2080 ENGINEERING FUNCTIONAL SPECIFICATION

Major edition: 3

Edit: 40

CHAPTER 1

INTRODUCTION

1.1 TABLE OF CONTENTS

1.2 PREFACE

The major goal of the engineering functional specification is to provide a current, accurate, and easily updatable description of the functional characteristics of the 2080 system. The specifications will describe the goals and functions of the system as a whole as well as the functions of the individual components. The specification will also describe the motivations and the "whys" behind the functional design. The components will be described as functional units (e.g. an IBOX or cache) not as physical parts of the system such as a particular board.

Emphasis will be placed on clarity of description and speed of distribution. English, style, and appearance will be considered only of secondary importance. Update pages will be issued frequently. All major changes will be marked with change bars.

The functional specifications describe the hardware and firmware. They will be the documents from which the diagnostic engineers and software engineers will write their programs. The functional specification will serve as input to the technical writers. This function specification should never be released to customers.

The functional specification will describe the test procedures used to verify the specification, especially the RAMP specification. This will allow ECOs and new software releases to be verified with the same thoroughness as the original product.

Sections enclosed in double square brackets [[...]] are editorial comments and questions. They should be resolved as time goes on.

This function specification describes the entire 2080 system. Not all items will be available at the same time. The current schedules will be part of the "2080 System Development Plan".

CHAPTER 2

SYSTEM DESCRIPTION

2.1 OVERVIEW

2.1.1 PRODUCT STRATEGY

The 208Ø provides the basic upgrade path for current TOPS-1Ø and TOPS-2Ø customers. The 208Ø also will allow 36-bit customers to coexist with VAX by using the corporate interconnect strategy.

When making trade-offs the 208Ø project will consider the following list of primary product goals:

1. Earliest possible time to market (most important)
2. Performance should be 3.5 to 5 X KL1ØE (+/- 15 percent). We will assure that performance is at least 4.ØØ X KL1ØE. The compile times for MR1CD1 (a COBOL program) and SP1111 (a FORTRAN program) run under TOPS-2Ø will be used as the benchmark to measure logic performance.
3. FORTRAN performance with the APA will be at least 3.5 X KL1ØE. This will be measured as the execution time for SP1111. Without the APA FORTRAN execution speed will not be much faster than 2 X KL1ØE and some heavy double precision programs may run slower than the KL1ØE.
4. Lowest possible cost of ownership. This includes the cost of the 208Ø, the cost of field service and the impact of downtime.
5. Provide for coexistence with VAX.

In addition there are the following secondary 208Ø product goals:

1. Provide adequate microcode address space and storage to allow for future functionality or performance improvements.
2. Perform comprehensive error detection at the interfaces between modular units to prevent errors from propagating into the rest of the system without detection.

3. All detected errors will be logged in non-volatile storage.

4. Provide remote diagnostic capability greater than that provided on the KL10 or KS10.

2.1.2 CPU CLUSTER BLOCK DIAGRAM

```
           +------------+        +------------+         +------------+
           !ARITHMETIC  !        !            !         !            !
           !     HOT    !<=======!    EBOX    !<=======>!    IBOX    !
           !     BOX    !        !            !         !            !
           +------------+        +------------+         +------------+
             !   ^                  !   ^  !                !  ^
             !   !                  !   !  !                !  !
             !   +-----------+------!---+-------------------!--+ DATA BUS
             !_______________!______!______!________________! ADDRESS BUS
                             !    !        !
                      +------+-------+     !
CONNECTIONS TO DIAG   !              !     !
LOGIC ON EACH BOARD   !    MBOX      !     !
       ^              !              !     !
       !              +--------------+   +-------+
       !                      !          !  ECL  !
       !                      +----------!  TO   !========================
 +------------+               !          !  TTL  !    !     !     !     !
 !            !       +--------------+   +-------+    !     !     !     !
 ! DIAGNOSTIC !       !              !       ^        !     !     !     !
 !  CONSOLE   !       !   MEMORY     !       !    +-------+ ! +-------+ !
 !            !       !              !       !    !  ICCS ! ! !  IBM  ! !
 +------------+       +--------------+       !    !  PORT ! ! !  PORT ! !
        !                                    !    +-------+ ! +-------+ !
        +------------------------------------+              !           !
                                                            !           !
                                                       +----+      +----+
                                                       !           !
                                                  +-------+    +--------+
                                                  !  UBA  !    !  MBA   !
                                                  !       !    !        !
                                                  +-------+    +--------+
```

NOTE

Some details (e.g. packet buffers, port modules) are omitted for clarity

2.1.3 SYSTEM CONFIGURATIONS

2.1.3.1 INITIAL SHIPMENTS - In order to reduce the risk and schedule for the 2080 system, the configurations are being split into two phases. First is a MASSBUS and UNIBUS based system· to support shipments during the first year of production. Second is a CI based system targeted for the majority of the 2080 product life.

The MASSBUS based system will consist of the following options:

2080 SYSTEM PACKAGE

KC10 Central Processor with 16 entry I-buffer and 8K word cache

512K words main memory

Two RH28 MASSBUS adaptors (for disk and tape)

One DU28 UNIBUS adaptor

One RP06-AA disk pack drive.

One front end with 16 Asynchronous lines consisting of:
One DN20-MA Front End
One DN20-CC Cable Cabinet
One DN25-EC Asynchronous expansion cabinet
One DN25-BA Asynchronous expansion group

One LA34 and one VT100 console terminal package.

TOPS-20 Operating System with Extended Features and TOPS-20 License

Initial Support package

MOS MEMORY

MA28	512K memory expansion kits.

DISK AND TAPE CONTROLLER

RH28	Massbus Controller for disk and tape. This consists of an IO port and a MASSBUS adaptor module.

ADD-ON DISK PACK DRIVES

RP06-AA	Single access 40 million word add-on disk pack drive.
RP06-BA	Dual access 40 million word add-on disk pack drive.

RP06-C	RP06 dual access kit.

RP20 DISK SUBSYSTEMS

RTP20-EA	Single channel 967Mb master TOPS-10.
RTP20-EC	Dual channel 967Mb master TOPS-10.
RTP20-FA	Single channel 929Mb master TOPS-20.
RTP20-FD	Dual channel 929Mb master TOPS-20
RP20-AC	Add-on single channel TOPS-10 disk.
RP20-AE	Add-on single channel TOPS-20 disk.
RP20-CB	Slave dual port upgrade kit. Converts RP20-AC or RP20-AE to dual channel disks.

TU78 SERIES MAGNETIC TAPE DRIVES

TU78-CB	Magnetic tape subsystem including controller and one master tape drive 9-track, 125 inches per second, 1600/6250 bits per inch.
TU78-AF	9-track, 125 ips, 1600/6250 bpi add on tape drive. PREREQUISITE: TU78-CB.

TU72 SERIES MAGNETIC TAPE DRIVES

TX02-EE	Magnetic tape controller and DX20 channel for TU72 series tapes.
TU72-EC	9-track, 125 ips, 1600/6250 bpi add-on tape drive.
TX03-EE	Two channel switch option and DX20 channel.
TX03-FB	Two channel switch for two TX02 controllers.
TX05-EC	Two control unit tape switch option and TX02.

LINE PRINTERS

LP20-AA	300 LPM, LP05, 64-character, EDP
LP20-AC	300 LPM, LP05, 64-character, SCIENTIFIC
LP20-BA	240 LPM, LP05, 96-character, EDP

LP20-BC	240 LPM, LP05, 96-character, SCIENTIFIC
LP20-CA	900 LPM, LP14, 64-character, EDP
LP20-CC	900 LPM, LP14, 64-character, SCIENTIFIC
LP20-DA	660 LPM, LP14, 96-character, EDP
LP20-DC	660 LPM, LP14, 96-character, SCIENTIFIC
LP200-BA	LP07 CHARABAND line printer with LP20 control & cable kit, 900/1200 LPM.

CARD READERS

CD20-CA	1200 CPM console-model reader and control.

SYNCHRONOUS LINE INTERFACES

DN20-BA	Low speed synchronous line controller.
DN20-DA	Synchronous expansion drawer. Including one DN20-BB with slots for up to 3 additional DN20-BBs.
DN20-BB	Low speed synchronous line interface (2400 baud to 19.2Kb)
DN21-DA	Synchronous expansion drawer. Including one DN21-BA with slots for up to 3 additional DN21-BAs.
DN21-BA	High speed synchronous line interface (19.2Kb to 56Kb)
DN21-xx	DMC-11 based high speed synchronous line interface (up to 1Mb)

ASYNCHRONOUS LINE INTERFACES

DN25-EC	Asynchronous expansion cabinet and first drawer including one 8-line multiplexer.
DN25-DA	Asynchronous expansion drawer, including one 8-line multiplexer.
DN25-AA	8-line asynchronous line multiplexer.
DN25-AB	8-line asynchronous multiplexer with distribution panel.
DN25-BA	8-line asynchronous expansion multiplexer. Expands the DN25-AA or DN25-AB to 16 lines.

| 2.1.3.2 TARGET SYSTEMS - Here is a picture of a possible dual 2080 configuration which should give a good idea of the sorts of things that can be done. There are two 2080 systems hooked together with ICCS. Each 2080 has an HSC50 which can access a number of disks. Terminals are attached using a MERCURY communications controller on the ICCS bus.

In practice, this system would have more ICCS lines connecting the 2080's to the various I/O controllers.

```
        +----------+                            +----------+
        !          !---+----------+---------+---!          !
        !          !   !    +---------+     !   !          !
        !  2080    !   !    ! MERCURY !     !   !  2080    !
     +--!          !   !    +---------+     !   !          !
     !  !          !   !     ! ! ! ! !      !   !          !
     !  +----------+   !     ! ! ! ! !      !   +----------+
     !                 !     TERMINALS      !
 +---------+       +-------+            +-------+
 ! MERCURY !       ! HSC50 !            ! HSC50 !
 !  COMM   !       +-------+            +-------+
 +---------+           !      +------+      ! !      +------+
  ! ! ! ! !            !      !      !      ! !      !      !
  ! ! ! ! !            +------! DISK !------+ +------! TAPE !
  TERMINALS            !      !      !      !        !      !
                       !      +------+      !        +------+
                       !                    !
                       !      +------+      !
                       !      !      !      !
                       +------! DISK !------+
                              !      !
                              +------+
```

2.2 HARDWARE ORGANIZATION

This section provides a summary of the major blocks in the 2080 CPU cluster. Full details of each block will be found in later chapters.

2.2.1 CONSOLE SUMMARY

The console performs the traditional "lights and switches" function on the 2080 system. It performs three functions for the 2080 system: bootstrap, operator support, and diagnostics.

2.2.1.1 BOOTSTRAP - This function turns the machine into something that the software bootstrap can use to load the monitor or diagnostics. The following actions need to take place:

1. Reset the 2080 and verify that nothing is obviously broken.

2. Load the microcode (IBOX, EBOX, MBOX, PORT).

3. Load and start a pre-boot from disk or tape.

2.2.1.2 OPERATOR SUPPORT - This mode provides several simple commands to start and stop the system. It also provides operating system support for the console terminal so the operator can talk to the operating system.

2.2.1.3 DIAGNOSTIC SUPPORT - The console can control the hardware and monitor outputs to enable programs to execute in the console and diagnose the machine.

2.2.2 IBOX SUMMARY

The part of the 2080 most responsible for its speed, (and complexity) is the IBOX (or instruction unit). The purpose of the IBOX is to pre-fetch instructions and operands, so that the EBOX (or execution unit) will have a steady stream of instructions to execute.

The IBOX attempts to gather up to 16 instructions and operands ahead of the EBOX, so as to smooth the flow between the memory system and the EBOX, both of which have considerable variation in their operation times.

Central to the IBOX is a buffer of 8 even-odd pairs of instructions. The buffer is allocated in pairs since the MBOX can deliver an even-odd pair in one of its cycles. The storage in the instruction buffer is dynamically allocated among up to 3 instruction streams, depending on the branching characteristics of the program fragment being pursued. Associated with each of the 16 slots in the instruction buffer are one or two words of operand buffer, the second operand buffer being included with the optional FPA (floating point accelerator).

Since the MBOX may return words in a different order than requested, a system of "tags" is used to keep the returning words straight. The tag consists of the buffer identifier (instruction, first or second operand) and the slot number. Each request to the MBOX is accompanied by a tag, which is returned by the MBOX when it completes the request. Along with the returned tag, the MBOX may indicate some type of failure which prevented it from returning the requested data.

The IBOX is controlled by an interruptable microcoded engine. The top level program in the IBOX basically tries to keep the instruction buffers full by issuing MBOX requests for instructions. It decides which of the three possible streams to follow, and issues tags for the resultant requests.

As words come back from the MBOX, IBOX microcode interrupts are taken, which cause the words to be stored away, and analyzed. In the case of returning instructions, the effective address is computed, and, if the instruction requires, another MBOX request is issued for the data word(s). If the instruction is a branch, then, unless all three instruction streams are active, another stream is activated, causing both branch paths to be pursued. An associative comparison is made at each branch to determine whether the target is already located within the instruction buffer, thus allowing short loops to run without fetching instructions from the MBOX.

A three stage delay pipe is employed to allow the tags of the possible successor instructions to be stored with each instruction. This is necessary since the instruction buffer storage allocation is dynamic, and may be entirely scrambled.

All of the above described pre-fetching is done without regard to the validity of either the index registers or the actual data being fetched. Cache thrashing which occurs as a result of referencing words which may not, after all, be required is hoped to be minimal. Attempts to reference words in pages not contained in the page translation cache are simply aborted, and the appropriate error indication is returned by the MBOX. These references are deferred until the word is actually required by the EBOX. In this case, it is hoped that not too many legitimate translation misses occur, since that will bottle up the IBOX pre-fetching.

Simultaneous with keeping its buffers filled from the MBOX, the IBOX is emptying itself by passing instructions and data to the EBOX. Although the EBOX microcode engine takes two cycles per microinstruction, the IBOX supplies the first microinstruction from its own fast RAMs, allowing many instructions to pass thru the EBOX in one cycle. If two microinstructions are required in the EBOX, the second one comes from the EBOX microcode, and will be ready to go after the first IBOX-originated cycle, thus allowing completion in two cycles. Successive microinstructions take 2 cycles each.

As an instruction is passed to the EBOX, the tags of the two possible successor instructions become available, and both are read out of the buffer simultaneously. A static predictor based on the instruction type selects one of the two to be presented to the EBOX. If the previous instruction takes the contrary branch, one cycle is lost while switching to the alternate successor.

The remaining problem is that of "conflicts", which occur when a result of one instruction affects a successive instruction already in the IBOX. Since the architecture of the PDP-10 allows arbitrary storing into the instruction stream, anything done in speculation might be invalid. Thus, the IBOX keeps in associative memory all locations, both in memory and in general registers, which were used in the process of filling its buffers.

Stored in the associative memory are the PC (program counter) values from which instructions were fetched, the index register locations used to compute effective addresses, and the accumulator locations which either are themselves data, or might point to data.

When the EBOX does an instruction which writes an AC which is either the AC operand or the memory operand of the next instruction, the result is "fed forward", substituting for the actual register contents at no cost in time. The writing of an AC also forces an associative lookup against all registers used in the calculation of the effective addresses for instructions in the IBOX buffers. Similarly, the writing of a result to memory forces the associative comparison to be made against memory locations used to fill the IBOX buffers.

Should any matches against the IBOX occur, a priority encoder forces as many refetches as are necessary, although the order in which they are done does not bear any relation to the order in which they may be needed. Since the same location may be overwritten, and refetches issued, up to twice before the MBOX responds, a count of outstanding MBOX references is kept against each buffer slot. This count must be zero for the buffer contents to be considered valid.

A special test is made in the IBOX for unconditional jumps (or at least the most common of them: JRST 0,). If a JRST is the next instruction up, and the EBOX signals that it is not storing over the JRST, which is known one cycle early, then the IBOX will pass over the JRST, and be ready to feed the successor instruction to the EBOX, with no net time spent on the JRST. Since JRSTs account for some 10% of all instructions executed, this is a significant advantage.

2.2.3 EBOX SUMMARY

The EBOX does all of the "real" computation in the 2080. The EBOX has a wide microcode word which directly controls all of the logic in the EBOX.

The EBOX does all of the stores required by the program. This keeps everything in sync so that we can recover from a page failure.

2.2.4 ARITHMETIC PROCESSING ACCELERATOR SUMMARY

The 2080 APA monitors the IBOX and EBOX and speeds up selected operations. These include single and double precision floating point add, subtract, multiply and divide and single precision integer multiply and divide.

The following classes of instructions will be processed in the APA:

1. Single Precision Floating Point

2. Single Precision Integer

3. Double Precision Floating Point

4. Expanded Range Double Precision Floating Point

5. Conversion Instructions

2.2.5 MBOX SUMMARY

The MBOX is the central block of the 2080 system. It contains the cache memory and all of the hardware required to do paging and to read and write physical memory.

The MBOX has a microprocessor which handles many of the complex MBOX functions (like a cache refill). Simple virtual memory reads do not require the microprocessor to do anything.

2.2.6 MEMORY SUMMARY

The 2080 has internal 64K MOS memory. The memory has ECC on every word. The memory always reads 4 words at one time. These four words are latched on the memory array cards and sent serially to the MBOX where they are written into the cache.

2.2.7 I/O SUMMARY

The 2080 uses a microprogrammed IO processor called a "PORT". The port performs several common functions for all 2080 IO interfaces:

1. Interface to the 2080 TTL bus.
2. Data repacking. In particular, conversion of 36 bit words into 8 bit byte streams in any one of several formats.
3. Command processing. The port maintains several queues of outstanding commands and schedules data transfers.
4. Processing channel command lists and buffer descriptors.
5. Interrupt processing.

There are up to eight ports in the 2080. Each port can control any one of four types of IO interface:

1. ICCS link - connection to other processors, HSC50 and MERCURY.
2. MASSBUS - Connection to current disks and tapes (e.g. RP06, RP20, TU78).
3. UNIBUS (or 10/11 interface) - connection to DN20-style communications options.
4. IBM BLOCK MUX - connection to PCM disks and tapes.

2.3 DIFFERENCES FROM THE KL10

2.3.1 SUMMARY OF MAJOR DIFFERENCES

1. Lower transfer cost than a KL10.
2. Smaller in size.
3. Much faster than a KL10.
4. Both TOPS-10 and TOPS-20 paging available under program control.

2.3.2 EXEC AND USER INSTRUCTIONS

2.3.2.1 NEW EXEC AND USER INSTRUCTIONS - [[this section will be completed in a future edition]]

2.3.2.2 CHANGED EXEC AND USER INSTRUCTIONS - [[this section will be completed in a future edition]]

2.3.2.3 DISCONTINUED EXEC AND USER INSTRUCTIONS - UFA, DFN, FADL, FSBL, FMPL and FDVL are not available on the 2080. They all trap as MUUOs.

These instructions are considered obsolete.

2.3.2.4 OTHER EXEC AND USER CHANGES -

2.3.2.4.1 PUBLIC MODE - Public mode is not implemented on the 2080. All instructions behave as if the machine is in concealed mode.

There is also no supervisor mode. The only exec mode is kernel mode.

2.3.3 EXEC MODE ONLY INSTRUCTIONS

2.3.3.1 NEW EXEC MODE ONLY INSTRUCTIONS - [[THIS SECTION WILL BE COMPLETED IN A FUTURE EDITION]]

2.3.3.2 CHANGED EXEC MODE ONLY INSTRUCTIONS - [[THIS SECTION WILL BE COMPLETED IN A FUTURE EDITION]]

2.3.3.3 DISCONTINUED EXEC MODE ONLY INSTRUCTIONS - [[THIS SECTION WILL BE COMPLETED IN A FUTURE EDITION]]

2.3.3.4 OTHER EXEC MODE ONLY CHANGES - [[THIS SECTION WILL BE COMPLETED IN A FUTURE EDITION]]

2.3.3.4.1 OVERFLOW FLAG - In the 2080 the overflow flag comes back as PC flag bit 0. In the KL10 last-instruction-public comes back as PC flag bit 0.

2.3.4 IO DIFFERENCES

[[THIS SECTION WILL BE COMPLETED IN A FUTURE EDITION]]

2.3.5 ADDRESSING DIFFERENCES

[[THIS SECTION WILL BE COMPLETED IN A FUTURE EDITION]]

2.3.6 INTERRUPT DIFFERENCES

[[THIS SECTION WILL BE COMPLETED IN A FUTURE EDITION]]

2.3.7 PAGING DIFFERENCES

Both TOPS-10 and TOPS-20 paging are implemented in one version of the microcode and selected by bit 21 in WREBR.

2.3.7.1 PAGE FAIL CODES - [[this section will be completed in a future edition]]

2.3.8 ERROR RECOVERY DIFFERENCES

[[this section will be completed in a future edition]]

2.3.9 MISC. CHANGES

[[this section will be completed in a future edition]]

COMPANY CONFIDENTIAL -- Do not duplicate 9-JUL-80

9/30 Ted says this is not the latest. See VULCAN:<HESS.D KC10.MEM

CHAPTER 3

EXEC MODE PROGRAMMING

3.1 INTRODUCTION

This document describes the operation of the Exec Mode and I/O instructions on the KC10. All I/O instructions are ordinary instructions with the same format as normal instructions (opcode, AC and effective address). The opcodes are in the range 700 thru 777.

Any operation code in the range 700 thru 777, AC number, field or bit not described in this document should be considered reserved to DEC. The microcode will attempt to generate an illegal instruction trap whenever a reserved action is attempted.

Opcodes 700 thru 737 are for exec mode only (I/O legal) instructions. Opcodes 740 thru 777 are for exec and user instructions.

Included in this document is a description of the implementation of all 4096 sections of virtual address space, the EPT/UPT layout, trap handling and process context variables. Chapter 4 contains a proposal for new instructions for manipulating queues and physical memory.

NOTE

Items still to be added to this document:

1. Interrupt protocol description.
2. Address break facility description (if any).
3. System timers and accounting facilities.

OPCODE Assignment Map

	0	1	2	3	4	5	6	7
700	APRØ	APR1	APR2	-	PMOVE	PMOVEM	-	-
710	RNGB	RNGBW	SNBSY	-	-	-	-	-
720	INSQHI	INSQTI	REMQHI	REMQTI	-	-	-	-
730	-	-	-	-	-	-	-	-
740	INSQUE	REMQUE	-	-	-	-	-	-
750	-	-	-	-	-	-	-	-
760	-	-	-	-	-	-	-	-
770	-	-	-	-	-	-	-	-

AC Field Assignments

AC	700	701	702
ØØ	APRID	-	RDSPB
Ø1	-	RDUBR	RDCSB
Ø2	-	CLRPT	RDPUR
Ø3	-	WRUBR	RDCSTM
Ø4	WRAPR	WREBR	RDTIM
Ø5	RDAPR	RDEBR	RDINT
Ø6	-	WRIOP	UPDTIM
Ø7	-	RDIOP	-
1Ø	-	-	
11	-	SWPIA	WRCSB
12	-	SWPVA	WRPUR
13	-	SWPUA	WRCSTM
14	WRPI	SWPPT	WRTIM
15	RDPI	-	WRINT
16	-	-	-
17	-	-	-

NOTE

New instructions proposed in this document are still subject to review and approval by Software Engineering and the Architecture Committee.

3.2 INTERNAL I/O INSTRUCTIONS

These instructions control the CPU. They transfer no data between the outside world and the CPU or memory.

