Pathways through the ROM

Guide to Level II BASIC And DOS Source Code

by George Blank,
Roger Fuller,
John Hartford,
John T. Phillipp,
and Robert M. Richardson

Pathways Through the ROM

Guide to Level II BASIC and DOS

Edited by George Blank

Robert M. Richardson
Roger Fuller
John T. Phillipp
George Blank
John Hartford

Cover Illustration
Elaine Cheever

SoftSide Publications
6 South Street
Milford, New Hampshire 03055

ABOUT THIS BOOK

As this book is being prepared for publication, sales of the Radio Shack **TRS-80** microcomputer are approaching 200,000 and another computer, the **Video Genie**, is being developed that will use the same software. These small but powerful general purpose computers are driven by the same BASIC interpreter, **Microsoft** BASIC, known to **TRS-80** purchasers as Level II BASIC.

Finding adequate documentation on **Microsoft** BASIC has been a continuing problem. **Microsoft** has a dominating position in the field as the author of **APPLE**, **PET**, **ALTAIR**, **KIM**, and **Heathkit** BASIC as well as the one described in this book, and they have been understandably reluctant to make it easy on a potential competitor by selling the documented source code for their product.

We respect their rights to their product, and for that reason do not supply source and object code in this book. We do provide sufficient comments and a disassembler so that you can create your own source code, but to do this you must first purchase one of these computers, so that **Microsoft** receives its due royalties.

This book is a compilation of four items previously published separately. **The TRS-80 Disassembled Handbook** was published by Richcraft Engineering Ltd. of Chautauqua, NY at $10.00 (copyright © 1980 Richcraft Engineering, Ltd). **Supermap, Level II ROM Documentation**, was published by Fuller Software of Grand Prairie, Texas for $18.95 (copyright (c) 1979 Fuller Software). **HEX-MEM**, the BASIC monitor by John T. Phillipp, was published in the February 1980 issue of Prog/80 Magazine and **The Z-80 Disassembler** by George Blank in the June, 1980 issue (copyright (c) 1980 SoftSide Publications, Milford, NH 03055 — subscription price, $15 a year for 6 issues). We are grateful to the authors for granting us permission to collect their works for the convenience of our readers.

The Forward, Introduction, and first 9 chapters of this book are from **The TRS-80 Disassembled Handbook** by Robert M. Richardson, and are presented as a series of lessons in the use of ROM subroutines. Chapter 10 is **Supermap** by Roger Fuller, which lists, in sequential order, comments on the contents of the Level II BASIC ROM indexed to hexadecimal memory locations. Chapter 11 is **HEX-MEM**, a monitor program written in BASIC to enable readers to examine the ROM directly and experiment with machine language programming. Chapter 12 is a complete Z-80 disassembler, which in conjunction with a computer, a printer and the comments in Chapter 10, will enable the reader to produce a commented source listing of the Level II BASIC ROM for personal use.

Chapter 13 contains comments on the Disk Operating Systems TRSDOS and NEWDOS by John Hartford. Chapter 14 is the specification sheet for the floppy disk controller chip used in the TRS-80, manufactured by Western Digital. Our thanks to Western Digital for permission to reprint this material.

These programs, while very useful, will not satisfy the serious programmer, who will desire more powerful programming tools. For those who do their programming on cassette based machines, we recommend the Radio Shack EDITOR ASSEMBLER and a machine language monitor like T-BUG, RSM-1, RSM-1S, RSM-2, or STAD. Of the monitors, we recommend STAD (for symbolic trace and debug) which has more features, including multiple breakpoints, creation of symbol tables, and the ability to handle both tape and disk I/O, at an excellent price. Programmers with disk systems have a greater choice, with a fine monitor, DEBUG, included in the TRS-DOS operating system. This lacks a disassembler, tape I/O ability, and the trace routines of STAD, so you may still find the other program useful. For assembly language programming, Apparat provides a disk version of the Radio Shack EDITOR ASSEMBLER as part of their NEWDOS+ package (you must still purchase the original from Radio Shack to get the documentation), or you can purchase Microsoft's powerful MACRO ASSEMBLER. T-BUG, TRS-DOS and the EDITOR ASSEMBLER are available from Radio Shack. STAD, NEWDOS+, RSM-2, and the Microsoft M-80 MACRO ASSEMBLER may be purchased from **The Software Exchange**, P.O. Box 68, Milford, NH (toll free telephone order line (800) 258-1790.)

We should not neglect to give credit where credit is due: TRS-80 and TRS-DOS are trademarks of Radio Shack, Division of Tandy Corporation. Z-80 is a trademark of Zilog Corporation, and the Z-80 assembly language mnemonics are copyrighted by the same company.

Pathways through the ROM

Table of Contents

Part 1

(from THE TRS-80 DISASSEMBLED HANDBOOK by Robert M. Richardson)

Part II

(from SUPERMAP by Roger Fuller)

Part III

(from PROG/80 Magazine, February and June 1980)

Part IV

Introduction to the TRS-80 Disassembled Handbook

This handbook started out as a collection of lectures prepared as "fill-routines" by the author while on the tour circuit promoting his new book, "The Gunnplexer Cookbook — A 10 GHz Microwave Primer." It is a rather long jump (at least in frequency and wavelength) from the 1.77 MHz clock of the TRS-80 to the 10250 MHz band where the Gunnplexer operates, but surprisingly the microwave buff and computer buff have more in common than meets the eye; i.e., a need to communicate whether it be by simple fm voice modulation or ASCII digital data link.

This handbook is NOT for the beginning assembly language programmer who should certainly learn the fundamentals of the art before attempting to use the shortcuts and "tight code" programming made possible by using the myriad excellent Level II ROM subroutines presented. The moderately experienced assembly language programmer who understands the difference between his/her JPs and JRs, SETs and RESets, and BITs and bytes will find that utilizing Level II ROM subroutines opens up an entirely new vista to the wonderful world of assembly language programming. Many dull but demanding rote subroutines may now be accomplished often with a single CALL compared with previous multi-line/multipage programming that had to be entered a line at a time. Programmers arise. Cast off the yoke of ignorance and START HAVING FUN writing assembly language programs that run 300+ times faster than BASIC in less than 1/10th the memory. With practice, you (and the author) will soon be writing assembly language programs as quickly and with as little effort as those written in BASIC, FORTRAN, COBOL, or PASCAL.

Disassembled machine code of any variety WITHOUT comments is worth about as much as a TRS-80 without electric power. Constants, address table entry points, and data lists are translated into utter meaningless/misleading garbage by the disassembler program. Disassembled Level II ROM prints out EX AF,AF' from memory locations 005AH, 1479H, 1619H, 18BAH, 18BCH, etc., etc., when in actuality none of the alternate register pairs are ever used in this ROM's BASIC program.

Decoding and making any sense out of any object code program is impossible if one does not at least have a clue to the program's intended function. Fortunately we know exactly what Level II ROM's functions and capabilities are, so what appeared to be an impossible decoding problem is reduced to only an extremely difficult one. So difficult in fact, that even Radio Shack's computer division in Fort Worth, Texas does not fully understand this excellent "Tight Code" written by Microsoft's Paul Allen and Bill Gates. An example that proves this point is the Radio Shack book, "TRS-80 Assembly Language Programming," #62-2006, that was introduced midyear 1979 has virtually no references to Level II ROM subroutines of any variety.

The level of difficulty in decoding the TRS-80 ROM may be measured by the fact that after 2 years of worldwide use by over 200,000 computer buffs ranging from beginners to advanced programmers with years of experience, the Level II ROM entry points for virtually ALL the BASIC functions and related subroutines have never been published with comments, or disclosed by anyone, anywhere, EXCEPT to a very limited audience by a genius named Andrew Hildebrand, located in the southwestern wilderness of the US.

This handbook is dedicated to Mr. Hildebrand's genius, to his persistence in unraveling a very tangled web, and to him personally for this very considerable accomplishment. The author's only contribution is this handbook which will hopefully remove the shroud of mystery from a previous "black hole" by attempting to make the subject understandable to hobbyists, high school and college students, and even computer science professors (many of whom do not have the slightest idea of how to efficiently utilize the myriad Level II ROM subroutines in TRS-80 assembly language programming). This handbook will also assist users in decoding the majority of ROM subroutines in ALL Microsoft BASIC's including those used by APPLE, PET, KIM, Heathkit, et al microcomputer systems.

Each section of this handbook is hopefully self-explanatory. The self-test questions and answers in chapters 8 and 9 allow you to check your progress after each Chapter.

Acknowledgements:

Without the generous aid and assistance of both Nancy A. Courtney and Margaret C. Merz this handbook would have never come to pass. Their encouragement, hard work, and persistent insistence that we keep at it are gratefully appreciated.

Additional Thank Yous:

To the late Charles Tandy for his courage and investment in making the TRS-80 a happening; to Mumford Micro for a 3-speed clock; to Western I/O for an IBM Selectric printer that never quits; to Apparat for NEWDOS+, to Shrayer for Electric Pencil; to Microsoft for the world's most extensively used BASIC; and lastly to Radio Shack, who in spite of other shortcomings, build the WORLD'S MOST COST EFFECTIVE MICRO-COMPUTER. Thank you each and every one.

Robert M. Richardson March 1980
Chautauqua Lake, New York 14722

Summary

Chapter 1 illustrates how the majority of the ROM functions' CALL locations are decoded.

Chapter 2 includes three source/object code programs and text illustrating the use of simple ROM +-*/ integer, single precision, and double precision arithmetic subroutines.

Chapter 3 describes how to use the trigonometric, exponent, log, CINT, CSGN, CDBL, et al, functions in concert with the RAM accumulator and "CS" store, plus a demonstration program.

Chapter 4 covers a number of the more useful and important Level II ROM ancillary subroutines.

Chapter 5 is a summary of virtually all BASIC function CALL addresses.

Chapter 6 presents and explains an extremely useful number conversion program that is virtually a must for the serious assembly language programmer working with the TRS-80. It is written in BASIC for comprehension/modification and covers:

— DECIMAL TO BINARY ENTER D
— BINARY TO DECIMAL ENTER B
— HEXADECIMAL TO BINARY ENTER HB
— DECIMAL TO HEXADECIMAL ENTER DH
— HEXADECIMAL TO DECIMAL ENTER HD
— SPLIT TRS-80 TO DECIMAL ENTER SP
— DECIMAL TO SPLIT HEXADECIMAL ENTER DS
— SPLIT HEXADECIMAL TO DECIMAL ENTER SD

The first five conversions are obvious ones. The sixth, SPLIT TRS-80 TO DECIMAL, is unique in that it takes decimal values displayed via the TRS-80 PEEK command from two adjacent memory locations (two INPUTS required), then converts them to hex, reverses the two hex numbers as the Z-80 stores LSB first and MSB second, then converts the four digit hex number to decimal, and displays it on video. This conversion is a real time saver when extracting addresses (0 to 65535) from ROM or RAM.

Chapter 7 is a useful print all zeroes with a slash program.

Chapter 8 includes self-test questions for Chapters 1 through 7.

Chapter 9 is a bibliography and answers to the self-test.

Chapter 10 is a listing of comments on the Level II ROM, in sequential order and indexed to the hexadecimal memory address.

Chapter 11 is a monitor program provided here for convenience in making HEX and ASCII memory dumps of the Level II ROM.

Chapter 12 is a Z-80 disassembler with the ability to assign labels to individual memory locations on printout. It is provided to make it possible for readers to assign labels based on the comments in Chapter 10 and print out commented source code from their own computer.

Chapter 13 is a listing of comments on TRSDOS and NEWDOS, in sequential order and indexed to the hexadecimal memory address.

Chapter 14 is the specification sheet for the WD1771 Controller Chip.

Chapter 1

Decoding Level II ROM Function Call Locations

Introduction:

During the past two years, the Level II BASIC written by Microsoft originally for the TRS-80 has become the standard de facto BASIC used by virtually every significant microcomputer manufacturer. As of January 1980, the number of microcomputers delivered to end users with Level II or modest variations of Level II BASIC is estimated to exceed 300,000. With the number of BASICs in the marketplace counted in the dozens, including Hewlett-Packard BASIC, General Electric BASIC, and even monolithic IBM's "VS BASIC," there must be a number of good reasons for the near universal adoption of Microsoft's BASIC.

1. Is it cheap? Answer: no, in actuality the license to use this BASIC is incredibly expensive. Heathkit does not charge hobbyists $100 a copy (just for the program) for fun.

2. Is it efficient? Answer: you bet it is. Previous BASICs the author has studied required 22K to 32K memory to minimally perform the same functions, if indeed as many.

3. Is it cost-effective? Answer: even at the high licensing price it is VERY cost-effective when one considers the tradeoffs between available program memory remaining with the inherent 64K maximum imposed by most all 8 bit microprocessors of the current generation. Some of the other BASICs mentioned above only leave the user about 18K of useful RAM when a disk operating system and extended disk BASIC program is added. This is ridiculous in a 64K MEM system.

Let's now take a look at the prodigious functions and their CALL locations in this marvel of "tight code" programming written by Paul Allen and Bill Gates. We doubt if there was any intentional encrypting involved when the program was written as encryption takes memory and memory costs money either directly or indirectly as pointed out above. The main reason it has been difficult to decode the Level II ROM was brought about primarily by its compactness; i.e., nothing wasted, nothing unused, and no easily deciphered points telling the code breaker, "here I am, use me."

Level II ROM Function Name Locations:

These are the easiest to find of all. Look at memory locations 5712 through 6172. Figure 1 is an excruciatingly simple BASIC program that will display these names and their location on video for you. The first letter of each name's MSB is masked by subtracting 128 to obtain its ASCII equivalent. Figure 2 is a print out of this program. Remember, the numbers are the name's location, NOT the CALL location.

Figure 1

```
10 ' MICROSOFT BASIC FUNCTIONS NAME LIST
 IN LEVEL II ROM
15 '
20 CLS:FORN=5712TO6175:Y=PEEK(N):IFY>127
THENY=Y-128:M=N
25 Z=N+1:IFPEEK(Z)>127THENPRINTCHR$(Y);"
 =";ELSEGOTO35
30 PRINTM,:GOTO40
35 PRINTCHR$(Y);
40 NEXT
```

Figure 2

END	= 5712	FOR	=5715	RESET	= 5718	SET	= 5723
CLS	= 5726	CMD	= 5729	RANDOM	= 5732	NEXT	= 5738
DATA	= 5742	INPUT	= 5746	DIM	= 5751	READ	= 5754
LET	= 5758	GOTO	= 5761	RUN	= 5765	IF	= 5768
RESTORE	= 5770	GOSUB	= 5777	RETURN	= 5782	REM	= 5788
STOP	= 5791	ELSE	= 5795	TRON	= 5799	TROFF	= 5803
DEFSTR	= 5808	DEFINT	= 5814	DEFSNG	= 5820	DEFDBL	= 5826
LINE	= 5832	EDIT	= 5836	ERROR	= 5840	RESUME	= 5845
OUT	= 5851	ON	= 5854	OPEN	= 5856	FIELD	= 5860
GET	= 5865	PUT	= 5868	CLOSE	= 5871	LOAD	= 5876
MERGE	= 5880	NAME	= 5885	KILL	= 5889	LSET	= 5893
RSET	= 5897	SAVE	= 5901	SYSTEM	= 5905	LPRINT	= 5911
DEF	= 5917	POKE	= 5920	PRINT	= 5924	CONT	= 5929
LIST	= 5933	LLIST	= 5937	DELETE	= 5942	AUTO	= 5948
CLEAR	= 5952	CLOAD	= 5957	CSAVE	= 5962	NEW	= 5967
TAB(	= 5970	TO	= 5974	FN	= 5976	USING	= 5978
VARPTR	= 5983	USR	= 5989	ERL	= 5992	ERR	= 5995
STRING$	= 5998	INSTR	= 6005	POINT	= 6010	TIME$	= 6015
MEM	= 6020	INKEY$	= 6023	THEN	= 6029	NOT	= 6033
STEP	= 6036	+	= 6040	-	= 6041	*	= 6042
/	= 6043	↑	= 6044	AND	= 6045	OR	= 6048
>	= 6050	=	= 6051	<	= 6052	SGN	= 6053
INT	= 6056	ABS	= 6059	FRE	= 6062	INP	= 6065
POS	= 6068	SQR	= 6071	RND	= 6074	LOG	= 6077
EXP	= 6080	COS	= 6083	SIN	= 6086	TAN	= 6089
ATN	= 6092	PEEK	= 6095	CVI	= 6099	CVS	= 6102
CVD	= 6105	EOF	= 6108	LOC	= 6111	LOF	= 6114
MKI$	= 6117	MKS$	= 6121	MKD$	= 6125	CINT	= 6129
CSNG	= 6133	CDBL	= 6137	FIX	= 6141	LEN	= 6144
STR$	= 6147	VAL	= 6151	ASC	= 6154	CHR$	= 6157
LEFT$	= 6161	RIGHT$	= 6166	MID$	= 6172		

Matching BASIC Functions with ROM CALL Addresses:

Now the decoding game becomes somewhat more interesting. Not difficult yet, because no tricky encipherment was used. We should remember that encoding costs memory and memory = money. One does not have to search very far in memory for the location of each BASIC function's CALL address.

These addresses are split into two groups. The first group begins a few bytes after the end of the function name list at MEM location 6178 and runs through 6451. This group covers all functions from END through 'less than'. The second group begins at MEM location 5640 and runs through 5711. This group includes all BASIC functions from SGN through the end of the function list, MID$.

With the exception of those BASIC functions from TAB to 'less than', all CALL locations are stored in MEM using the standard Zilog Z-80 format (the genius of Federico Faggin, Z-80 creator appears again), with the LSB (least significant byte) first, and the MSB (most significant byte) second, in the following memory location. Figure 3 illustrates a little BASIC program that will display the BASIC function, an = sign, then the MEM location of the stored CALL address for this specific function and lastly the CALL address in standard TRS-80 "PEEK" format.

Figure 3

```
10 ' PROGRAM TO MATCH BASIC FUNCTIONS
 WITH CALL ADDRESSES
15 '
20 CLS:PRINT"FUNCT=ADDRESS     LSB-MSB
     FUNCT=ADDRESS     LSB-MSB
25 A=6176:FORX=5712TO6175:Y=PEEK(X):IFY>
127THENY=Y-128
30 Z=X+1:IFPEEK(Z)>127THENPRINTCHR$(Y);"
 =";ELSEGOTO45
35 A=A+2:IFA=6352THENA=5640
40 PRINTA,PEEK(A);"-";PEEK(A+1), :GOTO50
45 PRINTCHR$(Y);
50 NEXT
```

Figure 4 is a printout of those addresses whose LSB/MSB are directly translatable to CALL addresses. The other functions' CALL addresses are fully covered in Chapter 5. Figure 5 is a printout of Level II ROM MEM locations 1600H through 18FFH to illustrate how Level II ROM BASIC function CALL locations are determined.

Figure 4
Function=CALL Address MEM Location CALL-Address

FUNCT	= ADDRESS	LSB-MSB	FUNCT	= ADDRESS	LSB-MSB
END	= ~~6718~~ 6178	174 -29	FOR	= 6180	161 -28
RESET	= 6182	56 - 1	SET	= 6184	53 - 1
CLS	= 6186	201 - 1	CMD	= 6188	115 -65
RANDOM	= 6190	211 - 1	NEXT	= 6192	182 -34
DATA	= 6194	5 -31	INPUT	= 6196	154 -33
DIM	= 6198	8 -38	READ	= 6200	239 -33
LET	= 6202	33 -31	GOTO	= 6204	194 -30
RUN	= 6206	163 -30	IF	= 6208	57 -32
RESTORE	= 6210	145 -29	GOSUB	= 6212	177 -30
RETURN	= 6214	222 -30	REM	= 6216	7 -31
STOP	= 6218	169 -29	ELSE	= 6220	7 -31
TRON	= 6222	247 -29	TROFF	= 6224	248 -29
DEFSTR	= 6226	0 -30	DEFINT	= 6228	3 -30
DEFSNG	= 6230	6 -30	DEFDBL	= 6232	9 -30
LINE	= 6234	163 -65	EDIT	= 6236	96 -46
ERROR	= 6238	244 -31	RESUME	= 6240	175 -31
OUT	= 6242	251 -42	ON	= 6244	108 -31
OPEN	= 6246	121 -65	FIELD	= 6248	124 -65
GET	= 6250	127 -65	PUT	= 6252	130 -65
CLOSE	= 6254	133 -65	LOAD	= 6256	136 -65
MERGE	= 6258	139 -65	NAME	= 6260	142 -65
KILL	= 6262	145 -65	LSET	= 6264	151 -65
RSET	= 6266	154 -65	SAVE	= 6268	160 -65
SYSTEM	= 6270	178 - 2	LPRINT	= 6272	103 -32
DEF	= 6274	91 -65	POKE	= 6276	177 -44
PRINT	= 6278	111 -32	CONT	= 6280	228 -29
LIST	= 6282	46 -43	LLIST	= 6284	41 -43
DELETE	= 6286	198 -43	AUTO	= 6288	8 -32
CLEAR	= 6290	122 -30	CLOAD	= 6292	31 -44
CSAVE	= 6294	245 -43	NEW	= 6296	73 -27
INT	= 5642	55 -11	ABS	= 5655	119 - 9
FRE	= 5646	212 -39	INP	= 5648	239 -42
POS	= 5650	245 -39	SQR	= 5652	231 -19
RND	= 5654	201 -20	LOG	= 5656	9 - 8
EXP	= 5658	57 -20	COS	= 5660	65 -21
SIN	= 5662	71 -21	TAN	= 5664	168 -21
ATN	= 5666	189 -21	PEEK	= 5668	170 -44
CVI	= 5670	82 -65	CVS	= 5672	88 -65
CVD	= 5674	94 -65	EOF	= 5676	97 -65
LOC	= 5678	100 -65	LOF	= 5680	103 -65
MKI$	= 5682	106 -65	MKS$	= 5684	109 -65
MKD$	= 5686	112 -65	CINT	= 5688	127 -10
CSNG	= 5690	177 -10	CDBL	= 5692	219 -10
FIX	= 5694	38 -11	LEN	= 5696	3 -42
STR$	= 5698	54 -40	VAL	= 5700	197 -42
ASC	= 5702	15 -42	CHR$	= 5704	31 -42
LEFT$	= 5706	97 -42	RIGHT$	= 5708	145 -42
MID$	= 5710	154 -42			

Figure 5

Converting BASIC Function CALL Addresses to Hex and Decimal:

```
1600    6C AA AA 7F 00 00 00 81 8A 09 37 0B 77 09 D4 27
1610    EF 2A F5 27 E7 13 C9 14 09 08 39 14 41 15 47 15
1620    A8 15 BD 15 AA 2C 52 41 58 41 5E 41 61 41 64 41
1630    67 41 6A 41 6D 41 70 41 7F 0A B1 0A DB 0A 26 0B
1640    03 2A 36 28 C5 2A 0F 2A 1F 2A 61 2A 91 2A 9A 2A
1650    C5 4E 44 C6 4F 52 D2 45 53 45 54 D3 45 54 C3 4C
1660    53 C3 4D 44 D2 41 4E 44 4F 4D CE 45 58 54 C4 41
1670    54 41 C9 4E 50 55 54 C4 49 4D D2 45 41 44 CC 45
1680    54 C7 4F 54 4F D2 55 4E C9 46 D2 45 53 54 4F 52
1690    45 C7 4F 53 55 42 D2 45 54 55 52 4E D2 45 4D D3
16A0    54 4F 50 C5 4C 53 45 D4 52 4F 4E D4 52 4F 46 46
16B0    C4 45 46 53 54 52 C4 45 46 49 4E 54 C4 45 46 53
16C0    4E 47 C4 45 46 44 42 4C CC 49 4E 45 C5 44 49 54
16D0    C5 52 52 4F 52 D2 45 53 55 4D 45 CF 55 54 CF 4E
16E0    CF 50 45 4E C6 49 45 4C 44 C7 45 54 D0 55 54 C3
16F0    4C 4F 53 45 CC 4F 41 44 CD 45 52 47 45 CE 41 4D
1700    45 CB 49 4C 4C CC 53 45 54 D2 53 45 54 D3 41 56
1710    45 D3 59 53 54 45 4D CC 50 52 49 4E 54 C4 45 46
1720    D0 4F 4B 45 D0 52 49 4E 54 C3 4F 4E 54 CC 49 53
1730    54 CC 4C 49 53 54 C4 45 4C 45 54 45 C1 55 54 4F
1740    C3 4C 45 41 52 C3 4C 4F 41 44 C3 53 41 56 45 CE
1750    45 57 D4 41 42 28 D4 4F C6 4E D5 53 49 4E 47 D6
1760    41 52 50 54 52 D5 53 52 C5 52 4C C5 52 52 D3 54
1770    52 49 4E 47 24 C9 4E 53 54 52 D0 4F 49 4E 54 D4
1780    49 4D 45 24 CD 45 4D C9 4E 4B 45 59 24 D4 48 45
1790    4E CE 4F 54 D3 54 45 50 AB AD AA AF DB C1 4E 44
17A0    CF 52 BE BD BC D3 47 4E C9 4E 54 C1 42 53 C6 52
17B0    45 C9 4E 50 D0 4F 53 D3 51 52 D2 4E 44 CC 4F 47
17C0    C5 58 50 C3 4F 53 D3 49 4E D4 41 4E C1 54 4E D0
17D0    45 45 4B C3 56 49 C3 56 53 C3 56 44 C5 4F 46 CC
17E0    4F 43 CC 4F 46 CD 4B 49 24 CD 4B 53 24 CD 4B 44
17F0    24 C3 49 4E 54 C3 53 4E 47 C3 44 42 4C C6 49 58
1800    CC 45 4E D3 54 52 24 D6 41 4C C1 53 43 C3 48 52
1810    24 CC 45 46 54 24 D2 49 47 48 54 24 CD 49 44 24
1820    A7 80 AE 1D A1 1C 38 01 35 01 C9 01 73 41 D3 01
1830    B6 22 05 1F 9A 21 08 26 EF 21 21 1F C2 1E A3 1E
1840    39 20 91 1D B1 1E DE 1E 07 1F A9 1D 07 1F F7 1D
1850    F8 1D 00 1E 03 1E 06 1E 09 1E A3 41 60 2E F4 1F
1860    AF 1F FB 2A 6C 1F 79 41 7C 41 7F 41 82 41 85 41
1870    88 41 8B 41 8E 41 91 41 97 41 9A 41 A0 41 B2 02
1880    67 20 5B 41 B1 2C 6F 20 E4 1D 2E 2B 29 2B C6 2B
1890    08 20 7A 1E 1F 2C F5 2B 49 1B 79 79 7C 7C 7F 50
18A0    46 DB 0A 00 00 7F 0A F4 0A B1 0A 77 0C 70 0C A1
18B0    0D E5 0D 78 0A 16 07 13 07 47 08 A2 08 0C 0A D2
18C0    0B C7 0B F2 0B 90 24 39 0A 4E 46 53 4E 52 47 4F
18D0    44 46 43 4F 56 4F 4D 55 4C 42 53 44 44 2F 30 49
18E0    44 54 4D 4F 53 4C 53 53 54 43 4E 4E 52 52 57 55
18F0    45 4D 4F 46 44 4C 33 D6 00 6F 7C DE 00 67 78 DE
```

```
1600    . . . . . . . . . . 7 . . . . '
1610    . * . ' . . . . . . 9 . A . G .
1620    . . . . . , R A X A ^ A . A . A
1630    . A . A . A . A . . . . . . & .
1640    . * 6 ( . * . * . * . * . * . *
1650    . N D . O R . E S E T . E T . L
1660    S . M D . A N D O M . E X T . A
1670    T A . N P U T . I M . E A D . E
1680    T . O T O . U N . F . E S T O R
1690    E . O S U B . E T U R N . E M .
16A0    T O P . L S E . R O N . R O F F
16B0    . E F S T R . E F I N T . E F S
16C0    N G . E F D B L . I N E . D I T
16D0    . R R O R . E S U M E . U T . N
16E0    . P E N . I E L D . E T . U T .
16F0    L O S E . O A D . E R G E . A M
1700    E . I L L . S E T . S E T . A V
1710    E . Y S T E M . P R I N T . E F
1720    . O K E . R I N T . O N T . I S
1730    T . L I S T . E L E T E . U T O
1740    . L E A R . L O A D . S A V E .
1750    E W . A B ( . O . N . S I N G .
1760    A R P T R . S R . R L . R R . T
1770    R I N G $ . N S T R . O I N T .
1780    I M E $ . E M . N K E Y $ . H E
1790    N . O T . T E P . . . . . . N D
17A0    . R . . . . G N . N T . B S . R
17B0    E . N P . O S . Q R . N D . O G
17C0    . X P . O S . I N . A N . T N .
17D0    E E K . V I . V S . V D . O F .
17E0    O C . O F . K I $ . K S $ . K D
17F0    $ . I N T . S N G . D B L . I X
1800    . E N . T R $ . A L . S C . H R
1810    $ . E F T $ . I G H T $ . I D $
1820    . . . . . . 8 . 5 . . . . A . .
1830    . " . . . ! . & . ! ! . . . . .
1840    9   . . . . . . . . . . . . . .
1850    . . . . . . . . . . . A . . . .
1860    . . . * . . . A . A . A . A . A
1870    . A . A . A . A . A . A . A . .
1880    .   [ A . , .   . . . + ) + . +
1890    .   . . . , . + I . . . . . . P
18A0    F . . . . . . . . . . . . . . .
18B0    . . . . . . . . . G . . . . . .
18C0    . . . . . . $ 9 . N F S N R G O
18D0    D F C O V O M U L B S D D / 0 I
18E0    D T M O S L S S T C N N R R W U
18F0    E M O F D L 3 . . . . . . . . .
```

Converting the BASIC function CALL addresses printed out in Figure 4 to hexadecimal and decimal is certainly simple, but nevertheless a tedious job whether done with a calculator or using Disk BASIC's &H function. There is a considerably easier way to do it.

The Multi-Base Number Conversion Program presented in Chapter 6 makes the conversion of the CALL locations to hexadecimal and decimal a real pleasure instead of a rote chore. This program may be easily modified to take the LSB and MSB from Figure 3's program output and automatically generate the CALL's decimal and hexadecimal location, but for simplicity's sake let's run through Chapter 6's standard number conversion routine for generating the CALL address for the first function shown in Figure 4, which is END.

Load the conversion program. We will use the program's SPLIT DECIMAL TO DECIMAL routine so press 'SP' ENTER. Figure 4 shows that END's CALL location is LSB = 174 and MSB = 29. The program will display "DECIMAL?". Press 174 then ENTER. After a moment, the 174 will disappear and "DECIMAL?" will again be displayed on video. Now, press 29 then ENTER. Faster than a speeding bullet, HEXADECIMAL 1DAE will be displayed on video followed by DECIMAL 7598 a second or two later. Aha, it really does work.

The conversion program's logic and flow is as follows:

1. Convert '174' to hexadecimal and store it.
2. Convert '29' to hexadecimal and store it.
3. Reverse the two hexadecimal numbers so it will read MSB/LSB.
4. Display the hexadecimal number on video.
5. Convert the hexadecimal number to decimal.
6. Display the decimal number on video.

Figure 6 is an exact printout of Figure 4's BASIC function CALL addresses in both hexadecimal and decimal. Those CALL addresses above 12288 are of course for coupling to DOS/Disk. So far, so good. Now let's JP to Chapter 2 and get some experience using Level II ROM's subroutines with integer, single precision, and double precision arithmetic. We will initially only be doing 3rd grade add, subtract, multiply, and divide, but we will be doing it with only a few CALLs instead of lines/pages of assembly language programming.

Figure 6

FUNCTION	HEX	DECIMAL	FUNCTION	HEX	DECIMAL
END	1DAE	7598	FOR	1CA1	7329
RESET	0138	312	SET	0135	309
CLS	01C9	457	CMD	4173	16755
RANDOM	01D3	467	NEXT	22B6	8886
DATA	1F05	7941	INPUT	219A	8602
DIM	2608	9736	READ	21EF	8687
LET	1F21	7969	GOTO	1EC2	7874
RUN	1EA3	7843	IF	2039	8249
RESTORE	1D91	7569	GOSUB	1EB1	7857
RETURN	1EDE	7902	REM	1F07	7943
STOP	1DA9	7593	ELSE	1F07	7943
TRON	1DF7	7671	TROFF	1DF8	7672
DEFSTR	1E00	7680	DEFINT	1E03	7683
DEFSNG	1E06	7686	DEFDBL	1E09	7689
LINE	41A3	16803	EDIT	2E60	11872
ERROR	1FF4	8180	RESUME	1FAF	8111
OUT	2AFB	11003	ON	1F6C	8044
OPEN	4179	16761	FIELD	417C	16764
GET	417F	16767	PUT	4182	16770
CLOSE	4185	16773	LOAD	4188	16776
MERGE	418B	16779	NAME	418E	16782
KILL	4191	16785	LSET	4197	16791
RSET	419A	16794	SAVE	41A0	16800
SYSTEM	02B2	690	LPRINT	2067	8295
DEF	415B	16731	POKE	2CB1	11441
PRINT	206F	8303	CONT	1DE4	7652
LIST	2B2E	11054	LLIST	2B29	11049
DELETE	2BC6	11206	AUTO	2008	8200
CLEAR	1E7A	7802	CLOAD	2C1F	11295
CSAVE	2BF5	11253	NEW	1B49	6985
INT	0B37	2871	ABS	0977	2423
FRE	27D4	10196	INP	2AEF	10991
POS	27F5	10229	SQR	13E7	5095
RND	14C9	5321	LOG	0809	2057
EXP	1439	5177	COS	1541	5441
SIN	1547	5447	TAN	15A8	5544
ATN	15BD	5565	PEEK	2CAA	11434
CVI	4152	16722	CVS	4158	16728
CVD	415E	16734	EOF	4161	16737
LOC	4164	16740	LOF	4167	16743
MKI$	416A	16746	MKS$	416D	16749
MKD$	4170	16752	CINT	0A7F	2687
CSNG	0AB1	2737	CDBL	0ADB	2779
FIX	0B26	2854	LEN	2A03	10755
STR$	2836	10294	VAL	2AC5	10949
ASC	2A0F	10767	CHR$	2A1F	10783
LEFT$	2A61	10849	RIGHT$	2A91	10897
MID$	2A0A	10906			

Chapter 2

Integer, Single, and Double Precision Arithmetic

Introduction:

After one has recovered from the shock of learning the fundamentals of assembly language programming it is ridiculous to "reinvent the wheel" by writing dozens of lines or pages of source code to perform simple single and double precision arithmetic calculations when these routines already exist in Level II ROM and may be accessed with a single call.

Assembly language programming with its resulting source code programs running 300+ times faster than BASIC and requiring on the average only 1/10th as much memory to perform the same functions as BASIC is very helpful for the serious amateur programmer who wishes to advance beyond the inherent limitations of BASIC, FORTRAN, COBOL, PASCAL or any of the high level computer languages. Prior to this book, assembly language programmers were forced to learn by rote those assembly language subroutines for ALL the functions that were already extant in the Level II ROM because no one had ever figured out exactly how to access all these subroutines; i.e., break the beautifully "TIGHT CODE" code written by Microsoft's Paul Allen and Bill Gates.

Assembly language programmers arise! The Level II ROM code has now been broken. As every cryptographer knows, every lock has a key. It is just that some locks take a bit longer to pick than others, (ask N.S.A. or MI.5 about this). For some perverse reason, (probably money and the Chinese secrecy syndrome), neither Radio Shack or Microsoft have been willing to come forward and tell the 200,000+ TRS-80 users how to use the myriad Level II ROM subroutines in assembly language programs. This point is best illustrated by Radio Shack's book, "TRS-80 Assembly Language Programming," introduced in mid-1979 which leads the would-be assembly language programmer into T-Bug (surely the height of backwardness/retrogression), and then goes on with multiline demonstration programs covering keyboard scan, video display, fill, move, muladd, mulsub, compare, mul16, div16, etc. that could be accomplished with only a few lines of assembly language programming IF the extant Level II ROM subroutines had been used. If you have mastered Level II BASIC, you should have great fun with this totally NEW approach to assembly language programming. By mastering Level II BASIC you have demonstrated that you have the skills and persistence to become an advanced assembly language programmer with only a few weeks study rather than what heretofore took many months or years. The supposed "experts" in the field of assembly language programming have created an aura and mystique about the subject which is totally undeserved and seeks only to promote their own self esteem. Let us take a brief look at how very simple assembly language programming can be by illustrating our point with a few simple arithmetic programs that almost exclusively use Level II ROM subroutines.

Fundamentals of Level II ROM Arithmetic:

ROM arithmetic subroutines are virtually identical to those you would HAVE to write were they not NOW available to the assembly language programmer. This is true whether we are discussing integer, single precision, double precision, addition, subtraction, multiplication, division, as well as ALL the trigonometric, exponential, and log functions too. In fact, it is true for all Level II ROM functions which are nothing more than binary bytes we may manipulate as long as we know where they are located. Let's get on with the primer for + - * and /.

ROM (read-only-memory) means just what it says, READ-ONLY. Since it may be only read from, Level II ROM uses the RAM memory from 14302 to 17129 for all its housekeeping chores. The keyboard, from 14336 to 15360 is not really RAM at all, but a simple key/switch matrix which the rest of the system thinks is RAM. Video memory occupies memory locations 15360 to 16383. Except for memory locations 14302 to 14336, all the non-disk Level II RAM housekeeping chores are done between 16384 and 17129. Three RAM memory locations are of particular interest while discussing arithmetic + - * / subroutines. They are the ACCUMulator, CDBL store (or "CS" abbreviation), and NT (number type). Arithmetic numbers stashed in RAM use the following conventions: integer = LSB first and MSB second using two's complement format, and single and double precision numbers = normalized exponential format with 129 added to the exponent and the high bit of the MSB reflecting the + or - sign of the number. Do not concern yourself with these number formats as our Level II ROM will handle all the conversions necessary if we use them properly. The ACCUM occupies memory locations 411DH through 4124H (8 bytes) and CDBL store occupies 4127H through 412EH, also 8 bytes. We must concern ourselves with NT (number type) as it will "blow" our whole subroutine if we try to perform arithmetic operations with dissimilar number types; i.e., add an integer to a double precision number, etc. Do not fret though, ROM lets us use its CINT, CSGN, CDBL functions with only a single CALL to make the numbers we are using compatible.

CINT = CALL 0A7FH CSNG = CALL 0AB1H CDBL = CALL 0ADBH

The 3 programs in this chapter provide these functions in each routine, so it takes real effort to foul them up if you abide by each number type's minimal rules.

The NT (number type) single byte storage in RAM is located at 40AFH. NT (40AFH) = 2 for an integer number, = 3 for a string, = 4 for a single precision number, and = 8 for a double precision number. To change these numbers to ASCII and display them on video, simply ADD 30H to the contents of MEM location 40AFH and output to the video display as follows:

```
LD      A,(40AFH)   ;NT location
ADD     A,30H       ;convert to ASCII
CALL    032AH       ;display on video
```

Integer Arithmetic + - * / **:**

Figure 7 is the demonstration program that will allow you to add, subtract, multiply, or divide integers strictly using the ROM subroutines. Fast? You bet it is. As soon as you press ENTER you'll have the answer. Remember, your Model I TRS-80 with its clock running at approximately 1,774,000 cycles per second is no slowpoke. Instead of BASIC's "slowsville", you are now conversing with your Z-80 microprocessor directly, IN ITS OWN LANGUAGE. With no interpreter (BASIC) required, it will zap along at what appears to be the speed of light. All this integer program does is to place the first number you input into the DE register, the second number you input into the HL register, and then CALL whatever +-*/ operation you requested. This simple program is completely straightforward except for line 330's PUSH HL and line 400's POP DE. The stack begins at RAM memory location 4288H when operating in the SYSTEM mode. What we are doing here is "saving" the first integer number in the stack by PUSHing HL in line 330. The program then uses the HL register in obtaining the second number you input in line 340. The POP DE in line 400 merely takes the previous HL value from the stack and places it into the DE register. The stack couldn't care less where its contents go as it is just a sophisticated FILO (first-in-last-out) memory created and controlled by your Z-80/ROM (unless you choose to modify its location with the LD SP (stack pointer) opcode and operand instruction. Remember, integer arithmetic is nothing more than placing the 'F'irst number in the DE register, the 'S'econd number in the HL register, and specifying which +-*/ operation you desire with the following CALLS:

ADD CALL 0BD2H SUBTRACT = CALL 0BD7H
MULTIPLY = CALL 0BF2H DIVIDE = CALL 2490H

The result of any of these operations is always placed in the ACCUM. To display the result on video, merely:

CALL	0FBDH	;convert ACCUM to ASCII string
CALL	28A7H	;display ASCII string on video

That is all there is to it. Simplesville personified.

Single Precision Arithmetic + - * / :

This is very similar to integer arithmetic, except ROM now wants the 'F'irst number in registers BC and DE, and the 'S'econd number in the ACCUM. The desired operation is performed by:

ADD = CALL 0716H SUBTRACT = CALL 0713H
MULTIPLY = CALL 0847H DIVIDE = CALL 08A2H

For memory storage, we again use the stack as shown in lines 340 and 350 PUSH instructions, and lines 420 and 430 POP same. Figures 9 and 10 are the source code and object code for the single precision arithmetic demonstration program.

Double Precision Arithmetic + - * / :

Is not significantly different from either integer or single precision arithmetic subroutines, except now ROM wants the 'F'irst number in the ACCUM and the 'S'econd number in the CDBL store RAM location. Desired operation is performed by:

ADD = CALL 077CH SUBTRACT = CALL 0C70H
MULTIPLY = CALL 0DA1HDIVIDE = CALL 0DE5H

Figures 11 and 12 are the source and object code for the double precision arithmetic demonstration program.

Summary:

Each of these source code programs may be input by the user in about 5 minutes time using the EXCELLENT Radio Shack Editor/Assembler that was written by Zilog's President and in-house-genius, Federico Faggin, and his staff. Previous assembly language teaching programs required months of study and page upon page of source code to accomplish all the double precision arithmetic routines. Now the average TRS-80 buff can master the subject in a matter of hours.

