVE STACK
0097 00              417         LR   A,KU         ;
0098 06              418         LR   QU,A         ;
0099 01              419         LR   A,KL         ;
009A 07              420         LR   QL,A         ;
009B 2A0611          421         DCI  MODMSG       ;PRINT MODE MESSAGE
009E 28029F          422         PI   OUTMSG       ;
00A1 6A              423         LISL HOUR.AND.7   ;POINT TO HOURS
00A2 280234          424         PI   DIGIT2       ;GET DIGIT (0-1)
00A5 15              425         SL   4            ;PUT INTO BIT 7
00A6 13              426         SL   1            ;
00A7 13              427         SL   1            ;
00A8 13              428         SL   1            ;
00A9 5C              429         LR   S,A          ;STORE IT AT HOURS
00AA 0D              430         LR   PO,Q         ;RETURN
```

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ************************
                          *                      *
                          * DAY INPUT SUBROUTINE *
                          *                      *
                          ************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE ASKS THE USER FOR THE DAY AND
                          * INPUTS THE ANSWER.
                          *
                          * ENTRY STATUS:
                          * NONE.
                          *
                          * EXIT STATUS:
                          * THE DAY OF THE WEEK IS IN THE DAY BUFFER.
                          *
00AB 08                448 DAYIN   LR    K,P          ;SAVE STACK
00AC 00                449         LR    A,KU         ;
00AD 06                450         LR    QU,A         ;
00AE 01                451         LR    A,KL         ;
00AF 07                452         LR    QL,A         ;
00B0 2A05F0            453         DCI   DAYMSG       ;PRINT DAY MESSAGE
00B3 28029F            454         PI    OUTMSG       ;
00B6 6B                455         LISL  DAY.AND.7    ;POINT TO DAY
00B7 280222            456         PI    DIGIT7       ;GET DIGIT (1-7)
00BA 5C                457         LR    S,A          ;STORE IT AT DAY
00BB 0D                458         LR    P0,Q         ;RETURN
```

V

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                           ********************************
                           *                              *
                           * AM/PM SELECT INPUT SUBROUTINE *
                           *                              *
                           ********************************
                           *
                           * FUNCTION:
                           * THIS SUBROUTINE ASKS THE USER FOR THE AM OR PM
                           * SETTING. THE ANSWER IS ACQUIRED AND THE PROPER MODE
                           * IS SET. THIS ROUTINE IS CALLED IN THE 12 HOUR
                           * MODE ONLY.
                           *
                           * ENTRY STATUS:
                           * NONE.
                           *
                           * EXIT STATUS:
                           * THE AM/PM FLAG IS SET OR RESET IN THE HOUR BUFFER.
                           *
00BC 08                 478 AMPMIN  LR    K,P          ;SAVE STACK
00BD 00                 479         LR    A,KU         ;
00BE 06                 480         LR    QU,A         ;
00BF 01                 481         LR    A,KL         ;
00C0 07                 482         LR    QL,A         ;
00C1 2A0631             483         DCI   AMPMSG       ;PRINT AM/PM MESSAGE
00C4 28029F             484         PI    OUTMSG       ;
00C7 6A                 485         LISL  HOUR.AND.7   ;POINT TO HOURS
00C8 280234             486         PI    DIGIT2       ;GET DIGIT (0-1)
00CB 15                 487         SL    4            ;PUT INTO BIT 5
00CC 13                 488         SL    1            ;
00CD EC                 489         XS    S            ;
00CE 5C                 490         LR    S,A          ;STORE IT AT HOURS
00CF 0D                 491         LR    P0,Q         ;RETURN
```

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ************************
                          *                      *
                          * TIME INPUT SUBROUTINE *
                          *                      *
                          ************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE ASKS THE USER FOR THE TIME. IT
                          * INPUTS THE TIME AND SETS THE CLOCK UP ACCORDINGLY.
                          * THE TIME IS INPUT IN THE HR:MIN:SEC FORMAT. LEADING
                          * ZEROS MUST BE INPUT.
                          *
                          * ENTRY STATUS:
                          * NONE.
                          *
                          * EXIT STATUS:
                          * THE TIME OF DAY IS SET IN THE HOUR, MINUTE, AND
                          * SECOND BUFFER.
                          *
00D0 08              512 TIMEIN  LR   K,P          ;SAVE STACK
00D1 00              513         LR   A,KU         ;
00D2 06              514         LR   QU,A         ;
00D3 01              515         LR   A,KL         ;
00D4 07              516         LR   QL,A         ;
00D5 2A0648          517         DCI  TIMMSG       ;PRINT TIME MESSAGE
00D8 28029F          518         PI   OUTMSG       ;
                          *
                          * CHECK IF 12 OR 24 HOUR MODE.
                          *
00DB 6A              522         LISL HOUR.AND.7   ;POINT TO HOURS
00DC 4C              523         LR   A,S          ;CHECK IF 24 HOUR MODE
00DD FC              524         NS   S            ;
00DE 8115            525         BP   HOUR24       ;BRANCH IF SO
                          *
                          * 12 HOUR MODE, SO VALID HOURS ARE 01-12.
                          *
00E0 280234          529         PI   DIGIT2       GET DIGIT (0-1)
00E3 840B            530         BZ   HOUR0X       ;BRANCH IF 0 ENTERED
00E5 15              531         SL   4            ;STORE IT AT TENS
00E6 EC              532         XS   S            ;
00E7 5C              533         LR   S,A          ;
00E8 280239          534         PI   DIGIT3       ;GET DIGIT (0-2)
00EB EC              535 HOUR1   XS   S            ;STORE IT AT UNITS
00EC 5C              536         LR   S,A          ;
00ED 901B            537         BR   MIN          ;GO TO MINUTES
00EF 28022A          538 HOUR0X  PI   DIGIT9       ;GET DIGIT (1-9)
00F2 90F8            539         BR   HOUR1        ;STORE IT AND CONTINUE
                          *
                          * 24 HOUR MODE, SO VALID HOURS ARE 00-23.
                          *
00F4 280239          543 HOUR24  PI   DIGIT3       ;GET DIGIT (0-2)
00F7 15              544         SL   4            ;STORE IT AT TENS
00F8 5C              545         LR   S,A          ;
00F9 2520            546         CI   TWO.SHL.4    ;SEE IF DIGIT WAS '2'
00FB 8408            547         BZ   HOUR2X       ;BRANCH IF SO
00FD 28024D          548         PI   DIGIT0       ;GET DIGIT (0-9)
0100 EC              549 HOUR2   XS   S            ;STORE IT AT UNITS
```

V

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

0101 5C              550         LR   S,A            ;
0102 9006            551         BR   MIN            ;GO TO MINUTES
0104 28023E          552 HOUR2X  PI   DIGIT4         ;GET DIGIT (0-3)
0107 90F8            553         BR   HOUR2          ;STORE AND CONTINUE
                         *
                         * VALID MINUTES ARE 00-59.
                         *
0109 28026A          557 MIN     PI   OUTCOL         ;PRINT COLON SEPARATOR
010C 69              558         LISL MINUTE.AND.7 ;POINT TO MINUTES
010D 280243          559         PI   DIGIT6         ;GET DIGIT (0-5)
0110 15              560         SL   4              ;STORE IT AT TENS
0111 5C              561         LR   S,A            ;
0112 28024D          562         PI   DIGIT0         ;GET DIGIT (0-9)
0115 EC              563         XS   S              ;STORE IT AT UNITS
0116 5C              564         LR   S,A            ;
                         *
                         * VALID SECONDS ARE 00-59
                         *
0117 28026A          568         PI   OUTCOL         ;PRINT COLON SEPARATOR
011A 68              569         LISL SECOND.AND.7 ;POINT TO SECONDS.
011B 280243          570         PI   DIGIT6         ;GET DIGIT (0-5)
011E 15              571         SL   4              ;STORE IT AT TENS
011F 5C              572         LR   S,A            ;
0120 28024D          573         PI   DIGIT0         ;GET DIGIT (0-9)
0123 EC              574         XS   S              ;STORE IT AT UNITS
0124 5C              575         LR   S,A            ;
0125 0D              576         LR   P0,Q           ;RETURN
```

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                           *************************
                           *                       *
                           * DATE INPUT SUBROUTINE *
                           *                       *
                           *************************
                           *
                           * FUNCTION:
                           * THIS SUBROUTINE ASKS THE USER FOR THE DATE. IT
                           * INPUTS THE DATE AND SETS THE CALENDAR ACCORDINGLY.
                           * THE DATE IS INPUT IN THE YR:MNTH:DAY FORMAT.
                           * LEADING ZEROS MUST BE INPUT.
                           *
                           * ENTRY STATUS:
                           * NONE.
                           *
                           * EXIT STATUS:
                           * THE DATE IS IN THE YEAR, MONTH, AND DATE BUFFER.
                           *
0126 08                 596 DATEIN  LR   K,P          ;SAVE STACK
0127 00                 597         LR   A,KU         ;
0128 06                 598         LR   QU,A         ;
0129 01                 599         LR   A,KL         ;
012A 07                 600         LR   QL,A         ;
012B 2A05FD             601         DCI  DATMSG       ;PRINT DATE MESSAGE
012E 28029F             602         PI   OUTMSG       ;
                           *
                           * VALID YEARS ARE 00-99.
                           *
0131 6E                 606         LISL YEAR.AND.7   ;POINT TO YEAR
0132 28024D             607         PI   DIGIT0       ;GET DIGIT (0-9)
0135 15                 608         SL   4            ;STORE IT AT TENS
0136 5C                 609         LR   S,A          ;
0137 28024D             610         PI   DIGIT0       ;GET DIGIT (0-9)
013A EC                 611         XS   S            ;STORE IT AT UNITS
013B 5C                 612         LR   S,A          ;
                           *
                           * VALID MONTHS ARE 01-12.
                           *
013C 28026A             616         PI   OUTCOL       ;PRINT COLON SEPARATER
013F 6D                 617         LISL MONTH.AND.7  ;POINT TO MONTH
0140 280234             618         PI   DIGIT2       ;GET DIGIT (0-1)
0143 15                 619         SL   4            ;STORE IT AT TENS
0144 5C                 620         LR   S,A          ;
0145 8408               621         BZ   MNTHOX       ;BRANCH IF DIGIT IS '0'
0147 280239             622         PI   DIGIT3       ;GET DIGIT (0-2)
014A EC                 623 MONTH1  XS   S            ;STORE IT AT UNITS
014B 5C                 624         LR   S,A          ;
014C 9006               625         BR   DDATE        ;GO TO DATE
014E 28022A             626 MNTHOX  PI   DIGIT9       ;GET DIGIT (1-9)
0151 90F8               627         BR   MONTH1       ;STORE AND CONTINUE
                           *
                           * CHECK MONTH. IF MONTH IS FEBRUARY, ALLOW 28 OR
                           * 29 DAYS IN THE MONTH. IF MONTH IS APRIL, JUNE,
                           * SEPTEMBER OR NOVEMBER, ALLOW 30 DAYS IN THE
                           * MONTH. FOR OTHER MONTHS, ALLOW 31 DAYS.
                           *
0153 28026A             634 DDATE   PI   OUTCOL       ;PRINT COLON SEPARATOR
```

V

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

0156 4E              635          LR    A,D          ;GET MONTH, POINT DATE
0157 2502            636          CI    FEB          ;CHECK IF 'FEBRUARY'
0159 842F            637          BZ    FEBXX        ;BRANCH IF SO
                         *
                         * NOT FEBRUARY, SO ALLOW 30 OR 31 DAYS.
                         *
015B 28023E          641          PI    DIGIT4       ;GET DIGIT (0-3)
015E 15              642          SL    4            ;STORE IT AT TENS
015F 5C              643          LR    S,A          ;
0160 840B            644          BZ    DAY0X        ;BRANCH IF DIGIT WAS 0
0162 2530            645          CI    THREE.SHL.4  ;CHECK IF DIGIT WAS '3'
0164 840C            646          BZ    DAY3X        ;BRANCH IF SO
0166 28024D          647 DDATE3   PI    DIGIT0       ;GET DIGIT (0-9)
0169 EC              648 DDATE1   XS    S            ;STORE IT AT UNITS
016A 5C              649          LR    S,A          ;
016B 0D              650          LR    P0,Q         ;RETURN
016C 28022A          651 DAY0X    PI    DIGIT9       ;GET DIGIT (1-9)
016F 90F9            652          BR    DDATE1       ;STORE AND RETURN
                         *
                         * CHECK FOR APRIL, JUNE, SEPTEMBER AND NOVEMBER.
                         *
0171 6D              656 DAY3X    LISL  MONTH.AND.7  ;POINT TO MONTH
0172 2004            657          LI    4            ;LOOP COUNT = 4
0174 55              658          LR    DCOUNT,A     ;
0175 4E              659          LR    A,D          ;GET MONTH, POINT DATE
0176 2A02DB          660          DCI   TAB30        ;POINT TO 30-DAY TABLE
0179 8D              661 DLOOP    CM                 ;CHECK IF IN TABLE
017A 8409            662          BZ    DAY30        ;BRANCH IF SO
017C 35              663          DS    DCOUNT       ;DECREMENT COUNT
017D 94FB            664          BNZ   DLOOP        ;BRANCH IF NOT DONE
017F 280234          665          PI    DIGIT2       ;GET DIGIT (0-1)
                         *
                         * 31 DAY MONTH, SO ALLOW DAYS OF 01-31.
                         *
0182 90E6            669          BR    DDATE1       ;STORE AND RETURN
                         *
                         * 30 DAY MONTH, SO ALLOW DAYS OF 01-30.
                         *
0184 28022F          673 DAY30    PI    DIGIT1       ;GET DIGIT (0)
0187 90E1            674          BR    DDATE1       ;STORE AND RETURN
                         *
                         * FEBRUARY, SO ALLOW 28 OR 29 DAYS.
                         *
0189 280239          678 FEBXX    PI    DIGIT3       ;GET DIGIT (0-2)
018C 15              679          SL    4            ;STORE IT AT TENS
018D 5C              680          LR    S,A          ;
018E 84DD            681          BZ    DAY0X        ;BRANCH IF DIGIT WAS 0
0190 2520            682          CI    TWO.SHL.4    ;CHECK IF DIGIT WAS '2'
0192 94D3            683          BNZ   DDATE3       ;BRANCH IF NOT
                         *
                         * CHECK IF IT IS A LEAP YEAR.
                         *
0194 6E              687          LISL  YEAR.AND.7   ;POINT TO YEAR
0195 4C              688          LR    A,S          ;GET YEAR
0196 6C              689          LISL  DATE.AND.7   ;POINT TO DATE
0197 2113            690          NI    LEAP1        ;CHECK IF LEAP YEAR
0199 84CC            691          BZ    DDATE3       ;BRANCH IF IT IS
```

```
CLOCK/RAM DEMONSTRATION MODULE     F8/3870 MACRO CROSS ASSM.  V2.2
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO    DEMO    DEMO    ABS

019B 2312              692          XI    LEAP2       ;CHECK AGAIN
019D 84C8              693          BZ    ODATE3      ;BRANCH IF IT IS
                            *
                            * NOT A LEAP YEAR, SC ALLOW DAYS OF 01-28.
                            *
019F 280248            697          PI    DIGIT8      ;GET DIGIT (0-8)
01A2 90C6              698          BR    ODATE1      ;STORE AND RETURN
```

V

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          *************************
                          *                       *
                          * AM/PM PRINT SUBROUTINE *
                          *                       *
                          *************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE PRINTS THE MESSAGE 'GOOD MORNING'
                          * IF THE AM/PM BIT IS CLEAR, OR 'GOOD AFTERNOOD' IF
                          * THE AM/PM BIT IS SET.
                          *
                          * ENTRY STATUS:
                          * THE MODE AND AM/PM BITS MUST BE IN THE HOUR BUFFER.
                          *
                          * EXIT STATUS:
                          * IF THE MODE IS 12 HOUR, THEN THE 'GOOD MORNING' OR
                          * 'GOOD AFTERNOON' MESSAGE WAS PRINTED (DEPENDING ON
                          * THE STATUS OF THE AM/PM BIT.) OTHERWISE, THE FIRST
                          * LINE OF THE CRT WAS BLANKED.
                          *
01A4 08                720 AMPMOT  LR   K,P          ;SAVE STACK
01A5 2A0512            721         DCI  GOODPT       ;CURSOR TO LINE 1
01A8 28029F            722         PI   OUTMSG       ;
                          *
                          * CHECK IF IN 12 OR 24 HOUR MODE. SKIP THIS
                          * ROUTINE IF 24 HOUR MODE.
                          *
01AB 6A                727         LISL HOUR.AND.7   ;POINT TO HOURS
01AC 4C                728         LR   A,S          ;CHECK 12/24 HOUR BIT
01AD FC                729         NS   S            ;SET FLAGS
01AE 8110              730         BP   MLTRY1       ;BRANCH IF 24 HOUR
                          *
                          * 12 HOUR MODE, SO CHECK AM/PM FLAG. PRINT 'GOOD
                          * MORNING' IF AM, 'GOOD AFTERNOON' IF PM.
                          *
01B0 13                735         SL   1            ;CHECK AM/PM FLAG
01B1 13                736         SL   1
01B2 8106              737         BP   AMPM1        ;BRANCH IF AM
01B4 2A0527            738         DCI  GDAFTR       ;POINT TO PM MSG
01B7 9004              739         BR   AMPM2        ;CONTINUE
01B9 2A0519            740 AMPM1   DCI  GDMORN       ;POINT TO AM MSG
01BC 28029F            741 AMPM2   PI   OUTMSG       ;PRINT MESSAGE
01BF 0C                742 MLTRY1  PK                ;RETURN
```

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ************************
                          *                      *
                          * DAY PRINT SUBROUTINE *
                          *                      *
                          ************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE PRINTS THE DAY.
                          *
                          * ENTRY STATUS:
                          * THE DAY OF THE WEEK MUST BE IN THE DAY BUFFER.
                          *
                          * EXIT STATUS:
                          * THE DAY IS PRINTED ON THE CRT.
                          *
01C0 08                759 DAYOT   LR   K,P          ;SAVE STACK
01C1 2A0537            760         DCI  DAYPT        ;CURSOR TO LINE 3
01C4 28029F            761         PI   OUTMSG       ;
01C7 6B                762         LISL DAY.AND.7    ;POINT TO DAY
01C8 280295            763         PI   FNDOUT       ;PRINT DAY MESSAGE
01CB 0C                764         PK                ;RETURN
```

V

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          *************************
                          *                       *
                          * DATE PRINT SUBROUTINE *
                          *                       *
                          *************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE PRINTS THE DATE.
                          *
                          * ENTRY STATUS:
                          * THE DATE MUST BE IN THE YEAR, MONTH, AND DATE
                          * BUFFERS.
                          *
                          * EXIT STATUS:
                          * THE DATE IS PRINTED ON THE CRT.
                          *
01CC 08                782 DATEOT  LR   K,P          ;SAVE STACK
01CD 2A0576            783         DCI  DATEPT       ;CURSOR TO LINE 5
01D0 28029F            784         PI   OUTMSG       ;
01D3 6D                785         LISL MONTH.AND.7  ;POINT TO MONTH
                          *
                          * MAKE BCD MONTH BINARY.
                          *
01D4 4C                789         LR   A,S          ;GET MONTH
01D5 2110              790         NI   TENBCD       ;SEE IF MONTH > 9
01D7 4C                791         LR   A,S          ;RECALL MONTH
01D8 8405              792         BZ   DATE1        ;BRANCH IF <= 9
01DA 210F              793         NI   MNLSD        ;KEEP ONLY LSD
01DC 240A              794         AI   TEN          ;ADD 10
01DE 5C                795 DATE1   LR   S,A          ;PUT IT ALL BACK
                          *
                          * FIND MONTH IN MESSAGE AREA AND PRINT IT.
                          * THEN PRINT DATE AND YEAR.
                          *
01DF 280295            800         PI   FNDOUT       ;PRINT MONTH
01E2 6C                801         LISL DATE.AND.7   ;POINT TO DATE
01E3 280278            802         PI   OUTMSD       ;PRINT DATE
01E6 28027E            803         PI   OUTLSD       ;
01E9 2A05DF            804         DCI  SEPAR        ;PRINT SEPARATER
01EC 28029F            805         PI   OUTMSG       ;
01EF 6E                806         LISL YEAR.AND.7   ;POINT TO YEAR
01F0 280278            807         PI   OUTMSD       ;PRINT YEAR
01F3 28027E            808         PI   OUTLSD       ;
01F6 0C                809         PK                ;RETURN
```

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT·PASS2 DEMO   DEMO   DEMO   ABS