APRID

```
+---------+----+-+----+-----------------+
!   700   ! 00 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This instruction returns the microcode version number, the CPU serial number, and processor options. This data is stored at E and E+1 in the following format:

E:

0-8	Reserved for microcode options.
9-17	Hardware options (These are all zero at present)
18-35	Processor serial number

E+1:

0-35	Microcode version number (in Standard format)

WRAPR

```
+---------+----+-+----+-----------------+
!   700   ! 04 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This immediate mode instruction decodes its effective address to control the processor. The effective address bits are used as follows:

19	I/O reset. When this bit is set all internal devices are reset. In addition, an input/output init is generated.
20	Enable conditions selected by bits 24 thru 31 to cause interrupts.
21	Disable interrupts for conditions selected by bits 24 thru 31.
22	Clear flags indicated by bits 24 thru 31.
23	Set flags indicated by bits 24 thru 31.

WARNING

The action of the processor is not defined when both bits 20 and 21 or 22 and 23 are set in the same instruction.

24-30	To be described
31	Cache sweep done
33-35	PIA

RDAPR

```
+---------+----+-+----+-----------------+
!   700   ! 05 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This instruction stores the APR status in the word addressed by E. The status is as follows:

06-13	Flags enabled to interrupt (tbd)
13	Cache sweep done
19	Cache sweep busy
24-31	Flags, cause of interrupt (tbd)
31	Cache sweep done
32	Interrupt requested
33-35	PIA

WRPI

```
+---------+----+-+----+-----------------+
!   700   ! 14 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This immediate mode instruction decodes its effective address to control the priority interrupt system. The effective address bits are used as follows:

18-20	Processor error insertion (to be defined).
22	Turn off program requests on selected levels (bits 29-35)
23	Clear PI system.
24	Initiate interrupts on selected levels (bits 29-35)
25	Turn on selected levels (bits 29-35)
26	Turn off selected levels (bits 29-35)

WARNING

The action of the processor is not defined when both 25 and 26 or 22 and 24 are set in the same instruction.

29-35	Select levels for bits 22, 24, 25, and 26.

RDPI

```
+---------+----+-+----+------------------+
!   700   ! 15 !@! XR !        Y         !
+---------+----+-+----+------------------+
```

This instruction store the PI status in the word addressed by E. The status is a follows:

11-17	Program requests on levels.
18-20	To be defined (set by WRPI).
21-27	Interrupt in progress on levels.
28	PI system on.
29-35	Levels on.

WRUBR

```
+---------+----+-+----+-----------------+
!   701   ! 03 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This instruction loads user process context with the words at E and E+1. The format of the first word (E) is:

0	Load AC block numbers
1	Load PCS
2	Load UBR
3	Clear "kept" pages from the hardware page table.
4	Do not update accounts.
18-35	Physical page number of UPT.

The format of the second word (E+1) is:

6-17	Previous Context Section
30-32	Current AC block
33-35	Previous AC block

If bit 2 is on perform the following functions:

1. If bit 4 in E is 0, update the user accounts.
2. Load bits 18-35 of E into the User Base Register
3. If bit 3 in E is 0, clear only those pages marked "kept" in the Page Table, else; If bit 3 is 1, clear all entries.

RDUBR

```
+---------+----+-+----+-----------------+
!   701   ! 01 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This instruction reads back the user process context and returns two words at E and E+1 in exactly the same format as used by WRUBR. In order to allow these words to be used directly in a WRUBR instruction, bits 0 thru 2 are set to 1 and bits 3 and 4 are set to 0 in E.

CLRPT

```
+---------+----+-+----+-----------------+
!   701   ! 02 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This immediate mode instruction clears the hardware page table entry associated with the EA of this instruction so that the next virtual reference to the word specified by EA will cause a pager refill trap to occur.

WREBR

```
+---------+----+-+----+-----------------+
!   701   ! 04 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This instruction loads the exec mode context from the word at E. The format of the word is:

0-1	Cache enable (Look and Load).
2	KL10 compatible paging (TOPS-20 see sec 3.5)
3	Extended paging (TOPS-20 see sec 3.5)

WARNING

The effect of turning on both bits 2 and 3 will be undefined.

4	Trap and paging enable.
18-35	Physical page number of EPT.

RDEBR

```
+---------+----+-+----+-----------------+
!   701   ! 05 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This instruction returns the value given to WREBR and stores it in the word addressed by E.

WRIOP

```
+---------+----+-+----+-----------------+
!   701   ! 06 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This immediate mode instruction sets the physical page number of the I/O page to be bits 18-35 of E.

RDIOP

```
+---------+----+-+----+-----------------+
!   701   ! 07 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

This instruction returns the value given to WRIOP and stores it in the word addressed by E.

SWPIA

```
+---------+----+-+----+-----------------+
!   701   ! 11 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Sweep Cache, Invalidate All Pages

Set Sweep Busy and clear the valid and written state in all cache entries. At the completion of the sweep, clear Sweep Busy and set Sweep Done, requesting an interrupt on the level assigned to the processor.

SWPVA

```
+---------+----+-+----+------------------+
!   701   ! 12 !@! XR !        Y         !
+---------+----+-+----+------------------+
```

Sweep Cache, Validate All Pages

Set Sweep Busy and write into storage all cache entries that are in the written state. Clear the written state associated with those words sent to storage, but do not change the validity of any entries. At the completion of the sweep, clear Sweep Busy and set Sweep Done, requesting an interrupt on the level assigned to the processor.

SWPUA

```
+---------+----+-+----+------------------+
!   701   ! 13 !@! XR !        Y         !
+---------+----+-+----+------------------+
```

Sweep Cache, Unload All Pages

Set Sweep Busy and write into storage all cache entries that are in the written state. Invalidate all entries (i.e. clear valid and written state). At the completion of the sweep, clear Sweep Busy and set Sweep Done, requesting an interrupt on the level assigned to the processor.

SWPPT

```
+---------+----+-+----+-----------------+
!   701   ! 14 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Sweep Page Table

Interpret the EA of this instruction as a physical address and clear all the associated entries in the hardware page table that contain a virtual to physical mapping to this address.

WARNING

This instruction may actually be slower than doing a WRUBR followed by the pager refill cycles caused by clearing all of the hardware page table entries.

WRSPB

```
+---------+----+-+----+------------------+
!   702   ! 10 !@! XR !         Y        !
+---------+----+-+----+------------------+
```

Write SPT Base Register

Load the word at E into the SPT base register.

BASE REGISTER FORMAT

All base registers are loaded with a physical word address. All high order bits must be zero. The address need not be on a page boundary and may be any place in physical memory. There is no range check on SPT or CST offsets. The monitor is assumed to always put correct data into page tables.

RDSPB

```
+---------+----+-+----+-----------------+
!   702   ! 00 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Read SPT Base Register

Store the SPT base register at E.

WRCSB

```
+---------+----+-+----+-----------------+
!   702   ! 11 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Write Core Status Table Base Register

Load the CST base register with the word at E.

BASE REGISTER FORMAT

All base registers are loaded with a physical word address. All high order bits must be zero. The address need not be on a page boundary and may be any place in physical memory. There is no range check on SPT or CST offsets. The monitor is assumed to always put correct data into page tables.

RDCSB

```
+---------+----+-+----+-----------------+
!   702   ! 01 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Read Core Status Table Base Register

Store the CST base register at E.

WRPUR

```
+---------+----+-+----+-----------------+
!   702   ! 12 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Write Process Use Register

Load the process use register from E. The process use register contains the AGER (age register) in the left few bits. The bits containing the AGER are cleared by anding the CST entry with the CST mask, then the entire PUR is ored with the CST entry.

RDPUR

```
+---------+----+-+----+-----------------+
!   702   ! 02 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Read Process Use Register

Store the PUR at E.

WRCSTM

```
+---------+----+-+----+-----------------+
!   702   ! 13 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Write CST Mask Register

Load the CST mask register from E. The CST mask register should contain a 0 for every bit in the ACER and a 1 in all other bit positions.

RDCSTM

```
+---------+----+-+----+-----------------+
!   702   ! 03 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Read CST Mask Register

Store the CST mask register at E.

WRTIM

```
+----------+----+-+-+----+-----------------+
!   702    ! 14 !@! XR !        Y          !
+----------+----+-+-+----+-----------------+
```

Write Time Base

This immediate instruction decodes its effective address to control various system time and accounting meters. For a complete explanation of the time and accounting meters see sec. 3.7. The effective address bits are used as follows:

18	Set up accounts (bits 21-23).
21	Enable PI accounting.
22	Enable Kernel mode accounting.
23	Turn on accounting.
24	Turn off time base.
25	Turn on time base.
26	Clear time base.
33-35	PIA for interval timer.

Briefly, the time base consists of two counters that count at a 1 ms. and 10 us. rate. The millisecond counter consists of 1 word in the EPT at location 512. The 10 microsecond counter consists of 2 words in the EPT at locations 510 and 511. These two counters are driven from the same source and are updated in parallel by the microcode.

RDTIM

```
+---------+----+-+----+-----------------+
!   702   ! 04 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Read Time Base

Read the status of the accounting meters and time base, and the interrupt level assigned to the interval timer into the word addressed by E. The status is as follows:

21	PI accounting enabled.
22	Kernel mode accounting enabled.
23	Accounting on.
25	Time base on.
33-35	PIA for interval timer.

UPDTIM

```
+---------+----+-+----+-----------------+
!   702   ! 06 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Update Time Base

Update the time base count from the hardware counter, and transfer the result from locations 510, 511 and 512 in the EPT to locations E, E+1, and E+2. Updating clears the hardware counter.

WRINT

```
+---------+----+-+----+-----------------+
!   702   ! 15 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Write Interval Timer

This immediate mode instruction decodes its effective address to setup the interval timer. The effective address bits are used as follows:

18	Clear interval timer.
21	Turn interval timer on.
22	Clear interval flags.
24-35	Interval period.

RDINT

```
+---------+----+-+----+-----------------+
!   702   ! 05 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Read The Interval Register

Read the status of the interval timer into the word addressed by E. The status is as follows:

6-17	Interval count (current contents of the counter).
21	Interval timer on.
22	Interval timer done (causes interrupt).
23	Overflow (implies bit 22).
24-35	Interval period.

3.3 EXTERNAL I/O

The external I/O instructions on the KC10 allow a program to communicate with the I/O ports and the console. In particular they will manipulate the I/O Command/Response Queues and Port Doorbell mechanism. See chapter 10 of the 2080 EFS for a complete description of the Queue and Doorbell features. The interface to the 2080 ports is primarily data areas called "mailboxes" and a doorbell. It is the doorbell mechanism that the following instructions manipulate. The Command/Response Queues will be covered by the queue instructions in the next section.

RNGB

```
+---------+----+-+----+-----------------+
!   710   ! 00 !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Ring Doorbell

This immediate mode instruction will assert a port or console number on the TTL bus and set RING ("doorbell"). The EA of this instruction is the port number and is interpreted as follows:

18	Interrupt 2080 Console
33-35	Port number (Ignored if bit 18 set)

RNGBW

```
+---------+----+-+----+-----------------+
!   711   ! ØØ !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Ring Doorbell and Wait

This immediate mode instruction will assert a port or console number on the TTL bus and set RING ("doorbell"). The EA of this instruction is the port number and is interpreted as follows:

18	Interrupt 2080 Console (does not wait - same function as RNGB)
33-35	Port number (Ignored if bit 18 set)

SNBSY

```
+---------+----+-+----+------------------+
!   712   ! 00 !@! XR !        Y         !
+---------+----+-+----+------------------+
```

Skip if BUSY not set

This instruction skips to PC+2 if the BUSY line of the TTL busy is not set. When used in combination with the RNGB instruction, you can achieve the identical effect of RNGBW as follows:

```
        RNGB    pn              ; Assert RING to port "pn"
        SNBSY                   ; Busy set?
         JRST   .-1             ; Yes, wait.
        ...                     ; No - proceed
```

3.4 NEW INSTRUCTIONS

The following instructions have been added to aid the monitor to manipulate new data structures and to save overhead time and space.

PMOVE

```
+---------+----+-+----+-----------------+
!   704   ! AC !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Physical Move from Memory

Move the physical memory location addressed by E into the AC. The effective address is calculated as a 30-bit virtual address (i.e. all indirect words will come from Exec Virtual Addresses) but only bits 11-35 will be used for data references.

NOTE

Addresses 0-17 will reference physical memory 0-17, not the ACs.

PMOVEM

```
+---------+----+-+-+----+------------------+
!   705   ! AC !@! XR !         Y          !
+---------+----+-+-+----+------------------+
```

Physical Move to Memory

Move AC into the physical memory location addressed by E. The effective address is calculated as a 30-bit virtual address (i.e. all indirect words will come from Exec Virtual Addresses) but only bits 11-35 will be used for data references.

A queue is a circular, doubly linked list. A queue entry is specified by its address. Each queue entry is linked to the next via a pair of words. The first word is the forward link (FLINK): it specifies the location of the succeeding entry. The second word is the backward link (BLINK): it specifies the location of the preceeding entry. The KC10 implements two types of links: physical and virtual. A physical link contains a 25-bit physical address (EXEC mode only) of the entry that it points to. A virtual link contains a 30-bit virtual address of the entry that it points to. A queue is classified by the type of link it uses.

A queue is specified by a queue header which is identical to a pair of queue linkage words. The forward link of the header is the address of the entry termed the head of the queue. The backward link of the header is the address of the entry termed to be the tail of the queue. The forward link of the tail points to the header.

Two general operations can be performed on queues: insertion of entries and removal of entries. Generally entries can be inserted or removed only at the head or tail of a queue. (Under certain restrictions they can be inserted or removed elsewhere; this is discussed later.)

The following is an example of queue operations. An empty queue is specified by its header at address H:

```
                0                                                  35
              +----------------------------------------------------+
    H:        !                        H                           !
              +----------------------------------------------------+
    H+1:      !                        H                           !
              +----------------------------------------------------+
```

If an entry at address B is inserted into an empty queue (at either the head or tail), the queue is as shown below:

```
                0                                                  35
              +----------------------------------------------------+
    H:        !                        B                           !
              +----------------------------------------------------+
    H+1:      !                        B                           !
              +----------------------------------------------------+


                0                                                  35
              +----------------------------------------------------+
    B:        !                        H                           !
              +----------------------------------------------------+
    B+1:      !                        H                           !
              +----------------------------------------------------+
```

If an entry at address A is inserted at the head of the queue, the queue is show below:

```
                 0                                           35
                +----------------------------------------------+
      H:        !                     A                        !
                +----------------------------------------------+
      H+1:      !                     B                        !
                +----------------------------------------------+

                 0                                           35
                +----------------------------------------------+
      A:        !                     B                        !
                +----------------------------------------------+
      A+1:      !                     H                        !
                +----------------------------------------------+

                 0                                           35
                +----------------------------------------------+
      B:        !                     H                        !
                +----------------------------------------------+
      B+1:      !                     A                        !
                +----------------------------------------------+
```

Finally, if an entry at address C is inserted at the tail, the queue appears as follows:

```
                 0                                           35
                +----------------------------------------------+
      H:        !                     A                        !
                +----------------------------------------------+
      H+1:      !                     C                        !
                +----------------------------------------------+

                 0                                           35
                +----------------------------------------------+
      A:        !                     B                        !
                +----------------------------------------------+
      A+1:      !                     H                        !
                +----------------------------------------------+

                 0                                           35
                +----------------------------------------------+
      B:        !                     C                        !
                +----------------------------------------------+
      B+1:      !                     A                        !
                +----------------------------------------------+

                 0                                           35
                +----------------------------------------------+
      C:        !                     H                        !
                +----------------------------------------------+
      C+1:      !                     B                        !
                +----------------------------------------------+
```

If more than 1 process can perform operations on a queue simultaneously, insertions and removals should only be done at the head or tail of the queue. If only 1 process (or 1 process at a time) can perform operation on a queue, insertions and removals can be made at other that the head or tail of the queue. In the example above with the queue containing entries A, B, and C, the entry at address B can be removed giving:

```
              0                                                   35
             +-----------------------------------------------------+
   H:        !                       A                             !
             +-----------------------------------------------------+
   H+1:      !                       C                             !
             +-----------------------------------------------------+

              0                                                   35
             +-----------------------------------------------------+
   A:        !                       C                             !
             +-----------------------------------------------------+
   A+1:      !                       H                             !
             +-----------------------------------------------------+

              0                                                   35
             +-----------------------------------------------------+
   C:        !                       H                             !
             +-----------------------------------------------------+
   C+1:      !                       A                             !
             +-----------------------------------------------------+
```

The reason for the above restriction is that operations at the head or tail are always valid because the queue header is always present; operations elsewhere in the queue depend on specific entries being present and may become invalid if another process is simultaneously performing operations on the queue.

Two instructions are provided form manipulating virtual queues: INSQUE and REMQUE. INSQUE inserts an entry specified by an entry operand (AC) into the queue following the entry specified by the predecessor operand (EA). REMQUE remove the entry specified by the entry operand (EA). Both INSQUE and REMQUE are implemented as non-interruptable instructions.

Four operations can be performed on physical queues: insert at head, insert at tail, remove from head, and remove from tail. Furthermore, these operations are interlocked to allow cooperating processes in a multiprocessor system (CPU and I/O ports) to access a shared queue without additional synchronization. A hardware supported interlocked mechanism is used to read the queue header. Bit 0 of the queue header is used as a secondary interlock and is set when the queue is being accessed. If an interlocked queue instruction encounters the secondary interlock set, the CPU waits until it can access the queue (timeout? page fail?).

INSQUE

```
+---------+----+-+-+----+------------------+
!   740   ! AC !@! XR !         Y          !
+---------+----+-+-+----+------------------+
```

Insert Entry in Queue

The entry specified by the 30-bit virtual address contained in AC is inserted into the queue following the entry specified by E. If the entry inserted was the first one in the queue (i.e. C(E) = C(E+1) after insertion), the instruction does not skip; otherwise it skips to PC+2. The insertion is a non-interruptable operation. Before performing any part of the operation, the processor validates that the entire operation can be completed. This ensures that if a page fail trap occurs, the queue is left in a consistent state.

Because the insertion is non-interruptable, processes running in kernel mode can share queues with interrupt service routines. The INSQUE and REMQUE instructions are implemented in such a way that cooperating processes in a single processor may access a queue without additional synchronization if the insertions and removals are only at the head or tail of the queue.

REMQUE

```
+---------+----+-+----+-----------------+
!   741   ! AC !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Remove Entry from Queue

The queue entry specified by E is removed from the queue. The 30-bit virtual address of the entry removed is placed in AC. If there was no entry in the queue (i.e. C(E) = C(E+1) before removal), the instruction does not skip; otherwise it skips to PC+2. The removal is a non-interruptable operation. Before performing any part of the operation, the processor validates that the entire operation can be completed. This ensures that if a page fail trap occurs, the queue is left in a consistent state.

Because the removal is non-interruptable, processes running in kernel mode can share queues with interrupt service routines. The INSQUE and REMQUE instructions are implemented in such a way that cooperating processes in a single processor may access a queue without additional synchronization if the insertions and removals are only at the head or tail of the queue.

INSQHI

```
+---------+----+-+----+-----------------+
!   720   ! AC !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Insert Entry into Queue at Head, Interlocked

The entry specified by the 25-bit physical address contained in AC is inserted into the queue following the header specified by E. If the entry inserted was the first one in the queue (i.e. C(E) = C(E+1) after insertion), the instruction does not skip; otherwise it skips to PC+2. The insertion is a non-interruptable operation. The insertion is interlocked to prevent concurrent interlocked insertion or removals at the head or tail of the same queue by another process even in a multiprocessor environment. Before performing any part of the operation, the processor validates that the entire operation can be completed. This ensures that if a page fail trap occurs, the queue is left in a consistent state.

Because the insertion is non-interruptable, processes running in kernel mode can share queues with interrupt service routines. The INSQHI, INSQTI, REMQHI, and REMQTI instructions are implemented in such a way that cooperating processes in a multiprocessor may access a shared queue without additional synchronization.

INSQTI

```
+---------+----+-+----+------------------+
!   721   ! AC !@! XR !         Y        !
+---------+----+-+----+------------------+
```

Insert Entry into Queue at tail, Interlocked

The entry specified by the 25-bit physical address contained in AC is inserted into the queue preceding the header specified by E. If the entry inserted was the first one in the queue (i.e. C(E) = C(E+1) after insertion), the instruction does not skip; otherwise it skips to PC+2. The insertion is a non-interruptable operation. The insertion is interlocked to prevent concurrent interlocked insertion or removals at the head or tail of the same queue by another process even in a multiprocessor environment. Before performing any part of the operation, the processor validates that the entire operation can be completed. This ensures that if a page fail trap occurs, the queue is left in a consistent state.

Because the insertion is non-interruptable, processes running in kernel mode can share queues with interrupt service routines. The INSQHI, INSQTI, REMQHI, and REMQTI instructions are implemented in such a way that cooperating processes in a multiprocessor may access a shared queue without additional synchronization.

REMQHI

```
+---------+----+-+----+-----------------+
!   722   ! AC !@! XR !        Y        !
+---------+----+-+----+-----------------+
```

Remove Entry from Queue at Head, Interlocked

The queue entry following the header specified E is removed from the queue. The 25-bit physical address of the entry removed is place in AC. If there was no entry in the queue (i.e. C(E) = C(E+1) before removal), the instruction does not skip; otherwise it skips to PC+2. The removal is a non-interruptable operation. The removal is interlocked to prevent concurrent interlocked insertion or removals at the head or tail of the same queue by another process even in a multiprocessor environment. Before performing any part of the operation, the processor validates that the entire operation can be completed. This ensures that if a page fail trap occurs, the queue is left in a consistent state.

Because the removal is non-interruptable, processes running in kernel mode can share queues with interrupt service routines. The INSQHI, INSQTI, REMQHI, and REMQTI instructions are implemented in such a way that cooperating processes in a multiprocessor may access a shared queue without additional synchronization.

REMQTI

```
+---------+----+-+----+------------------+
!   723   ! AC !@! XR !        Y         !
+---------+----+-+----+------------------+
```

Remove Entry from Queue at Tail, Interlocked

The queue entry preceding the header specified E is removed from the queue. The 25-bit physical address of the entry removed is place in AC. If there was no entry in the queue (i.e. C(E) = C(E+1) before removal), the instruction does not skip; otherwise it skips to PC+2. The removal is a non-interruptable operation. The removal is interlocked to prevent concurrent interlocked insertion or removals at the head or tail of the same queue by another process even in a multiprocessor environment. Before performing any part of the operation, the processor validates that the entire operation can be completed. This ensures that if a page fail trap occurs, the queue is left in a consistent state.

Because the removal is non-interruptable, processes running in kernel mode can share queues with interrupt service routines. The INSQHI, INSQTI, REMQHI, and REMQTI instructions are implemented in such a way that cooperating processes in a multiprocessor may access a shared queue without additional synchronization.

3.5 VIRTUAL ADDRESSING

This section has been extracted from a memo titled "4095 Sections" by Len Bosack dated 12-Oct-78.

3.5.1 Introduction

The KL processor implemented 32 sections of virtual address and has a pager data structure that can accommodate at most 32 sections. The KC10 will implement all 4096 sections of virtual address space forcing a change of the pager data structure to accommodate 4096 section pointers. In addition there will be a "KL10" compatible paging mode that will implement only 32 sections (see WREBR description).

The following paging description is only valid when TOPS-20 style paging is enabled:

3.5.1.1 Pager Data Structure

3.5.1.1 Pager Data Structure - An address would be converted to a physical page number (PPN) as follows, (KL10 compatibility off):

VMA6:8> is used to index into a "Super" Section Table (SST) in the EPT/UPT. One of 2 pointer types can occur here: No Access, or Shared (see below). The Shared Pointer (SPT) index yields the PPN of a Section Table page (ST). VMA9:17> is used to index into the Section Table to obtain a section pointer. Address translation then proceeds as on the KL10 after the section pointer fetch. (See DECsystem10/20 Processor Reference Manual, EK-10/20-HR for a complete description).

3.5.1.2 Super Section Pointers - The following pointer types will only be valid in the Super Section Table in the EPT/UPT locations 520-527. All other pointers will be interpreted as in the KL10, including section pointers found in Section Tables.

No Access:

```
+-----------------------------------------------------+
! 000 !                                               !
+-----------------------------------------------------+
```

Shared:

```
+-----------------------------------------------------+
! 002 !                    !          SPT Index       !
+-----------------------------------------------------+
```

3.6 PROCESS CONTEXT VARIABLES

3.6.1 Introduction

This section contains a proposal for handling process context variables using an expanded PC flags word. The source for this proposal was a memo by Dan Murphy dated 15-Feb-77. Further discussion on a process context switch instruction is still needed. I hope that this section will stimulate some suggestions along these lines.

3.6.1.1 New PC Double Word - The format of the flags word will be:

```
+----------------------------------------------------------+
!        FLAGS         !  !LEN !CAB!PAB !        PCS        !
!        (13)          !  !(3) !(3)!(3) !        (12)       !
+----------------------------------------------------------+
```

1. In order to facilitate backing up a stored PC to the beginning of an instruction that caused a trap, a new field (LEN) has been added that will contain the number of words in an instruction - 1. This will be compatible with all existing software since this field is normally zero.

2. In kernel mode, or when stored on a page fail or MUUO, all of the above fields will be stored as defined. In kernel mode, an XJRSTF will restore all fields.

3. In user mode, PCS, PAB, and CAB will always be stored as 0. An XJRSTF in user mode will treat these fields as it does the user mode and user I/O flag now (i.e. ignore them).

3.6.1.2 Context Changing - Returning to a previous context may be done with an XJRSTF with restores the context variables stored in the previously saved PC word.

Entering a new context will be done as follows: All of the "previous context" variables will be set to their corresponding values in the "current context", and the remainder will be set to pre-defined values taken from the new PC flag word. The following operations are defined as entering a new context:

1. Monitor call (MUUO).

2. Page fail trap.

3. Priority interrupt initiation.

Each of these operations will store a PC double-word containing the "current context" variables and then load a new PC double-word to set new values for those variables not set automatically. See next section for a description of the changes necessary to the EPT/UPT to implement this proposal.

3.7 TRAP HANDLING

3.7.1 Introduction

The current implementation of Overflow Trap handling while quite general, has some problems relating to extended addressing. A new method of transferring control to "sections" other than the current PC section is proposed in the following section. This proposal limits the number of possible actions that can be taken on a trap compared to what the KI10 and KL10 now offer. I do not think however, that any existing software will be severly compromised. It may be necessary for TOPS10 to modify its existing Monitor Call associated with this feature.

3.7.2 Trap Function Word

Instead of an instruction in EPT/UPT locations 421-423, there will be a function word:

```
+-----------------------------------------------------------+
!FN! AC  !                Virtual Address                  !
+-----------------------------------------------------------+
```

The format of this word is interpreted as follows:

0-1	Function code: 00 - Do nothing on overflow (ignore). 01 - Do MUUO (use trap MUUO new PC word). 10 - Transfer control to kernel/user (simulate LUUO). 11 - Transfer control to kernel/user (simulate PUSHJ).
2-5	AC - used to address push-down list register for function code 3 (PUSHJ), Ignored for codes 0-2
6-35	Virtual Address: Function 2 - pointer to 4-word block in current context (kernel/user) to be used to store instruction and effective address and pickup a new PC as in LUUO trapping. Function 3 - routine address in current context (kernel/user) that will be invoked in PUSHJ simulation.

Format of 4-word trap block:

```
!=========================================================!
!        FLAGS         !  !LEN !   OPCODE    ! AC  !  00   !
!---------------------------------------------------------!
!   00    !                      PC                       !
!---------------------------------------------------------!
!   00    !                      E                        !
!---------------------------------------------------------!
!   00    !                    NEW PC                     !
!=========================================================!
```

3.8 SPECIAL SYSTEM PAGES (EPT / UPT / IOP)

The following EPT/UPT layouts are proposed for the KC10. In addition, there is a new page called the I/O page (IOP) that is used by the 2080 ports and the console for communication with the CPU.

Upon processor reset, the base address of the EPT and UPT will be 0. The base address of the IOP will reset to 1000(8).

NOTE

All areas that differ from the KL10 are marked with an asterisk (*).

Exec Page Table

*	0-177	Reserved
	200-377	TOPS-10 paging (Kernel 400-777).
	400-420	Reserved.
	421	Kernel Arithmetic Overflow trap function.
	422	Kernel Push-down List Overflow trap function.
	423	Kernel Trap Type 3 trap function.
	424-507	Reserved.
	510-511	Time Base 2 (10 usec timer).
*	512	Time Base 1 (Millisecond timer).
	513-517	Reserved.
*	520-527	"Super" section table (TOPS-20 paging).
	530-537	Reserved.
	540-577	KL10 Compatible Paging (TOPS-20)
	600-757	TOPS-10 paging (Kernel 0-337).
	760-777	Reserved.

User Page Table

	0-377	TOPS-10 paging (User 0-777).
	400-417	TOPS-10 paging (Kernel 340-377).
	420	Address of LUUO block (TOPS-20).
	421	User Arithmetic Overflow trap function.
	422	User Push-down List Overflow trap function.
	423	User Trap Type 3 trap function.
*	424	MUUO old PC flags.
*	425	MUUO old PC.
*	426	MUUO Opcode and AC.
*	427	MUUO Effective Address.
*	430-431	Kernel no-trap MUUO new PC double word.
*	432-433	Kernel trap MUUO new PC double word.
*	430-431	User no-trap MUUO new PC double word.
*	432-433	User trap MUUO new PC double word.
*	440	Page fail code.
*	441	Page identifier (TOPS-20 paging).
*	442	Page fail virtual address.
*	443-444	Page fail old PC double word.
*	445-446	Page fail new PC double word.
*	447-503	Reserved.
*	504-505	User runtime (10 usec clock).
*	506-507	User accounting meter ("chargons").
	510-517	Reserved.
*	520-527	"Super" section table (TOPS-20 paging).
	530-537	Reserved
	540-577	KL10 Compatible Paging (TOPS-20)
	600-777	Reserved.

I/O Page

0-77	Port Register Access Blocks (8)
100	Port Interrupt Logout Word
101	APR Interrupt Vector
102	Console Interrupt Vector
103	Interval Timer Interrupt Vector
104-107	Reserved
110-157	Port Interrupt Vectors (4 per port)
160-377	Reserved
400-777	Console Communications Region

CHAPTER 4

INSTRUCTION EXECUTION

4.1 EXECUTION UNITS PARTITIONING

control storage 5.23

shift / control / meters / FRU 5.12

5.8 matrix

5.2

```
                     2080 EBOX/IBOX PARTITIONING

      Shift/ac/Regs    slow                move/ALU      fast
                       RAMS                              RAMS
+-----+-----+-----+-----+-----+-----+-----+-----+-----+     +-----+-----+-----+
!     !     !     !     !     !     !     !     !     !     !     !     !     !
! SCA ! SHL ! SHR ! CRA ! CLK ! MVL ! MVR ! ESE !     !     ! TAG ! IDL ! IDR !
!     !     !     !     !     !     !     !     !     !     !     !     !     !
+-----+-----+-----+-----+-----+-----+-----+-----+     +-----+-----+-----+-----+
     K  o      s     i   c        Hooper      Kosic          H o o p E R
\                                               /     \                 /
 \_____________________  ______________________/       \______  _______/
                       \/                                     \/
                      EBOX                      CLK          IBOX
```

```
                     2080 FPA PARTITIONING

      +-----+-----+-----+-----+-----+-----+-----+
      !     !     !     !     !     !     !     !
      ! FPE ! FPD ! FPC ! FPB ! FPA !     ! FPI !
      !     !     !     !     !     !     !     !
      +-----+-----+-----+-----+-----+-----+-----+
```

4.1.1 Overall

The 2080, in its maximum configuration, contains 3 units which are directly involved with the execution of instructions: IBOX, EBOX, and FPA. The IBOX fetches streams of instructions, performs effective address calculations, and fetches the memory operand, <or operands>, necessary for the instructions it determines may be required by the EBOX. The EBOX accepts AC addresses, an instruction code, a memory operand <or operands>, and paging or error information pertaining to memory addresses of each instruction, as each is called up for execution by the "last cycle" micro-order. It then performs the operation indicated by the instruction code. The FPA provides a speedup of multiply, divide, and instruction requiring two memory fetches, and as such is a logical <and optional> extension of the IBOX and EBOX.

4.1.2 IDR IBOX

IDR <IBOX Datapath Right> contains bits 18-35 of the IBOX memory address path, operand data path and program counter logic. Operand memory address conflict detection is present on bits 18-35 of the VMA <Virtual Memory Address>. Instruction memory address conflict detection of VMA 18-34 is also located on this module.

4.1.3 IDL

IDL <IBOX Datapath Left> contains bits 6-17 of the IBOX memory address path and program counter logic, and bits 0-17 of the operand data path logic.

4.1.4 TAG IBOX

TAG contains control logic for IDR and IDL, along with instruction code and AC address buffers. The TAG is a 4 bit value representing the effective address of an instruction which has been assigned to the IBOX by IPUT <IBOX Micro-Code>. The TAG is used to address 16 word register files into which will be placed information pertinent to the instruction, such as instruction code, global bit, AC address, effective address calculation, memory operand, jump target address, etc. IPUT accepts priority trap requests from EBOX control and IBOX state logic, and thereupon directs address and operand path flow and TAG assignments.

4.1.5 MVR EBOX

Move Datapath Left

MVR contains bits 9-17 and 27-35 of the fast execution path of the EBOX. It executes binary adds and subtracts, decimal adds and subtracts, Boolean functions, halfword operations, and mask and test operations. A duplicate copy of all AC sets and temporary registers <referred to as the master AC backup> is located on this module.

4.1.6 MVL

MVL is physically identical to MVR, and performs the same function as MVR with logical bits 0-8 and 18-26 of the fast execution path of the EBOX.

4.1.7 ESE EBOX

Ebox setup

ESE contains EBOX microcode RAMS which control instruction execution during the first <1st 22ns period after which operands become available> and subsequent cycles. First cycle control comes from fast 256X4 RAMS, which setup EBOX control fields directly from lookup tables addressed by the instruction code. Second cycle control comes from slow 4KX1 RAMS which are also addressed by the instruction code. Third and subsequent cycle control comes from the same slow 4KX1 RAMS addressed by the next address path.

4.1.8 CRA EBOX

Control RAM

CRA contains slow 4KX1 RAMS which control sections of the EBOX during the 2nd and subsequent cycles of execution. During the first cycle some control fields default to specific options which allow data path gating via forwarding controls. Next address control, APR flags, and some of the PC flags are also located on this module.

4.1.9 SHR EBox

Shift Right

SHR contains bits 18-35 of the shifter <which performs a 72 input, 36 output left shift>, bits 18-35 of two REGFILES and two copies of the current AC set <one normally addressed by bits 32-35 of the effective address calculation, the other normally addressed by the AC field of the instruction>, and bits 18-35 of the master AC set, a RAM file containing 8 AC sets plus 128 temporary registers.

4.1.10 SHL EBOX

SHL contains bits 0-17 of the shifter, register files, AC copies, and master AC set, and is physically identical to SHR.

4.1.11 SCA EBOX

Shift control addr

SCA contains a 13 bit shift control path with addition and subtraction capability. Also located on this module are an accounting meter, interval times, 1MS/10US clock, pipeline stopping controls, PC flags associated with shift and floating point operations, and FRU code generation. The FRU code, generated as a result of an EBOX, IBOX, or FPA error, points to the 1st and 2nd most probable modules causing the failure.

4.1.12 FPI

FPI <Floating Point Interface> is the interface between the EBOX, IBOX and the floating point array modules. It contains OP3L and OP3R, which are 16 word register files addressed and loaded by the IBOX. OP3L is a duplicate of the operand buffer in the IBOX <OP3L contains words fetched from address E>. OP3R, for doubleword instructions, contains words fetched from address E+1. OP3R, for byte instructions, contains words addressed by the byte pointer or incremented byte pointer. Also located on this module is a small control storage which sequences the FPA, data paths which produce an 18 bit multiplier from OP3L and OP3R, a reciprocal lookup for divide instructions <so that the multiply array may also be used for divide>, and a priority encode for post normalization shift determination.

next page ~~What's that~~?

4.1.13 FPA

FPA generates an 18 bit hi-order spillover from an 18X72 bit multiply. It is physically identical to FPB-FPE. Recode multipliers and Wallace tree adders are used to develop a partial product in 22ns, with a binary ALU to achieve full product in another 22ns. 18 bit sections of the multiplier are gated into the array in pipeline fashion at a 22ns rate. The array is therefore capable of retiring 18 bits per 22ns. Modulo 3 base numbers are generated from both the multiplier and the multiplicand, and a parallel multiply occurs against these numbers. The results are compared to the modulo 3 generation off the full product to check for a multiply array error <called residue check>. Each array card is separately checked for fault isolation.

4.1.14 FPB

FPB contains bits 0-17 of the 18X72 array, in addition to bits 0-17 of the AC addressed by the AC field.

4.1.15 FPC

FPC contains bits 18-35 of the 18X72 array, in addition to bits 18-35 of the AC addressed by the AC field.

4.1.16 FPD

FPD contains bits 36-53 of the 18X72 array, in addition to the hi-order 18 bits of the AC addressed by the AC field +1.

4.1.17 FPE

FPE contains bits 54-71 of the 18X72 array, in addition to the lo-order 18 bits of the AC addressed by the AC field +1.

4.2 EXECUTION UNITS COMMON CONTROL

4.2.1 General

The functional logic of the 2080 execution units interfaces to, and is supported by certain common controls which maintain a consistent convention through all modules. These are the diagnostic interface, error control logic, microcode control word, and system clocks.

4.2.2 Diagnostic Interface (1)

The diagnostic interface is the vehicle through which the console A> presets execution unit registers to desired states, B> forces control decodes to manipulate the data paths, C> gains visibility into the data and control paths, and D> loads control storage areas. The 100141 shift register is used to achieve all of these. It provides the ability to locate the scan in/scan out mechanism as a functional part of the data path. All 100141 shift registers used in this manner and strung together <one's shift output connected to the other's shift input> are considered to be part of the active scan path. Other 100141 shift registers, not part of functional logic, are used for presetting signals to desired states, and scanning data path and control states independently of E unit operation. These are linked together as the passive data path.

```
       MSB     LSB   MSB     LSB       MSB     LSB   MSB     LSB
        +-------+     +-------+         +-------+     +-------+
   +--->! DSR   !---->! DSR   !-->+  +->! DSR   !---->! DSR   !-->+
   !    +-------+     +-------+   !  !  +-------+     +-------+   !
   !        ^  MODULE A           +  +         MODULE B           !
   !        !                     !  !                            !
   !    data path                 +--+                            !
                     MSB                 LSB        !             !
                    +-----------------------+       !             !
            +------>! DIAGNOSTIC SHIFT REG  !------>+             !
            !       +-----------------------+                     !
            !                   ^  !                              !
            !                   !  !                              !
            !            CONSOLE DATA PATH                        !
            !                LOAD/READ                            !
            +----<----------------------------------<-------------+
```

4.2.3 Active Scan Paths

The 2080 CPU contains three active scan paths. Each is sourced at the console and consists of shift registers which, under normal operation <not clocks stopped due to error or console control> perform a parallel load when their corresponding system clocks occur. When the system clocks have been blocked by error freeze or a clock control stop, the console may perform one of two operations; A> a parallel load via its own clock, B> a shift (normally hi-order to lo-order) of 1. The three active scan loops encompass the free-running memory access logic in the MBOX, IBOX, and IOBCX, error stoppable logic in the IBOX and EBOX, and error-stoppable logic in the FPA.

4.2.4 Passive Scan Path

The 2080 CPU contains two passive scan loops. One passive scan loop is dedicated to the CLK CONTROL module, while the other encompasses all other 2080 CPU passive controls and readouts. A command ENABLE from the console will be used to gate those bits which are used as presets or RAM write-enables once the desired section of a passive loop has been loaded.

4.2.5 Scan Read And Load

A 16 bit shift register in the console completes the scan path loop. The register is loaded at the console through a 16 bit mask, thus permitting individual bits to be loaded without altering the masked bits. The error-stoppable EBOX/IBOX loop is estimated to be 1248 bits in length. This means that, in order to shift the contents out of positions 1232 to 1247 and into the console shift reg at bits 0-15, a diag shift control must be asserted for this loop, followed by 16 diag clocks to all shift registers in this loop. Conversely, in order to shift the console shift reg contents back to positions 1232-1247, the diag shift control must be asserted, followed by 1232 diag clocks.

4.2.6 Error Control Logic (2)

Error control logic for 2080 is designed in such a way as to support 3 RAMP goals; a) detection and recording of intermittent as well as solid failures, b) hardware recovery from a high percentage of CPU errors, c) fault isolation <to a module> of a high percentage of errors prior to running of diagnostics and, in the case of intermittents, without taking the system down. 20 to 25 % of 2080 logic is devoted to the implementation of these features.

4.2.6.1 ERROR DETECTION AND RECORDING - Odd parity is generated for every bus greater than 4 bits wide. A high emphasis is placed in three areas:

1. Fault isolation by module
2. Console access to internal data paths
3. 'Catching' intermittent errors as close as possible to the point of failure.

36 bit data paths develop 1 parity bit for every 9 data bits. Microcode storage areas contain 1 parity bit per word except in the case of EBOX control storage, which contains 2 parity bits for module fault isolation. The main ALU is compared to a duplicate of itself every cycle. Parity generation occurs at the output of boolean, ALU, and shift functions; parity checking occurs at the input of boolean, ALU, and shift functions. Parity is modified and propagated with 1 and 2 bit shifts. A modulo 3 residue checking system is employed for verification of all FPA multiply and divide results.

The scan-in, scan-out mechanism described in section 3.1 is the method by which console may assert control or data path signals, and read out a high percentage of the data path. 10C141 shift registers constitute approximately 12% of 2080 execution unit logic.

Diagnostic control has reserved to it a section of the microcode control storage, thus allowing it to route, via set up of next address to read a microword with desired control decodes, virtually all registers <if not already a 141> through mux paths and into 141s for scanning. This includes access to 16 word register files and RAM arrays.

Strategic registers are frozen if an error is detected at their output. This approach helps in locating the source of an error likely to be propagated through the system or lost, by the time the IBOX, EBOX, and FPA module clocks can be stopped.

4.2.6.2 HARDWARE RECOVERY - When an error has been detected, an error latch is set at the module level. The error condition may then block the next clock to the failing register. The error signal is also sent to the SCA module where it is funneled into the error stop latch, which will then stop the next clock 0, 1 , 2, and 3 in succession. A 2 bit code is loaded into the FRU reg which identifies to the console the status of the IBOX SEL I tag, a 4 bit value pointing to the next instruction to be executed. The code is used by the console to determine the number of PC instructions to step back in preparation for a retry of the failing instruction stream. The console may also elect to arm the EBOX AUTO RETRY latch, which will force a microcode trap on an error stop condition. A restart routine will branch on the value of the restart code to a microword containing the appropriate command to the IBOX. The RETRY CODE is defined as follows:

RETRY CODE	RECOVERY
0	restart from PC BUFFER addressed by SEL I-1
1	restart from PC BUFFER addressed by SEL I-2
2	restart from PC BUFFER addressed by SEL I-2
3	restart from PC BUFFER addressed by SEL I-3

THE FRU REG