Figure 7 Source Code   Figure 8 Object Code

```
                00100 ;        INTEGER ARITHMETIC DEMONSTRATION PROGRAM
                00110
                00120 ;        USING LEVEL II ROM SUBROUTINES  + - * /
                00130
7D00            00140 W4UCH    EQU    7D00H       ;= 32000 DECIMAL
7D00            00150          ORG    W4UCH       ;PROGRAM WILL START HERE
7D00 3E4F       00160 BEGIN    LD     A,4FH       ;"O" OPERATION DESIRED
7D02 CD2A03     00170          CALL   032AH       ;DISPLAY "O" ON VIDEO
7D05 3E3F       00180          LD     A,3FH       ;= ASCII ?
7D07 CD2A03     00190          CALL   032AH       ;DO IT - ON VIDEO
7D0A 3E20       00200          LD     A,20H       ;= ASCII SPACE
7D0C CD2A03     00210          CALL   032AH       ;DO IT - ON VIDEO
7D0F CD4900     00220          CALL   049H        ;KYBD INPUT + - * /
7D12 CD2A03     00230          CALL   032AH       ;DISPLAY FUNCTION
7D15 327B7D     00240          LD     (FUNCT),A   ;STASH DESIRED OPERATION
7D18 3E0D       00250          LD     A,0DH       ;0DH = SKIP A LINE
7D1A CD2A03     00260          CALL   032AH       ;DO IT - ON VIDEO
7D1D 3E46       00270          LD     A,46H       ;"F" = FIRST NUMBER
7D1F CD2A03     00280          CALL   032AH       ;DO IT - ON VIDEO
7D22 CDB31B     00290          CALL   1BB3H       ;KYBD/VIDEO INPUT ROUTINE
7D25 D7         00300          RST    10H         ;SCAN $ SET C FLAG
7D26 CD6C0E     00310          CALL   0E6CH       ;ASCII-ACCUM RET MIN
7D29 CD7F0A     00320          CALL   0A7FH       ;CONVERT TO INTEGER
7D2C E5         00330          PUSH   HL          ;SAVE INTEGER IN STACK
7D2D 3E53       00340          LD     A,53H       ;"S" = SECOND NUMBER
7D2F CD2A03     00350          CALL   032AH       ;DISPLAY "S" ON VIDEO
7D32 CDB31B     00360          CALL   1BB3H       ;KYBD/VIDEO INPUT ROUTINE
7D35 D7         00370          RST    10H         ;SCAN $ SET C FLAG
7D36 CD6C0E     00380          CALL   0E6CH       ;ASCII TO ACCUM RET MIN
7D39 CD7F0A     00390          CALL   0A7FH       ;CONVERT TO INTEGER
7D3C D1         00400          POP    DE          ;PREVIOUS HL TO DE REG
7D3D 3A7B7D     00410          LD     A,(FUNCT)   ;RECALL + - * / FROM MEM
7D40 FE2B       00420          CP     2BH         ;IS IT + ?
7D42 2823       00430          JR     Z,ADD       ;IF SO GOTO ADD
7D44 FE2D       00440          CP     2DH         ;IS IT - ?
```

```
7D46 2824            JR      Z,SUB          ;IF SO GOTO SUBTRACT
7D48 FE2A            CP      2AH            ;IS IT * ?
7D4A 2825            JR      Z,MULT         ;IF SO GOTO MULTIPLY
7D4C FE2F            CP      2FH            ;IS IT / ?
7D4E 2826            JR      Z,DIVIDE       ;IF SO GOTO DIVIDE
7D50 3E3D     VIDEO  LD      A,3DH          ;3DH IS ASCII = SIGN
7D52 CD2A03          CALL    032AH          ;DO IT - ON VIDEO
7D55 3E20            LD      A,20H          ;=ASCII SPACE
7D57 CD2A03          CALL    032AH          ;DO IT - ON VIDEO
7D5A CDBD0F          CALL    0FBDH          ;CONV ACCUM TO STRING
7D5D CDA728          CALL    28A7H          ;DISPLAY STRING ON VIDEO
7D60 3E0D            LD      A,0DH          ;= SKIP A LINE
7D62 CD2A03          CALL    032AH          ;DO IT - ON VIDEO
7D65 1899            JR      BEGIN          ;REPEAT ROUTINE
7D67 CDD20B   ADD    CALL    0BD2H          ;ADD DE + HL
7D6A 18E4            JR      VIDEO          ;OUTPUT RESULT
7D6C CDC70B   SUB    CALL    0BC7H          ;SUBTRACT DE - HL
7D6F 18DF            JR      VIDEO          ;OUTPUT RESULT
7D71 CDF20B   MULT   CALL    0BF2H          ;MULTIPLY DE * HL
7D74 18DA            JR      VIDEO          ;OUTPUT RESULT
7D76 CD9024   DIVIDE CALL    2490H          ;DIVIDE DE / HL
```

Figure 9 Figure 10

```
              ;      SINGLE PRECISION ARITHMETIC DEMONSTRATION PROGRAM

              ;      USING LEVEL II ROM SUBROUTINES  + - * /  BY W4UCH

7D00          W4UCH  EQU     7D00H          ;= 32000 DECIMAL
7D00                 ORG     W4UCH          ;PROGRAM WILL START HERE
7D00 3E4F     BEGIN  LD      A,4FH          ;"O" OPERATION DESIRED
7D02 CD2A03          CALL    032AH          ;DISPLAY "O" ON VIDEO
7D05 3E3F            LD      A,3FH          ;= ASCII ?
7D07 CD2A03          CALL    032AH          ;DO IT - ON VIDEO
7D0A 3E20            LD      A,20H          ;= ASCII SPACE
7D0C CD2A03          CALL    032AH          ;DO IT - ON VIDEO
7D0F CD4900          CALL    049H           ;KYBD INPUT + - * /
7D12 CD2A03          CALL    032AH          ;DISPLAY FUNCTION
7D15 32807D          LD      (FUNCT),A      ;STASH DESIRED OPERATION
7D18 3E0D            LD      A,0DH          ;0DH = SKIP A LINE
7D1A CD2A03          CALL    032AH          ;DO IT - ON VIDEO
7D1D 3E46            LD      A,46H          ;"F" = FIRST NUMBER
7D1F CD2A03          CALL    032AH          ;DO IT - ON VIDEO
7D22 CDB31B          CALL    1BB3H          ;KYBD/VIDEO INPUT ROUTINE
7D25 D7              RST     10H            ;SCAN $ SET C FLAG
7D26 CD6C0E          CALL    0E6CH          ;ASCII-ACCUM RET MIN
7D29 CDB10A          CALL    0AB1H          ;CONV SINGLE PRECISION
7D2C CDBF09          CALL    09BFH          ;LOAD BCDE FROM ACCUM
7D2F C5              PUSH    BC             ;STORE IN STACK
7D30 D5              PUSH    DE             ;STORE IN STACK
7D31 3E53            LD      A,53H          ;"S" = SECOND NUMBER
7D33 CD2A03          CALL    032AH          ;DISPLAY "S" ON VIDEO
7D36 CDB31B          CALL    1BB3H          ;KYBD/VIDEO INPUT ROUTINE
7D39 D7              RST     10H            ;SCAN $ SET C FLAG
7D3A CD6C0E          CALL    0E6CH          ;ASCII TO ACCUM RET MIN
7D3D CDB10A          CALL    0AB1H          ;CONV TO SINGLE PRECISION
7D40 D1              POP     DE             ;RESTORE DE REGISTER
7D41 C1              POP     BC             ;RESTORE BC REGISTER
7D42 3A807D          LD      A,(FUNCT)      ;RECALL + - * / FROM MEM
7D45 FE2B            CP      2BH            ;IS IT + ?
```

```
7D47 2823     00460          JR     Z,ADD        ;IF SO GOTO ADD
7D49 FE2D     00470          CP     2DH          ;IS IT - ?
7D4B 2824     00480          JR     Z,SUB        ;IF SO GOTO SUBTRACT
7D4D FE2A     00490          CP     2AH          ;IS IT * ?
7D4F 2825     00500          JR     Z,MULT       ;IF SO GOTO MULTIPLY
7D51 FE2F     00510          CP     2FH          ;IS IT / ?
7D53 2826     00520          JR     Z,DIVIDE     ;IF SO GOTO DIVIDE
7D55 3E3D     00530 VIDEO    LD     A,3DH        ;3DH IS ASCII = SIGN
7D57 CD2A03   00540          CALL   032AH        ;DO IT - ON VIDEO
7D5A 3E20     00550          LD     A,20H        ;=ASCII SPACE
7D5C CD2A03   00560          CALL   032AH        ;DO IT - ON VIDEO
7D5F CDBD0F   00570          CALL   0FBDH        ;CONV ACCUM TO STRING
7D62 CDA728   00580          CALL   28A7H        ;DISPLAY STRING ON VIDEO
7D65 3E0D     00590          LD     A,0DH        ;= SKIP A LINE
7D67 CD2A03   00600          CALL   032AH        ;DO IT - ON VIDEO
7D6A 1894     00610          JR     BEGIN        ;REPEAT ROUTINE
7D6C CD1607   00620 ADD      CALL   0716H        ;ADD BCDE REGS TO ACCUM
7D6F 18E4     00630          JR     VIDEO        ;OUTPUT RESULT
7D71 CD1307   00640 SUB      CALL   0713H        ;SUB ACCUM FM BCDE REGS
7D74 18DF     00650          JR     VIDEO        ;OUTPUT RESULT
7D76 CD4708   00660 MULT     CALL   0847H        ;MULT ACCUM * BCDE REGS
7D79 18DA     00670          JR     VIDEO        ;OUTPUT RESULT
7D7B CDA208   00680 DIVIDE   CALL   08A2H        ;DIV ACCCUM INTO BCDE
7D7E 18D5     00690          JR     VIDEO        ;OUTPUT RESULT
7D80 00       00700 FUNCT    DEFB   0            ;SAVE BYTE-STASH FUNCTION
7D00          00710          END    W4UCH        ;EL FIN = EL PRIMERO
00000 TOTAL ERRORS
ADD     7D6C 00620   00460
BEGIN   7D00 00160   00610
DIVIDE  7D7B 00680   00520
FUNCT   7D80 00700   00240 00440
MULT    7D76 00660   00500
SUB     7D71 00640   00480
VIDEO   7D55 00530   00630 00650 00670 00690
W4UCH   7D00 00140   00150 00710
```

Figure 11 Figure 12

```
              00100 ;        DOUBLE PRECISION DEMONSTRATION PROGRAM
              00110
              00120 ;        USING LEVEL II ROM SUBROUTINES   + - * /
              00130
7D00          00140 W4UCH    EQU    7D00H        ;= 32000 DECIMAL
7D00          00150          ORG    W4UCH        ;PROGRAM WILL START HERE
7D00 3E4F     00160 BEGIN    LD     A,4FH        ;"O" OPERATION DESIRED
7D02 CD2A03   00170          CALL   032AH        ;DISPLAY "O" ON VIDEO
7D05 3E3F     00180          LD     A,3FH        ;= ASCII ?
7D07 CD2A03   00190          CALL   032AH        ;DO IT - ON VIDEO
7D0A 3E20     00200          LD     A,20H        ;= ASCII SPACE
7D0C CD2A03   00210          CALL   032AH        ;DO IT - ON VIDEO
7D0F CD4900   00220          CALL   049H         ;KYBD INPUT + - * /
7D12 CD2A03   00230          CALL   032AH        ;DISPLAY FUNCTION
7D15 328C7D   00240          LD     (FUNCT),A    ;STASH DESIRED OPERATION
7D18 3E0D     00250          LD     A,0DH        ;0DH = SKIP A LINE
7D1A CD2A03   00260          CALL   032AH        ;DO IT - ON VIDEO
7D1D 3E46     00270          LD     A,46H        ;"F" = FIRST NUMBER
7D1F CD2A03   00280          CALL   032AH        ;DO IT - ON VIDEO
7D22 CDB31B   00290          CALL   1BB3H        ;KYBD/VIDEO INPUT ROUTINE
7D25 D7       00300          RST    10H          ;SCAN $ SET C FLAG
7D26 CD650E   00310          CALL   0E65H        ;ASCII$ TO ACCUM RET CDBL
```

```
7D29 111D41    00320          LD     DE,411DH      ;MOVE FROM ACCUM RAM MEM
7D2C 218D7D    00330          LD     HL,TACCUM     ;TO TEMPORARY ACCUM STASH
7D2F 0608      00340          LD     B,8           ;NUMBER OF BYTES TO MOVE
7D31 CDD709    00350          CALL   09D7H         ;MOVE IT - SUBROUTINE
7D34 3E53      00360          LD     A,53H         ;"S" = SECOND NUMBER
7D36 CD2A03    00370          CALL   032AH         ;DISPLAY "S" ON VIDEO
7D39 CDB31B    00380          CALL   1BB3H         ;KYBD/VIDEO INPUT ROUTINE
7D3C D7        00390          RST    10H           ;SCAN $ SET C FLAG
7D3D CD650E    00400          CALL   0E65H         ;ASCII$ TO ACCUM RET CDBL
7D40 CDFC09    00410          CALL   09FCH         ;TRANSFER ACCUM TO CDBL
7D43 118D7D    00420          LD     DE,TACCUM     ;MOVE ACCUM FROM STASH TO
7D46 211D41    00430          LD     HL,411DH      ;PERMANENT RAM LOCATION
7D49 0608      00440          LD     B,8           ;NUMBER OF BYTES TO MOVE
7D4B CDD709    00450          CALL   09D7H         ;MOVE IT - RIGHT NOW
7D4E 3A8C7D    00460          LD     A,(FUNCT)     ;RECALL + - * / FROM MEM
7D51 FE2B      00470          CP     2BH           ;IS IT + ?
7D53 2823      00480          JR     Z,ADD         ;IF SO GOTO ADD
7D55 FE2D      00490          CP     2DH           ;IS IT - ?
7D57 2824      00500          JR     Z,SUB         ;IF SO GOTO SUBTRACT
7D59 FE2A      00510          CP     2AH           ;IS IT * ?
7D5B 2825      00520          JR     Z,MULT        ;IF SO GOTO MULTIPLY
7D5D FE2F      00530          CP     2FH           ;IS IT / ?
7D5F 2826      00540          JR     Z,DIVIDE      ;IF SO GOTO DIVIDE
7D61 3E3D      00550 VIDEO    LD     A,3DH         ;3DH IS ASCII = SIGN
7D63 CD2A03    00560          CALL   032AH         ;DO IT - ON VIDEO
7D66 3E20      00570          LD     A,20H         ;=ASCII SPACE
7D68 CD2A03    00580          CALL   032AH         ;DO IT - ON VIDEO
7D6B CDBD0F    00590          CALL   0FBDH         ;CONV ACCUM TO STRING
7D6E CDA728    00600          CALL   28A7H         ;DISPLAY STRING ON VIDEO
7D71 3E0D      00610          LD     A,0DH         ;= SKIP A LINE
7D73 CD2A03    00620          CALL   032AH         ;DO IT - ON VIDEO
7D76 1888      00630          JR     BEGIN         ;REPEAT ROUTINE
7D78 CD770C    00640 ADD      CALL   0C77H         ;ADD ACCUM TO CDBL
7D7B 18E4      00650          JR     VIDEO         ;OUTPUT RESULT
7D7D CD700C    00660 SUB      CALL   0C70H         ;SUB CDBL FROM ACCUM
7D80 18DF      00670          JR     VIDEO         ;OUTPUT RESULT
7D82 CDA10D    00680 MULT     CALL   0DA1H         ;MULT ACCUM * CDBL
7D85 18DA      00690          JR     VIDEO         ;OUTPUT RESULT
7D87 CDE50D    00700 DIVIDE   CALL   0DE5H         ;DIV ACCUM BY CDBL
7D8A 18D5      00710          JR     VIDEO         ;OUTPUT RESULT
7D8C 00        00720 FUNCT    DEFB   0             ;SAVE BYTE-STASH FUNCTION
0008           00730 TACCUM   DEFS   8             ;TEMPORARY ACCUM STASH
7D00           00740          END    W4UCH         ;AMATEUR RADIO CALL LTRS
00000 TOTAL ERRORS
ADD     7D78 00640   00480
BEGIN   7D00 00160   00630
DIVIDE  7D87 00700   00540
FUNCT   7D8C 00720   00240 00460
MULT    7D82 00680   00520
SUB     7D7D 00660   00500
TACCUM  7D8D 00730   00330 00420
VIDEO   7D61 00550   00650 00670 00690 00710
W4UCH   7D00 00140   00150 00740
7D79 18D5      00660          JR     VIDEO         ;OUTPUT RESULT
7D7B 00        00670 FUNCT    DEFB   0             ;SAVE BYTE-STASH FUNCTION
7D00           00680          END    W4UCH         ;AMATEUR RADIO CALL LTRS
00000 TOTAL ERRORS
ADD     7D67 00590   00430
BEGIN   7D00 00160   00580
DIVIDE  7D76 00650   00490
FUNCT   7D7B 00670   00240 00410
MULT    7D71 00630   00470
SUB     7D6C 00610   00450
VIDEO   7D50 00500   00600 00620 00640 00660
W4UCH   7D00 00140   00150 00680
```

Chapter 3

Using Level II ROM Subroutines in Advanced Assembly Language Programming Trigonometric, Log, Exponent, and Other Functions

(notes from a lecture)

Here is an interesting test program for the advanced assembly language programmer. It allows the user to access and test many of the myriad arithmetic/trigonometric subroutines that are extant in the excellent TRS-80 Level II ROM that was written by Microsoft's Bill Gates and Paul Allen.

The beginning assembly language programmer should certainly be taught and learn the how, why, and wherefores of writing fundamental arithmetic/trig functions by him/her self, but once these techniques have been mastered as part of the learning process, it is certainly inefficient, time wasting, and rather ridiculous to reinvent the wheel by duplicating in assembly language those subroutines already extant in the Level II ROM.

Table 1 lists those functions and their addresses that may be accessed and tested by this mini-program that only occupies 144 bytes of high memory and may be entered using the TRS-80 Editor/Assembler in about 5 minutes.

Figure 13 is a print-out of the test program's source code and Figure 14 a listing of the program's object code. As may be easily seen, the majority of this program is written using Level II ROM subroutines. Were these subroutines not used in this particular assembly language test program, it would require approximately 10 times as much program memory and occupy 550 rather than 55 assembly language program lines.

Program Flow:

The comments included with the source code program delineate each line's function. There is no need to duplicate or expand upon the comments here as they are largely self-explanatory. This program operates equally well with non-disk Level II, DOS 2.1, DOS 2.2 and NEWDOS+. Program operation is as follows:

1. Load the program under the SYSTEM or DOS command. Give it any name you wish. We like the program name DISCOV, for discovery, since that is what the program is all about. After loading is complete, type in /32000 to activate the program (with disk you must first load BASIC, then type SYSTEM, ENTER, and then type in /32000 ENTER, if you loaded the program in DOS).

2. The letter 'N' ? will appear on video. The program is asking you for a number to work on. Any number up to 16 digits is all right depending on the function you wish to test. Let us start out with a simple example by entering the number 10000, a nice round easily visualized number, ENTER.

3. The numbers '2' '10000' will appear on the next line of the video display. The '2' is the number 'type' brilliantly calculated by the Level II ROM. Since we are dealing only with numbers in this article we will blithely skip over strings, etc. for the time being. The number types are as follows: 2 = integer, 4 = single precision, and 8 = double precision. Table 1 lists those operations that can be performed on a number for a given number type; i.e., it is against the rules to take the square root SQR of an integer. We must first change it.

4. On the following line of the video display you will see 'C ?'. The program is asking you what type of CONVERSION you wish. Let's enter 2737 which = CSNG, change our number from an integer to single precision, ENTER. The next line will show, '4' '10000'. We now have a single precision number to work with, so let's now try taking its square root by typing in 5095 = SQR, then ENTER. ZAP...the next line shows '100'. Ah, the miracle of modern computer science at work. It sure was easier than writing a complete stand-alone assembly language square root subroutine. Let's try it again. Type in 5095 ENTER. Again the line below displays the square root; this time the numeral 10.

5. Stick around as this is only the beginning. To insert a new number to try your program on merely type in 32000 ENTER. This brilliantly brings us back to where we started by displaying 'N ?'. Is 32000 a subroutine? Sure it is. You wrote it. Our assembly language program does not discriminate between ROM or RAM. It doesn't care.

6. We could go on and on converting numbers like deriving the natural LOG of any number and then restoring it with the EXP command, and/or deriving the TANgent of a number, then its arc tangent ATN, and then the TANgent again...ad infinitum. You may escape this conversion routine any time you wish by typing 6681 ENTER which will take you back to BASIC with a READY displayed. All you need do to return to your conversion routine is type SYSTEM then ENTER and type /32000 then ENTER.

This exercise covers only a few of the subroutines extant in Level II BASIC ROM that are illustrated in Table 1.

Assembly language programming is the Mt. Everest of our hobby. You master it and you climb it because it is there. An assembly language program may run 300 times faster and use 1/10th as much memory as the same program in BASIC.

Learning to talk to your computer in its own language rather than through an interpreter (BASIC, FORTRAN, PASCAL, or what-have-you), is probably one of the most satisfying and rewarding experiences you will ever have if you have the patience and fortitude to master it.

Addendum:

The demonstration program illustrated in Figure 13 will easily perform many more functions than the short list covered in Table 1. Make a note to come back to this program after you have finished Chapter 4 and have become familiar with data movement and data conversion subroutines.

Most of the arithmetic + - * / subroutines using integer, single precision, and double precision numbers may be used by judiciously storing one number in "CS", the CDBL Store. CALL 2556 decimal = 09FCH moves data from the ACCUM to "CS". The next number is then input by loading "32000" into CONV ? and then entering this number in the ACCUM.

By keeping close track of the NT (number type) so you call the appropriate arithmetic/conversion subroutine, and using the data movement subroutines covered in the next chapter, it is quite easy for this demo program to calculate LOG to the base 10, manipulate trig functions as desired, etc. Though you will never write a real-time program such as that given in Figure 13, it nevertheless offers you an excellent opportunity to practice and become familiar with Chapter 4's data conversion and data movement subroutines. Besides, it is an intellectual challenge...and challenges are fun when you WIN.

note: number types
2 = integer; 4 = single precision; 8 = double precision

FUNCTION	NUMBER TYPE	DECIMAL	HEXADECIMAL
ABS	2-4-8	2423	0977
ATN	4-8	5565	15BD
BASIC	(RETURN L II)	6681	1A19
BASIC	(RETURN DISK)	112	0075
BREAK	(RST ADDRESS)	16396	400C
CDBL	2-4	2779	0ADB
CINT	4-8	2687	0A7F
CLS	2-4-8	457	01C9
COS	4-8	5441	1541
CSNG	2-8	2737	0AB1
EXP	4-8	5177	1439
FIX	2-4	2854	0B26
INT	2	2871	0B37
INVERT SIGN	2	3153	0C51
INVERT SIGN	4-8	2434	0982
LOG	4-8	2057	0809
MEMORY	(DEFINE SIZE)	181	00B5
RANDOM	2-4-8	467	01D3
RETURN	(TO SUBROUTINE)	32000	7D00
RND (see limits)	2-4-8	5321	14C9
SGN	2	2442	098A
SIN	4-8	5447	1547
SQR	4-8	5095	13E7
TAN	4-8	5544	15A8

Figure 13

```
7D00              W4UCH   EQU    7D00H        ;7D00H = 32000 DECIMAL
7D00                      ORG    W4UCH        ;PROGRAM WILL START HERE
7D00 3E4E                 LD     A,4EH        ;4EH="N"=NUMBER DESIRED ?
7D02 CD2A03               CALL   032AH        ;DISPLAY "N" ON VIDEO
7D05 CDB31B               CALL   1BB3H        ;KYBD/VIDEO INPUT ROUTINE
7D08 D7                   RST    10H          ;SCAN STRING - SET C FLAG
7D09 CD6C0E               CALL   0E6CH        ;ASCII-ACCUM RET MINIMUM
7D0C 08           RETURN  EX     AF,AF'       ;EXCHANGE REGISTERS-
7D0D D9                   EXX                 ;TO PRESERVE VALUES.
7D0E 111D41               LD     DE,411DH     ;MOVE MEM ACCUM DATA FROM
7D11 21837D               LD     HL,STORE     ;TO TEMPORARY STASH
7D14 0608                 LD     B,8          ;NUMBER OF BYTES TO MOVE
7D16 CDD709               CALL   09D7H        ;MOVE IT - SUBROUTINE
7D19 112741               LD     DE,4127H     ;MOVE CDBL DATA FROM-
7D1C 217B7D               LD     HL,CDBL      ;TO TEMPORARY STASH.
7D1F 0608                 LD     B,8          ;NUMBER OF BYTES TO MOVE
7D21 CDD709               CALL   09D7H        ;MOVE IT - SUBROUTINE
7D24 3AAF40               LD     A,(40AFH)    ;NUMBER TYPE MEM LOCATION
7D27 32787D               LD     (FLAG),A     ;MOVE TO TEMPORARY STASH
7D2A C630                 ADD    A,48         ;CONVERT TO ASCII NUMBER
7D2C CD2A03               CALL   032AH        ;DISPLAY NUMBER TYPE
7D2F 3E20                 LD     A,20H        ;20H = ASCII SPACE
7D31 CD2A03               CALL   032AH        ;DISPLAY SPACE ON VIDEO
7D34 CDBD0F               CALL   0FBDH        ;CONV MEM ACCUM TO ASCII$
```

```
7D37 CDA728   00340          CALL   28A7H       ;DISPLAY CONVERTED NUMBER
7D3A 3E0D     00350          LD     A,0DH       ;0DH=SKIP A LINE/CARR RTN
7D3C CD3200   00360          CALL   032H        ;DO IT - ON VIDEO DISPLAY
7D3F 3E43     00370          LD     A,43H       ;"C" = CONVERSION NUMBER?
7D41 CD2A03   00380          CALL   32AH        ;DISPLAY "C" ON VIDEO
7D44 CDB31B   00390          CALL   1BB3H       ;KYBD/VIDEO INPUT ROUTINE
7D47 D7       00400          RST    10H         ;SCAN STRING - SET C FLAG
7D48 CD6C0E   00410          CALL   0E6CH       ;ASCII-ACCUM RET MINIMUM
7D4B CD7F0A   00420          CALL   0A7FH       ;CONVERT TO INTEGER
7D4E 22797D   00430          LD     (CONV),HL   ;STORE CONVERSION ADDRESS
7D51 117B7D   00440          LD     DE,CDBL     ;MOVE CDBL DATA FM STASH-
7D54 212741   00450          LD     HL,4127H    ;TO PERMANENT ADDRESS
7D57 0608     00460          LD     B,8         ;NUMBER OF BYTES TO MOVE
7D59 CDD709   00470          CALL   09D7H       ;MOVE IT - SUBROUTINE
7D5C 11837D   00480          LD     DE,STORE    ;MOVE MEM ACCUM FM STASH-
7D5F 211D41   00490          LD     HL,411DH    ;TO PERMANENT ADDRESS
7D62 0608     00500          LD     B,8         ;NUMBER OF BYTES TO MOVE
7D64 CDD709   00510          CALL   09D7H       ;MOVE IT - SUBROUTINE
7D67 3A787D   00520          LD     A,(FLAG)    ;NUMBER TYPE FROM STASH-
7D6A 32AF40   00530          LD     (40AFH),A   ;TO PERMANENT ADDRESS
7D6D 210C7D   00540          LD     HL,RETURN   ;RETURN MEM LOCATION-
7D70 E5       00550          PUSH   HL          ;LOADED INTO STACK
7D71 2A797D   00560          LD     HL,(CONV)   ;CONVERSION MEM LOCATION-
7D74 E5       00570          PUSH   HL          ;LOAD ON TOP OF STACK
7D75 08       00580          EX     AF,AF'      ;RESTORE REGISTERS-
7D76 D9       00590          EXX                ;TO ORIGINAL VALUES
7D77 C9       00600          RET                ;SNEAKY CALL-TOP OF STACK
0001          00610 FLAG     DEFS   1           ;NUMBER TYPE STASH
0002          00620 CONV     DEFS   2           ;CONVERSION ADDRESS STASH
0008          00630 CDBL     DEFS   8           ;CDBL DATA STASH
0008          00640 STORE    DEFS   8           ;ACCUMULATOR STASH
7D00          00650          END    W4UCH       ;AMATEUR RADIO CALL LTRS
00000 TOTAL ERRORS
CDBL   7D7B 00630   00240 00440
CONV   7D79 00620   00430 00560
FLAG   7D78 00610   00280 00520
RETURN 7D0C 00170   00540
STORE  7D83 00640   00200 00480
W4UCH  7D00 00100   00110 00650
```

Chapter 4

Ancillary Level II ROM Subroutines

Introduction:

Chapters 2 and 3 used a number of Level II ROM ancillary subroutines that were not fully explained except for a few words in the source code comment column. Using Level II ROM BASIC functions' CALL subroutines efficiently requires a modest understanding of how to use other ancillary ROM subroutines that are there just waiting to be used. They include: KEYBOARD input, MOVE data, COMPARE data, CONVERT data, VIDEO output, and LINE PRINTER output amongst others. This chapter will present the CALL addresses and briefly explain the most useful ROM ancillary subroutines that will truly make shorthand assembly language programming a reality for you.

Keyboard Review:

Every advanced assembly language programmer knows that the keyboard is simply nothing more than a key/switch matrix that "looks like" RAM memory to Level II ROM. The keyboard's eight MEM locations are shown below in decimal. UA = up arrow, DA = down arrow, LA = left arrow and RA = right arrow.

PEEK (14337)	=	@	A	B	C	D	E	F	G
PEEK (14338)	=	H	I	J	K	L	M	N	O
PEEK (14340)	=	P	Q	R	S	T	U	V	W
PEEK (14344)	=	X	Y	Z					
PEEK (14352)	=	0	1	2	3	4	5	6	7
PEEK (14368)	=	8	9	:	;	,	-	.	/
PEEK (14400)	=	ENT	CLR	BRK	UA	DA	LA	RA	SPA
PEEK (14464)	=	SHIFT							
VALUE	=	1	2	4	8	16	32	64	128

The keyboard/switch matrix will output the VALUEs shown above when a single key is pressed at the corresponding MEM location. For multiple keys pressed simultaneously, add up the values for each key; i.e., "JKL" = 4 + 8 + 16 = 28 total at MEM location 14338 decimal. With these facts in hand it is easy to see how very simply NEWDOS+ includes the feature of line printing out the video display contents whenever "JKL" are pressed simultaneously. Now, go write a brief assembly language program that will do so with non-disk Level II. For the inveterate experimenter, try this little 1 line program and press any combination of keys in the PEEK (14338) row:

```
10 X=PEEK(14338) :PRINT@478,X:GOTO10
```

The three most useful Level II ROM ancillary subroutines for the keyboard follow. CALL locations are in hex.

CALL 002B: This is the most fundamental keyboard subroutine that scans the entire keyboard and returns the ASCII character in the "A" register. A JR -Z loop must be created to repeat the scan as shown below:

```
KYBD  CALL 002BH
      CP A,00H
      JR Z,KYBD
```

Whenever a key is pressed, the CP (compare) "A" register with 00H is NOT zero so the program falls through to the line following JR Z,KYBD. This was the most commonly used keyboard subroutine by early assembly language programmers who did not know any better. It is seldom used any longer.

CALL 0049: This is the ROM subroutine most similar to BASIC's INKEY$ function. It automatically scans the keyboard UNTIL a key is pressed and then places the value in "A" register. No assembly language loop subroutine is required. A giant step forward from the 002BH fundamental keyboard CALL.

CALL 1BB3: This is the main keyboard subroutine you will be using most frequently. It first displays the "?" prompt. Then input via the keyboard is converted to string format and terminated with a zero (up to 255 bytes). This string is stored at 40A7H + string length. It is usually followed by a RST 10 which sets the "C" flag. See the programs in Chapters 2 and 3 which use this call. This call automatically outputs keyboard input to the device specified by the contents of MEM location (409CH) : -1 = cassette, 0 = video display and +1 = line printer. ROM initializes (409CH)=0. This subroutine is surely one of the most time saving, valuable, and frequently used ROM subroutines you will be using henceforth. A single CALL 1BB3H will replace dozens of lines, if not pages, of assembly language programming for you.

Data Movement:

Level II ROM ancillary subroutines exist for moving data in assembly language programs FROM - TO virtually any and all locations conceivable, often with only a single CALL. They are tremendous time and line savers and well worth becoming acquainted with on a first-hand basis.

Once of the most useful data movement subroutines is that given in the data movement tables's first line, CALL 09A4H. This CALL automatically transfers either an integer or single precision number from the ACCUM at MEM locations 411DH through 4124DH to the stack. POP BC and then POP DE will retrieve the number. If the number is single precision, registers BCDE will contain it. If an integer, register DE will hold the number.

Though ALL the data movement subroutines are very useful, those in lines "J" and "K" deserve special note. Each will MOVE up to 255 bytes from (DE) to (HL). The subroutine at "J" rquires that the number of bytes to be moved be in the "A" register, and the subroutine at "K" requires that the number of bytes to be moved be in the "B" register.

NT = number type which is stored in MEM at 40AFH: 2 = integer, 4 = double precision, and 8 = double precision. CS=CDBL store.

Data Movement Table

NO.	FROM	TO	CALL	NT(40AFH)
A.	ACCUM	STACK	09A4H/2468	2,4
B.	(HL)+	ACCUM	09B1H/2481	4
C.	BCDE	ACCUM	09B4H/2484	4
D.	ACCUM	BCDE	09BFH/2495	4
E.	(HL)+	BCDE	09C2H/2498	4
F.	ACCUM	(HL)+	09CBH/2507	4
G.	(DE)+	(HL)+	09CEH/2510	4
H.	(HL)+	(DE)+	09D2H/2514	2,4,8
I.	(DE)+	(HL)+	09D3H/2515	2,4,8
J.	(DE)+	(HL)+	09D6H/2518	A REG
K.	(DE)+	(HL)+	09D7H/2519	B REG
L.	"CS"	ACCUM	09F4H/2548	2,4,8
M.	ACCUM	"CS"	09FCH/2556	2,4,8
N.	HL	ACCUM	0A9AH/2714	2
O.	DE	HL	EX DE,HL	2
P.	HL	DE	EX DE,HL	2
Q.	BC	STACK	PUSH BC	2,4
R.	DE	STACK	PUSH DE	2,4
S.	HL	STACK	PUSH HL	2
T.	STACK	HL	POP HL	2
U.	STACK	DE	POP DE	2,4
V.	STACK	BC	POP BC	2,4

Note: Lines "O" through "V" are just plain old Z-80 OPCODES, but are included to remind the programmer that when dealing with integers or single precision numbers they often are the simplest means of moving or temporarily storing data. See Chapter 2 where PUSH and POP are used to store both integer and single precision numbers while these registers are being used for other purposes.

Data Comparisons:

Equal to, less then, and greater than are some of the most frequently used functions in computer programming. Level II ROM very thoughtfully includes these functions that may be performed with a single CALL. The result is returned in the "A" register and = zero if the compare is equal, = +1 if the compare is >, and = 255 (0FFH) if the compare is <.

Compare Table

NO.	ITEM # 1	SUBTRACT	ITEM #2	CALL	NT(40AFH)
A.	HL	-	DE	1C90H	2
B.	ACCUM	-	BCDE	0A0CH	4
C.	HL	-	DE	0A39H	2
D.	ACCUM	-	"CS"	0A4FH	8
E.	"CS"	-	ACCUM	0A78H	8
F.	ACCUM	DETERMINE	SIGN	0994H	2,4,8

Note: No. F above is same as the BASIC SGN function, but returns to register "A": zero if ACCUM = 0, +1 if ACCUM greater than zero, and 255 (0FFH) if ACCUM is less than zero.

Data Conversions:

Are straightforward and very necessary in most all arithmetic operations as the NT (number type) must match-up with the CALL subroutine's function; i.e., integer, single precision or double precision + - * /. The most useful conversions are:

DATA CONVERSIONS

CALL 0A7FH: any ACCUM to integer ACCUM (CINT).
CALL 0AB1H: any ACCUM to single precision ACCUM (CSNG).
CALL 0ACCH: integer ACCUM to single precision ACCUM.
CALL 0ACFH: integer HL to single precision ACCUM.
CALL 0ADBH: any ACCUM to double precision ACCUM.
CALL 0E65H: ASCII string to ACCUM in double precision format.
CALL 0E6CH: ASCII string to ACCUM;NT will = minimum required.

Arithmetic Call Summary

	Integer No.	Single Precision	Double Precision
Add	0BD2H/3026 DE+HL	0716H/1814 BCDE+ACCUM	0C77H/3191 ACCUM+"CS"
Subtract	0BC7H/3015 DE-HL	0713H/1811 BCDE-ACCUM	0C70H/3184 ACCUM-"CS"
Multiply	0BF2H/3058 DE*HL	0847H/2119 BCDE*ACCUM	08A2H/2210 ACCUM*"CS"
Divide	2490H/10560 DE/HL	08A2H/2210 BCDE/ACCUM	0DE5H/3557 ACCUM/"CS"

Note: NT (number type) at (40AFH) must agree with operation CALLed. NT: 2 = integer, 3 = string, 4 = single precision, and 8 = double precision.

* ACCUM at MEM locations 411DH through 4124H.
* "CS" = CDBL Store at MEM 4127H through 412EH.

Video Display:

Most TRS-80 video display subroutines have been well known to computer buffs the last 2 years, including the fundamental ROM video subroutine, CALL 033H which displays the "A" register on video. CALL 032AH which displays the "A" register on video if MEM location 409CH contains a zero which is the value stored upon initialization. Most IMPORTANTLY, CALL 032AH does indeed store the video display LINE cursor position at MEM location 40A6H which is very useful and eliminates redundant programming on your part.

One of the most important subroutines for reflecting keyboard input on video is CALL 1BB3H which was covered earlier in this Chapter. This CALL displays the string beginning at (HL) and terminated with a zero on video if MEM location 409CH contains a zero. The video display control block, page D/1, Level II Manual in conjunction with the line cursor position at 40A6H allows you to modify and/or use the video display as you wish.

One of the more fascinating assembly language exercises using the video display, is to write a "tight" source code program that creates SPLIT-SCREEN video operation. The upper half of the video display serves as the RECEIVE sector for Morse code, radio teletype (ASCII now allowed), or even simple phone line MODEM communications, while the lower half of the video display would serve as the TRANSMIT segment. This segment allows the user either "look-ahead" or "type-ahead" FIFO (first-in-first-out) operation from RAM at whatever output baud rate desired. If you have the uppercase/lowercase modification recommended by Electric Pencil, it is a simple matter to have those characters that have already been transmitted in uppercase, and those characters yet to be transmitted in lowercase. Alternatively, a moving cursor or a moving CHR$(170) figure may be used to indicate what data has been transmitted versus data yet to be transmitted in the TRANSMIT sector of the video display. Both halves of the video display operate entirely independently; i.e., from their own separate video MEMs in RAM with their own scrolling, etc. The transmit sector utilizes the Z-80's interrupt mode for "type-ahead" simplex operation. Remember, the last 7 bits of video memory is just plain old RAM and may be used for storage (as in FIFO) just like any other RAM memory segment.

Line Printer:

Much like the video display, there are few significant new surprises about line printer ROM subroutines. Again, the value stored in MEM location 409CH determines where the output from CALL 032AH and 1BB3H keyboard subroutines goes; i.e., if (409CH) contains +1, the output will go to the line printer. As shown on page D/1 of the Level II Manual, the line printer address is 37E8H and line printer control block from MEM locations 4025H to 204CH. MEM location 37E8H will contain the value 63 decimal = ASCII ? when your line printer is ready to accept another character (handshake). A few cheap surplus printers do not have this handshake feature and should be avoided like the plague; caveat emptor, especially with old Datel printers, even those with the handshake feature, do not even make good boat anchors for small dinghys. As soon as MEM location 37E8H receives the 63 handshake from your line printer it is ok (in most cases) to load the next character to be printed into (37E8H) via "A" register and the LD opcode. An exception to this rule is illustrated in Chapter 7's "Print All Zeroes With A Slash Program", where an extra 20 millisecond delay was required to allow the vibration from a BACKSPACE to dampen/die out.

This slash/zero program has not been previously published and illustrates a few interesting points about line printer programming. It intercepts the NEXT character to be printed by modifying the line printer driver address at 4026H and 4027H to allow a moment's branching to this brief routine. It justs so happens that the "C" register contains the NEXT character to be printed when using the LLIST command with non-disk Level II, DOS 2.1, DOS 2.2, and NEWDOS+, as well as the NEWDOS+ "JKL" feature that LPRINTS the contents of the video display. By simply testing the next character to be printed, a variety of options are made available to the programmer.

To illustrate a few points, let us assume that your line printer utilizes IBM's highest-quality heavy-duty Selectric mechanism like the Western I/O (IBM #2970) Printer Terminal. These can be purchased used and refurbished with interface to the TRS-80, for $1100. Only 8 years ago, these IBM #2970's sold new for over $7000 each. Today, they are the industry's most cost-effective printers. They are the best choice for those who demand IBM quality print-out in both lower and upper case,, in addition to the decided advantage of being able to use any or all of the myriad type faces offered in inexpensive IBM Selectric snap-in elements.

Even the excellent ASCII IBM elements do not include the zero with a slash across it as it looks strange indeed to non-computer types reading a business letter. In some program listings though, it is of considerable assistance to the reader to have the slash/zero printed as such, to avoid confusing zeros with capital "O". Chapter 7's short program prints all zeros with a slash and all lines with 64 characters. It may be modified to print all > = GT and all < = LT, etc., as desired. As it is a teaching program, it may be compacted considerably. Try cutting it down by 1/3rd.

Chapter 5

Summary of Level II ROM CALL Addresses In Alphabetical Order

This summary includes the coupling CALL addresses for DOS and Disk BASIC because they are called from Level II ROM. In addition, the link address for BREAK is included as it may be intercepted at 400CH before calling an RST and either rendered inoperative or used for whatever purpose the programmer wishes. One extra bonus for disk users under the heading "MASTER" is the master password "F3GUM", which will allow you to access ANY protected file in either DOS or Disk BASIC when using DOS 2.1, DOS 2.2, or NEWDOS+. (Thank you, Manny Garcia.) Every lock has a key and "F3GUM" will allow you to LOAD, KILL, transfer or do whatever you wish with any disk file/program whether SIP (system-invisible-protected) or otherwise.