                          *************************
                          *                       *
                          * TIME PRINT SUBROUTINE *
                          *                       *
                          *************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE PRINTS THE TIME.
                          *
                          * ENTRY STATUS:
                          * THE TIME MUST BE IN THE HOUR, MINUTE, AND SECOND
                          * BUFFERS.
                          *
                          * EXIT STATUS:
                          * THE TIME IS PRINTED ON THE CRT.
                          *
01F7 08                 827 TIMEOT  LR   K,P          ;SAVE STACK
01F8 2A05E4             828         DCI  TIMEPT       ;CURSOR TO LINE 7
01FB 28029F             829         PI   OUTMSG       ;
                          *
                          * CHECK IF 12 OR 24 HOUR MODE. FOR 12 HOUR MODE,
                          * FLAGS MUST BE STRIPPED FROM HOURS BYTE.
                          *
01FE 6A                 834         LISL HOUR.AND.7   ;POINT TO HOURS
01FF 4C                 835         LR   A,S          ;CHECK 12/24 HOUR BIT
0200 FC                 836         NS   S            ;
0201 8105               837         BP   MLTRY2       ;BRANCH IF 24 HOUR
0203 4C                 838         LR   A,S          ;STRIP FLAGS FROM HOURS
0204 211F               839         NI   HR1MSD+HRLSD ;
0206 5C                 840         LR   S,A          ;
                          *
                          * PRINT HOURS, MINUTES AND SECONDS.
                          *
0207 280278             844 MLTRY2  PI   OUTMSD       ;PRINT HOURS
020A 28027E             845         PI   OUTLSD       ;
020D 28026A             846         PI   OUTCOL       ;PRINT COLON
0210 69                 847         LISL MINUTE.AND.7 ;POINT TO MINUTES
0211 280278             848         PI   OUTMSD       ;PRINT MINUTES
0214 28027E             849         PI   OUTLSD       ;
0217 28026A             850         PI   OUTCOL       ;PRINT COLON
021A 68                 851         LISL SECOND.AND.7 ;POINT TO SECONDS
021B 280278             852         PI   OUTMSD       ;PRINT SECONDS
021E 28027E             853         PI   OUTLSD       ;
0221 0C                 854         PK                ;RETURN
```

V

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ************************
                          *                      *
                          * GET DIGIT SUBROUTINE *
                          *                      *
                          ************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE, WITH ITS VARIOUS ENTRY POINTS,
                          * GETS A DIGIT FROM THE KEYBOARD AND ECHOES IT TO THE
                          * CRT.
                          *
                          * ENTRY STATUS:
                          * THE RANGE IS SPECIFIED BY A CALL TO THE APPROPRIATE
                          * ENTRY POINT.
                          *
                          * NORMAL EXIT STATUS:
                          * THE DIGIT IS ECHOED TO THE CRT, AND RETURNED
                          * IN A AS A BINARY VALUE.
                          *
                          * ERROR EXIT STATUS:
                          * THE CHARACTER IS NOT ECHOED TO THE CRT, AND THE
                          * ROUTINE LOOPS BACK FOR ANOTHER CHARACTER UNTIL
                          * A CHARACTER THAT IS IN THE RANGE IS INPUT.
                          *
                          * GET DIGIT (1-7) SUBROUTINE
                          *
0222 08              882 DIGIT7  LR   K,P          ;SAVE STACK
0223 2007            883         LI   SEVEN        ;COUNT = 7
0225 2A02D2          884 DGT     DCI  TAB19        ;POINT TO SECOND TABLE
0228 902A            885         BR   DIGITT       ;GO GET A DIGIT
                          *
                          * GET DIGIT (1-9) SUBROUTINE
                          *
022A 08              889 DIGIT9  LR   K,P          ;SAVE STACK
022B 2009            890         LI   NINE         ;COUNT = 9
022D 90F7            891         BR   DGT          ;GO GET A DIGIT
                          *
                          * GET DIGIT (0) SUBROUTINE
                          *
022F 08              895 DIGIT1  LR   K,P          ;SAVE STACK
0230 2001            896         LI   ONE          ;COUNT = 1
0232 901D            897         BR   DIGIT        ;GO GET A DIGIT
                          *
                          * GET DIGIT (0-1) SUBROUTINE
                          *
0234 08              901 DIGIT2  LR   K,P          ;SAVE STACK
0235 2002            902         LI   TWO          ;COUNT = 2
0237 9018            903         BR   DIGIT        ;GO GET A DIGIT
                          *
                          * GET DIGIT (0-2) SUBROUTINE
                          *
0239 08              907 DIGIT3  LR   K,P          ;SAVE STACK
023A 2003            908         LI   THREE        ;COUNT = 3
023C 9013            909         BR   DIGIT        ;GO GET A DIGIT
                          *
                          * GET DIGIT (0-3) SUBROUTINE
                          *
```

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

023E 08                913 DIGIT4  LR   K,P          ;SAVE STACK
023F 2004              914         LI   FOUR         ;COUNT = 4
0241 900E              915         BR   DIGIT        ;GO GET A DIGIT
                           *
                           * GET DIGIT (0-5) SUBROUTINE
                           *
0243 08                919 DIGIT6  LR   K,P          ;SAVE STACK
0244 2006              920         LI   SIX          ;COUNT = 6
0246 9009              921         BR   DIGIT        ;GO GET A DIGIT
                           *
                           * GET DIGIT (0-8) SUBROUTINE
                           *
0248 08                925 DIGIT8  LR   K,P          ;SAVE STACK
0249 2009              926         LI   NINE         ;COUNT = 9
024B 9004              927         BR   DIGIT        ;GO GET A DIGIT
                           *
                           * GET DIGIT (0-9)
                           *
024D 08                931 DIGIT0  LR   K,P          ;SAVE STACK
024E 200A              932         LI   TEN          ;COUNT = 10
0250 2A02D1            933 DIGIT   DCI  TAB09        ;POINT TO 0-9 TABLE
                           *
                           * SAVE COUNT AND POINTER IN CASE A CHARACTER IS
                           * ENTERED WHICH IS NOT WITHIN RANGE.
                           *
0253 54                938 DIGITT  LR   CNTSAV,A     ;SAVE COUNT FOR ERROR
0254 11                939         LR   H,DC         ;SAVE POINTER FOR ERROR
0255 55                940 DGTBAD  LR   DCOUNT,A     ;SAVE COUNT
0256 10                941         LR   DC,H         ;POINT TO TABLE
0257 280283            942         PI   INCHR        ;GET A CHARACTER
025A 8D                943 DGTLOP  CM                ;SEE IF IT IS IN TABLE
025B 8407              944         BZ   DGTOK        ;BRANCH IF IT IS
025D 35                945         DS   DCOUNT       ;DECREMENT COUNT
025E 94FB              946         BNZ  DGTLOP       ;BRANCH IF NOT DONE
0260 44                947         LR   A,CNTSAV     ;RESET COUNTER
0261 90F3              948         BR   DGTBAD       ;TRY AGAIN
                           *
                           * GOT A VALID CHARACTER, SO ECHO IT TO SCREEN
                           * AND MAKE IT BINARY.
                           *
0263 28026D            953 DGTOK   PI   OUTCHR       ;ECHO CHARACTER
0266 43                954         LR   A,TEMP       ;MAKE IT BCD
0267 210F              955         NI   LSD          ;
0269 0C                956         PK                ;RETURN
```

V

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                           ******************************
                           *                            *
                           * CHARACTER OUTPUT SUBROUTINE *
                           *                            *
                           ******************************
                           *
                           * FUNCTION:
                           * THIS SUBROUTINE, WITH ITS VARIOUS ENTRY POINTS,
                           * OUTPUTS THE SPECIFIED CHARACTER TO THE CRT.
                           *
                           * ENTRY STATUS:
                           * THE CHARACTER TO BE OUTPUT IS DETERMINED BY THE
                           * ENTRY POINT TO THE ROUTINE.
                           *
                           * EXIT STATUS:
                           * THE CHARACTER IS OUTPUT TO THE CRT.
                           *
                           * OUTPUT COLON SUBROUTINE
                           *
026A 203A                977 OUTCOL  LI    ':'          ;LOAD COLON INTO TEMP
026C 53                  978         LR    TEMP,A
                           *
                           * CHARACTER OUTPUT SUBROUTINE
                           *
                           * THE CHARACTER IS OUTPUT FROM REGISTER TEMP.
                           *
026D 20A2                984 OUTCHR  LI    XMIT         ;PUT INTO XMIT MODE
026F BD                  985         OUTS  RXSTAT       ;
0270 43                  986         LR    A,TEMP       ;GET CHARACTER
0271 13                  987         SL    1            ;START BIT = 0
0272 BF                  988         OUTS  LSBYTE       ;SEND IT
0273 AD                  989 LOOP1   INS   RXSTAT       ;WAIT TILL IT'S SENT
0274 13                  990         SL    1            ;
0275 81FD                991         BP    LOOP1        ;
0277 1C                  992         POP                ;RETURN
                           *
                           * OUTPUT MOST SIGNIFICANT DIGIT SUBROUTINE
                           *
                           * THE DIGIT IS OUTPUT FROM BITS 7-4 OF THE BYTE AT
                           * THE LOCATION POINTED TO BY ISAR.
                           *
0278 4C                  999 OUTMSD  LR    A,S          ;GET MSD
0279 14                 1000         SR    4            ;
027A 2230               1001 ASCII   OI    030H         ;MAKE IT ASCII
027C 90EF               1002         BR    OUTCOL+2     ;SEND IT OUT
                           *
                           * OUTPUT LEAST SIGNIFICANT DIGIT SUBROUTINE
                           *
                           * THE DIGIT IS OUTPUT FROM BITS 3-0 OF THE BYTE AT
                           * THE LOCATION POINTED TO BY ISAR.
                           *
027E 4C                 1009 OUTLSD  LR    A,S          ;GET LSD
027F 210F               1010         NI    LSD          ;
0281 90F8               1011         BR    ASCII        ;MAKE IT ASCII AND PRINT
```

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ****************************
                          *                          *
                          * CHARACTER INPUT SUBROUTINE *
                          *                          *
                          ****************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE INPUTS A CHARACTER FROM THE
                          * KEYBOARD.
                          *
                          * ENTRY STATUS:
                          * NONE.
                          *
                          * EXIT STATUS:
                          * THE CHARACTER IS RETURNED IN A IN ASCII FORMAT.
                          *
0283 20B0            1029 INCHR   LI   RCV          ;PUT INTO RCV MODE
0285 BD              1030         OUTS RXSTAT       ;
0286 AF              1031         INS  LSBYTE       ;CLEAR READY BIT
0287 AD              1032 INCHR2  INS  RXSTAT       ;WAIT TILL INPUT READY
0288 81FE            1033         BP   INCHR2       ;
028A AF              1034         INS  LSBYTE       ;GET BITS 1 AND 0
028B 14              1035         SR   4            ;
028C 12              1036         SR   1            ;
028D 12              1037         SR   1            ;
028E 53              1038         LR   TEMP,A       ;SAVE THEM
028F AE              1039         INS  MSBYTE       ;GET BITS 7 THRU 2
0290 13              1040         SL   1            ;
0291 13              1041         SL   1            ;
0292 E3              1042         XS   TEMP         ;MIX BITS INTO BYTE
0293 53              1043         LR   TEMP,A       ;SAVE INPUT
0294 1C              1044         POP               ;RETURN
```

V

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ****************************
                          *                          *
                          * PRINT MESSAGE SUBROUTINE *
                          *                          *
                          ****************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE PRINTS THE MESSAGE WHOSE NUMBER IS
                          * IN THE LOCATION AT THE POINTER.
                          *
                          * ENTRY STATUS:
                          * ISAR MUST POINT TO THE LOCATION CONTAINING THE
                          * NUMBER OF THE MESSAGE TO BE PRINTED. (TYPICALLY
                          * THE NUMBER OF THE MONTH OR DAY.) DC MUST POINT TO
                          * THE START OF THE STRING OF MESSAGES. EACH MESSAGE
                          * MUST END WITH AN 'EOT' CHARACTER.
                          *
                          * EXIT STATUS:
                          * THE APPROPRIATE MESSAGE WAS PRINTED ON THE CRT.
                          *
0295 3C              1066 FNDOUT  DS   S          ;DECREMENT MSG COUNT
0296 8408            1067         BZ   OUTMSG     ;BRANCH IF FOUND
0298 16              1068 FNDLOP  LM               ;GET CHARACTER
0299 2504            1069         CI   EOT        ;CHECK FOR END OF TEXT
029B 94FC            1070         BNZ  FNDLOP     ;BRANCH IF NOT FOUND
029D 90F7            1071         BR   FNDOUT     ;ELSE, CHECK COUNT
                          *
                          * MESSAGE LOCATED, SO FALL THROUGH TO PRINT IT.
```

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                          ****************************
                          *                          *
                          * MESSAGE OUTPUT SUBROUTINE *
                          *                          *
                          ****************************
                          *
                          * FUNCTION:
                          * THIS SUBROUTINE PRINTS THE MESSAGE STARTING AT THE
                          * POINTER.
                          *
                          * ENTRY STATUS:
                          * DC MUST POINT TO THE START OF THE MESSAGE TO BE
                          * PRINTED. IT MUST END WITH AN 'EOT' CHARACTER.
                          *
                          * EXIT STATUS:
                          * THE MESSAGE IS PRINTED ON THE CRT.
                          *
029F 20A2            1092 OUTMSG  LI   XMIT        ;PUT INTO XMIT MODE
02A1 BD              1093         OUTS RXSTAT
02A2 16              1094 LOOP3   LM               ;GET CHARACTER
02A3 2504            1095         CI   EOT         ;CHECK FOR END OF TEXT
02A5 84CD            1096         BZ   LOOP1       ;BRANCH IF END
02A7 13              1097         SL   1           ;START BIT = 0
02A8 BF              1098         OUTS LSBYTE      ;SENT CHARACTER
02A9 AD              1099 LOOP4   INS  RXSTAT      ;WAIT TILL READY FOR NEXT
02AA 81FE            1100         BP   LOOP4       ;BRANCH IF NOT READY
02AC 90F5            1101         BR   LOOP3       ;NEXT CHARACTER
```

V

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                           ********************************
                           *                              *
                           * CLOCK/RAM SUBRCUTINE PATCHES *
                           *                              *
                           ********************************
                           *
                           * FUNCTION:
                           * THESE PATCHES ARE TO SET UP THE COMMAND REGISTER
                           * FOR THE CLOCK/RAM SUBROUTINE. THE DIFFERENT ENTRY
                           * POINTS SET UP DIFFERENT COMMANDS.
                           *
                           * ENTRY STATUS:
                           * FOR WRITE COMMANDS, THE DATA MUST BE IN THE CLOCK
                           * BUFFER AREAS.
                           *
                           *
                           * EXIT STATUS:
                           * THE DATA IS TRANSFERRED BETWEEN SCRATCH PAD AND THE
                           * CLOCK/RAM.
                           * READ CLOCK/RAM STATUS SUBROUTINE
                           *
                           * READ CLOCK/RAM STATUS SUBRUTINE
                           *
02AE 60              1126 STATRD  LISU CTRL.SHR.3  ;POINT TO CTRL REG
02AF 6F              1127         LISL CTRL.AND.7  ;
02B0 208F            1128         LI   RDSTAT      ;SET UP COMMAND
02B2 52              1129         LR   CMD,A       ;
02B3 29FFFF          1130         JMP  CLKRAM      ;EXECUTE IT
                           *
                           * WRITE CLOCK/RAM STATUS SUBROUTINE
                           *
02B6 60              1134 STATWR  LISU CTRL.SHR.3  ;POINT TO CTRL REG
02B7 6F              1135         LISL CTRL.AND.7  ;
02B8 208E            1136         LI   WRSTAT      ;SET UP COMMAND
02BA 52              1137         LR   CMD,A       ;
02BB 2000            1138         LI   CRCTRL      ;SET UP CONTROL BYTE
02BD 57              1139         LR   CTRL,A      ;
02BE 2902B4          1140         JMP  CLKRAM      ;EXECUTE IT
                           *
                           * READ CLOCK SUBROUTINE
                           *
02C1 62              1144 CLKRD   LISU SECOND.SHR.3 ;POINT TO CLOCK BUFFER
02C2 68              1145         LISL SECOND.AND.7 ;
02C3 20BF            1146         LI   RDCLK       ;SET UP COMMAND
02C5 52              1147         LR   CMD,A       ;
02C6 2902BF          1148         JMP  CLKRAM      ;EXECUTE IT
                           *
                           * WRITE CLOCK SUBROUTINE
                           *
02C9 62              1152 CLKWR   LISU SECOND.SHR.3 ;POINT TO CLOCK BUFFER
02CA 68              1153         LISL SECOND.AND.7 ;
02CB 20BE            1154         LI   WRCLK       ;SET UP COMMAND
02CD 52              1155         LR   CMD,A       ;
02CE 2902C7          1156         JMP  CLKRAM      ;EXECUTE IT
```

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

                           ******************
                           *                *
                           * PROGRAM TABLES *
                           *                *
                           ******************
                           *
                           * DIGIT CHECK TABLE
                           *
02D1 30                1166 TAB09   DEFB '0'
02D2 31323334          1167 TAB19   DEFB '1','2','3','4','5','6','7','8','9'
     35363738
     39
                           *
                           * TABLE OF 30 DAY MONTHS
                           *
02DB 04060911          1171 TAB30   DEFB 4,6,9,11H
                           *
                           ********************
                           *                  *
                           * PROGRAM MESSAGES *
                           *                  *
                           ********************
                           *
                           * FEATURES MESSAGE
                           *
02DF 0C1B592A          1181 SIGNON  DEFB FF,ESC,'Y','*',' '
     20
02E4 2A2A2A2A          1182         DEFM '**********     PRESENTING MOSTEK'S '
     2A2A2A2A
     2A2A2020
     20202050
     52455345
     4E54494E
     47204D4F
     5354454B
     275320
0307 4E455720          1183         DEFM 'NEW CLOCK/RAM PERIPHERAL CHIP     '
     434C4F43
     4B2F5241
     4D205045
     52495048
     4552414C
     20434849
     50202020
     2020
0329 2A2A2A2A          1184         DEFM '**********'
     2A2A2A2A
     2A2A
0333 0D0A0A0A          1185         DEFB CR,LF,LF,LF
0337 46454154          1186         DEFM 'FEATURES:'
     55524553
     3A
0340 0D0A0A            1187         DEFB CR,LF,LF
0343 2A20434D          1188         DEFM '* CMOS DESIGN FOR EXTREMELY LOW POWER'
     4F532044
     45534947
     4E20464F
```

V

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

     52204558
     5452454D
     454C5920
     4C4F5720
     504F5745
     52
0368 20434F4E        1189          DEFM ' CONSUMPTION.'
     53554D50
     54494F4E
     2E
0375 0D0A            1190          DEFB CR,LF
0377 2A204153        1191          DEFM '* ASYNCHRONOUS SERIAL COMMUNICATION AT'
     594E4348
     524F4E4F
     55532053
     45524941
     4C20434F
     4D4D554E
     49434154
     494F4E20
     4154
039D 20564952        1192          DEFM ' VIRTUALLY ANY BAUD RATE.'
     5455414C
     4C592041
     4E592042
     41554420
     52415445
     2E
03B6 0D0A            1193          DEFB CR,LF
03B8 2A203132        1194          DEFM '* 12/24 HOUR CLOCK/CALENDAR WITH AUTO'
     2F323420
     484F5552
     20434C4F
     434B2F43
     414C454E
     44415220
     57495448
     20415554
     4F
03DD 2041444A        1195          DEFM ' ADJUST FOR SHORT MONTHS AND LEAP'
     55535420
     464F5220
     53484F52
     54204D4F
     4E544853
     20414E44
     204C4541
     50
03FE 20594541        1196          DEFM ' YEARS.'
     52532E
0405 0D0A            1197          DEFB CR,LF
0407 2A203234        1198          DEFM '* 24 BYTES OF RAM FOR POWER DOWN'
     20425954
     4553204F
     46205241
     4D20464F
     5220504F
```

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

     57455220
     444F574E
0427 2053544F        1199          DEFM ' STORAGE OF VITAL INFORMATION.'
     52414745
     204F4620
     56495441
     4C20494E
     464F524D
     4154494F
     4E2E
0445 0D0A            1200          DEFB CR,LF
0447 2A204F4E        1201          DEFM '* ON CHIP OSCILLATOR THAT PROVIDES A'
     20434849
     50204F53
     43494C4C
     41544F52
     20544841
     54205052
     4F564944
     45532041
046B 20434C4F        1202          DEFM ' CLOCK SIGNAL FOR YOUR MICROPROCESSOR.'
     434B2053
     49474E41
     4C20464F
     5220594F
     5552204D
     4943524F
     50524F43
     4553534F
     522E
0491 0D0A            1203          DEFB CR,LF
0493 2A205349        1204          DEFM '* SIMPLE INTERFACING TO ANY'
     4D504C45
     20494E54
     45524641
     43494E47
     20544F20
     414E59
04AE 204D4943        1205          DEFM ' MICROPROCESSOR.'
     524F5052
     4F434553
     534F522E
04BE 0D0A0A0A        1206          DEFB CR,LF,LF,LF
04C2 2A2A2A2A        1207          DEFM '**********     SEE YOUR MOSTEK REPRE'
     2A2A2A2A
     2A2A2020
     20202053
     45452059
     4F555220
     4D4F5354
     454B2052
     45505245
04E6 53454E54        1208          DEFM 'SENTATIVE FOR FURTHER DETAILS     '
     41544956
     4520464F
     52204655
     52544845
```

V

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

     52204445
     5441494C
     53202020
     20
0507 2A2A2A2A        1209          DEFM '**********'
     2A2A2A2A
     2A2A
0511 04              1210          DEFB EOT
                          *
                          * AM/PM MESSAGES
                          *
0512 1B592020        1214 GOODPT   DEFB ESC,'Y',' ',' ',ESC,'K',EOT
     1B4B04
0519 474F4F44        1215 GDMORN   DEFM 'GOOD MORNING!'
     204D4F52
     4E494E47
     21
0526 04              1216          DEFB EOT
0527 474F4F44        1217 GDAFTR   DEFM 'GOOD AFTERNOON!'
     20414654
     45524E4F
     4F4E21
0536 04              1218          DEFB EOT
                          *
                          * DAY MESSAGES
                          *
0537 1B592220        1222 DAYPT    DEFB ESC,'Y','"',' ',ESC,'K',EOT
     1B4B04
053E 53554E44        1223          DEFM 'SUNDAY'
     4159
0544 04              1224          DEFB EOT
0545 4D4F4E44        1225          DEFM 'MONDAY'
     4159
054B 04              1226          DEFB EOT
054C 54554553        1227          DEFM 'TUESDAY'
     444159
0553 04              1228          DEFB EOT
0554 5745444E        1229          DEFM 'WEDNESDAY'
     45534441
     59
055D 04              1230          DEFB EOT
055E 54485552        1231          DEFM 'THURSDAY'
     53444159
0566 04              1232          DEFB EOT
0567 46524944        1233          DEFM 'FRIDAY'
     4159
056D 04              1234          DEFB EOT
056E 53415455        1235          DEFM 'SATURDAY'
     52444159
                          *
                          * MONTH MESSAGES
                          *
0576 1B592420        1239 DATEPT   DEFB ESC,'Y','$',' ',ESC,'K',EOT
     1B4B04
057D 4A414E55        1240          DEFM 'JANUARY '
     41525920
0585 04              1241          DEFB EOT
```

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

0586 46454252        1242          DEFM 'FEBRUARY '
     55415259
     20
058F 04              1243          DEFB EOT
0590 4D415243        1244          DEFM 'MARCH '
     4820
0596 04              1245          DEFB EOT
0597 41505249        1246          DEFM 'APRIL '
     4C20
059D 04              1247          DEFB EOT
059E 4D415920        1248          DEFM 'MAY '
05A2 04              1249          DEFB EOT
05A3 4A554E45        1250          DEFM 'JUNE '
     20
05A8 04              1251          DEFB EOT
05A9 4A554C59        1252          DEFM 'JULY '
     20
05AE 04              1253          DEFB EOT
05AF 41554755        1254          DEFM 'AUGUST '
     535420
05B6 04              1255          DEFB EOT
05B7 53455054        1256          DEFM 'SEPTEMBER '
     454D4245
     5220
05C1 04              1257          DEFB EOT
05C2 4F43544F        1258          DEFM 'OCTOBER '
     42455220
05CA 04              1259          DEFB EOT
05CB 4E4F5645        1260          DEFM 'NOVEMBER '
     4D424552
     20
05D4 04              1261          DEFB EOT
05D5 44454345        1262          DEFM 'DECEMBER '
     4D424552
     20
05DE 04              1263          DEFB EOT
                               *
                               * YEAR SEPARATOR MESSAGE
                               *
05DF 2C203139        1267 SEPAR    DEFM ', 19'
05E3 04              1268          DEFB EOT
                               *
                               * SEND CURSOR TO TIME LINE MESSAGE
                               *
05E4 1B592620        1272 TIMEPT   DEFB ESC,'Y','&',' ',ESC,'K',EOT
     1B4B04
                               *
                               * SEND CURSOR HOME MESSAGE
                               *
05EB 1B59376F        1276 HOME     DEFB ESC,'Y','7','o',EOT
     04
                               *
                               * PROMPT MESSAGES
                               *
05F0 0C              1280 DAYMSG   DEFB FF
05F1 44415920        1281          DEFM 'DAY (1-7)? '
     28312D37
```

V