```
+--------------------+--------------------+----------+---------+
!        FRU1        !        FRU2        !  RETRY   !  SPARE  !
!                    !                    !  CODE    !         !
+--------------------+--------------------+----------+---------+
         0-5                  6-11           12-13     14-15
```

Successful retrys appear to be possible for nearly all instructions with only 1 result to be stored. However in instructions with multi-word results which overlay previous operands, successful retry is not probable.

4.2.6.3 FAULT ISOLATION - For each error stop, the FRU REG is loaded at the SCA module. The FRU REG is 16 bits in length and identifies to the console the 1st and 2nd most probable failing modules, and whether a failing module containing reloadable storage requires a transfer from backup storage. The errors are prioritized so that the source of the error will be identified, rather than the path which merely propagated or received erroneous data. Since the errors are frozen dynamically, and fed through a priority encode to generate the FRU CODE, an accurate method of trouble-shooting one error at a time is achieved. A 6 bit code, labeled FRU1, identifies the most probable failing module. Another 6 bit code, FRU2, identifies the 2nd most probable failing module < if no FRU detected, the code is set to all ones>. The high order bit off in the code indicates that reloadable storage has possibly failed in one of the following areas:

IDL,IDR AC set
IPUT microcode
ESE microcode
CRA microcode
SHF,SHL AC sets
SHF,SHL MAC set
FPB-E AC sets
FPI divide lookup
FPI microcode

How does console know which error occurred?

Each of these areas has a backup. In the case of an AC set error, console will rewrite the 16 word set from the MAC backup. In the case of a microcode or divide lookup error, the diagnostic load address reg for that storage contains the failing address; therefore console may reload that word with data from it's own microstorage backup. In the case of a MAC error, the hi order 4 bits of the write address are held at the MAC backup on MVL and MVR, and will be used by the console to reload the failing 16 word set from MAC backup. Below is a list of errors versus FRU codes<0=highest priority>.

ERROR	FRU1	FRU2
I MICROCODE ERROR	0 TAG	
ESE MICROCODE ERROR	1 ESE	
CRA MICROCODE ERROR	2 CRA	
DIVIDE LOOKUP ERROR	3 FPI	
I AC ERROR BYTE 2-3	4 IDR	
I AC ERROR BYTE 0-1	5 IDL	
R REG OR AC ERROR	6 SHR	
L REG OR AC ERROR	7 SHL	
R MAC OR Z ERROR	10 SHR	
L MAC OR Z ERROR	11 SHL	
FPE AC ERROR	14 FPE	
FPD AC ERROR	15 FPD	
FPC AC ERROR	16 FPC	
FPB AC ERROR	17 FPB	
M DATA RIGHT	40 MDR	
M DATA LEFT	41 MDL	
OP ERROR BYTE2-3	42 IDR	
OP ERROR BYTE 0-1	43 IDL	

what does it mean to have the high order bit on?

ERROR	FRU1	FRU2
I ADDR ERR BYTE 2-3	44 IDR	
I ADDR ERR BYTE 0-1	45 IDL	
M TAG ERROR	46 MCC	
I TAG ERROR	47 TAC	
MVR OUTPUT ERRØR	50 MVR	
MVL OUTPUT ERROR	51 MVL	
MVR ALU ERROR	52 MVR	
MVL ALU ERROR	53 MVL	
SCAD OUTPUT ERROR	54 SCA	
MVR INPUT ERROR	55 MVR	
MVL INPUT ERROR	56 MVL	
SCA INPUT ERROR	57 SCA	
FPI OP BYTE 2-3 ERROR	60 FPI	42 IDR
FPI OP BYTE 0-1 ERROR	61 FPI	43 IDL
FPE RESIDUE CHECK	62 FPE	
FPD RESIDUE CHECK	63 FPD	
FPC RESIDUE CHECK	64 FPC	
FPB RESIDUE CHECK	65 FPB	
FPA RESIDUE CHECK	66 FPA	

4.2.7 Microcode Control Specification (3)

The 2080 EBOX control storage is comprised of 4.5K words of 102 bits each. A wide word permits coding flexibility without extending control storage depth. The fields are defined as follows:

CRA MODULE

BITS	NAME	TIMING
0-11	NEXT ADDRESS	EARLY
12-17	NA SWITCH	EARLY
18	CALL	INTERMEDIATE
19-21	X	INTERMEDIATE
22-23	Y	INTERMEDIATE
24	SPARE	
25-27	D	INTERMEDIATE
28	L EARLY	EARLY
29-30	J	INTERMEDIATE
31-33	K	INTERMEDIATE
34	SC ALU	INTERMEDIATE
37-38	SP FUNCTION (LO 2 BITS)	INTERMEDIATE
39	PARITY 0-34,37-47	INTERMEDIATE
40	MQ	LATE
41	RF	LATE
42-44	SC	LATE
45-46	FE	LATE
47	T	LATE

ESE MODULE

BITS	NAME	TIMINC
35-36	SP FUNCTION (HI 2 BITS)	INTERMEDIATE
48-49	X1	LATE
50-51	Y1	LATE
52-55	ALU	EARLY
56-58	CARRY	EARLY
59	OP1/WR REG FILE	EARLY
60-63	SW/ X REG RD	INTERMEDIATE
64-67	R/ Y REG RD	INTERMEDIATE
68-69	COMP	INTERMEDIATE
70	LAST CYCLE	INTERMEDIATE
71-73	STORE CONTROL	INTERMEDIATE
74-77	FLAG CONTROL	INTERMEDIATE
78	PARITY 35-36,48-101	INTERMEDIATE
79	L	LATE
80-83	AC/ REG WR CTRL	LATE
84-101	NUMBER FIELD 0-17	LATE

4.2.7.1 Next Address Field <Bits 0-11> - In any given instruction, the first cycle control word comes from fast 256X4 RAMS addressed by the instruction code, the second cycle control word <22ns later> comes from 4KX1 RAMS also addressed by the instruction code, and third and subsequent control words come at 44ns intervals from the same 4KX1 RAMS addressed by the next address path. If control bits 12-17 are not 0, the hi-order next address is determined by hi-order next address field bits, and the next address lo-order is determined by the inclusive or of lo-order next address field bits and the selected skip or dispatch bits.

4.2.7.2 THE NA SWITCH <BITS 12-17> - The NA SWITCH field determines the source of the NEXT ADDRESS path.

decode	function
0	NA FIELD 0-11 to NA BUS 0-11
1	NA FIELD 0-10 to NA BUS 0-10, INT REQ inclusive or with NA FIELD 11 to NA BUS 11
2	NA FIELD 0-10 to NA BUS 0-10, instr AC ADDR not=0 inclusive or with NA FIELD 11 to NA BUS 11
3	NA FIELD 0-10 to NA BUS 0-10, X REG bit 0 inclusive or with NA FIELD 11 to NA BUS 11
4	NA FIELD 0-10 to NA BUS 0-10, X REG bit 18 inclusive or with NA FIELD 11 to NA BUS 11
5	NA FIELD 0-10 to NA BUS 0-10, SCAD ALU bit 0 inclusive or with NA FIELD 11 to NA BUS 11
6	NA FIELD 0-10 to NA BUS 0-10, SC REG bit 0 inclusive or with NA FIELD 11 to NA BUS 11
0 7	NA FIELD 0-10 to NA BUS 0 to 10, F BUS 0 inclusive or with NA FIELD 11 to NA BUS 11 (from ALU operation of the 1st half of the current microcycle if L EARLY, from ALU operation of the 2nd half of the previous microcycle if not L EARLY)

10	NA FIELD 0-10 to NA BUS 0-10, ALU 0-35 not=0 inclusive or with NA FIELD 11 to NA BUS 11 (same timing as decode 7)
11	NA FIELD 0-10 to NA BUS 0-10, ALU 0-17 not=0 inclusive or with NA FIELD 11 to NA BUS 11 (same timing as decode 7)
12	NA FIELD 0-10 to NA BUS 0-10, ALU 18-35 not=0 inclusive or with NA FIELD 11 to NA BUS 11 (same timing as decode 7)
13	NA FIELD 0-10 to NA BUS 0-10, PC section # not=0 inclusive or with NA FIELD 11 to NA BUS 11
14	NA FIELD 0-10 to NA BUS 0-10, USER MODE bit inclusive or with NA FIELD 11 to NA BUS 11
15	NA FIELD 0-10 to NA BUS 0-10, CARRY OUT of ALU inclusive or with NA FIELD 11 to NA BUS 11 (same timing as decode 7)
16	NA FIELD 0-10 to NA BUS 0-10, SC REG not=0> inclusive or with NA FIELD 11 to NA BUS 11
17	NA FIELD 0-10 to NA BUS 0-10, SCAD ALU not=0 inclusive or with NA FIELD 11 to NA BUS 11
20	NA FIELD 0-10 to NA BUS 0-10, SC REG bit 1 inclusive or with NA FIELD 11 to NA BUS 11
21	NA FIELD 0-10 to NA BUS 0-10, OP2=AC anded with EACALC global inclusive or with NA FIELD 11 to NA BUS 11
22-57	spare
60	NA FIELD 0-7 to NA BUS 0-7, Instr AC addr inclusive or with NA FIELD 8-11 to NA BUS 8-11
61	NORMALIZE: NA FIELD 0-7 to NA BUS 0-7, F BUS 0,8, 9,-1 inclusive or with NA FIELD 8-11 to NA BUS 8-11 (same timing as decode 7)
62	ENORMALIZE: NA FIELD 0-7 to NA BUS 0-7, F BUS 0, 11, 12, -1 inclusive or with NA FIELD 8-11 to NA BUS 8-11 <same timing as decode 7>
63	DIVIDE: NA FIELD 0-8 to NA BUS 0-8, SC REG bit 0, X REG bit 0, CARRY OUT of ALU inclusive or with NA FIELD 9-11 to NA BUS 9-11 (same timing as decode 7)
64	RETURN: AD STACK 0-11 inclusive or with NA FIELD 0-11 to NA BUS. The AD STACK is a 16 word LIFO <last in,first out> file containing NA BUS values stored under the CALL microorder.

65	MUL: NA FIELD 0-8 to NA BUS 0-8, SC REG 0, MQ REG 34 and 35 inclusive or with NA FIELD 9-11 to NA BUS 9-11
66	BYTE: NA FIELD 0-8 to NA BUS 0-8, F BUS 12 anded with PC sect not=0, SCAD ALU bit 0, FIRST PART DONE flag inclusive or with NA FIELD 9-11 to NA BUS 9-11
67	ALU SIGN: NA FIELD 0-9 to NA BUS 0-9, Y REG 0, ALU not=0 inclusive or with NA FIELD 10-11 to NA BUS 10-11
70	SIGNS: NA FIELD 0-9 to NA BUS 0-9, X REG 0, Y REG 0 inclusive or with NA FIELD 10-11 to NA BUS 10-11
71	RETRY: NA FIELD 0-9 to NA BUS 0-9, RETRY CODE <FRU REG 12-13> inclusive or with NA FIELD 10-11 to NA BUS 10-11
72-77	<TO BE SUPPLIED>

4.2.7.3 CALL <BIT 18> - The CALL microorder will cause the address dispatch stack to be loaded with the value of the previous NA BUS. The AD STACK is a 16 word LIFO register file, and may be read by the RETURN microorder.

4.2.7.4 X FIELD <BITS 19-21> - The X field controls inputs into the X BUS via the X MUX at the SHL and SHR modules. The field decodes are as follows:

decode	function
0	X AC 0-35 to X BUS 0-35, X AC bit 0 to X BUS -2 and -1
1	XREG FILE 0-35 to X BUS -2-33,zeros to 34,35
2	XREG FILE 0-35 to X BUS -1-34,zero to 35, XREG FILE bit 0 to X BUS -2
3	XREG FILE 0-35 to X BUS 0-35, XREG FILE bit 0 to X BUS -2 and -1
4	D MUX 0-35 to X BUS 0-35, D MUX bit 0 to X BUS -2 and -1
5	XREG FILE bit 0 to X BUS -2-35
6	# FIELD 0-17 to X BUS 0-17, # FIELD 0-17 to X BUS 18-35, # FIELD 0 to X BUS -2 and -1
7	Z BUS 0-35 to X BUS 0-35, Z BUS bit 0 to X BUS -2 and -1

4.2.7.5 Y FIELD <bits 22-23> - The Y field controls inputs into the Y BUS via the Y MUX at the SHL and SHR modules. The field decodes are as follows:

decode	function
0	Y AC 0-35 to Y BUS 0-35
1	YREG FILE 0-35 to Y BUS 0-35
2	D MUX 0-35 to Y BUS 0-35
3	MQ 0-35 to Y BUS 0-35

4.2.7.6 D FIELD <bits 25-27> - The D field controls the D MUX at the SHL and SHR modules. The field decodes are as follows:

decode	function
0	L1 REG 0-35 to D MUX 0-35
1	L1 REG 1-35 to D MUX 0-34, MQ REG 0 to D MUX 35,MQ REG 1-35 to MQ REG 0-34, zero to MQ REG 35
2	L1 REG 0-34 to D MUX 1-35, L1 REG 35 to MQ REG 0, zero to D MUX 0, MQ REG 0-34 to MQ REG 1-35
3	L1 REG 0-33 to D MUX 2-35, L1 REG 34-35 to MQ REG 0-1, zeros to D MUX 0-1, MQ REG 0-33 to MQ REG 2-35
4	Z MUX 0-35 to D MUX 0-35
5	Z MUX 6-35 to D MUX 6-35, SCAD ALU 0-5 to D MUX 0-5
6	Z MUX 9-35 to D MUX 9-35, SCAD ALU 0-8 to D MUX 0-8
7	Z MUX 12-35 to D MUX 12-35, SCAD ALU 0-11 to D MUX 0-11

4.2.7.7 L EARLY <bit 28> - The L EARLY bit determines one of two modes of 44ns cycle operation:

1. L EARLY off indicates that an ALU or MOVE operation occurring in the second half of a 44ns microcycle may store results in the second half of the next microcycle if a write to REG FILE or AC is issued. In this mode, a read of REG FILE is performed during the first half of the current microcycle and may be used for an ALU or MOVE operation in the second half of the current microcycle. L EARLY off indicates that L1 BUS is valid at CLK 3 of the first half of the next micro cycle. During the first cycle of any instruction, L EARLY is forced off.

   L EARLY off also indicates that all SHIFT MATRIX inputs are clocked at CLK 2 of the 1st half of the current microcycle <X REG, Y REG, SC REG, SCAD ALU> and SHIFT MATRIX results are clocked into Z MUX at CLK 3 of the 2nd half of the current microcycle.

2. L EARLY on indicates that an ALU or MOVE operation occurring in the 1st half of a microcycle may store results in the first half of the next microcycle if a write to REG FILE or AC is issued. In this mode, a read of REG FILE is performed during the second half of the current microcycle and may be used for an ALU or MOVE operation in the first half of the next microcycle. L EARLY on indicates that L1 BUS is valid at CLK 3 of the 2nd half of the current microcycle.

L EARLY on also indicates that all SHIFT MATRIX inputs are loaded at CLK 2 of the 2nd half of the current microcycle, and SHIFT MATRIX results are clocked into Z MUX at CLK 3 of the 1st half of the next microcycle.

EXAMPLE: -L EARLY on in 1st cycle, off in 2nd microcycle

```
        !  FIRST CYCLE  !      SECOND MICROCYCLE       !
        !               !               !               !               !

        0               0               0               0               0
        +               +---------------+               +---------------+
        ! -L EARLY A    !       2       !           3   !       2
        +---------------+       !       +---------------+       !
          !         !         X REG                 !         X REG
          !       Z MUX       Y REG               Z MUX       Y REG
          !                   FE REG                          FE REG
          !                   SC REG                          SC REG
         DLY
          !     2               2               2               2
          !     +               +---------------+               +--------
          +---> ! -L EARLY B    !               !               !
                +_______________+               +---------------+
                  !     0                               0
                  !     !                               !
                  !     !                               !
                  !   F BUS                           F BUS
                  !   RF MUX                          RF MUX

                  !
                 DLY
                  !
                  !         1               1               1
                  !         +               +---------------+
                  +-------> ! -L EARLY C    !               !
                            +---------------+               +------------
                              !     3                               3
                              !     !                               !
                              !   L1 REG                          L1 REG
                              !
                             DLY
                              !
                              !         0               0               0
                              !         +               +---------------+
                              +-------> ! -L EARLY D    !               !
                                        +---------------+               +
                                                2
                                                !
                                              STORE
```

4.2.7.8 J FIELD <bits 29-30> - The J field controls the J input to the SC ALU <SCA module> via the J MUX. The field decodes are as follows:

decode	function
0	SC REG 0-11 to J MUX 0-11
1	# FIELD 0-11 to J MUX 0-11
2	X REG 18 TO J MUX 0-3, X REG 28-35 to J MUX 4-11
3	zeros to J MUX 0-5, Y REG 0-5 to J MUX 6-11

4.2.7.9 K FIELD <bits 31-33> - The K field controls the K input to the SC ALU <SCA module> via the K MUX. The field decodes are as follows:

decode	function
0	# FIELD 0-11 to K MUX 0-11
1	zeros to K MUX 0-5, Y REG 6-11 to K MUX 6-11
2	zeros to K MUX 0-5, FPA SHF ENCODE to K MUX 6-11
3	zero to K MUX 0, OE Y REG 0 with Y REG 1-8 to K MUX 1-8, zeros to K MUX 9-11
4	zero to K MUX 0, OE Y REG 0 with Y REG 1-11 to K MUX 1-11
5	FE REG 0-11 to K MUX 0-11
6	zeros to K MUX 0-7, instruction XR value 14-17 to K MUX 8-11
7	zeros to K MUX 0-7, Y REG 2-5 to K MUX 8-11

4.2.7.10 SC ALU <bit 34> - The SC ALU bit determines the SC ALU function:

0=add
1=subtract K from J

4.2.7.11 SP FUNCTION Field <bits 35-38> - The SPECIAL FUNCTION field issues control commands to the EBOX under the following decodes:

decode	function
0	no-op
1	turn on accounting meter
2	turn off accounting meter
3	select MASTER AC 0-35 to Z BUS 0-35
4	force 1s to SEL OP2
5	update accounting meter +1
6	SSEL: Send encoded port address of accepted IO interrupt to the console
7	spare
10	force 1s to SEL OP1
11	select FPA SHF ENCODE 0-5 to SC MUX 0-5
12-17	spare

4.2.7.12 PARITY 0-47 <bit 39> - PARITY 0-47 is turned on or off so that odd parity is achieved for bits 0-47.

4.2.7.13 MQ <bit40> - The MQ bit determines the mode of the MQ REG. If on, the MQ REG clock is enabled. If off, the MQ REG is held at the previously clocked value.

4.2.7.14 RF <bit 41> - The RF bit determines selection of either R BUS or A BUS to the L MUX input at the MVL and MVR modules. First cycle control of this field forces selection of R BUS. Since the RF bit is a late control, the results of the move path operation during first cycle are selected into the RF BUS. The decode is as follows:

```
0= R BUS 0-35 to RF BUS 0-35
1= A BUS 0-35 to RF BUS 0-35
```

4.2.7.15 SC FIELD <bits 42-44> - The SC field controls the SC REG and SC MUX on the SCA module. Bit 42 on causes a hold of previously clocked contents of the SCAD ALU output. Bit 42 off allows SC REG to clock. Bits 43-44 decode as follows:

decode	function
0	SC REG 6-11 to SC MUX 0-5
1	SC ALU 6-11 to SC MUX 0-5
2	# FIELD 12-17 TO SC MUX 0-5
3	SC ALU 3-8 TO SC MUX 0-5

A third signal causes selection of the FPA shift value 0-5 <determined from post normalization priority encoders> into the SC MUX. This signal is generated by SP FUNCTION decode=4.

4.2.7.16 FE FIELD <bits 45-46> - The FE field controls the FE REG and FE MUX on the SCA module. Bit 45 on causes a hold of previously clocked contents of the FE MUX output. Bit 45 off allows FE REG to clock. Bit 46 selects FE MUX as follows:

Bit 46 off	K MUX 0-11 to FE MUX 0-11
Bit 46 on	SCAD ALU 0-11 to FE MUX 0-11

4.2.7.17 T <BIT 47> - T bit on will cause the current microword to hold for an additional 44ns cycle.

4.2.7.18 X1 <bits 48-49> - The X1 field controls the X1 MUX at MVL and MVR. During the first cycle of an instruction, forwarding compare logic from TAC and SH modules determine selection. During second and subsequent cycles firmware has control with the following field decodes:

decode	function
0	F BUS 0-35 to X1 BUS 0-35, F BUS 0 to X1 BUS -2,-1
1	RF BUS 0-35 to X1 BUS 0-35, RF BUS 0 to X1 BUS -2,-1
2	X BUS -2-35 to X1 BUS -2-35
3	OP2 BUS 0-35 to X1 BUS 0-35,OP2 BUS 0 to X1 BUS -2,-1

4.2.7.19 Y1 <bits 50-51> - The Y1 field controls the Y1 MUX and AF MUX at MVL and MVR < an enable to Y1 in first cycle comes from the IBOX>. During the first cycle, forwarding compare logic from TAC and SH modules determine selection. During second and subsequent cycles firmware has control with the following decodes:

decode	function
0	F BUS 0-35 to AF,Y1 BUS 0-35
1	RF BUS 0-35 to AF,Y1 BUS 0-35
2	X BUS 0-35 to AF,Y1 0-35
3	Y BUS 0-35 to AF,Y1 0-35

ALU input -2,-1 are equal the value of Y1 BUS 0

4.2.7.20 ALU <bits 52-55> - The ALU field controls the 36 bit ALU on the MVL and MVR modules. Carry into 35 must be asserted for all subtract operations. For BCD, bits 0,9,18,27 are ignored <carry will cross from 10 to 8, 19 to 17, 28 to 26> during the ALU operation and forced to zero at the F BUS. The decodes are as follows:

decode	function
0	X1 plus Y1 BCD (32 bit add, 1-8,10-17,19-26,28-35)
1	X1 minus Y1 BCD (32 bit subtract)
2	Y1 minus X1 BCD (32 bit subtract)
3	0 minus Y1 BCD (32 bit subtract)
4	X1 plus Y1 Binary (36 bit)
5	invalid op
6	Y1 minus X1 Binary (36 bit)
7	invalid op
10	X1 equal Y1
11	X1 EOR Y1
12	X1 + Y1(inclusive or)
13	X1
14	not Y1
15	Y1
16	X1.Y1(logical and)
17	zeros

4.2.7.21 CARRY <bits 56-58> - The CARRY field controls the carry in to 17 and carry in to 35 of the 36 binary ALU and 32 bit decimal ALU. Decodes are as follows:

DECIMAL OPERATION <bits 52,53=00>

decode	function
0-1	invalid op
2	carry into 35
3-5	invalid op
6	no carry into 35
7	invalid op

BINARY OPERATION <bits 52-53 not equal 00>

decode	function
0	sign Y= Cn to 35, 36 bit ALU
1	no Cn to 35, 36 bit ALU
2	no Cn to 35, 18 bit ALUs
3	no Cn to 35, 18 bit ALUs if EACALC not global no Cn to 35, 36 bit ALU if EACALC global
4	Cn to 35, force Cn to 17
5	Cn to 35, 36 bit ALU
6	Cn to 35, 18 bit ALUs
7	Cn to 35, force Cn to 17 if EACALC not global Cn to 35, 36 bit ALU if EACALC global

4.2.7.22 OP1/WR REG FILE <bit59> - During first cycle bit 59 controls the OP1 MUX at MVL and MVR as follows:

59=0	AF MUX 0-35 to SEL OP1 0-35
59=1	AF MUX 0-17 to SEL OP1 18-35,AF MUX 18-35 to SEL OP1 0-17

During second and subsequent cycles, OP1 MUX control is forced to zero <AF 0-35 to SEL OP1 0-35>, and bit 59 performs the following operation in conjunction with the L EARLY bit:

Bit 59	L EARLY<bit 28>	function
0	0	read REG FILE in 1st half current cycle
0	1	read REG FILE in 2nd half current cycle
1	0	write REG FILE in 2nd half next cycle, read REG FILE in 1st half current cycle
1	1	write REG FILE in 1st half next cycle, read REG FILE in 2nd half current cycle

4.2.7.23 SW/ X REG RD <bits 60-63> - The SW/ X REG RD field performs a dual function. It addresses the X REG FILE words 0-15 on SHL and SHR modules, and controls the swapping and merge operation of the SW MUX on MVL and MVR modules. SW MUX is controlled by the following decodes:

Left swap control <bits 60-61>

decode	function
0	OP1 18-35 to PASS 0-17
1	OP1 0-17 to PASS 0-17
2	OP2 18-35 to PASS 0-17
3	OP2 0-17 to PASS 0-17

Right swap control <bits 62-63>

decode	function
0	OP1 18-35 to PASS 18-35
1	OP1 0-17 to PASS 18-35
2	OP2 18-35 to PASS 18-35
3	OP2 0-17 to PASS 18-35

4.2.7.24 R/Y REG RD <BITS 64-67> - The R/Y REG RD field performs a dual function. It addresses the Y REG FILE words 0-15 on SHL and SHR modules, and controls the operation of the R MUX on MVL and MVR modules. R control bits are decoded as follows:

decode	function
0	zeros to R BUS 0-17, PASS 18-35 to R BUS 18-35
1	zeros to R BUS 0-35
2	PASS 0-17 to R BUS 0-17, zeros to 18-35
3	zeros to R BUS 0-35
4	SEL OP2 bit 0 to R BUS 0-17,PASS 18-35 to R BUS 18-35
5	PASS 0-35 to R BUS 0-35
6	PASS 0-17 to R BUS 0-17, SEL OP2 bit 0 to R BUS 18-35
7	COMPARE BUS 0-17 to R BUS 0-17
10	ones to R BUS 0-17, PASS 18-35 to R BUS 18-35
11	ones to R BUS 0-35
12	PASS 0-17 to R BUS 0-17, ones to R BUS 18-35
13	ones to R BUS 0-35
14	SEL OP2 bit 18 to R BUS 0-17,PASS 18-35 to R BUS 18-35
15	-PASS 0-35 to R BUS 0-35
16	PASS 0-17 to R BUS 0-17, SEL OP2 bit 18 to R BUS 18-35
17	-COMPARE 0-35 to R BUS 0-35

4.2.7.25 COMP <bits 68-69> - The COMP field determines the mode of operation of the comparators on the MVL and MVR modules per the following decodes:

```
     Bit 68 on= compare to ones<7s>
               a) compare COMP BUS 1-34 to ones
               b) compare F BUS 0-35 to ones
     bit 68 off= compare to zeros<not 7s>
               a) compare COMP BUS 1-34 to zeros
               b) compare F BUS 0-35 to zeros
     Bit 69 on= compare to 1
               a) compare COMP BUS 35 to one
     Bit 69 off= compare to zero
               a) compare COMP BUS 35 to zero
```

4.2.7.26 LAST CYCLE <bit 70> - The LAST CYCLE bit controls the update of the PC, Instruction Code, AC addresses, and selection of CE, CI, and CL MUXes during the first cycle of the next instruction. When the last cycle bit is on and L EARLY is also on, PC, IC, and AC addresses are updated in the next 22ns clock period (if in 44ns microcycle,during the 2nd half of the cycle containing the last cycle bit. When the LAST CYCLE bit is on and L EARLY is off, PC, IC, and AC addresses are updated in the 1st 22ns period following this microcycle.

CE, CI, and CL MUXes select between first and subsequent cycle control words for early <CLK 0>, intermediate <CLK 1 or 2> and late <CLK 3 or 0> control bits. LAST CYCLE bit on causes fast 256X4 RAMS and forwarding controls to be selected into the control word of the next cycle, which represents the first cycle of the next instruction.

An interrupt may block the operation of the LAST CYCLE bit, thus allowing slow microcode to trap to an interrupt handling routine. Once this routine is completed, a LAST CYCLE microorder must again be issued to cause instruction execution to continue.

Instruction invalid coming from the IBOX at LAST CYCLE will cause the microword containing the LAST CYCLE order to hold if no interrupts are pending. L EARLY will be forced off in the next cycle so that only one store operation will occur.

4.2.7.27 STORE CONTROL <bits 71-73> - The STORE CONTROL field causes store, MBOX, and FPA operations to occur per the following decodes:

decode	function
0	no op
1	Write MEM with data on L BUS. Write AC if EACALC is an AC. L EARLY on will cause data to latch at M or write to occur at AC in the 1st half of the next cycle. L EARLY off will cause data to latch at M or write to occur at AC in the 2nd half of the next cycle.
2	Write to AC and MASTER AC. L EARLY on will cause the write to occur in the 1st half of the next cycle. L EARLY off will cause the write to occur in the 2nd half of the next cycle.
3	Write AC and MAC if AC field not=0. Timing is the same as decode 2.
4	Write MAC only with data on L BUS. L EARLY on will cause the write to occur in the 1st half of the next cycle. L EARLY off will cause the write to occur in the 2nd half of the next cycle.
5	Start FPA at address 000, #field 13-17
7	STEP CTRS: Read the MASTER AC twice followed by an update of it's lo-order 4 address by +1. In the next microcycle write into AC copies twice followed by an update of their write address by +1. This microorder repeated 8 times will cause a load of all AC copies with an AC set from the MASTER AC block.

4.2.7.28 FLAG CONTROL <bits 74-77> - The FLAG CONTROL field causes flag and miscellaneous operations to occur under control of # FIELD and data path bits.

decode	function
0	no op
1	LD FLAGS: load PC flags from F BUS 0-12. If # field bit 5 is on protect USER and USER IO from illegal modification. If # field bit 6 is on load PCU from previous USER flag.

F BUS	flag
00	OV
01	CRY0
02	CRY1
03	FOV
04	FPD
05	USER
06	USER IO/ PCU
09	TRAP2
10	TRAP1
11	FXU
12	NO DIV

decode	function
2	PC FLAGS: set or clear selected PC flags from # field.
3	LD STATE:Set or clear selected STATE REC bits from # FIELD 0-15. Set overrides clear.

# field	function
00	set OV
01	set CRY0
02	set CRY1
03	set FOV
04	set FPD
09	set TRAP2
10	set TRAP1
11	set FXU
12	set NO DIV
13	clear OV
14	clear CRY0
15	clear CRY1
16	clear TRAP1 and TRAP2
17	clear FPD

# FIELD	FUNCTION
00	set 00
01	set 01
02	set 02
03	set 03
04	set 04
05	set 05
06	set 06
07	set 07
08	reset 00
09	reset 01
10	reset 02
11	reset 03
12	reset 04
13	reset 05
14	reset 06
15	reset 07

4 AD FLAGS: Update CRY0, CRY1, and OV from the ALU results of the current microcycle.

5 JFCL: Clear OV if OV and AC field bit 9 are on; clear CRY0 if CRY0 and AC field bit 10 are on; clear CRY1 if CRY1 and AC field bit 11 are on; clear FOV if FOV and AC field bit 12 are on. If any of these flags are cleared, force jump successful at the IBOX.

6 LD APR: Set selected APR flags from X REG 24-31, reset selected APR flags from Y REG 24-31. Set overrides reset if both are on.

7 EN APR: Enable selected APR interrupts per X REG 24-31.

10 command to IBOX per # field 14-17

#14-17 decode	function
0	preset IBOX
1	load PC from L BUS
2	read PC<via OP2 BUS> of SEL I
3	read EACALC<via OP2 BUS> of SEL I
4	read mem op<via OP2 BUS> of SEL I
5	read mem, VMA from L BUS, data via OP2 BUS
6	write mem, VMA from L BUS, data from L BUS one cycle later
7	read mem,IBOX EACALC of L BUS 18-35 plus XR 18-35 addressed by WR MUX, VMA 6-18 from PC
10	initialize PC to curr instruction
11	initialize PC to curr instr -1
12	initialize PC to curr instr -2
13	backstep SEL I -1
14	step SEL I +1
15	START IBOX
16-17	spare

11	LD PI: Set interrupt requests for PI levels 1-7 from X REG 29-35, reset interrupt requests for PI levels 1-7 from Y REG 29-35. Reset overrides set if both are on.
12	EN PI: Enable interrupts for selected PI levels per X REG 29-35.
13	Load interval timer from Y REG 0-17.
14	PDL OV: Set trap 2 if F BUS compare equals value of COMP field.
15	Read flags or meters per # FIELD 15-17 into the next micro cycle's number field 0-17.

decode	function
0	APR 24-31 to # FIELD 6-13
1	Interval counter 0-17 to # FIELD 0-17
2	Accounting meter 0-17 to # FIELD 0-17
3	PI flags to # FIELD 11-17
4	STATE flags to # FIELD 0-7
5	PC flags to # FIELD 0-12
6-7	spare

16	Load accounting meter from Y REG 0-17
17	spare

4.2.7.29 PARITY 48-101 <bit 78> - Parity 48-101 is turned on or off so that odd parity is achieved for bits 48-101.

4.2.7.30 L <bit 79> - The L bit controls the selection of the L MUX on MVL and MVR modules according to the following decode:

L=0	F BUS 0-35 to L MUX 0-35
L=1	RF BUS 0-35 to L MUX 0-35

4.2.7.31 AC/ REG WR CTRL <bits 80-83> - The AC/ REG WR CTRL field performs a dual function. IF bit 59 is on, the X and Y REG FILES will be written into in the next microcycle according to the setting of L EARLY. The X and Y REG write address is generated from bits 80-83. If bit 59 is off, the AC/ REG WR field will be used to control the addressing of the ACs at SHL, SHR, IDL, IDR, and FPB-E modules. In this mode, bits 80-83 have the following significance:

bit 80=0	WR MUX plus # FIELD 14-17 to XRD MUX <bit 80=0 and bit 59=0 causes XRD MUX to clock>.
bit 80=1	EACALC 32-35 to XRD MUX if LAST CYCLE, hold previously clocked contents if not LAST CYCLE.
bit 81=0	SC bits 2-5 to YRD MUX, # FIELD 10-13 to HMA MUX <bit 81=0 and bit 59=0 causes YRD MUX to clock>
bit 81=1	AC field 14-17 of instruction to YRD MUX if LAST CYCLE, hold previously clocked contents if not LAST CYCLE.
bit 82=0	Select XRD MUX to WR MUX
bit 82=1	Select YRD MUX to WR MUX
bit 83=0	Select CURR REC to HMA MUX
bit 83=1	Select PREV REC to HMA MUX

4.2.7.32 NUMBER FIELD 0-17 <CL bits 84-101> - The number field is an 18 bit field which is used in conjunction with other microorders as a direct input into data or control logic.

4.2.8 SYSTEM CLOCKS 4

The 2080 clock controls provide the gating of a single 181.8 Mhz master oscillator source in such a way that synchronous clock phases will result at the IO Box, IBOX, EBOX, FPA, MBOX, and memory array. The clock control logic will also include start, stop, and count mechanisms for diagnostic and error recovery purposes.

4.2.8.1 IO BOX INTERFACE TO THE SYSTEM CLOCK - Outputs from clock control module to IO Box:

IO Clock

The clock control module will send a 45.45 Mhz pulse, 11ns on, 11ns off, to the IO Box. The positive transition of this pulse will correspond to the negative transition of phase Ø at the CPU. It will be free-running at all times.

```
  Ø time
      !
------+     +-----------------+     +----
      ! PHØ !                 !     !
      +-----+                 +-----+
------------+     +-----------------+     +----
            ! PH1 !                 !     !
            +-----+                 +-----+
+-----------------+     +-----------------+     +----
                  ! PH2 !                 !     !
                  +-----+                 +-----+
      +-----------------+     +-----------------+     +----
                        ! PH3 !                 !     !
                        +-----+                 +-----+
      +-----------+           +-----------+           +----
      !  IO CLK   !   11ns    !   11ns    !   11ns    !
      +           +-----------+           +-----------+
```

Diag CLK Return

This signal is the return path of the shifted out clock control register.

CLK Error

CLK Error represents a synchronization error between the CLK CONTROL module and the CPU free-running clocks.

Inputs to Clock Control Module From IO Box:

The IO BOX sends controls to the clock control module to set up and execute the clock start/stop features. A serial line permits the IO BOX to shift into a control register a field representing a 16 bit command.

CMD to CLK

CMD to CLK is the input to the clock control register shift in path.

CLK CMD ENABLE

CLK CMD ENABLE gates a previously loaded and decoded command to control the start/stop mechanisms of the CLK Control module.

Command Bits of the Clock Control Register:

```
                          CLOCK CONTROL REG

     +---------+-----+------+------+------+------+------------+
     !  SPARE  ! TIC ! DISA ! DISE ! ESTP ! SYNC ! COUNT 0-255 !
     +---------+-----+------+------+------+------+------------+
       15-13     12    11     10      9      8        7-0
```

COUNT Field (bits 7-0)

The COUNT field represents a count of clock phases to be stepped in the MBOX, IBOX, EBOX and FPA, when clock TIC (bit 12) is on. For example, Disable A (bit 11) on will stop all clocks in the CPU at the next active phase 0 (phase 0 on, phase 1, 2, and 3 off). Following this, a count of 001 and TIC on will step all clocks in the CPU to phase 1. Count increments of 4 represent machine cycles (22ns).

SYNC (bit 8)

The SYNC command, when on will cause a load of the clock distribution chips (100141 shift register) on all CPU modules to an active phase 0, inactive phase 1, 2, and 3. The SYNC command must always be preceded by turning on and executing the DISABLE A command, and followed by turning of and executing DISABLE A. This procedure eliminates any need for adjustment of the SYNC signal going to each module. This command will be used only if the 100141 shift register is employed as a distribution method.

ESTOP (bit 9)

The ESTOP command will cause a stop of all EBOX, FPA, and EBOX controlled IBOX clocks at the phase 0 following the rise of "ERROR STOP" from the EBOX. The console may then scan the EBOX, FPA, and IBOX module phase 0 scan bits to determine if the ERROR STOP was caused by a clock sync error in one of these modules.

DISABLE E (bit 10)

The DISABLE E command will stop all EBOX, FPA and EBOX controlled IBOX clocks at the next active phase 0. Turning DISABLE E off will start the clocks in these areas in the same manner as DISABLE A.

DISABLE A (bit 11)

The DISABLE A command will stop all CPU clocks at the next active phase 0. Turning DISABLE A off will start all CPU clocks with a positive (trailing edge phase 0/negative (leading) edge phase 1.

TIC (bit 12)

The clock TIC command will step all CPU clocks the number of phases specified by the count.

4.2.8.2 CPU INTERFACE TO THE SYSTEM CLOCK -

Outputs From Clock Control Module to the CPU:

SYSTEM CLK to (module name) (total = 23, 46, or 92)

Each CPU module will receive either a 181.8 Mhz clock, two 45.45 Mhz, or the four phase clocks (depending upon decision by 2080 technology group). These clocks will be powered independently at the clock control card. Two identical sets will go to the IBOX data path cards to allow for separate control by DISABLE A and DISABLE B.

SYNC

Four sync lines will be driven to the 21 CPU modules; SYNC A will go to the MBOX, SYNC B to IBOX and MV modules, SYNC C will go to the remaining EBOX modules; and SYNC D will go to the FPA modules. The SYNC signals will be stubbed, with termination at the module farthest from the clock control. These lines are logically identical, driven by the enabled SYNC command bit.

Inputs to Clock Control Module From the CPU:

Check Phase 0

The five MBOX, along with 2 IBOX data path modules, each will send a buffered phase 0 back to the clock control. These will be compared to internal clock control phase generation to determine whether or not a synchronization error has occurred. Only free running clocks utilizing the 100141 shift register will be checked in this manner. The 7 check bits will be loaded into a register at the clock control so that its bits may be shifted out as part of the active diagnostic scan path. The serialized input and output of this scan register connect to modules adjacent to the clock control module.

ERROR STOP

ERROR STOP from the CPU SCAD module indicates that an IBOX, EBOX, or FPA error has occurred, and that all gated clocks in these areas have been stopped.

4.2.8.3 MEMORY INTERFACE TO THE SYSTEM CLOCK - Outputs From Clock Control to Memory Modules:

MEM CLK

Individually driven, free-running, 45.45 Mhz clocks will be sent to each of 16 memory array cards, and each of two repeater cards. The clock will be 11ns on, 11ns off, with positive transition corresponding to the positive transition of the IO CONSOLE CLOCK.

major sections:
5.2 move/ALU
5.8 shift/AC/MQ
5.12 shift control/meters/micro control
5.23 control storage

CHAPTER 5

EBOX

5.1 GENERAL

The 2080 EBOX is designed in such a way that maximum performance is derived from a limited number of ICs. Emphasis is placed on the fast execution of commonly used instruction groups including mask and test, full and halfword moves, arithmetic, stack, and boolean. In approaching the design, it was found to be more efficient to make an entire group run fast rather then focus on a few highly used instructions. This in turn also served to reduce considerably the amount of control logic required to deal with the exception cases.

Since instruction mix analysis showed that the above-named groups occupied from 80-90% of dynamic execution, it was decided to create a hierarchical design, with MOVE <executing move and mask&test> and ALU <executing arithmetic, stack, and boolean> were controlled from fast 256X4 RAMS, and all other instructions executed in the traditional method of microprogram control. Thus, the 2080 EBOX executes nearly 300 instructions in one or two 22ns cycles.

22ns prior to the start of execution, the IBOX supplies the EBOX with the instruction code, AC addresses <for both AC field and EACALC 32-35>, and an instruction/operand valid bit which permits the instruction execution to proceed. The instruction code addresses the fast 256X4 and slower 4kX1 RAMs simultaneously. In the next cycle <first cycle of execution>, the microword controlling the EBOX is selected from the fast RAMs. The second cycle of execution may be either a 22ns cycle or a 44ns cycle depending on the setting of the LAST CYCLE and L EARLY bits < LAST CYCLE and L EARLY both on indicate a 22ns cycle>. The second and subsequent micro words are selected from the 4kX1 RAMs each 44ns.

The fast 256X4 RAMS control only the 22ns path <MOVE, ALU>, whereas the 4kX1 RAMs control a 44ns shift and shift control path as well as utilizing the MOVE and ALU sections as part of a larger 44ns path. This provides the microprogram with a very powerful arithmetic, boolean, merge, compare, and shift capability.

5.2 MOVE/ALU PATH

The MOVE/ALU path is contained on two identical modules, MVL and MVR. It consists of two 22ns loops, each of which may receive inputs from the other, or from REC, FPA, or IBOX operand paths, under the control of X1, Y1, or RF fields.

5.3 MOVE OPERATION

Two operands are presented to the MOVE path at CLK 2, SEL OP1 and SEL OP2. SEL OP1 in the first cycle of execution receives a word from one of the following:

1. X or Y AC sets < X is addressed by EACALC 32-35, and Y is addressed by AC field> on the X or Y BUS

2. A forwarded result through L1, D, and X or Y <if current instructions requires an operand from an AC word which is in the process of being stored

3. F BUS < if conflict compare indicates that previous instruction result selected at L MUX is from F and current instruction uses it as an operand>

4. RF BUS < if conflict compare indicates that previous instruction result selected at L MUX is from RF and current instruction uses it as an operand>. SEL OP2 in the first cycle of execution receives a word from one of the following:
   a) the X AC on the X BUS <X AC addressed by EACALC 32-35>
   b) a forwarded result through L1, D, and X <if current instruction requires an operand from an AC word which is in the process of being stored>

5. The IBOX OP2 BUS if no results are being forwarded and the EACALC does not represent an AC

6. From R BUS or F BUS via the same forwarding controls applied to SEL OP1.

In second and subsequent cycles SEL OP1 and SEL OP2 inputs are controlled entirely from 4kX1 control storage RAM fields X1, Y1, and RF.

Half word instructions are executed via SW and R MUX controls. SW MUX permits merging of either left or right half of SEL OP2 with the left or right half of SEL OP1. R MUX may select either half= zeros or ones, or either half = SEL OP2 bit 0 or 18. Move, Zeros, and Ones instructions are also executed via SW and R MUX controls.

Masking for test instructions occurs at an AND function with inputs from SEL OP1 and SEL OP2. The AND true outputs connect to comparators which, under COMP field control perform a partial compare <in the first cycle> of the AND outputs to the values -1, 0, and +1. Compare results and SEL OP2 sign bit <0> go to the IBOX where skip/ jump successful determination will be made. In the case of TEST LEFT instructions, SEL OP1 receives the AC operand with it's left and right halves swapped at the OP1 MUX, and SEL OP2 receives OP2 BUS from the IBOX with zeros in it's left half, and the lo order 18 bits of the EACALC in the right half.

All MOVE path results are clocked into the RF MUX at CLK 0 of the next cycle, <CLK 0 gated by L EARLY B from the current cycle's microword>.

5.4 ALU OPERATION

ALU inputs are logically identical to MVE inputs, with the exception that in the first cycle, the Y1 MUX may be disabled to force zeros into the Y1 ALU input. This is used with an ALU decode 4 and forced carry to provide a fast add 1 or subtract 1 to either ALU half, while AF MUX, also under Y1 FIELD control, is gating the AC operand into the MOVE path.

The main ALU is 36 bits wide, divided into two 18 bit sections. an extension of two bits is made on the high order end to facilitate multiply operations, thus providing F BUS -2 and -1 to the SHL module. The ALU performs binary and decimal adds and subtracts, and boolean functions in 22ns. The ALU 0-35 may also be separated into two 18 bit sections with carry control to both.

ALU results are clocked into the F MUX at CLK 0 of the next cycle <CLK 0 gated by L EARLY B of the current cycle's microword>.

5.5 L BUS

The L BUS is used to transfer results from the R BUS <MOVE>, F BUS <ALU>, or A BUS <FPA> to the following modules:

negative L BUS <1 backpanel stub at ID>
- a) IBOX IDL and IDR
- b) MBOX MDL and MDR

positive L BUS <2 backpanel stubs, one each at SH and FP>
- c) EBOX SHL and SHR
- d) FPB,D and FPC,E

The L MUX is clocked at CLK 1. The L MUX select is valid during the CLK 1 of the 2nd 22ns period after the control word has been issued for the first cycle of execution, and is valid during the CLK 1 of the 2nd and 3rd 22ns periods after the control word has been issued for subsequent 44ns microcycles. Since it is a stubbed bus, 10ns are allowed before it's destination registers are clocked.

```
  +    +---------------+    +-----------------------------------+    +
  !    !  1ST CYCLE    !    !          2ND CYCLE                !    !
  +----+ EARLY CONTROL +----+        EARLY CONTROL              +----+
  Ø                    Ø                                        Ø

                  +    +---------------+    +--------------------------
                  !    !  1ST CYCLE    !    !        2ND CYCLE
                  +----+ LATE CONTROL  +----+       LATE CONTROL
                  3                    3

                            ^                    ^                   ^

                            !                    !                   !
                           CLK L                CLK L               CLK L
```

5.6 ERROR CONTROL

Parity generation occurs at the output of the F BUS < ALU result> and at the output of the AND function of the MOVE path. A parity bit is generated for each byte, composed of bits Ø-8, 9-17, 18-26, and 27-35, respectively. Parity checking is performed at the output of the SEL OP1 MUX, SEL OP2 MUX, and L MUX. If a parity error is detected at either SEL OP1 or SEL OP2, the following CLK Ø will set the MVL or MVR INPUT ERROR latch which will a) block the next CLK 2 to SEL OP1, SEL OP2, and ALU, and b) will signal to the error control logic on the SCA module to stop all gated IBOX, EBOX, and FPA clocks starting with CLK Ø 2Øns later, and sequentially stopping CLK 1, CLK 2, and CLK 3, in that order. If a parity error is detected at the L MUX, the following CLK 2 will set the MVL or MVR OUTPUT ERROR latch which will a) block the next CLK Ø to the L MUX and MAC BACKUP ADDRESS, and the next CLK 1 to the leading edge control of the MAC BACKUP write-enables, and b) will signal the error control logic on the SCA module to stop all gated IBOX, EBOX, and FPA clocks starting with CLK Ø 3Øns later.

The carry generates of the two duplicate ALUs are compared to determine whether an ALU error has occurred. If the results of the two ALUs combined produce an odd count, an error trigger is set at the next CLK 3, followed by a clocking of the MVL or MVR ALU ERR latch at the following CLK Ø. The MVL or MVR ALU ERR signal will cause the SCA error control to stop all IBOX, EBOX, and FPA clocks starting with CLK Ø 2Øns later.

System reset will reset F, RF, SEL OP1, SEL OP2, and L MUXes to zeros with good parity. Error reset will reset MV OUTPUT ERR, MV INPUT ERR, AND MV ALU ERR.

5.7 DIAGNOSTIC INTERFACE

The passive scan path of MVL is composed of 40 bits from the F BUS and MAC BKUP scan registers. The signals are labeled as follows:

```
        SCAN INPUT= 'DIAG PAS OUT ESE H'
        SCAN REG BITS= 'R BUS 00 H' to 'R BUS 07 H',
                       'R BUS 18 H' to 'R BUS 25 H',
                       'R BUS P00-08 H',
                       'R BUS P18-26 H',
                       'R BUS 08 H',
                       'R BUS 26 H',
                       'AC BACKUP P00-08'
                       'AC BACKUP P18-26'
                       'AC BACKUP 08 H'
                       'AC BACKUP 26 H'
                       'AC BACKUP 00 H' to 'AC BACKUP 07 H'
                       'AC BACKUP 18 H' to 'AC BACKUP 25 H'
        SCAN OUTPUT= 'DIAG PAS OUT MVL H'
```

The MVR passive scan path signals are labeled as follows:

```
        SCAN INPUT= 'DIAG PAS OUT MVL H'
        SCAN REG BITS= 'F BUS 09 H' to 'F BUS 16 H',
                       'F BUS 27 H' to 'F BUS 34 H',
                       'F BUS P09-17 H',
                       'F BUS P27-35 H',
                       'F BUS 17 H',
                       'F BUS 35 H',
                       'AC BACKUP P09-17 H',
                       'AC BACKUP P28-35 H',
                       'AC BACKUP 17 H',
                       'AC BACKUP 35 H',
                       'AC BACKUP 09 H' to 'AC BACKUP 16 H',
                       'AC BACKUP 27 H' to 'AC BACKUP 34 H'
        SCAN OUTPUT= 'DIAG PAS OUT MVR H'
```

The Active scan path of MVL is composed of 32 bits from the MASTER AC BACKUP ADDRESS, L REG, and error control bits. The signals are labeled as follows:

```
        SCAN INPUT= 'DIAG ACT OUT ESE H'
        SCAN REG BITS= 'L AC BACKUP 0 H' to 'L AC BACKUP 7 H',
                       'L REG 00 H' to 'L REG 07 H',
                       'L REG 18 H' to 'L REG 25 H',
                       'L REG P00-08 H',
                       'L REG P18-26 H',
                       'L REG 08 H',
                       'L REG 26 H',
                       'MVL INPUT ERR H',
                       'MVL ALU ERR H',
                       'MVL OUTPUT ERR H'
                       'SPARE'
        SCAN OUTPUT= 'DIAG ACT OUT MVL H'
```

The MVR active scan path signals are labeled as follows:

```
        SCAN INPUT= 'DIAC ACT OUT MVL H'
        SCAN REG BITS= 'R AC BACKUP 0 H' to 'R AC BACKUP 7 H',
                       'L REG 09 H' to 'L REG 16 H',
                       'L REG 27 H' to 'L REG 34 H',
                       'L REG P09-17 H',
                       'L REG P27-35 H',
                       'MVR INPUT ERR H',
                       'MVR ALU ERR H',
                       'MVR OUTPUT ERR H',
                       'SPARE'
        SCAN OUTPUT= 'DIAC ACT OUT MVR H'
```

5.8 SHIFT/AC/REG PATH

The SHIFT/AC/REG path is contained on two identical modules, SHL and SHR. It consists of a 44ns loop through a SHIFT MATRIX, AC sets and register files, and muxing necessary to route the outputs of all of these into either the SHIFT MATRIX or the MOVE/ALU path.

5.9 SHIFT OPERATION

The SHIFT MATRIX is a 72 bit input, 36 bit output left shifter with a maximum shift value of 36 (100100). It's hi-order input comes from the X REG, loaded from the X BUS at CLK 2 gated by not L EARLY A. It's lo-order input comes from the Y REG, loaded from the Y BUS at CLK 2 gated by not L EARLY A. The shift output is available at the Z MUX 30ns after the clocking of the X and Y REGs, at CLK 0 gated by L EARLY B.

5.10 REGISTER FILE OPERATION

Two 16 word register files are present on SHL and SHR (SHL= bits 0-17 and SHR= bits 18-35). These are used as temporary storage files which are identical in content. The write address is determined by control bits 80-83; write timing is determined by L EARLY. A write will occur in the second or subsequent cycles if bit 59 is on. X REG RD is controlled by bits 60-63 and Y REG RD is controlled by bits 64-67.

5.11 AC/MASTER AC OPERATION

Two copies of the current AC set are kept on SHL and SHR. The X AC read port is addressed by bits 32-35 of the EACALC in the last cycle of an instruction, so that the X MUX may select between the data of an AC store in progress <if conflict compare> and the output of the X AC for an operation in the first cycle of the next instruction. d

The X1 MUX at the ALU will select between a memory operand <if EACALC is a memory location> or the X BUS <if EACALC is an AC>, or R or RF BUS if forwarding from a previous cycle's results. The Y AC read port is addressed by instruction bits 14-17 in the last cycle of an instruction, so that the Y MUX may select between the data of an AC store in progress <if conflict compare> and the output of the Y AC for an operation in the first cycle of the next instruction.

The X AC and Y AC addressing is determined in second and subsequent cycles by control bits 80-82.

The MASTER AC set is a 256 word array of 128 AC sets and 128 temporary registers. It may read or write every 22ns, as opposed to the current AC sets which may read and write every 22ns.

The addressing of the MASTER AC set is determined by control bits 80-83. Write enable timing is controlled by L EARLY and STORE CONTROL. The STEP CTRS microorder will cause the MASTER AC to be read followed by an increment of its 4 lo-order bits by 1. Two increments of one will occur during the 40ns period containing STEP CTRS. One microcycle later,under dly control, the copies of the selected AC set will be written with the data from the MASTER AC, followed by an update of the AC sets' address +1 <this occurs twice in the second microcycle>. The 16 words of each AC set may thus be loaded at a rate of 1 per 22ns, with the data travelling from the MASTER AC set through the Z MUX, X MUX, X1 MUX, R MUX, RF MUX, L MUX, L1 MUX, and the MQ MUX.

5.12 ERROR CONTROL

Parity is generated at the output of the SHIFT MATRIX. Four parity bits are carried with all 36 bit wide data paths of SHL and SHR. Parity is modified at the D and X MUXes to compensate for the 1 and 2 bit shifts. Parity checking occurs at the output of the Z MUX, X REG, and Y REG.

A parity error at the Z MUX will set R MAC OR Z ERR (for SHR) and L MAC OR Z ERR (for SHL) at the CLK 1 following the CLK 3 to Z MUX. This error condition will a) block the next clock to the Z MUX, and b) signal to error control logic at SCA to stop all system clocks starting with CLK 0 35ns later.

A parity error at X REG or Y REG will set R REG OR AC ERR (for SHR) and L REG OR AC ERR (for SHL) at the CLK 0 following the CLK 2 to X AND Y REG. This error condition will a) block the next clock to the X AND Y REGs, and b) signal to error control logic at SCA to stop all system clocks at CLK 0 20ns later.

System reset will reset L1 REG, X REG FILE, Y REG FILE, and Z MUX to zero with good parity. Error reset will reset the error latches 'MAC OR Z ERR' and 'REG OR AC ERR'.

5.13 DIAGNOSTIC INTERFACE

There is no passive scan path on SHL and SHR. The Active scan path OF SHL is composed of 80 bits, arranged in the following sequence:

```
        SCAN INPUT= 'DIAG ACT OUT SCA H'
        SCAN REG BITS= 'X REG 00 H' to 'X REG 08 H',
                       'X REG PARITY BYTE 0 H',
                       'X REG 09 H' to 'X REG 17 H',
                       'X REG PARITY BYTE 1 H',
                       'Y REG 00 H' to 'Y REG 08 H',
                       'Y REG PARITY BYTE 0 H',
                       'Y REG 09 H' to 'Y REG 17 H',
                       'Y REG PARITY BYTE 1 H',
                       'MAC REG BIT 00 H' to 'MAC REG BIT 08 H',
                       'MAC REG PAR BYTE 0 H',
                       'MAC REG BIT 09 H' to 'MAC REG BIT 17 H',
                       'MAC REG PAR BYTE 1 H',
                       'L2MQ REG PAR BYTE 0 H',
                       'L2MQ REG PAR BYTE 1 H',
                       'L2MQ REG BIT 17 H' to 'L2MQ REG BIT 00 H',
        SCAN OUTPUT= 'DIAG ACT OUT SHL'
```

The active scan path of SHR is composed of 80 bits, arranged in the following sequence:

```
        SCAN INPUT= 'DIAG ACT OUT SHL'
        SCAN REG BITS= 'X REG 18 H' to 'X REG 26 H',
                       'X REG PARITY BYTE 2 H',
                       'X REG 27 H' to 'X REG 35 H',
                       'X REG PARITY BYTE 3 H',
                       'Y REG 18 H' to 'Y REG 26 H',
                       'Y REG PARITY BYTE 2 H',
                       'Y REG 27 H' to 'Y REG 35 H',
                       'Y REG PARITY BYTE 3 H',
                       'MAC REG BIT 18 H' to 'MAC REG BIT 26 H',
                       'MAC REG PAR BYTE 2 H',
                       'MAC REG BIT 27 H' to 'MAC REG BIT 35 H',
                       'MAC REG PAR BYTE 3 H',
                       'L2MQ REG PAR BYTE 2 H',
                       'L2MQ REG PAR BYTE 3 H',
                       'L2MQ REG BIT 17 H' to 'L2MQ REG BIT 00 H',
        SCAN OUTPUT= 'DIAG ACT OUT SHR'
```

5.14 SHIFT CONTROL/METERS/ERROR CONTROL

The SCA module contains a 12 bit ALU path used for controlling the SHIFT MATRIX shift value and floating point exponents. Also on this module are the accounting meter, interval timer, PI and APR flags, pipeline stopping mechanism, and FRU code generation.

5.15 SCAD

The shift control ALU is a 12 bit binary add or subtract function receiving inputs from J and K MUXes. The output of the ALU is latched into the SCAD REG which provides the floating point exponent to the D MUX on the SHL module, and dispatches for branching under control of the next address switch. The output of the ALU is also latched into the SC REG, which may be held more than one cycle, and is used for branching and shift control purposes. The FE MUX receives inputs from either the ALU or the K MUX, and loads the FE REG at CLK2 gated by L EARLY A and FE FIELD bit 45 off. Both the SCAD REG and the SC REG are also clocked at CLK2 gated by L EARLY A.

5.16 1MS/10US CLOCK

<TO BE SUPPLIED>

5.17 INTERVAL COUNTER

<TO BE SUPPLIED>

5.18 ACCOUNTING METER

An accounting meter is kept at the SCA module. An 18 bit count is updated by one of two methods: a) at the last cycle of an instruction, the meter is updated by a weighted value from a lookup table addressed by the instruction code. This value will be greater for a long instruction than for a short instruction. b) during a long instruction,iterations of subroutine loops may issue an update to the accounting meter via an SP FUNCTION decode 5. The accounting meter may be turned off (no increments occur) or on (increments allowed to occur) by SP FUNCTION decodes 2 and 3. The accounting meter may be loaded or read via FLAG CONTROL decodes 15 or 16.

5.19 PIPELINE STOP MECHANISM

A two bit shift reg is serial loaded with the and of LAST CYCLE and L EARLY A each cycle. If the ERROR STOP latch is set at any CLK1 time, the contents of the shift reg (RETRY CODE) are frozen. A decode of the retry code determines how many times the PC has stepped since the error occurred:

What is relationship between this section + 4.2.6.2? Is this "EBOX AUTO RETRY"?

retry code	PC
0	N+1
1	N+2
2	N+2
3	N+3

To retry starting with instruction N, a trap is taken by the microcode, which causes it to branch on the retry code to a subroutine issuing a command to the IBOX to step SEL I back 1, 2, or 3 times, and to reload the PC from the PC BUFFER addressed by SEL I. Once the ERROR STOP latch has been set at CLK 1, the system clocks are stopped at the CLK CTRL module starting with the next CLK 0.

5.20 FRU CODE GENERATION

<TO BE SUPPLIED>

5.21 SCAD ERROR CONTROL

<TO BE SUPPLIED>

5.22 DIAGNOSTIC INTERFACE

<TO BE SUPPLIED>

5.23 CONTROL STORAGE

fast slow

EBCX control storage is located on the ESE and CRA modules. It contains 102 bits of fast and slow storage, plus the necessary output decodes and branching controls to operate the EBCX data paths and initiate operations at IBOX, MEOX, and FPA.

5.23.1 RAM ARRAYS

4Kx1

Two RAM arrays are present in EBCX control storage: a slow 4K deep array capable of cycling in 44ns, and a fast 512 word array which 256x4 develops a short microword for control of the first cycle of each instruction. Each array is divided into 3 sections, early, intermediate, and late. Early control bits come from a section in the array that is directly powered by either the instruction code <fast RAMs in first cycle> or by a mux which selects instruction code or NA BUS <slow RAMs for second and subsequent cycles, respectively>. All early control bits are clocked at CLK 0. Intermediate control bits come from a section of the array that is skewed, at the address inputs and at the latched data outputs, 5 to 10 ns later than the early section. All intermediate control bits are clocked at either CLK 1 or CLK 2, depending upon the timing of the logic to be controlled. Late control bits come from a section which is further skewed by 5 to 10ns. All late control bits are clocked at either CLK 3 or CLK 0, depending the timing of the logic to be controlled.

The early, intermediate, and late control bits are deskewed into the CTL latches, which serve the dual purpose of holding that module's portion of the microword for parity checking, and providing a scan path input to the arrays for initial control storage load.

Two additional bits of fast and slow RAMs are located on the ESE module, 102 and 103. CTL bit 102 may be loaded independently by the console and is used as a MARK bit for diagnostic or troubleshooting synchronization purposes. CTL bit 103 is a spare.

All 104 bits are addressable by instruction code for second cycle operation and next address path for third and subsequent cycle operation. Only 34 bits of fast RAM are used for first cycle operation, primarily to control the MOVE/ALU path and the store and flag setting controls for one cycle instructions. The control bits which do not have fast RAMs allocated default to zero in the first cycle. Control bits usable in the first cycle are:

35-36	SP FUNCTION
48-49	X1
50-51	Y1
52-55	ALU
56-58	CARRY
59	OP1
60-63	SW
64-67	R
68-69	COMP
70	LAST CYCLE
71-73	STORE CONTROL
74-75	FLAG CONTROL
78	PARITY
79	L
82	WR MUX
102	MARK

5.24 NEXT ADDRESS GENERATION

The NA BUS bits 0-11 are generated at the CRA module. The value of this bus is determined by the control bits of the Next Address Field and the NA SWITCH. The NA BUS addresses the slow array for control word readouts starting with the third cycle of an instruction.

5.25 FLAGS

The 2080 EBOX controls 4 sets of flags: PC, PI, APR, and STATE. The eleven PC flags are located on the CRA module, and may be loaded from bits 0-12 of the F BUS. They are read into the next # FIELD 0-12 when RD FLAGS micro-order is issued.

PC FORMAT

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+
!    !    !    !    !    !    !USER!    !    !    !    !    ! NO !
! OV !CRY0!CRY1!FOV !FPD !USER! IO !    !    !TRP2!TRP1!FOV !DIV !
!    !    !    !    !    !    !PCU !    !    !    !    !    !    !
+----+----+----+----+----+----+----+----+----+----+----+----+----+
  0    1    2    3    4    5    6    7    8    9   10   11   12
```

The PI and APR flags are located on the SCA module.

The STATE REG is an 8 bit register of independently settable and resettable flags. STATE REG bits 0-7 are set by # FIELD 0-7, respectively, and reset by # FIELD 8-15, respectively, when the LD STATE micro-order is issued. The STATE REG bits are used to remember the status of microcode subroutines as portions of an instruction are completed. They may be read into # FIELD 0-7 if the RD FLAGS micro-order is isued. STATE REG bits are primarily used as dispatches into the NA BUS under control of the NA SWITCH.

Useful for us to put in user file?

5.26 ERROR CONTROL

Correct parity is stored into the ESE and CRA RAM arrays upon control storage loading from the console. As the microwords are read out, the portion of the word contained on each module is checked for odd parity (the early and intermediate CTL bits are relatched into registers clocked at the same time as the late CTL bits, so that the entire portion may be checked). If a parity check occurs, the CTL bits of that module are held, along with the NA DLY latch, which represents the address of the failing word (if the failure occurred in third or subsequent cycle). If a parity error occurs in the first or second cycle, the console may read the instruction code of the failing instruction by stepping back the SEL I in the IBOX. The 34 fast RAMs are parity checked along with zeros (for unused positions) in the first cycle. A parity error detect will set ESE MICROCODE ERR or CRA MICROCODE ERR latches at CLK 2 following the late CLK 0 readout of a microcycle. These error latches will signal the error control logic at SCA to stop IBOX, EBOX, and FPA system clocks starting with CLK 0, 30ns later.

System reset will reset all PC flags and STATE REG bits to 0. Error reset will reset ESE and CRA MICROCODE ERR latches.

5.27 DIAGNOSTIC INTERFACE

The passive scan path of ESE is composed of 3 bits arranged in the following sequence:

```
        SCAN INPUT= 'DIAG PAS OUT CRA H'
        SCAN REG BITS= 'F WR CTL H',
                       'S WR CTL H',
                       'WR MARK BIT H'
        SCAN OUTPUT= 'DIAG PAS OUT ESE H'
```

'F WR CTL' is gated by 'DIAG CMD ENABLE' and enables a write to fast RAMs to occur with the cycling of system CLKS 0, 1, 2, and 3 once. 'S WR CTL' is also gated by 'DIAG CMD ENABLE' and enables a write to slow RAMs to occur with the cycling of system CLKS 0, 1, 2, and 3 twice. 'WR MARK BIT' when gated by "DIAG CMD ENABLE" will cause CTL bit 102 to be written into fast or slow RAMs as governed by 'F WR CTL H' and 'S WR CTL H'.

The active scan path of ESE is composed of 56 bits arranged in the following sequence:

```
        SCAN INPUT= 'DIAG ACT OUT CRA H'
        SCAN REG BITS= 'CTL BIT 48 IN H' to 'CTL BIT 78 IN H',
                       'CTL BIT 103 IN H',
                       'CTL BIT 79 IN H' to 'CTL BIT 102 H'
        SCAN OUTPUT= 'DIAG ACT OUT ESE H'
```

Does software do "system reset" and "error reset"?

major sections:
6.2 memory addr path
6.10 memory operand path
6.15 IBOX control logic

CHAPTER 6

IBOX

Listed in Nat's plan Index?

6.1 GENERAL

The 2080 IBOX is contained on 3 modules: IDR, IDL, and TAC. The primary purpose of the IBOX is to prefetch from the MBOX operands and other information necessary for the execution of instructions by the EBOX, thus eliminating much of the delay penalty usually associated with cache and memory accesses. It serves as the interface between the EBOX and the MBOX for all memory reads and writes. Although the 2080 IBOX prefetches instructions in a pipeline fashion, it is a microcode controlled unit, and as such provides much flexibility in arriving at algorithms which provide maximum throughput with minimum hardware cost. The TAC system unique to the 2080 allows multiple, simultaneous access of the MBOX by the IBOX without the part count or conflict overhead usually associated with a hard-wired pipeline structure.

6.2 MEMORY ADDRESS PATH

The memory address path is located on the IDL and IDR modules. It's output is the VMA (virtual memory address) bus, a 30 bit bus which is used for any of four types of accesses: a) program counter to read memory, b) instruction stream prefetch to read memory, c) effective address calculation to read or write memory, and d) direct address, address+1, or address+2 from EBOX or INDEX AC to read or write memory.

6.3 INDEX AC

A copy of the current AC set is maintained on IDL and IDR, and is updated from the L BUS each time the EBOX stores into its current AC set or loads a new set from the MASTER AC block. This AC set is used primarily for EACALCs, where the index reg is added to the Y portion of the instruction (bits 18-35), according to the algorithm described on page 1-22 of the hardware reference manual. It is also used for the memory address determination of the stack instructions PUSH, POP, PUSHJ, and POPJ.

6.4 EA ALU

The EA ALU produces a 30 bit binary add, add+1, or add+2 in 22ns. During instruction stream prefetching, the EA ALU is used as the incrementer of the stream addresses, with the value 1 forced into the E2 MUX and the output of the stream buffer (via VMA) routed in to the E1 MUX. In this mode, with a forced carry in, the EA ALU adds 2 to the instruction address to arrive at the next doubleword instruction address boundary. Instructions are fetched 2 at a time from the M BOX from a single VMA address. The results of the stream update are stored in the I STR buffer in addition to being sent to the M BOX over the VMA bus. I stream updates by the EA ALU affect only bits 18-35 of the VMA. Bits 6-17 come from the PC. What is the effect of this? In indirect references?

When an instruction arrives at the IBOX, a request is made to the IPUT microcode to initiate an EACALC. Provided that the IBOX is not busy with a higher priority request (such as a command from EBOX), it will initiate an EACALC as a function of the INDEX AC value and the Y portion of the instruction. The results of the EACALC may be stored in the EA BUFFER or the PC BUFFER, or may be sent to the VMA bus for a memory read, to obtain an operand, or a test for page fail of an address representing a write destination.

The EA ALU is also used to pass an address directly (or +1 or +2) from the INDEX AC or from the EBOX L BUS to the VMA bus. Following a read PC command from the EBOX the EA ALU is used as a path for the EBOX to retrieve the value of the PC (via the VMA bus).

6.5 I STREAM PREFETCH

16? see p.6-12 (two instr. per slot)

When the PC is loaded after a preset of the IBOX, all 8 buffer slots are empty. The PC will be gated onto the VMA bus to fetch the first instruction. This starting PC value will be tested for a lo-order bit. If on, the EA ALU will increment the VMA +1 to get the next doubleword in the instruction stream. If off, the EA ALU will increment the VMA +2 to get the next doubleword. Each instruction address fetched is stored in the I STR BUFFER under a stream address A, B, or C. If no jumps are detected in the instructions coming back, a single stream will be updated, and its most recently addressed location will reside in it's stream buffer word. If a jump out of the stream is detected, a second stream is created, which starts from the the jump target address, is updated and loaded into that stream's word in the STR BUFFER.

IPUT (instruction prefetching unit) microcode controls the prefetching of up to 3 separate instruction streams. This is described in greater detail in section 5.3.

? not after ?

6.6 PC

The program counter is a 30 bit register representing the address of the instruction prior to the instruction in execution at the EBOX. It is updated at CLK 0 at a maximum rate of once per 22ns. The PC may be loaded either from the PC buffer, which loads the PC when a jump occurs, or the PC ALU, which updates the lo-order 18 bits of the PC by +1 or +2. The PC buffer is loaded when an EACALC of a JUMP instruction is present at the EA MUX, and when the PC is updated. IPUT will hold the EA MUX until the EBOX is in a cycle where LAST CYCLE is not in progress, since LAST CYCLE will, under direct hardware control (Last CYCLE and no interrupt present), select the PROGRAM COUNTER into the PC BUFFER. The PC BUFFER therefore contains a trace of program execution flow, and may be used for hardware recovery purposes.

6.7 CONFLICT COMPARE

why only 9 bits?

The addresses of both instructions and operands are subject to possible writes by the EBOX. Therefore, indexes are kept to determine whether the 9 lo-order bits of the VMA of a write compare with any instruction addresses or operand addresses of instructions or operands in the I buffers. If a compare occurs against the index, a request is made to IPUT to refetch the contents of the conflicting address. Another comparator checks the AC stores against an INDEX REG index, containing INDEX REG values of all I BUFFER entries. If a conflict occurs here, a request is made to IPUT to perform an EACALC with the new INDEX AC value, and refetch the operand.

why 18 bits here, 9 above?

IBIX performs not only a lo-order 9 bit compare against the instruction address of memory writes, but also an 18 bit lo-order compare against the instruction address of EACALCs resulting from jump instructions. This compare is used by IPUT in determining whether or not a new stream should be created (if a jump target address resides in the I BUFFERS, no new stream is created).

6.8 ERROR CONTROL

Parity is checked at both inputs of the EA ALU and PC ALU. Parity is generated at the output of the EA MUX and PC ALU. One bit parity per 9 data bits is carried for bytes 1, 2, and 3 of the memory address path; one parity bit is carried for bits 6-8 (address byte 0). If an error has been detected at the L BUS on the MVL or MVR modules, the write into the INDEX AC will be blocked.

6.9 DIAGNOSTIC INTERFACE

A passive scan register captures the new PC each time it is updated. This PC scan reg may be held for console readout of realtime PC values.

6.10 MEMORY OPERAND PATH

The IBOX prefetches/precalculates the operands associated with all instructions assigned to the IBOX by IPUT. In the case of immediate operands, the EA BUFFER will be loaded with the effective address calculation,and a selection is made in the first cycle of EBOX execution at the OP2 MUX of the immediate value, whether it be directly from the EA MUX, or from the EA BUFFER. In the case of memory operands, IPUT will, once EACALC has been performed, send the EACALC to the MBOX via the VMA BUS, along with a tag identifying the OP BUFFER slot into which returning data will be written. If the EBOX is in the wait loop (waiting for operand valid bits) when the operand arrives at the IBOX from the MBOX, OP2 MUX selects the MD MUX. If the operand valid bit is on (in OP BUFFER), the OP2 MUX selects the OP BUFFER.

6.11 OP BUFFER

The OP BUFFER is a 16 word register file which ~~is~~ represents the contents of the locations addressed by EACALCs of up to 16 instructions. The OP BUFFER is read 22ns prior to the start of execution by the EBOX and is gated on to the OP2 bus if the instruction does not require an immediate operand (OP2 is the 2nd operand of an instruction, whereas OP1, the word addressed by the AC field , is the 1st operand of an instruction).

The OP BUFFER may be loaded from the MD BUS under control of a returning tag whose hi-order 3 bits identify an operand fetch or refetch. It is read by SEL I , a 4 bit value which identifies the buffer position of the next instruction to be executed.

6.12 EA BUFFER & TAG BUFFER

The EA BUFFER is a 16 word register file which is loaded with the EACALCs (or in the case of STACK instructions the address AC or AC+1) of up to 16 instructions. It is written under control of a tag out of the TAG BUFFER. The TAG BUFFER is written with the lo-order 4 bits of a returning tag if the 3 hi-order bits of the tag identify an instruction fetch or refetch (the TAG BUFFER is written by a counter which is then incremented+1, and read by another counter which also incremented +1; counters not equal cause a request to IPUT microcode to initiate an EACALC).