BASIC FUNCTION	HEX CALL ADDRESS
ABS	0977
ASC	2A0F
AUTO	2008
CDBL	0ADB
CINT	0A7F
CLOAD	2C1F
CLS	01C9
CONT	1DE4
CSAVE	2BF5
CVD	415E
CVS	4158
DEF	415B
DEFINT	1E03
DEFSTR	1E00
DIM	2608
ELSE	1F07
EOF	4161
ERR	24CF
EXP	1439
FIX	0B26
FN	4155
GET	417F
GOTO	1EC2
INKEY$	019D
INPUT	219A
INT	0B37
LEFT$	2A61
LET	1F21
LIST	2B2E
LOC	4164
LOG	0809
LPRINT	2067
MASTER	F3GUM
MERGE	418B
MKD$	4170
MKS$	416D
NEW	1B49
NOT	25C4
OPEN	4179
PEEK	2CAA
POKE	2CB1
PRINT	206F
RANDOM	01D3
REM	1F07
RESTORE	1D91
RESUME	1FAF
RND	14C9
RUN	1EA3
SET	0135
SIN	1547
STOP	1DA9
STRING$	2A2F
TAN	15A8
TROFF	1DF8
USR	27FE
VAL	2AC5

* footnote: ELSE = 1F07 hex is questionable even though ROM points to his address. ROM also points to 1F07 for the REM function which appears correct. It may be only an improperly shadow-masked bit on this particular Level II ROM chip.

BASIC FUNCTION	HEX CALL ADDRESS
&H	4194
ATN	15BD
BREAK	400C
CHR$	2A1F
CLEAR	1E7A
CLOSE	4185
CMD	4173
COS	1541
CSNG	0AB1
CVI	4152
DATA	1F05
DEFDBL	1E09
DEFSNG	1E06
DELETE	2BC6
EDIT	2E60
END	1DAE
ERL	24DD
ERROR	1FF4
FIELD	417C
FOR	1CA1
FRE	27D4
GOSUB	1EB1
IF	2039
INP	2AEF
INSTR	419D
KILL	4191
LEN	2A03
LINE	41A3
LOAD	4188
LOF	4167
LLIST	2B29
LSET	4197
MEM	27C9
MID$	2A9A
MKI$	416A
NAME	418E
NEXT	22B6
ON	1F6C
OUT	2AFB
POINT	0133
POS	27F5
PUT	4218
READ	21EF
RESET	0138
RETURN	1EDE
RIGHT$	2A91
RSET	419A
SAVE	41A0
SGN	098A
SQR	13E7
STR$	2836
SYSTEM	02B2
TIME$	4176
TRON	1DF7
VARPTR	24EB

Chapter 6

```
500 CLS:CLEAR200
510 CMD"T
520 PRINT
530 'ONERRORGOTO540
540 'RESUME550
550 PRINT" W 4 U C H   N U M B E R   C O N V E R S I O N   P R O
 G R A M
560 PRINT
570 PRINT"          - DECIMAL TO BINARY ENTER D
580 PRINT"          - BINARY TO DECIMAL ENTER B
590 PRINT"          - HEXADECIMAL TO BINARY ENTER HB
600 PRINT"          - DECIMAL TO HEXADECIMAL ENTER DH
610 PRINT"          - HEXADECIMAL TO DECIMAL ENTER HD
620 PRINT"          - SPLIT DECIMAL TO DECIMAL ENTER SP
630 PRINT"          - DECIMAL TO SPLIT HEXADECIMAL ENTER DS
640 PRINT"          - SPLIT HEXADECIMAL TO DECIMAL ENTER SD ";
650 INPUTAA$:CLS
660 IFAA$="HB"GOTO1550
670 IFAA$="DH"THENQB$=""
680 IFAA$="DS"THENAA$="DH":QB$="DS
690 IFAA$="SD"GOTO1500
700 IFAA$="HD"GOTO1550
710 IFAA$="B"GOTO860
720 REM DECIMAL TO BINARY CONVERSION
730 CLS:INPUT" DECIMAL NO. ";X:IFX>65535GOTO730
740 A=INT(X/2):AA=X-2*A:B=INT(A/2):BB=A-2*B:C=INT(B/2):CC=B-2*C
750 D=INT(C/2):DD=C-2*D:E=INT(D/2):EE=D-2*E:F=INT(E/2):FF=E-2*F
760 G=INT(F/2):GG=F-2*G:H=INT(G/2):HH=G-2*H:I=INT(H/2):II=H-2*I
770 J=INT(I/2):JJ=I-2*J:K=INT(J/2):KK=J-2*K:L=INT(K/2):LL=K-2*L
780 M=INT(L/2):MM=L-2*M:N=INT(M/2):NN=M-2*N:O=INT(N/2):OO=N-2*O
790 PP=O-2*INT(O/2)
800 Y$=STR$(PP)+STR$(OO)+STR$(NN)+STR$(MM)+STR$(LL)+STR$(KK)+STR
$(JJ)+STR$(II)+STR$(HH)+STR$(GG)+STR$(FF)+STR$(EE)+STR$(DD)+STR$
(CC)+STR$(BB)+STR$(AA)
810 IFAA$="SP"THENNO=NO+1:QB$="DS":GOTO1200
820 IFAA$="DH"ANDQB$=""THENPRINT:PRINT" HEXADECIMAL  ";:GOTO1200
830 IFQB$="DS"GOTO1200
840 PRINT" BINARY #"Y$:INPUTR:GOTO730
850 REM BINARY TO DECIMAL CONVERSION
860 CLS:INPUT" 16,  8, OR  4 DIGIT BINARY NO. ";AA
870 CLS:PRINTAA;"DIGIT BINARY NO. ";:INPUTX$
880 IFLEN(X$)<>AAGOTO870
890 FORZ=1TOAA:X=VAL(MID$(X$,Z,1)):IFZ=1THENA=X
900 IFZ=2THENB=X
910 IFZ=3THENC=X
920 IFZ=4THEND=X
930 IFAA=8GOTO970ELSEIFAA=16GOTO970
940 NEXT
950 Y=A*8+B*4+C*2+D
960 PRINT"          DECIMAL"Y:INPUTR:GOTO870
970 IFZ=5THENE=X
980 IFZ=6THENF=X
990 IFZ=7THENG=X
1000 IFZ=8THENH=X
1010 IFAA=16GOTO1050
1020 NEXT
1030 Y=A*128+B*64+C*32+D*16+E*8+F*4+G*2+H
1040 PRINT"          DECIMAL"Y:INPUTR:GOTO870
1050 IFZ=9THENI=X
```

```
1060 IFZ=10THENJ=X
1070 IFZ=11THENK=X
1080 IFZ=12THENL=X
1090 IFZ=13THENM=X
1100 IFZ=14THENN=X
1110 IFZ=15THENO=X
1120 IFZ=16THENP=X
1130 NEXT
1140 Y=A*32768+B*16384+C*8192+D*4096+E*2048+F*1024+G*512+H*256+I
*128+J*64+K*32+L*16+M*8+N*4+O*2+P
1150 PRINT:PRINT"          DECIMAL";Y:INPUTR
1160 IFAA$="SP"GOTO730
1170 IFAA$="B"GOTO870
1180 IFAA$="HD"GOTO1540
1190 IFAA$="SD"GOTO1500
1200 DH$=MID$(Y$,1,8):GOSUB1250
1210 DH$=MID$(Y$,9,8):GOSUB1250
1220 DH$=MID$(Y$,17,8):GOSUB1250
1230 DH$=MID$(Y$,25,8):GOSUB1250
1240 PRINT"                  ";:INPUTR:GOTO730
1250 DH=VAL(DH$):IFDH=0THENDH$="0
1260 IFDH=1THENDH$="1
1270 IFDH=10THENDH$="2
1280 IFDH=11THENDH$="3
1290 IFDH=100THENDH$="4
1300 IFDH=101THENDH$="5
1310 IFDH=110THENDH$="6
1320 IFDH=111THENDH$="7
1330 IFDH=1000THENDH$="8
1340 IFDH=1001THENDH$="9
1350 IFDH=1010THENDH$="A
1360 IFDH=1011THENDH$="B
1370 IFDH=1100THENDH$="C
1380 IFDH=1101THENDH$="D
1390 IFDH=1110THENDH$="E
1400 IFDH=1111THENDH$="F
1410 IFQB$="DS"THENQQ=QQ+1
1420 IFQQ=1THENRR$=DH$:RETURN
1430 IFQQ=2THENSS$=DH$:RETURN
1440 IFQQ=3THENTT$=DH$:RETURN
1450 IFQQ=4THENUU$=DH$
1460 IFQB$="DS"ANDAA$="DH"THENPRINT:PRINT" SPLIT HEX   ";TT$;UU$
;RR$;SS$:QQ=0:INPUTR:GOTO730
1470 IFQB$="DS"ANDAA$="SP"ANDNO=1THENFF$=TT$+UU$:QQ=0:GOTO730
1480 IFQB$="DS"ANDAA$="SP"ANDNO=2THENSS$=TT$+UU$:X$=SS$+FF$:NO=0
:QQ=0:PRINT" HEXADECIMAL ";X$:GOTO1560
1490 PRINTDH$;:RETURN
1500 'SPLIT HEX TO DECIMAL CONVERSION
1510 CLS:INPUT"SPLIT HEX NO.     ";SH$
1520 SI$=LEFT$(SH$,2):SJ$=RIGHT$(SH$,2)
1530 X$="":X$=SJ$+SI$:GOTO1560
1540 REM HEX TO DECIMAL AND HEX TO BINARY CONVERSION
1550 X$="":INPUT"HEXADECIMAL NO. ";X$
1560 IFLEN(X$)<>4GOTO1550
1570 FORXX=1TO4:A$=MID$(X$,XX,1)
1580 IFA$="0"THENA$="0000
1590 IFA$="1"THENA$="0001
1600 IFA$="2"THENA$="0010
```

```
1610 IFA$="3"THENA$="0011
1620 IFA$="4"THENA$="0100
1630 IFA$="5"THENA$="0101
1640 IFA$="6"THENA$="0110
1650 IFA$="7"THENA$="0111
1660 IFA$="8"THENA$="1000
1670 IFA$="9"THENA$="1001
1680 IFA$="A"THENA$="1010
1690 IFA$="B"THENA$="1011
1700 IFA$="C"THENA$="1100
1710 IFA$="D"THENA$="1101
1720 IFA$="E"THENA$="1110
1730 IFA$="F"THENA$="1111
1740 NN=NN+1
1750 ONNNGOTO1760,1770,1780,1790
1760 BB$=A$:              NEXTXX:GOTO1570
1770 CC$=A$:              NEXTXX:GOTO1570
1780 DD$=A$:              NEXTXX:GOTO1570
1790 EE$=A$
1800 X$=BB$+CC$+DD$+EE$:NN=0
1810 IFAA$="HB"THENPRINTX$;" BINARY  ";:INPUTR:CLS:GOTO1550
1820 AA=16:GOTO890
1830 END
1840 '
1850 ' NOTE: PROGRAM UTILIZES 3908 BYTES OF MEM
```

Chapter 7

```
                00100 ; SOURCE CODE PROGRAM TO PRINT ALL ZEROS WITH A SLASH
                00110
                00120 ; WITH AUTO CARRIAGE RETURN @ 64 CHARACTERS WHEN LLIST
                00130
7F15            00140           ORG     7F15H   ;=32553 DECIMAL ORIGIN
7F15            00150 COUNT     EQU     7F15H   ;MEM LOCATION FOR CHAR. COUNTER
7F15 157F       00160           DEFW    COUNT   ;ASSEMBLER SAVES 2 BYTES @ 7F15H
7F17 F5         00170 START     PUSH    AF      ;SAVE REGISTERS IN STACK MEMORY
7F18 C5         00180           PUSH    BC
7F19 D5         00190           PUSH    DE
7F1A E5         00200           PUSH    HL
7F1B 79         00210           LD      A,C     ;NEXT CHARACTER TO PRINT IN C REG
7F1C FE0D       00220           CP      0DH     ;IS IT A CARRIAGE RETURN?
7F1E 281B       00230           JR      Z,NOINC ;EXIT W/O INCREMENTING COUNTER
7F20 3A157F     00240           LD      A,(COUNT)       ;COUNTER VALUE FROM MEM
7F23 FE40       00250           CP      040H    ;IS IT 64 DECIMAL = END OF LINE?
7F25 2822       00260           JR      Z,CARRET        ;IF SO, GOTO CARR. RETURN
7F27 3A157F     00270 FINIS     LD      A,(COUNT)       ;IF RETURN FM CARRET
7F2A 3C         00280           INC     A       ;ADD +1 CHARACTER ON LINE
7F2B 32157F     00290           LD      (COUNT),A       ;NEW COUNT NUMBER TO MEM
7F2E 79         00300           LD      A,C     ;NEXT CHARACTER TO PRINT IN C REG
7F2F FE30       00310           CP      30H     ;30H=ASCII ZERO  IS IT A ZERO?
7F31 CC5B7F     00320           CALL    Z,ZERO
7F34 E1         00330 ELFIN     POP     HL      ;RESTORE REGS TO ORIG. CONDX
7F35 D1         00340           POP     DE
7F36 C1         00350           POP     BC
7F37 F1         00360           POP     AF
7F38 C38D05     00370           JP      058DH   ;GOTO ROM STD. PRINTER ROUTINE
7F3B 3E00       00380 NOINC     LD      A,00H   ;RESET CHARACTER COUNTER-
7F3D 32157F     00390           LD      (COUNT),A       ;IN MEMORY
7F40 18F2       00400           JR      ELFIN   ;QUICK DEPARTURE
7F42 3E00       00410 RESET     LD      A,00H   ;RESET  CHARACTER-
7F44 32157F     00420           LD      (COUNT),A       ;COUNTER IN MEMORY.
7F47 18DE       00430           JR      FINIS   ;ALL DONE
7F49 CD537F     00440 CARRET    CALL    TEST    ;TEST IF PRINTER READY?
7F4C 3E0D       00450           LD      A,0DH   ;0DH = ASCII CARRIAGE RETURN
7F4E 32E837     00460           LD      (37E8H),A       ;DO CARRIAGE RETURN
7F51 18EF       00470           JR      RESET   ;GOTO RESET CHARACTER COUNTER
7F53 3AE837     00480 TEST      LD      A,(37E8H)       ;PRINTER MEM LOCATION
7F56 CB7F       00490           BIT     7,A     ;PRINTER READY HANDSHAKE?
7F58 20F9       00500           JR      NZ,TEST         ;LOOP TIL HANDSHAKE
7F5A C9         00510           RET             ;RETURN TO LINE AFTER CALL
7F5B CD537F     00520 ZERO      CALL    TEST    ;IS PRINTER READY?
7F5E 3E2F       00530           LD      A,2FH   ;2FH=ASCII SLASH
7F60 32E837     00540           LD      (37E8H),A       ;PRINT '/' SLASH
7F63 CD537F     00550           CALL    TEST    ;IS PRINTER READY?
7F66 CD6E7F     00560           CALL    DELAY   ;20 MILLISECOND DELAY
7F69 3E08       00570           LD      A,08H   ;08H = ASCII BACKSPACE
7F6B 32E837     00580           LD      (37E8H),A       ;DO BACKSPACE
7F6E 0E0A       00590 DELAY     LD      C,0AH   ;INITIALIZE DELAY-
7F70 0600       00600 DELAY1    LD      B,0H    ;LOOPS TO ALLOW SELECTRIC-
7F72 10FE       00610 DELAY2    DJNZ    DELAY2  ;PRINT HEAD TIME TO SETTLE DOWN-
7F74 0D         00620           DEC     C       ;FROM BACKSPACE VIBRATION AND-
7F75 C2707F     00630           JP      NZ,DELAY1       ;FOR TRACK TO LOCK.
7F78 C9         00640           RET             ;RETURN TO LINE AFTER CALL DELAY
4026            00650           ORG     4026H   ;PUT ADDRESS OF START IN PRINTER-
4026 177F       00660           DEFW    START   ;DRIVER ADDRESS AT 4026H MEMORY.
```

```
7F15          00670         ORG      COUNT    ;CHARACTER COUNTER MEM ADDRESS-
7F15 00       00680         DEFB     00H      ;INITIALIZE AT ZERO.
7F15          00690         END      COUNT    ;FIRST LINE OF SUBROUTINE
00000 TOTAL ERRORS
CARRET 7F49 00440   00260
COUNT  7F15 00150   00160 00240 00270 00290 00390 00420 00670
                    00690
DELAY  7F6E 00590   00560
DELAY1 7F70 00600   00630
DELAY2 7F72 00610   00610
ELFIN  7F34 00330   00400
FINIS  7F27 00270   00430
NOINC  7F3B 00380   00230
RESET  7F42 00410   00470
START  7F17 00170   00660
TEST   7F53 00480   00440 00500 00520 00550
ZERO   7F5B 00520   00320
```

Chapter 8

Self-Test Questions

The following pages of self-test questions cover a number of important points in the preceding chapters and demonstration programs. If you understand the logic, program flow, and rationale of each chapter/program's contents, you should have little difficulty in answering the questions. If a question's answer is not clear, re-read/study the appropriate chapter and/or demonstration program till osmosis occurs, as in time it will indeed permeate. If all else fails, try putting this handbook under your pillow and sleeping on it. (ED.)

Self-Test Questions for Chapter 1:

1. What is meant by masking the most significant bit (MSB)?

____

2. What are the decimal and hexadecimal locations in ROM for the Level II functions' NAMES?

From ______________________________

/ ______________________________

to ______________________________

/ ______________________________

______________________________.

3. Why are disk BASIC's functions' names and coupling addresses in non-disk Level II BASIC ROM?

____

4. How would you mask the MSB of any MEM location in a simple program written in BASIC?

____

5. How many MEM groups (separate address segments) do the BASIC functions' CALL addresses occupy? Beginning locations?

A. ______________________________

B. ______________________________

C. ______________________________

D. ______________________________

____

6. Name the 2 geniuses at Microsoft who wrote Level II and disk BASIC (luckily they did not write DOS 2.1 or 2.2).

A ______________________________

G ______________________________

7. Why are the BASIC functions' CALL addresses in Level II ROM expressed LSB first and MSB second?

____

8. What are the decimal values of PEEK 5666 and 5667? What Level II function's CALL address do they represent?

A. ______________________________

/ ______________________________

B. ______________________________

9. What are the MEM locations in decimal and hex of the following PEEK values? A. 194-30 B. 15-42 C. 255-255 D. 0-255

A. ______________________________

/ ______________________________

B. ______________________________

/ ______________________________

C. ______________________________

/ ______________________________

D. ______________________________

/ ______________________________

Self-Test - Chapter 2:

1. On the average, how much faster will efficiently written assembly language programs run than BASIC? MEM required?

A.

B.

2. What do we call the 2 RAM arithmetic storage locations?

A.

B.

3. What is the NT (number type) for the following numbers?

A. 10000

B. 33000

C. 1.011

D. 1.1111111

4. Where is the NT (number type) stored in RAM memory?

5. What type of numbers are represented by NT (number type) =?

A. 2 =

B. 3 =

C. 4 =

D. 8 =

6. In Figure 7, what is accomplished by line 310?

7. In Figure 7, what is accomplished by line 320?

8. In Figure 9, what is accomplished by lines 340/350?

9. In Figure 11, what is accomplished by line 310?

10. How many significant digits in a CDBL argument?

NOTES

Self-Test Questions - Chapter 3:

1. Does Level II ROM ever use the Z-80 alternate register pairs AF' - BC' - DE' - HL' ?

2. Where is the ROM CALL location to move "B" register bytes from MEM location XXXX to MEM location YYYY?

3. How many MEM locations in the ACCUM and CDBL store? Where?

4. How is the NT (number type) converted to ASCII?

5. What CALL converts the ACCUM to an ASCII string?

6. What function does Figure 13's line 420 perform? Why?

7. What is the difference between CINT and INT?

8. What is the difference between RANDOM and RND?

9. How is the conversion MEM location CALLed in Figure 13?

10. How is the RETURN MEM location CALLed in Figure 13?

Self-Test Questions - Chapter 4:

1. What is the value of PEEK (14464) if one shift key is pressed? If both shift keys are pressed?

2. What is the value of PEEK (14338) if H, I, J, K, L, M, N, and O keys are pressed simultaneously?

3. What is the major difference between CALL 002BH and 0049H?

4. What data is stored at MEM location 409CH?

5. What is the simplest way to store an integer and single precision number in RAM (not counting the ACCUM and CDBL store)?

6. Where are data comparison results stored? What values?

7. What is the value of the "A" register if CALL 0994H is used and the ACCUM contains -1.9999999999?

8. What CALL is used to change any value to an integer? To a single precision number? To a double precision number?

9. What is the value of NT RAM storage for a string?

10. What value will MEM location 37E8H contain when the line printer is ready (handshake)? What MEM location is loaded with the NEXT character to be printed?

11. Write a brief assembly language program to output to the video display a message of ANY length using the CALL 28A7H subroutine.

Note: This subroutine will output a string of up to 63 characters with non-disk Level II. By concatenating the strings with additional DEFM OPCODES, the message length is only limited to 240 bytes/CALL 28A7H.

Self-Test Questions - Chapter 5:

Fill in decimal CALL locations for these Level II functions using Chapter 7's Multi-Base Conversion Program:

BASIC FUNCTION	CALL ADDRESS	BASIC FUNCTION	CALL ADDRESS
ABS		ASC	
ATN		AUTO	
CDBL		CHR$	
CINT		CLEAR	
CLOAD		CLS	
CONT		COS	
CSAVE		CSNG	
DATA		DEFDBL	
DEFINT		DEFSNG	
DEFSTR		DELETE	
DIM		EDIT	
ELSE		END	
ERR		ERL	
ERROR		EXP	
FIX		FOR	
FRE		GOSUB	
GOTO		IF	
INKEY$		INP	
INPUT		INSTR	
INT		LEFT$	
LEN		LIST	
LOG		LLIST	
LPRINT		MEM	
MID$		NEW	
NEXT		NOT	
ON		OUT	
PEEK		POINT	
POKE		POS	
PRINT		RANDOM	
READ		REM	
RESET		RESTORE	
RESUME		RIGHT$	
RND		RUN	
SET		SGN	
SIN		SQR	
STOP		STR$	
STRING$		SYSTEM	
TAN		TROFF	
TRON		USR	
VARPTR		VAL	

Self-Test Questions - Chapters 6 and 8:

1. A. In Chapter 6's "Number Conversion Program", is line 510 only for disk BASIC? B. Should it be deleted for non-disk?

---

2. How many fundamental number conversions are performed by Chapter 6's "Number Conversion Program"?

---

3. How is the "SPLIT DECIMAL TO DECIMAL" conversion performed?

---

4. In Chapter 7's program, rewrite lines 170-200 and 330-360 to use EX AF,AF' and EXX OPCODES instead of the stack for storage. Be VERY careful.

---

5. A. In line 490, rewrite this line to use the CP (compare) OPCODE instead of BIT 7,A. B. What is the difference?

---

6. Why are lines 590-630 necessary with IBM Selectric printers?

---

7. If you had a line printer with 130 characters per line capability (most IBM Selectric Printer Terminals have it) how would you modify this program to use all 130 positions?

Bibliography and Answers to Self-Test Questions

Bibliography and Books Available:

Using the consumers' scale of: BEST buy, GOOD buy, FAIR buy, POOR buy, and utterly WRETCHED, there is nothing on the market today for the TRS-80 assembly programmer that rates much higher than FAIR in this reviewer's opinion, with two exceptions. They are both for the advanced assembly language programmer and are:

BEST:

Andrew Hildebrand's
"Software Technical Manual", @ $40.
Houston Micro Computer Technologies
5313 Bissonnet Street
Bellaire, Texas 77401

GOOD:

Lance Leventhal's
"Z-80 Assembly Language Programming", @ $15.
Adam Osborne & Associates
630 Bancroft Way
Berkeley, California 94710

Both of the above books assume that the reader has at least a few years of experience of 8080A assembly language programming experience before jumping into the Z-80.

There is really nothing truly worthwile available for the beginning assembly language programmer. This includes Radio Shack's "TRS-80 Assembly Language Programming," at $3.95. Even at $3.95 it is grossly OVERPRICED and in our kindest moments deserves our WRETCHED rating. There is nothing else for the beginner but garbage, but if one is truly hungry enough and holds one's nose, garbage is better than starving to death.

The following books have received our FAIR consumers' rating because of considerable generosity on our reviewer's part.

FAIR:

"Z-80 Programming For Logic Design"
great for typewriter mechanics

"The Z-80 Microcomputer Handbook"
the cover should be a clue to the buyer

"Z-80 Software Gourmet Guide and Cookbook"
cooked tripe is still tripe

What the 200,000+ TRS-80 owners above the 5th grade level are awaiting is W6OVP's new book on the subject of assembly language programming. W6OVP is Dr. David Lien, Dean of San Diego University, a professional educator.

Dave Lien wrote the "User's Manual for Level I," which is certainly the finest tutorial for beginning BASIC programmers from age 12 to 80 ever published, and followed it up in the fall of '79 with "Learning Level II," which also deserves an excellent rating. If and when Dave publishes tutorials on disk BASIC and assembly language programming, they will indeed be on the best-seller list.

Chapter 9

Answers to Questions on Chapters 1 to 7

Note:

Advanced assembly language programmers' will undoubtedly think many of the questions naive to the point of ridiculousness. Indeed, many questions are absurdly simple to DRIVE HOME points to those readers who may NOT be as advanced as others. It is a difficult balancing act, so we ask for your patience and understanding.

Answers to Chapter 1:

1. Assume that PEEK (5712) = 197 decimal. 5712 is MEM location 1650H. Now, 197 decimal = 11000101 binary. The left most bit is a 1 which is masked (eliminated). It now = 1000101 binary = 69 decimal = ASCII "E". Since 10000000 binary = 128 decimal this same bit may be masked by simply subtracting 128 from the PEEK decimal value.

2. From 5712 decimal/1650H to 6172 decimal/181CH.

3. They are all CALLed by non-disk Level II ROM. If no disk is present or operating = L3/ERROR.

4. Subtract 128 from the PEEK value.

5. Two.

6. From 6178 decimal/1822H to 6297 decimal/1899H, and from 5642 decimal/160AH to 5711 decimal/164FH.

7. Convenience plus allowing modest upward compatibility between the earlier 8080 microprocessor and the newer Z-80, both invented/designed by Dr. Frederico Faggin (President of Zilog).

8. 189 and 21 decimal.

9. ATN = arc tangent with output value in radians.

10. A. 7874 decimal/1EC2H
B. 10767 decimal/2A0FH
C. 65535 decimal/FFFFH
D. 65280 decimal/FF00H

Answers to Chapter 2:

1. A. 300 to 350+ times faster.
B. 1/10th as much memory.

2. ACCUM and CDBL Store; ("CS" abbreviation).

3. A. = 2
B. = 4 (> 32767)
C. = 4
D. = 8
NOTE: 3 = string

4. NT MEM location at 40AFH.

5. A. 2 = integer
B. 4 = single precision
C. 8 = double precision.

6. CALL 0E6CH changes an arithmetic string to minimum NT and stores it in the ACCUM in the appropriate format, plus storing the NT in MEM location 40AFH.

7. CALL 0A7FH = CINT which changes ANY number in the range of +32767 to -32768 to an integer.

8. Uses stack MEM in RAM to store the single precision value of registers BCDE while BCDE are used in following CALLs.

9. See answer to question 6, above. CALL 0E65H is similar, but stores the number in the ACCUM in double precision format and sets the NT in MEM location 40AFH to 8.

10. THIS MAY SURPRISE YOU. Answer: only as many digits as contained in the argument. Up to 17 digits maximum. If the argument contains 10 significant digits, the last 7 digits will be meaningless. See Radio Shack's "Microcomputer Newsletter", October '79, page 2 for an excellent explanation.

Answers to Chapter 3:

1. NO, they are never used in Level II ROM as this Microsoft BASIC is sort of the son of an earlier 8080 BASIC and the 8080 has neither OPCODE. This allows YOU to use it whenever desired and save 4+ bytes compared with PUSH and POP.

2. CALL 09D7H/2519 decimal.

3. Eight in each one.

4. ADD A,48 ;converts to ASCII.

5. CALL 0FBDH converts ACCUM to a string (address in HL), and NT in MEM location 40AFH = 3.

6. A. Converts any number in CINT range in ACCUM to integer.
B. All ROM CALL locations are integers, so acts as a trap for erroneous input.

7. RANDOM re-initializes the Z-80 random number generator. RND with NT = 2 = integer, generates a pseudo-random number between 1 and the integer value in the ACCUM. RND with NT = 4 = a single precision number, generates a pseudo random single precision number between zero and the number in the ACCUM which must = or > 1.

8. INT returns a round number (no decimal points) for ANY number. CINT changes any number with up to 7 digits to an integer within the range of +32767 and -32768.

9. Line 600's OPCODE is a RET which effectively POPs the top number off the stack, thus CALLing the CONV MEM location.

10. Since each listed CONVersion subroutine ends with a RET, this effects a POP the top number off of the stack, which is the RETURN address. This is a very useful ploy at times.

Answers to Chapter 4:

1. A. 1
B. 1

2. 255 decimal

3. CALL 002BH scans the keyboard ONCE. CALL 0049H scans the keyboard till NOT zero (till any key is pressed). Virtuallay the same as INKEY$ in BASIC.

4. MEM location 409CH stores "output" directions for CALL 1BB3H (among others). Initialized at Zero = video display, +1 = line printer, and -1 = cassette.

5. A. PUSH DE or PUSH HL.
B. PUSH BC and PUSH DE.
C. Move 8 bytes from ACCUM to MEM via CALL 09D6H or 09D7H.

6. A. In the "A" register.
B. Zero.
C. +1.
D. 255 (0FFH).

7. 255 decimal - 11111111 binary - 0FFH hex

8. A. CALL 0B37H/2871 decimal = INT; CINT's range is limited.
B. CALL 0AB1H/2737 decimal = CSNG
C. CALL 0ADBH/2779 decimal = CDBL.

9. 3 decimal

10. A. 63 decimal = ASCII ? (what next?).
B. 37E8H

11.

```
W4UCH    EQU     7D00H                  ;MESSAGE PROGRAM
         ORG     W4UCH                  ;7D00H = 32000 DECIMAL
         LD      HL,STRING              ;STRING MEM ADDRESS
         CALL    28A7H                  ;DISPLAY $ SUBROUTINE
                                        ;SEE NOTE BELOW
STRING   DEFM    'USING LEVEL II ROM    SUBROUTINES'
         DEFB    0                      ;END OF MESSAGE DELIMITER
         JP      1A19H                  ;RETURN TO BASIC
         END     W4UCH                  ;CONCATENATE ANY LENGTH TO 240 BYTES.
```

Answers to Chapter 5

Level II BASIC Function Call Addresses in Decimal

BASIC FUNCTION	CALL ADDRESS	BASIC FUNCTION	CALL ADDRESS
ABS	2423	ASC	10767
ATN	5565	AUTO	8200
CDBL	2779	CHR$	10783
CINT	2687	CLEAR	7802
CLOAD	11295	CLS	457
CONT	7652	COS	5441
CSAVE	11253	CSNG	2737
DATA	7941	DEFDBL	7689
DEFINT	7683	DEFSNG	7686
DEFSTR	7680	DELETE	11206
DIM	9736	EDIT	11872
ELSE	7943	END	7598
ERL	9437	ERR	9423
ERROR	8180	EXP	5177
FIX	2854	FOR	7329
FRE	10196	GOSUB	7857
GOTO	7874	IF	8249
INKEY$	413	INP	10991
INPUT	8602	INSTR	16787
INT	2871	LEFT$	10849
LEN	10755	LIST	10054
LOG	2057	LLIST	11049
LPRINT	8295	MEM	10185
MID$	10906	NEW	6985
NEXT	8886	NOT	9668
ON	8044	OUT	11003
PEEK	11434	POINT	307
POKE	11441	POS	10229
PRINT	8303	RANDOM	467
READ	8687	REM	7943
RESET	312	RESTORE	7569
RESUME	8111	RIGHT$	10897
RND	5321	RUN	7843
SET	309	SGN	2442
SIN	5447	SQR	5095
STOP	7593	STR$	10294
STRING$	10799	SYSTEM	690
TAN	5544	TROFF	7672
TRON	7671	USR	10238
VARPTR	9461	VAL	10949

Answers to Chapters 6 and 7:

1. A. YES: to speed-up disk a bit
   B. YES: remove from non-disk

2. A. To round-off division by 2 to an integer = a short-cut in the decimal to binary conversion equation.
   B. 4 fundamental conversions — rest are only messages.

- decimal to binary	: lines	740 -	840
- binary to decimal	: lines	860 -	1150
- binary to hexadecimal	: lines	1200 -	1490
- hexadecimal to binary	: lines	1550 -	1810

3. The first decimal number is converted to hex and stored in FF$ in line 1470. The second decimal number is then converted to hex and stored in SS$ in line 1480. FF$ and SS$ are then reversed and made = to X$ in line 1480. The correct hex number which is X$ is then displayed on video in line 1480. This hex number, X$, is then converted to binary, and next converted to decimal, and then displayed on video. Though very SHOW when decimal, and then displayed on video. Though very SLOW when written in BASIC, it still beats a Hewlett-Packard calculator.

4.

```
00170 START    EX     AF,AF'    ;EXCHANGE ALTERNATE REGS.
00180          LD     A,C       ;NEXT CHAR. TO "A" REGISTER
00190          EXX              ;EXCHANGE BC DE HL ALT. REG
```

delete lines 00200 and 00210

NOTE: Since the NEXT character to be printed is in the "C" register, we MUST load it into "A" BEFORE exchange.

```
00330 ELFIN    EX     AF,AF'    ;EXCHANGE ALTERNATE REGS.
00340          EXX              ;EXCHANGE BD DE HL ALT. REG
```

delete lines 00350 and 00360

NOTE: This little exercise deleted 4 lines and 4+ bytes. A very decided improvement and good technique.

5. 00490 CP 3FH ;is it ASCII ? = 63 decimal

6. Backspace vibrates most IBM Selectric printers quite severely. Try your printer without these lines. Use only if necessary.

7. Change line 250 to read:

```
00250          CP     082H      ;is it 130 decimal = end of line?
```

FINAL REQUEST: Please send ANY errors you find to:

RICHCRAFT ENGINEERING LTD
DRAWER 1065
CHAUTAUQUA, NY 14722

Chapter 10

Level II ROM Function Address Table

Function	Token	Address	Function	Token	Address	Function	Token	Address
ABS	D9	0977	GOSUB	91	1EB1	READ	8B	21EF
AND	D2	25FD	GOTO	8D	1EC2	REM	93	1F07
ASC	F6	2A0F	1F	8F	2039	RESET	82	0138
ATN	E4	15BD	INKEY$	C9	019D	RESTORE	90	1D91
AUTO	B7	2008	INP	DB	2AEF	RESUME	9F	1FAF
CDBL	F1	0ADB	INPUT	89	219A	RETURN	92	1EDE
CHR$	F7	2A1F	INSTR	C5	419D	RIGHT$	F9	2A91
CINT	EF	0A7F	INT	D8	0B37	RND	DE	14C9
CLEAR	B8	1E7A	KILL	AA	4191	RSET	AC	419A
CLOAD*	B9	2C1F	LEFT$	F8	2A61	RUN	8E	1EA3
CLOSE	A6	4185	LEN	F3	2A03	SAVE	AD	41A0
CLS	84	01C9	LET	8C	1F21	SET	83	0135
CMD	85	4173	LINE	9C	41A3	SGN	07	098A
CONT	B3	1DE4	LIST	B4	2B2E	SIN	E2	1547
COS	E1	1541	LLIST	B5	2B29	SQR	CD	13E7
CSAVE	BA	2BF5	LOAD	A7	4188	STEP	CC	2B01
CSNG	F0	0AB1	LOC	EA	4164	STOP	94	1DA9
CVD	E8	415E	LOF	EB	4167	STR$	F4	2836
CVI	E6	4152	LOG	DF	0809	STRING$	C4	2A2F
CVS	E7	4158	LPRINT	AF	2067	SYSTEM*	AE	02B2
DATA	88	1F05	LSET	AB	4197	TAB(	BC	2137
DEF	BD	415B	MEM	C8	27C9	TAN	E3	15A8
DEFDBL	9B	1E09	MERGE	A8	418B	THEN	CA	----
DEFINT	99	1E03	MID$	FA	2A9A	TIME$	C7	4176
DEFSNG	9A	1E06	MKD$	EE	4170	TO	BD	----
DEFSTR	98	1E00	MKI$	EC	416A	TROFF	97	1DF8
DELETE	B6	2BC6	MKS$	ED	416D	TRON	96	1DF8
DIM	8A	2608	NAME	A9	418E	USING	BF	2CBD
EDIT	9D	2E60	NEW	BB	1B49	USR	C1	27FE
ELSE	95	1F07	NEXT	87	22B6	VAL	FF	2AC5
END	80	1DAE	NOT	CB	25C4	VARPTR	C0	24EB
EOF	E9	4161	ON	A1	1F6C	+	CD	249F
ERL	C2	24DD	OPEN	A2	4179	-	CE	2532
ERR	C3	24CF	OR	D3	25F7	*	CF	----
ERROR	9E	1FF4	OUT	A0	2AFB	/	D0	----
EXP	E0	1439	PEEK	E5	2CAA	↑	D1	----
FIELD	A3	417C	POINT	C6	0132	>	D4	----
FIX	F2	0B26	POKE	B1	2CB1	=	D5	----
FN	BE	4155	POS	DC	27F5	<	D6	----
FOR	81	1CA1	PRINT	B2	206F	&	26	4194
FRE	DA	27D4	PUT	A5	4182	' 3A 93	FB	----
GET	A4	4174	RANDOM	86	01D3			

This alphabetic list of functions will help you find ROM routines quickly. The hex addresses refer to memory and the comments by memory address on the following pages.

NOTE: SUPERMAP is designed to be transferred to a disassembled listing of LEVEL II BASIC. Three and five digit numbers are decimals. All 4 digit numbers are hexadecimal.

SUPERMAP by Roger Fuller

0000 Power on routine Turn off clock Zero A Then jump

0008 RST 8H: (HL)-((SP)) SN ERROR if non zero

0010 RST 10H: Increment HL, pass through string ignore CR and spaces. Set C if next character numeric. Reset C if not.

0013 Keyboard routine (see 002B)

0018 RST 18H: HL-DE Z set if equal. C set if DE>HL.

001B Display routine

0020 RST 20: If NTF=8 C is reset else C set. A=NTF-3 S and Z flags valid. Maintains BC, DE, HL.

0028 RST 28H BREAK vector

002B Scan keyboard return with char in A Uses AF, DE

0033 Display byte in A on screen

003B Printer driver entry
0049 Scan keys wait for key pressed Uses AF, DE

KEYBOARD LOOKUP TABLE

0050 (ENTER)
0051 (ENTER) SHIFT
0052 (CLEAR)
0053 (CLEAR) SHIFT
0054 (BREAK)
0055 (BREAK) SHIFT
0056 (UP ARROW)
0057 (UP ARROW) SHIFT
0058 (DOWN ARROW)
0059 (DOWN ARROW) SHIFT
005A (LEFT ARROW)
005B (LEFT ARROW) SHIFT
005C (RIGHT ARROW)
005D (RIGHT ARROW) SHIFT
005E (SPACE)
005F (SPACE) SHIFT

NOTE: SUPERMAP is designed to be transferred to a disassembled listing of LEVEL II BASIC. Three and five digit numbers are decimals.

*See table of tape formats at end of chapter.

0060 Delay loop BC is counter 14.65 microseconds each loop

0066 NMI RESET

0075 Non DOS initialization area move 18F7-191C to 4080-40A6
008B 41E8 to input buffer address pointer (40A7)
0091 Load dummy jump vectors in DOS commands jump addresses. Jump will be to L3 ERROR (012D) instead of DOS. Used during Level II interpretation of BASIC program. Command entry points.
009F Place return commands in DOS link area (these are used by Level II machine routines)
00B2 'MEMORY SIZE' routine Clear screen
00B5 Point to 'MEMORY SIZE' message
00B8 Display it
00BB Wait for user input
00BE If (BREAK) ask again
00C0 Locate 1st char
00C1 Is it anything?
00C2 if so skip memory size routine
00C4 Test for end of actual memory. Used when (ENTER) is given to memory size question
00C7 HL=memory pointer
00CC Get a byte in memory
00CD Save it in B for later
00CE Complement it
00CF Put it back where you got it
00D0 See if memory was there to receive it
00D1 Put back original byte
00D2 Do until memory fails test

. . . .

00D6 Convert input
00DA SN ERROR if not numeric input
00DF Load test byte
00E1 Save current memory byte
00E2 Put in test byte
00E3 Was memory there to receive it?
00E4 Restore memory
00E5 Go back to memory size routine if user was wishful thinking (NOTE: you end up with less memory available even if you exceed actual size by 1)
00E7 Point to end of memory -1
00E8 Load minimum memory size
00EB Check for under size
00EC OM ERROR if under 17430 Then since BC=0000 return in error routine will result in a JP to 0000
00EF Prepare to reduce memory by 050
00F2 Save end of memory
00F5 Reduce by 050 (CLEAR 050)
00F6 Save string space pointer
00F9 Revelation 21:5
00FC Point to 'RADIO SHACK...' message
00FF Display it
0102 On to the farm

. . . .

0105 'MEMORY SIZE' message
0111 'RADIO SHACK LEVEL II BASIC' message

012D L3 ERROR entry point

0132* POINT
0135* SET
0138* RESET

019D* INKEY$

01C9* Clear screen Displays code for CLS

01D3* RANDOM Uses refresh register

01D9 Make a cassette pulse

. . . .

01F8 Turn cassette off
01F9 Bit 2 controls motor 1=on 0=off

. . . .