```
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 DEMO   DEMO   DEMO   ABS

     293F20
05FC 04                1282          DEFB EOT
                            *
05FD 0D0A              1284 DATMSG   DEFB CR,LF
05FF 44415445          1285          DEFM 'DATE (YR:MO:DA)? '
     20285952
     3A4D4F3A
     4441293F
     20
0610 04                1286          DEFB EOT
                            *
0611 0D0A              1288 MODMSG   DEFB CR,LF
0613 4D4F4445          1289          DEFM 'MODE (0=24 HOUR, 1=12 HOUR)? '
     2028303D
     32342048
     4F55522C
     20313D31
     3220484F
     5552293F
     20
0630 04                1290          DEFB EOT
                            *
0631 0D0A              1292 AMPMSG   DEFB CR,LF
0633 414D2F50          1293          DEFM 'AM/PM (0=AM, 1=PM)? '
     4D202830
     3D414D2C
     20313D50
     4D293F20
0647 04                1294          DEFB EOT
                            *
0648 0D0A              1296 TIMMSG   DEFB CR,LF
064A 54494D45          1297          DEFM 'TIME (HR:MN:SC)? '
     20284852
     3A4D4E3A
     5343293F
     20
065B 04                1298          DEFB EOT
                            *
065C                   1300          END
```

```
CLOCK/RAM DEMONSTRATION MODULE     F8/3870 MACRO CROSS ASSM.  V2.2
 NAME TYP VALUE DEF   REFERENCES       PASS2 DEMO   DEMO   DEMO   ABS

AMPM      0020  140
AMPM1  '  0189  740    737
AMPM2  '  01BC  741    739
AMPMIN '  00BC  478    363
AMPMOT '  01A4  720    287
AMPMSG '  0631 1292    483
APRIL     0004   93
ASCII  '  027A 1001   1011
AUG       0008   97
BAUD      000B  168    226
CE1       0002  160
CHIPEN    0001   26    222*
CLKRAM E  02CF            4 1130 1140 1148 1156
CLKRD  '  02C1 1144    286
CLKWR  '  02C9 1152    365
CMD       0002   27   1129*1137*1147*1155*
CNTSAV    0004   33    938* 947
CR        000D   68   1185 1187 1190 1193 1197 1200 1203 1206 1284 1288 1292
                      1296
CRCHIP    0002  176    221
CRCTRL    0000  175   1138
CRDATA    0004   51
CTRL      0007   36    236  237 1126 1127 1134 1135 1139*
DATA      0001  159
DATAOK '  0082  386    238
DATE      0014   41    689  801
DATE1  '  01DE  795    792
DATEIN '  0126  596    360
DATEOT '  01CC  782    289
DATEPT '  0576 1239    783
DATLSD    000F  146
DATMSD    0030  145
DATMSG '  05FD 1284    601
DAY       0013   40    455  762
DAYOX  '  016C  651    644  681
DAY30  '  0184  673    662
DAY3X  '  0171  656    646
DAYIN  '  00AB  448    359
DAYLSD    0007  144
DAYMSG '  05F0 1280    453
DAYOT  '  01C0  759    288
DAYPT  '  0537 1222    760
DC3       0013   69    335
DCOUNT    0005   34    658* 663* 940* 945*
DDATE  '  0153  634    625
DDATE1 '  0169  648    652  669  674  698
DDATE3 '  0166  647    683  691  693
DEC       000C  101
DGT    '  0225  884    891
DGTBAD '  0255  940    948
DGTLOP '  025A  943    946
DGTOK  '  0263  953    944
DIGIT  '  0250  933    897  903  909  915  921  927
DIGIT0 '  024D  931    548  562  573  607  610  647
DIGIT1 '  022F  895    673
DIGIT2 '  0234  901    424  486  529  618  665
DIGIT3 '  0239  907    534  543  622  678
```

V

```
 NAME TYP VALUE DEF   REFERENCES      PASS2 DEMO   DEMO   DEMO   ABS

DIGIT4 '  023E  913    552  641
DIGIT6 '  0243  919    559  570
DIGIT7 '  0222  882    456
DIGIT8 '  0248  925    697
DIGIT9 '  022A  889    538  626  651
DIGITT '  0253  938    885
DLOOP  '  0179  561    664
EIGHT     0008  113
EOT       0004   65   1069 1095 1210 1214 1216 1218 1222 1224 1226 1228 1230
                       1232 1234 1239 1241 1243 1245 1247 1249 1251 1253 1255
                       1257 1259 1261 1263 1268 1272 1276 1282 1286 1290 1294
                       1298
ESC       001B   70   1181 1214 1214 1222 1222 1239 1239 1272 1272 1276
FEB       0002   91    636
FEBXX  '  0189  678    637
FF        000C   67   1181 1280
FINISH '  0043  299    279  336
FIVE      0005  110
FNDLOP '  0298 1068   1070
FNDOUT '  0295 1066    763  800 1071
FOUR      0004  109    914
FRI       0006   85
GDAFTR '  0527 1217    738
GDMORN '  0519 1215    740
GOODPT '  0512 1214    721
HALT      0080  134
HOME   '  05EB 1276    291
HOUR      0012   39    423  485  522  727  834
HOUR0X '  00EF  538    530
HOUR1  '  00EB  535    539
HOUR2  '  0100  549    553
HOUR24 '  00F4  543    525
HOUR2X '  0104  552    547
HR1MSD    0010  142    839
HR2MSD    0030  141
HRLSD     000F  143    839
INCHR  '  0283 1029    942
INCHR2 '  0287 1032    334 1033
INIT   '  0001  211    217
ISMASK    003F  130    216
JAN       0001   90
JULY      0007   96
JUNE      0006   95
LEAP1     0013  125    690
LEAP2     0012  126    692
LF        000A   66   1185 1185 1185 1187 1187 1190 1193 1197 1200 1203 1206
                       1206 1206 1284 1288 1292 1296
LOOP1  '  0273  989    991 1096
LOOP3  '  02A2 1094   1101
LOOP4  '  02A9 1099   1100
LSBYTE    000F   57    988 1031 1034 1098
LSD       000F  120    955 1010
MARCH     0003   92
MAXCNT    0024  154    284  388
MAY       0005   94
MIN    '  0109  557    537  551
MINLSD    000F  138
```

```
 NAME TYP VALUE DEF   REFERENCES       PASS2 DEMO   DEMO   DEMO   ABS

MINMSD    0070  137
MINUTE    0011   38    558  847
MLTRY1 '  01BF  742    730
MLTRY2 '  0207  844    837
MNLSD     000F  148    793
MNMSD     0010  147
MNTHOX '  014E  626    621
MODE      0080  139
MODEIN '  0096  416    361
MODMSG '  0611 1288    421
MON       0002   81
MONTH     0015   42    617  656  785
MONTH1 '  014A  623    627
MSBYTE    000E   56    229 1039
MSD       00F0  121
NINE      0009  114    890  926
NOV       000B  100
OCT       000A   99
ONE       0001  106    896
OUTCHR '  026D  984    953
OUTCOL '  026A  977    557  568  616  634  846  850 1002
OUTLSD '  027E 1009    803  808  845  849  853
OUTMSD '  0278  999    802  807  844  848  852
OUTMSG '  029F 1092    292  393  422  454  484  518  602  722  741  761  784
                       805  829 1067
PARITY    00FE  164    228
PT4IMG    0000   25
RCV       00B0  170   1029
RCVI      00B1  171    297  394
RDCLK     00BF  179   1146
RDSTAT    008F  177   1128
RXCTRL    000C   54    227
RXSTAT    000D   55    298  395  985  989 1030 1032 1093 1099
SAT       0007   86
SECLSD    000F  136
SECMSD    0070  135
SECOND    0010   37    357  358  569  851 1144 1145 1152 1153
SEPAR  '  05DF 1267    804
SEPT      0009   98
SET1   '  007C  364    362
SETCLK '  006C  357    243
SEVEN     0007  112    883
SIGNON '  02DF 1181    392
SIX       0006  111    920
STATRD '  02AE 1126    235
STATWR '  02B6 1134    242
STOP   '  0094  397    397
SUN       0001   80
TAB09  '  02D1 1166    933
TAB19  '  02D2 1167    884
TAB30  '  02DB 1171    660
TEMP      0003   32    954  978* 986 1038*1042 1043*
TEN       000A  115    794  932
TENBCD    0010  116    790
THREE     0003  108    645  908
THURS     0005   84
TICTRL    0006   52    391
```

V

```
 NAME TYP VALUE DEF   REFERENCES      PASS2 DEMO   DEMO   DEMO   ABS

TIMCNT     0006   35    278* 285* 389*
TIMEIN '   00D0  512    364
TIMEOT '   01F7  827    290
TIMEPT '   05E4 1272    828
TIMER      0007   53    387
TIMMSG '   0648 1296    517
TMCTRL     00EA  155    390
TUES       0003   82
TWO        0002  107    546  682  902
WED        0004   83
WRCLK      00BE  180   1154
WRSTAT     008E  178   1136
XMIT       00A2  169    984 1092
YEAR       0016   43    606  687  806
YRLSD      000F  150
YRMSD      00F0  149
ZERO       0000  105
```

LISTING 2 - CLOCK/RAM COMMUNICATIONS SUBROUTINE

```
CLOCK/RAM COMMUNICATION MODULE     F8/3870 MACRO CROSS ASSM.  V2.2
LOC  OBJ.CODE      STMT-NR SOURCE-STMT PASS2 CLKRAM CLKRAM CLKRAM REL

                    1          TITLE CLOCK/RAM COMMUNICATION MODULE
                    2          NAME CLKRAM
                    3          PSECT REL
                    4          GLOBAL CLKRAM
                      *
                      * THIS MODULE MUST BE LINKED WITH OTHER MODULES
                      * IN ORDER TO CREATE A WORKING PROGRAM.
                      *
                      ************************************
                      *                                  *
                      * CLOCK/RAM COMMUNICATION SUBROUTINE *
                      *                                  *
                      ************************************
                      *
                      * THIS SUBROUTINE IS CALLED BY THE APPLICATION
                      * PROGRAM TO SEND AND RECEIVE DATA TO AND FROM THE
                      * CLOCK/RAM CHIP. WHEN CALLED, THE COMMAND TO BE
                      * EXECUTED MUST BE IN THE SCRATCH-PAD REGISTER
                      * 'CMD', THE CHIP ENABLE CODE MUST BE IN REGISTER
                      * 'CHIPEN' AND THE ISAR MUST POINT TO THE TOP OF
                      * THE DATA AREA.
                      *
                      * THIS ROUTINE ALLOWS THE PORT 4 BITS THAT ARE NOT
                      * USED FOR CHIP ENABLE LINES TO BE USED FOR OTHER
                      * PURPOSES. TO DO THIS, AN IMAGE OF WHATEVER IS
                      * WRITTEN TO THE PORT BY OTHER ROUTINES MUST BE
                      * KEPT IN REGISTER 'PT4IMG'. IN THIS WAY, THOSE
                      * PORT LINES NOT USED BY THIS ROUTINE WILL NOT BE
                      * ALTERED. HOWEVER, ANY OF THE PORT 4 LINES THAT
                      * ARE USED FOR THE CLOCK/RAM MUST ALWAYS BE LEFT
                      * AT A LOGICAL 0.
                      *
                      * COMMAND BYTE FORMAT:
                      * BIT 7 - MUST BE 1
                      * BIT 6 - SOURCE/DESTINATION (1=RAM, 0=CLOCK)
                      * BITS 5 THRU 1 - ADDRESS
                      * BIT 0 - DIRECTION (1=READ, 0=WRITE)
                      *
                      * FOR BYTE MODE, THE ADDRESS OF THE BYTE IS PUT
                      * INTO THE ADDRESS FIELD OF THE COMMAND. FOR BURST
                      * MODE, THE ADDRESS SHOULD BE 01FH. NOTE THAT A
                      * CLOCK BURST FUNCTION TRANSFERS ONLY THE 7 CLOCK
                      * BYTES. IT DOES NOT TRANSFER THE CONTROL BYTE.
                      * VALID ADDRESSES FOR THE COMMAND BYTE (FOR BYTE
                      * MODE) ARE:
                      * CLOCK - 0 THRU 07H
                      * RAM   - 0 THRU 017H
                      *
                      * CHIP ENABLE CONTROL BYTE FORMAT:
                      * BIT 7 THRU 1 - CONTROLS PORT 4 BITS 7 THRU 1
                      * BIT 0 - MUST BE 0 (USED FOR DATA I/O LINE)
                      *
                      * TO SELECT A CLOCK/RAM CHIP WITH ITS /CE PIN
                      * TIED TO A PORT 4 PIN, THE CORRESPONDING BIT
                      * POSITION SHOULD BE SET TO A 1 (ALL OTHER BITS
                      * SHOULD BE 0).
                      *
```

V

```
                    * CALLING SEQUENCE:
                    * 1) DATA SHOULD BE IN DATA AREA (WRITE ONLY)
                    * 2) LOAD ISAR TO POINT TO BOTTOM OF DATA AREA
                    * 3) CHIPEN BYTE SHOULD BE IN REGISTER 'CHIPEN'
                    * 4) COMMAND BYTE SHOULD BE IN REGISTER 'CMD'
                    * 5) PORT 4 IMAGE SHOULD BE IN REGISTER 'PT4IMG'
                    * 6) CALL CLKRAM
                    * 7) RETURN WITH DATA AREA FILLED (READ ONLY)
                    *
                    * PORT 4 IS USED FOR ALL I/O SO THAT ITS /STROBE
                    * SERVES AS THE SHIFT REGISTER CLOCK (SRCLK) TO
                    * THE CLOCK/RAM.
                    *
                    * AS PRESENTED HERE, THIS SUBROUTINE MUST NOT BE
                    * INTERRUPTED. BUT THE USER MAY EASILY MODIFY THE
                    * CODE TO SUPPORT INTERRUPTS.
```

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 CLKRAM CLKRAM CLKRAM REL

                          *************
                          *           *
                          * CONSTANTS *
                          *           *
                          *************
                          *
                          * GLOBAL REGISTERS. THESE REGISTERS MUST BE THE SAME
                          * AS IN THE APPLICATION MODULE(S).
                          *
=0000              84 PT4IMG  EQU  0          ;PORT 4 IMAGE STORAGE
=0001              85 CHIPEN  EQU  1          ;CHIP ENABLE STORAGE
=0002              86 CMD     EQU  2          ;COMMAND STORAGE
                          *
                          * LOCAL REGISTERS. THESE REGISTERS DO NOT NEED TO BE
                          * MADE KNOWN TO THE APPLICATION MODULE(S). HOWEVER,
                          * THEY ARE DESTROYED. SO THE APPLICATION MODULE(S)
                          * SHOULD NOT KEEP NEEDED INFORMATION IN THEM.
                          *
=0003              93 TEMP    EQU  3          ;TEMPERARY STORAGE
=0004              94 BITCNT  EQU  4          ;BIT COUNTER
=0005              95 BYTCNT  EQU  5          ;BYTE COUNTER
                          *
                          * PORT DEFINITIONS
                          *
=0004              99 PORT4   EQU  4          ;PORT 4
                          *
                          * BIT MASK DEFINITIONS
                          *
=0001             103 BIT0    EQU  01H        ;BIT 0 MASK
=0080             104 BIT7    EQU  80H        ;BIT 7 MASK
                          *
                          * COUNTER VALUES
                          *
=0001             108 ONE     EQU  1          ;COUNT IS 1
=0007             109 SEVEN   EQU  7          ;COUNT IS 7
=0008             110 EIGHT   EQU  8          ;COUNT IS 8
=0018             111 TWFOUR  EQU  24         ;COUNT IS 24
                          *
                          * COMMAND BIT DEFINITIONS
                          *
=0001             115 RDWR    EQU  01H        ;READ/WRITE IS BIT 0
=003E             116 ADR     EQU  3EH        ;ADDRESS IS BITS 1-5
=0040             117 CKRM    EQU  40H        ;CLOCK/RAM IS BIT 6
```

V

```
LOC  OBJ.CODE     STMT-NR SOURCE-STMT PASS2 CLKRAM CLKRAM CLKRAM REL

                          ****************************
                          *                          *
                          * START OF CLOCK/RAM DRIVER *
                          *                          *
                          ****************************
                          *
0000'41                 125 CLKRAM  LR    A,CHIPEN     ;PUT CHIP ENABLE INTO PT4IMG
0001 E0                 126         XS    PT4IMG       ;
0002 50                 127         LR    PT4IMG,A     ;
                          *
                          * SEND OUT COMMAND TO CLOCK/RAM
                          *
0003 42                 131         LR    A,CMD        ;GET COMMAND
0004 53                 132         LR    TEMP,A       ;SAVE COMMAND FOR OUTPUT
0005 2008               133         LI    EIGHT        ;BIT COUNT = 8
0007 54                 134         LR    BITCNT,A     ;
0008'43                 135 BLOOP   LR    A,TEMP       ;GET COMMAND BYTE
0009'2101               136 BLOOP1  NI    BIT0         ;MASK OFF ALL BUT BIT 0
000B 2301               137         XI    BIT0         ;COMPLEMENT BIT 0
000D E0                 138         XS    PT4IMG       ;MIX IT WITH CONTROL BYTE
000E B4                 139         OUTS  PORT4        ;SEND IT OUT
000F 43                 140         LR    A,TEMP       ;SHIFT FOR NEXT BIT
0010 12                 141         SR    1            ;
0011 53                 142         LR    TEMP,A       ;
0012 34                 143         DS    BITCNT       ;DECREMENT BIT COUNT
0013 94F5               144         BNZ   BLOOP1       ;BRANCH IF NOT DONE
                          *
                          * SET BYTE COUNT TO PROPER LENGTH
                          *
0015 42                 148         LR    A,CMD        ;GET COMMAND
0016 213E               149         NI    ADR          ;MASK OFF ALL BUT ADDRESS
0018 253E               150         CI    ADR          ;CHECK IF BYTE OR BURST
001A 940D               151         BNZ   BYTE         ;BRANCH IF BYTE
001C 42                 152         LR    A,CMD        ;GET COMMAND BACK
001D 13                 153         SL    1            ;CHECK RAM/CLOCK BIT
001E 9105               154         BM    RAM          ;BRANCH IF RAM
0020'2007               155 CLOCK   LI    SEVEN        ;CLOCK, SO BYTE COUNT = 7
0022 9007               156         BR    CONT         ;CONTINUE
0024'2018               157 RAM     LI    TWFOUR       ;RAM, SO BYTE COUNT = 24
0026 9003               158         BR    CONT         ;CONTINUE
0028'2001               159 BYTE    LI    ONE          ;BYTE, SO BYTE COUNT = 1
002A'55                 160 CONT    LR    BYTCNT,A     ;
                          *
                          * MAIN BYTE TRANSFER LOOP
                          *
002B'42                 164 MLOOP   LR    A,CMD        ;CHECK READ/WRITE BIT
002C 2101               165         NI    RDWR         ;
002E 70                 166         CLR                ;
002F 9402               167         BNZ   XFER         ;BRANCH IF READ DIRECTION
0031 4C                 168         LR    A,S          ;WRITE, SO LOAD BYTE
0032'53                 169 XFER    LR    TEMP,A       ;
0033 2008               170         LI    EIGHT        ;BIT COUNT = 8
0035 54                 171         LR    BITCNT,A     ;
0036 42                 172         LR    A,CMD        ;CHECK READ/WRITE BIT
0037 2101               173         NI    RDWR         ;
0039 841B               174         BZ    WRITE        ;BRANCH IF WRITE DIRECTION
```

```
LOC   OBJ.CODE      STMT-NR SOURCE-STMT PASS2 CLKRAM CLKRAM CLKRAM REL

                            * READ A BYTE
                            *
003B'43                 178 READ     LR    A,TEMP       ;SHIFT FOR NEXT BIT
003C'12                 179 READ1    SR    1            ;
003D 53                 180          LR    TEMP,A       ;
003E 40                 181          LR    A,PT4IMG     ;SEND OUT DUMMY CLOCK
003F B4                 182          OUTS  PORT4        ;
0040 A4                 183          INS   PORT4        ;INPUT DATA BIT
0041 2101               184          NI    BIT0         ;MASK ALL EXCEPT DATA BIT
0043 70                 185          CLR                ;IF DATA=1, FORCE BIT-7=0
0044 9403               186          BNZ   READ2        ;BRANCH IF DATA = 1
0046 2080               187          LI    BIT7         ;DATA=0, FORCE BIT-7=1
0048'E3                 188 READ2    XS    TEMP         ;MIX WITH PREVIOUS BITS
0049 34                 189          DS    BITCNT       ;DECREMENT BIT COUNT
004A 94F1               190          BNZ   READ1        ;BRANCH IF NOT 8 BITS
004C 5C                 191          LR    S,A          ;STORE BYTE
                            *
                            * CHECK IF ALL BYTES WERE TRANSFERRED
                            *
004D'35                 195 ENDCK    DS    BYTCNT       ;DECREMENT BYTE COUNT
004E 8415               196          BZ    EXIT         ;BRANCH IF DONE
0050 0A                 197          LR    A,IS         ;INCREMENT POINTER
0051 1F                 198          INC                ;
0052 0B                 199          LR    IS,A         ;
0053 90D7               200          BR    MLOOP        ;LOOP BACK FOR NEXT BYTE
                            *
                            * WRITE A BYTE
                            *
0055'43                 204 WRITE    LR    A,TEMP       ;GET DATA BYTE
0056'2101               205 WRITE1   NI    BIT0         ;MASK OFF ALL BUT BIT 0
0058 2301               206          XI    BIT0         ;COMPLEMENT BIT 0
005A E0                 207          XS    PT4IMG       ;MIX IT WITH CONTROL BYTE
005B B4                 208          OUTS  PORT4        ;SEND IT OUT
005C 43                 209          LR    A,TEMP       ;SHIFT FOR NEXT BIT
005D 12                 210          SR    1            ;
005E 53                 211          LR    TEMP,A       ;
005F 34                 212          DS    BITCNT       ;DECREMENT BIT COUNT
0060 94F5               213          BNZ   WRITE1       ;BRANCH IF NOT 8 BITS
0062 90EA               214          BR    ENDCK        ;CONTINUE
                            *
                            * EXIT FROM SUBRCUTINE
                            *
0064'41                 218 EXIT     LR    A,CHIPEN     ;RESTORE PORT 4 IMAGE
0065 E0                 219          XS    PT4IMG       ;
0066 50                 220          LR    PT4IMG,A     ;
0067 B4                 221          OUTS  PORT4        ;DISABLE CHIP
0068 1C                 222          POP                ;FINISHED
```

V

```
 NAME TYP VALUE DEF   REFERENCES      PASS2 CLKRAM CLKRAM CLKRAM REL

ADR        003E  116    149  150
BIT0       0001  103    136  137  184  205  206
BIT7       0080  104    187
BITCNT     0004   94    134* 143* 171* 189* 212*
BLOOP   '  0008  135
BLOOP1  '  0009  136    144
BYTCNT     0005   95    160* 195*
BYTE    '  0028  159    151
CHIPEN     0001   85    125  218
CKRM       0040  117
CLKRAM  I  0000  125      4
CLOCK   '  0020  155
CMD        0002   86    131  148  152  164  172
CONT    '  002A  160    156  158
EIGHT      0008  110    133  170
ENDCK   '  004D  195    214
EXIT    '  0064  218    196
MLOOP   '  002B  164    200
ONE        0001  108    159
PORT4      0004   99    139  182  183  208  221
PT4IMG     0000   84    126  127* 138  181  207  219  220*
RAM     '  0024  157    154
RDWR       0001  115    165  173
READ    '  003B  178
READ1   '  003C  179    190
READ2   '  0048  188    186
SEVEN      0007  109    155
TEMP       0003   93    132* 135  140  142* 169* 178  180* 188  204  209  211*
TWFOUR     0018  111    157
WRITE   '  0055  204    174
WRITE1  '  0056  205    213
XFER    '  0032  169    167
```

LISTING 3 - LOAD MAP AND GLOBAL CROSS REFERENCF

```
LOAD MAP

DK1:DEMO   .OBJ[1]     ABS     BEG ADDR 0000     END ADDR 065B
DK1:CLKRAM.OBJ[1]      REL     BEG ADDR 065C     END ADDR 06C4

 GLOBAL CROSS REFERENCE TABLE

 SYMBOL ADDR  REFERENCES
 CLKRAM 065C  02CF 02C7 02BF 02B4
```

V

MOSTEK®

CMOS MK3805 PROVIDES REAL TIME CLOCK/CALENDAR TO A Z80 BUS

Application Note

INTRODUCTION

Mostek's new MK3805 CLOCK/RAM chip provides the capability of a real time clock/calendar and/or memory that can be battery powered with very low power drain. While the MK3805 was designed to be used with the serial port on Mostek's MK3873 single chip microcomputer, it can be easily interfaced to a microprocessor with a parallel bus. The simplicity of interfacing allows easy addition of a real time clock to existing systems. This application note describes several methods of interfacing the MK3805 to a Z80 bus or any parallel microprocessor bus.