The EA BUFFER is read by SEL I, 22ns prior to actual execution of the instruction in the EBOX. It's output will be gated to the OP2 BUS if the instruction requires an immediate operand.

6.13 ERROR CONTROL

The OP2 BUS is parity checked as it leaves IDL and IDR. Parity modification occurs at the MD MUX to correct bad parity coming in from the MBOX. If parity is bad on the incoming M DATA, a flag is stored in the OP BUFFER which will identify the error type to EBOX error control at the SCA module when and if the instruction is called up for execution by SEL I.

Another flag is stored in the OP BUFFER signifying a page failure. This flag, when the OP BUFFER is read to be gated onto the OP2 BUS will cause a page fail trap to occur (if no interrupts or other errors exist at last cycle, or if no other errors exist, not at last cycle).

6.14 DIAGNOSTIC INTERFACE

The OP2 BUS has a passive scan register which is clocked every cycle, and is held if an error is detected on the bus. The L3 REC, which may be selected into the MD MUX, the INDEX AC, or the E2 MUX, is in the active scan loop.

6.15 IBOX CONTROL LOGIC

IBOX control logic consists primarily of IPUT microcode and TAG controls. IPUT is a 256 word, 32 bit wide control storage for the IBOX data paths. The TAG system is the method by which all instructions, operands, and related information are tracked.The format of the TAG is shown below:

```
+-----------------+----------------------+
!   TAG SOURCE    !    BUFFER POSITION   !
+-----------------+----------------------+
       0-2                  3-6
```

Tag source is decoded as follows:

decode	source
0	Instruction refetch
1	OP3 (FPA) refetch
2	OP2 refetch
3	Operand initial fetch
4	Instruction fetch stream A
5	Instruction fetch stream B
6	Instruction fetch stream C
7	Instruction fetch - initial fill

To initiate the IBOX into a prefetching sequence for instructions starting at a given PC address, the EBOX must first preset the IBOX by issuing FLAG CONTROL field:10 and # FIELD 14-17 = 0. IPUT will trap to an address which will reset all buffer valid bits, the program counter, PC BUFFER, EA BUFFER, OP BUFFER, TAG BUFFER, and initial fill counter. It will then enter a wait loop, enabling only an EBOX command to trap out of it. The EBOX may then use the IBOX for whatever it wants to do, such as reading memory and setting the PC. When the EBOX issues the START command via its FLAG CONTROL field:10 and # FIELD 14-17 = 15, the IBOX will begin prefetching instructions starting with the PC address (the EBOX must wait until it receives MBOX NOT BUSY before issuing the START command, to insure that all pending IBOX prefetches have completed).

Following the START, IPUT will issue I/E REQ TO M, gate the PC onto the VMA bus, and the initial fill counter onto the TAG bus. It will then update the fill counter +1, update the VMA +1 or +2 (and store this STREAM address into the STR BUFFER),issue another I/E REQ TO M, gate the EA MUX onto the VMA BUS, and the updated fill counter onto the TAG BUS. Each time IPUT repeats this sequence, it will be asking MBOX for two instructions. After requesting the first 6 instructions starting with the PC, IPUT will wait for the first two instructions to return (most probably a cache miss), before proceeding with the buffer fill operation. A trap request will be made to IPUT when an incoming TAG arrives, an indication that the TAG is in the process of storing MD BUS data into the I BUFFERS. IPUT will stack its previous address (I WAIT LOOP) and proceed with the EACALC for the first instruction. It will then issue an I/E REQ TO M, and gate the EACALC onto the VMA BUS. A table lookup addressed by the instruction code may indicate that the instruction EACALC is the 2nd operand, and as such would block the memory request. The memory request may also be blocked by an indication from a compare of EACALC results and global bit that the 2nd operand is an AC. The EACALC on the VMA is accompanied by a tag whose source is identified as being an initial fetch of an operand (bits 0-2) into an OP BUFFER position (bits 3-6).

IPUT will process all EACALC requests, and, with no other trap requests pending, return to the previous routine where it was assigning and prefetching instructions. At this time it may branch to a subroutine which begins a new stream of I fetches, if a jump out of the current stream has been detected during the EACALC of received instructions.

6.16 IPUT

IPUT consists of a 256 word by 32 bit array along with supporting branch, trap, and decode logic. The fields of the IPUT microword are defined as follows:

BITS	NAME
0-4	INA
5-8	BR
9-10	VMA
11	ISTR
12	OP2
13-14	WR
15-16	XR
17-18	E1
19-20	E2
21	EA CARRY
22-25	SP1
26-27	STR
28-30	SRCE
31	P 0-31

6.16.1 INA <BITS 0-4>

The INA field determines the next address in conjunction with the BR field, provided that no trap requests are active. If the BR field decode is 0, INA bit 0 will control next address as follows:

bit 0 on	current IAD 0-3 to next IAD 0-3 INA field 1-4 to next IAD 4-7
bit 0 off	INA field 1-4 to next IAD 0-3 0001 to next IAD 4-7

6.16.2 BR <BITS 5-8>

The BR field determines IAD 5-7 if bit 0 (INA) is on.

decode	function
0	EBOX 14-17 to next IAD 4-7
1	return 4-7 to next IAD 4-7
2	0010 to next IAD 4-7
3	current INA 0-3 to next IAD 4-7
4	spare
5	0 to next IAD 4, early decode 0-2 to next IAD 5-7
6	00 to next IAD 4-5, HI IFET BR to next IAD 6, LO IFET BR to next IAD 7
7	01 to next IAD 4-5, HI IFET BR to next IAD 6, MED IFET BR to next IAD 7
10	1 to next IAD 4, HI IFET BR to next IAD 5, MED IFET BR to next IAD 6, LO IFET BR to next IAD 7
11	prev IAD 4-5 to next IAD 4-5, M Grant to next IAD 6, SPOP to next IAD 7
12	prev IAD 4-6 to next IAD 4-6, M Grant to next IAD 7
13	spare
14	1 to next IAD 4, stream A, B, C full latches to next IAD 5-7
15	indirect bit to next IAD 5 early decode instr type to next IAD 6-7 as follows: 00 normal EACALC 01 E2 to EA MUX 10 E2+1 to EA MUX 11 instr= JUMP
16-17	spare

.HL2 VMA <BITS 9-10>
The VMA field controls the VMA MUX on the IDL and IDR modules.
The field decodes are as follows:
.TP 8
.LIT

decode	function
0	select PC 6-17 to VMA 6-17, select STR MUX 18-35 to VMA 18-35

1	select EA MUX 6-35 to VMA 6-35
2	select PC 6-17 to VMA 6-17, select PC MUX 18-35 to VMA 18-35
3	select EA BUFFER 6-35 to VMA 6-35

6.16.3 ISTR <BIT 11>

The ISTR bit determines the selection of the ISTR MUX as follows:

bit 11 off	select ISTR BUFFER 18-35 to ISTR 18-35
bit 11 on	select IBIX 18-34 to ISTR 18-34, zero to ISTR 35

6.16.4 OP2 <BIT 12>

The OP2 bit determines the selection of the OP2 MUX, provided that 1st CYCLE control (last cycle delayed) is not on, as follows:

bit 12 off	select OP BUFFER or MD MUX 0-35 to OP2 0-35
bit 12 on	select zeros to OP2 0-5, select EA BUFFER or EA MUX 6-35 to OP2 6-35

6.16.5 WR <BITS 13-14>

The WR field controls write enables into the I BUFFERS according to the following decodes:

decode	function
0	no op
1	write to STR BUFFER
2	write to PC BUFFER if JUMP instruction, to to EA BUFFER if not JUMP instruction
3	write to EA BUFFER

6.16.6 XR <BITS 15-16>

The XR field controls the read address port of the INDEX AC via selection at the XR MUX, according to the following decodes:

decode	function
0	select XR buffer 0-3 to XR MUX 0-3 if not I TAG CTRS EQUAL
1	select XR BUFFER 0-3 addressed by MD MUX 14-17 if I TAG CTRS EQUAL
2	select EBOX WR MUX 0-3 to XR MUX 0-3
3	spare

6.16.7 E1 <BITS 17-18>

The E1 field selects the E1 MUX input to the EA ALU according to the following decodes:

decode	function
0	VMA 6-35 to E1 6-35
1	sign extend/global control to E1 6-17, Y BUFFER 18-35 to E1 18-35
2	zeros to E1 6-35
3	force global: zeros from VMA, select VMA to E1 6-17 if not bit 18, force ones to E1 6-17 if bit 18. Select Y BUFFER to E1 18-35

6.16.8 E2 <BITS 19-20>

The E2 field determines the selection of the E2 MUX as follows:

decode	function
0	L3 6-35 to E2 6-35
1	XR 6-35 to E2 6-35
2	zeros to E2 6-35
3	zeros to E2 6-34, not VMA 35 to E2 35

6.16.9 EA CARRY <BIT 21>

The EA CARRY bit when off causes a carry into bit 35 of the EA ALU. When on, no carry into 35 occurs.

6.16.10 SP1 <BITS 22-25>

If bit 22 is off, 23-25 have the following significance:

decode	function
0	no-op
1	preset IBOX
2	reset stream A full latch
3	reset stream B full latch
4	reset stream C full latch
5	trap taken (to EBOX)
6	return from trap
7	select L3 0-35 to MD 0-35

Bit 22 on causes MBOX REQUEST. With bit 22 on. 23-25 have the following significance:

Bit 23 on	EBOX held M CTRL to MBOX
Bit 23 off	IBOX M CTRL to MBOX
Bits 24-25	function
0	2 word access if VMA even, TAC bit 6= 0 for first word, TAC bit 6= 1 for 2nd word
1	2 word access if VMA even, second word= OP3
2	1 word access OP2
3	1 word access OP3

6.16.11 STR <BITS 26-27>

The STR field identifies the stream to be updated in the LRU BUFFERS or in the STR BUFFER, according to the following decodes:

decode	function
0	no LRU update
1	stream A update
2	stream B update
3	stream C update

6.16.12 SRCE <BITS 28-30>

The SRCE field determines the selection of the hi-order 3 bits of the TAC, identifying the source VMA and destination buffers of the word being prefetched.

decode	function
0	Instruction refetch. Source of VMA is the IBIX slot with the highest priority compare; destination of returning instruction is XR BUFFER, Y BUFFER, AC BUFFERS, IC BUFFER, and I FLAG BUFFER, addressed by returning TAG 3-6.
1	OP3 refetch. Source of VMA is the EA BUFFER addressed by OP3 compare encode; destination of returning data is OP3R and FLAG BUFFER in the FPA addressed by returning TAG 3-6.

2 OP2 refetch. Source of VMA is the EA BUFFER addressed by OP2 compare encode. Tag source is the OP2 compare encode; destination of returning data is the OP BUFFER, FLAG BUFFER, and OP3L in the FPA, addressed by returning TAG 3-6.

3 Operand initial fetch. Source of the VMA is the EA MUX. TAG source is TAG BUFFER (delayed), addressed by TAG BUFFER LRU. Destination of returning data is the OP BUFFER, FLAG BUFFER, and OP3L in the FPA, addressed by returning TAG 3-6.

4 Instruction fetch stream A. Source of the VMA is the EA MUX. TAG source is the stream A LRU. Destination of returning instruction is XR BUFFER, Y BUFFER, AC BUFFERS, IC BUFFER, and FLAG BUFFER, addressed by returning TAG 3-6.

5 Instruction fetch stream B. Source of the VMA is the EA MUX. TAG source is the stream B LRU. Destination of returning instruction is XR BUFFER, Y BUFFER, AC BUFFERS, IC BUFFER, and FLAG BUFFER, addressed by returning TAG 3-6.

6 Instruction fetch stream C. Source of the VMA is the EA MUX. TAG source is the stream C LRU. Destination of returning instruction is XR BUFFER, Y BUFFER, AC BUFFERS, IC BUFFER, and FLAG BUFFER, addressed by returning TAG 3-6.

7 Instruction fetch - initial fill. Source of the VMA is the PC (if first fetch after PC load) or EAMUX (from a stream update). Source of TAG is the initial fill counter. Destination of returning instruction is XR BUFFER, Y BUFFER, AC BUFFERS, IC BUFFER, and FLAG BUFFER, addressed by returning TAG 3-6.

6.16.13 P 0-31 <BIT 31>

Bit 31 causes odd parity to occur on bits 0-31 of the IPUT microword.

6.17 STREAM LRU

Three stream LRU buffers are located on the TAG module. Each consists of a 16X4 register file, a read port counter, and a write port counter. The counters are initialized to zero at IBOX preset.

The I BUFFERS are divided into 8 slots, each representing two instructions which are adjacent to each other in virtual memory and aligned to a doubleword boundary. Following preset, the 1st 8 instruction prefetches have their TAGs assigned by the initial fill counter, a 0 to 7 binary counter. When the initial fill counter reaches a count of 7, the A, B, and C full latches are set. From this point on, TAG assignment for instruction prefetches is under

what are implications on software for perfor-mance what is IBIX?

control of the A, B, and C LRU BUFFERS.

During initial fill, the STR field of IPUT will cause an update of the write port counter for the selected stream when a TAC with SRCE=7 is sent out. When control is turned over to the LRU BUFFERS, the read port counter for each active stream points to the least recently filled SLOT position. IPUT will stop prefetching once all SLOTS have been filled and a JUMP compare to a location in IBIX has been detected. it may continue prefetching into the least recently filled location if no JUMP within the IBIX has occurred.IPUT selects the stream to be prefetched based on JUMP detects of returning instructions and jump successful/not successful of currently executing instructions. JUMP detects of returning instructions will cause IPUT to create a new stream if the JUMP is out of IBIX. Execution of a JUMP instruction will cause cancellation of the stream not taken and reset of the respective stream full latch.

The 8 slots are divided among the three LRU systems. For example, stream A may be allocated 3 slots, stream B (due to detect of a JUMP out) may be allocated 2 slots, and stream C (due to detect of a 2nd JUMP out) may be allocated the remaining 3 slots. Slot assignment would look like this:

```
            +--------------+---------+---------------+
            !     STR A    !  STR B  !     STR C     !
            +--------------+---------+---------------+
TAC 3-5
DECODE=     0    1    2    3    4    5    6    7
```

Now let's suppose a JUMP has been executed, in which the execution went from stream A to stream B. The stream A full latch is reset, thus opening slots 0, 1, and 2. IPUT branches to a subroutine which allows it to fill stream B and stream C. Stream B's LRU read counter=0 and LRU write counter=2. As IPUT goes to fetch into stream B it increments stream B's write counter to 3 and assigns to it a TAC from the STR A LRU, and increments stream A's read counter. Slot assignment will now look like this:

```
            +----+---------+---------+---------------+
            !STRB!  STR A  !  STR B  !     STR C     !
            +----+---------+---------+---------------+
              0    1    2    3    4    5    6    7
```

The difference between a stream's read and write counters is equal to the number of slots assigned to it. The sum of the differences will always add up to the value of the initial fill counter. In the manner described above, 3 LRU systems may share a common buffer space, with one displacing another based upon the flow of program execution.

6.18 AC ADDRESS, IC BUFFERS

A returning 1 TAG will load the various I BUFFERS with the segments of the instruction. Among these are the AC ADDRESS BUFFERS, 2 pairs of 16X4 register files which are loaded with bits 9-12 and 32-35 of the instruction. Prior to the first cycle of execution in the EBOX, the AC BUFFER pairs will be read out, with one of a pair addressed by the TAG identifying the buffer position of the next sequential instruction, and the other of a pair addressed by the TAG identifying the buffer position of the skip or jump target. Skip/jump successful from the instruction in execution will select either the next sequential or jump/skip AC address to be sent to the EBOX.

The IC BUFFER is a pair of 16X9 register files, loaded by a returning I TAG by the same controls that load the AC BUFFERS. The duplicate sets contain bits 0-8 of the instruction, and are read and selected in the same way that the AC BUFFERS are read and selected. The selected IC is sent to the EBOX at the CLK 1 prior to the first cycle of execution, and directly address fast ISET and ESET RAMS to set up first cycle controls, and also directly address the slower control RAM array for second cycle control. Another copy of the IC is sent to the SCA module where the accounting meter may be updated according to the instruction type.

6.19 ISET

ISET is an 8 bit lookup table which provides first cycle controls to IBOX and EBOX. ISET is similar to ESET in that it is addressed by the instruction code. Since it is located on the same module as the IC BUFFERS, it is able to provide control earlier than ESET (CLK 3 of LAST CYCLE). ISET control fields are defined as follows:

bits	field	timing
0-2	skip/jump	late (CLK3 1st cycle)
3	X1 disable	early (CLK3 last cycle)
4	Y1 disable	early (CLK3 last cycle)
5-6	OP MUX ctrl	early (CLK3 last cycle)
7	P	

6.19.1 SKIP/JUMP <BITS 0-2>

The SKIP/JUMP field controls skip or jump determination from the compare results of the AND function at the MVL and MVR modules according to the following decodes:

decode	function
0	no skip/jump
1	skip/jump if less
2	skip/jump if equal
3	skip/jump if less than or equal
4	skip/jump
5	skip/jump if greater than or equal
6	skip/jump if not equal
7	skip/jump if greater

6.19.2 X1 DISABLE <BIT 3>

The X1 DISABLE bit, when on, forces zeros to the X1 MUX at the MVL and MVR modules during the first cycle. When the X1 DISABLE bit is off or EBOX is not in first cycle, X1 MUX falls under control of the X1 field.

6.19.3 Y1 DISABLE <BIT 4>

The Y1 DISABLE bit, when on, forces zeros to the Y1 MUX at the MVL and MVR modules during the first cycle. When the Y1 DISABLE bit is off or EBOX is not in first cycle, Y1 MUX falls under control of the Y1 field.

6.19.4 OP MUX <BITS 5-6>

The OP MUX controls the OP2 MUX of IDL and IDR to gate the correct operand onto the OP2 BUS in the first cycle. It overrides the OP2 field of IPUT whenever the LAST CYCLE micro-order is issued by the EBOX.

bit 5 off	OP2 MUX under bit 6 control
bit 5 on	zeros to bits 0-17 of OP2 MUX
bit 6 off	select OP BUFFER or MD MUX to OP2
bit 6 on	select EA BUFFER or EA MUX to OP2

6.19.5 P <BIT 7>

The P bit of ISET causes odd parity to occur for bits 0-7 of the ISET control word.

6.20 INSTRUCTION VALID BITS

As described earlier, multiple prefetches from the MBOX may be in progress at any one time. Once IPUT has sent a TAG out with a VMA address, it relinquishes control over that fetch. The returning TAG itself controls the loading of I BUFFERS with incoming memory data. It is possible that TAGs will return in a different sequence than they were originated, if a cache miss occurs, for prefetches may also be made from cache while another fetch from main memory is going on. The IBOX valid bit algorithm insures that the operands that the EBOX receives for it's first cycle are correct.

Three count buffers are maintained which contain a 2 bit count of the total number of fetches pending for each I BUFFER position. Counts are kept of instruction fetches, initial memory operand fetches (OP2), and additional memory operand fetches (OP3). A count of 1 indicates that a single prefetch is pending for the buffer position into which it is stored. A count of more than 1 indicates that one or more refetch requests due to write conflicts have been initiated against that buffer position. Counts are incremented when TAGs are sent out, are decremented when TAGs are received. At most, two TAGs may be outstanding for a buffer position, given that the maximum rate at which TAGs may be sent is 1 per 44ns, and cache access = 88ns. A count of 3 will set an error latch.

If a cache miss occurs against a buffer position, refetch requests for that position are blocked until data returns from main memory accompanied by that position's TAG. However, fetches for all other positions are still allowed.
The instruction is valid if:

count=0 for instruction

and

count=0 for OP2 if OP2 needed

and

count=0 for OP3 if OP3 needed

and

no instr., OP2, or OP3 refetch request latch set for SEL I

and

SEL I assigned a valid stream

6.21 SELECTED I CONTROLS

The SEL I is a 4 bit value that addresses the CP2, EA. and PC buffers during the last cycle of one instruction in order to have operands available to the EBOX, an address available to the MBOX, and a jump target address available to the PC, for the first cycle of the next instruction. It comes from the SI MUX, whose inputs are the NEXT I BUFFER, the JUMP I BUFFER, and the SI REG.

The NEXT I BUFFER is a 16X4 reg file read by SEL I to provide the ~~the~~ position in the I BUFFERS containing the next sequential instruction to be executed. The JUMP I BUFFER is also a 16X4 reg file read by SEL I to provide the the possible jump or skip target position in the I BUFFERS which may contain the next instruction if a skip or jump is taken.

As each I TAG is sent to the MBOX to prefetch an instruction, it is fed into one of three pipes, depending upon the STR field decode. Each pipe, 4 bits wide and 3 stages long, appears as a dynamic window of the I BUFFER positions of three sequential instructions for its stream. Thus, the first stage represents a given instruction, the second stage represents the next sequential instruction, and the third stage represents the following, or skip target, instruction. By addressing the NEXT I and JUMP I BUFFERS with the first stage, the second stage can be written (as data in) into the NEXT I BUFFER, and the third stage can be written into the JUMP I BUFFER (initially as a skip target). Later, when the EACALC is performed for any JUMP instruction, the encode from an IBIX compare is written into the JUMP I BUFFER addressed by the TAG BUFFER delayed (the buffer position of the instruction performing the EACALC).

In the manner described above, JRSTF (jump always) is executed by the IBOX as part of its SEL I process, and does not appear in the instruction execution flow at the EBOX, unless two JRSTFs in a row occur. Does it go into pc buffer ?

Two valid bits are maintained, one each for the JUMP I and NEXT I BUFFERS, to indicate those positions which IPUT was not able to send a TAG out to fetch the target instruction. This may occur if all slots are full and there is a jump within the I BUFFERs, or if a VMA request has not been granted by the MBOX. If the EBOX comes to the point in execution where it needs an instruction that has not been assigned a TAG (e.g.,NEXT I is not valid, and new SEL I= NEXT I), IPUT takes a high priority trap and begins assigning new entries by LRU, starting with the instruction required by the previous SEL I.

6.22 INDEX REG CONTROLS

The INDEX REG is a 16X4 content addressable memory which contains bits 14-17 of all instructions prefetched into the IBOX. A compare of it's contents occurs with every write to the current AC set; an INDEX REG compare will initiate a new EACALC for the instruction, followed by an operand fetch, if necessary. As in instruction and operand conflict compares, more than one simultaneous conflict will be taken via priority encoder. The INDEX REG is used in the EACALC to address the INDEX AC for the X portion of the calculation.

6.23 ERROR CONTROL

Parity from MD MUX 9-12 and 32-35 to form a parity bit for the AC ADDRESS BUFFERS; byte 0 parity from the MD MUX is carried with the instruction code. Parity is generated for TAG bits 0-6 as they exit the TAG module for the M BOX. Parity is checked at the instruction code and AC address readouts, at the TAG input (returning from M), at ISET readout, and at IPUT control word readout.

6.24 DIAGNOSTIC INTERFACE

IBOX control logic active scan path includes the TAC REG, IPUT REG, NS REG and JS REG, the 3 next instruction pipes, DI REG (valid bit count buffer write address), CM REG (updated +1 count), TAC BUFFER delay, and INDEX AC write address REG.

IBOX control logic passive scan path consists of the AD REG (which addresses IPUT and ISET), the data load REG of the EARLY DECODE ARRAY (the lookup which provides information regarding the instruction, such as EACALC= JUMP target, write pretest required, memory operand read required, and skip/jump probable), and the write enables to IPUT, ISET,and the EARLY DECODE ARRAY.