0212 Define drive 1,0 in A

. . . .

021E Reset the tape input circuit
0221 Out reg A to port FF

. . . .

0234 Blink*

. . . .

0235 Read A byte

. . . .

0241 Read bit (1 or 0)
0243 Wait for timing 'PIP'
0248 Delay
024C Reset input circuit
0251 Delay
0253 Look for 'PIP' if present then bit=1. If not then bit=0

. . . .

0264 Write a byte
0268 8 bits to send
026A Save byte in A
026B Send timing PIP
026E Restore byte
026F Bit to carry
0270 Save byte
0271 Jump if bit is to be a 0
0273 Send PIP for a bit=1
0276 Next bit
0277 Loop til done

. . . .

027E Delay for a bit=0
0284 Turn on cassette
0287 Write leader and sync byte Ready 255 count
0289 Byte to send is 00
028A Send byte
028D Loop til done
028F Sync byte
0291 Send it, return when done

. . . .

0293 Turn on cassette
0296 Find leader and sync byte
0297 Zero A
0298 Get a bit
029B Sync bit yet?
029D No loop til so
029F Get A *
02A1 Send left *
02A4 Send right *

02A9 Get 2 bytes from tape
02AC Save
02AF Turn off cassette
02B2* System RET if non DOS
02B5 Initialize stack in input buffer area
02B8 Return to beginning of line
02BA Get * prompt
02BD Display it
02C0 Wait for input
02C3 Bail out if (BREAK)
02C6 Locate first character
02C7 SN ERROR if dry run (no input)
02CA Check for jump command (/)
02CC If jump
02CE Find leader
02D1 Read a byte
02D4 System header?
02D6 No? Loop back
02D8 File name block length (6)
02DA First character
02DB End of block?
02DC Yes? Blink *
02DE Read a byte
02E1 Correct character?
02E2 Point to next character
02E3 If not correct start over
02E5 Finish a 6 pack
02E7 'Twinkle Twinkle Little *'
02EA Read a byte
02ED Entry point header?
02EF If yes
02F1 Data header?
02F3 If not
02F5 Read byte
02F8 Save # data bytes
02F9 Get load address
02FC Low order byte of address to initialize check sum
02FD Save check sum
02FE Read a byte
0301 Put it in place
0302 Point to next place
0303 Retreive check sum
0304 Update it
0305 Loop til block read
0307 Read check sum
030A Check it
030B Loop back if okay
030D Get a C
030F Display it
0312 Loop back anyway

See SYSTEM and EDITOR/ASSEMBLER tape formats at end of chapter.

....

0314 Get 2 bytes and put in HL
031D Jump routine
0322 Check for an address, evaluate input
0329 Jump to it

032A Output byte to tape, video, or printer (409C)= -1,0,1
032B Save byte
032C Return if non DOS
032F Get device type flag
0333 Restore byte
0335 Write to tape
0338 Write to printer
033A Write to video
0342 Update cursor position (0-3F limit)

0348 Check for double width. (403D)=8 for 32 character line. (403D)=0 for 64 character line.
0355 Reset if cursor position beyond end of line

0361 Keyboard input to buffer. Input routine for keyboard
0362 Reset last key storage
0365 Reset line cursor position
036C Load buffer pointer (normally 41E8)
036F Buffer length=240. Use insert to add more
0371 Return with full buffer
0375 C = input length
0376 BC = input length
0378 HL = end of input length pointer
0379 Terminate input with 00 flag
037B Return pointer to beginning of buffer
0380 Back up for RST 10
0381 If (BREAK)

0384 Called by DOS exit 41AF. Loop til key pressed
038B Return printer carriage to beginning of next line if required
038C Select video output
038F Get printer position
0382 Check for beginning of line
0393 If so return
0394 Get CR
0397 Send it

039C Output to line printer
03AD Get a 0 to reset print position
034A If top of form
03A8 If line feed
03AA Get CR
03AD Send it
03B1 Get printer position
03BA Get character

03C2 Driver entry routine
03C9 Save return address
03CD Put character to display in C
03CE Get device type flag, (DE)= 1,7,6 for keyboard, video, line printer
03CF B=1 keyboard B=2 video or printer
03D1 Non DOS put 0 in A
03DC Jump to appropriate driver routine

03E3 Keyboard driver HL=keyboard buffer pointer
03E6 BC=Row address pointer
03E9 D=Row counter (0 to 6)
03EC Save in E
03EB Get row byte at (BC)
03ED Keep America beautiful
03F0 Go if key pressed
03F2 Bump row counter
03F3 Bump keyboard buffer pointer
03F4 Point to next row
03F6 Go next row til all but (SHIFT) tested
03F9 Return if no key pressed (A=0)
02FA Save row byte
03FB Get row count
03FF Put row count *8 in D
0400 Ready teting mask Bit position = Column #
0402 Load mask
0403 Test for key
0404 Jump if found
0406 Bump column # for test
0407 Move test bit to next column
0409 Do again

0405 Get shift bit
040E Save in B
040F Load row bit * 8+column #
0410 Add 064 to it (this adjusts for ASCII letters)
0412 Test for non letter code
0414 Go if non letter (last 3 rows)
0416 Send shift bit to carry
0418 Skip lower case adjustment if no shift
041A Convert to lower case
041C Save in D
041D Get row 6 bits
0420 Test for down arrow
0422 Go if not down arrow
0424 Retrieve character
0425 Adjust

0429 Test for last row
042B Jump if last row
042D Readjust to rows 4, 5   A=row * 8 + column # + 016
042F Check for = or) or ?
0431 Jump if not
0433 Adjust ASCII
0435 Get shift bit
0437 Jump if not shift
0439 Adjust ASCII
043B Jump

043D A=(row * 8 + column # - 048) * 2
043E Get shift bit
0440 Jump if not
0442 A=column # * 2 + 1
0443 Point to code table
0446 Displacement
0449 Compute position in table
044A Get ASCII code
044B Save character
044C Load delay count
044F Delay
0452 Retreive character
0453 Check for (BREAK)
0455 Leave with character
0457 Return

0458 Video driver
045B Screen cursor position in HL
045E Jump if just entered from 3DC

0467 Get character
046A If control character
046F If graphic or space compression
0473 If not a letter
0479 If upper case
047B Change to upper case
04A1 Cursor to beginning of line

04A6 Check for space compression code
04A8 If graphic
04AA Remove bias
04AF Get space
04B1 Send it
04B4 Loop til decompressed

04B8 Turn on cursor
04BD Turn off cursor
04C0 Home cursor
04C3 Change to 64 character mode

04CE Backspace and erase
04D2 Check for 64 character

....

04DA Backspace cursor

....

04EC TAB

....

04F6 Go double width
04FB Set mode flag
04FE Go double

....

....

0506 Save return address
050C If backspace
050E If no function 2,3,4,5,6,7,9
0513 If turn on cursor
051D If double width
0521 If shift back arrow
0523 If right arrow
0529 If down arrow
052D If up arrow
0531 If home cursor
0535 If beginning of line
053A If erase to end of line
053C If erase to end of frame

....

0554 Scroll display Point to top of display
0557 Point to row 2
055B 15 rows to move
055E Hup one two Hup one two ....
0562 Blank out row 16

058D Printer driver routine
0591 Top of form wanted?
0595 Top of form wanted?
0599 Top of form
059F Get lines per page (4028)
05A2 Subtract current line position (4029)
05A5 Place # remaining lines in B
05A6 Printer ready?
05A9 Wait til so
05AD Send a LF
05B0 Til job is done
05B2 Zero count

....

05B4 Send character out
05B5 Printer ready?
05B8 Wait til so
05BB Send character
05BE CR?
05C0 If not
05C1 Bump line counter
05C7 Page?
05C8 If not
05CC Zero line counter

....

05D1 Printer hand shake. Ready if A=(0011XXXX). Bit 7=0 (not busy).
Bit 6=0 (Paper OK). Bit 5=1 (Device selected).
Bit 4=1 (No Fault). Bits 3,2,1,0 not teted.

05D9 Input routine HL points to input area
05DF C=buffer bytes remaining (240 at start)
05ED Return with key pressed in A
060C Put character in buffer

0613 Print it

. . . .

0619 If clear screen hit, clear buffer. (This is why clearing screen with keyboard erases typed but unentered code)

. . . .

0630 Backup cursor

. . . .

0641 Double width

0674 Initialization Move 06D2-0707 to 4000-4035
0683 Do above 128 times (beat a dead horse)
0689 Zero 4036 to 405D
068B Test for BREAK
0690 If not BREAK
0693 Load new stack
0696 Get disc status
069C If non DOS

. . . .

06A1 Select drive
06A4 Point to controller
06AA Restore
06AF Twiddle thumbs
06B2 Test for READY
06B7 Zero sector register
06BD Read
06C9 Go DOS loader

06CC Entered on NMI RESET (see 0066)

06D2-0707 RST jump addresses, I/O device control block

070B Floating point addition ARITH=(HL) + ARITH

. . . .

0710 Floating point subtraction ARITH=(HL)-ARITH
0713 Floating point subtraction ARITH=BCDE-ARITH
0716 Floating point addition ARITH=BCDE+ARITH
07B2 OV ERROR entry point

0809* LOG ARITH=LOG(ARITH)

NOTE: See 411D to 4124 for ARITH

. . . .

0847 Floating point multiplication ARITH=BCDE*ARITH

. . . .

08A2 Floating point division : ARITH=BCDE/ARITH

. . . .

0955 Check ARITH for Zero

0977* ABS: ARITH=ABS(ARITH). Integer or single in and out.
NTF required and maintained.
0982 ARITH= -ARITH: Single only. Maintains BC, DE

. . . .

098A* SGN accept floating point or integer. Output integer in ARITH

. . . .

0994 Check sign of ARITH, FLOAT or INTEGER. Requires NTF. A=00 if ARITH=0. A=01 if ARITH greater than 0. A=FF if ARITH less than 0. S and Z flags also valid

. . . .

09A4 Load single ARITH to stack. To retrieve POP BC, POP DE. A,BC,HL unaltered.
09B1 Load single: ARITH=(HL)(HL+1)(HL+2)(HL+3)
09B4 Load single: ARITH=BCDE. HL unaltered

09BF Load single: BCDE=ARITH.
09C2 Load single: BCDE=(HL)(HL+1)(HL+2)(HL+3)
09CB Move from (ARITH) to (HL) 4 bytes
09CE Move from (DE) to (HL) 4 bytes

. . . .

09D2 Move from (HL) to (DE) NTF bytes
09D3 Move from (DE) to (HL) NTF bytes
09D6 Move from (DE) to (HL) A bytes
09D7 Move from (DE) to (HL) B bytes
09F4 ARITH=ARITHX + NTF
09FC ARITHX=ARITH + NTF

0A0C Single compare: ARITH-BCDE

. . . .

0A39 Integer compare: HL-DE

. . . .

0A4F Double compare: ARITH-ARITHEX
0A78 Double compare: ARITHEX-ARITH

NOTE: See 4127 to 412E for ARITHEX

0A7F* CINT
0A83 Already integer
0A84 TM ERROR if string
0A87 If double convert to single

0A9A Return to BASIC with output of user routine in HL.
0A9D Flag it integer

0AB1* CSNG

. . . .

0ACC INTEGER ARITH to SINGLE ARITH conversion
0ACF INTEGER HL to SINGLE ARITH conversion

0ADB* CDBL: ARITH(DOUBLE)=ARITH(INTEGER OR SINGLE). Requires NTF

0AF4 TEST NTF=3 (STRING). If string return else error.
BC,DE,HL unaltered.

. . . .

0AF6 TM ERROR entry point

0B26* FIX: If floating point truncate to integer and return floating POINT. If integer return. If string error.

. . . .

0B37* INT
0B38 If integer
0B39 If double
0B3B TM ERROR if string

0C70 DBL precision subtraction: ARITH=ARITH-ARITHEX
0C77 DBL precision addition: ARITH=ARITH+ARITHEX

. . . .

0DA1 DBL precision multiplication: ARITH=ARITH*ARITHEX
0DE5 DBL precision division: ARITH=ARITH/ARITHEX
0E65 Load double precision ASCII constant to ARITH. Point HL to input string delimited by 0 or comma. After load HL points to delimiter
0E6C Load ASCII constant to ARITH. Return the least necessary number type (see Level II manual for rules) Point HL to input string delimited by 0 or comma
0E7B If -
0E80 If +
0E84 If numeric
0E89 If .

0E8E If E
0E92 If %
0E97 If #
0E9C If !

0F40 Multiply HL by ten

0FAB Output 'IN' message

.....

0FAF Output A line #

0FBD ARITH and NTF to ASCII conversion HL points to string

13E7* SQR
1439* EXP

14C9* RND

1541* COS
1547* SIN
15A8* TAN

.....

15BD* ATN

1608 TABLE OF ENTRY POINTS FOR LEVEL II BASIC COMMANDS

1650 RESERVED WORD LIST FOR LEVEL II COMMANDS
1821 End of table marker

See chart at beginning of Chapter

.....

1822 Table of jump addresses for entry points of BASIC instructions.

191D 'ERROR', 'IN', 'READY', 'BREAK' strings for BASIC messages
1930 If (409A)=2 SN ERROR output GOTO edit mode
1955 IQ testing service
197A OM ERROR entry point

.....

199A Divide by zero ERROR entry point
199D NF ERROR entry point
19A0 RW ERROR entry point
19A2 Error output routine. Error code in E
19E6 Return to beginning of line. Zero A
19E9 Point HL to bottom of error message table
19EF Non DOS return
19EF Zero D
19F0 Get A '?'
19F2 Display it
19F5 Add error code displacement to pointer
19F6 Get error message
19F7 Display 1st charcter
19FA Numeric check
19FB Display 2nd character
19FE Point to 'ERROR'
1A06 Display it
1A0D If DE=(40EA)
1A0E Then power up RESET
1A14 Display 'IN' line #
1A17 Load A TAB(1)
1A19 Return to BASIC command mode ('READY' routine) Print A
1A1C Return if non DOS
1A1F Turn off cassette
1A22 Return to beginning of line or CR. Zero A

1A25	Point to 'READY' message
1A28	Output it
1A2B	Get error code
1A2E	Test for SN ERROR
1A30	Call if SN ERROR
1A33	Return address
1A39	Get AUTO flag (AUTO=non zero)
1A3D	Jump for non AUTO

1A3F	Get current line #
1A43	Output line #
1A48	See if line # occupied (Carry set) Read to display if match
1A4B	Get an asterisk
1A4D	Print a space or an asterisk
1A4F	Get a space otherwise
1A51	Display correct character
1A54	Input into buffer
1A58	(BREAK) sets carry

1A5A	Turn off AUTO and jump back

1A60	Get line increment
1A63	Add to current line #
1A69	(BREAK) if oversize line # results
1A6C	(BREAK) if line # > 65529X

1A76	Get prompt
1A78	Display it
1A7B	Input into buffer
1A7E	Jump back if BREAK
1A81	Find first character
1A84	Jump back if null
1A88	Check for numeric then scan past line # (line # is in DE)
1A8B	Scan
1A99	Encode input into Level II tokens
1A9D	Flags decide if command mode
1A9E	Encoded statement pointer
1AA1	Non DOS return
1AA4	If command mode?
1AA7	Save line #
1AA8	Save line length
1AA9	Zero
1AAA	Reset resume + return flag
1AAD	Scan 1st token
1AB1	Save line #
1ABF	Save the line #
1AB5	Search for a matching line # C=none Z=found
1AB9	If none make room
1ABF	Jump for match
1AC6	Line length
1AC7	HL=new end of BASIC program
1AC9	Make sure brain won't overflow
1ACD	Store end of BASIC program pointer
1AD0	HL=line to be moved
1AD3	HL=line # pointer
1AD6	DE=line #
1AD9	HL=line pointer (text)
1AE1	Move new line into place
1AE6	Loop til line is moved
1AE9	Fix line pointers
1AEC	Non DOS return
1AEF	Here's why editing a program destroys variables, etc.
1AF2	Non DOS return
1AF5	Back to the farm

1AFC	Fix the line pointers routine
1B01	Return if end of BASIC
1B02	Move

1B03 Past
1B04 The next line pointer and line #
1B06 Check for end of line
1B08 Don't quit til you succeed
1B0A Get the job done
1B0E Rerun Roadrunner cartoon

1B2C Search for matching line # in BASIC. DE=desired line #. Get first line from (40A4)
1B2F Save pointer in BC
1B31 Check for end of BASIC program (stay out of junkyards)
1B35 Return if end
1B36 Point to current line #
1B3B HL=current line #
1B3C Match? Z=yes C=HL<DE
1B3D Next line pointer to HL
1B44 Return if match found (carry set)
1B46 Return (no such line)
1B47 Try next line #

1B49* NEW
1B4A Clear screen
1B4D Start of BASIC program
1B50 TROFF
1B53 Turn off AUTO
1B56 Erase program by making its leaders zilch
1B5A Reset end of program pointer
1B64 26 variables
1B6C Set to single precision here
1B6F Reset resume flag
1B74 Reset on error storage
1B77 Reset CONT location
1B7A Get End of Memory

1B88 Restore DATA pointer
1B83 Get end of BASIC location
1B86 Reset variables pointer
1B89 Reset arrays pointer
1B90 Get start of string space pointer
1B95 Set stack pointer to start of string space - 2
1B9A SP=string space pointer
1BA1 Select video Finish printing
1BA4 Turn off cassette

1BB3 Print '?' and input from keyboard Go on CR

1BC0 Encode buffer into tokens
1BC1 Reset flag
1BC6 HL=input buffer pointer
1BCC Get 1st character from buffer
1BCD If space
1BD3 If string
1BD9 If end of line
1BE0 Get buffer character
1BE4 Check for print abbreviation
1BE6 Get print token
1BE8 Substitute
1BEB Get character
1BEE If non alpha numeric
1BF2 If numeric
1BF5 Save input pointer-2
1BF6 Load reserved word list pointer-1
1BF9 Save line length
1BFD Save continuation address
1BFE End of reserved word list test mask
1C00 Get character

The LEVEL II ROM does not use the alternate registers.

1C03 If not lowercase
1C07 If not uppercase
1C09 Convert to uppercase
1C0C Save character
1C0E Point to reserved word list
1C0F Check for beginning of word (CHAR+80)
1C10 Try again if not
1C13 Bump count
1C14 Get reserved word character
1C15 Check for end of list
1C17 Continuation if word not found
1C18 Is it same as buffer character?
1C19 Next reserved word if not
1C27 If not GOTO
1C31 Make upper case if needed
1C37 Next character

....

1C46 If not ELSE
1C4C If not '
1C4E Load colon
1C50 Next
1C53 Load REM (1 for the price of 3)

1C90 RST 18H code

1C96 RST 08H code

1CA1* FOR
1CFB Check for 'STEP' token
1CFD Default value of 1
1CFF If not 'STEP'
1D4A '(LINE NUMBER)' TRON usage

....

1D5A BASIC interpreter
1D60 Remove bias
1D62 Check for a token
1D62 Jump if not
1D6A Double remainder (Required for 2 byte addresses)
1D6B Save offset
1D6C IN B
1D6F Point to vectors
1D72 Locate desired routine address
1D73 Low byte to C
1D75 High byte to B
1D76 Save on stack

....

1D78 RST 16

1D91* RESTORE
1D92 Get beginning of program
1D96 Restore DATA pointer

1D9B Display line number

1DA0 Check for pause
1DA2 Wait for keystroke to resume
1DA5 Save it
1DA8 01=BREAK
1DA9* STOP
1DD4 Select video
1DD7 Return to beginning of line

1DAE* END

1DE4* CONT
1DEB Output CN ERROR if (40F7)=0

1DF7* TRON AF=TRON
1DF8* TROFF

1E00* DEFSTR
1E03* DEFINT
1E06* DEFSNG
1E09* DEFDBL
1E0B Check for syntax (Letter needed in DEF---)
1E0E Get address of SN ERROR
1E11 Save on stack for possible use
1E12 SN ERROR if no letter
1E13 Convert ASCII letter to displacement into table of 26 letters
1E15 Save displacement in C
1E16 Save displacement in B
1E17 Get next character
1E18 Is it —
1E1A If not don't use a range
1E1D Check for letter
1E20 SN ERROR if not
1E21 Get displacement
1E23 Put in B
1E24 Get to next character
1E25 Load ending point
1E26 Subtract beginning point
1E27 SN ERROR if variables reversed
1E28 Bump count (in case variables same)
1E29 Save next character pointer and clear SN ERROR vector
1E2A Load start of variable definition area
1E2D Zero B
1E2F Determine ending point
1E30 Set variable type flag
1E31 Bump table pointer
1E32 Reduce count
1E33 Loop til count zero
1E35 Return next character pointer
1E36 Get next character
1E37 Is it a comma?
1E39 If not
1E3A Get next variable
1E3B DEF--- again

....

1E3D Check for letter in (HL). Set C if not else reset

1E4F Get character
1E50 Is it a period?
1E53 Get period address
1E57 Jump if period
1E5A For RST 10
1E5B Initialize DE DE=line # on exit
1E5E Locate 1st character and numeric check
1E5F Return if non numeric
1E60 Save location
1E61 Save ASCII numeric digit
1E62 Oversize limit (65520)
1E65 Pre-flight
1E66 SN ERROR if DE > 1998
1E69 HL=DE
1E6B HL=(HL + DE)
1E6C HL=(HL + DE) + (HL + DE)
1E6D HL=(HL + DE + HL + DE) + DE
1E6E HL=(HL+DE+HL+DE+DE) + (HL+DE+HL+DE+DE)=4*HL + 6*DE=010*DE
1E6F Retrieve ASCII numeric digit
1E70 Convert ASCII code to #
1E72 Save # in E
1E73 Zero D so DE=#
1E75 Add # to subtotal (HL)

1E76 DE=010*DE + #
1E77 Restore pointer

1E7A* CLEAR
1E7D Compute the amount as an integer
1E84 END OF MEM pointer
1E8D OM ERROR if HL< DE
1E9C Load string pointer

1EA3* RUN
1EB1* GOSUB

1EC2* GOTO Evaluate line #
1ED9 UL ERROR entry point

1EDE* RETURN
1EEC RG ERROR entry point
1F05* DATA
1F07* REM
* ELSE

1F21* LET

1F6C* ON
1F70* ON ERROR
1F80 If UL ERROR
1F89 Get error flag
1F8E Get error code
1F91 Load into E for error routine
1F92 Jump

1FAF* RESUME Point to error flag
1FB3 Check it
1FB4 RW ERROR if zero

1FF4* ERROR
2003 UE ERROR entry point

2008* AUTO
200B Save default value of 10
2019 Save line # increment
2022 If SN ERROR
2025 Check for zero increment
2028 FC ERROR if Z
202B Save increment
202E Set AUTO flag
2036 Back to the farm

2039* IF
2044 IF THEN
2060 IF NOT ELSE

2067* LPRINT Select line printer for output
207B If expression isn't integer
207F Point to display
2082 Compute location
2083 Save it
2089 Update cursor

....

206F* PRINT
2076* PRINT @ Evaluate expression
2093* PRINT # Write leader and sync byte
20A5 If PRINT USING

20AA If PRINT TAB (
20B0 If comma
20B5 If semicolon
20BE If string
20FE Used to return to beginning of line and zero A
2137* TAB (

2169 Output device T,V,P,-1,0,1

....

216D Turn off cassette if needed
2171 Select video

....

2178 'REDO' message string

219A* INPUT
219D REDO
21A9* INPUT #
21AD 250 bytes limit
21AF POINT
21B2 Read a byte
21B5 Into the buffer
21B7 CR yet?
21BB Loop back
21BE End of file marker
21C0 Turn off tape

....

21EF* READ

....

227C 'EXTRA IGNORED' string

....

22B6* NEXT

2337 Evaluate expression Put in ARITH : Point HL to address of 1st character Terminate with 00 or , or) or :

....

2490 Integer divide. Output in single precision

....

249F* +
24A2 MO ERROR if Z
24A5 IF numeric
24A8 Check for letter
24AB IF letter
24B0 IF +
24B4 IF .
24B9 IF -
24BE IF QUOTE
24C3 IF NOT
24C8 IF &

....

24CD IF NOT ERR
24CF* ERR

....

24DB IF NOT ERL
24DD* ERL

....

24E9 IF NOT VARPTR
24EB* VARPTR

....

2501 IF USR
2506 IF INSTR
250B IF MEM

2510 IF TIME$
2515 IF POINT
251A IF INKEY$
251F IF STRING$
2524 IF FN

2532* -

2540 ASCII variable to ARITH : Put variable in ARITH and set NTF. Point HL to 1st character Returns with HL pointing to next character after variable.

25D9 RST 20H
25F7* OR ....

25FD* AND

2608* DIM
260D Locate or create if not found variables: Point HL to 1st character of variable. Returns with DE pointing to variable's address and HL pointing to the next character after variable.
2612 Check for letter
2615 SN ERROR if C
261B If numeric
2620 If not a letter
2624 If numeric
2626 Check for letter
2629 If numeric
262E Set return address to 2652 before going
2633 If % (INTEGER D=2)
2637 If $ (STRING D=3)
263B If ! (SINGLE D=4)
2640 If # (DOUBLE D=8)
273D BS ERROR entry point

27C9* MEM
27CB Zero NTF
27CE Call FRE routine

27D4* FRE Get free space pointer
27DD If not a string
27E5 Get start of string space
27E9 Get end of string space
27F1 HL=END - START

27F5* POS Get cursor position
27FB Return via USR reentry HL is position

27FE* USR
2801 Next character
2806 Set reentry point
280A Get NTF
2810 Call AF string
2815 (408E) contains entry point to USR routine

....

2831 ID ERROR entry point

2836* STR$
2866 QUOTE
2891 NTF = string
28A1 ST ERROR entry point

28A7 Output a message; Point HL to starting address of string; Mark end with a 00 or 22 Output device selected by (409C). Updates line cursor position

---

2A03* LEN

2A0F* ASC

2A1F* CHR$

2A2F* STRING$

2A61* LEFT$

2A91* RIGHT$

2A9A* MID$

2AC5* VAL

2AEF* INP Get port addresses
2AF2 Load it
2AF5 Input from correct port
2AF8 Return via USR code

2AFB* OUT Get byte
2AFE Output it

....

2B01* Step
2B02 Compute value of expression
2B06 Convert it to integer
2B0B MSB to A
2B0C Check for overflow

....

2B0E Get port #
2B11 Set port # for input
2B14 Set port # for output
2B17 Syntax check
2B1C Compute value
2B1F Convert to integer
2B22 FC ERROR if overflow

2B29* LLIST (same as LIST only output device is line printer)
2B2E* LIST
2B2F Get first line pointer
2B32 Save it
2B38 BC=Next line pointer
2B40 Check for end of BASIC program
2B41 Back to the farm when the chores are done
2B44 DOS link
2B5E Output a line #
2B61 Get a space
2B64 Output it
2B67 Call Mr. Spock for his opinion of this message
2B6A Get buffer pointer
2B6D Spock's interpretation
2B70 CR+LF if needed
2B73 Next line

....

2B75 Get character
2B76 End of text
2B77 If so
2B78 Output character to correct device
2B7B Next character
2B7C Loop til done

2B7E Convert line to human readable form
2B7F Buffer pointer to HL
2B85 Line limit

2B8C Get first character
2B8D Is it end of text marker?
2B8E Point to the next character
2B8F Save in (BC)
2B90 If end of text
2B91 If not a token
2B96 If not REM
2BA5 Remove bias on token
2BA8 Load token's word position in BASIC reserved word list
2BA9 Get pointer to reserved word table
2BAC Get a byte
2BAD Check for beginning of a word
2BAE Point to next character
2BAF Loop til correct word found
2BB2 Drop count
2BB3 Loop back if not correct word
2BB5 Convert to upper case
2BB7 Put character in buffer
2BB8 Bump pointer
2BB9 Drop limit
2BBA Jump if end of line
2BBD Get next character of word
2BBE Point to next character
2BBF Check for end of word
2BC0 Loop back if not end of word
2BC3 Restore buffer pointer
2BC4 Next word

2BC6* DELETE

2BF5* CSAVE Write leader and sync byte
2BF8 Evaluate character following CSAVE
2BFF D3=Header for BASIC tape
2C04 Write once
2C04 Write twice
2C07 Send file name to tape
2C0B Start of program
2C0F End of program
2C12 Get a byte
2C13 Point to next
2C14 Send byte
2C17 Done?
3C18 Loop til done
2C1A Off cassette

2C1F* CLOAD
2C40 TROFF + NEW
2C47 3 count for header
2C49 Get a byte
2C4C D3?
2C4E If not reset count and try again
2C50 Loop til header found
2C52 Get file name (1 byte)
2C57 If only CLOAD
2C59 Is tape file name correct?
2C5A Go bad if not
2C5C Beginning of BASIC ID HL
2C5F Load count for 3 zeros which determine end of BASIC tape
2C61 Read a byte
2C64 Save it for later
2C65 Same as original?
2C66 Same location?
2C67 NZ means bad
2C69 Put byte into memory note: Validity is checked before placement

See BASIC tape format at end of this chapter.

2C6A Check for OM ERROR
2C6D Retrieve byte
2C6E Check for a zero
2C6F Point to next memory location
2C70 If not a zero (end of line) Resets zero count
2C72 Twinkle Twinkle (end of line)
2C75 Loop til end of program (3 zeros)
2C77 Update end of program pointer
2C74 Point to 'READY' message
2C7D Display it
2C80 Turn off cassette
2C83 Beginning of BASIC to HL
2C86 Save it
2C87 Return to farm after fixing addresses

. . . .

2C8A Point to 'BAD' message
2C93 Give the news
2C90 Back to the farm
2C93 Display file letter
2C96 3 count for end of program test
2C98 Read byte
2C9B Zero?
2C9C If not reset count try again
2C9E Loop til 3 zeros
2CA0 Search for next program
2CA3 Try next program

. . . .

2CAA* PEEK Evaluate expression as integer
2CAD Get byte
2CAE Return via USR code

2CA5 'BAD' message string

2CB1* POKE Compute address
2CB4 Save it
2CB5 Check syntax
2CB6 Next
2CB7 Compute value of operand
2CBA Retrieve address
2CBB POKE it in

2CBD* USING

. . . .

2E49 Print a plus if D non zero

2E60* EDIT Get line
2E64 DE=line #
2E66 Put line # in storage
2E6A Search for matching line #
2E6D UL ERROR if not found (NC)
2E70 Address of line to HL
2E72 Point to line #
2E74 Put line # in BC
2E78 Save line # on stack
2E79 Convert line to ASCII
2E7E Output line #
2E81 Output blank
2E86 Point to buffer
2E89 Send cursor
2E8E Save buffer pointer
2E91 C=length of line
2E95 Search for end of line
2E98 Zero A
2E99 Zero D
2E9B Get a key

. . . .

2EBB (BACK SPACE)
2EC0 (ENTER)?
2EC7 (SPACE)?
2EC9 Upper case?
2ECB Change to lowercase
2ECF Q uit?
2ED4 L ist line?
2ED9 S earch?
2EDD I nsert?
2EE2 D elete?
2EE7 C hange?
2EEC E nd?
2EF1 X tra?
2EF6 K ill?
2EFA H ack?
2EFF A gain?
2F01 Not valid character if NZ get another
2F07 Start over

....

2F0A Space routine

....

2F16 KILL routine
2F1C Search routine
2F1D Get the character

....

2F40 Output area pointed by HL

....

2F4A DELETE routine
2F4D Print '!'
2F5F Print '!'

....

2F65 Change routine
2F68 Get character
2F6C Print it
2F71 Reduce count
2F72 Loop til done

....

2F75 HACK routine
2F78 XTRA routine
2F7D Insert routine
2F84 Back space
2F88 CR?
2F8D Escape?

....

2FE0 Leave edit mode after displaying line

....

2FF6 Quit routine

3000- Reserved there is nothing here no memory at all

37DE DOS communication status address
37DF DOS communication data address
37E0 Interrupt latch address
37E1 Disk drive select latch address
37E2 Cassette drive latch address
37E8 Line printer port address
37EC Floppy disk controller address

KEYBOARD MEMORY

	COL # =	0	1	2	3	4	5	6	7
3801	ROW 0	@	A	B	C	D	E	F	G
3802	ROW 1	H	I	J	K	L	M	N	O
3804	ROW 2	P	Q	R	S	T	U	V	W
3808	ROW 3	X	Y	Z					
3810	ROW 4	 0	! 1	" 2	# 3	$ 4	% 5	& 6	' 7
3820	ROW 5	(8	) 9	* :	+ ;	< ,	= -	> .	? /
3840	ROW 6	ENT	CLS	BRK	↑	↓	←	→	SPC
3880	ROW 7	SHIFT							

3C00-
3FFF Video memory

4000 RST 8
4003 RST 10
4006 RST 24
400C RST 32
400D RST 40
400F RST 48
4012 RST 56
---------KEYBOARD CONTROL BLOCK
4015 Device type
4016 Driver address (intercept here for debounce)
4018 0
4019 0
401A 0
401B 'K'
401C 'I'
---------VIDEO CONTROL BLOCK
401D Device type
401E Driver address
4020 Cursor position in memory (2 bytes)
4022 Cursor character
4023 'D'
4024 'O'
---------LPRINTER CONTROL BLOCK
4025 Device type
4026 Driver address
4028 # lines per page kept here
4029 Current line # printer is on
402A 0
402B 'P'
402C 'R'

....

4036-
403C 7 byte work area for keyboard routine

....

403D Print size flag 0=64 characters 8=32 characters Also used in tape output to prevent resetting size during an OUT 255
403E-
407F Not used in Level II non DOS (good place for debounce routine)
---------TIME$ STORAGE AREA
4040 25 MS ticks
4041 Seconds
4042 Minutes
4043 Hours
4044 Year
4045 Day
4046 Month
4080-
408D Calculate remainder in floating point division
408E Entry pointer to USR routines
4090 - Random number generator secondary seed

4092	
4093	INP routine
4094	Port #
4096	OUT routine
4097	Port #
4099	INKEY$ storage (and SHIFT @ Pause release key)
409A	Error code storage for RESUME use
409B	Printer line width counter (for pretty printing!)
409C	Device type flag -1=TAPE 0=VIDEO 1=LPRINTER
409D	Print # use
40A0	Start of string space pointer
40A2	Current line being processed
40A4	Start of BASIC program pointer
40A6	Line cursor position used for tab
40A7	Input buffer pointer
40AA	LSB of seed for RND
40AB	LSB of seed for RND Also used in RANDOM
40AC	MSB of seed for RND
40AE	Flag byte for DIM statement
40AF	NTF (number type flag) 2=INTEGER 3=STRING 4=SINGLE 8=DOUBLE
40B0	Compressor flag for 1BC0
40B1	Top of BASIC memory pointer
40B3	String work area pointer
40B5	String work area
40D3	String length
40D4	Start address of string/next string address
40D6	Memory size
40D8	Comma control matrix for PRINT USING
40DC	DIM use
40DE	PRINT USING
40DF	Entry point storage for SYSTEM tapes
40E1	AUTO flag 0=not AUTO Else AUTO
40E2	Auto increment
40E4	Auto line number
40E6	Encoded statement pointer
40E8	Stack pointer pointer
40EA	Line number of error in RESUME
40EC	Line number for edit and list when you use period instead of number
40EE	Used during RESUME
40F5	Last line # executed
40F7	Used to CONT
40F9	Simple variables pointer
40FB	Arrays pointer
40FD	Free space
40FF	Data pointer
---------	VARIABLE TYPE DECLARATION TABLE
4101-410A	2=INTEGER 4=SINGLE 8=DOUBLE 3=STRING
411B	TRON FLAG 0=TROFF

NOTES ON THE EDITOR ASSEMBLER

4113	Top of memory pointer
4115	Start of buffer pointer
41C3	Start of symbol table pointer
4301	Keyboard driver entry address pointer
4309	Video driver address pointer
4311	Lprinter driver address pointer
45AA	Lprinter driver (patch your printer here)
4905	Command table (have a feast)
4925	B command (go someplace new)

ARITH

	INTEGER	SINGLE	DOUBLE
411D			LSB
411E			LSB
411F			LSB
4120			LSB
4121	LSB	LSB	LSB
4122	MSB	LSB	LSB
4123		MSB	MSB
4124		EXP	EXP

ARITHEX

4127	LSB	LSB	LSB
4128	MSB	LSB	LSB
4129	LSB	LSB	LSB
412A		EXP	LSB
412B			LSB
412C			LSB
412D			MSB
412E			EXP

4125 Sign Byte
4412F Sign Byte
4130 Line # work area pointer

---------DOS ENTRY POINTS

414A- 4151 Additional section of RAM like ARITH and ARITHEX for division and trig functions. Does not have a sign byte

4152 CVI
4155 FN
4158 CVS
415B DEF
415E CVD
4161 EOF
4164 LOC
4167 LOF
416A MKI$
416D MKS$
4170 MKD$
4173 CMD
4176 TIME$
4179 OPEN
417C FIELD
417F GET
4182 PUT
4185 CLOSE
4188 LOAD
418B MERGE
418E NAME

4191 KILL
4194 &
4197 LSET
419A RSET
419D INSTR
41A0 SAVE
41A3 LINE

Level II and Disk BASIC relays from ROM routines (this is the area to modify to intercept BASIC routines)

41A6 Error code index relay
41A9 USR relay
41AC Command mode at initialization
41AF Fill BASIC keyboard buffer relay
41B2 Command mode relay at EDIT/DIRECT switch
41B5 Command mode relay at termination before initialize
41B8 Command mode relay before restart
41BB NEW and END relay
418E Relay to change default in routine to transfer control from OUTPUT (032A) back to device held at 409C
41C1 OUTPUT relay device transfer
41C4 Keyboard routine relay
41C7 RUN relay
41CA PRINT relay
41CD secondary PRINT relay
41D0 Relay for carriage return output (add your linefeed this way!)
41D3 PRINT TAB (relay)
41D6 INPUT relay
41D9 MID$ relay
41DC Secondary READ relay
41DF LIST relay (disable LIST here!)
41E2 SYSTEM relay (put E9 here and get AUTO start!)

....

---------INPUT BUFFER AREA---

I/O Buffer

41E5 Buffer control bytes
41E8-42E7 Keyboard Buffer (Level II only)
42E8 Separator byte for Level II

....

42E9 BASICally only the beginning!

TAPE FORMATS

BASIC TAPE FORMAT

LEADER	256 zeros followed by an A5 sync byte
D3 D3 D3	BASIC header
XX	File name
LSB	Next line's address
MSB	Pointer
LSB	Line number
MSB	
XX ... XX	Line contents
00	End of line marker
.	
.	
.	
00 00	End of file markers

SYSTEM TAPE FORMAT

LEADER	256 zeros followed by a A5 sync byte
55	System format header byte
XX XX XX XX XX XX	6 character file name
3C	Data header
XX	Data length 00=256 bytes
LSB	Loading
MSB	Address
XX ... XX	Line itself
XX	Checksum of line bytes and load address
.	
.	
.	
78	End of file marker
LSB	Entry
MSB	Address

EDITOR ASSEMBLER SOURCE TAPE FORMAT

LEADER	256 zeros followed by an A5 sync byte
D3	Source header
XX XX XX XX XX XX	File name
#1 #2 #3 #4 #5	Line # in ASCII (bit 7 is set)
20	Data header
XX ... XX	Line (128 bytes maximum)
OD	End of line marker
.	
.	
.	
1A	End of file marker

BASIC RAM STORAGE FORMAT

LSB	Address of
MSB	next line
LSB	Line # in
MSB	binary form
XX ... XX	Line contents
00	End of line marker
.	
.	
.	
00 00	End of file marker

Chapter 11

HEX MEM by John T. Phillipp, M.D.

If you are seriously investigating Level II ROM routines, you will need a way to examine memory. This particular monitor is quite limited, as it will not save a machine language program, set breakpoints, execute, or display and access the Z-80 registers. If you are using a disk system, you will find DEBUG more useful. In a tape system, we recommend STAD from **The Software Exchange**. However, this monitor will provide several memory examination capabilities in the absence of a more sophisticated monitor.

HEXMEM is a BASIC program which duplicates some of the functions of machine language monitors like TRS-DOS DEBUG and the RSM-2 monitor by Small Systems Software. Although it doesn't support all of the functions of these sophisticated monitors (it does not access the Z-80 registers, for example), it does enable the user to convert hexadecimal numbers, display memory in hex or ASCII in a format similar to the machine language monitors, modify memory, enter machine language programs directly, load other BASIC programs into memory with HEXMEM and save HEXMEM and other BASIC programs on tape.

The code for HEXMEM takes 2,399 bytes, leaving 13,173 bytes for the other programs in a Level II, 16K TRS-80. It is densely packed into line numbers 1-28 without REM statements. This makes the program logic hard to follow, but was necessary since any program loaded into memory with HEXMEM must have line numbers greater than HEXMEM itself. HEXMEM can reside in memory with any BASIC program whose line numbers begin with line 30 or higher.

Commands — all commands are single letters. When HEXMEM asks COMMAND? type the command letter (ENTER). HEXMEM will return to COMMAND? after the command has been executed.

SUMMARY OF COMMANDS

H—	Hexadecimal	-Converts hex to decimal value
D—	Decimal	-Converts decimal to hexadecimal value
G—	Graphics	-Converts any hex or decimal value to ASCII character
M—	Memory Dump	-Displays block of memory as hex values
A—	ASCII Dump	-Displays block of memory as ASCII characters
E—	Edit	-Displays and edits the contents of a memory location
O—	Object Code	-Enters machine language programs
L—	Load	-CLOADs a BASIC program into memory with HEXMEM
C—	Combine	-MERGES a BASIC program on tape with one in memory
S—	Save	—CSAVEs a BASIC program and HEXMEN on tape

EXPLANATION OF COMMANDS

H — Hexadecimal
This routine will convert any hexadecimal number up to 10 digits to its decimal equivalent, although very large numbers will be displayed in decimal by exponential notation. Leading zeroes need not be entered 0AF2 and AF2 will be converted the same way. Type the hexadecimal number (ENTER).