TECHNICAL DESCRIPTION

Figure 2 is a block diagram of the CLOCK/RAM chip. The main components are the oscillator and divider, the real time clock/calendar, the static RAM, the command register and logic, the control register and logic, and the serial shift register.

The shift register is the MK3805's communication with the outside world. Data on the I/O line is either input or output on each shift register clock pulse when the chip is enabled. If the chip is in the input mode, the data on the I/O line is input to the shift register on the rising edge of SCLK. If in the output mode, data is shifted out onto the I/O line on the falling edge of SCLK.

The command register receives the first byte input by the shift register after $\overline{CE}$ goes true (low). This byte must be the command byte and will direct further operations within the CLOCK/RAM. The command specifies whether subsequent transfers will be read or written, and what register or RAM location will be involved.

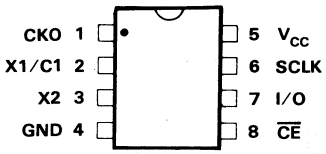
The control register has bits defined which control the divider for the internal real-time clock and the external system clock. One bit serves as the write protect control flag, preventing accidental write operations during power-up or power-down situations.

The real-time clock/calendar is accessed via seven registers. These registers contain seconds, minutes, hours, day, date, month, and year information. Certain bits within these registers also control a run/stop function, 12/24 hour clock mode, and indicate AM or PM (12 hour mode only). These registers can be accessed either randomly in byte mode, or sequentially in Burst Mode.

The static RAM is organized as 24 bytes of 8-bits each. They can be accessed either randomly in byte mode, or sequentially in Burst Mode.

PIN OUT

Figure 1



PIN DESCRIPTION

Table 1

PIN	NAME	DESCRIPTION
1	CKO	Buffered System Clock Output
2	X1/C1	Crystal or External Clock Input
3	X2	Crystal Input
4	GND	Power Supply Pin
5	$\overline{CE}$	Chip Enable for Serial I/O Transfer
6	I/O	Data Input/Output Pin
7	SCLK	Shift Clock for Serial I/O Transfer
8	V_{CC}	Power Supply Pin

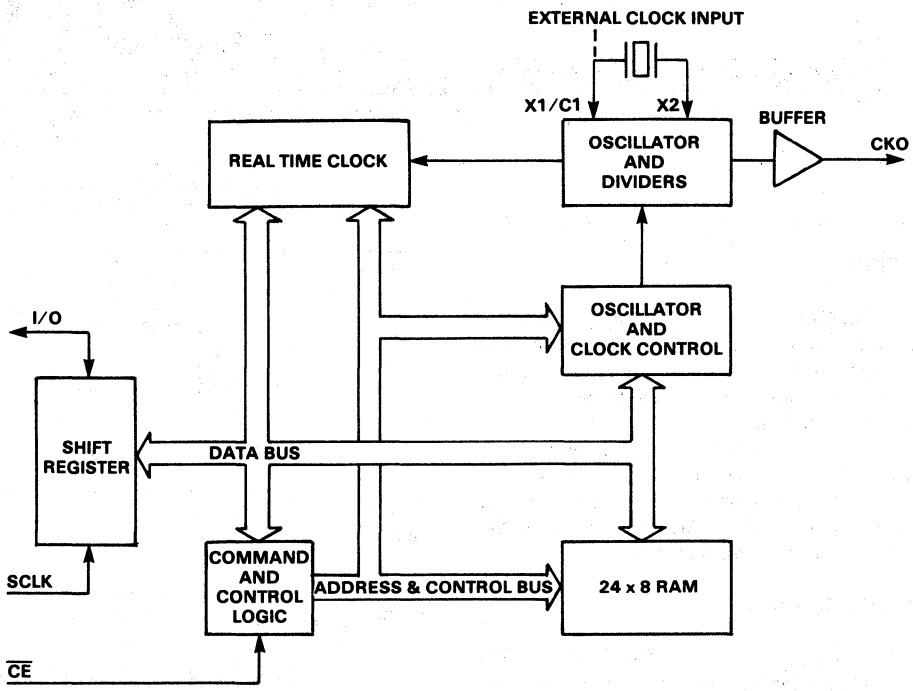
DATA TRANSFER

Data Transfer is accomplished under control of the $\overline{CE}$ and SCLK inputs by an external microcomputer. Each transfer consists of a single byte ADDRESS/COMMAND input followed by a single byte or multiple byte (if Burst Mode is specified) data input or output, as specified by the ADDRESS/COMMAND byte. The serial data transfer occurs with LSB first, MSB last format.

V

BLOCK DIAGRAM

Figure 2



ADDRESS/COMMAND BYTE

The ADDRESS/COMMAND Byte is shown below:

7	6	5	4	3	2	1	0
1	RAM / $\overline{CK}$	A4	A3	A2	A1	A0	Rd / $\overline{W}$

As defined, the MSB (bit 7) must be a logical 1; bit 6 specifies a Clock/Calendar/Control register if logical 0 or a RAM register if logical 1; bits 1-5 specify the designated register(s) to be input or output; and the LSB (bit 0) specifies a WRITE operation (input) if logical 0 or READ operation (output) if logical 1.

BURST MODE

Burst Mode may be specified for either the Clock/Calendar/Control registers or for the RAM registers by addressing location 31 Decimal (ADDRESS/COMMAND bits 1-5 = logical 1). As before, bit 6 specifies Clock or RAM, and bit 0 specifies READ or WRITE.

There is no data storage capability at location 31 in either the Clock/Calendar/Control registers or the RAM registers.

SCLK AND $\overline{CE}$ CONTROL

All data transfers are initiated by $\overline{CE}$ going low. After $\overline{CE}$ goes low, the next 8 SCLK cycles input an ADDRESS/COMMAND byte of the proper format. An SCLK cycle is the sequence of a positive edge followed by a negative edge. For data inputs, the data must be valid during the SCLK cycle. If bit 7 is not a logical 1, indicating a valid CLOCK/RAM ADDRESS/COMMAND, the ADDRESS/COMMAND byte is ignored as are all SCLK cycles until $\overline{CE}$ goes high and returns low to initiate a new ADDRESS/COMMAND TRANSFER. See Figure 3.

ADDRESS/COMMAND bits and DATA bits are input on the rising edge of SCLK, and DATA bits are output on the falling edge of SCLK.

A data transfer terminates if $\overline{CE}$ goes high, and the transfer must be reinitiated by the proper ADDRESS/COMMAND when $\overline{CE}$ again goes low. The data I/O pin is high impedance when $\overline{CE}$ is high.

DATA INPUT

Following the 8 SCLK cycles that input the WRITE Mode ADDRESS/COMMAND byte (bit 0 = logical 0), a DATA byte is input on the rising edge of the next 8 SCLK cycles (per byte, if Burst Mode is specified). Additional SCLK cycles are ignored should they inadvertently occur.

DATA OUTPUT

Following the 8 SCLK cycles that input the READ Mode ADDRESS/COMMAND byte (bit 0 = logical 1), a DATA byte is output on the falling edge of the next 8 SCLK cycles (per byte, if Burst Mode is specified). Additional SCLK cycles

retransmit the data byte(s) should they inadvertently occur, so long as $\overline{CE}$ remains low. This operation permits continuous Burst Read Mode of the Clock registers.

DATA TRANSFER SUMMARY

A data transfer summary is shown in Figure 3.

INTERFACING CONCEPTS

This application note will deal with two different concepts of interfacing the serial MK3805 CLOCK/RAM to a parallel bus. The first approach is to generate all of the required signals and timing by setting and resetting individual bits of an I/O Port latch. The second approach is to use a serial communication chip, such as the Mostek MK3801 Serial Timer Interrupt Controller.

I/O PORT INTERFACE

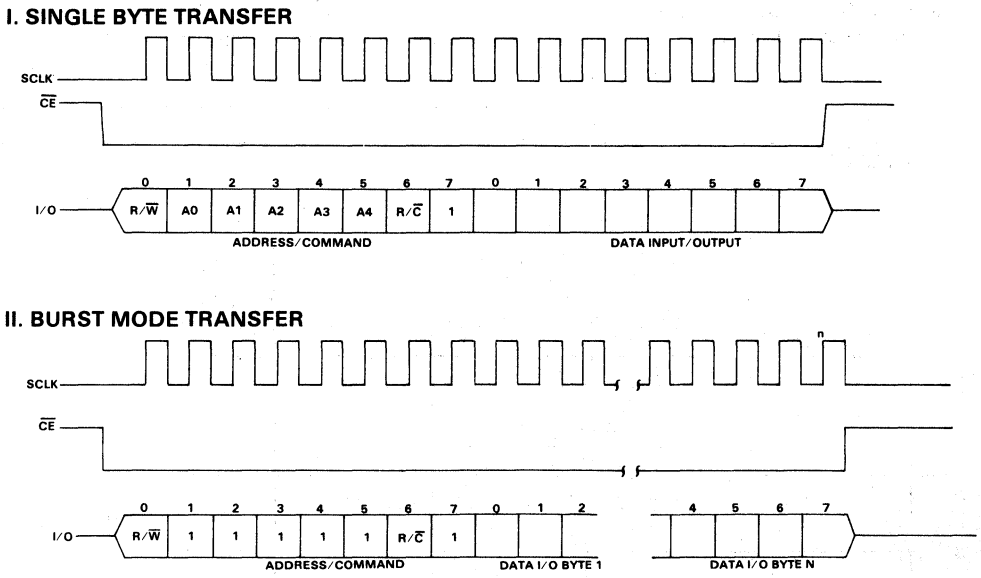
There are three input signals required to communicate with the CLOCK/RAM chip. They are:

1. SCLK — Serial Clock
2. $\overline{CE}$ - Chip Enable
3. I/O - Data Input and Output

SCLK, $\overline{CE}$, and Data Input can be generated by setting and resetting 3 bits of an I/O port latch in a specific sequence. Data out of the CLOCK/RAM can be read from the I/O pin by buffering the pin to the processor databus.

Since the I/O pin on the CLOCK/RAM chip is bi-directional, the output of the Port latch, used to send data to the CLOCK/RAM I/O pin, must be tri-stateable. The buffer

DATA TRANSFER SUMMARY
Figure 3



FUNCTION	BYTE N	SCLK n
CLOCK	8	72
RAM	24	200

NOTES

1) Data input sampled on rising edge of clock
2) Data output changes on falling edge of clock
3) Rising edge of $\overline{CE}$ terminates operation and resets address/command

used to read data into the processor must also be tri-stateable, to avoid driving the processor databus continuously.

Figure 4 is a schematic diagram of a TTL interface between a Z80 bus and the MK3805 CLOCK/RAM.

The 74LS138 (U1) is used as an I/O port decoder. An OUT instruction to port 10H results in the Y0 output pulsing low and back high while valid output data is on the Z80 data bus (See Figure 5). This latches the Z80 output data into the 74LS175 (U2) I/O Port latch.

An IN instruction from Port 10H, results in Y4 of port decoder, (U1), pulsing low (See Figure 5). This enables the 74LS125 (U3-1) tristate buffer, which allows the Z80 to read in the value of the I/O pin of the CLOCK/RAM.

The 4th bit in the port latch (U2) is used as an enable bit for the 74LS125 (U3-1) transmit buffer. The transmit buffer is disabled when data is to be read from the CLOCK/RAM.

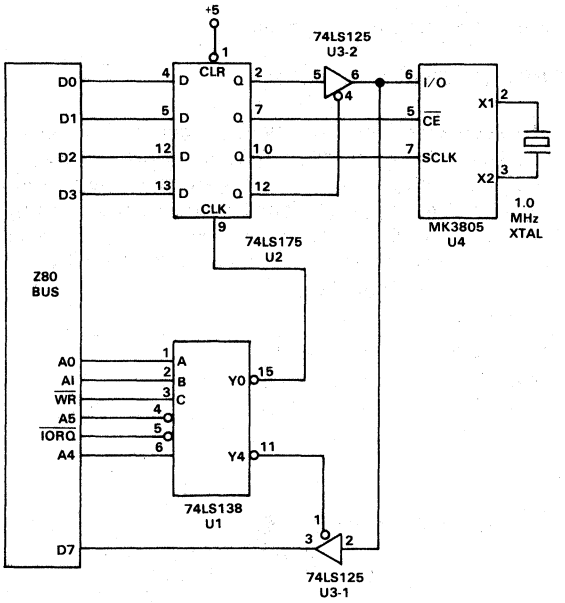
SOFTWARE — TTL INTERFACE

Communication between the Z80 and the CLOCK/RAM is accomplished by calling Z80 subroutines in the required sequence. The four subroutines needed are:

CHPENA
SNDBYT
GETBYT
CHPDIS

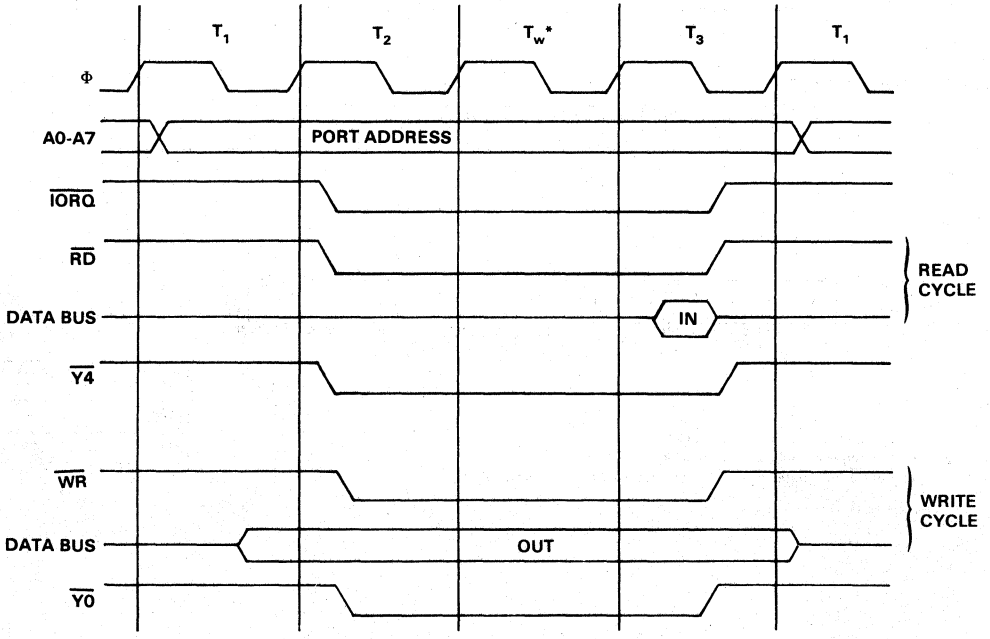
TTL INTERFACE

Figure 4



INPUT OR OUTPUT CYCLES

Figure 5



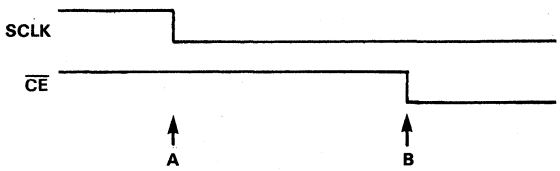
The least significant 4 bits of the byte that is output to the I/O Port latch must always be set to the correct value anytime an OUT instruction to the latch port is executed. Data bit 0 is used for data to be sent to the CLOCK/RAM. Data bit 1 is the CLOCK/RAM Chip enable ($\overline{CE}$) signal, data bit 2 is the SCLK signal, and data bit 3 is the transmit buffer disable bit. In the following software subroutines, when data bit 3 (SND) = 1, the transmit buffer will (U3-2 Figure 4) be disabled.

The purpose of subroutine CHPENA is to set the Chip Enable line on the CLOCK/RAM low. CHPENA is entered with both SCLK and $\overline{CE}$ high. The first OUT instruction sets SCLK low while keeping $\overline{CE}$ high (See Figure 6 — Point A). The second OUT instruction sets $\overline{CE}$ low while leaving SCLK low and the transmit buffer (SND) enabled (See Figure 6 — Point B).

Subroutine SNDBYT is called to send a data byte to the CLOCK/RAM chip. SNDBYT is entered with $\overline{CE}$ and SCLK already set low (See Figure 7 — Point A). The Z80 register D contains the byte to be sent to the CLOCK/RAM. One bit of the data byte is shifted into the Carry bit from register D, then shifted from the Carry bit into data bit 0 of the Accumulator. The data bit is then latched into the I/O Port latch with an OUT instruction (See Figure 7 — Point B). During the same OUT instruction, SCLK and $\overline{CE}$ are left low, and SND is left enabled.

```
;
; SUBROUTINE CHPENA
;           ON ENTRY:  3805 IS DISABLED
;
;           ON EXIT:   3805 CHIP ENABLE IS LOW
;                      3805 SCLK IS LOW, AND SND IS ENABLED
;
CHPENA      LD     A, 02H      ; SET SCLK = 0 WITH CE = 1
            OUT    (10H), A    ;  SND IS ENABLED
            LD     A,00H       ; SET CE = 0 WITH SCLK = 0
            OUT    (10H), A    ;  SND IS ENABLED
            RET                ; RETURN
;
```

CHPENA TIMING
Figure 6



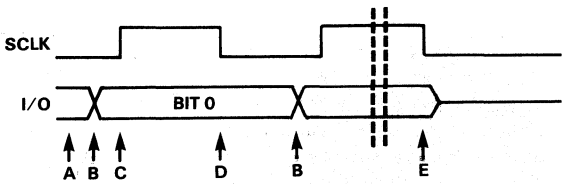
The SCLK bit in the Accumulator is then set, and the next OUT instruction sets SCLK high (See Figure 7 — Point C). The SCLK bit in the Accumulator is then set low, but, before the bit is sent to the CLOCK/RAM, a determination is made to see if all 8 bits have been sent. This is a delay tactic to ensure that a minimum SCLK high time is achieved. If there are bits remaining to be sent, the program jumps to NXTBIT, where the OUT instruction sets SCLK low (See Figure 7 —Point D).

When all 8 bits have been sent, bit 3 (SND) is set and the OUT instruction, just prior to return, sets SCLK low and disables the transmit buffer, while keeping $\overline{CE}$ low (See Figure 7 — point E).

If the byte sent was an ADDRESS/COMMAND byte for a CLOCK/RAM read operation, disabling the transmit buffer simultaneously with setting SCLK low will prevent any possibility of bus contention on the CLOCK/RAM I/O pin. Since the last bit of an ADDRESS/COMMAND byte is always a "1", the CLOCK/RAM I/O pin will remain at a logic 1 level until the CLOCK/RAM begins driving the pin.

```
;
; SUBROUTINE SNDBYT
;
;           ON ENTRY:  D CONTAINS DATA BYTE TO BE WRITTEN
;                      CE AND SCLK SHOULD ALREADY BE LOW.
;                      SND SHOULD BE ALREADY ENABLED.
;
;           ON EXIT:   CE AND SCLK=, AND SND IS DISABLED
;
SNDBYT      LD     B,8         ; LOAD B WITH BIT COUNT
            LD     A,00H       ; SCLK=0, CE=0, SND=ENABLE
NXTBIT      OUT    (10H),A
            SRL    A
            SRL    D           ; SHIFT DATA BIT INTO CARRY
            RLA                ; SHIFT CARRY INTO ACCUM
            OUT    (10H),A     ; OUTPUT DATA BIT
            SET    2,A         ; SET SCLK=1
            OUT    (10H),A
            RES    2,A         ; SET SCLK=0
            DEC    B           ; DECREMENT BIT COUNTER
            JR     NZ,NXTBIT   ; JUMP IF 8 BITS NOT SENT YET
            SET    3,A         ; SCLK=0, CE=0, SND=DISABLE
            OUT    (10H),A
            RET                ; RETURN
```

SNDBYT TIMING
Figure 7



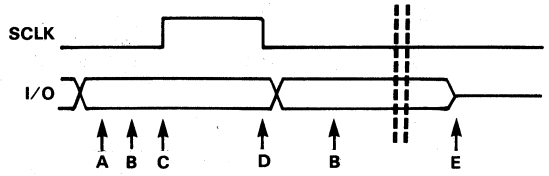
The purpose of subroutine GETBYT is to read one byte of data from the CLOCK/RAM. On entry to GETBYT, Z80 register pair HL points to the location in memory where the data byte is to be stored. Both SCLK and $\overline{CE}$ are also low (See Figure 8 — Point A). At this point, the CLOCK/RAM should be driving the I/O pin with the first valid data bit.

In order to read a byte from the CLOCK/RAM, an ADDRESS/COMMAND word had to have been written to the CLOCK/RAM, using subroutine SNDBYT. On exit from the SNDBYT routine, the last time SCLK was written low, and the CLOCK/RAM began transmitting data by placing the first data bit on the I/O pin. Since there were no further transitions of SCLK, the first valid data bit from the CLOCK/RAM should still be on the I/O pin.

```
;
; SUBROUTINE GETBYT
;
;          ON ENTRY:  HL POINTS TO LOCATION BYTE IS TO BE
;                     STORED
;
;          ON EXIT:   SCLK AND CE ARE LOW, SND IS DISABLED
;
GETBYT     LD         B,8          ; SET BIT COUNT TO 8
           LD         C,10H        ; SET C TO I/O PORT 10H
GETBIT     LD         D,0CH        ; SCLK=1, CE=0, SND=DISABLED
           IN         A,(10H)      ; READ A BIT FROM 3805
           OUT        (C),D        ; SET SCLK=1
           AND        80H          ; MASK UNWANTED BITS
           LD         D,08H        ; SCLK=0, CE=0, SND=DISABLED
           OR         E            ; MERGE DATA BIT INTO REG E
           LD         E,A
           SRL        E            ; SHIFT BIT RIGHT 1
           OUT        (C),D        ; SET SCLK=0
           DEC        B            ; DECREMENT BIT COUNT
           JR         NZ,GETBIT    ; JUMP IF THERE ARE MORE BITS
           LD         (HL),A       ; STORE RECEIVED BYTE AT HL
           RET                     ; RETURN
```

GETBYT TIMING

Figure 8



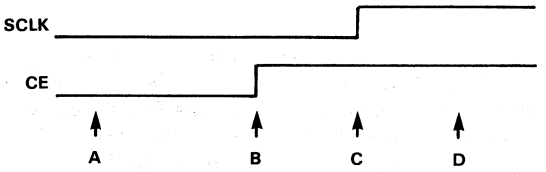
The IN instruction reads a data bit into bit 7 of the Accumulator (See Figure 8 — Point B). The OUT instruction, which follows, sets SCLK high (See Figure 8 — Point C). The data bit is then masked and merged into Z80 register E for storage; this is accomplished prior to setting SCLK back low to ensure that the minimum SCLK high specification is met. The next OUT instruction sets SCLK low (See Figure 8 —Point D), which clocks another data bit out of the CLOCK/RAM.

If all 8 bits have been read, the data byte is stored at the memory location pointed to by HL. On exit from GETBYT, SCLK and $\overline{CE}$ are low, and the transmit buffer (Figure 4/U3-2) is disabled.

```
;
; SUBROUTINE CHPDIS
;
;          ON ENTRY:  3805 IS ENABLED
;
;          ON EXIT:   CHIP ENABLE IS HIGH
;                     3805 SCLK IS HIGH, AND SND IS DISABLED
;
CHPDIS     LF         A,0AH        ; SET CE HIGH WITH SCLK=0
           OUT        (10H),A      ;  SND IS DISABLED
           LD         A,0EH        ; SET SCLK HIGH WITH CE HIGH
           OUT        (10H),A      ;  SND IS DISABLED
           RET                     ; RETURN
;
```

CHPDIS TIMING

Figure 9



The purpose of subroutine CHPDIS is to bring the CLOCK/RAM Chip Enable pin back high, disabling the chip. On entry to CHPDIS, SCLK and $\overline{CE}$ are both low (See Figure 9 — Point A). The first OUT instruction brings $\overline{CE}$ high (Figure 9 — Point B). The second OUT instruction brings SCLK high (Figure 9 — Point C). On exit from CHPDIS, SCLK and $\overline{CE}$ are high, and the transmit buffer (SND) is disabled (Figure 9 —Point D).

PROGRAMMING EXAMPLE

An example on the proper calling sequence of the previous four subroutines is given below. The following sequence would be used to read one byte of data from the CLOCK/RAM. The ADDRESS/COMMAND byte (8FH) will cause a read of the Clock Control register:

```
CALL    CHPENA    ;
LD      A,8FH     ; ADDRESS/COMMAND BYTE
CALL    SNDBYT    ; SEND BYTE
LD      HL, INBUF ; LOCATION TO STORE RECEIVED BYTE
CALL    GETBYT    ; GET BYTE FROM 3805
CALL    CHPDIS    ; DISABLE 3805
```

The MK3805 is first enabled by calling CHPENA. Next, the Accumulator is loaded with the ADDRESS/COMMAND byte for a Clock Control register read (8FH). The ADDRESS/COMMAND byte is sent to the MK3805 by calling subroutine SNDBYT. Then by calling GETBYT, a data byte, (the Clock Control register), is read from the MK3805 and stored at location HL. The MK3805 is then disabled by calling subroutine CHPDIS.

When Burst Mode operation is desired, the following sequence of instructions would be used. The following example uses an ADDRESS/COMMAND byte for a Clock Burst write (0BEH). The clock burst write mode sends 8 data bytes to the MK3805 after the ADDRESS/COMMAND byte is sent:

```
        CALL    CHPENA
        LD      D,WCKBST    ; SEND ADDRESS/COMMAND
                              WRITE
        CALL    SNDBYT      ;  CLOCK BURST MODE
        LD      HL,CKDATI   ; POINT AT CLOCK DATA
        LD      B,8         ; LOAD CLOCK BURST COUNTER
NXTWRD  LD      D,(HL)      ; GET CLOCK DATA BYTE
        PUSH    BC
        CALL    SNDBYT      ; WRITE BYTE TO CLOCK
                              REGISTERS
        INC     HL
        POP     BC          ; RECOVER BURST COUNT
        DJNZ    NXTWRD      ; HAVE ALL REGISTERS BEEN
                              WRITTEN
        CALL    CHPDIS      ; DISABLE 3805
```

ALTERNATE APPROACH

If the CLOCK/RAM is to be used in a system with a Parallel I/O Controller, such as the Mostek MK3881 PIO, chips U1 and U2 in Figure 4 could be eliminated. The concept of using I/O Port bits to generate SCLK, $\overline{CE}$, I/O data, and SND would remain the same, only the signals would be coming from the PIO port pins rather than from a discrete latch, such as the 74LS175 in Figure 4.

PROGRAM LISTING

The Program listing, CLOCK.RAM, is a Z80 program written to set and read the Clock registers in the MK3805 CLOCK/RAM, using the interface in Figure 4 and the Mostek FLP-80 Disk Operating System.

SERIAL COMMUNICATION CHIP INTERFACE

The new Mostek MK3801 Serial Timer Interrupt Controller (STI) provides a reliable and simple interface between a Z80 and the MK3805 CLOCK/RAM.

The MK3801 Z80 STI (Serial Timer Interrupt) is a Z80 microprocessor peripheral designed to serve a broad range of applications. By incorporating multiple functions within the Z80 STI, the designer is offered maximum flexibility while keeping the device count to a minimum. The STI integrates four functions within a 40 pin package: Binary Timers, Parallel I/O, Interrupts, and a USART. Given these features, the STI becomes a versatile device which can serve not only a specific design requirement, but a combination thereof. A few examples of these features include:

* Full Duplex USART with modem controls, DMA Handshake, and Baud rate generator
* 8 bit parallel I/O port with timers
* Multifunctional Programmable Timers with Interrupts
* Interrupt Controller

The Interrupt Controller includes 16 prioritized, vectored interrupts which provide maximum speed and efficiency in servicing the various device functions. If interrupts are not desired, each channel may be operated in a polled mode. The STI was designed not only to interface to the Z80 CPU, but also to virtually any microprocessor. Because the STI uses an asynchronous clock, all timing parameters are referenced from the control signals (unlike other Z80 peripherals, which are referenced to the system clock). There is also a special provision for handling interrupts in non-Z80 systems.

The STI has several features which facilitate interfacing to the MK3805.

It is possible to use Timer C or D to generate the required transmit and receive clocks. The Serial Out (SO) line can be tri-stated after the last data bit is transmitted, and one marking bit is sent prior to the beginning of data transmission, which aids in synchronization of the MK3805 to the incoming data stream.

STI INTERFACE OPERATION

The STI USART will be operated in the synchronous, divide by 1 Mode. This will prevent start, stop, and parity bits in the data stream, when operating the MK3805 in the Burst Mode. Since the MK3805 requires an ADDRESS/COMMAND byte prior to each operation, that byte will be the STI Sync byte also. Thus, when data is to be read from the CLOCK/RAM, the ADDRESS/COMMAND byte for a read is transmitted to the MK3805 and the MK3801 receiver simultaneously. Since the byte is the MK3801 Sync byte, the STI receiver will achieve Sync just as the CLOCK/RAM I/O pin switches from Input to Output and begins transmitting data to the STI.

STI SOFTWARE SUBROUTINES

Communication between the STI and CLOCK/RAM is accomplished by calling a specific sequence of Z80 subroutines. The six subroutines are shown below:

```
;
;           SUBROUTINE CHIP ENABLE (CHPENA)
;
;           ON ENTRY:  Serial Output (SO) should be low.
;
;           ON EXIT:   External J-K Flip Flop is enabled.
;                      Receive Buffer and Status Register cleared.
;
CHPENA      LD      A,80H       ; ENABLE 3805 CE FLIP-FLOP
            OUT     (GPIP),A    ;  TO LOOK FOR 1ST "ONE" BIT
CLRBUF      IN      A,(UDR)     ; CLEAR RECEIVER BUFFER
            IN      A,(RSR)     ; WAS THERE A PREVIOUS
                                    OVERRUN?
            BIT     6,A
            JR      NZ,CLRBUF   ; JUMP IF YES, CLEAR IT
            RET
;
```

```
;
;           SUBROUTINE ADDRESS/COMMAND FOR WRITE
;               (ADCMDW)
;
;           ON ENTRY:  "A" contains Address/Command Byte for a
;                      write to the 3805.
;
;           ON EXIT:   Transmitter is enabled, Receiver is not
;                      enabled.
;
ADCMDW      OUT     (IDR),A     ; OUTPUT BYTE TO SYNC
                                    REGISTER
            LD      A,03M
            OUT     (TSR),A     ; ENABLE XMITTER, (SO) LOW
            RET
;
;
;           SUBROUTINE ADDRESS/COMMAND FOR A READ
;               (ADCMDR)
;
;           ON ENTRY:  "A" contains Address/Command Byte for a
;                      Read from the 3805.
;
;           ON EXIT:   Transmitter is disabled with (SO) set to
;                      tristate when transmitter stops sending.
;
ADCMDR      OUT     (IDR),A     ; OUTPUT BYTE TO SYNC
                                    REGISTER
            LD      A,03H
            OUT     (RSR),A     ; ENABLE RECEIVER, STRIP
                                    SYNC, SEARCH
            OUT     (TSR),A     ; ENABLE XMITTER, (SO) LOW
            LD      B,08H       ; DELAY COUNT
WAIT1       DEC     B           ; DELAY TO ENSURE TRANS-
            JR      NZ,WAIT1    ;   MITTER HAS BEGUN TRANS-
                                ;   MITTING
            LD      A,00H       ; DISABLE XMITTER, SET (SO) TO
            OUT     (TSR),A     ;   TRISTATE WHEN XMITTER
                                ;   STOPS.
            RET
;
;
;           SUBROUTINE READ BYTE (RDBYTE)
;
;           ON EXIT:  Location at HL contains received data byte.
;
RDBYTE      IN      A,(RSR)     ; GET RECEIVE BUFFER FULL BIT
            BIT     7,A         ; IS IT SET?
            JR      Z,RDBYTE    ; JUMP IF NOT
            IN      A,(UDR)     ; GET DATA BYTE
            LD      (HL),A      ; STORE IT
            RET
;
;           SUBROUTINE SEND BYTE (SNDBYT)
;
;           ON ENTRY:  "A" contains data byte to be sent to 3805.
;
SYNDYT      LD      C,TSR
WAIT2       IN      B,(C)       ; GET XMIT BUFFER EMPTY BIT
            BIT     7,8         ; IS IT SET?
            JR      Z,WAIT2     ; JUMP IF NOT
            OUT     (UDR),A     ; WRITE DATA BYTE TO USART
            RET
;
;           SUBROUTINE CHIP DISABLE (CHPDIS)
;
;           ON EXIT:  (SO) is low and 3805 CE is high
;
CHPDIS      LD      A,00H       ;
            OUT     (RSR),A     ; DISABLE RECEIVER
WAIT3       IN      A,(TSR)     ; GET XMIT BUFFER EMPTY BIT
            BIT     7,A         ; IS BUFFER EMPTY?
            JR      Z,WAIT3     ; JUMP IF NOT AND WAIT
            LD      A,00H       ;
            OUT     (TSR),A     ; DISABLE XMITTER, SET (SO)
                                ;   TRISTATE
WAIT4       IN      A,(TSR)     ; GET "END" BIT
            BIT     4,A         ; HAS XMITTER STOPPED
                                ;   SENDING?
            JR      Z,WAIT4     ; JUMP IF NOT AND WAIT
            LD      A,00H
            OUT     (GPIP),A    ; SET 3805 CE HIGH
            LD      A,02H       ; SET (SO) LOW
            OUT     (TSR),A
            RET
;
```

Subroutine CHPENA sets GPIP7 high which enables the 74LS73 flip flop to start looking for the first mark bit. Next the subroutine clears the receiver buffer.

Subroutine ADCMDW is called when there is to be a write to the MK3805 operation. The routine loads the ADDRESS/COMMAND byte into the STI Sync register and then enables the transmitter. The receiver is not enabled because this is to be a WRITE only operation.

Subroutine ADCMDR is called when there is to be a READ from the MK3805 operation. The routine loads the ADDRESS/COMMAND byte into the STI Sync register, then enables the receiver followed by the transmitter. The delay loop at WAIT1 is to ensure the STI Transmitter has started transmitting before the Transmitter Enable bit is reset. The transmitter is set to disable when the last bit of the Sync byte is sent and the Serial Out (SO) line is set to tri-state at the same time. Tri-stating (SO) allows a direct connection between the STI Serial Out (SO) and Serial In (SI) pins.

Subroutine RDBYTE gets one data byte from the STI receiver buffer and stores it at the memory location pointed to by the Z80 register pair HL.

Subroutine SNDBYT loads one data byte into the STI transmit buffer.

Subroutine CHPDIS disables the receiver, ensures the transmitter has completed transmitting, then disables it. Then GPIP7 is set low, which clears the external 74LS73 latch, causing the MK3805 $\overline{CE}$ pin to go high. Last, the STI Serial Out pin is set low, which leaves the STI and external latch setup for the next data transfer.

The schematic for the STI interface is shown in Figure 10. The STI Timer C is used as a Baud rate clock generator.

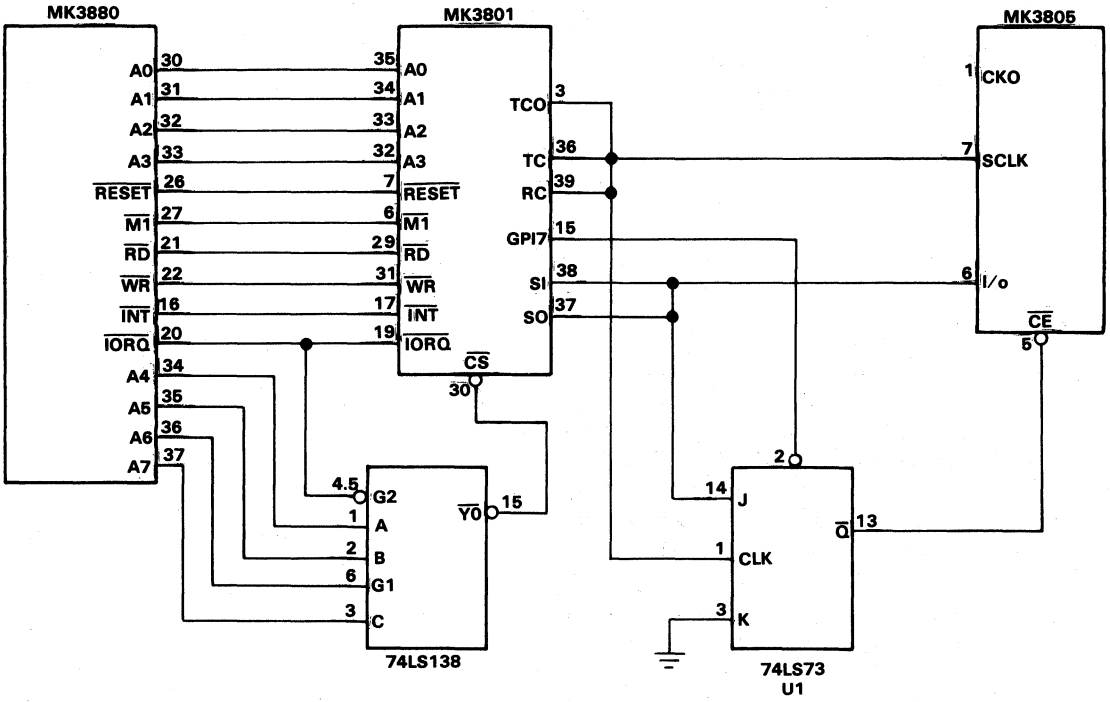
PROGRAMMING EXAMPLE

The proper sequence of Z80 subroutine calls to read one byte from the MK3805 CLOCK/RAM is shown below. The ADDRESS/COMMAND byte (8FH) is for a Read Clock Control Register operation:

```
CALL    CHPENA      ;
LD      HL,INBUF    ; SET HL TO INPUT BUFFER LOCATION
LD      A,8FH       ; READ CLOCK CONTROL REGISTER
CALL    ADCMDR      ; SEND AN ADDRESS/COMMAND READ
CALL    RDBYTE      ; GET ONE BYTE FROM THE MK3805
CALL    CHPDIS      ; DISABLE 3805 and SETUP FOR
                    ;   NEXT DATA TRANSFER
```

MK3801-STI TO MK3805-CLOCK/RAM INTERFACE

Figure 10



The following instruction sequence would be used to do a Burst Mode read of the 8 Clock registers:

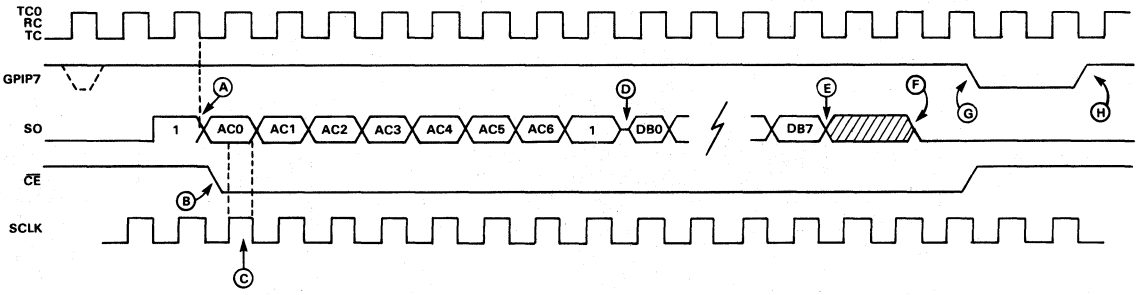
```
KREAD   CALL   CHPENA    ; ENABLE CHIP 3805
        DL     A,RCKBST  ; SEND ADDRESS/COMMAND
        CALL   ADCMDR    ;  READ CLOCK CONTROL
        LD     HL,CKDAT0 ; SET HL TO POINT AT LOCATION
        LD     B,8       ; SET BURST BYTE COUNT
XWRD    PUSH   BC        ; SAVE BURST BYTE COUNT
        CALL   RDBYTE    ; READ A BYTE FROM 3805
        INC    HL
        POP    BC        ; RECOVER BURST BYTE COUNT
        DJNZ   NXWRD     ; HAVE 8 BYTES BEEN READ?
        CALL   CHPDIS    ; DISABLE CHIP
```

INTERFACE TIMING

Figure 11 shows the timing associated with reading one byte of data from the CLOCK/RAM. The STI is initialized by setting H and L in the Transmit Status Register (RSR) so the Serial Out pin will be low. Then GPIP7 is written low and back high to ensure that the 74LS73 latch (Figure 10) is cleared, causing the MK3805 $\overline{CE}$ pin to be high.

STI INTERFACE TIMING

Figure 11



The first step is to call subroutine CHPENA, which ensures GPIP7 is high, enabling the 74LS73 latch to start looking for the first mark bit. Next, we call subroutine ADCMDR to send the ADDRESS/COMMAND byte for a READ operation. During this routine the transmitter is enabled and a mark bit is transmitted. The falling edge of TC clocks the bit into the 74LS73 latch (See Figure 11 - Point A). This results in Q of the latch going low, which pulls the MK3805 $\overline{CE}$ pin low (Figure 11 - Point B).

The next high cycle of SCLK clocks the ADDRESS/COMMAND least significant bit (AC0) into the MK3805 (Figure 11 — Point C). Once the transmitter is enabled, and has started transmitting, we write H and L in the TSR, so the STI Serial Out pin will tri-state after the last Sync bit is sent. The transmitter is then disabled without loading any data into the transmit buffer. When the transmitter is disabled while transmitting a Sync character, transmission will continue until the entire Sync character has been sent. Then the transmitter automatically disables, and in this case will cause the Serial Out line to tri-state immediately.

After the last bit of the ADDRESS/COMMAND Sync character is sent, the receiver should have achieved sync and the Serial Out line should be tri-stated (Figure 11 — Point D).

Subroutine RDBYTE is called and starts looking for the Receive Buffer Full bit to be set. When the buffer is full the received data byte is read (Figure 11 — Point E).

When the data has been received, subroutine CHPDIS is called which sets the STI Serial Out pin low (Figure 11 —Point F). GPIP7 is then written low and high (Figure 11 -Points G and H) which causes the MK3805 $\overline{CE}$ pin to go high, disabling the chip.

To write data to the CLOCK/RAM, subroutine ADCMDW is called to send an ADDRESS/COMMAND write operation byte to the MK3805. Subroutine SNDBYT is then called for each data byte that is to be sent to the CLOCK/RAM.

PROGRAM LISTING

Program listing STI.SRC is a Z80 program written to set and read the clock registers in the MK3805 CLOCK/RAM using the interface in Figure 10 and the Mostek FLP-80 Disk Operating System. The messages at the end of the program are identical to the messages shown in the CLOCK.RAM listing and are not repeated.