D — Decimal
This routine will convert any decimal number from 0 to 65,535to its hexadecimal equivalent. Larger numbers will not be converted as the routine will not produce a hexadecimal number more than 4 digits in length. Type the decimal number (ENTER).

Addition and subtraction of hexadecimal numbers may be done using the H and D commands providing the difference is not less than 0 nor the sum greater than FFFF hex (65,535 decimal). Use the H command to convert both numbers to decimal, perform the desired operation, then use the D command to convert the result to hexadecimal.

G — Graphics
This routine will convert a hexadecimal number in the range 0 to FF or a decimal number in the range 0 to 255 to its ASCII character, graphics character, or space compression code representation.

Hexadecimal numbers need only be typed, then (ENTER) but decimal numbers **must** be followed by an X (enter as 127X, for example). If the X is omitted, HEXMEM will consider the number to be hexadecimal and an erroneous conversion will result.

M — Memory Dump (HEX)
see also A — ASCII Dump
This routine will display the contents of any block of memory in hexadecimal.

HEXMEM asks for ADDRESS (HEX)#1? Type any hexadecimal address from 0 to the top of memory (4FFF for a 4K machine, 7FFF for a 16K machine) then (ENTER).

Use the D command to convert decimal addresses into hex for the M command.

HEXMEM then asks for ADDRESS (HEX)#2? Type a hexadecimal address which is larger than ADDRESS (HEX)#1. If no ADDRESS (HEX)#2 is typed, and only (ENTER) is pressed, HEXMEM will display from ADDRESS (HEX)#1 to the top of memory.

NOTE: HEXMEM is programmed for a 16K machine. If you have more or less memory line 8 in the program must be changed:

If H2$ = "" THEN D6 =

```
32767(16K)
49151(32K)
65545(48K)
20479( 4K)
```

HEXMEM will display memory in lines of 16 bytes each. The address of each line is on the far left of the screen. For example, if the starting address is 4FB3 HEX the screen will display:

```
4FBO:  XX XX XX XX XX XX XX XX
       XX XX XX XX XX XX XX XX
4FCO:  XX XX XX XX XX XX XX XX
       XX XX XX XX XX XX XX XX
```

and so on, where XX is any hexadecimal number from 00 to FF. The address of the first byte of each line will always end in 0.

To stop the dump at any point, press any key on the keyboard. To continue the dump, press any key except K. Pressing K will return to COMMAND?

A — ASCII Dump
This routine will display the contents of any block of memory as its ASCII equivalents. Control characters and graphics characters are displayed as periods (.). (The ASCII equivalent of any HEX or decimal digit (00-255) can be displayed using the G command.)

The ADDRESS (HEX)? prompts should be answered in the same manner as for the M command, and the memory display is in the same 16 byte line format.

The A command is particularly useful for searching the memory for a BASIC program. Try displaying addresses

1600 to 1700 HEX using this command. This is the area where the Level II BASIC ROM stores its command table. HEXMEM itself resides in addresses 42EA to 4C48 HEX (17130 to 19528 decimal) and can be displayed by the A command.

E — Edit Memory
see also O - Object Code Enter

This routine is used to modify a single memory location by replacing its current value with a new one.

HEXMEM asks ADDRESS TO CHANGE? Type in any Hex or decimal address from 0 to the top of memory, then (ENTER). Decimal numbers must be followed by an x, or HEXMEM will consider them to be hexadecimal and errors will result.

The current contents of that memory location will be displayed in Hex and decimal.

HEXMEM will then ask for the NEW VALUE? Type in the desired value from 0 to FF Hex or 0-25 decimal. Again, decimal numbers must be followed by an X. Then press (ENTER). The contents of the memory location will be erased, the new value will be entered and HEXMEM will return to COMMAND?

Pressing (ENTER) after the NEW VALUE? prompt leaves the contents unchanged and returns to COMMAND? This may be used to PEEK at one memory location rather than using the M command, especially if you want to know the address contents in decimal.

Addresses from 0 to 3000 HEX (0 to 1288 decimal) comprise the Level II ROM and cannot be changed by the E command. Addresses from 3001 to 42E8 HEX (12290 to 17128 decimal) are RAM and may be changed although they are used by Level II BASIC for housekeeping and Input and Output.

Changing them may cause the system to crash. Try it though — there is no danger to the TRS-80 hardware from the keyboard. At worst, you may have to press RESET and re-load HEXMEM.

O — Object Code Enter

This routine is similar to the E command and is used for modifying a consecutive series of memory locations as when entering a machine language (object code) program.

After the starting address is entered and its contents changed with a new HEX (or decimal plus X) value, HEXMEM advances to the next memory location, displays its value in HEX and decimal and accepts the new value. Pressing (ENTER) with no value leaves the memory location unchanged and advances to the next.

Typing K for the value leaves the memory location unchanged and returns to COMMAND?

L — Load BASIC Program

This routine allows a BASIC program to be CLOADed into memory co-resident with the HEXMEM monitor.

In order to CLOAD a BASIC program without erasing the program in memory (HEXMEM), some PEEKing and POKEing is necessary. HEXMEM does most of the work. However, after the CLOAD is complete, two more memory locations must be POKEd. These values — POKE 16548, 233 : POKE 16549,66 — should be typed in and (ENTER) pressed. The combined program (HEXMEM + BASIC program) may then be LISTed, RUN, CSAVEd (with the S command), EDITed, etc.

The line numbers of the BASIC program must be higher than those of HEXMEM, or HEXMEM will be erased during the CLOAD HEXMEM uses line numbers 1-29 so the BASIC program should start at line 30 or higher.

C — Combine BASIC Programs

This routine will MERGE a BASIC program on tape with one in memory. As with the L command, the program on tape must have higher line numbers than the one in memory, or the program in memory will be erased during the CLOAD.

The same two memory locations must be POKEd after the CLOAD as with the L command. Follow the prompts on the screen.

The C command will enable frequently used subroutines to be stored on tape and then added to new main programs as needed, saving the trouble of retyping them.

After the programs are combined in memory, type DELETE 1-29 (ENTER) to erease HEXMEM. The combined program may then be LISTed, RUN or CSAVEd on tape.

S — Save

This routine will CSAVE HEXMEM and any other BASIC program in memory on cassette tape.

This is an example of how BASIC stores program text. The ASCII dump and the HEX dump are looking at the same block of memory — address.

```
100 REM * THIS IS LINE 100
110 ' LINE 110 - ABBREVIATION FOR REM *
120 PRINT "THIS IS LINE 120 ? "
     REM
```

```
COMMAND? M
ADDRESS (HEX) #1? 7210
ADDRESS (HEX) #2? 725F
7210:   41 31 24 3A BA 22 41 22    00 32 72 64 00 93 20 2A
7220:   20 54 48 49 53 20 49 53    20 4C 49 4E 45 20 31 30
7230:   30 00 5C 72 6E 00 3A 93    FB 20 4C 49 4E 45 20 31
7240:   31 30 20 2D 20 41 42 42    52 45 56 49 41 54 49 4F
7250:   4E 20 46 4F 52 20 52 45    4D 20 2A 00 7F 72 78 00
COMMAND?        << HEX DUMP  (M COMMAND) >>_
```

```
COMMAND? A
ADDRESS (HEX) #1? 7210
ADDRESS (HEX) #2? 725F
7210:    A  1  $  :  .  "  A  "     .  2  R  D  .  .     *
7220:       T  H  I  S     I  S        L  I  N  E     1  0
7230:    0  .  \  R  N  .  :  .     .     L  I  N  E     1
7240:    1  0     -     A  B  B     R  E  V  I  A  T  I  O
7250:    N     F  O  R     R  E     M     *  .  _  R  X  .
COMMAND?       << ASCII DUMP  (A COMMAND)  >>_
```

```
1 CLEAR100:CLS:DIMH(16),H$(16),X$(16):X$="0123456789ABCDEF":DATA
"0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F":
V$="28":U$="HIGHEST LINE IN MEMORY":D$="":T$="TO ERASE HEXMEM, T
YPE:  DELETE 1-28  <ENTER>":'
* HEXMEM VER 1.1 *
2 P=0:O=0:INPUT"COMMAND";C$:IFC$="H"THEN3ELSEIFC$="D"THEN4ELSEIF
C$="G"THEN5ELSEIFC$="M"THEN8ELSEIFC$="A"THENP=1:GOTO8ELSEIFC$="E
"THEN15ELSEIFC$="O"THENO=1:GOTO15ELSEIFC$="L"THEN24ELSEIFC$="C"T
HENV$=U$:D$=T$:GOTO24ELSEIFC$="S"THEN28ELSEPRINT"+INVALID+":GOTO
2
3 INPUT"HEXADECIMAL";A$:A$="0"+A$:GOSUB19:PRINT"  DECIMAL =";D:G
OTO2
4 INPUT"DECIMAL";D:GOSUB21:PRINT"  HEXADECIMAL = ";H$(1)+H$(2)+H
$(3)+H$(4):GOTO2
5 INPUT"HEX OR DEC (TYPE X AFTER DECIMAL)";M$:IFRIGHT$(M$,1)<>"X
"THENA$=M$:GOSUB19:MM=DELSEMM=VAL(M$)
6 IFMM<32THENG$="CONTROL CHARACTER"ELSEIFMM=32ORMM=128THENG$="SP
ACE"ELSEIFMM<192THENG$=CHR$(MM)ELSEIFMM<=255THENG$="TAB FOR"+STR
$(MM-192)+" SPACES"
7 PRINT"  ASCII CHARACTER = ";G$:GOTO2
8 H1$="":H2$="":INPUT"ADDRESS (HEX) #1";H1$:A$=H1$:GOSUB19:GOSUB
27:D5=D:INPUT"ADDRESS (HEX) #2";H2$:A$=H2$:GOSUB19:D6=D:LF=1:W=D
5:PRINTH1$":    ";:IFH2$=""THEND6=32767
9 FORM=D5TOD6:G$=INKEY$:IFG$<>""GOSUB26
10 IFLF=17THENLF=1:W=W+16:PRINTCHR$(29):D=W:GOSUB21:PRINTH$(1);H
$(2);H$(3);H$(4);":    ";
11 M1=M:IFM1>32767THENM1=M1-65535
12 D=PEEK(M1):IFP=1THENGOSUB23ELSEGOSUB22
13 PRINTH$(3);H$(4);" ";:LF=LF+1:IFLF=9PRINT"    ";
14 NEXTM:PRINT:GOTO2
15 INPUT"ADDRESS TO CHANGE (HEX OR DEC - TYPE X AFTER DEC)";M$:I
FM$="K"THENGOTO2ELSEIFRIGHT$(M$,1)<>"X"THENA$=M$:GOSUB19:MM=DELS
EMM=VAL(M$)
16 PRINT"CURRENT CONTENTS: ";:IFMM>32767THENMM=MM-65535
17 CC=PEEK(MM):D=CC:GOSUB22:PRINTH$(3)+H$(4);" HEX  (";CC;"DECIM
AL)":M$="":PRINT:INPUT"NEW VALUE (HEX OR DEC - TYPE X AFTER DEC)
";M$:IFM$=""THENNN=CCELSEIFM$="K"THENGOTO2ELSEIFRIGHT$(M$,1)<>"X
"THENA$=M$:GOSUB19:NN=DELSENN=VAL(M$)
18 POKEMM,NN:IFO=0THENGOTO15ELSEPRINT"NEXT ADDRESS - ";:MM=MM+1:
GOTO16
19 D=0:K=1:FORJ=1TOLEN(A$)-1:K=K*16:NEXTJ:K=INT(K+.01):FORI=1TOL
EN(A$):FORJ=1TO16:IFMID$(A$,I,1)=MID$(X$,J,1)THENGOTO20ELSENEXTJ
20 D=D+K*(J-1):K=K/16:NEXTI:RETURN
21 H(1)=INT(D/4096):D1=D-H(1)*4096:H(2)=INT(D1/256):D2=D1-H(2)*2
56:H(3)=INT(D2/16):D3=D2-H(3)*16:H(4)=D3:FORX=1TO4:RESTORE:FORZ=
0TOH(X):READH$(X):NEXTZ:NEXTX:RETURN
22 H(3)=INT(D/16):D1=D-H(3)*16:H(4)=D1:FORX=3TO4:RESTORE:FORZ=0T
OH(X):READH$(X):NEXTZ:NEXTX:RETURN
```

```
23 H$(3)=" ":IFD<32THENH$(4)=". ":RETURNELSEIFD>127THENH$(4)=". ":
RETURNELSEH$(4)=CHR$(D):RETURN
24 PRINT"BE SURE LINE NUMBERS ARE GREATER THAN ";V$;". ":PRINT"WH
EN TAPE RECORDER STOPS, TYPE:":PRINT"POKE 16548,233: POKE 16549,
66  <ENTER>":PRINTD$:PRINT:PRINT"PRESS <ENTER> WHEN READY TO   *
 CLOAD *":INPUTA1$
25 IFPEEK(16633)>=2THENPOKE16548,PEEK(16633)-2:POKE16549,PEEK(16
634):CLOADELSEPOKE16548,PEEK(16633)+254:POKE16549,PEEK(16634)-1:
CLOAD
26 G$=INKEY$:IFG$=""THENGOTO26ELSEIFG$="K"THENPRINT:GOTO2ELSERET
URN
27 IF(D<16)OR(H1$="")THEND=0:H1$="0000":RETURNELSEIFD/16=INT(D/1
6)THENRETURNELSED=D-(D-INT(D/16)*16):GOSUB21:H1$=H$(1)+H$(2)+H$(
3)+H$(4):RETURN
28 CLS:PRINT"PRESS <ENTER> WHEN READY TO   * CSAVE *":INPUTA1$:C
SAVE"A"
```

One HEX number of 2 digits OO - FF HEX can store the numbers 0-255 decimal, the same as 1 8-bit binary byte.

By looking at the LISTing, and comparing it to the HEX and ASCII dump one can learn how BASIC stores the program lines.

OO is used by BASIC as the terminator of every line. Following 00 are 4 bytes (4 HEX digits) which are housekeeping for BASIC. The first 2 bytes are the memory address of the start of the next line, given with the least significant byte first, most significant byte last.

At location 7219, the address bytes are 7232 meaning that the next program line (line 110) starts at memory address 7232. Looking at the HEX dump, it is seen that the terminator of line 100 (00) is at address 7231 and 7232 is the beginning of the next line.

The next two bytes 64 00 in addresses 721B-721C are the BASIC line number of the line given in HEX least significant byte first -00 64 HEX equals 100 decimal (use the H command). This is why BASIC line numbers cannot exceed 65,535. 2 bytes can only store up to FFFF HEX.

Lines 110 (006E HEX), 120 (0078 HEX), 130 (0082 HEX) up to line 200 (00C8 HEX) of the demonstration program may be identified the same way as line 100 was.

If the byte following the line terminator 00 is 0 (meaning address of the start of the next line is 0) BASIC assumes it has reached the end of the program. This is as far as the program can be LISTed on the screen, even though the complete program remains in memory and can be accessed by the A or M commands.

After the four housekeeping bytes, BASIC starts the text of the line. To conserve memory space as 1 byte abbreviations:

80 HEX = END
81 HEX = FOR
8A HEX = DIM
87 HEX = NEXT
8D HEX = GOTO
91 HEX = GOSUB
92 HEX = RETURN
93 HEX = REM
D5 HEX = = (Equals sign)
and so on.

The rest of the line is stored as the HEX equivalent of the decimal ASCII character codes. The complete list (in decimal which can be converted to HEX by the D command) may be found on page C/2 of the LEVEL II BASIC REFERENCE MANUAL.

08 — 1F are the cursor control codes.
20 — 7F are ASCII character codes
80 — BF are graphic codes
CO — FF are space compression codes (tab 0-63 spaces)

This method of storage and housekeeping explains the "garbage" seen on the screen after a bad CLOAD is LISTed.

If any byte of the code in memory has been changed to 00, BASIC assumes the next 2 bytes are the starting address of the next line, and the 2 bytes after that are the line number. Following bytes are LISTed as the full printing of the command abbreviations (see BASIC TOKENS). This mess is complicated by the fact that HEX digits 08-IF are moving the cursor all over the screen and causing the commands to be printed anywhere at random.

Chapter 12

Z-80 Disassembler
by George Blank

Perhaps the most useful form in which this book could have been printed would have been with a complete disassembled and commented line listing of Level II BASIC. The cost of purchasing publishing rights for this was prohibitive, so we are instead providing a way for you to make your own, if you have a Level II BASIC computer.

This program will disassemble the full Z-80 (trademark of Zilog Corporation) set of amost 700 instructions. In addition, the ability to construct a symbol table and reserve data blocks in the disassembled listing has been added for a truly useful listing.

Since one of the best features of BASIC programming is the ease with which programs may be modified for special uses, many remark statements have been included. Particular routines that users may wish to modify include the automatic printout routine in Line 62 and the count-and-stop routine in Line 69.

The automatic printout routine works by testing the printer-ready status bit at Location 14312. If it finds the value 63, indicating that the printer is on and ready for data, it sends data to the printer. If you have no printer, you may wish to remove this routine, and if your printer does not use the handshake at 14312, you may wish to change the print option.

If you want a continuous printout, you may simply delete Line 69, or you may modify the counter (C is the number of lines printed) to fit the page size on your printer. For example, you might use:

```
IF (PEEK(14312)=63 AND C>15) OR C>60 THEN INPUT"(PRESS
ENTER)";X$:C=0
```

If you wanted to stop after 15 lines on the screen or 60 on the printer.

The line numbers begin at 30 to make it easy to combine this program with HEXMEM. Then you could either patch the programs with a GOTO from HEXMEM or use the command "RUN 30" to use the disassembler. By removing all the REM statements, you may still be able to load another BASIC program above HEXMEM and the disassembler, as long as there are no conflicts in the numbering of the programs.

If you wish to allow for the entry of hexadecimal addresses for the start and finish of your disassembly, there is a conversion routine from hex to decimal at Lines 178 - 186. The routines to convert decimal to hex are located at Lines 70, 73-78 and 84. If you wish to print displacement addresses in $ or + and - form, the routine currently used to calculate hex address jumped to is in Lines 80 - 82.

To use the symbol table, answer "Y" when asked if you wish to construct a symbol table. Then choose hex or decimal entry and enter first the memory location and then your chosen symbol. Table entry will end when you fail to enter a value or symbol, or when your address exceeds the ending point you have chosen for your symbolic dump. If you wish to print the symbol table after your memory dump, add a flag equal to the value of S in Line 176 just before the final GOTO (SE=S), change Line 63 to:

```
63 if M>=ME then 192
```

and add a routine to take the addresses stored in the array MS(0) to MS(SE), convert them to hexadecimal if you wish, and print the corresponding symbol from the array AS(0) to AS(SE). If you really want to be fancy, add a routine to ignore the data blocks on the first pass through the table, then print the data blocks after the symbol table.

The symbol table routine reserves the symbols "DATA" and "EOD" for the start and finish data blocks. If the program, during execution, comes across the symbol "DATA", it sets DA to 1 in Line 165 and the program jumps to the data routine at 188 to 191 from Line 72. Then, once it comes across the symbol "EOD", it sets DA to 2 in Line 165 and back to 0 (Data flag off) in Line 188.

```
30 REM * Z-80 DISASSEMBLER * COPYRIGHT (C) 1980 GEORGE BLANK *
31 CLEAR1000:DEFSTRA:DEFINTB-L,N-Z:DIMAH(15):DIMA(200)
32 FORB=0TO15:READAH(B):NEXT:FORB=0TO7:READAD(B):NEXT:FORB=0TO9:READAP(B):NEXT:FORB=0TO7:READAI(B):N
EXT:FORB=0TO7:READAF(B):NEXT:FORB=0TO7:READAA(B):NEXT:AC(2)="HL":AC(3)="A"
33 REM * AH(0-15) *
34 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
35 REM * AD(0-7)  *
36 DATA B,C,D,E,H,L,(HL),A
37 REM * AP(0-15) *
38 DATA BC,DE,HL,SP,AF,(BC),(DE),(HL),(SP),AF'
39 REM * AI(0-7)  *
40 DATA ADD,ADC,SUB,SBC,AND,XOR,OR,CP
41 REM * AF(0-7)  *
42 DATA NZ,Z,NC,C,PO,PE,P,M
43 REM * AA(0-7)  *
44 DATA RLC,RRC,RL,RR,SLA,SRA,SRL,SRL
45 REM * INPUT ADDRESSES TO DISASSEMBLE *
46 C=0:CLS:PRINT"Z-80 DISASSEMBLER":INPUT"STARTING ADDRESS (DECIMAL)";MB:INPUT"ENDING ADDRESS";ME:M=
MB:INPUT"DO YOU WANT TO CREATE A SYMBOL TABLE";A:IFLEFT$(A,1)="Y"GOSUB169
47 A="":GOTO65:REM * CONTENTS OF 4 BYTES OF MEMORY *
48 D=PEEK(N1):D1=PEEK(N2):D2=PEEK(N3):D3=PEEK(N4):GOSUB86:D4=D/8:D5=D/16:DR=D-8*D4:DF=(D/16-D5)*2:IF
D4<8THEN50ELSEIFD4<16THEN61:ELSEIFD5<12THEN88 ELSE90
49 REM * OP CODES 00H TO 3FH *
50 AD=AD(D4):AP=AP(D5):IFDR<>1THEN51 ELSEIFDF=0THENAP=AP(D5):A="LD "+AP+",":GOSUB76:GOTO62 ELSEA="AD
```

```
D HL,"+AP:GOTO62
51 IFDR<>2THEN52 ELSEIFDF=0ANDD4<4THENA="LD ("+AP+"),A":GOTO62 ELSEIFDF=0ANDD4>3THENA="LD ":GOSUB74:
A=A+","+AC(D5):GOTO62 ELSEA="LD A,("+AP+")":IFD4>3THENA="LD "+AC(D5)+",":GOSUB74:GOTO62 ELSE62
52 IFDR<>3THEN53 ELSEIFDF=0THENA="INC "+AP:GOTO62 ELSEA="DEC "+AP:GOTO62
53 IFDR=4THENA="INC "+AD:GOTO62
54 IFDR=5THENA="DEC "+AD:GOTO62
55 IFDR=6THENA="LD "+AD+",":GOSUB78:GOTO62
56 IFDR=7THEN58 ELSEIFD4=0THENA="NOP"ELSEIFD4=1THENA="EX AF,AF'"ELSEIFD4=2THENA="DJNZ"ELSEIFD4=3THEN
A="JR"ELSEIFD4=4THENA="JR NZ"ELSEIFD4=5THENA="JR Z"ELSEIFD4=6THENA="JR NC"ELSEA="JR C"
57 IFD4<2THEN62 ELSE80
58 IFD4=0THENA="RLCA"ELSEIFD4=1THENA="RRCA"ELSEIFD4=2THENA="RLA"ELSEIFD4=3THENA="RRA"ELSEIFD4=4THENA
="DAA"ELSEIFD4=5THENA="CPL"ELSEIFD4=6THENA="SCF"ELSEA="CCF"
59 GOTO62
60 REM * OP CODES 40 - 7F * LD R,R *
61 A="LD "+AD(D4-8)+","+AD(DR):IFD=118THENA="HALT"
62 N=N+1:M=M+N:AX=LEFT$(AX,N*3):N=0:PRINTTAB(6)AXTAB(20)ASTAB(30)A:IFPEEK(14312)=63LPRINTA1TAB(6)AXT
AB(20)ASTAB(30)A
63 IFM>METHENINPUT"<PRESS ENTER>";X$:GOTO46 ELSE47
64 REM * NEXT MEMORY LOCATION *
65 IFM<32768THENN1=MELSEN1=M-65536
66 IFM<32767THENN2=M+1ELSEN2=M-65536+1
67 IFM<32766THENN3=M+2ELSEN3=M-65536+2
68 IFM<32765THENN4=M+3ELSEN4=M-65536+3
69 C=C+1:IFC=15THENINPUT"<PRESS ENTER>";X$:C=0
70 H0=INT(M/4096):H1=INT((M-4096*H0)/256):H2=INT((M-(4096*H0+256*H1))/16):H3=M-(4096*H0+256*H1+16*H2
):A1=AH(H0)+AH(H1)+AH(H2)+AH(H3)+" ":PRINTA1;" ";
71 IFS>0THEN164
72 IFDB>0THEN188ELSE48
73 REM * PRINT 2 DIGIT HEX CODE IN ( ) *
74 A=A+"(":GOSUB76:A=A+")":RETURN
75 REM * PRINT 2 DIGIT HEX CODE *
76 H=D2:GOSUB84:GOSUB78:N=N+1:RETURN
77 REM * PRINT 1 DIGIT HEX CODE *
78 H=D1:GOSUB84:N=N+1:RETURN
79 REM * CALCULATE DISPLACEMENT *
80 A=A+" ":H=D1:IFH>127THENH=H-256
81 REM * PROGRAM COUNTER = +2 * PRINT HEX ADDRESS *
82 MH=H+M+2:D2=INT(MH/256):D1=MH-256*D2:GOSUB76:N=N-1:GOTO62
83 REM * CONVERT BYTE TO HEX *
84 H1=INT(H/16):A=A+AH(H1):H1=H-H1*16:A=A+AH(H1):RETURN
85 REM * CONVERT CONTENT OF MEMORY TO HEXADECIMAL *
86 H=D:GOSUB84:A=A+" ":H=D1:GOSUB84:A=A+" ":H=D2:GOSUB84:A=A+" ":H=D3:GOSUB84:AX=A+" ":A="":RETURN
87 REM * OP CODES 80 - BF * REGISTER ARITHMETIC *
88 A=AI(D4-16)+" "+AD(DR):GOTO62
89 REM * OP CODES A0 - FF EXCEPT CB DD ED FD *
90 D4=D4-24:AF=AF(D4):IFDR=0THENA="RET "+AF:GOTO62ELSEIFDR=2THENA="JP "+AF+",":GOSUB76:GOTO62ELSEIFD
R=4THENA="CALL "+AF+",":GOSUB76:GOTO62ELSEIFDR=7THENA="RST ":H=D4*8:GOSUB84:GOTO62ELSEIFDR=6THENA=AI
(D4)+" A,":H=D1:GOSUB84:GOTO62
91 IFDF=1THEN93ELSEIFDR=1THENAP(3)="AF":A="POP "+AP(D5-12):AP(3)="SP":GOTO62ELSEIFDR=5THENAP(3)="AF"
:A="PUSH "+AP(D5-12):AP(3)="SP":GOTO62
92 IFD4=0THENA="JP ":GOSUB76:GOTO62ELSEIFD4=2THENA="OUT ":GOSUB78:A=A+",A":GOTO62ELSEIFD4=4THENA="EX
 SP,HL":GOTO62ELSEA="DI":GOTO62
93 IFDR=5THEN94 ELSEIFDR=1THEN95 ELSEIFD4=1THEN97 ELSEIFD4=3THENA="IN A,":GOSUB78:GOTO62 ELSEIFD4=5T
HENA="EX DE,HL":GOTO62 ELSEA="EI":GOTO62
94 IFD4=1THENA="CALL ":GOSUB76:GOTO62ELSEIFD4=3THEN104ELSEIFD4=5THEN133ELSE106
95 IFD4=1THENA="RET":GOTO62ELSEIFD4=3THENA="EXX":GOTO62ELSEIFD4=5THENA="JP (HL)":GOTO62ELSEA="JP M,"
:GOSUB76:GOTO62
96 REM * OP CODES CB XX  *
97 N=N+1:DA=D1/8:DB=D1-8*DA:IFDA>7THEN98ELSEA=AA(DA)+" "+AD(DB):GOTO62
98 IFDA>15THEN99ELSEA="BIT "+AH(DA-8)+AD(DB):GOTO62
```

```
99 IFDA>23THEN100ELSEA="RES "+AH(DA-16)+AD(DB):GOTO62
100 A="SET "+AH(DA-24)+AD(DB):GOTO62
101 REM * ADJUST NN FOR 4 BYTE OP CODE *
102 D1=D2:D2=D3:RETURN
103 REM * OP CODES DD XX *
104 AY="IX":GOTO108
105 REM * OP CODES FD XX *
106 AY="IY"
107 REM * AY = IX OR IY * AZ = (IX+DIS) OR (IY+DIS) *
108 N=N+1:A="":H=D2:GOSUB84:AZ="("+AY+"+"+A+")":A="":D4=D1/8
109 REM * HEX HALF BYTES OF SECOND BYTE: AW=MSHB AX=LSHB *
110 H=D1:GOSUB84:AW=LEFT$(A,1):AX=RIGHT$(A,1):A="":IFD1=203THEN128
111 REM * XD09 TO XD39 *
112 IFD1>57THEN114 ELSEIFAX="9"THENAP(2)=AY:A="ADD "+AY+","+AP(VAL(AW)):AP(2)="HL":GOTO62
113 REM * XD21 TO XD36 *
114 IFD1=33THENA="LD "+AY+",":GOSUB102:GOSUB76:GOTO62ELSEIFD1=34THENA="LD ":GOSUB102:GOSUB74:A=A+","
+AY:GOTO62ELSEIFD1=35THENA="INC "+AY:GOTO62ELSEIFD1=42THENA="LD "+AY+",":GOSUB102:GOSUB74:GOTO62ELSE
IFD1=43THENA="DEC "+AY:GOTO62
115 IFD1=52THENA="INC "+AZ:N=N+1:GOTO62ELSEIFD1=53THENA="DEC "+AZ:N=N+1:GOTO62ELSEIFD1=54THENA="LD "
+AZ+",":H=D3:GOSUB84:GOTO62
116 REM * XD86XX TO XD6EXX *
117 IFD1>111THEN119ELSEIFD1<70OR(NOT(AX="6"ORAX="E"))THEN140 ELSEA="LD "+AD(D4-8)+","+AZ:N=N+1:GOTO6
2
118 REM * XD70XX TO XD7EXX *
119 IFD1>117THEN120ELSEA="LD "+AZ+","+AD(D1-112):N=N+1:GOTO62
120 IFD1>133THEN122 ELSEIFD1=119THENA="LD "+AZ+",A":N=N+1:GOTO62 ELSEIFD1=126THENA="LD A,"+AZ:N=N+1:
GOTO62 ELSE140
121 REM * CULL INOPERABLE CODES *
122 IFD1>190THEN126ELSEIFAX="6"THEN124ELSEIFAX="E"THEN124ELSE140
123 REM * XD86XX TO XDBEXX *
124 A=AI(D4-16)+" A,"+AZ:N=N+1:GOTO62
125 REM * XDE1 TO XDF9 *
126 IFD1=225THENA="POP "+AY:GOTO62ELSEIFD1=227THENA="EX (SP),"+AY:GOTO62ELSEIFD1=229THENA="PUSH "+AY
:GOTO62ELSEIFD1=233THENA="JP ("+AY+")":GOTO62ELSEIFD1=249THENA="LD SP,"+AY:GOTO62ELSE140
127 REM * INDEXED BIT AND ROTATE GROUP * DD CB  AND  FD CB *
128 D4=D3/8:DR=D3-8*D4:IFDR<>6THEN140
129 IFD4<8THENA=AA(D4)+" "+AZ:N=N+2:IFD4=6THEN140 ELSE62
130 D4=D4-8:IFD4<8THENA="BIT"ELSED4=D4-8:IFD4<8THENA="RES"ELSED4=D4-8:A="SET"
131 A=A+STR$(D4)+","+AZ:N=N+2:GOTO62
132 REM * ED GROUP * CULL INOPERATIVE CODES *
133 N=N+1:IFD1<64ORD1>188OR(D1>163ANDD1<169)OR(D1>171ANDD1<175)OR(D1>179ANDD1<184)THEN140
134 IFD1<124THEN142 ELSEIFD1<143THEN140
135 REM * EDA0 TO EDBB * BLOCK TRANSFER AND SEARCH *
136 D=D1:IFD>159A="LDI":IFD>160A="CPI":IFD>161A="INI":IFD>162A="OUTI":IFD>167A="LDD":IFD>168A="CPD":
IFD>169A="IND":IFD>170A="OUTD":IFD>175A="LDIR":IFD>176A="CPIR":IFD>177A="INIR":IFD>178A="OTIR":IFD>1
83A="LDDR":IFD>184A="CPDR":IFD>185THENA="INDR"
137 IFD=187THENA="OTDR"
138 GOTO62
139 REM * INOPERATIVE CODE * ADJUST FOR SINGLE BYTE *
140 N=N-1:A="-DATA-":GOTO62
141 REM * ED40 TO ED7BXXXX *
142 D=D1-64:D4=D/8:D5=D/16:DR=D-8*D4:DF=(D/16-D5)*2:AC="(C)"
143 REM * EDX0 *
144 IFDR>0THEN146ELSEA="IN "+AD(D4)+","+AC:IFD4=6THEN140ELSE62
145 REM * EDX1 EDX9 *
146 IFDR>1THEN148ELSEA="OUT (C),"+AD(D4):IFD4=6THEN140ELSE62
147 REM * EDX2 EDXA *
148 IFDR>2THEN150ELSEIFDF=0THENA="SBC HL,"+AP(D5):GOTO62ELSEA="ADC HL,"+AP(D5):GOTO62
149 REM * EDX3 EDXB *
150 IFDR>3THEN154ELSEA="LD ":GOSUB150:IFDF=0THENGOSUB74:A=A+","+AP(D5)ELSEA=A+AP(D5)+",":GOSUB74
```

```
151 REM * NO ED63 ED6B *
152 IFD5=2THEN140ELSE62
153 REM * ED44 *
154 IFDR>4THEN156ELSEIFD4=0THENA="NEG":GOTO62 ELSE140
155 REM * ED45 ED4D *
156 IFDR>5THEN158ELSEIFD5>0THEN140ELSEIFDF=0THENA="RETN":GOTO62ELSEA="RETI":GOTO62
157 REM * ED46 ED56 ED5E *
158 IFDR>6THEN161ELSEIFD=6THENA="IM 0":ELSEIFD=22THENA="IM 1"ELSEIFD=30THENA="IM 2"ELSE140
159 GOTO62
160 REM * EDX7 *
161 IFD=7THENA="LD I,A"ELSEIFD=15THENA="LD R,A"ELSEIFD=23THENA="
LD A,I"ELSEIFD=31THENA="LD A,R"ELSEIFD=39THENA="RRD"ELSEIFD=47TH
ENA="RLD"ELSE140
162 GOTO62
163 I$=INKEY$:IFI$=""THEN163 ELSEPRINTI$:RETURN
164 AS="":IFM>=MS(SN)THENAS=AS(SN):SN=SN+1:IFSN>STHENS=0
165 IFAS="DATA"THENDB=1ELSEIFAS="EOD"THENDB=2
166 GOTO72
167 S=S-MS(S):IFS<0THENS=0
168 GOTO173
169 CLS:PRINT"SYMBOL TABLE CONSTRUCTION":PRINT:PRINT"IF YOU WANT A SYMBOL TABLE, YOU MUST ENTER EACH
 ADDRESS AND THE":PRINT"SYMBOL YOU WANT. YOU MUST ENTER THE ADDRESSES IN NUMERICAL":PRINT"ORDER. SYM
BOLS ARE LIMITED TO SIX CHARACTERS AND 100 SYMBOLS."
170 PRINT:PRINT"TWO SYMBOLS ARE RESERVED FOR SPECIAL USE:":PRINTTAB(6)"USE 'DATA' TO INDICATE THE ST
ART OF A BLOCK OF DATA":PRINTTAB(6)"USE 'EOD' TO INDICATE THE END OF A BLOCK OF DATA."
171 DIM AS(100):DIM MS(100):S=0:SN=0
172 PRINT:PRINT"PRESS <ENTER> AFTER LAST SYMBOL TO END INPUT":PRINT"DO YOU WISH TO ENTER HEX OR DECI
MAL ADDRESSES (H/D)":GOSUB163
173 IFI$="H"THEN177
174 PRINTS;:INPUT"MEMORY LOCATION (DECIMAL)";MS(S)
175 IFMS(S)<0THEN167ELSEIFMS(S)>MEORMS(S)<MPTHENPRINT:RETURN
176 INPUT"<SYMBOL>";AS(S):IFAS(S)=""THENPRINT:RETURNELSEMP=MS(S):S=S+1:GOTO173
177 INPUT"MEMORY LOCATION <HEXADECIMAL>";A
178 MS=0:L=LEN(A):ONLGOSUB183,182,181,180
179 PRINT:RETURN
180 AH=LEFT$(A,1):A=RIGHT$(A,3):GOSUB185:MS=MH*4096
181 AH=LEFT$(A,1):A=RIGHT$(A,2):GOSUB185:MS=MS+MH*256
182 AH=LEFT$(A,1):A=RIGHT$(A,1):GOSUB185:MS=MS+MH*16
183 AH=A:GOSUB185:MS=MS+MH:PRINT"DECIMAL: ";MS;:MS(S)=MS:GOTO175
184 REM * CONVERT HEX TO DECIMAL *
185 FORX=0TO15:IFAH(X)=AHTHENMH=X:RETURN
186 NEXT:MH=0:RETURN
187 REM * DATA BLOCK PRINTOUT *
188 D=PEEK(M1):H=D:GOSUB84:AX=A:IFDB=2THENDB=0
189 IF D>31 AND D<96 THEN A=CHR$(A) ELSE A=" "
190 GOTO 62
```

Chapter 13

Map of TRSDOS and NEWDOS

by John Hartford

Using Disk Operating System routines is complex for two reasons. The DOS does a lot of different things and it does them in a complex way with overlay techniques. Overlay technique means that the same area in memory is used for a number of different programs. Not only that, but the different programs that use the same memory can even call each other.

How the System Initializes

When you turn on your computer, control is transferred to memory location 0000H in the ROM. If you press RESET, control is transferred to 0066H. Both of these routines jump to 0674H, which initializes pointers in RAM and checks the floppy disk controller chip to see if an expansion interface is connected. If the chip is active, control jumps to a routine at 069FH, which is the bootstrap routine.

The bootstrap loads sector 0, track 0, from drive 0, into locations 4200H to 42FFH in RAM from the disk drive. Then control is transferred to 4200 to run the boot routine. The boot looks in the directory on the disk for SYS0/SYS, and if it is on the disk, loads it into memory, then jumps to the execution address for SYS0/SYS.

Pages

Recall the memory map of the TRS-80. It is organized as follows:

0000	to	2FFF	Level II BASIC ROM
3000	to	37DF	empty
37E0	to	37FF	Memory Mapped Input/ Output
3800	to	3BFF	Keyboard
3C00	to	3FFF	Video Display memory
4000	to	7FFF	First 16K RAM
8000	to	BFFF	Second 16K RAM
C000	to	FFFF	Third 16K RAM

If we divide RAM into 256 byte pages, then the first two hex digits of the memory address is the page number. For example, 4200 to 42FF would be page 42, and 0000 to 00FF is page 0.

Pages 40 and 41 are reserved memory for both Level II BASIC and the Disk Operating System. This area holds pointers, addresses, data, and modifiable programs used by either (or both) BASIC and DOS.

Page 42 is an overlay area. It is the I/O buffer page and operating area for the BOOT, and later a general I/O buffer for the DOS. In Level II BASIC it is part of the keyboard buffer. A buffer is any section of memory that holds data temporarily before it is moved to the place it belongs or where it can be checked for correctness before being used.

Pages 43 through 4C are loaded from SYS0/SYS and form the core of the Disk Operating System, remaining in the system at all times. Page 43 has reserved memory from 4300 to 4317, a keyboard buffer from 4318 to 4357, and a number of patches fixing mistakes in other programs, primarily Level II I/O routines in TRSDOS and patches to TRSDOS in NEWDOS.

Page 44 is largely a jump table, allowing fixed jumps from ROM or RAM to be redirected to any area in memory.

Page 45 contains interrupt data and routines for handling interrupts.

Page 46 contains the programs that actually talk to the floppy disk controller in order to move data to and from the disk and random access memory.

Pages 47, 48, and 49 handle random access disk operations including calculations.

Pages 4A and 4B (to 4BA1) read and write the directory, including the granule allocation table, but not the hash index table.

The page from 4BA2 to 4CAB controls and loads the overlays, while the rest of page 4C prints the clock, the calendar, and the trace program counter number. Yes, the calendar is there, it is just another feature like devices that were forgotten before being fully implemented.

Page 4D is a disk I/O buffer.

Overlay Area 1 (4E00 to 51FF)

Pages 4E through 51 contain overlay area number 1. The contents of an overlay area depend on what the disk operating system is doing at the time. Overlay number 1 is very busy. Immediately after RESET, this area contains the second part of SYS0/SYS. This initializes the pointers in reserved memory, and sets up the data control blocks, the stack, and the interrupt mode.

Once this is done, SYS0/SYS is replaced by SYS1/SYS in overlay 1. This program tells you DOS READY and interprets what you type in once you press [ENTER]. After that, the contents of overlay one are determined by your DOS command.

When you open files so that they can be read by DOS or BASIC, this is done by SYS2/SYS in overlay 1. When you close or kill files, this area holds SYS3/SYS. NEWDOS COPY also uses SYS3/SYS to format disks for full disk copying. SYS4/SYS loads the error messages here when you make a mistake. DEBUG (SYS5/SYS) also loads into overlay one.

Overlay Area 2 (5200 up, potentially as far as FFFF)

SYS6/SYS uses 5200 to 60FF to execute the DOS library commands, except for directory, which also uses 6100 through 68FF.

BASIC, BACKUP, COPY, and many purchased utility programs all start around page 52. If one program resides here, obviously another cannot use the same memory. But do not assume that just because you did not change what was in this area, nothing else did.

Summary

It is convenient to think of two different types of memory use in the command area. First, there is the permanent area, from 4400 to 4CFF which operates the disks, performs certain calculations, and services interrupt requests. Pages 40, 41, and 43 are used as reserved or scratchpad memory, and pages 42 and 4D as disk input/ output buffers. Think of this area as a collection of utility programs waiting to be called.

The second kind of memory use is for transient programs which are used once and then discarded. If you want to use them again they must be reloaded from disk, with a few exceptions. Most of them load to the same place; SYS1, SYS2, SYS3, SYS4, and SYS5 all load into pages 4E to 51.

Other programs use a different area of RAM starting at page 52. This location is used by SYS6, BASIC, BACKUP, and COPY, among others. The relationship in the overlay areas is not greatly different from a series of BASIC programs calling each other from disk, but never coexisting in memory. Note that when you are in BASIC, the BASIC interpreter extensions for disk BASIC are always in the second overlay area.