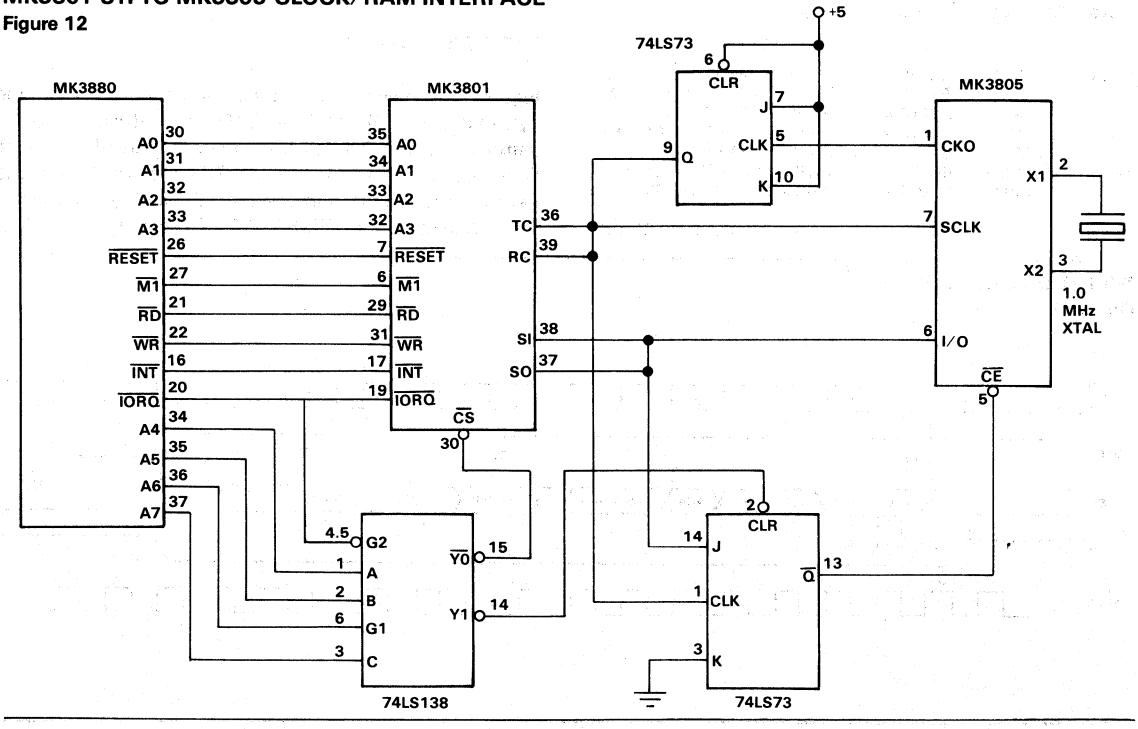
STI INTERFACE ALTERNATIVES

Figure 12 shows two modifications of the STI interface.

The maximum SCLK rate of the MK3805 is 250 KHz. The CKO output of the MK3805 CLOCK/RAM powers up at 1/2 the crystal frequency. By using the second half of the 74LS73 flip flop, and a 1.0 MHz crystal, a 250 KHz clock can be generated. This frees up the STI Timer C.

Additionally, if an extra I/O Port is decoded, it can be used to clear the 74LS73 latch (Figure 12). This frees up the STI FPIP7 pin.

MK3801-STI TO MK3805-CLOCK/RAM INTERFACE
Figure 12



```
ADDR  OBJECT     ST # SOURCE STMT    DATASET = DK0:CLOCK .RAM[1]  21-AUG-81

                 0001 ;
                 0002 ;                CLOCK.RAM
                 0003 ;
                 0004 ;
                 0005 ;
                 0006 ;        OCTOBER 4, 1981          6:00  PM
                 0007 ;
 >008E           0008 WCKCNL   EQU     8EH      ;WRITE TO CLOCK CONTROL REGISTER
 >0008           0009 DISWP    EQU     08H      ;DISABLE WRITE PROTECT DATABYTE
                 0010                           ;8.0 MHZ XTAL FREQ
 >00BE           0011 WCKBST   EQU     0BEH     ;ENABLE WRITE CLOCK BURST MODE
 >00BF           0012 RCKBST   EQU     0BFH     ;ENABLE READ CLOCK BURST MODE
 >008F           0013 RCKCNL   EQU     8FH      ;READ FROM CLOCK CONTROL REGISTER
                 0014 ;
'0000  CD9500'   0015 START    CALL    CHPENA
'0003  168F      0016          LD      D,RCKCNL         ;READ CLOCK CONTROL REGISTER
'0005  CD6100'   0017          CALL    SNDBYT
'0008  21E300'   0018          LD      HL,INBUF         ;LOCATE CONTROL BYTE AT INBUF
'000B  CD7C0C'   0019          CALL    GETBYT           ;GO GET CLOCK CONTROL REG
                 0020 ;
                 0021 ;
'000E  CD9E00'   0022 D        CALL    CHPDIS           ;DISABLE 3805
'0011  7E        0023          LD      A,(HL)
'0012  CB7F      0024          BIT     7,A              ;IS WRITE PROTECT BIT SET?
'0014  C44000'   0025          CALL    NZ,PWRINT        ;IF YES DO POWER ON
                 0026 ;                                          INITIALIZATION
'0017  1E01      0027          LD      E,1              ;SET LUN = CONSOLE
'0019  CDFFFF    0028          CALL    CRLF
'001C  CD1A00'   0029          CALL    CRLF
'001F  217B02'   0030          LD      HL,MSG1
'0022  CDFFFF    0031          CALL    PTXT             ;PRINT MENU MESSAGE
'0025  CD1D00'   0032          CALL    CRLF
'0028  CDB600'   0033          CALL    GETLN            ;GET FUNCTION REQUESTED NUMBER
'002B  7E        0034          LD      A,(HL)           ;LOAD C WITH FUNCTION NUMBER
'002C  D630      0035          SUB     30H
'002E  4F        0036          LD      C,A
'002F  0600      0037          LD      B,00
'0031  CB21      0038          SLA     C
'0033  DD210B01' 0039          LD      IX,FUNPT
'0037  DD09      0040          ADD     IX,BC
'0039  DD6E00    0041          LD      L,(IX+0)
'003C  DD6601    0042          LD      H,(IX+1)
'003F  E9        0043          JP      (HL)
                 0044 ;
'0040  CD9500'   0045 PWRINT   CALL    CHPENA           ;ENABLE 3805
'0043  168E      0046          LD      D,WCKCNL         ;SEND ADDRESS/COMMAND TO
'0045  CD6100'   0047          CALL    SNDBYT           ;  WRITE CLOCK CONTROL REGISTER
'0048  1608      0048          LD      D,DISWP          ;SEND WRITE PROTECT DISABLE TO
'004A  CD6100'   0049          CALL    SNDBYT           ;   CLOCK CONTROL REGISTER
'004D  CD9E00'   0050          CALL    CHPDIS
'0050  CD9500'   0051          CALL    CHPENA
'0053  168E      0052          LD      D,WCKCNL
'0055  CD6100'   0053          CALL    SNDBYT
'0058  1608      0054          LD      D,DISWP
'005A  CD6100'   0055          CALL    SNDBYT
'005D  CD9E00'   0056          CALL    CHPDIS
'0060  C9        0057          RET
                 0058 ;
```

V

```
ADDR  OBJECT    ST # SOURCE STMT
                0059 ; SUBROUTINE SNDBYT
                0060 ;
                0061 ;        ON ENTRY: D CONTAINS DATA BYTE TO BE WRITTEN
                0062 ;                  CE AND SCLK SHOULD ALREADY BE LOW.
                0063 ;                  SND SHOULD BE ALREADY ENABLED.
                0064 ;
                0065 ;        ON EXIT:  CE AND SCLK = 0, AND SND IS DISABLED
                0066 ;
'0061  0608     0067 SNDBYT  LD      B,8              ;LOAD B WITH BIT COUNT
'0063  3E00     0068         LD      A,00H            ;SCLK=0, CE=0, SND=ENABLE
'0065  D310     0069 NXTBIT  OUT     (10H),A
'0067  CB3F     0070         SRL     A
'0069  CB3A     0071         SRL     D                ;SHIFT DATA BIT INTO CARRY
'006B  17       0072         RLA                      ;SHIFT CARRY INTO ACCUM
'006C  D310     0073         OUT     (10H),A          ;OUTPUT DATA BIT
'006E  CBD7     0074         SET     2,A              ;SET SCLK = 1
'0070  D310     0075         OUT     (10H),A
'0072  CB97     0076         RES     2,A              ;SET SCLK = 0
'0074  05       0077         DEC     B                ;DECREMENT BIT COUNTER
'0075  20EE     0078         JR      NZ,NXTBIT        ;JUMP IF 8 BITS NOT SENT YET
'0077  CBDF     0079         SET     3,A              ;SCLK=0, CE=0, SND=DISABLE
'0079  D310     0080         OUT     (10H),A
'007B  C9       0081         RET                      ;RETURN
                0082 ;
                0083 ;
                0084 ; SUBROUTINE GETBYT
                0085 ;
                0086 ;        ON ENTRY:  HL POINTS TO LOCATION BYTE TO BE STORED
                0087 ;
                0088 ;        ON EXIT:   SCLK AND CE ARE LOW, SND IS DISABLED
                0089 ;
'007C  0608     0090 GETBYT  LD      B,8              ;SET BIT COUNT
'007E  0E10     0091         LD      C,10H            ;SET C TO I/O PORT 10
'0080  160C     0092 GETBIT  LD      D,0CH            ;SCLK=1 , CE=0, AND SND IS DISABLED
'0082  DB10     0093         IN      A,(10H)          ;READ A BYTE FROM 3805
'0084  ED51     0094         OUT     (C),D            ;SET SCLK=1
'0086  E680     0095         AND     80H              ;MASK UNWANTED BITS
'0088  1608     0096         LD      D,08H            ;SCLK=0 , CE=0, AND SND IS DISABLED
'008A  B3       0097         OR      E                ;MERGE DATA BIT INTO REG E
'008B  5F       0098         LD      E,A
'008C  CB3B     0099         SRL     E                ;SHIFT BIT RIGHT 1
'008E  ED51     0100         OUT     (C),D            ;SET SCLK=0
'0090  05       0101         DEC     B                ;DECREMENT BIT COUNTER
'0091  20ED     0102         JR      NZ,GETBIT
'0093  77       0103         LD      (HL),A           ;STORE RECEIVED BYTE AT HL
'0094  C9       0104         RET                      ;RETURN
                0105 ;
                0106 ;
                0107 ;  SUBROUTINE CHPENA
                0108 ;
                0109 ;        ON ENTRY:  3805 IS DISABLED
                0110 ;
                0111 ;        ON EXIT:   3805 CHIP ENABLE IS LOW
                0112 ;                   3805 SCLK IS LOW, AND SND IS ENABLED
                0113 ;
'0095  3E02     0114 CHPENA  LD      A,02H            ; SET SCLK=0 WITH CE=1
'0097  D310     0115         OUT     (10H),A          ;   SND IS ENABLED
'0099  3E00     0116         LD      A,00H            ; SET CE=0 WITH SCLK=0
```

```
ADDR  OBJECT    ST # SOURCE STMT
'009B D310      0117         OUT     (10H),A          ;   SND IS ENABLED
'009D C9        0118         RET                      ; RETURN
                0119 ;
                0120 ;
                0121 ;  SUBROUTINE CHPDIS
                0122 ;
                0123 ;       ON ENTRY:  3805 IS ENABLED
                0124 ;
                0125 ;       ON EXIT:   CHIP ENABLE IS HIGH
                0126 ;                  3805 SCLK IS HIGH, AND SND IS DISABLED
                0127 ;
'009E 3E0A      0128 CHPDIS  LD      A,0AH            ;SET CE HIGH WITH SCLK=0
'00A0 D310      0129         OUT     (10H),A          ;    SND IS DISABLED
'00A2 3E0E      0130         LD      A,0EH            ;SET SCLK HIGH WITH CE HIGH
'00A4 D310      0131         OUT     (10H),A          ;    SND IS DISABLED
'00A6 C9        0132         RET                      ;RETURN
                0133 ;
                0134 ;
                0135 ;  SUBROUTINE ASCBIN
                0136 ;       CONVERTS TWO CONSECUTIVE BYTES FROM ASCII TO ONE BINARY
                0137 ;                BYTE.
                0138 ;
'00A7 7E        0139 ASCBIN  LD      A,(HL)           ;GET FIRST ASCII BYTE
'00A8 CB27      0140         SLA     A
'00AA CB27      0141         SLA     A                ;SHIFT LEFT 4 LOCATIONS
'00AC CB27      0142         SLA     A
'00AE CB27      0143         SLA     A
'00B0 23        0144         INC     HL               ;POINT AT THE 2ND BYTE
'00B1 ED67      0145         RRD                      ;A CONTAINS BINARY BYTE
'00B3 23        0146         INC     HL
'00B4 23        0147         INC     HL
'00B5 C9        0148         RET
                0149 ;
                0150 ;
                0151 ;
'00B6 3E09      0152 GETLN   LD      A,9
'00B8 21E300'   0153         LD      HL,INBUF
'00BB 163F      0154         LD      D,'?'
'00BD CDFFFF    0155         CALL    JTASK
'00C0 C9        0156         RET
                0157 ;
                0158 ;
'00C1 00000000  0159 CKDATI  DEFB    0,0,0,0,0,0,0,0
      00000000
                0160 ;
'00C9 00000000  0161 CKDATO  DEFB    0,0,0,0,0,0,0,0
      00000000
                0162 ;
                0163 ;  SUBROUTINE BINASC
                0164 ;       ON ENTRY:   A CONTAINS BINARY NUMBER TO BE CONVERTED TO ASCII
                0165 ;       ON EXIT:    A AND C CONTAIN ASCII EQUIVALENT
                0166 ;
'00D1 47        0167 BINASC  LD      B,A              ;SAVE BINARY NUMBER
'00D2 E60F      0168         AND     0FH              ;
'00D4 F630      0169         OR      30H              ;CONVERT TO ASCII NUMBER
'00D6 4F        0170         LD      C,A              ;SAVE IN REG C
'00D7 78        0171         LD      A,B              ;RECOVER BINARY NUMBER
'00D8 CB3F      0172         SRL     A
```

V

```
 ADDR  OBJECT    ST # SOURCE STMT    SRL     A
'00DA  CB3F      0173          SRL     A
'00DC  CB3F      0174          SRL     A
'00DE  CB3F      0175          SRL     A
'00E0  F630      0176          OR      30H             ;CONVERT TO ASCII NUMBER
'00E2  C9        0177          RET
                 0178 ;
                 0179 ;
'>00E3           0180 INBUF    DEFS    40
                 0181 ;
'010B  1101'     0182 FUNPT    DEFW    CKLOAD
'010D  B501'     0183          DEFW    CKREAD
'010F  7602'     0184          DEFW    MONITR
                 0185 ;
                 0186 ;
'0111  0608      0187 CKLOAD   LD      B,8
'0113  3E00      0188          LD      A,0
'0115  21C100'   0189          LD      HL,CKDATI
'0118  77        0190 CRLOOP   LD      (HL),A
'0119  23        0191          INC     HL
'011A  05        0192          DEC     B
'011B  20FB      0193          JR      NZ,CRLOOP
'011D  21EF02'   0194          LD      HL,MSG2
'0120  CD2300'   0195          CALL    PTXT
'0123  CDB600'   0196          CALL    GETLN           ;GET DATE FROM TERMINAL
'0126  DD21C100' 0197          LD      IX,CKDATI       ;INITIALIZE IX TO CLOCK IN BUFFER
'012A  CDA700'   0198          CALL    ASCBIN          ;GO CONVERT MONTH TO BINARY BCD
'012D  DD7704    0199          LD      (IX+4),A        ;STORE BCD MONTH
'0130  CDA700'   0200          CALL    ASCBIN          ;GO CONVERT DAY
'0133  DD7703    0201          LD      (IX+3),A        ;STORE BCD DAY
'0136  CDA700'   0202          CALL    ASCBIN          ;GO CONVERT YEAR
'0139  DD7706    0203          LD      (IX+6),A        ;STORE BCD YEAR
'013C  217A03'   0204          LD      HL,MSG3
'013F  CD2101'   0205          CALL    PTXT
'0142  CDB600'   0206          CALL    GETLN           ;GET DAY OF WEEK FROM TERMINAL
'0145  7E        0207          LD      A,(HL)
'0146  D630      0208          SUB     30H             ;CONVERT FROM ASCII TO BINARY
'0148  DD7705    0209          LD      (IX+5),A        ;STORE BCD DAY OF THE WEEK
'014B  DDCB02BE  0210          RES     7,(IX+2)        ;RESET 12-24 MODE TO 24 HOUR MODE
'014F  21D703'   0211          LD      HL,MSG4
'0152  CD4001'   0212          CALL    PTXT
'0155  CDB600'   0213          CALL    GETLN           ;GET CLOCK MODE
'0158  7E        0214          LD      A,(HL)
'0159  FE31      0215          CP      31H             ;WAS 24 HOUR MODE SELECTED?
'015B  281A      0216          JR      Z,CLK24         ;JUMP IF YES
'015D  DDCB02FE  0217          SET     7,(IX+2)        ;SET 12 HOUR MODE BIT
'0161  215504'   0218          LD      HL,MSG5
'0164  CD5301'   0219          CALL    PTXT
'0167  CDB600'   0220          CALL    GETLN           ;GET AM OR PM RESPONSE
'016A  DDCB02EE  0221          SET     5,(IX+2)        ;SET PM INDICATOR
'016E  7E        0222          LD      A,(HL)
'016F  FE41      0223          CP      41H             ;WAS 1ST CHARACTER AN A?
'0171  2004      0224          JR      NZ,CLK24        ;JUMP IF NOT
'0173  DDCB02AE  0225          RES     5,(IX+2)        ;RESET AM/PM INDICATOR TO AM
'0177  217D04'   0226 CLK24    LD      HL,MSG6
'017A  CD6501'   0227          CALL    PTXT
'017D  CDB600'   0228          CALL    GETLN           ;GET TIME TO BE SET
'0180  CDA700'   0229          CALL    ASCBIN          ;CONVERT TO BCD
'0183  DDB602    0230          OR      (IX+2)          ;INCLUDE MODE BITS
```

```
ADDR   OBJECT     ST # SOURCE STMT    DATASET = DK0:CLOCK .RAM[1]  21-AUG-81

'0186  DD7702     0231          LD      (IX+2),A          ;STORE IN CLOCK BUFFER
'0189  CDA700'    0232          CALL    ASCBIN            ;GO CONVERT MINUTES
'018C  DD7701     0233          LD      (IX+1),A          ;STORE BCD MINUTES
'018F  CDA700'    0234          CALL    ASCBIN
'0192  DD7700     0235          LD      (IX+0),A          ;STORE BCD SECONDS
'0195  DD360708   0236          LD      (IX+7),DISWP      ;SET CLOCK CRYSTAL FREQ SELECT
'0199  CD9500'    0237          CALL    CHPENA
'019C  16BE       0238          LD      D,WCKBST          ;SEND ADDRESS/COMMAND WRITE
'019E  CD6100'    0239          CALL    SNDBYT            ;   CLOCK BURST MODE
'01A1  21C100'    0240          LD      HL,CKDATI         ;POINT AT CLOCK DATA
'01A4  0608       0241          LD      B,8               ;LOAD CLOCK BURST COUNTER
'01A6  56         0242 NXTWRD   LD      D,(HL)            ;GET CLOCK DATA BYTE
'01A7  C5         0243          PUSH    BC
'01A8  CD6100'    0244          CALL    SNDBYT            ;WRITE BYTE TO CLOCK REGISTERS
'01AB  23         0245          INC     HL
'01AC  C1         0246          POP     BC                ;RECOVER BURST COUNT
'01AD  10F7       0247          DJNZ    NXTWRD            ;HAS ALL REGISTERS BEEN WRITTEN TO?
'01AF  CD9E00'    0248          CALL    CHPDIS            ;DISABLE 3805
'01B2  C30000'    0249          JP      START             ;JUMP TO START AND PRINT MENU
                  0250 ;
                  0251 ;
'01B5  CD9500'    0252 CKREAD   CALL    CHPENA            ;ENABLE CHIP 3805
'01B8  16BF       0253          LD      D,RCKBST          ;SEND ADDRESS/COMMANC READ CLOCK
'01BA  CD6100'    0254          CALL    SNDBYT            ;  CONTROL
'01BD  21C900'    0255          LD      HL,CKDATO         ;SET HL TO POINT AT LOCATION
'01C0  0608       0256          LD      B,8               ;SET BURST BYTE COUNT
'01C2  C5         0257 NXWRD    PUSH    BC                ;SAVE BURST BYTE COUNT
'01C3  CD7C00'    0258          CALL    GETBYT            ;READ A BYTE FROM 3805
'01C6  23         0259          INC     HL
'01C7  C1         0260          POP     BC                ;RECOVER BURST BYTE COUNT
'01C8  10F8       0261          DJNZ    NXWRD             ;HAVE 8 BYTES BEEN READ?
'01CA  CD9E00'    0262          CALL    CHPDIS            ;DISABLE CHIP
'01CD  DD21F905'  0263          LD      IX,DAYWK          ;LOAD IX WITH START OF MESSAGE LOC
'01D1  21C900'    0264          LD      HL,CKDATO         ;LOAD HL WITH CLOCK OUTPUT DATA
'01D4  7E         0265          LD      A,(HL)            ;GET 1ST BYTE OF DATA
'01D5  CB7F       0266          BIT     7,A               ;IS CLOCK IN HALTED MODE?
'01D7  C46702'    0267          CALL    NZ,CLKHLT         ;CALL HALT WARNING ROUTINE
'01DA  CDD100'    0268          CALL    BINASC            ;CONVERT BINARY SECONDS TO ASCII
'01DD  DD7746     0269          LD      (IX+70),A         ;STORE TENS OF SECONDS
'01E0  DD7147     0270          LD      (IX+71),C         ;STORE UNITS OF SECONDS
'01E3  23         0271          INC     HL
'01E4  7E         0272          LD      A,(HL)            ;GET BINARY MINUTES
'01E5  CDD100'    0273          CALL    BINASC            ;GO CONVERT TO ASCII
'01E8  DD7743     0274          LD      (IX+67),A         ;STORE TENS OF MINUTES
'01EB  DD7144     0275          LD      (IX+68),C         ;STORE UNITS OF MINUTES
'01EE  23         0276          INC     HL
'01EF  7E         0277          LD      A,(HL)            ;GET HOURS
'01F0  012020     0278          LD      BC,'  '           ;LOAD BC WITH ASCII BLANKS
'01F3  CB7F       0279          BIT     7,A               ;IS 12 HOUR MODE BIT SET?
'01F5  280C       0280          JR      Z,M24             ;JUMP IF NOT
'01F7  0E4D       0281          LD      C,'M'
'01F9  0641       0282          LD      B,'A'             ;LOAD B WITH AN ASCII "A"
'01FB  CB6F       0283          BIT     5,A               ;IS PM BIT SET?
'01FD  2802       0284          JR      Z,M241            ;JUMP IF NOT
'01FF  0650       0285          LD      B,'P'             ;LOAD B WITH AN ASCII "P"
'0201  E65F       0286 M241     AND     5FH               ;RESET BITS 5 AND 7
'0203  DD704A     0287 M24      LD      (IX+74),B         ;STORE ASCII AM OR PM
'0206  DD714B     0288          LD      (IX+75),C
```