Calling an Overlay

The DOS calls overlays by loading the accumulator with a one byte code and executing a restart 5. In this case the low nibble contains the number of the SYSTEM file plus two, and the high nibble contains eight plus the number of the command to be executed.

For example, to do a KILL, you want command 2 of SYS3/SYS. The command number would be 8+2, or A, and the system is 3+2, or 5. Therefore you load A5 into the accumulator and RST5. If you wanted to load a program without executing it, treat it as a command 0. SYS3/SYS would require a 85, 8+0 for the command and 3+2 for the program.

This transfers control to the overlay control routine at 4BA2 in SYS0. RST5 (EF Hex) is a one byte call, but routine 4BA2 discards the return address. Therefore, in order to call 4BA2, you must call the three byte routine which loads the accumulator with the desired command and does a RST5. At 4BA2, the routine checks the leftmost bit of accumulator A. If it is set, it is accepted as a valid command. It splits the high and low nibble and saves the byte at 430E, comparing the low nibble with the last one saved to see if the overlay being called is already in RAM. If not, it loads the new overlay. Then the overlay is called. The call is usually to a jump table, except for SYS4 (error messages) and SYS5 (DEBUG), which have only one command. The jump table in the other routines uses the high nibble in accumulator A to select the appropriate sub command.

```
                 TRSDOS SYMBOL TABLE - SYS0
                          (KERNAL)

4300 - TRACK number of DRIVE 0
4301 - TRACK number of DRIVE 1
4302 - TRACK number of DRIVE 2
4303 - TRACK number of DRIVE 3
4304 - Directory TRACK of DRIVE 0
4305 - Directory TRACK of DRIVE 1
4306 - Directory TRACK of DRIVE 2
4307 - Directory TRACK of DRIVE 3
4308 - DRIVE number in binary = DATA for 4600
4309 - BIT SET indicates DRIVE number
430A - Stacker Storage   -   Pointer Caller
430E - Last command to DOS
430F -   Semaphore byte:
         HIGHBIT = DEBUG FLAG
         LOWBIT  = PROTECTION FLAG
         BIT 4   = SYS6

             7 6 5 4 3 2 1 0
             !   ! !       PROTECTION FLAG
             !   ! SYS6 FLAG
             !   BASIC CHAINING
             DEBUG (SYS5) FLAG

4312 - JUMP director for positive DOS commands
4318 - DOS keyboard buffer
4358 -   New Keyboard DCB               --- (TRSDOS ONLY)
4378 -   Change keyboard driver         --- (TRSDOS ONLY)
4398 -   Wait for NOT BUSY              --- (TRSDOS ONLY)
43AC -   Select Drive control           --- (TRSDOS ONLY)
43B6 -   "DEVICE" DATA                  --- (TRSDOS ONLY)
43D1 -   Fetch A = End of File offset   --- (TRSDOS ONLY)
43D8 -   DOS keyboard driver            --- (TRSDOS ONLY)
4358 - BIT 7 = CONTROL SWITCH           --- (NewDOS ONLY)
435F - RELAY for 47CF                   --- (NewDOS ONLY)
4366 - READ SECTOR and decrement NRN    --- (NewDOS ONLY)
4377 - KEYBOARD modification DRIVER     --- (NewDOS ONLY)
43B1 - SCREEN printer routine           --- (NewDOS ONLY)
```

```
43DD - Get TRACK number of SYSTEM FILES   --- (NewDOS ONLY)
4400 -   (93/EF) = VECTOR RESTART
4405 -   (B3/EF) = (#3 OF SYS1)
4409 - JUMP to 44B0 = DISPLAY ERROR
440D - JUMP to 44B4 = which is  RST 6(87) = Get DEBUG
4410 - JUMP to 4596 = turn on INTERRUPT
4413 - JUMP to 4593 = turn off INTERRUPT
4416 - JUMP to 45A5 = turn on CALLER
4419 - JUMP to 458E = turn off CALLER
441C -   (C3/EF) = move a FILE NAME-(FCB)-from (HL) to (DE)
4420 -   (A4/EF) = INIT
4424 -   (94/EF) = OPEN
4428 -   (95/EF) = CLOSE
442C -   (A5/EF) = KILL
4430 - JUMP TO 4C16 = LOAD MACHINE LANGUAGE FILE
4433 - JUMP TO 4C06 = LOAD and RUN MACHINE LANGUAGE FILE
4436 - JUMP TO 476D = READ
4439 - JUMP TO 478B = WRITE
443C - JUMP TO 47A8 = VERIFY
443F - JUMP TO 4756 = POSITION to first RECORD NUMBER
4442 - JUMP TO 4700 = POSITION to (BC) RECORD NUMBER
4445 - JUMP TO 4737 = POSITION to next RECORD NUMBER
4448 - JUMP TO 475F = POSITION to last RECORD NUMBER
444B -   "RESTORE"
444D - Issue a DISK COMMAND
4455 - WAIT = and RENAME DISK DRIVE the SAME
       (to prevent timeout)
4467 - JUMP to 44CF = OUT 1 line to VIDEO
446A - JUMP to 44DF = OUT 1 line to PRINTER
446D - JUMP to 4CB7 = GET TIME
4470 - JUMP to 4CD2 = GET DATE
4473 -   (D3/EF) = (#5 OF SYS1) = add a default extension
                                  If none- to which
                                  HL points
4476 -   (E3/EF) = (#6 of SYS1)
4480 - Full-sized BUFFER for FILE FCB, DCB, and I/O
44A0 - Short DATA BUFFER = OPEN DCB DATA
44B0 -   (86/EF) = Display an ERROR MESSAGE
44B4 -   RST 6(F7) = Get DEBUG
44B8 -   LD HLA,(HL) = and (PUT-GET 47AE) or (JP (HL))
44CF -   (0D/03) = Out 1 line to VIDEO
44DF -   (0D/03) = Out 1 line to PRINTER
         (0D or 03 terminates output)

Page 45 - INTERRUPT Page

4500 - INDIRECT ADDRESS DATA FOR 4560 = INTERRUPT ADDR DATA
4518 -   RST 7(FF) = INTERRUPTS
4538 - BRANCH = SAVE REGISTER, FETCH ADDRESS, and EXECUTE
454F - BRANCH = If BREAK key down
4560 -   LD HL,((HL)) = do five times and JUMP
458E - TURN OFF
45A3 -   (A2)       / ADDRESS
45A4 -   (45)       /  of C9
45A5 - Turn on caller (Do not RETURN)
45AF -   (B2)       / ADDRESS
45B0 -   (45)       /   and
45B1 -   (05)       / SWITCH
45B2 - Adjust CLOCK and CALENDAR each SECOND
45CE - TIME CONSTANTS                          --- (NewDOS ONLY)
```

```
45CF - PATCHES                                    --- (NewDOS ONLY)
45E3 - PART of DISK COMMANDS                      --- (NewDOS ONLY)
45F3 - PAUSE and FETCH STATUS                     --- (NewDOS ONLY)
```

Page 46 - DISK COMMAND PAGE

```
4600 - SELECT DRIVE = REGISTER C
4639 - SET BIT in A = Value in A register Mod 8
4647 - SET TRACK and LOAD SECTOR REGISTER
       D = TRACK   E = Sector
4658 - Do DISK COMMAND in A
4661 - Issue DISK COMMAND and wait until done
4669 - wait until DISK CONTROLLER not busy
4671 - DISK COMMAND common
46DD - Give READ command
46E6 - Give WRITE command
46EF - Give WRITE/FA command (SYSTEM files)
46F3 - Give VERIFY command
4700 - Position to BC RECORD NUMBER       = (FROM 4442)
4717 - Common for all positions
4737 - Position to next RECORD NUMBER     = (FROM 4445)
4756 - Position to first RECORD NUMBER    = (FROM 443F)
475F - Position to last RECORD NUMBER     = (FROM 4448)
476D - READ      = (FROM 4436)
478B - WRITE     = (FROM 4439)
47A8 - VERIFY    = (FROM 443C)
47AE - GET or PUT next BYTE in BUFFER
47BB -   GET = READ  subroutine
47DA -   PUT = WRITE subroutine
47FF - Store (I/O BLOCK) EOF DATA
480E - READ next SECTOR
482D - Get values for 46DD, give READ, and EXECUTE
483E - Do the WRITE with parameters
4878 - On CLOSE, check status for write on last sector
4883 - Fetch current byte in buffer -   C  = BYTE COUNT
                                        DE = ADDRESS
4892 - STACKER = Point IX to the I/O BUFFER
48B3 - RETURN  = UNSTACKER
48B9 -   CP NRN,ERN = next FIND RECORD NUMBER
48DC - Get ready to READ or WRITE
49B5 - Find file location on DISK DIRECTORY ENTRY
4A00 - Does DIRECTORY WRITE for updating file
4AC1 - DIRECTORY READ
4AD6 - DIRECTORY WRITE
4AF0 - GRANULE ALLOCATION READ   = GAT READ
4B03 - GRANULE ALLOCATION WRITE  = GAT WRITE
4B1E - Convert DECIMAL into TRACK/SECTOR/BYTE in DIRECTORY
4B35 - READ 1 SECTOR - expected to be a SECTOR of DIRECTORY
4B55 - LOAD REGISTER D with DIRECTORY TRACK NUMBER
4B5D - TEST the A-TH BIT in B
4B6A - Multiply HL by A, put answer in HLA, put HL into DE
4B84 - Divide HL by A, put answer in HL, and remainder in A
4B9F -   (AF)      /  GOOD
4BA0 -   (C9)      / RETURN
4BA2 -   (EF/RST 5/NO RETURN) = DOS OVERLAY caller
```

```
                7 6 5 4 3 2 1 0
          A =  N/C C C/N N N N
                /  CMD# / SYS#+2
               /
           If BIT 7 ZERO, then NOT VALID, JUMP to 4312
```

4C06 - LOAD and GO - a machine language file = (from 4433)
4C16 - LOAD - a machine language file = (from 4430)
4C89 - Fetch next byte from buffer
4CA9 - DISPLAY CLOCK - which is at 4CAC
4CD2 - DISPLAY CALENDAR
4CD9 - DISPLAY "TRACE" = (PC of INTERRUPT)

PAGE 4D - BUFFER

PAGE 4E - OVERLAY

PAGE 52 - OVERLAY FOR LIBRARY

6FFF - END OF ALL DOS

PAGE 70 and following - RAM

DCB = Device Control Block
DEC = DIRectory Entry Code
ERN = End Record Number
FCB = File Control Block
GAT = Granule Allocation Table
NRN = Next Record Number

TRSDOS SYMBOL TABLE - SYS1
(Human I/O Interface)

```
4E00 - Start of JUMP TABLE
4E1F -  (93) //Reinitialize DOS and
        (A3) //PROMPT and ACCEPT COMMANDS
4E48 -  JP (DE)  = JUMP to LOCATION in REGISTER DE
4E4E -  (B3) = OBEY COMMAND (LIB or MACHINE LANGUAGE PROG)
4E87 - Displays  "WHAT?"
4E90 - LIB Calls   (SYS6)
4EA4 - LIB Part 1 = ASCII list of BASIC2, DEBUG, and TRACE
4EBD - LIB Listing Part 2
4F56 -  (C3) = Move file name from location pointed to
               by HL to location pointed to by DE
4FA7 - CONNECTORS
4FC0 -  (D3) = Add default extension pointed
               to by HL to file name
4FF4 - String Mover (ALPHA-NUMERIC only)
502A - Match-Maker            In    BC = Match list
                                    DE = Data
                              Out   C  = Counter
                                    DE = Address
507D -  (E3) = Evaluate inside parentheses
50DF - Evaluate  (YES/NO):(ON/OFF)    (Default is OFF)
5104 - Convert ASCII DECIMAL to BINARY INTEGER
511F - Convert ASCII HEX to BINARY INTEGER
514C - BASIC2
5162 - DEBUG
5191 - TRACE
51A0 - Checks for parentheses
51AF - Six-byte buffer
51B5 - "CMD"
51B9 - "DOS READY"
51C3 - "WHAT?"
```

TRSDOS SYMBOL TABLE - SYS2
(OPEN)

```
4E00 - Start of JUMP TABLE
4E12 -   (94) = OPEN
4ED8 -   (A4) = INITIAL
4F50 -   (B4) = Add a file to the DIRectory
4FA7 - Change file DCB to "OPEN" = Set Flags, Load pointers
                                   (Replaces file name)
5027 - Generate Special FCB at 5154
       from buffer pointed to by HL
5081 - Move B ALPHA-NUMERIC bytes starting at location
       pointed to by HL to location pointed to by DE
509B - Generate HIT Hash value
50AB -     /Random start to
50B6 -     /Find an empty spot in HIT
50D1 - Password encoder
50FD - Disk Drive Tester (  Exits with Z flag set on  )
                         (disk-in door-closed power-on)
5123 - Load HIT into RAM (Page 4D)
5141 - Save HIT from RAM onto disk
5154 - Buffer for total file description
5180 - Decrement EOF markers
```

DCB = Device Control Block
EOF = End-of-File
FCB = File Control Block
HIT = Hash Index Table

TRSDOS SYMBOL TABLE - SYS3

4E00 - Start of JUMP TABLE (old) --- (TRSDOS only)
4E0D - (95) = CLOSE a file
4E95 - On CLOSE, WRITE file name into FCB
4F72 - (A5) = KILL a file
4FCE - /SECOND PART OF 5047
4FF0 - Reset the A-th bit of B
4FFD - READ HIT into page 51
500C - WRITE HIT into disk from page 51
501B - Divide (HL) by (A) but decrement if no remainder
5022 - DIR READ but increment EOF if byte number = 0
5033 - Pseudo-subroutine to get D (Track)
503C - Get DIRectory track number
5040 - DIRectory WRITE
5047 - GAT READ and free granules
504F - NEWDOS beginning of JUMP TABLE --- (NEWDOS only)
5059 - Formatter for COPY/CMD --- (NEWDOS only)

EOF = End-of-File
FCB = File Control Block
GAT = Granule Allocation Table
HIT = Hash Index Table

TRSDOS SYMBOL TABLE - SYS4

4E00 - Decode error coded in a Register into ASCII
4F36 - ASCII data

TRSDOS SYMBOL TABLE - SYS5 (DEBUG)

4E00 - Enter and Save registers
4E4B - DEBUG Command loop
4E99 - X = Register format
4E9A - S = Full scan
4E9E - U = Update
4EA8 - D = Display
4EAE - + = ; = Increment memory display
4EC6 - - = Decrement memory display
4ECB - A & H = Presentation
4ECF - Draw a screen display
4EE5 - Do another line
4EEA - Line formatter
4F2E - Terminate process
4F45 - Branch to fetch LEVEL II cursor for own use
4F54 - Register ASCII data
4F80 - GO
4FB0 - Load register and GO
4FCA - Store a break-point
4FDB - Memory modify
5011 - Register change
505D - C & I = Single step
50B7 - Single step return calculator

```
50C0 - Relative JUMP calculator
50CC - Indexed calculator
50DB - Common loop point for calculators
50E0 - Disassembly data
510E -   (ED prefix)
5115 -   (FD & DD prefix)
5131 - Subroutine for scan display
5165 - Subroutine=OUT a memory marker(To video) or 2 blanks
5180 -   (Branch for following marker)
518A - Fetch and echo key from keyboard
51A3 - Get a number -(ASCII HEX)- from keyboard
51BF - Convert ASCII to HEX  (C=NO/NC=OK)
51D0 - OUT byte pointed to by HL to video   --- (HEX)
51D4 - OUT HL to video                      --- (HEX)
51D9 - OUT A to video                       --- (HEX)
51E2 - OUT right nibble                     --- (HEX)
51EF - OUT A and blank
51F2 - OUT blank
51F6 - OUT 3 bytes                         --- (ASCII)
51F9 - OUT 2 bytes                         --- (ASCII)
51FC - OUT 1 byte                          --- (ASCII)
```

TRSDOS SYMBOL TABLE - SYS6
(Obey LIB Commands)

```
5200 - Start of JUMP TABLE
5251 - Refuse to obey
526B - Do nothing
5283 - AUTO
529D - Get date
52B1 - Get time
52C5 - Turn clock ON or OFF
52D3 - Display ERROR  ('BAD FORMAT')
52EA - Convert 2-digit numbers to BINARY
5315 -   "DEVICE" (Do nothing at all)
533A - LIBrary
536A - PROTect
5454 - Encode new password
5477 - ASCII
5504 - VERIFY (ON or OFF)
5515 - Check keyboard for BREAK or PAUSE
553A -   ERROR = 'FILE SPEC REQUIRED'
5543 -   ERROR = 'DEVICE SPEC REQUIRED'
554C -   ERROR = 'DISK ERROR IN A'
5551 - FCB buffer A
5571 - FCB buffer B
5591 - ASCII

PAGE 56 -- Input/Output buffer

5700 - APPEND
5748 - Relay to  57B6
574F - COPY file (disk-to-disk)
57B6 - Transfer function --  use buffer A as input DCB
                             use buffer B as output DCB
57D7 - Dump
589F - Get transfer address for dump
58BE - Display a line and quit
58C4 - ASCII
5914 - KILL
```

592C - LIST
5994 - LOAD
59AC - PRINT

PAGE 5A -- Input/Output buffer

```
5B00 - ATTRIBute
5B3A - JUMP TABLE for ATTRIBute types
5B4D - Protection           (of ATTRIBute)
5B69 - Change protection
5B97 - Invisible            (of ATTRIBute)
5BA7 - Access password      (of ATTRIBute)
5BBC - Update password      (of ATTRIBute)
5BD1 - Subroutine to search for terminator
5BE0 - Branch from 5BD1 if '=' is found
5BF7 - Set new ATTRIBute s into DIRectory
5C6F - DIRectory
5DE1 - Erase DIRectory from RAM
5DF1 - DIRectory with attributes
5E3E -   LD DE = Track and sector of file pointed to by HL
5E67 - Match tables and ASCII
5EE0 - FREE
5F9C - RENAME
6028 - Display ERROR = 'DRIVE SPEC ERROR'
604E - Display ERROR = 'DUPLICATE FILE NAME'
606C - Convert BINARY into DECIMAL ASCII
60A7 - Divide = (3 bytes pointes to by HL)/C
```

PAGES 61 to 68 - buffer to hold DIRectory in 'DIR'

DCB = Device Control Block
FCB = File Control Block

TRSDOS RESIDENT MEMORY

```
4000 -   (C3)      /  RST 1 (CF), JP 0005, and CALL 0005
4001 -   (96)      /  Compares (HL) to byte following CF
4002 -   (1C)      / (RST 1 code). If true, jumps to 1D78
                   / (RST 2), otherwise gives SYNTAX ERROR
4003 -   (C3)   /  RST 2 (D7) fetches next non blank byte
4004 -   (78)   /  using HL as a text pointer. C flag set
4005 -   (1D)   /  for a number, S flag for a BASIC token
                   Z and S are set by accumulator
4006 -   (C3)      / RST 3 (DF)
4007 -   (90)      /  Compares HL and DE and sets flags
4008 -   (1C)      /
4009 -   (C3)   / RST 4 (E7) Compares contents of 40AF
400A -   (D9)   /  (number type) with 8. C clear indicates
400B -   (25)   /  double precision, S and C set for integer
                   C set and S clear for single precison
                   Z set for string
400C -   (C3)      /
400D -   (A2)      /  RST 5(EF)       // JP 4BA2 = OVERLAY
400E -   (4B)      /                  //           CONTROL
400F -   (C3)   /
4010 -   (B4)   /  RST 6(F7)          // JP 44B4 = TO
4011 -   (44)   /                     //           DEBUG
4012 -   (C3)      /
4013 -   (18)      /  RST 7(FF)       // JP 4518 = TO
4014 -   (45)      /                  //           INTERRUPTS
```

```
     * KEYBOARD DCB *
4015 -   (01)  /  DCB type 1
4016 -   (D8)     /  43D8 CONTAINS THE
4017 -   (43)     /  ADDRESS OF THE DRIVER
4018 -   (00)
4019 -   (00)
401A -   (00)
401B -   (4B)     /  KI (device name)
401C -   (49)

     * VIDEO DCB *
401D -   (07)  /  DCB type 7
401E -   (5B)     /  ADDRESS OF
401F -   (04)     /  THE DRIVER
4020 -   (8F)  /  The
4021 -   (3C)  /  CURSOR position
4022 -   (00)     /  Character in cursor
4023 -   (44)  /  DO (device name)
4024 -   (4F)

     * LINE PRINTER DCB *
4025 -   (06)  / DCB type 6
4026 -   (8D)     /  ADDRESS OF
4027 -   (05)     /  THE DRIVER
4028 -   (43)  / Lines per page
4029 -   (00)     / Line counter
402A -   (00)  (not used)
402B -   (50)     / PR (device name)
402C -   (52)

     * TRSDOS GO VECTOR *
402D -   (C3)     /  JUMP     (Calls SYS1 -
402E -   (00)     /  TO        beginning of DOS)
402F -   (44)     /  4400

4030 -   (3E)     /  LD A,A3              /  C7 = RST 0
4031 -   (A3)        (conditional jump if DEBUG active)
4032 -   (EF)     /  RST 5
4033 -   (C3)  /  JUMP
4034 -   (BB)  /  TO
4035 -   (44)  /  44BB

     * KEYBOARD MEMORY RAM *
4036 -   (00)
4037 -   (00)    (Row storage for debounce and keyboard
4038 -   (00)     rollover routines)
4039 -   (00)
403A -   (00)
403B -   (00)
403C -   (00)

     * CASSETTE PULSER and WIDE MODE VIDEO MEMORY *
403D -   (00)
403E -   (22)
403F -   (01)

4040 -   (00)     /  CLOCK = COUNT of 25 MSEC INTERRUPTS
```

```
     * BINARY TIME *
4041 -   (00)      /  SECONDS
4042 -   (00)      /  MINUTES
4043 -   (00)      /  HOURS

     * BINARY DATE *
4044 -   (00)      /  YEAR
4045 -   (00)      /  DAY
4046 -   (00)      /  MONTH

4047 -   (00)      (Storage for second overlay address)
4048 -   (52)

4049 -   (FF)      /  TOP MEMORY
404A -   (BF)      /  In TRSDOS and Disk BASIC

404B -   (3F)      /  INTERRUPT
404C -   (C0)      /  DATA

     * INTERRUPT ADDRESSES *
404D -   (37)
404E -   (45)
404F -   (37)
4050 -   (45)
4051 -   (37)
4052 -   (45)       (All point to 4537)
4053 -   (37)
4054 -   (45)
4055 -   (37)
4056 -   (45)
4057 -   (37)
4058 -   (45)
4059 -   (37)
405A -   (45)
405B -   (37)
405C -   (45)

     * DEBUG STORAGE inactive // active *
405D -   //  RST          //  Flag for type of display
405E -   //  ADDRESS      //  for use with RST "INTERRUPTS"
405F -       /  RST DATA
4060 -   //  RST          //     //  also - MEMORY POINTER
4061 -   //  ADDRESS      //     //         during display
4062 -       /  RST DATA
4063 -   (LOW)      /  DEBUG DISPLAY     //  POINTS to DEBUG
4064 -   (HIGH)     /  POINTER           //  REGISTER DATA

     * DEBUG REGISTER HOLDERS (OFF STACK) *
4065 -   F
4066 -   A
4067 -   C
4068 -   B
4069 -   E
406A -   D
406B -   L
406C -   H
406D -   F'
406E -   A'
406F -   C'
```

```
4070 -   B'
4071 -   E'
4072 -   D'
4073 -   L'
4074 -   H'
4075 -   IX (LOW)
4076 -   IX (HIGH)
4077 -   IY (LOW)
4078 -   IY (HIGH)
4079 -   SP (LOW)   //  OLD
407A -   SP (HIGH)  //  OLD
407B -   PC (LOW)
407C -   PC (HIGH)
407D -      //   Null in DOS  (Initialization stack pointer
407E -      //                 in Level II BASIC)
407F -      //
4300 -   (11)     //  TRACK of DRIVE 0
4301 -   (11)     //  TRACK of DRIVE 1
4302 -   (11)     //  TRACK of DRIVE 2
4303 -   (11)     //  TRACK of DRIVE 3
4304 -   (11)     //  DIRECTORY TRACK of DRIVE 0
4305 -   (11)     //  DIRECTORY TRACK of DRIVE 1
4306 -   (11)     //  DIRECTORY TRACK of DRIVE 2
4307 -   (11)     //  DIRECTORY TRACK of DRIVE 3
4308 -   (00)  /  Drive number in binary (Data for 4600)
4309 -   (01)  /  Bit SET indicates DRIVE number
430A -   (80/FF)
430B -   (44/66)
430C -   (FE)
430D -   (4B)
430E -   (87)     //  Hold last command to DOS
430F -   (01)  /  HIGHBIT = DEBUG FLAG
                  LOWBIT  = PROTECTION FLAG
                  BIT 4   = FLAG for SYS6
4310 -   (00)
4311 -   (11)
4312 -   (C3)     //  Jump director for position
                      commands to EF - DOS commands
4313 -   (4D/43)
4314 -   (4B/5D)
4315 -   (00/C3)  //  CLEAR while doing overlay
                      C3 when DEBUG FLAG SET
4316 -   (FF/0F)
4317 -   (FF/40)

     * KEYBOARD BUFFER (64 BYTES?) *
4318 -   (42)
4319 -   (41)
431A -   (53)
431B -   (49)
431C -   (43)
431D -   (0D)
431E -   (2C)
431F -   (49)
4320 -   (29)
4321 -   (0D)
4322 -   (FF)
  .   .   .
4328 -   (FF)
```

4330 - (FF)

4338 - (FF)

4340 - (00)

4348 - (00)

4350 - (00)

Data Organization on Disk

Data is organized on disk in a definite format. Sections of the format are separated by filler blocks to allow synchronization. The filler blocks usually contain 14 bytes of FF and 6 bytes of 00. Here is a typical file organization.

FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
FF	FF	FF	FF	00	00	00	00	00	00	Filler block
FE										Address mark
XX	00									Track number, separator
XX										Sector number
XX										Format multiplier
XX	XX									2 Byte checksum
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
FF	FF	FF	FF	00	00	00	00	00	0	Filler block
FB										Data mark
XX	XX	...	XX							Data
XX	XX									Checksum
FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
FF	FF	FF	FF	00	00	00	00	00	00	Filler block

Chapter 14

FD1771-01 Floppy Disk Formatter/Controller

Western Digital Corporation

FEATURES

- SOFT SECTOR FORMAT COMPATIBILITY
- AUTOMATIC TRACK SEEK WITH VERIFICATION
- READ MODE
  Single/Multiple Sector Write with Automatic Sector Search or Entire Track Read
  Selectable 128 Byte or Variable Length Sector
- WRITE MODE
  Single/Multiple Sector Write with Automatic Sector Search
  Entire Track Write for Diskette Formatting
- PROGRAMMABLE CONTROLS
  Selectable Track-to-Track Stepping Time
  Selectable Head Setting and Head Engage Times
  Selectable Three Phase or Step and Direction and Head Positioning Motor Controls
- SYSTEM COMPATIBILITY
  Double Buffering of Data 8-Bit Bi-Directional Bus for Data, Control and Status
  DMA or Programmed Data Transfers
  All Inputs and Outputs are TTL Compatible

APPLICATIONS

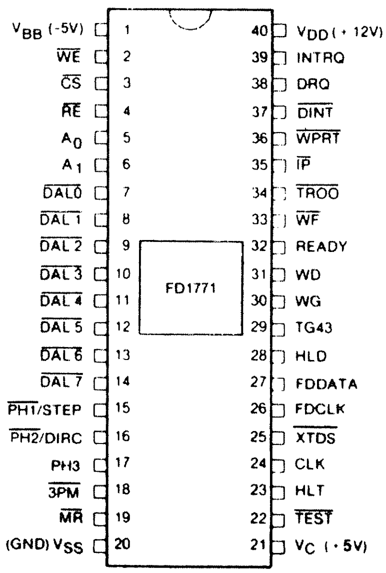
- FLOPPY DISK DRIVE INTERFACE
- SINGLE OR MULTIPLE DRIVE CONTROLLER/FORMATTER
- NEW MINI-FLOPPY CONTROLLER

GENERAL DESCRIPTION

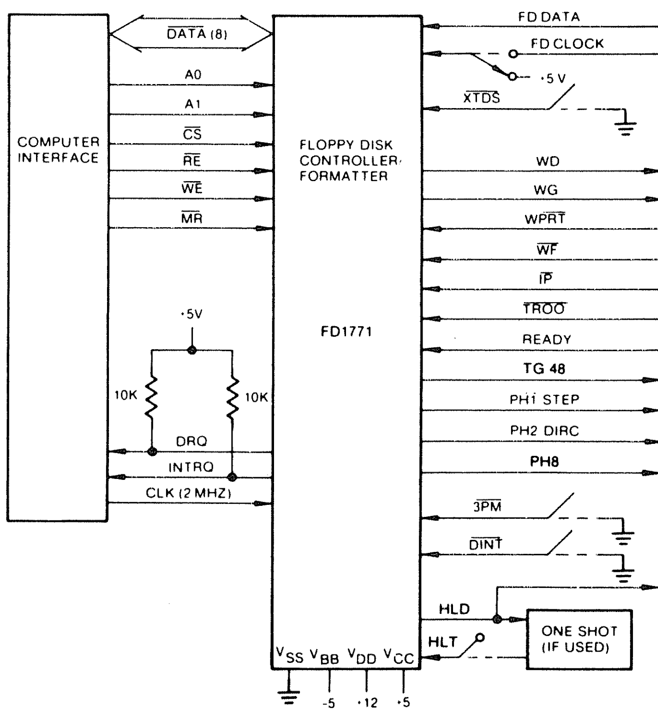
The FD1771 is a MOS/LSI device that performs the functions of a Floppy Disk Controller/Formatter. The device is designed to be included in the disk drive electronics, and contains a flexible interface organization that accommodates the interface signals from most drive manufacturers. The FD1771 is compatible with the IBM 3740 data entry system format.

The processor interface consists of an 8-bit bi-directional bus for data, status, and control word transfers. The FD1771 is set up to operate on a multiplexed bus with other bus-oriented devices.

The FD1771 is fabricated in N-channel Silicon Gate MOS technology and is TTL compatible on all inputs and outputs. The A and B suffixes are for ceramic and plastic packages, respectively.



PIN CONNECTIONS



FD1771 SYSTEM BLOCK DIAGRAM

ORGANIZATION

The Floppy Disk Formatter block diagram is illustrated below. The primary sections include the parallel processor interface and the Floppy Disk interface.

Data Shift Register: This 8-bit register assembles serial data from the Read Data input (FDDATA) during Read operations and transfers serial data to the Write Data output during Write operations.

Data Register: This 8-bit register is used as a holding register during Disk Read and Write operations. In Disk Read operations the assembled data byte is transferred in parallel to the Data Register from the Data Shift Register. In Disk Write operations information is transferred in parallel from the Data Register to the Data Shift Register.

PIN OUTS

Pin No.	Pin Name	Symbol	Function
1	Power Supplies	V_{BB}/NC	-5V
19	$\overline{\text{MASTER RESET}}$	$\overline{MR}$	A logic low on this input resets the device and loads "03" into the command register. The Not Ready (Status bit 7) is reset during $\overline{MR}$ ACTIVE. When $\overline{MR}$ is brought to a logic high, a Restore Command is executed, regardless of the state of the Ready signal from the drive.
20		V_{SS}	Ground
21		V_{CC}	+5V
40		V_{DD}	+12V
Computer Interface			
2	$\overline{\text{WRITE ENABLE}}$	$\overline{WE}$	A logic low on this input gates data on the DAL into the selected register when $\overline{CS}$ is low.
3	$\overline{\text{CHIP SELECT}}$	$\overline{CS}$	A logic low on this input selects the chip and enables computer communication with the device.
4	$\overline{\text{READ ENABLE}}$	$\overline{RE}$	A logic low on this input controls the placement of data from a selected register on the DAL when $\overline{CS}$ is low.
5, 6	REGISTER SELECT LINES	A_0, A_1	These inputs select the register to receive/transfer data on the DAL lines under $\overline{RE}$ and $\overline{WE}$ control: A_1 A_0 $\overline{RE}$ $\overline{WE}$ 0 0 Status Register Command Register 0 1 Track Register Track Register 1 0 Sector Register Sector Register 1 1 Data Register Data Register
7-14	$\overline{\text{DATA ACCESS LINES}}$	$\overline{\text{DAL0-DAL7}}$	Eight bit inverted bidirectional bus used for transfer of data, control, and status. This bus is a receiver enabled by $\overline{WE}$ or a transmitter enabled by $\overline{RE}$.
24	CLOCK	CLK	This input requires a free-running 2 MHz ± 1% square wave clock for internal timing reference.
38	DATA REQUEST	DRQ	This open drain output indicates that the DR contains assembled data in Read operations, or the DR is empty in Write operations. This signal is reset when serviced by the computer through reading or loading the DR in Read or Write operation, respectively. Use 10K pull-up resistor to +5.
39	INTERRUPT REQUEST	INTRQ	This open drain output is set at the completion or termination of any operation and is reset when a new command is loaded into the command register. Use 10K pull-up resistor to +5.
Floppy Disk Interface:			
15	$\overline{\text{Phase 1}}$/Step	$\overline{PH1}$/STEP	If the $\overline{3PM}$ input is a logic low the three-phase motor control is selected and $\overline{PH1}$, $\overline{PH2}$, and PH3 outputs

Pin No.	Pin Name	Symbol	Function
16	Phase $\overline{2}$/Direction	$\overline{\text{PH2}}$/DIRC	form a one active low signal out of three. $\overline{\text{PH1}}$ is active low after $\overline{\text{MR}}$. If the $\overline{\text{3PM}}$ input is a logic high the step and direction motor control is selected. The step output contains a 4 usec high signal for each step and the direction output is active high when stepping in; active low when stepping out.
17	Phase 3	PH3	
18	$\overline{\text{3-Phase Motor Select}}$	$\overline{\text{3PM}}$	
22	$\overline{\text{TEST}}$	$\overline{\text{TEST}}$	This input is used for testing purposes only and should be tied to +5V or left open by the user.
23	HEAD LOAD TIMING	HLT	The HLT input is sampled after 10 ms. When a logic high is sampled on the HLT input the head is assumed to be engaged.
25	$\overline{\text{EXTERNAL DATA SEPARATION}}$	$\overline{\text{XTDS}}$	A logic low on this input selects external data separation. A logic high or open selects the internal data separator.
26	FLOPPY DISK CLOCK (External Separation)	FDCLOCK	This input receives the externally separated clock when $\overline{\text{XTDS}}$ = 0. If $\overline{\text{XTDS}}$ = 1, this input should be tied to a logic high.
27	FLOPPY DISK DATA	FDDATA	This input receives the raw read disk data if $\overline{\text{XTDS}}$=1, or the externally separated data if $\overline{\text{XTDS}}$=0.
28	HEAD LOAD	HLD	The HLD output controls the loading of the Read-Write head against the media.
29	Track Greater than 43	TG43	This output informs the drive that the Read-Write head is positioned between tracks44-76. This output is valid only during Read and Write commands.
30	WRITE GATE	WG	This output is made valid when writing is to be performed on the diskette.
31	WRITE DATA	WD	This output contains both clock and data bits of 500 ns duration.
32	Ready	READY	This input indicates disk readiness and is sampled for a logic high before Read or Write commands are performed. If Ready is low, the Read or Write operation is not performed and an interrupt is generated. A Seek operation is performed regardless of the state of Ready. The Ready input appears in inverted format as Status Register bit 7.
33	$\overline{\text{WRITE FAULT}}$	$\overline{\text{WF}}$	This input detects wiring faults indications from the drive. When WG=1 and $\overline{\text{WF}}$ goes low, the current Write command is terminated and the Write Fault status bit is set. The $\overline{\text{WF}}$ input should be made inactive (high) when WG becomes inactive.
34	$\overline{\text{TRACK 00}}$	$\overline{\text{TR00}}$	This input informs the FD1771 that the Read-Write head is positioned over Track 00 when a logic low.
35	$\overline{\text{INDEX PULSE}}$	$\overline{\text{IP}}$	Input, when low for a minimum of 10 usec, informs the FD1771 when an index mark is encountered on the diskette.
36	$\overline{\text{WRITE PROTECT}}$	$\overline{\text{WPRT}}$	This input is sampled whenever a Write command is received. A logic low terminates the command and sets the Write Protect status bit.
37	$\overline{\text{DISK INITIALIZATION}}$	$\overline{\text{DINT}}$	The iput is sampled whenever a Write Track command is received. If $\overline{\text{DINT}}$=0, the operation is terminated and the Write Protect status bit is set.

When executing the Seek command, the Data Register holds the address of the desired Track position. This register can be loaded from the DAL and gated onto the DAL under processor control.

Track Register: This 8-bit register holds the track number of the current Read/Write head position. It is incremented by one every time the head is stepped in (towards track 76) and decremented by one when the head is stepped out (towards track 00). The contents of the register are compared with the recorded track number in the ID field during disk Read, Write, and Verify operations. The Track Register can be loaded from or transferred to the DAL. This Register should not be loaded when this device is busy.

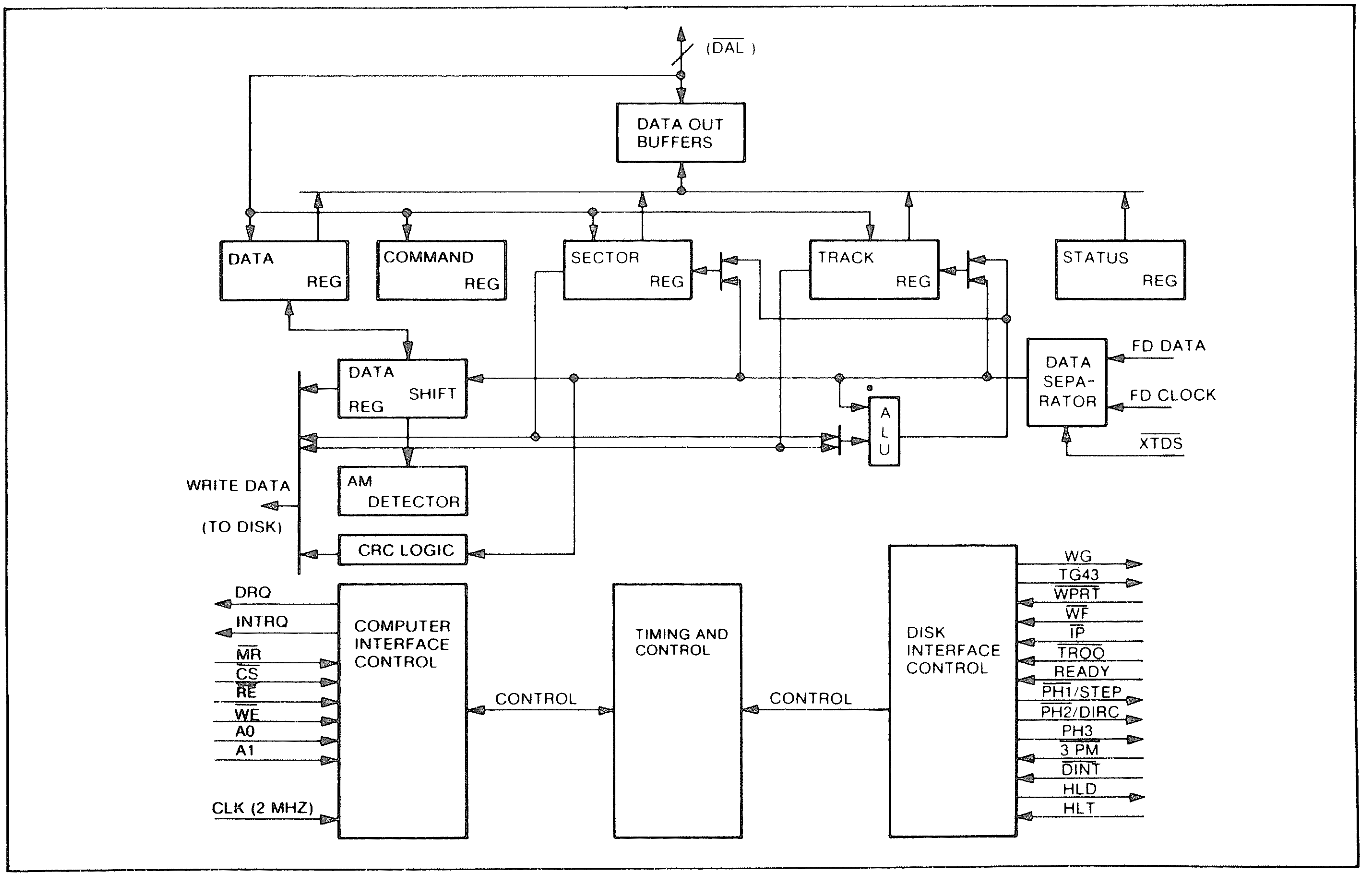
Sector Register (SR): This 8-bit register holds the address of the desired sector position. The contents of the register are compared with the recorded sector number in the ID field during disk Read or Write operations. The Sector Register contents can be loaded from or transferred to the DAL. This register should not be loaded when the device is busy.

Command Register (CR): This 8-bit register holds the command presently being executed. This register should not be loaded when the device is busy unless the execution of the current command is to be overridden. This latter action results in an interrupt. The command register can be loaded from the DAL, but not read onto the DAL.

Status Register (STR): This 8-bit register holds device Status information. The meaning of the Status bits are a function of the contents of the Command Register. This register can be read onto the DAL, but not loaded from the DAL.

CRC Logic: This logic is used to check or to generate the 16-bit Cyclic Redundancy Check (CRC). The polynomial is: $G(x) = x^{16} + x^{12} + x^5 + d1$.

The CRC includes all information starting with the address mark and up to the CRC characters. The CRC register is preset to ones prior to data being shifted through the circuit.



FD1771 BLOCK DIAGRAM

Arithmetic/Logic Unit (ALU): The ALU is a serial comparator, incrementer, and decrementer and is used for register modification and comparisons with the disk recorded ID field.

AM Detector: The Address Mark detector is used to detect ID, Data, and Index address marks during Read and Write operations.

Timing and Control: All computer and Floppy Disk Interface controls are generated through this logic. The internal device timing is generated from a 2.0 MHz external crystal clock.

PROCESSOR INTERFACE

The interface to the processor is accomplished through the eight Data Access Lines (DAL) and associated control signals. The DAL are used to transfer Data, Status, and Control words out of, or into the FD1771. The DAL are three-state buffers that are enabled as output drivers when Chip Select (CS) and Read Enable (RE) are active (low logic state) or act as input receivers when CS and Write Enable (WE) are active.

When transfer of data with the Floppy Disk Controller is required by the host processor, the device address is decoded and CS is made low. The least-significant address bits A1 and A0, combined with the signals RE during a Read operation or WE during a Write operation are interpreted as selecting the following registers:

A1-A0	READ ($\overline{RE}$)	WRITE ($\overline{WE}$)
0 0	Status Register	Command Register
0 1	Track Register	Track Register
1 0	Sector Register	Sector Register
1 1	Data Register	Data Register

During Director Memory Access (DMA) types of data transfers between the Data Register of the FD1771 and the Processor, the Data Request (DRQ) output is used in Data Transfer control. This signal also appears as status bit 1 during Read and Write operations.

On Disk Read operations the Data Request is activated (set high) when an assembled serial input byte is transferred in parallel to the Data Register. This bit is cleared when the Data Register is read by the processor. If the Data Register is read after one or more characters are lost, by having new data transferred into the register prior to processor readout, the Lost Data bit is set in the Status Register. The Read operation continues until the end of sector is reached.

On Disk Write operations the Data Request is activated when the Data Register transfers its contents to the Data Shift Register, and requires a new data byte. It is reset when the Data Register is loaded with new data by the processor. If new data is not loaded at the time the next serial byte is required by the Floppy Disk, a byte of zeroes is written on the diskette and the Lost Data bit is set in the Status Register.

The Lost Data bit and certain other bits in the Status Register will activate the interrupt request (INTRQ). The interrupt line is also activated with normal completion or abnormal termination of all controller operations. The INTRQ signal remains active until reset by reading the Status Register to the processor or by the loading of the Command Register. In addition, the INTRQ is generated if a Force Interrupt command condition is met.

FLOPPY DISK INTERFACE

The Floppy Disk interface consists of head positioning controls, write gate controls, and data transfers. A 2.0 MHz ± 1% square wave clock is required at the CLK input for internal control timing (may be 1.0 MHz for mini floppy).

HEAD POSITIONING

Four commands cause positioning of the Read-Write head (see Command Section). The period of each positioning step is specified by the r field in bits 1 and 0 of the command word. After the last directional step, an additional 10 milliseconds of head setting time takes place. The four programmable stepping rates are tabulated below.

The rates (shown in Table 1) can be applied to a Three-Phase Motor or a Step-Direction Motor through the device interface. When the $\overline{3PM}$ input is connected to ground, the device operates with a three phase motor control interface, with one active low signal per phase on the three output signals $\overline{PH1}$, $\overline{PH2}$, and PH3. The stepping sequence, when stepping in, is Phases 1-2-3-1, and when stepping out, Phases 1-3-2-1. Phase 1 is active low after Master Reset. Note: PH3 needs an inverter if used.

The Step-Direction Motor Control interface is activated by leaving input $\overline{3PM}$ open or connecting it to +5V. The Phase 1 pin $\overline{PH1}$ becomes a Step pulse of 4 microseconds width. The Phase 2 pin $\overline{PH2}$ becomes a direction control with a high voltage on this pin indicating a Step In, and a low voltage indicating a Step Out. The Direction output is valid a minimum of 24 μs prior to the activation of the Step pulse.

When a Seek, Step or Restore command is executed, an optional verification of Read-Write head position can be performed by setting bit 2 in the command word to a logic 1. The verification operation begins at the end of the 10 millisecond settling time after the head is loaded against the media. The track number from the first encountered ID Field is compared against the contents of the Track Register. If the track numbers compare and the ID Field Cyclic Redundancy Check (CRC) is correct, the verify operation is complete. If track comparison is not made but the CRC checks, an interrupt is generated, the Seek Error status (Bit 4) is set and the Busy status bit is reset.

Table 1. STEPPING RATES

r1 r0	1771-X1 CLK=2 MHz TEST=1	1771-X1 CLK=1 MHz TEST=1	1771 or -X1 CLK=2 MHz TEST=0	1771 or -X1 CLK=1 MHz TEST=0
0 0	6ms	12ms	Approx. 400us*	Approx. 800us*
0 1	6ms	12ms		
1 0	10ms	20ms		
1 1	20ms	40ms		

*For exact times consult WDC.

The Head Load (HLD) output controls the movement of the read/write head against the disk for data recording or retrieval. It is activated at the beginning of a Read, Write (E flag On) or Verify operation, or a Seek or Step operation with the head load bit, h, a logic one remains activated until the third index pulse following the last operation which uses the read/write head. Reading or Writing does not occur until a minimum of 10 msec delay after the HLD signal is made active. If executing the type 2 commands with the E flag off, there is no 10 msec delay and the head is assumed to be engaged. The delay is determined by sampling of the Head Load Timing (HLT) input after 10 msec. A high state input, generated from the Head Load output transition and delayed externally, identifies engagement of the head against the disk. In the Seek and Step commands, the head is loaded at the start of the command execution when the h bit is a logic one. In a verify command the head is loaded after stepping to the destination track on the disk whenever the h bit is a logic zero.

DISK READ OPERATION

The 2.0 MHz external clock provided to the device is internally divided by 4 to form the 500 kHz clock rate for data transfer. When reading data from a diskette this divider is synchronized to transitions of the Read Data (FDDATA) input. When a transition does not occur on the 500 kHz clock active state, the clock divider circuit injects a clock to maintain a continuous 500 kHz data clock. The 500 kHz data clock is further divided by 2 internally to separate the clock and information bits. The divider is phased to the information by the detection of the address mark.

In the internal data read and separation mode the Read Data input toggles from one state to the opposite state for each logic one bit of clock or information. This signal can be derived from the amplified, differentiated, and sliced Read Head signal, or by the output of a flip-flop toggling on the Read Data pulses. This input is sampled by the 2 MHz clock to detect transitions.

The chip can also operate on externally separated data, as supplied by methods such as Phase Lock loop, One Shots, or variable frequency oscillators. This is accomplished by grounding the External Data Separator (XTDS) INPUT. When the Read Data input makes a high-to-low transition, the information input to the FDDATA line is clocked into the Data Shift Register. The assembled 8-bit data from the Data Shift Register are then transferred to the Data Register.

The normal sector length for Read or Write operations with the IBM 3740 format is 128 bytes. This format or binary multiples of 128 bytes will be adopted by setting a logic 1 in Bit 3 of the Read and Write commands. Additionally, a variable sector length feature is provided which allows an indicator recorded in the ID Field to control the length of the sector. Variable sector lengths can be read or written in Read or Write commands, respectively, by setting a logic 0 in Bit 3 of the command word. The sector length indicator specifies the number of 16 byte groups of 16 x N, where N is equal to 1 to 256 groups. An indicator of all zeroes is interpreted as 256 sixteen byte groups.

DISK WRITE OPERATION

After data is loaded from the processor into the Data Register, and is transferred to the Data Shift Register, data will be shifted serially through the Write Data (WD) output. Interlaced with each bit of data is a positive clock pulse of 0.5 microsecond duration. This signal may be used to externally toggle a flip-flop to control the direction of Write Current flow.

When writing is to take place on the diskette the Write Gate (WG) output is activated, allowing current to flow into the Read/Write head. As a precaution to erroneous writing, the first data byte must be loaded into the Data Register in response to a Data Request from the FD1771 before the Write Gate signal can be activated.

Writing is inhibited when the Write Protect input is a logic low, in which case any Write command is immediately terminated, an interrupt is generated and the Write Protect status bit is set. The Write Fault input, when activated, signifies a writing fault condition detected in disk drive electronics such as failure to detect write current flow when the Write Gate is activated. On detection of this fault the FD1771 terminates the current command, and sets the Write Fault bit (bit 5) in the Status Word. The Write Fault input should be made inactive when the Write Gate output becomes inactive.

Whenever a Read or Write command is received the FD1771 samples the READY input. If this input is logic low the command is not executed and an interrupt is generated. The Seek or Step commands are performed regardless of the state of the READY input.

COMMAND DESCRIPTION

The FD1771 will accept and execute eleven commands. Command words should only be loaded in the Command Register when the Busy status bit is off (status bit 0). The one exception is the Force Interrupt command. Whenever a command is being executed, the Busy status bit is set. When a command is completed, an interrupt is generated and the Busy status bit is reset. The Status Register indicates whether the completed command encountered an error or was fault-free. For ease of discussion, commands are divided into four types. Commands and types are summarized in Table 2.

TYPE 1 COMMANDS

The Type 1 Commands include the RESTORE, SEEK, STEP, STEP-IN, and STEP-OUT commands. Each of the Type 1 Commands contain a rate field ($r_0 r_1$), which determines the stepping motor rate as defined in Table 1, below.

The Type 1 Commands contain a head load flag (h) which determines if the head is to be loaded at the

Table 2. COMMAND SUMMARY

		BITS							
TYPE	COMMAND	7	6	5	4	3	2	1	0
I	Restore	0	0	0	0	h	V	r1	r0
I	Seek	0	0	0	1	h	V	r1	r0
I	Step	0	0	1	u	h	V	r1	r0
I	Step In	0	1	0	u	h	V	r1	r0
I	Step Out	0	1	1	u	h	V	t1	r0
II	Read Command	1	0	0	m	b	E	0	0
II	Write Command	1	0	1	m	b	E	a1	a0
III	Read Address	1	1	0	0	0	E	0	0
III	Read Track	1	1	1	0	0	1	0	s
III	Write Track	1	1	1	1	0	1	0	0
IV	Force Interrupt	1	1	0	1	I_3	I_2	I_1	I_4

Note: Bits shown in TRUE form.

Table 3. FLAG SUMMARY

TYPE 1
h=Head Load flag (Bit 3)
h = 1, Load head at beginning h = 0, Do not load head at beginning
V=Verify flag (Bit 2)
V = 1, Verify on last track V = 0, No verify
r1r0=Stepping motor rate (Bits 1-0)
Refer to Table 1 for rate summary
u=Update flag (Bit 4)
u = 1, Update Track register u = 0, No update

Table 4. FLAG SUMMARY

TYPE II
m = Multiple Record flag (Bit 4)
m=0, Single Record m=1, Multiple Records
b = Block length flag (Bit 3)
b=1, IBM format (128 to 1024 bytes) b=0, Non-IBM format (16 to 4096 bytes)
a1a0=Data Address Mark (Bits 1-0)
a1a0 = 00,FB (Data Mark) a1a0 = 01,FA (User defined) a1a0 = 10,F9 (User defined) a1a0 = 11,F8 (Deleted Data Mark)

Table 5. FLAG SUMMARY

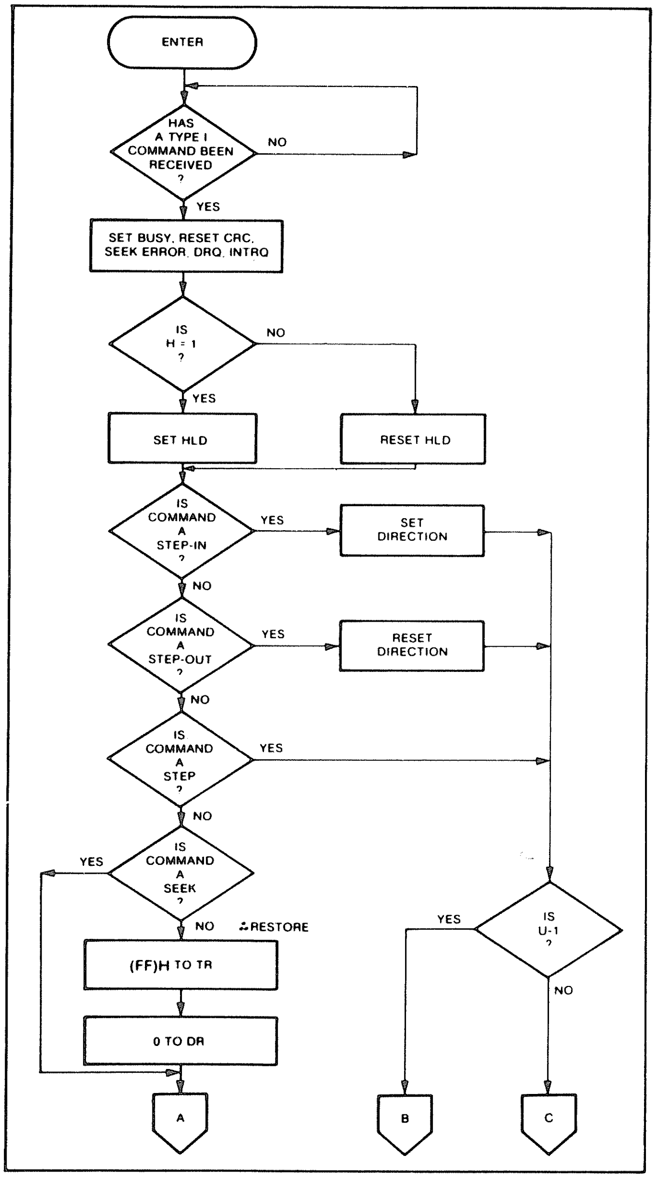
TYPE III
s = Synchronize flag (Bit 0) $\overline{s}$=0, Synchronize to AM $\overline{s}$=1, Do Not Synchronize to AM
TYPE IV
Ii = Interrupt Condition flags (Bits 3-0) I_0=1, Not Ready to Ready Transition I_1=1, Ready to Not Ready Transition I_2=1, Index Pulse I_3=1, Immediate interrupt E = Enable HLD to 10 msec Delay
E=1, Enable HLD, HLT and 10 msec Delay E=0, Head is assumed Engaged and there is no 10 msec Delay

beginning of the command. If h=1, the head is loaded at the beginning of the command (HLD output is made active). If h=0, HLD is deactivated. Once the head is loaded, the head will remain engaged until the FD1771 receives a command that specifically disengages the head. If the FD1771 does not receive any commands after two revolutions of the disk, the head will be automatically disengaged (HLD made inactive). The Head Load Timing Input is sampled after a 10 ms delay, wahen reading or writing on the disk is to occur.

The Type 1 Commands also contain a verification (V) flag which determines if a verification operation is to take place on the destination track. If V=1, a verification is performed; if V=0, no verification is performed.

During verification, the head is loaded and after an internal 10 ms delay, the HLT input is sampled. When HLT is active (logic true), the first encountered ID field is read off the disk. The track address of the ID Field is then compared to the Track Register, if there is a match and a valid ID CRC, the verification is complete, an interrupt is generated and the BUSY status bit is reset. If there is not a match but there is valid ID CRC, an interrupt is generated, the Seek Error status bit (Status Bit 4) is set and the BUSY status bit is reset. If there is a match but not a valid CRC the CRC error status bit is set (Status Bit 3), and the next encountered ID Field is read from the disk for the verification operation. If an ID Field with a valid CRC cannot be found after two revolutions of the disk, the FD1771 terminates the operation and sends an interrupt (INTRQ).

The STEP, STEP-IN, and STEP-OUT commands contain an UPDATE flag (U). When U=1, the track register is updated by one for each step. When U=0, the track register is not updated.



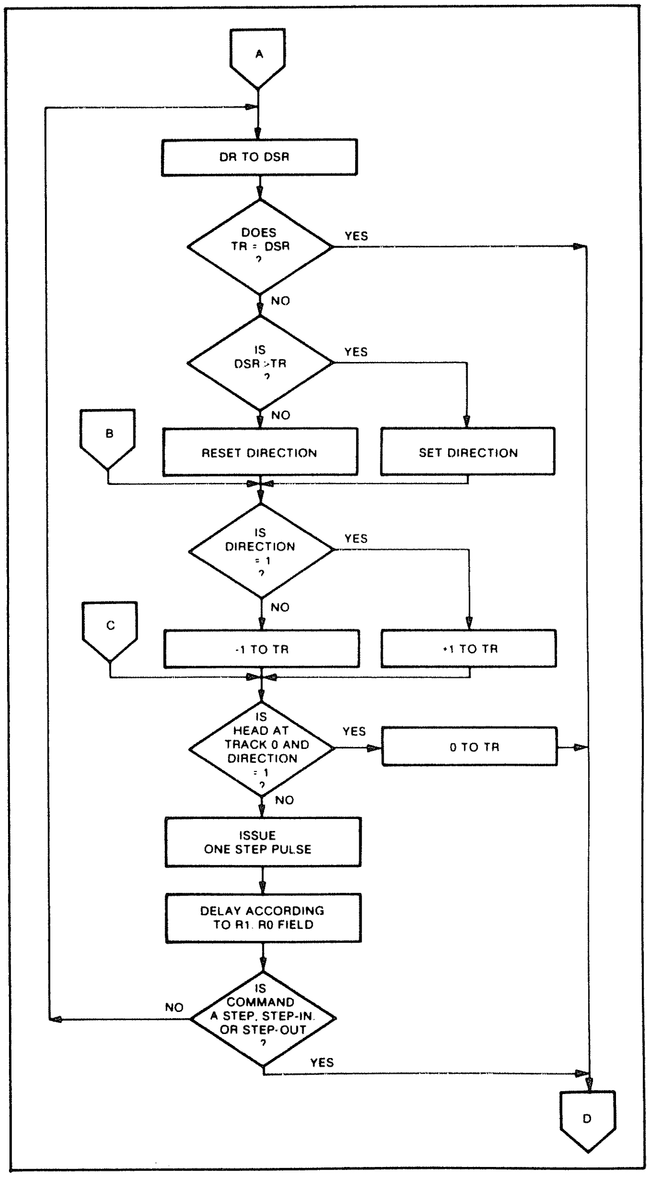
TYPE I COMMAND FLOW

RESTORE (SEEK TRACK 0)

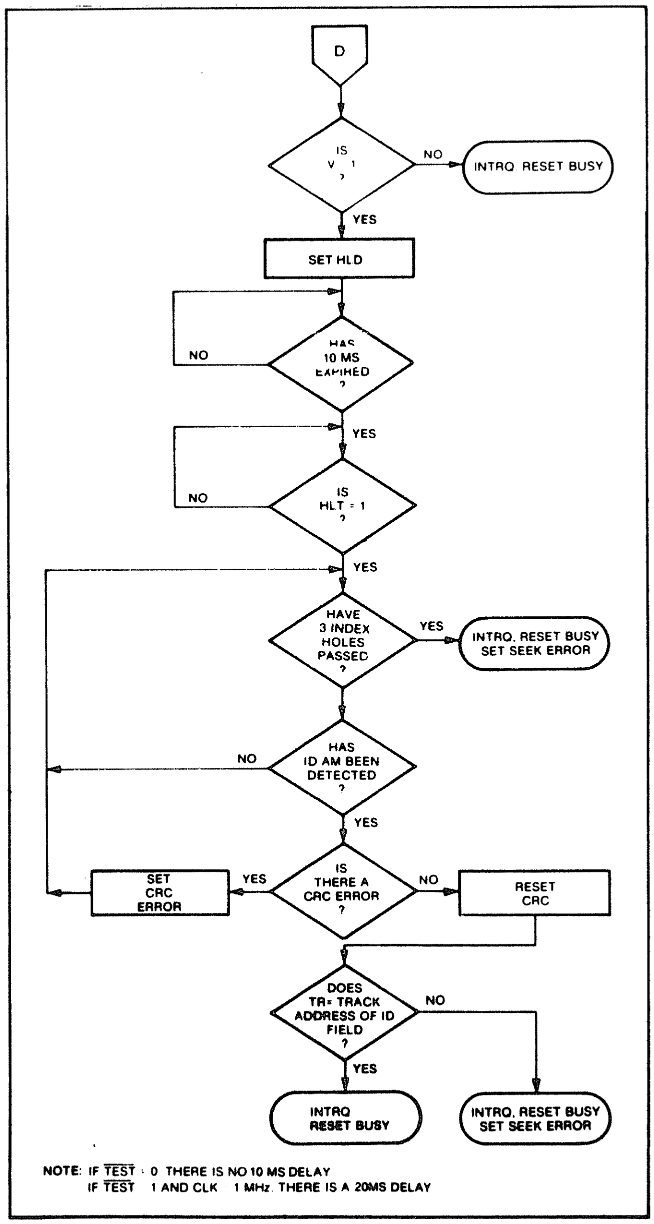
Upon receipt of this command the Track 00 ($\overline{TR00}$) input is sampled. If $\overline{TR00}$ is active low indicating the Read-Write head is positioned over track 0, the Track Register is loaded with zeroes and an interrupt is generated. If $\overline{TR00}$ is not active low, stepping pulses (pins 15 to 17) at a rate specified by the r1r0 field are issued until the $\overline{TR00}$ input is activated. At this time the TR is loaded with zeroes and an interrupt is generated. If the $\overline{TR00}$ input does not go active low after 255 stepping pulses, the FD1771 terminates operation, interrupts, and sets the Seek error status bit. Note that the RESTORE command is executed when $\overline{MR}$ goes from an active to an inactive state. A verification operation takes place if the V flag is set. The h bit allows the head to be loaded at the start of command.

SEEK

The command assumes that the Track Register contains the track number of the current position of the Read-Write head and the Data Register contains the desired track number. The FD1771 will update the Track register and issue stepping pulses in the appropriate direction until the contents of the Track register are equal



TYPE I COMMAND FLOW



TYPE I COMMAND FLOW

to the contents of the data register (the desired track location). A verification operation takes place if the V flag is on. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command.

STEP

Upon receipt of this command, the FD1771 issues one stepping pulse to the disk drive. The stepping motor direction is the same as in the previous step command. After a delay determined by the r_1r_0 field, a verification takes place if the V flag is on. If the U flag is on, the TR is updated. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command.

STEP-IN

Upon receipt of this command, the FD1771 issues one stepping pulse in the direction towards track 76. If the U flag is on, the Track Register is incremented by one. After a delay determined by the r_1r_0 field, a verification takes place if the V flag is on. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command.

STEP-OUT

Upon receipt of this command, the FD1771 issues one stepping pulse in the direction towards track 0. If the U flag is on, the TR is decremented by one. After a delay determined by the r_1r_0 field, a verification takes place if the V flag is on. The h bit allows the head to be loaded at the start of the command. An interrupt is generated at the completion of the command.

TYPE II COMMANDS

The Type II Commands include the Read Sector(s) and Write Sector(s) commands. Prior to loading the Type II command into the COMMAND REGISTER, the computer must load the Sector Register with the desired sector number. Upon receipt of the Type II command, the Busy status bit is set. If the E flag=1 (this is the normal case), HLD is made active and HLT is sampled after a 10 msec delay. If the E flag is 0, the head is assumed to be engaged and there is no 10 msec delay. The ID field and the Data Field format are shown below.

When an ID field is located on the disk, the FD1771 compares the track number of the ID field with the Track Register. If there is not a match, the next encountered ID field is read and a comparison is again made. If there was a match, the next encountered ID field is compared with the Sector Register. If there is not a Sector match, the next encountered ID field is read off the disk and comparisons again made. If the ID field CRC is correct, the data field is then located and will be either written into, or read from depending on the command. The FD1771 must find an ID field with a track number, Sector number, and CRC within two revolutions of the disk; otherwise, the Record Not Found status bit is set (Status bit 3) and the command is terminated with an interrupt.

Each of the Type II Commands contain a (b) flag which in conjunction with the sector length field contents of the ID determines the length (number of characters) of the Data field.

For IBM 3740 compatibility, the b flag should equal 1. The numbers of bytes in the data field (sector) is the 128 x 2^n where n = 0,1,2,3.

GAP	ID AM	TRACK NUMBER	ZERO	SECTOR NUMBER	SECTOR LENGTH	CRC 1	CRC 2	GAP	DATA AM	DATA FIELD	CRC 1	CRC 2
ID FIELD									DATA FIELD			

IDAM = ID Address Mark — DATA = $(FE)_{16}$ CLK = $(C7)_{16}$

Data AM = Data Address Mark — DATA = (F8, F9, FA, or FB), CLK = $(C7)_{16}$

For b = 1

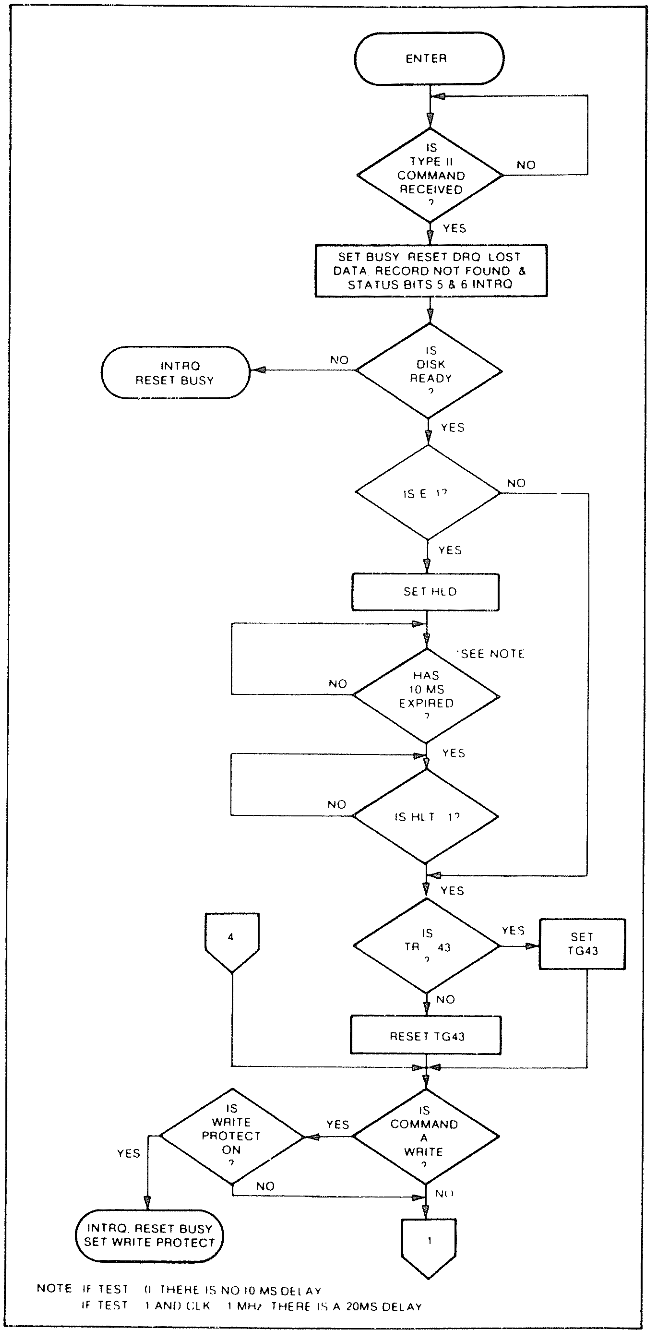
Sector Length Field (Hex)	Number of Bytes in Sector (Decimal)
00	128
01	256
02	512
03	1024

When the b flag equals zero, the sector length field (n) multiplied by 16 determines the number of bytes in the sector or data field as shown below.

For b = 0

Sector Length Field (Hex)	Number of Bytes in Sector (Decimal)
01	16
02	32
03	48
04	64
*	*
*	*
*	*
FF	4080
00	4096

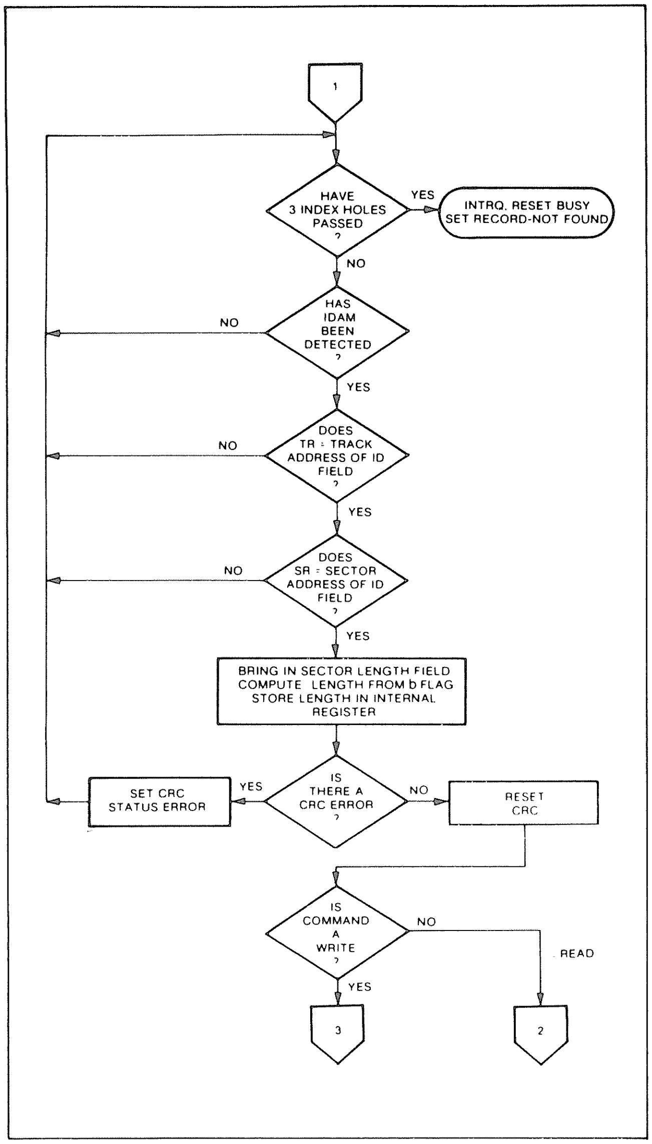
Each of the Type II commands also contain a (m) flag which determines if the multiple records (sectors) are to be read or written, depending upon the command. If m=0 a single sector is read or written and an interrupt is generated at the completion of the command. If m=1, mulitple records are read or written with the sector register internally updated so that an address verification can occur on the next record. The FD1771 will continue to read or write multiple records and update the sector register until the sector register exceeds the number of sectors on the track or until the Force Interrupt command is loaded into the command register, which terminated the command and generates an interrupt.



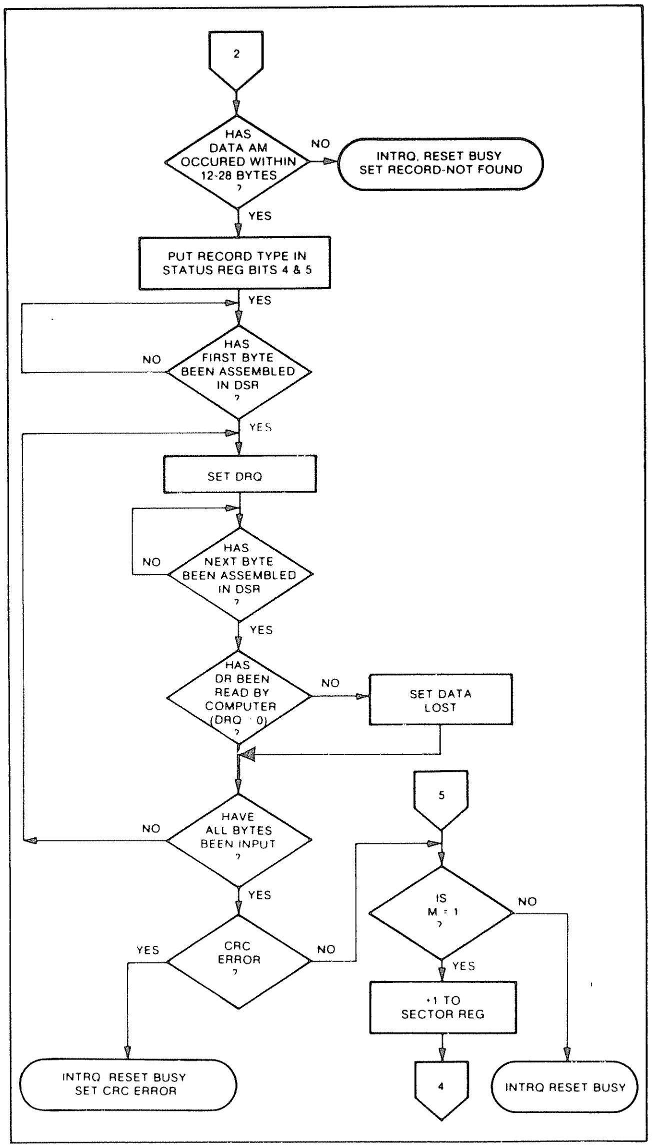
TYPE II COMMAND FLOW

READ COMMAND

Upon receipt of the Read command, the head is loaded, the BUSY status bit set, and when an ID field is encountered that has the correct track number, correct sector number, and correct CRC, the data field is presented to the computer. The Data Address Mark of the data field must be found within 28 bytes of the correct field; if not, the Record Not Found status bit is set and the operation is terminated. When the first character or byte of the data field has been shifted through the DSR, it is transferred to the DR, and DRQ is generated. When the next byte is accumulated in the DSR, it is transferred to



TYPE II COMMAND FLOW



TYPE II COMMAND FLOW

the DR and another DRQ is generated. If the computer has not read the previous contents of the DR before a new character is transferred that character is lost and the Lost Data status bit is set. This sequence continues until the complete data field has been input to the computer. If there is a CRC error at the end of the data field, the CRC error status bit is set, and the command is terminated (even if it is a multiple record command).

At the end of the Read operation, the type of Data Address Mark encountered in the data field is recorded in the Status Register (Bits 5 and 6) as shown below.

Status Bit 5	Status Bit 6	Data AM (Hex)
0	0	FB
0	1	FA
1	0	F9
1	1	F8

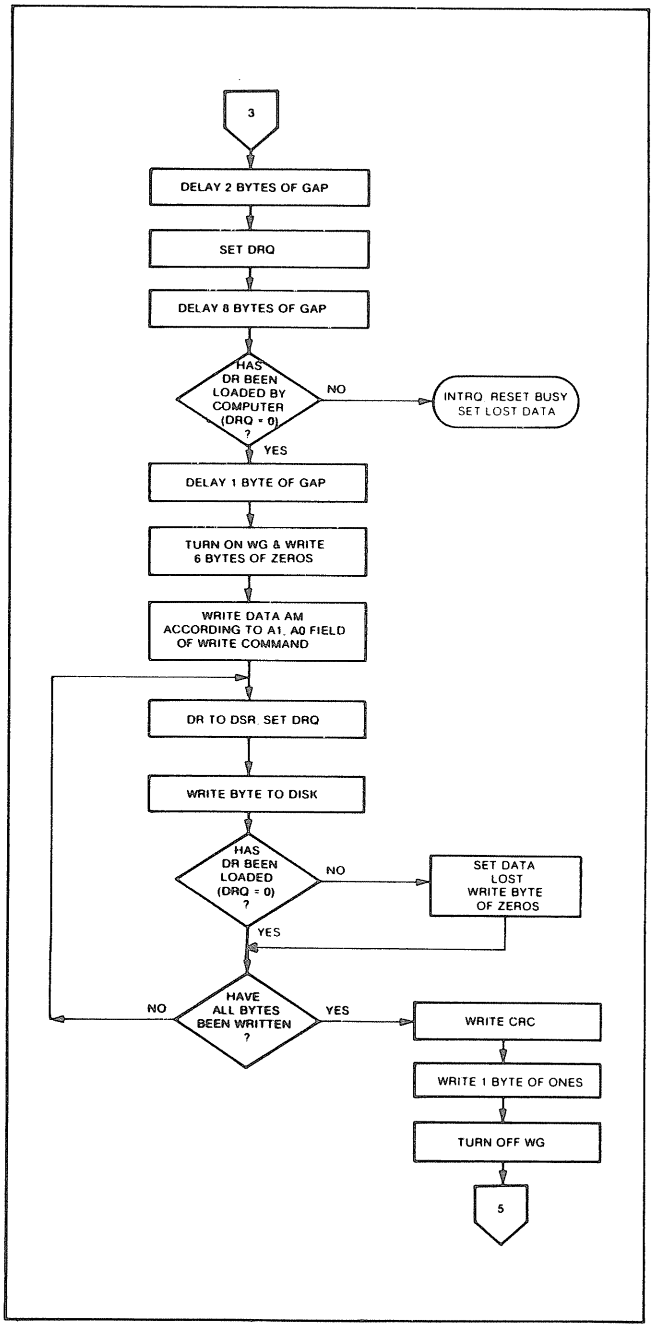
WRITE COMMAND

Upon receipt of the Write command, the head is loaded (HLD active) and the BUSY status bit is set. When an ID field is encountered that has the correct track number, correct sector number, and correct CRC, a DRQ is generated. The FD1771 counts off 11 bytes from the CRC field and the Write Gate (WG) output is made active if the DRQ is serviced (i.e., the DR has been loaded by the computer). If DRQ has not been serviced, the command is terminated and the Lost Data status bit is set. If the DRQ has been serviced, the WG is made active and six bytes of zeros are then written on the disk. At this time the Data Address Mark is then written on the disk as determined by the a_1a_0 field of the command as shown on next page.

The FD1771 then writes the data field and generates DRQs to the computer. If the DRQ is not serviced in time for continuous writing the Lost Data status bit is set and a byte of zeros is written on the disk. The command is not terminated. After the last data byte has been written on the

a1	a0	Data Mark (Hex)	Clock Mark (Hex)
0	0	FB	C7
0	1	FA	C7
1	0	F9	C7

disk, the two-byte CRC is computed internally and written on the disk followed by one byte gap of logic ones. The WG outputs is then deactivated.



TYPE II COMMAND FLOW

TYPE III COMMANDS

READ ADDRESS

Upon receipt of the Read Address command, the head is loaded and the BUSY Status bit is set. The next encountered ID field is then read in from the disk, and the six data bytes of the ID field are assembled and transferred to the DR, and a DRQ is generated for each byte. The six bytes of the ID field are shown below.

TRACK ADDR	SIDE NUMBER	SECTOR ADDRESS	SECTOR LENGTH	CRC 1	CRC 2
1	2	3	4	5	6

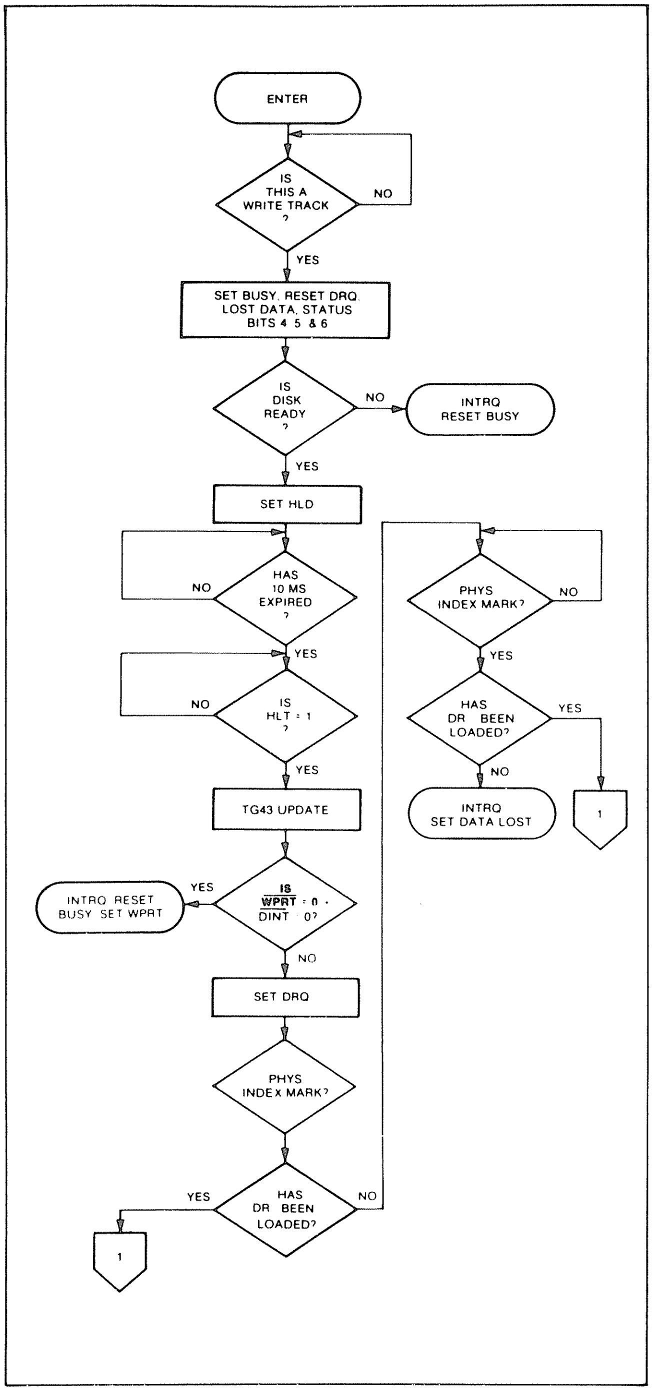
Although the CRC characters are transferred to the computer, the FD1771 checks for validity and the CRC error status bit is set if there is a CRC error. The Sector Address of the ID field is written into the Sector Register. At the end of the operation an interrupt is generated and the BUSY Status is reset.