V

```
 ADDR   OBJECT    ST # SOURCE STMT    CALL     BINASC          ;CONVERT HOURS TO ASCII
'0209   CDD100'   0289          CALL     BINASC
'020C   DD7740    0290          LD       (IX+64),A       ;STORE ASCII HOURS
'020F   DD7141    0291          LD       (IX+65),C
'0212   23        0292          INC      HL
'0213   7E        0293          LD       A,(HL)          ;GET DAY OF MONTH
'0214   CDD100'   0294          CALL     BINASC
'0217   DD7714    0295          LD       (IX+20),A       ;STORE ASCII DAY
'021A   DD7115    0296          LD       (IX+21),C
'021D   23        0297          INC      HL
'021E   7E        0298          LD       A,(HL)          ;GET MONTH IN BCD
'021F   FE10      0299          CP       10H             ;IS MONTH LESS THAN DECIMAL 10?
'0221   FA2802'   0300          JP       M,NOCONV        ;JUMP IF YES, NO BCD CONVERSION
'0224   C60A      0301          ADD      A,10            ;CONVERT BCD TO BINARY
'0226   CBA7      0302          RES      4,A             ;RESET BCD TENS DIGIT
'0228   47        0303 NOCONV   LD       B,A
'0229   E5        0304          PUSH     HL              ;SAVE HL
'022A   213A05'   0305          LD       HL,MCNTHL-10
'022D   110A00    0306          LD       DE,10           ;
'0230   19        0307 LOOP1    ADD      HL,DE           ;INCREMENT POINTER BY 10
'0231   10FD      0308          DJNZ     LOOP1           ;CONTINUE UNTIL CORRECT MONTH FOUND
'0233   010A00    0309          LD       BC,10           ;SET BYTE TRANSFER COUNT
'0236   110306'   0310          LD       DE,MONTH        ;DATA TO BE TRANSFERRED TO MONTH
'0239   EDB0      0311          LDIR                     ;TRANSFER ASCII MONTH
'023B   E1        0312          POP      HL              ;RECOVER POINTER
'023C   23        0313          INC      HL
'023D   46        0314          LD       B,(HL)          ;GET DAY OF WEEK
'023E   E5        0315          PUSH     HL
'023F   21F404'   0316          LD       HL,WEEKDA-10
'0242   110A00    0317          LD       DE,10
'0245   19        0318 LOOP2    ADD      HL,DE           ;INCREMENT POINTER BY 10
'0246   10FD      0319          DJNZ     LOOP2           ;CONTINUE UNTIL CORRECT DAY FOUND
'0248   010A00    0320          LD       BC,10
'024B   11F905'   0321          LD       DE,DAYWK
'024E   EDB0      0322          LDIR                     ;TRANSFER ASCII DAY TO DAYWK
'0250   E1        0323          POP      HL
'0251   23        0324          INC      HL
'0252   7E        0325          LD       A,(HL)          ;GET YEAR
'0253   CDD100'   0326          CALL     BINASC          ;CONVERT BINARY TO ASCII
'0256   DD771A    0327          LD       (IX+26),A
'0259   DD711B    0328          LD       (IX+27),C       ;STORE ASCII YEAR
'025C   21BC05'   0329          LD       HL,DATMSG
'025F   1E01      0330          LD       E,1
'0261   CD7B01'   0331          CALL     PTXT
'0264   C30000'   0332          JP       START           ;JUMP TO START AND PRINT MENU
                  0333 ;
                  0334 ;
                  0335 ;  SUBROUTINE CLKHLT
                  0336 ;
                  0337 ;        THIS ROUTINE PRINTS A WARNING THAT THE CLOCK IS HALTED.
                  0338 ;
'0267   E5        0339 CLKHLT   PUSH     HL
'0268   F5        0340          PUSH     AF
'0269   214806'   0341          LD       HL,HLTMSG
'026C   1E01      0342          LD       E,1
'026E   CD6202'   0343          CALL     PTXT
'0271   F1        0344          POP      AF
'0272   E1        0345          POP      HL
'0273   CBBF      0346          RES      7,A             ;CLEAR CLOCK HALT BIT
```

```
 ADDR   OBJECT     ST # SOURCE STMT     DATASET = DK0:CLOCK .RAM[1]  21-AUG-81

'0275   C9          0347          RET
                    0348 ;
                    0349 ;
'0276   3E01        0350 MONITR   LD      A,1
'0278   C3BE00'     0351          JP      JTASK            ;RETURN TO FLP-80DOS MONITOR
                    0352 ;
                    0353 ;
                    0354 ;
                    0355          INCLUDE MSG
                   +0001 ;
                   +0002 ;                    MSG.
                   +0003 ;                     BY
                   +0004 ;                JOHN KOVAR
                   +0005 ;
                   +0006 ;        OCTOBER 4, 1981          6:30 PM
                   +0007 ;
 >0A0D             +0008 CRL     EQU     0A0DH    ;CARRIAGE RETURN - LINE FEED
 >0003             +0009 ETX     EQU     03H      ;END OF TEXT
 >2020             +0010 SP      EQU     2020H    ;TWO ASCII SPACE CHARACTERS
                   +0011 ;
                   +0012         GLOBAL  JTASK
                   +0013         GLOBAL  CRLF
                   +0014         GLOBAL  PTXT
                   +0015 ;
                   +0016 ;
'027B   454E5445   +0017 MSG1    DEFM    'ENTER NUMBER CORRESPONDING TO DESIRED FUNCTION'
        52204E55
        4D424552
        20434F52
        52455350
        4F4E4449
        4E472054
        4F204445
        53495245
        44204655
        4E435449
        4F4E
'02A9   0D0A       +0018         DEFW    CRL
'02AB   0930202D   +0019         DEFM    '        0 - SET CLOCK'
        20534554
        20434C4F
        434B
'02B9   0D0A       +0020         DEFW    CRL
'02BB   0931202D   +0021         DEFM    '        1 - READ CLOCK'
        20524541
        4420434C
        4F434B
'02CA   0D0A       +0022         DEFW    CRL
'02CC   0932202D   +0023         DEFM    '        2 - RETURN TO FLP-80DOS MONITOR'
        20524554
        55524E20
        544F2046
        4C502D38
        30444F53
        204D4F4E
        49544F52
'02EC   0D0A       +0024         DEFW    CRL
'02EE   03         +0025         DEFB    ETX
```

V

```
 ADDR   OBJECT     ST # SOURCE STMT    
                   +0026 ;
'02EF   0D0A0D0A   +0027 MSG2     DEFW     CRL,CRL
'02F3   454E5445   +0028          DEFM     'ENTER THE DATE IN THE FOLLOWING FORMAT:'
        52205448
        45204441
        54452049
        4E205448
        4520464F
        4C4C4F57
        494E4720
        464F524D
        41543A
'031A   0D0A       +0029          DEFW     CRL
'031C   094D4D2F   +0030          DEFM     '       MM/DD/YY     WHERE:'
        44442F59
        59202020
        20205748
        4552453A
'0330   0D0A0D0A   +0031          DEFW     CRL,CRL
'0334   09094D4D   +0032          DEFM     '                MM = MONTH  01 - 12'
        203D204D
        4F4E5448
        20203031
        202D2031
        32
'0349   0D0A       +0033          DEFW     CRL
'034B   09094444   +0034          DEFM     '                DD = DAY    01 - 31'
        203D2044
        41592020
        20203031
        202D2033
        31
'0360   0D0A       +0035          DEFW     CRL
'0362   09095959   +0036          DEFM     '                YY = YEAR   00 - 99'
        203D2059
        45415220
        20203030
        202D2039
        39
'0377   0D0A       +0037          DEFW     CRL
'0379   03         +0038          DEFB     ETX
                   +0039 ;
'037A   0D0A0D0A   +0040 MSG3     DEFW     CRL,CRL
'037E   454E5445   +0041          DEFM     'ENTER A NUMBER FOR THE DAY OF THE WEEK.'
        52204120
        4E554D42
        45522046
        4F522054
        48452044
        4159204F
        46205448
        45205745
        454B2E
'03A5   0D0A       +0042          DEFW     CRL
'03A7   0931203D   +0043          DEFM     '       1 = MONDAY, 2 = TUESDAY, 3 = WEDNESDAY, ETC.'
        204D4F4E
        4441592C
        2032203D
```

```
ADDR   OBJECT     ST # SOURCE STMT
       20545545
       53444159
       2C203320
       3D205745
       444E4553
       4441592C
       20455443
       2E
'03D4  0D0A       +0044          DEFW    CRL
'03D6  03         +0045          DEFB    ETX
                  +0046 ;
'03D7  0D0A0D0A   +0047 MSG4     DEFW    CRL,CRL
'03D8  54574F20   +0048          DEFM    'TWO CLOCK MODES ARE AVAILABLE:'
       434C4F43
       4B204D4F
       44455320
       41524520
       41564149
       4C41424C
       453A
'03F9  0D0A       +0049          DEFW    CRL
'03FB  0931202D   +0050          DEFM    '        1 - 24 HOUR'
       20323420
       484F5552
'0407  0D0A       +0051          DEFW    CRL
'0409  0932202D   +0052          DEFM    '        2 - 12 HOUR WITH AM/PM INDICATOR'
       20313220
       484F5552
       20574954
       4820414D
       2F504D20
       494E4449
       4341544F
       52
'042A  0D0A0D0A   +0053          DEFW    CRL,CRL
'042E  454E5445   +0054          DEFM    'ENTER A 1 OR 2 FOR THE DESIRED MODE.'
       52204120
       31204F52
       20322046
       4F522054
       48452044
       45534952
       4544204D
       4F44452E
'0452  0D0A       +0055          DEFW    CRL
'0454  03         +0056          DEFB    ETX
                  +0057 ;
'0455  0D0A0D0A   +0058 MSG5     DEFW    CRL,CRL
'0459  49532054   +0059          DEFM    'IS TIME TO BE ENTERED, AM OR PM ?'
       494D4520
       544F2042
       4520454E
       54455245
       442C2041
       4D204F52
       20504D20
       3F
'047A  0D0A       +0060          DEFW    CRL
```

V

```
 ADDR  OBJECT     ST # SOURCE STMT    DATASET = DK0:CLOCK .RAM[1]  21-AUG-81

'047C  03        +0061          DEFB    ETX
                 +0062 ;
'047D  0D0A0D0A  +0063 MS66     DEFW    CRL,CRL
'0481  454E5445  +0064          DEFM    'ENTER THE TIME IN THE FOLLOWING FORMAT:'
       52205448
       45205449
       4D452049
       4E205448
       4520464F
       4C4C4F57
       494E4720
       464F524D
       41543A
'04A8  0D0A      +0065          DEFW    CRL
'04AA  0948483A  +0066          DEFM    '        HH:MM:SS        WHERE:'
       4D4D3A53
       53202020
       20202020
       20574845
       52453A
'04C1  0D0A0D0A  +0067          DEFW    CRL,CRL
'04C5  09094848  +0068          DEFM    '                HH = HOUR OF THE DAY'
       203D2048
       4F555220
       4F462054
       48452044
       4159
'04DB  0D0A      +0069          DEFW    CRL
'04DD  09094D4D  +0070          DEFM    '                MM = MINUTES'
       203D204D
       494E5554
       4553
'04EB  0D0A      +0071          DEFW    CRL
'04ED  09095353  +0072          DEFM    '                SS = SECONDS'
       203D2053
       45434F4E
       4453
'04FB  0D0A      +0073          DEFW    CRL
'04FD  03        +0074          DEFB    ETX
                 +0075 ;
                 +0076 ;
'04FE  4D4F4E44  +0077 WEEKDA   DEFM    'MONDAY    '
       41592020
       2020
'0508  54554553  +0078          DEFM    'TUESDAY   '
       44415920
       2020
'0512  5745444E  +0079          DEFM    'WEDNESDAY '
       45534441
       5920
'051C  54485552  +0080          DEFM    'THURSDAY  '
       53444159
       2020
'0526  46524944  +0081          DEFM    'FRIDAY    '
       41592020
       2020
'0530  53415455  +0082          DEFM    'SATURDAY  '
       52444159
```

```
                                                  MOSTEK FLP-80 ASSEMBLER V2.1 PAGE 0011
 ADDR  OBJECT     ST # SOURCE STMT    DATASET = DK0:CLOCK .RAM[1]  21-AUG-81

       2020
'053A  53554E44 +0083          DEFM    'SUNDAY    '
       41592020
       2020
                +0084 ;
'0544  20204A41 +0085 MONTHL   DEFM    '  JANUARY '
       4E554152
       5920
'054E  20464542 +0086          DEFM    ' FEBRUARY '
       52554152
       5920
'0558  20202020 +0087          DEFM    '    MARCH '
       4D415243
       4820
'0562  20202020 +0088          DEFM    '    APRIL '
       41505249
       4C20
'056C  20202020 +0089          DEFM    '      MAY '
       20204D41
       5920
'0576  20202020 +0090          DEFM    '     JUNE '
       204A554E
       4520
'0580  20202020 +0091          DEFM    '     JULY '
       204A554C
       5920
'058A  20202041 +0092          DEFM    '   AUGUST '
       55475553
       5420
'0594  53455054 +0093          DEFM    'SEPTEMBER '
       454D4245
       5220
'059E  20204F43 +0094          DEFM    '  OCTOBER '
       544F4245
       5220
'05A8  204E4F56 +0095          DEFM    ' NOVEMBER '
       454D4245
       5220
'05B2  20444543 +0096          DEFM    ' DECEMBER '
       454D4245
       5220
                +0097 ;
'05BC  0D0A0D0A +0098 DATMSG   DEFW    CRL,CRL,CRL
       0D0A
'05C2  20205448 +0099          DEFM    '  THE DATE AND TIME ARE:'
       45204441
       54452041
       4E442054
       494D4520
       4152453A
'05DA  0D0A0D0A +0100          DEFW    CRL,CRL,CRL
       0D0A
'05E0  20202020 +0101          DEFM    '                        '
       20202020
       20202020
       20202020
       20202020
       20202020
```

V

```
 ADDR   OBJECT     ST # SOURCE STMT
        20
'>05F9            +0102 DAYWK    DEFS     10
'>0603            +0103 MONTH    DEFS     10
'060D   20202C20  +0104 DAY      DEFM     '  , 19'
        3139
'0613   2020      +0105 YEAR     DEFM     '  '
'0615   0D0A      +0106          DEFW     CRL
'0617   20202020  +0107          DEFM     '                                    '
        20202020
        20202020
        20202020
        20202020
        20202020
        20202020
        20202020
        2020
'0639   20203A    +0108 HOUR     DEFM     '  :'
'063C   20203A    +0109 MIN      DEFM     '  :'
'063F   20202020  +0110 SEC      DEFM     '    '
'0643   2020      +0111 AMPM     DEFM     '  '
'0645   0D0A      +0112          DEFW     CRL
'0647   03        +0113          DEFB     ETX
                  +0114 ;
                  +0115 ;
'0648   0D0A0D0A  +0116 HLTMSG   DEFW     CRL,CRL,CRL
        0D0A
'064E   07        +0117          DEFB     07H                        ;ASCII BELL
'064F   2A2A2A20  +0118          DEFM     '*** WARNING CLOCK IS IN HALT MODE ***'
        5741524E
        494E4720
        434C4F43
        4B204953
        20494E20
        48414C54
        204D4F44
        45202A2A
        2A
'0674   0D0A0D0A  +0119          DEFW     CRL,CRL
'0678   03        +0120          DEFB     ETX
                  +0121 ;
                  +0122 ;
                   0356 ;
                   0357          END

ERRORS=0000
```

```
ADDR  OBJECT     ST # SOURCE STMT
                 0001 ;
                 0002 ;                STI.SRC
                 0003 ;                  BY
                 0004 ;               JOHN KOVAR
                 0005 ;
                 0006 ;        OCTOBER 5, 1981          12:30  AM
                 0007 ;
>008E            0008 WCKCNL   EQU     8EH      ;WRITE TO CLOCK CONTROL REGISTER
>000E            0009 DISWP    EQU     0EH      ;DISABLE WRITE PROTECT DATABYTE
                 0010                           ;1.0 MHZ XTAL FREQ.
>00BE            0011 WCKBST   EQU     0BEH     ;ENABLE WRITE CLOCK BURST MODE
>008F            0012 RCKBST   EQU     0BFH     ;ENABLE READ CLOCK BURST MODE
>008F            0013 RCKCNL   EQU     8FH      ;READ FROM CLOCK CONTROL REGISTER
                 0014 ;
                 0015 ;
                 0016 ;        INDIRECT STI REGISTERS
                 0017 ;
>0000            0018 SCR      EQU     00H              ;SYNC CHARACTER REGISTER
>0002            0019 TCDR     EQU     02H              ;TIMER C DATA REGISTER
>0006            0020 DDR      EQU     06H              ;DATA DIRECTION REGISTER
>0007            0021 TCDCR    EQU     07H              ;TIMER C CONTROL REGISTER
                 0022 ;
                 0023 ;
                 0024 ;        STI REGISTERS
                 0025 ;
                 0026 ;
>0040            0027 IDR      EQU     40H              ;INDIRECT DATA REGISTER
>0041            0028 GPIP     EQU     41H              ;GENERAL PURPOSE DATA REGISTER
>0048            0029 POINT    EQU     48H              ;INDIRECT REG POINTER
>004C            0030 UCR      EQU     4CH              ;USART CONTROL REGISTER
>004D            0031 RSR      EQU     4DH              ;RECEIVER STATUS REGISTER
>004E            0032 TSR      EQU     4EH              ;TRANSMIT STATUS REGISTER
>004F            0033 UDR      EQU     4FH              ;USART DATA REGISTER
                 0034 ;
                 0035 ;
'0000  3E02      0036 START    LD      A,TCDR
'0002  D348      0037          OUT     (POINT),A        ;SET INDIRECT ADDRESS
'0004  3E03      0038          LD      A,03H
'0006  D340      0039          OUT     (IDR),A          ;OUTPUT TIMER COUNT
'0008  3E07      0040          LD      A,TCDCR
'000A  D348      0041          OUT     (POINT),A        ;SET INDIRECT ADDRESS
'000C  3E10      0042          LD      A,10H            ; /4 PRESCALE, START TIMER C
'000E  D340      0043          OUT     (IDR),A
'0010  3E06      0044          LD      A,DDR
'0012  D348      0045          OUT     (POINT),A        ;SET DATA DIRECTION REGISTER
'0014  3E80      0046          LD      A,80H
'0016  D340      0047          OUT     (IDR),A          ;SET GPI7 AS AN OUTPUT
'0018  3E00      0048          LD      A,00H
'001A  D341      0049          OUT     (GPIP),A         ;SET GPI7 LOW
'001C  3E02      0050          LD      A,02H
'001E  D34E      0051          OUT     (TSR),A          ;SET (SO) LOW
'0020  3E00      0052          LD      A,SCR            ;SET INDIRECT POINTER TO
'0022  D348      0053          OUT     (POINT),A        ;   SYNC CHARACTER REGISTER
'0024  211901'   0054          LD      HL,INBUF         ;SET HL TO INPUT BUFFER
                 0055 ;
'0027  CD8500'   0056          CALL    CHPENA
'002A  3E8F      0057          LD      A,RCKCNL         ;READ CLOCK CONTROL REGISTER
'002C  CDB600'   0058          CALL    ADCMDR           ;SEND AN ADDRESS/COMMAND READ
```

V

```
ADDR  OBJECT     ST # SOURCE STMT    CALL    RDBYTE          ;GET A BYTE FROM 3805
'002F CDC800'    0059                CALL    CHPDIS
'0032 CD9200'    0060
                 0061 ;
'0035 7E         0062                LD      A,(HL)
'0036 CB7F       0063                BIT     7,A             ;IS WRITE PROTECT BIT SET?
'0038 C46400'    0064                CALL    NZ,PWRINT       ;IF YES DO POWER ON INITIALIZATION
                 0065 ;
'003B 1E01       0066                LD      E,1             ;SET LUN = CONSOLE
'003D CDFFFF     0067                CALL    CRLF
'0040 CD3E00'    0068                CALL    CRLF
'0043 21B102'    0069                LD      HL,MSG1
'0046 CDFFFF     0070                CALL    PTXT            ;PRINT MENU MESSAGE
'0049 CD4100'    0071                CALL    CRLF
'004C CDEC00'    0072                CALL    GETLN           ;GET FUNCTION REQUESTED NUMBER
'004F 7E         0073                LD      A,(HL)          ;LOAD C WITH FUNCTION NUMBER
'0050 D630       0074                SUB     30H
'0052 4F         0075                LD      C,A
'0053 0600       0076                LD      B,00
'0055 CB21       0077                SLA     C
'0057 DD214101'  0078                LD      IX,FUNPT
'005B DD09       0079                ADD     IX,BC
'005D DD6E00     0080                LD      L,(IX+0)
'0060 DD6601     0081                LD      H,(IX+1)
'0063 E9         0082                JP      (HL)
                 0083 ;
                 0084 ;
'0064 CD8500'    0085 PWRINT         CALL    CHPENA          ;ENABLE 3805
'0067 3E8E       0086                LD      A,WCKCNL        ;SEND ADDRESS/COMMAND TO
'0069 CDAF00'    0087                CALL    ADCMDW          ;  WRITE CLOCK CONTROL REGISTER
'006C 3E0E       0088                LD      A,DISWP         ;SEND WRITE PROTECT DISABLE TO
'006E CDD200'    0089                CALL    SNDBYT          ;   CLOCK CONTROL REGISTER
'0071 CD9200'    0090                CALL    CHPDIS
                 0091 ;
'0074 CD8500'    0092                CALL    CHPENA
'0077 3E8E       0093                LD      A,WCKCNL        ;SEND ADDRESS/COMMAND TO
'0079 CDAF00'    0094                CALL    ADCMDW          ;  WRITE CLOCK CONTROL REGISTER
'007C 3E0E       0095                LD      A,DISWP         ;SET 3805 XTAL FREQ TO
'007E CDD200'    0096                CALL    SNDBYT          ;   1.0 MHZ
'0081 CD9200'    0097                CALL    CHPDIS
'0084 C9         0098                RET
                 0099 ;
                 0100 ;
                 0101 ;              SUBROUTINE CHIP ENABLE (CHPENA)
                 0102 ;
                 0103 ;              ON ENTRY: Serial Output (SO) should be low.
                 0104 ;
                 0105 ;              ON EXIT:  External J-K Flip Flop is enabled.
                 0106 ;                        Receive Buffer and Status Register is cleared.
                 0107 ;
'0085 3E80       0108 CHPENA         LD      A,80H           ;ENABLE 3805 CE FLIP-FLOP
'0087 D341       0109                OUT     (GPIP),A        ;   TO LOOK FOR 1ST "ONE" BIT
'0089 DB4F       0110 CLRBUF         IN      A,(UDR)         ;CLEAR RECEIVER BUFFER
'008B DB4D       0111                IN      A,(RSR)         ;WAS THERE A PREVIOUS OVERRUN?
'008D CB77       0112                BIT     6,A
'008F 20F8       0113                JR      NZ,CLRBUF       ;JUMP IF YES, CLEAR IT
'0091 C9         0114                RET
                 0115 ;
                 0116 ;
```

```
ADDR  OBJECT   ST # SOURCE STMT
                0117 ;
                0118 ;        SUBROUTINE CHIP DISABLE (CHPDIS)
                0119 ;
                0120 ;        ON EXIT:  (SO) is low and 3805 CE is high
                0121 ;
'0092  3E00     0122 CHPDIS   LD      A,00H           ;
'0094  D34D     0123          OUT     (RSR),A         ;DISABLE RECEIVER
'0096  DB4E     0124 WAIT3    IN      A,(TSR)         ;GET XMIT BUFFER EMPTY BIT
'0098  CB7F     0125          BIT     7,A             ;IS BUFFER EMPTY?
'009A  28FA     0126          JR      Z,WAIT3         ;JUMP IF NOT AND WAIT
'009C  3E00     0127          LD      A,00H           ;
'009E  D34E     0128          OUT     (TSR),A         ;DISABLE XMITTER, SET (SO) TRISTATE
'00A0  DB4E     0129 WAIT4    IN      A,(TSR)         ;GET "END" BIT
'00A2  CB67     0130          BIT     4,A             ;HAS XMITTER STOPPED SENDING?
'00A4  28FA     0131          JR      Z,WAIT4         ;JUMP IF NOT AND WAIT
'00A6  3E00     0132          LD      A,00H
'00A8  D341     0133          OUT     (GPIP),A        ;SET 3805 CE HIGH
'00AA  3E02     0134          LD      A,02H           ;SET (SO) LOW
'00AC  D34E     0135          OUT     (TSR),A
'00AE  C9       0136          RET
                0137 ;
                0138 ;
                0139 ;        SUBROUTINE ADDRESS/COMMAND FOR WRITE (ADCMDW)
                0140 ;
                0141 ;        ON ENTRY:  "A" contains Address/Command Byte for a write
                0142 ;                        to the 3805.
                0143 ;
                0144 ;        ON EXIT:   Transmitter is enabled, Receiver is not enabled.
                0145 ;
'00AF  D340     0146 ADCMDW   OUT     (IDR),A         ;OUTPUT BYTE TO SYNC REGISTER
'00B1  3E03     0147          LD      A,03H
'00B3  D34E     0148          OUT     (TSR),A         ;ENABLE XMITTER, (SO) LOW
'00B5  C9       0149          RET
                0150 ;
                0151 ;
                0152 ;        SUBROUTINE ADDRESS/COMMAND FOR A READ (ADCMDR)
                0153 ;
                0154 ;        ON ENTRY:  "A" contains Address/Command Byte for a Read
                0155 ;                        from the 3805.
                0156 ;
                0157 ;        ON EXIT:   Transmitter is disabled with (SO) set to
                0158 ;                        tristate when transmitter stops sending.
                0159 ;
                0160 ;
'00B6  D340     0161 ADCMDR   OUT     (IDR),A         ;OUTPUT BYTE TO SYNC REGISTER
'00B8  3E03     0162          LD      A,03H
'00BA  D34D     0163          OUT     (RSR),A         ;ENABLE RECEIVER, STRIP SYNC, SEARCH
'00BC  D34E     0164          OUT     (TSR),A         ;ENABLE XMITTER, (SO) LOW
'00BE  060B     0165          LD      B,0BH           ;DELAY COUNT
'00C0  05       0166 WAIT1    DEC     B               ;DELAY TO ENSURE TRANSMITTER HAS
'00C1  20FD     0167          JR      NZ,WAIT1        ;    BEGUN TRANSMITTING
'00C3  3E00     0168          LD      A,00H           ;DISABLE XMITTER, SET (SO) TO
'00C5  D34E     0169          OUT     (TSR),A         ;    TRISTATE WHEN XMITTER STOPS.
'00C7  C9       0170          RET
                0171 ;
                0172 ;
                0173 ;
                0174 ;        SUBROUTINE READ BYTE (RDBYTE)
```

V

```
                    0175 ;
                    0176 ;        ON EXIT:  Location at HL contains received data byte.
                    0177 ;
'00C8  DB4D         0178 RDBYTE   IN      A,(RSR)          ;GET RECEIVE BUFFER FULL BIT
'00CA  CB7F         0179          BIT     7,A              ;IS IT SET?
'00CC  28FA         0180          JR      Z,RDBYTE         ;JUMP IF NOT
'00CE  DB4F         0181          IN      A,(UDR)          ;GET DATA BYTE
'00D0  77           0182          LD      (HL),A           ;STORE IT
'00D1  C9           0183          RET
                    0184 ;
                    0185 ;
                    0186 ;
                    0187 ;
                    0188 ;        SUBROUTINE SEND BYTE (SNDBYT)
                    0189 ;
                    0190 ;        ON ENTRY:  "A" contains data byte to be sent to 3805.
                    0191 ;
'00D2  0E4E         0192 SNDBYT   LD      C,TSR
'00D4  ED40         0193 WAIT2    IN      B,(C)            ;GET XMIT BUFFER EMPTY BIT
'00D6  CB78         0194          BIT     7,B              ;IS IT SET?
'00D8  28FA         0195          JR      Z,WAIT2          ;JUMP IF NOT
'00DA  D34F         0196          OUT     (UDR),A          ;WRITE DATA BYTE TO USART
'00DC  C9           0197          RET
                    0198 ;
                    0199 ;
                    0200 ;  SUBROUTINE ASCBIN
                    0201 ;        CONVERTS TWO CONSECUTIVE BYTES FROM ASCII TO ONE BINARY
                    0202 ;                BYTE.
                    0203 ;
'00DD  7E           0204 ASCBIN   LD      A,(HL)           ;GET FIRST ASCII BYTE
'00DE  CB27         0205          SLA     A
'00E0  CB27         0206          SLA     A                ;SHIFT LEFT 4 LOCATIONS
'00E2  CB27         0207          SLA     A
'00E4  CB27         0208          SLA     A
'00E6  23           0209          INC     HL               ;POINT AT THE 2ND BYTE
'00E7  ED67         0210          RRD                      ;A CONTAINS BINARY BYTE
'00E9  23           0211          INC     HL
'00EA  23           0212          INC     HL
'00EB  C9           0213          RET
                    0214 ;
                    0215 ;
                    0216 ;
'00EC  3E09         0217 GETLN    LD      A,9
'00EE  211901'      0218          LD      HL,INBUF
'00F1  163F         0219          LD      D,'?'
'00F3  CDFFFF       0220          CALL    JTASK
'00F6  C9           0221          RET
                    0222 ;
                    0223 ;
'00F7  00000000     0224 CKDATI   DEFB    0,0,0,0,0,0,0,0
       00000000
                    0225 ;
'00FF  00000000     0226 CKDATO   DEFB    0,0,0,0,0,0,0,0
       00000000
                    0227 ;
                    0228 ;  SUBROUTINE BINASC
                    0229 ;        ON ENTRY:   A CONTAINS BINARY NUMBER TO BE CONVERTED TO ASCII
                    0230 ;        ON EXIT:    A AND C CONTAIN ASCII EQUIVALENT
```

```
 ADDR  OBJECT     ST # SOURCE STMT
                  0231 ;
'0107  47         0232 BINASC  LD      B,A              ;SAVE BINARY NUMBER
'0108  E60F       0233         AND     0FH              ;
'010A  F630       0234         OR      30H              ;CONVERT TO ASCII NUMBER
'010C  4F         0235         LD      C,A              ;SAVE IN REG C
'010D  78         0236         LD      A,B              ;RECOVER BINARY NUMBER
'010E  CB3F       0237         SRL     A
'0110  CB3F       0238         SRL     A
'0112  CB3F       0239         SRL     A
'0114  CB3F       0240         SRL     A
'0116  F630       0241         OR      30H              ;CONVERT TO ASCII NUMBER
'0118  C9         0242         RET
                  0243 ;
                  0244 ;
'>0119            0245 INBUF   DEFS    40
                  0246 ;
'0141  4701'      0247 FUNPT   DEFW    CKLOAD
'0143  EB01'      0248         DEFW    CKREAD
'0145  AC02'      0249         DEFW    MONITR
                  0250 ;
                  0251 ;
'0147  0608       0252 CKLOAD  LD      B,8
'0149  3E00       0253         LD      A,0
'014B  21F700'    0254         LD      HL,CKDATI
'014E  77         0255 CRLOOP  LD      (HL),A
'014F  23         0256         INC     HL
'0150  05         0257         DEC     B
'0151  20FB       0258         JR      NZ,CRLOOP
'0153  212503'    0259         LD      HL,MSG2
'0156  CD4700'    0260         CALL    PTXT
'0159  CDEC00'    0261         CALL    GETLN            ;GET DATE FROM TERMINAL
'015C  DD21F700'  0262         LD      IX,CKDATI        ;INITIALIZE IX TO CLOCK IN BUFFER
'0160  CDDD00'    0263         CALL    ASCBIN           ;GO CONVERT MONTH TO BINARY BCD
'0163  DD7704     0264         LD      (IX+4),A         ;STORE BCD MONTH
'0166  CDDD00'    0265         CALL    ASCBIN           ;GO CONVERT DAY
'0169  DD7703     0266         LD      (IX+3),A         ;STORE BCD DAY
'016C  CDDD00'    0267         CALL    ASCBIN           ;GO CONVERT YEAR
'016F  DD7706     0268         LD      (IX+6),A         ;STORE BCD YEAR
'0172  21B003'    0269         LD      HL,MSG3
'0175  CD5701'    0270         CALL    PTXT
'0178  CDEC00'    0271         CALL    GETLN            ;GET DAY OF WEEK FROM TERMINAL
'017B  7E         0272         LD      A,(HL)
'017C  D630       0273         SUB     30H              ;CONVERT FROM ASCII TO BINARY
'017E  DD7705     0274         LD      (IX+5),A         ;STORE BCD DAY OF THE WEEK
'0181  DDCB02BE   0275         RES     7,(IX+2)         ;RESET 12-24 MODE TO 24 HOUR MODE
'0185  21DD04'    0276         LD      HL,MSG4
'0188  CD7601'    0277         CALL    PTXT
'018B  CDEC00'    0278         CALL    GETLN            ;GET CLOCK MODE
'018E  7E         0279         LD      A,(HL)
'018F  FE31       0280         CP      31H              ;WAS 24 HOUR MODE SELECTED?
'0191  281A       0281         JR      Z,CLK24          ;JUMP IF YES
'0193  DDCB02FE   0282         SET     7,(IX+2)         ;SET 12 HOUR MODE BIT
'0197  21BB04'    0283         LD      HL,MSG5
'019A  CD8901'    0284         CALL    PTXT
'019D  CDEC00'    0285         CALL    GETLN            ;GET AM OR PM RESPONSE
'01A0  DDCB02EE   0286         SET     5,(IX+2)         ;SET PM INDICATOR
'01A4  7E         0287         LD      A,(HL)
'01A5  FE41       0288         CP      41H              ;WAS 1ST CHARACTER AN A?
```

V

```
ADDR   OBJECT     ST # SOURCE STMT    
'01A7  2004       0289          JR      NZ,CLK24          ;JUMP IF NOT
'01A9  DDCB02AE   0290          RES     5,(IX+2)          ;RESET AM/PM INDICATOR TO AM
'01AD  21B304'    0291 CLK24    LD      HL,MSG6
'01B0  CD9801'    0292          CALL    PTXT
'01B3  CDEC00'    0293          CALL    GETLN             ;GET TIME TO BE SET
'01B6  CDDD00'    0294          CALL    ASCBIN            ;CONVERT TO BCD
'01B9  DDB602     0295          OR      (IX+2)            ;INCLUDE MODE BITS
'01BC  DD7702     0296          LD      (IX+2),A          ;STORE IN CLOCK BUFFER
'01BF  CDDD00'    0297          CALL    ASCBIN            ;GO CONVERT MINUTES
'01C2  DD7701     0298          LD      (IX+1),A          ;STORE BCD MINUTES
'01C5  CDDD00'    0299          CALL    ASCBIN
'01C8  DD7700     0300          LD      (IX+0),A          ;STORE BCD SECONDS
'01CB  DD36070E   0301          LD      (IX+7),0EH        ;SET CLOCK CRYSTAL FREQ SELECT
                  0302 ;
'01CF  CD8500'    0303          CALL    CHPENA
'01D2  3EBE       0304          LD      A,WCKBST          ;SEND ADDRESS/COMMAND WRITE
'01D4  CDAF00'    0305          CALL    ADCMDW            ;   CLOCK BURST MODE
'01D7  21F700'    0306          LD      HL,CKDATI         ;POINT AT CLOCK DATA
'01DA  0608       0307          LD      B,8               ;LOAD CLOCK BURST COUNTER
'01DC  7E         0308 NXTWRD   LD      A,(HL)            ;GET CLOCK DATA BYTE
'01DD  C5         0309          PUSH    BC
'01DE  CDD200'    0310          CALL    SNDBYT            ;WRITE BYTE TO CLOCK REGISTERS
'01E1  23         0311          INC     HL
'01E2  C1         0312          POP     BC                ;RECOVER BURST COUNT
'01E3  10F7       0313          DJNZ    NXTWRD            ;HAS ALL REGISTERS BEEN WRITTEN TO?
'01E5  CD9200'    0314          CALL    CHPDIS            ;DISABLE 3805
'01E8  C30000'    0315          JP      START             ;JUMP TO START AND PRINT MENU
                  0316 ;
                  0317 ;
'01EB  CD8500'    0318 CKREAD   CALL    CHPENA            ;ENABLE CHIP 3805
'01EE  3EBF       0319          LD      A,RCKBST          ;SEND ADDRESS/COMMANC READ CLOCK
'01F0  CDB600'    0320          CALL    ADCMDR            ;  CONTROL
'01F3  21FF00'    0321          LD      HL,CKDATO         ;SET HL TO POINT AT LOCATION
'01F6  0608       0322          LD      B,8               ;SET BURST BYTE COUNT
'01F8  C5         0323 NXWRD    PUSH    BC                ;SAVE BURST BYTE COUNT
'01F9  CDC800'    0324          CALL    RDBYTE            ;READ A BYTE FROM 3805
'01FC  23         0325          INC     HL
'01FD  C1         0326          POP     BC                ;RECOVER BURST BYTE COUNT
'01FE  10F8       0327          DJNZ    NXWRD             ;HAVE 8 BYTES BEEN READ?
'0200  CD9200'    0328          CALL    CHPDIS            ;DISABLE CHIP
'0203  DD212F06'  0329          LD      IX,DAYWK          ;LOAD IX WITH START OF MESSAGE LOC
'0207  21FF00'    0330          LD      HL,CKDATO         ;LOAD HL WITH CLOCK OUTPUT DATA
'020A  7E         0331          LD      A,(HL)            ;GET 1ST BYTE OF DATA
'020B  CB7F       0332          BIT     7,A               ;IS CLOCK IN HALTED MODE?
'020D  C49D02'    0333          CALL    NZ,CLKHLT         ;CALL HALT WARNING ROUTINE
'0210  CD0701'    0334          CALL    BINASC            ;CONVERT BINARY SECONDS TO ASCII
'0213  DD7746     0335          LD      (IX+70),A         ;STORE TENS OF SECONDS
'0216  DD7147     0336          LD      (IX+71),C         ;STORE UNITS OF SECONDS
'0219  23         0337          INC     HL
'021A  7E         0338          LD      A,(HL)            ;GET BINARY MINUTES
'021B  CD0701'    0339          CALL    BINASC            ;GO CONVERT TO ASCII
'021E  DD7743     0340          LD      (IX+67),A         ;STORE TENS OF MINUTES
'0221  DD7144     0341          LD      (IX+68),C         ;STORE UNITS OF MINUTES
'0224  23         0342          INC     HL
'0225  7E         0343          LD      A,(HL)            ;GET HOURS
'0226  012020     0344          LD      BC,'  '           ;LOAD BC WITH ASCII BLANKS
'0229  CB7F       0345          BIT     7,A               ;IS 12 HOUR MODE BIT SET?
'022B  280C       0346          JR      Z,M24             ;JUMP IF NOT
```

```
ADDR   OBJECT     ST # SOURCE STMT    LD      C,'M'
'022D  0E4D       0347                LD      B,'A'           ;LOAD B WITH AN ASCII "A"
'022F  0641       0348                BIT     5,A             ;IS PM BIT SET?
'0231  CB6F       0349                JR      Z,M241          ;JUMP IF NOT
'0233  2802       0350                LD      B,'P'           ;LOAD B WITH AN ASCII "P"
'0235  0650       0351                AND     5FH             ;RESET BITS 5 AND 7
'0237  E65F       0352 M241           LD      (IX+74),B       ;STORE ASCII AM OR PM
'0239  DD704A     0353 M24            LD      (IX+75),C
'023C  DD714B     0354                CALL    BINASC          ;CONVERT HOURS TO ASCII
'023F  CD0701'    0355                LD      (IX+64),A       ;STORE ASCII HOURS
'0242  DD7740     0356                LD      (IX+65),C
'0245  DD7141     0357                INC     HL
'0248  23         0358                LD      A,(HL)          ;GET DAY OF MONTH
'0249  7E         0359                CALL    BINASC
'024A  CD0701'    0360                LD      (IX+20),A       ;STORE ASCII DAY
'024D  DD7714     0361                LD      (IX+21),C
'0250  DD7115     0362                INC     HL
'0253  23         0363                LD      A,(HL)          ;GET MONTH IN BCD
'0254  7E         0364                CP      10H             ;IS MONTH LESS THAN DECIMAL 10?
'0255  FE10       0365                JP      M,NOCONV        ;JUMP IF YES, NO BCD CONVERSION
'0257  FA5E02'    0366                ADD     A,10            ;CONVERT BCD TO BINARY
'025A  C60A       0367                RES     4,A             ;RESET BCD TENS DIGIT
'025C  CBA7       0368 NOCONV         LD      B,A
'025E  47         0369                PUSH    HL              ;SAVE HL
'025F  E5         0370                LD      HL,MCNTHL-10
'0260  217005'    0371                LD      DE,10           ;
'0263  110A00     0372 LOOP1          ADD     HL,DE           ;INCREMENT POINTER BY 10
'0266  19         0373                DJNZ    LOOP1           ;CONTINUE UNTIL CORRECT MONTH FOUND
'0267  10FD       0374                LD      BC,10           ;SET BYTE TRANSFER COUNT
'0269  010A00     0375                LD      DE,MONTH        ;DATA TO BE TRANSFERRED TO MONTH
'026C  113906'    0376                LDIR                    ;TRANSFER ASCII MONTH
'026F  EDB0       0377                POP     HL              ;RECOVER POINTER
'0271  E1         0378                INC     HL
'0272  23         0379                LD      B,(HL)          ;GET DAY OF WEEK
'0273  46         0380                PUSH    HL
'0274  E5         0381                LD      HL,WEEKDA-10
'0275  212A05'    0382                LD      DE,10
'0278  110A00     0383 LOOP2          ADD     HL,DE           ;INCREMENT POINTER BY 10
'027B  19         0384                DJNZ    LOOP2           ;CONTINUE UNTIL CORRECT DAY FOUND
'027C  10FD       0385                LD      BC,10
'027E  010A00     0386                LD      DE,DAYWK
'0281  112F06'    0387                LDIR                    ;TRANSFER ASCII DAY TO DAYWK
'0284  EDB0       0388                POP     HL
'0286  E1         0389                INC     HL
'0287  23         0390                LD      A,(HL)          ;GET YEAR
'0288  7E         0391                CALL    BINASC          ;CONVERT BINARY TO ASCII
'0289  CD0701'    0392                LD      (IX+26),A
'028C  DD771A     0393                LD      (IX+27),C       ;STORE ASCII YEAR
'028F  DD711B     0394                LD      HL,DATMSG
'0292  21F205'    0395                LD      E,1
'0295  1E01       0396                CALL    PTXT
'0297  CDB101'    0397                JP      START           ;JUMP TO START AND PRINT MENU
'029A  C30000'    0398
                  0399 ;
                  0400 ;
                  0401 ;  SUBROUTINE CLKHLT
                  0402 ;
                  0403 ;        THIS ROUTINE PRINTS A WARNING THAT THE CLOCK IS HALTED.
                  0404 ;
```

V

```
ADDR  OBJECT    ST # SOURCE STMT    PUSH    HL
'029D  E5       0405 CLKHLT  PUSH    HL
'029E  F5       0406         PUSH    AF
'029F  217E06'  0407         LD      HL,HLTMSG
'02A2  1E01     0408         LD      E,1
'02A4  CD9802'  0409         CALL    PTXT
'02A7  F1       0410         POP     AF
'02A8  E1       0411         POP     HL
'02A9  CBBF     0412         RES     7,A            ;CLEAR CLOCK HALT BIT
'02AB  C9       0413         RET
                0414 ;
                0415 ;
'02AC  3E01     0416 MONITR  LD      A,1
'02AE  C3F400'  0417         JP      JTASK          ;RETURN TO FLP-80DOS MONITOR
                0418 ;
                0419 ;
                0420 ;
                0421         INCLUDE MSG
               +0001 ;
               +0002 ;                   MSG.
               +0003 ;
               +0004 ;
               +0005 ;
               +0006 ;       OCTOBER 4, 1981        6:30 PM
```

UNITED TECHNOLOGIES MOSTEK

Mostek Corporation, 1215 West Crosby Rd.
Carrollton, Texas 75006 USA; (214) 466-6000
In Europe, Contact: Mostek Brussels
270-272 Avenue de Tervuren (BTE21)
B-1150 Brussels, Belgium; Telephone: 762.18.80

PRINTED IN USA June 1982