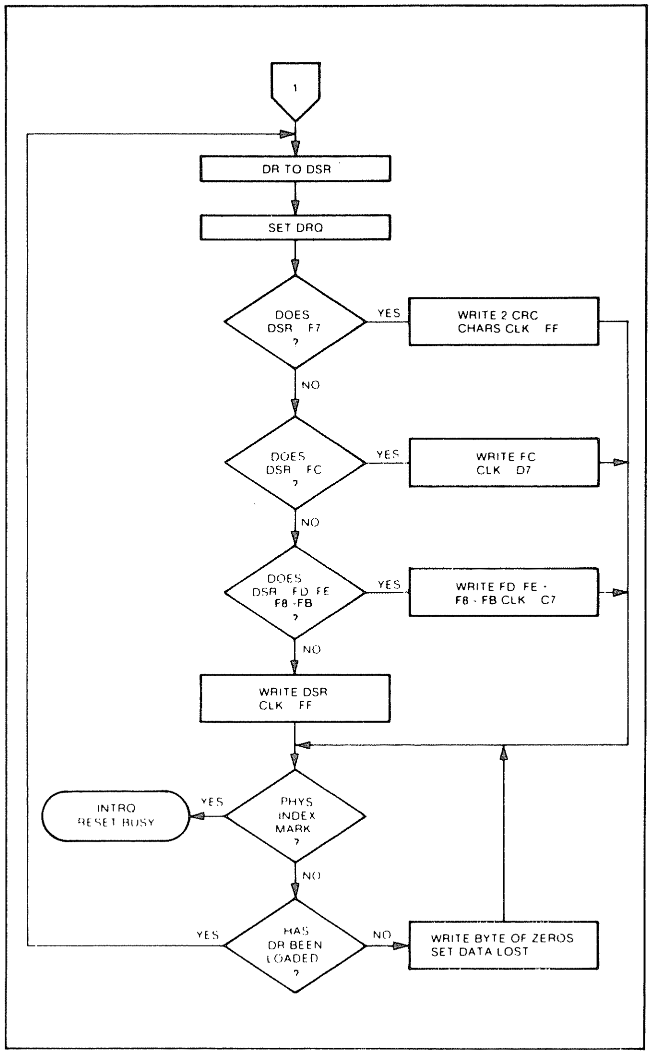
READ TRACK

Upon receipt of the Read Track command, the head is loaded and the BUSY status bit is set. Reading starts with the leading edge of the first encountered index mark and continues until the next index pulse. As each byte is assembled it is transferred to the Data Register and the Data Request is generated for each byte. No CRC checking is performed. Gaps are included in the input data stream. If bit 0(S) of the command is a 0, the accumulation of bytes is synchronized to each Address Mark encountered. Upon completion of the command, the interrupt is activated.

WRITE TRACK

Upon receipt of the Write Track command, the head is loaded and the BUSY status bit is set. Writing starts with the leading edge of the first encountered index pulse and continues until the next index pulse, at which time the interrupt is activated. The Data Request is activated immediately upon receiving the command, but writing will not start until after the first byte has been loaded into the Data Register. If the DR has not been loaded by the time the index pulse is encountered the operation is terminated making the device Not Busy, the Lost Data status bit is set, and the Interrupt is activated. If a byte is not present in the DR when needed, a byte of zeros is substituted. Address Marks and CRC characters are written on the disk by detecting certain data byte patterns in the outgoing data stream as shown in the table below . The CRC generator is initialized when any data byte from F8 to FE is about to be transferred from the DR to the DSR.





TYPE III COMMAND WRITE TRACK

CONTROL BYTES FOR INITIALIZATION

DATA PATTERN (HEX)	INTERPRETATION	CLOCK MARK (HEX)
F7	Write CRC Character	FF
F8	Data Address Mark	C7
F9	Data Address Mark	C7
FA	Data Address Mark	C7
FB	Data Address Mark	C7
FC	Index Address Mark	D7
FD	Spare	
FE	ID Address Mark	C7

TYPE III COMMAND WRITE TRACK

The Write Track Command will not execute if the $\overline{DINT}$ input is grounded; instead, the Write Protect status bit is set and the interrupt is activated. Note that one F7 pattern generates two CRC characters.

TYPE IV COMMAND

Force Interrupt

This command can be loaded into the command register at any time. If there is a current command under execution (BUSY status bit set), the command will be terminated and an interrupt will be generated when the condition specified in the I_0 through I_3 field is detected. The interrupt conditions are shown below:

I_0=Not-Ready-To-Ready Transition
I_1=Ready-To-Not-Ready Transition
I_2=Every Index Pulse
I_3=Immediate Interrupt (Requires reset, see note)

Note: If I_0 - I_3 = 0, there is no interrupt generated but the current command is terminated and busy is reset. This is the only command that will clear the immediate interrupt.

STATUS DESCRIPTION

Upon receipt of any command, except the Force Interrupt command, the Busy Status bit is set and the rest of the status bits are updated or cleared for the new command. If the Force Interrupt Command is received when there is a current command under execution, the Busy status bit is reset, and the rest of the status bits are unchanged. If the Force Interrupt command is received when there is not a current command under execution, the Busy Status bit is reset and the rest of the status bits are updated or cleared in this case, Status reflects the Type I commands.

The format of the Status Register is shown below.

(BITS)							
7 S7	6 S6	5 S5	4 S4	3 S3	2 S2	1 S1	0 S0

Status varies according to the type of command executed as shown in Table 6.

Table 6. STATUS REGISTER SUMMARY

BIT	ALL TYPE I COMMANDS	READ ADDRESS	READ	READ TRACK	WRITE	WRITE TRACK
S7	NOT READY	NOT READY	NOT READY	NOT READY	NOT READY	NOT READY
S6	WRITE PROTECT	0	RECORD TYPE	0	WRITE PROTECT	WRITE PROTECT
S5	HEAD ENGAGED	0	RECORD TYPE	0	WRITE FAULT	WRITE FAULT
S4	SEEK ERROR	ID NOT FOUND	RECORD NOT FOUND	0	RECORD NOT FOUND	0
S3	CRC ERROR	CRC ERROR	CRC ERROR	0	CRC ERROR	0
S2	TRACK 0	LOST DATA	LOST DATA	LOST DATA	LOST DATA	LOST DATA
S1	INDEX	DRQ	DRQ	DRQ	DRQ	DRQ
S0	BUSY	BUSY	BUSY	BUSY	BUSY	BUSY

STATUS FOR TYPE I COMMANDS

BIT	NAME	MEANING
S7	NOT READY	This bit when set indicates the drive is not ready. When reset it indicates that the drive is ready. This bit is an inverted copy of the READY input and logically "ored" with MR.
S6	PROTECTED	When set, indicates Write Protect is activated. This bit is an inverted copy of $\overline{\text{WRPT}}$ input.
S5	HEAD LOADED	When set, it indicates the head is loaded and engaged. This bit is a logical "and" of HLD and HLT signals.
S4	SEEK ERROR	When set, the desired track was not verified. This bit is reset to 0 when updated.
S3	CRC ERROR	When set, there was one or more CRC errors encountered on an unsuccessful track verification operation. This bit is reset to 0 when updated.
S2	TRACK 00	When set, indicates Read-Write head is positioned to Track 0. This bit is an inverted copy of the $\overline{\text{TR00}}$ input.
S1	INDEX	When set, indicates index mark detected from drive. This bit is an inverted copy of the IP input.
S0	BUSY	When set, command is in progress. When reset, no command is in progress.

STATUS BITS FOR TYPE II AND III COMMANDS

BIT	NAME	MEANING
S7	NOT READY	This bit when set indicates the drive is not ready. When reset, it indicates that the drive is ready. This bit is an inverted copy of the READY input and "ored" with MR. The TYPE II and III Commands will not execute unless the drive is ready.
S6	RECORD TYPE/WRITE PROTECT	On Read Record: It indicates the MSB of record-type code from data field address mark. On Read Track: Not Used. On any Write Track: It indicates a Write Protect. This bit is reset when updated.
S5	RECORD TYPE/WRITE FAULT	On Read Record: It indicates the LSB of record-type code from data field address mark. On Read Track: Not Used: On any Write Track: It indicates a Write Fault. This bit is reset when updated.
S4	RECORD NOT FOUND	When set, it indicates that the desired track and sector were not found. This bit is reset when updated.
S3	CRC ERROR	If S4 is set, an error is found in one or more ID fields; otherwise it indicates error in data field. This bit is reset when updated.
S2	LOST DATA	When set, it indicates the computer did not respond to DRQ in one byte time. This bit is reset to zero when updated.
S1	DATA REQUEST	This bit is a copy of the DRQ output. When set, it indicates the DR is full on a Read operation or the DR is empty on a Write operation. This bit is reset to zero when up
S0	BUSY	When set, command is under execution. When reset, no command is under execution.

FORMATTING THE DISK (Refer to section on Type III Commands for flow diagrams.)

Formatting the disk is a relatively simple task when operating programmed I/O or when operating under DMA control with a large amount of memory. When operating under DMA with limited amount of memory, formatting is a more difficult task. This is because gaps as well as data must be provided at the computer interface.

Formatting the disk is accomplished by positioning the R/W head over the desired track number and issuing the Write Track command. Upon receipt of the Write Track command, the FD1771 raises the Data Request signal. At this point in time, the user loads the Data Register with desired data to be written on the disk. For every byte of information to be written on the disk, a Data Request is generated. This sequence continues from one index mark to the next index mark. Normally, whatever data pattern appears in the Data Register is written on the disk with a clock mark of $(FF)_{16}$. However, if the FD1771 detects a data pattern on F7 through FE in the Data Register, this is interpreted as data address marks with missing clocks or CRC generation. For instance, an FE pattern will be interpreted as an ID address mark (DATA-FE,CLK-C7) and the CRC will be initialized. An F7 pattern will generate two CRC characters. As a consequence, the patterns F7 through FE must not appear in the gaps, data fields, or ID fields. Also, CRCs must be generated by an F7 pattern.

Disks may be formatted in IBM 3740 formats with sector lengths of 128,256,512, or 1024 bytes, or may be formatted in non-IBM format with sector lengths of 16 to 4096 bytes in 16-byte increments. IBM 3740 at the present time only defines two formats. One format with 128 bytes/sector and the other with 256 bytes/sector. The next section deals with the IBM 3740 format with 128 bytes/sector followed by a section of non-IBM formats.

IBM 3740 Formats—128 Bytes/Sector

The IBM format with 128 bytes/sector is depicted in the Track Format figure on the following page. In order to create this format, the user must issue the Write Track command, and load the data register with the following values. For every byte to be written there is one data request.

Number of Bytes	Hex Value of Byte Written
40	00 or FF
6	00
1	FC (Index Mark)
* 26	00 or FF
6	00
1	FE (ID Address Mark)
1	Track Number (0 through 4C)
1	00
1	Sector Number (1 through 1A)
1	00
1	F7 (two CRCs written)
11	00 or FF
6	00
1	FB (Data Address Mark)
128	Data (IBM uses E5)
1	F7 (two CRCs written)
27	00 or FF
247 * *	00 or FF

* Write bracketed field 26 times

* * Continue writing until FD1771 interrupts out. Approximately 247 bytes.

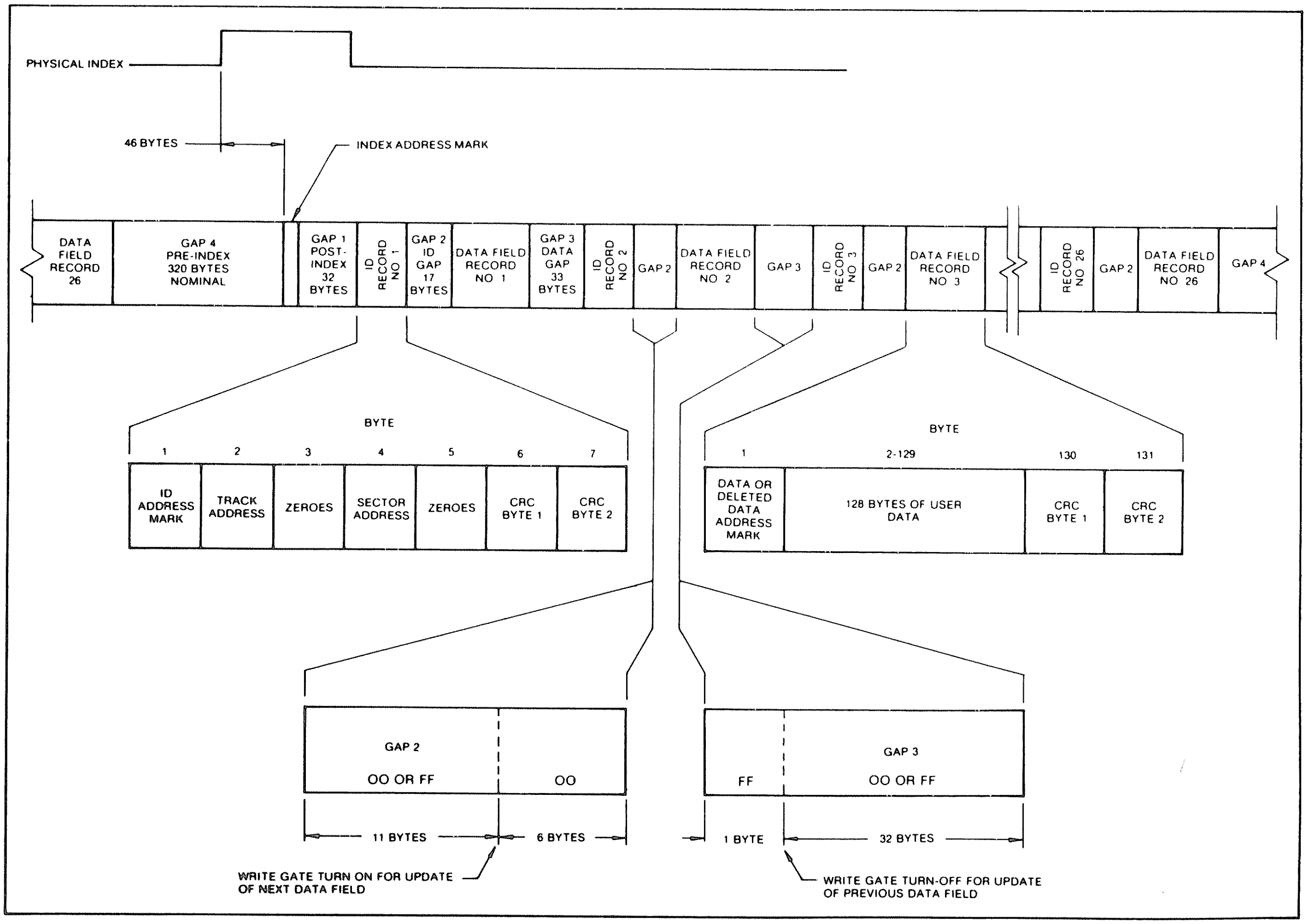
Non-IBM Formats

Non-IBM formats are very similar to the IBM formats except a different algorithm is used to ascertain the sector length from the sector length byte in the ID field. This permits a wide range of sector lengths from 16 to 4096 bytes. Refer to Section V, Type II commands with b flag equal to zero. Note that F7 through FE must not appear in the sector length byte of the ID field.

In formatting the FD1771, only two requirements regarding GAP sizes must be met. GAP 2 (i.e., the gap between the ID field and data field) must be 17 bytes of which the last 6 bytes must be zero and that every address mark be preceded by at least one byte of zeros. However, it is recommended that every GAP be at least 17 bytes long with 6 bytes of zeros. The FD1771 does not require the index address mark (i.e., DATA=FC,CLK=D7) and need not be present.

References:

1) IBM Diskette OEM Information GA21-9190-1.
2) SA900 IBM Compatibility Reference Manual—Shugart Associates.



TRACK FORMAT

ELECTRICAL CHARACTERISTICS

Maximum Ratings

V_{DD} with respect to V_{BB} (Ground)	+20 to -0.3V
Max Voltage to any input with respect to V_{BB}	+20 to -0.3V
Operating Temperature	0°C to 70°C
Storage Temperature	-55°C to +125°C

OPERATING CHARACTERISTICS (DC)

T_A=0°C to 70°C, V_{DD}=+12.0V ±.6V
V_{BB}=5.0 .5V, V_{SS}=0V, V_{CC}=+5V ±.25V
I_{DD}=10 ma Nominal, I_{CC}=30 ma Nominal
I_{BB}=0.4 μa Nominal

Symbol	Characteristic	Min.	Type.	Max.	Units	Conditions
I_{LI}	Input Leakage			10	μA	$V_{IN}=V_{DD}$
I_{LO}	Output Leakage			10	μA	$V_{OUT}=V_{DD}$
V_{IH}	Input High Voltage	2.6			V	
V_{IL}	Input Low Voltage (All Inputs)			0.8	V	
V_{OH}	Output High Voltage	2.8			V	I_O=-100μA
V_{OL}	Output Low Voltage			0.45	V	I_O=-1.0 mA

TIMING CHARACTERISTICS

TA=0°C to 70°C, V_{DD}=+12V ±.6V
V_{BB}=-5V .25V, V_{SS}=0V, V_{CC}=+5V ±.25V

NOTE: Timings are given for 2 MHz Clock. For those timings noted, values will double when chip is operated at 1 MHz. Use 1 MHz when using mini-floppy.

Read Operations

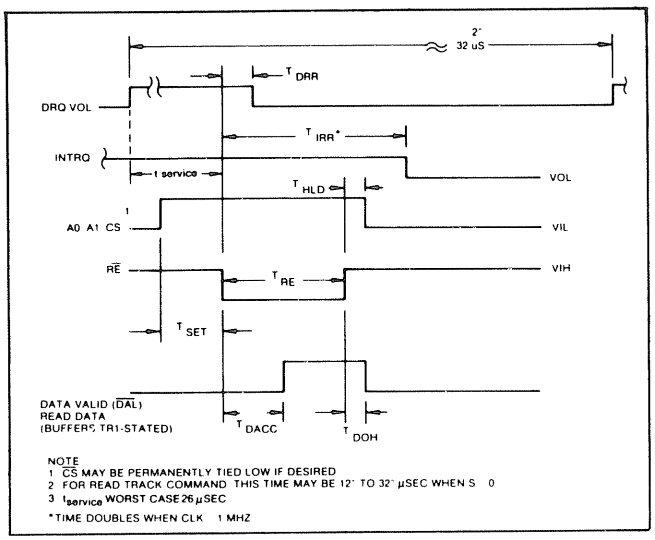
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TSET	Setup ADDR and CS to $\overline{RE}$	100			nsec	
THLD	Hold ADDR and CS from $\overline{RE}$	10			nsec	
TRE	$\overline{RE}$ Pulse Width	450			nsec	CL = 25 pf
TDRR	DRQ Reset from $\overline{RE}$			750	nsec	
TIRR	INTRQ Reset from $\overline{RE}$			3000	nsec	
TDACC	Data Access from $\overline{RE}$			450	nsec	CL = 25 pf
TDOH	Data Hold from $\overline{RE}$	50		150	nsec	CL = 25 pf

Write Operations

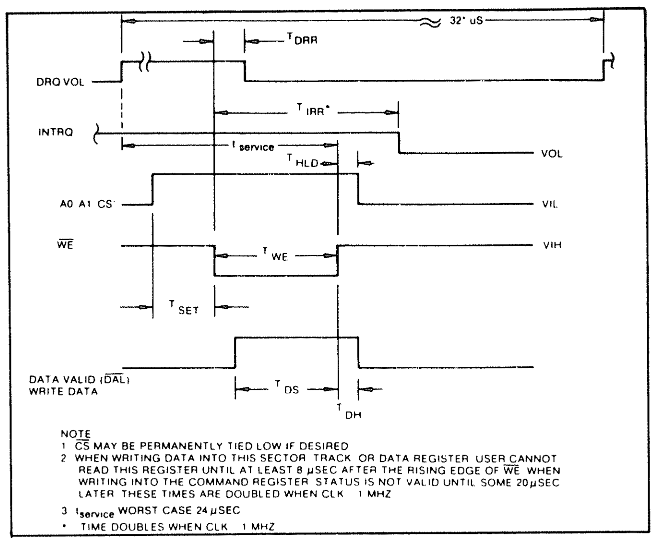
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TSET	Setup ADDR and CS to $\overline{WE}$	100			nsec	
THLD	Hold ADDR and CS from $\overline{WE}$	10			nsec	
TWE	$\overline{WE}$ Pulse Width	450	300		nsec	
TDRR	DRQ Reset from $\overline{WE}$			750	nsec	
TIRR	INTRQ Reset from $\overline{WWE}$			300	nsec	See Note
TDS	Data Setup to $\overline{WE}$	350			nsec	
TDH	Data Hold from $\overline{WE}$	150			nsec	

External Data Separation ($\overline{XTDS}$ = 0)

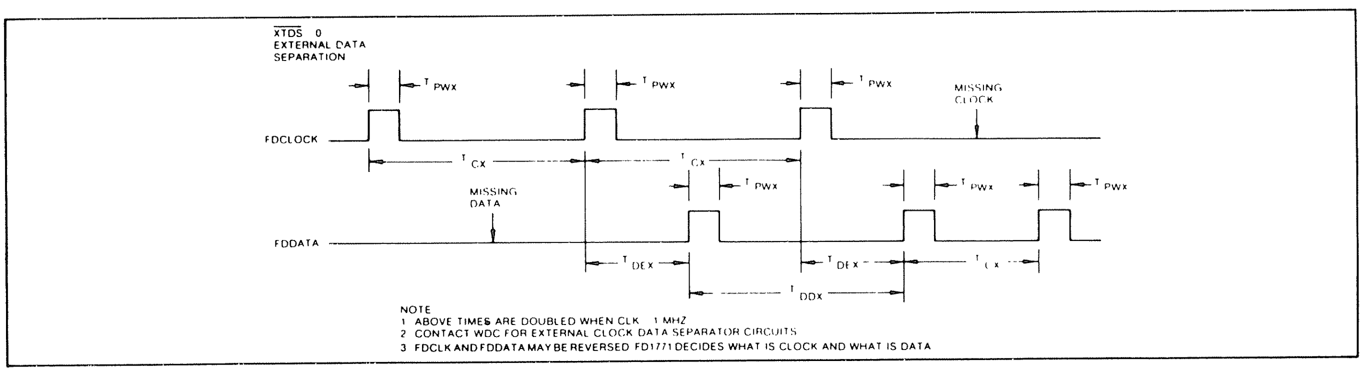
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TPWX	Pulse Width Read Data & Read Clock	150		350	nsec	
TCX	Clock Cycle External	2500			nsec	
TDEX	Data to Clock	500			nsec	
TDDX	Data to Data Cycle	2500			nsec	



READ ENABLE TIMING



WRITE ENABLE TIMING



Internal Data Separation (XTDS =1) READ TIMING ($\overline{XTDS}$ = 0)

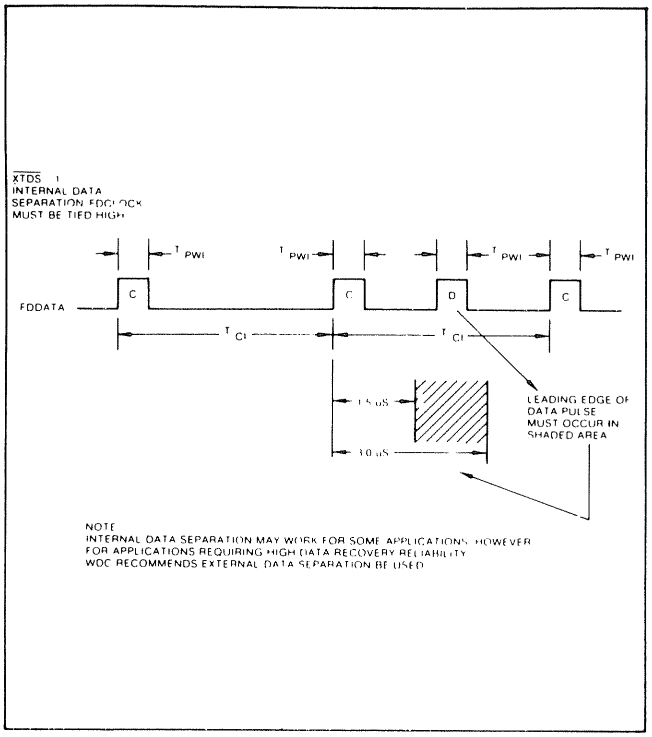
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TPWI	Pulse Width Data and Clock	150		1000	nsec	
TCI	Clock Cycle Internal	3500		5000	nsec	

Write Data Timing

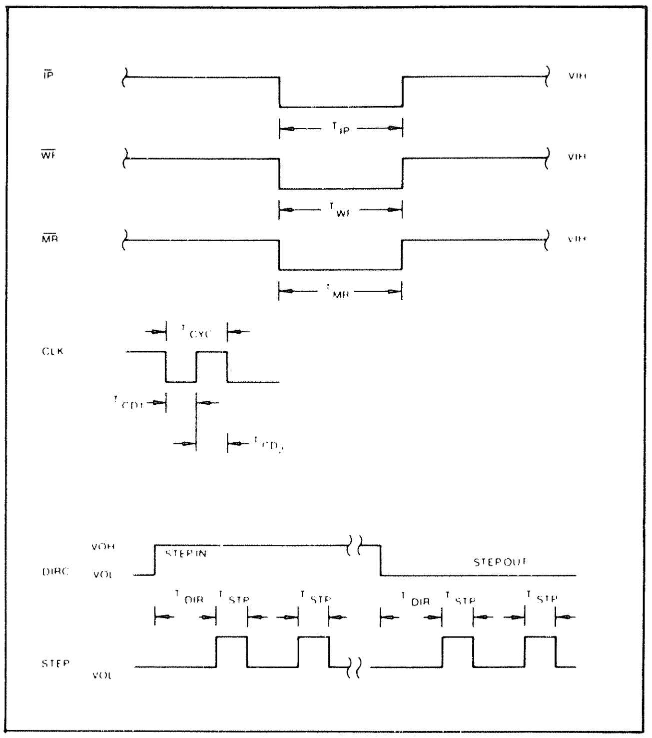
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TWGD	Write Gate to Data		1200		nsec	300 nsec ± CLK tolerance
TPWW	Pulse Width Write Data	500		600	nsec	
TCDW	Clock to Data		2000		nsec	± CLK tolerance
TCS	Clock Cycle Write		4000		nsec	± CLK tolerance
TWGH	Write Gate Hold to Data	0		100	nsec	

Miscellaneous Timing

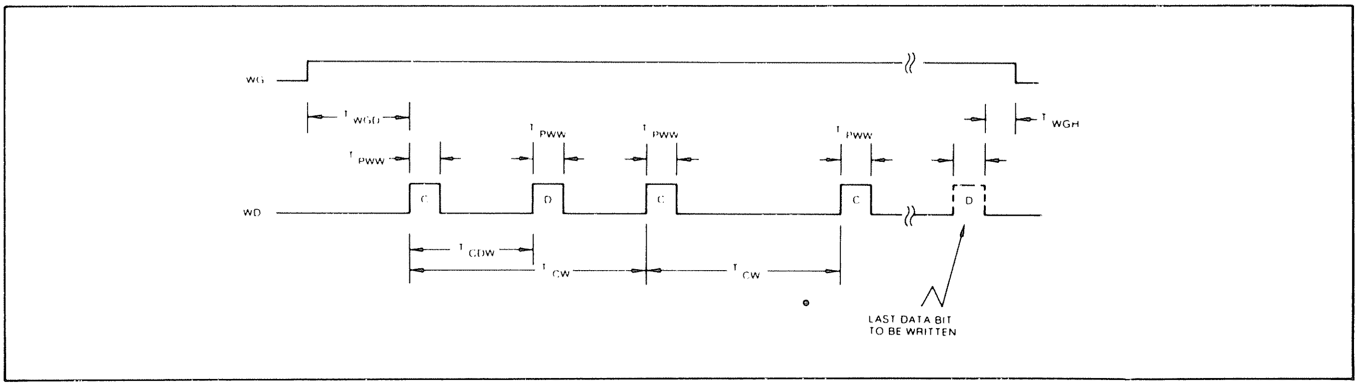
Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
TCD1	Clock Duty	175			nsec	2 MHz ± 1% See Note
TCD2	Clock Duty	210			nsec	
TSTP	Step Pulse Output	3800		4200	nsec	
TDIR	Direct Setup to Step	24			nsec	
TMR	Master Reset Pulse Width	10			usec	These times doubled when CLK = 1 MHz
TIP	Index Pulse Width	10			usec	
TWF	Write Fault Pulse Width	10			usec	



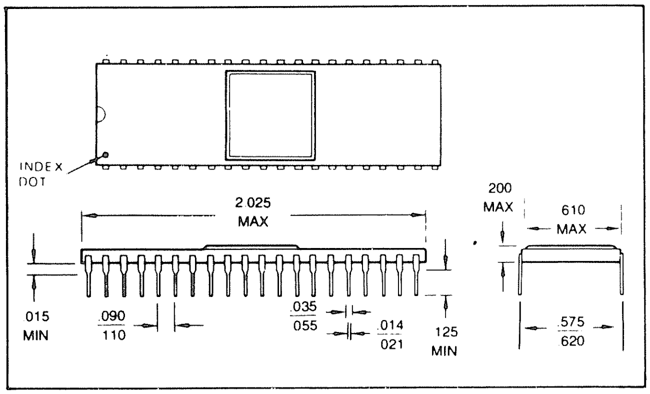
READ TIMING ($\overline{XTDS}$ = 1)



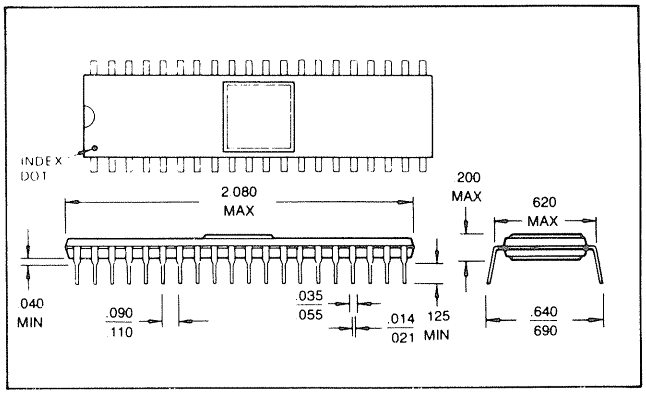
MISCELLANEOUS TIMING



WRITE DATA TIMING



FD1771A CERAMIC PACKAGE



FD1771B PLASTIC PACKAGE

WESTERN DIGITAL CORPORATION
3128 Redhill Avenue, Box 2180
Newport Beach, CA 92663
(714) 557-3550, TWX 910-595-1139

Appendix

APPENDIX

TABLE 1

DEC	HEX	EXT	Z-80 OP CODE	ASCII	TRS-80 CONTROLS
000	00		NOP	NUL	
001	01	NN NN	LD BC,NN	SOH	
002	02		LD (BC),A	STX	
003	03		INC BC	ETX	
004	04		INC B	EOT	
005	05		DEC B	ENQ	
006	06	NN	LD B,N	ACK	
007	07		RLCA	BEL	
008	08		EX AF,AF'	BS	BACKSPACE
009	09		ADD HL,BC	HT	
010	0A		LD A,(BC)	LF	LINE FEED
011	0B		DEC BC	VT	VERTICAL TAB
012	0C		INC C	FF	FORM FEED
013	0D		DEC C	CR	CARRIAGE RETURN
014	0E	NN	LD C,N	SO	CURSOR ON
015	0F		RRCA	SI	CURSOR OFF
016	10	DD	DJNZ DIS	DLE	
017	11	NN NN	LD DE,NN	DC1	
018	12		LD (DE),A	DC2	
019	13		INC DE	DC3	
020	14		INC D	DC4	
021	15		DEC D	NAK	
022	16	NN	LD D,N	SYN	
023	17		RLA	ETB	CONVERT TO 32 CHAR
024	18	DD	JR DIS	CAN	BACKSPACE CURSOR
025	19		ADD HL,DE	EM	ADVANCE CURSOR
026	1A		LD A,(DE)	SUB	CRT DOWN LINE
027	1B		DEC DE	ESC	CRT UP LINE
028	1C		INC E	FS	HOME CURSOR
029	1D		DEC E	GS	BEGIN LINE
030	1E	NN	LD E,N	RS	ERASE LINE
031	1F		RRA	US	CLEAR FRAME
032	20	DD	JR NZ,DIS	SP	(SPACE)
033	21	NN NN	LD HL,NN	!	
034	22	NN NN	LD (NN),HL	"	
035	23		INC HL	#	
036	24		INC H	$	
037	25		DEC H	%	
038	26	NN	LD H,N	&	
039	27		DAA	'	
040	28	DD	JR Z,DIS	(	
041	29		ADD HL,HL	)	
042	2A	NN NN	LD HL,(NN)	*	
043	2B		DEC HL	+	
044	2C		INC L	,	
045	2D		DEC L	-	
046	2E	NN	LD L,N	.	
047	2F		CPL	/	
048	30	DD	JR NC,DIS	0	
049	31	NN NN	LD SP,NN	1	
050	32	NN NN	LD (NN),A	2	
051	33		INC SP	3	
052	34		INC (HL)	4	
053	35		DEC (HL)	5	
054	36	NN	LD (HL),N	6	
055	37		SCF	7	
056	38	DD	JR C,DIS	8	
057	39		ADD HL,SP	9	
058	3A	NN NN	LD A,(NN)	:	
059	3B		DEC SP	;	
060	3C		INC A	<	
061	3D		DEC A	=	
062	3E	NN	LD A,N	>	
063	3F		CCF	?	

TABLE 2

DEC	HEX	Z-80 OP CODE	ASCII	TRS-80 BASIC
064	40	LD B,B	@	
065	41	LD B,C	A	
066	42	LD B,D	B	
067	43	LD B,E	C	
068	44	LD B,H	D	
069	45	LD B,L	E	
070	46	LD B,(HL)	F	
071	47	LD B,A	G	
072	48	LD C,B	H	
073	49	LD C,C	I	
074	4A	LD C,D	J	
075	4B	LD C,E	K	
076	4C	LD C,H	L	
077	4D	LD C,L	M	
078	4E	LD C,(HL)	N	
079	4F	LD C,A	O	
080	50	LD D,B	P	
081	51	LD D,C	Q	
082	52	LD D,D	R	
083	53	LD D,E	S	
084	54	LD D,H	T	
085	55	LD D,L	U	
086	56	LD D,(HL)	V	
087	57	LD D,A	W	
088	58	LD E,B	X	
089	59	LD E,C	Y	
090	5A	LD E,D	Z	
091	5B	LD E,E	↑	
092	5C	LD E,H	↓	
093	5D	LD E,L	←	
094	5E	LD E,(HL)	→	
095	5F	LD E,A	—	
096	60	LD H,B	@	
097	61	LD H,C	a	
098	62	LD H,D	b	
099	63	LD H,E	c	
100	64	LD H,H	d	
101	65	LD H,L	e	
102	66	LD H,(HL)	f	
103	67	LD H,A	g	
104	68	LD L,B	h	
105	69	LD L,C	i	
106	6A	LD L,D	j	
107	6B	LD L,E	k	
108	6C	LD L,H	l	
109	6D	LD L,L	m	
110	6E	LD L,(HL)	n	
111	6F	LD L,A	o	
112	70	LD (HL),B	p	
113	71	LD (HL),C	q	
114	72	LD (HL),D	r	
115	73	LD (HL),E	s	
116	74	LD (HL),H	t	
117	75	LD (HL),L	u	
118	76	HALT	v	
119	77	LD (HL),A	w	
120	78	LD A,B	x	
121	79	LD A,C	y	
122	7A	LD A,D	z	
123	7B	LD A,E		
124	7C	LD A,H		
125	7D	LD A,L		
126	7E	LD A,(HL)		
127	7F	LD A,A	DEL	

TABLE 3

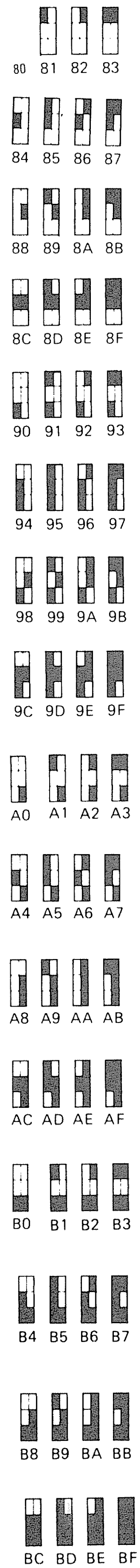
DEC	HEX	Z-80 OP CODE	GRAPHIC	TRS-80 BASIC
128	80	ADD A,B		END
129	81	ADD A,C		FOR
130	82	ADD A,D		RESET
131	83	ADD A,E		SET
132	84	ADD A,H		CLS
133	85	ADD A,L		CMD
134	86	ADD A,(HL)		RANDOM
135	87	ADD A,A		NEXT
136	88	ADC A,B		DATA
137	89	ADC A,C		INPUT
138	8A	ADC A,D		DIM
139	8B	ADC A,E		READ
140	8C	ADC A,H		LET
141	8D	ADC A,L		GOTO
142	8E	ADC A,(HL)		RUN
143	8F	ADC A,A		IF
144	90	SUB B		RESTORE
145	91	SUB C		GOSUB
146	92	SUB D		RETURN
147	93	SUB E		REM
148	94	SUB H		STOP
149	95	SUB L		ELSE
150	96	SUB (HL)		TRON
151	97	SUB A		TROFF
152	98	SBC A,B		DEFSTR
153	99	SBC A,C		DEFINT
154	9A	SBC A,D		DEFSNG
155	9B	SBC A,E		DEFDBL
156	9C	SBC A,H		LINE
157	9D	SBC A,L		EDIT
158	9E	SBC A,(HL)		ERROR
159	9F	SBC A,A		RESUME
160	A0	AND B		OUT
161	A1	AND C		ON
162	A2	AND D		OPEN
163	A3	AND E		FIELD
164	A4	AND H		GET
165	A5	AND L		PUT
166	A6	AND (HL)		CLOSE
167	A7	AND A		LOAD
168	A8	XOR B		MERGE
169	A9	XOR C		NAME
170	AA	XOR D		KILL
171	AB	XOR E		LSET
172	AC	XOR H		RSET
173	AD	XOR L		SAVE
174	AE	XOR (HL)		SYSTEM
175	AF	XOR A		LPRINT
176	B0	OR B		DEF
177	B1	OR C		POKE
178	B2	OR D		PRINT
179	B3	OR E		CONT
180	B4	OR H		LIST
181	B5	OR L		LLIST
182	B6	OR (HL)		DELETE
183	B7	OR A		AUTO
184	B8	CP B		CLEAR
185	B9	CP C		CLOAD
186	BA	CP D		CSAVE
187	BB	CP E		NEW
188	BC	CP H		TAB(
189	BD	CP L		TO
190	BE	CP (HL)		FN
191	BF	CP A		USING

TABLE 4

DEC	HEX	EXT	Z-80 OP CODE	TAB	TRS-80 BASIC
192	C0		RET NZ	0	VARPTR
193	C1		POP BC	1	USR
194	C2	NN NN	JP NZ,NN	2	ERL
195	C3	NN NN	JP NN	3	ERR
196	C4	NN NN	CALL NZ,NN	4	STRING$
197	C5		PUSH BC	5	INSTR
198	C6	NN	ADD A,N	6	POINT
199	C7		RST 0	7	TIME$
200	C8		RET Z	8	MEM
201	C9		RET	9	INKEY$
202	CA	NN NN	JP Z,NN	10	THEN
203	CB	(EXTENDED	INSTRUCTION SET)	11	NOT
204	CC	NN NN	CALL Z,NN	12	STEP
205	CD	NN NN	CALL NN	13	+
206	CE	NN	ADC A,N	14	-
207	CF		RST 8	15	*
208	D0		RET NC	16	/
209	D1		POP DE	17	↑
210	D2	NN NN	JP NC,NN	18	AND
211	D3	NN	OUT N,A	19	OR
212	D4	NN NN	CALL NC,NN	20	>
213	D5		PUSH DE	21	=
214	D6	NN	SUB N	22	<
215	D7		RST 10H	23	SGN
216	D8		RET C	24	INT
217	D9		EXX	25	ABS
218	DA	NN NN	JP C,NN	26	FRE
219	DB	NN	IN A,N	27	INP
220	DC	NN NN	CALL C,NN	28	POS
221	DD	(EXTENDED	INSTRUCTION SET)	29	SQR
222	DE	NN	SBC A,N	30	RND
223	DF		RST 18H	31	LOG
224	E0		RET PO	32	EXP
225	E1		POP HL	33	COS
226	E2	NN NN	JP PO,NN	34	SIN
227	E3		EX (SP),HL	35	TAN
228	E4	NN NN	CALL PO,NN	36	ATN
229	E5		PUSH HL	37	PEEK
230	E6	NN	AND N	38	CVI
231	E7		RST 20H	39	CVS
232	E8		RET PE	40	CVD
233	E9		JP (HL)	41	EOF
234	EA	NN NN	JP PE,NN	42	LOC
235	EB		EX DE,HL	43	LOF
236	EC	NN NN	CALL PE,NN	44	MKI$
237	ED	(EXTENDED	INSTRUCTION SET)	45	MKS$
238	EE	NN	XOR N	46	MKD$
239	EF		RST 28H	47	CINT
240	F0		RET P	48	CSNG
241	F1		POP AF	49	CDBL
242	F2	NN NN	JP P,NN	50	FIX
243	F3		DI	51	LEN
244	F4	NN NN	CALL P,NN	52	STR$
245	F5		PUSH AF	53	VAL
246	F6	NN	OR N	54	ASC
247	F7		RST 30H	55	CHR$
248	F8		RET M	56	LEFT$
249	F9		LD SP,HL	57	RIGHT$
250	FA	NN NN	JP M,NN	58	MID$
251	FB		EI	59	' (REM)
252	FC	NN NN	CALL M,NN	60	
253	FD	(EXTENDED	INSTRUCTION SET)	61	
254	FE	NN	CP N	62	
255	FF		RST 38H	63	

Notes

Pathways through the ROM

Program fast and easy!

The book contains a detailed breakdown of the BASIC and Disk Operating System of the world's most popular personal computer, the Radio Shack TRS-80 (trademark of Tandy Corporation) microcomputer.

You will find comments by memory address, tables of commands with their locations, arithmetic routines using ROM CALLs, lessons on using the ROM routines, and supplementary information.

Contains the TRS-80 Disassembled Handbook by Robert M. Richardson, Supermap by Roger Fuller, Hex Mem BASIC Monitor by John T. Phillipp, a Z-80 Disassembler by George Blank, DOS Map by John Hartford, the specification sheet for the Western Digital Floppy Disk Controller Chip, and an Appendix listing the possible uses of different bytes of memory.

SoftSide Publications