CHAPTER 7

FPA

7.1 GENERAL

<TO BE SUPPLIED AT A LATER DATE>

7.2 ARRAY CONTROL

<TO BE SUPPLIED AT A LATER DATE>

7.3 ARRAY CONTROL STORAGE

<TO BE SUPPLIED AT A LATER DATE>

7.4 OPERAND BUFFERS

<TO BE SUPPLIED AT A LATER DATE>

7.5 MULTIPLIER FORMATS

<TO BE SUPPLIED AT A LATER DATE>

7.6 DIVIDE ALGORITHM

<TO BE SUPPLIED AT A LATER DATE>

7.7 POST NORMALIZATION

<TO BE SUPPLIED AT A LATER DATE>

7.8 ERROR CONTROL

<TO BE SUPPLIED AT A LATER DATE>

7.9 DIAGNOSTIC INTERFACE

<TO BE SUPPLIED AT A LATER DATE>

7.10 MULTIPLY ARRAY

<TO BE SUPPLIED AT A LATER DATE>

7.11 AC COPIES

<TO BE SUPPLIED AT A LATER DATE>

7.12 PARTIAL PRODUCT GENERATION

<TO BE SUPPLIED AT A LATER DATE>

7.13 FULL PRODUCT GENERATION

<TO BE SUPPLIED AT A LATER DATE>

7.14 DIVIDE PATH

<TO BE SUPPLIED AT A LATER DATE>

7.15 MODULO 3 RESIDUE CHECKING

<TO BE SUPPLIED AT A LATER DATE>

7.16 ERROR CONTROL

<TO BE SUPPLIED AT A LATER DATE>

7.17 DIAGNOSTIC INTERFACE

<TO BE SUPPLIED AT A LATER DATE>

CHAPTER 8

MBOX

8.1 INTRODUCTION

The 2080 mbox is composed of three principal functional units, the data and page table cache, an integrated memory control and the I/O box interface.

8.2 TECHNOLOGY

The 2080 mbox will occupy five extended Hex multiwire modules. It will use 100K ECL logic and 1KX4 and 256X4 memories, packaged in 24 pin dips. Ten pin Sip terminator packages provide 55 ohm resistors and -2v decoupling capacitors.

8.3 MBOX FUNCTIONS AND FEATURES

1. Pipelined design
2. High bandwidth
3. Single or double word fetch
4. Integrated memory controller
5. 8192 word data cache.
6. ECC on cache
7. I/O thru the cache
8. 25 bit physical memory address space
9. Microcontroller design

8.4 PRINCIPLE OF OPERATION

8.4.1 Overview

The Mbox combines microcoded controllers with small amounts of control logic to realize high speed with a minimum of unchecked control logic. High performance is achieved using pipelined design, 100K logic, and operations such as "fetch under miss", and double word fetch.

8.5 CACHE DESCRIPTION

The 2080 cache consists of a page table cache and a data cache. The organization of the page table and data cache are very similar to the kl10, the main difference being the increased size of both 2080 caches. In the 2080 mbox, I/O memory operations access the cache before going to main memory. If a read operation finds its data in cache then that data is returned to the I/O device, otherwise a request is made to main memory. Write operations also check the cache first and will invalidate any locations which match the referenced address. Writes are always done to main memory and never write data into the cache.

not 2K? Impact on our use of STKVAR's?

what's in here? How is table indexed?

8.5.1 Page Table Cache

The page table cache is 1K words long, 1 way associative, and has a 1 word block size. Each entry in the table consists of an valid bit, user bit, 3 page state bits, 12 bit directory, and a 16 bit physical page number (PPN). In the KL10, only the paging operations for TOPS-20 paging were controlled by the Ebox microcode. In the 2080 Mbox, all paging is controlled by the Ebox. Page table cache refill time is reduced (from the KL10) with the use of a cache of section pointer entries. The section pointer cache will be accessed and controlled by the Ebox microcode and it will use part of the Ebox scratch memory. Since the virtual address has expanded to 30 bits plus "User", the directory for the paging cache, plus the physical page entry now total 33 bits. No refill algorithm is implemented in the Mbox, and all paging cache misses or access control violations are reported to the EBOX. As in TOPS-20 paging on the KL10, the Ebox must be able to follow the appropriate (TOPS-10 or TOPS-20) algorithm, and write the paging cache, without restarting the instruction.

Describe algorithm for assigning V.A. to location

4 entries per slot

8.5.2 Data Cache

The data cache contains 8K words long, is 4 way set associative, has a 4 word block size, and is physically addressed. The data cache is organized in 4 quadrants of 2K words each. Each quadrant consists of 2 sets of 1K rams which are interleaved and can be read in parallel to achieve double word fetch. The 2080 mbox achieves high performance thru both high bandwidth and high hit rate. To achieve a bandwidth of 50 megawords/sec., each cache quadrant contains two independent sets of rams, which can be cycled concurrently to produce words 0 and 1, every 40ns.The expected hit rate ,with a data cache size of 8K words,is 98% . In order to utilize an 8K cache it is necessary to convert the cpu virtual addresses to physical, which requires cycling the paging cache in series with the data cache. This incurs a small penalty in access time (10-15 ns) which is more than compensated for in higher bandwidth and hit rate. The physical address size has been expanded from 22 bits to 25 bits necessitating an increase in the size of the cache directory from 13 to 16 bits of physical page number. An autonomous cache sweeper is included, which will be invoked by the same instruction as in the KL10.

necessary for whom? *

32 ma

8.6 MICRO MACHINES DESCRIPTIONS

Most of the functions of the Mbox are controlled by two micro machines. There is a small amount of hardwired control logic which is required to speed up certain operations.

8.6.1 Data Mover Micro

The Data Mover micro deals with data transfers between main memory, I/O , and the data cache. The Data Mover contains an interface to these three units , a 16 word register file used as a general purpose buffer, and the ECC generator and checker. The micro cycle time is 20ns, however some operations such as memory data transfers and I/O box operations which take 40ns to complete but use one 20ns micro cycle per operation can be interleaved in the Data Mover, by using alternate 20ns micro cycles. Each interface contains its own state register which is updated by the Data Mover and is also used to form the micro address for some operations. The principal functions are the following.

8.6.1.1 Main Memory -

8.6.1.1.1 Read Return - The memory controller requests a data mover cycle n ticks before a memory array sends data back to the Mbox. Since the memory controller has highest priority, the next micro cycle services the memory request. In servicing the memory request, the data mover loads a word from the memory bus at the end of a 40ns memory bus transfer, does an ECC cycle, and stores the word in the Data Mover and Cache register files. This sequence is repeated 3 more times, regardless of the number of words in the original request. The tag associated with this request indicates the destination of the data; I/O box, cache refill, or CPU. In the case of an I/O return, the data mover calls an I/O box return subroutine within the data mover which sends the 1/4 words to the I/O box. This operation may be initiated before the read return is complete and could be interleaved with it. In either of the other two cases the data mover must request a cache micro cycle. The cache micro then returns the appropriate word(s) to the Ibox and writes the four word block into the data cache.

8.6.1.1.2 Write Start - At the begining of a Main memory write cycle, the memory controller requests a data mover cycle. A tag associated with the request indicates the buffer location in the register file from which to take the data. The data mover then executes a 20ns cycle to fetch word zero of the block and load it into the memory data register. The data mover then either pauses or services an I/O or cache request in the next 20ns cycle. Three more of these 40ns periods are repeated , regardless of the number of words actually written.

8.6.1.2 I/O Transfers -

8.6.1.2.1 Read Cycle - The I/O box requests a cycle by asserting Mbox request, which is sampled by a flip flop at the mbox every 20ns. The data mover micro will service the request when there are no higher requests active. The micro cycle will check the Mbox address and data queues as well as the state of the I/O interface and if all are in the appropriate condition will then assert Mbox grant for 40ns to the I/O box. At the end of a 40ns cycle, the data mover loads the I/O box C/A word into location xy of the register file and dispatches on the I/O request qualifiers. For a read cycle, the dispatch cycle reserves an entry in both the address and data queues, and requests a cache micro cycle. The I/O interface will not service another I/O request until the cache micro has accepted the previous read cycle. Once the cache micro has accepted the read cycle, the I/O interface then waits for a cache return or memory return to bring back the data, however it may service other I/O requests in the interim.

8.6.1.2.2 Write Cycle - The I/O box request and C/A cycle are handled just as in a read cycle however the dispatching is delayed until the one or four words are loaded into the register file. Mbox grant is asserted for an additional 2 or 5 40ns cycles in order to load the one or four words. When the last word is loaded the data mover then requests a cache micro cycle and as in a read cycle will not service another I/O request until the previous write is accepted by the cache micro.

8.6.1.3 Cache Cycles -

8.6.1.3.1 Cycles To Cache - Cycles directed to the cache are the I/O read/write or memory data return as described previously. In these cases the Data Mover need only send a request signal to the Cache micro, as the remaining information (address, cycle type, and data words) have already been written in the cache micro register file by the data mover.

8.6.1.3.2 Cycles From Cache - The cache micro requests the data mover to load data into the register file for an I/O cache hit, cache writeback, or LCC cycle.

8.6.2 Cache Micro

The cache micro handles externally generated requests such as Ibox or I/O (passed on the the data mover), internally generated requests such as writebacks, refills, sweeps, pager clears, and also queues requests to the memory controller. This micro interfaces to the Ibox, the Data Mover, and the memory controller. The basic micro cycle time is 20ns, however most operations take multiple cycles. A major cycle can be started every 40ns and generally take 80ns to complete.The micro also contains a 16 word register file, which is only written by the Data Mover and therefore always contains the identical data as the Data Mover register file.

8.6.2.1 Ibox References -

8.6.2.1.1 Read Cycles - In a typical operation, the Ibox would request a cycle one tick (20 ns) ahead of when its VMA register will be loaded. The Mbox controller would issue a grant signal, "MBOX GRANT" which will cause the Ibox VMA to be gated onto the VMA bus, and the cycle-type information from the Ibox to be examined. Assuming a virtual read request was issued, the controller would allow adequate time for the various paging and cache comparisons, and ECC checks to stabilize. At such time, if all conditions were successful, the data and check bits would appear on the Mbox output lines, and a signal would be supplied to the Ibox indicating it can proceed. During the last tick of the major cycle, arbitration would take place to award the next cycle.

If the desired word were not in the cache, the Mbox controller would retreive it from main memory. Memory requests will in general be 4 word requests, as in the KL10. If some of the words are not needed in the cache, the Mbox will discard them. When data finally comes back from memory, the Mbox returns it to the Ibox along with Mbox response and a tag indicating the Ibox source and register file location to store the data. The Mbox controller will then proceed to write the received words into the cache.

8.6.2.1.2 Write Cycles -

8.7 DETAILED IBOX INTERFACE DESCRIPTIONS

8.7.1 Ibox Virtual References

reads and writes:

The following are the signals passed between the Ibox and the Mbox for Virtual (paged) read and write references.

* Note: physical references use the same interface signals as the virtual reference signals with the exception of the items marked below with an asterisk.

When a reference is made by the Ibox with paging turned off but without PHYS REF asserted, the Virtual reference VMA BUS format is still used. Is this desirable?

1. * Ibox - "PHYS REF" (physical reference) false.

2. * Ibox - "VMA BUS" as per figure 3.1. These enable checking of sub parts of the VMA BUS.

3. Ibox - VMA bus parity - four parity bits over parts of the field.

4. Ibox - "READ" - Asserted on or before VMA and Mbox request.

5. Ibox - "WRITE" - Asserted on or before VMA and Mbox request.

6. Ibox - "2 WORD" - indicates Ibox is requesting 2 consecutive words begining at an even word location. Asserted on or before Mbox request.

7. * Ibox - "WRITE TEST" - Mbox will test legality of a write before doing any function. This signal is guaranteed to only be asserted on virtual references. The Mbox will do a write test if and only if this signal is asserted.

8. Ibox - "MBOX REQUEST" - This will be asserted not more than one Mbox clock tick before the VMA is valid. Request type signals such as "PHYS REF", "READ", "INTERLOCK", or "WRITE TEST" will be asserted on or before Mbox request time.

9. Ibox - "EBOX DATA" - 36 bits of data plus ECC check bits, 43 bits total. It must be valid at most one Mbox tick after "MBOX REQUEST".

10. Ibox - "LOOK" - Look in the cache for this reference. (If a virtual reference, the Mbox ands "LOOK" with PT cacheable.)

11. Ibox - "LOAD" - Load new data into the cache. Look must be true. (If a virtual reference, the Mbox ands "LOOK" and "LOAD" with PT cacheable.)

12. Ibox - "INTERLOCK" - Requests are only interlocked if "INTERLOCK" is true

13. Ibox - "WRITE DATA VALID" - Asserted by Ebox when the data being written is available at the Ebox Data Out Lines.

14. Ibox - "ABORT MBOX CYCLE" (As in KL10 AC Ref) This may be asserted at almost any time during any cycle. Exception cases will be specified later. It will also be used by the Ebox to clear a page fail state.

15. Mbox - "MBOX GRANT" - indicates to Ibox that Mbox is servicing current Ibox request.

16. Mbox - "MBOX DATA" - 36 bits of data plus ECC check bits, 43 bits total. Valid when "MBOX RESPONSE" true.

17. Mbox - "MBOX RESPONSE" - On write_; Mbox received write data. On read_; Data available at output latch.

18. Mbox - "PAGE FAIL" - This signal is the catch all for most exception conditions including page table access failures, ECC errors and NXM traps. The following conditions result in page fail:

How will IBOX respond to these conditions?

Will software see all of these? Where are bits defined?

1. * Page Table cache no match.

2. * Page table write access failure.

3. * Page table Written state transition.

4. ECC or parity error.

5. NXM (Non-existant memory) error.

6. Incomplete memory cycle. What does this mean?

7. CST update

19. Mbox - Five or six bit problem type code that indicates the type of failure for the above sources of page failure. This problem type code will be used by the Ebox microcode to dispatch to the appropriate service routine. References which receive a page fail will not get an Mbox response.

Figure 8.1

VMA bus Virtual Reference Format

```
 0     4 5 6               17 18          26 27          35
!-------------------------------------------------------!
!Must   ! !      Virtual     !   Virtual    !  Virtual    !
! be    !U!      Section     !    Page      !   Line      !
!zero   ! !                  !              !             !
!-------------------------------------------------------!
```

8.7.2 Ebox Physical References_

reads/writes:

* Note: physical references use the same interface signals as the virtual reference signals with the exception of the items marked above with an asterisk.

All others above plus the following apply.

1. Ibox - "Physical Reference" true

2. Ibox - VMA bus as per figure 8.2 below.

Figure 8.2

VMA bus Physical Reference Format

```
 0            8 9 11         17 18          26 27         35
!--------------!---------------------------------------------!
!Must          !                                             !
! be           !            Physical Address                 !
!zero          !                                             !
!--------------!---------------------------------------------!
```

25 bits? p. 8-3

where's 10?

What's this?

8.7.3 Ebox Page Table And PT Directory Read Special Function.

10 bits?

Page table read supplies the (page table address) on bits 17 to 26 of the VMA BUS. Both the page table and the page table directory data are returned on the MBOX DATA lines in the format specified by figure 8.3 below.

Figure 8.3

Page Table Cache Read/Write Format

```
 0     4 5 6         17  20                            35
!--------------------------------------------------------!
! Access!U!   Page     ! ! Physical page number           !
! state !S!   table    ! !                                !
! bits  !E! directory  ! ! 11 - 26                        !
!       !R! 6 - 17     ! !                                !
!--------------------------------------------------------!
```

8.7.4 Ebox Page Table And PT Directory Write Special Function.

The Ebox supplies the page table address on bits 17 to 26 of the VMA BUS. The data for the specified page table entry and the page table directory entry is supplied on the EBOX DATA OUT lines with the format of figure 8.3 above.

8.7.5 Sweep Functions

Sweep special functions will consist of a signal "SWEEP REFERENCE" presented to the Mbox with two or more qualifiers_; one from group 1 below and one or both from group 2 below:

Do we need any cache sweeping other than "all pages"? *

Group 1 -

1. Sweep one word
2. Sweep one four word block
3. Sweep one page
4. Sweep all pages

Group 2 -

1. Invalidate cache
2. Validate memory

8.8 DETAILED CACHE FUNCTIONAL DESCRIPTION

To be supplied.

8.8.1 Cache Sweeps

Cache sweeps will be done by the cache microcontroller. They will be significantly faster than the KL10 and will continue to be able to run asynchronously to and in parallel with the Ibox.

8.8.2 Cache Writeback/refill

8.9 DETAILED PAGE TABLE FUNCTIONAL DESCRIPTION

All of the required state information for access of a page will be encoded into three bits. This includes the states involving the access, writable, modified, cacheable and writeback conditions in the PT directory. They were to be encoded into three bits (eight states total) as follows:

1. Page Not in hardware PT (not valid)

2. Not cacheable, not writeable

3. Not cacheable, writeable, not written

4. Not cacheable, writeable, written

5. Cacheable, not writeable

6. Cacheable, writeable, not written

7. Cacheable, writeable, written

* Do we need "cacheable" as a page state ?

A fourth bit which is independent of the above states, would indicate that a CST update is needed on next refill of the page table.

A fifth bit which is independent of the above states, ("keep me around") would indicate that this location should not be cleared on an ordinary pager clear. An unconditional pager clear would override this bit and clear all entries.

[More Details to be supplied.]

8.10 FUNCTIONS NOT PERFORMED BY THE MBOX

8.10.1 All Handling Of The UBR/EBR

The UBR/EBR is kept in a scratch pad by the Ebox. It is solely responsible for their maintenance.

MBOX Can't do pager refills

8.10.2 The Map Instruction.

The Map instruction will be implemented in the microcode by tracing through page tables in memory. There is no Mbox assistance and therefore, it will be slower than in the KL10.

8.10.3 Sbus Diag Functions

Sbus Diag function is not required since the console is responsible for configuring main memory.

8.11 PERFORMANCE

8.11.1 Overall Goal And Committment

The goal for the 2080 Mbox is for a cache access time of 80 nanoseconds.

8.11.2 Page Table

2k?

The page table hit rate will be somewhat better than the KL10 for an equivalent software environment in that there are 1024 directory entries versus 128 in the KL10. This will somewhat counteract the effects of the increased number of sections used by software. Page table clears will be done by the cache micro. Entries will be cleared 4 words at a time, every 44ns resulting in a complete clear of the 1K entries in 11us.

8.11.3 Section Pointer Cache

The section pointer cache is provided and controlled by the Ebox microcode using the Ebox's scratch RAM. [See Ebox functional specification.]

8.12 I/O INTERFACE

The Data Mover micro handles the interface to the I/O box (Mbus), on a timeshared basis along with handling the cache micro interface and memory array interface. The data mover contains a state register for the Mbus interface. The data mover handles all of the handshaking and data transfers with the Mbus. As described in chapter 1, the data mover requests cache cycles for the Mbus transactions and the cache micro in turn requests main memory cycles when required.

The I/O-Box arbitrates the bus priority conflicts so that only one request signal and one grant signal are necessary between the M-Box and I/O-Box. The I/O-Box uses the rising edge of MCLKT to strobe data into or out of the multi-port ram circuits that are used as synchronizing buffers.

The following 50 signals constitute the MBUS:

MD<0:35> - This is a bidirectional Memory Data bus. Note that MD<0> is the most significant bit while MD<35> is the least significant.

P - This is a single odd parity bit that indicates the parity of ~~of~~ the data word accompanying it. Note: This is Parity on the MD word only. No other transaction information is tested for parity.

MI<0:3> - This is the four bit identification marker that is used to identify the source or destination of data in the event of inter-leaved data transactions. This is a direct copy of the TTL-Bus TAC bits.

C CYC - This bit is used to indicate a Command/Address cycle. The data present in MD<0:35> is identified as Command/Address information when this bit is set. This bit is never sent from the M-Box to a PORT.

D CYC - This bit is used to indicate a Data cycle. The data present in MD<0:35> is identified as Data only when this bit is set. This Bit is present with every transaction from the M-Box to the I/O-Box.

MCLK - This signal is derived from the master oscillator that is distributed through out the machine. This signal is inverted to provide a strobe signal for the M-Bus edge triggered flip-flops. Data output to the Mbox is synchronized to the leading edge of MCLK. Error information pertaining to the cycle immediately preceding the current cycle is presented with the leading edge of this signal.

MBREQ - This signal requests that the M-Box service the I/O-Box. MBREQ is generated by the I/O-Box only when an entire TTL-Bus transaction is complete (i.e. there are either two words or five word of information available in the MBus Buffer on the I/O-Box) The M-Bus is always granted for two cycles and is extended an additional three cycles when a multiple word transfer is identified.

MBG - This signal is a grant from the M-Box to the I/O-Box. It enables the I/O-Box to drive the MD<0:35> on the next rising edge of MCLKT.

MFAULT - This signal is a signal that is used to represent the detection of a parity error on the previous transaction. The MT bits are not checked for the proper address information.

MBACK -

MUNCOR -

I/O Box Full - this signal is used to indicate to the M-Box that the I/O-box resident buffers are full and that no further transfers from the M-Box to the I/O-Box should be performed until it is negated.

8.13 R.A.M.P. FEATURES

8.13.1 General

All major buses between the Mbox modules contain parity check bit(s). Parity is generated/checked going off each module and is checked again on the receiving module within a few gates. Parity is checked at the data input to all rams before they are written. The control rams for all of the micro machines are also parity checked.

8.13.2 Cache Memory

The page table has a parity check, while the cache directory and the data cache have ECC. When accessing the cache, a parity error in the page table generates a page fail condition for the Monitor to how? correct. A single bit error in the cache directory or the data cache are automatically corrected and the syndrome, address, and data are saved for use by the system. It has been agreed that in order to avoid a performance hit, double bit error detection in the cache directory and the data cache will not be implemented.

8.13.3 Mos Memory

Each memory array board is organized as 512K words by 22 bits. Each read access to a word of memory produces 44 bits of data from the two array boards, with each 22 bit half word including a parity bit for that half word. Each write to memory also generates 44 bits organized as two sets of 22 bits, with half word parity in each set. On read cycles the 44 bits are sent to the Data Mover, where the half word parities are checked and the ECC is recomputed. Single bit errors are corrected and operation is continued. Double bit errors are detected and a trap is generated. to the O/S? Upon occurrence of a detected error , the address, data, and syndrome of the failing access are latched in a register which is accessible to the operating system for error monitoring. Successive single bit errors destroy earlier data stored in the latch so that only the last error is available in the latch.ON occurrence of a double bit error, the error information is stored in the latch, and further single bit errors are corrected but not recorded. A trap to the operating system is also generated and a resetting action from the monitor is required before single bit errors can be recorded again.If a second double bit error occurs after one has been latched, but before resetting the latch, a trap to the console is generated. The trap on double error can be disabled, but whoever runs with trap disabled deserves what he gets.

RAM pla say "console"

X

8.13.4 Error Handling

tell us if there's an error. Why can't console do this?

The proposed method to maintain single error statistics in the main memory and the cache is to have the monitor periodically check the error registers. In the case of a stuck bit in the cache directory or data cache , the monitor can disable the quarter of the cache where the error has occurred until the fault is serviced. This provides for degraded operation of the system. Stuck bits on non ECC'd memories such as control rams or page table are failures where the system requires service before operation can be continued. Moreover, double bit errors are undetectable in non ECC'd memories.

how? And how do we detect it?

8.13.4.1 Address Parity - On the detection of an address parity error on a main memory write cycle the controller will disable the write enable signal to the arrays, thus forcing the MCS rams to do a read operation rather than a write. This prevents overwriting the wrong location. The normal reaction is to automatically retry the write operation a 2nd time in the event the 1st error was intermittent. If that is not successful the Mbox saves the failing address and data and sets an address error flag.

who?

8.13.5 Error Logging

All uncorrectable errors will be reported to the Ebox microcode by means of an page fail.

Error state information will be latched in registers where applicable.

* Should the Mbox microcode have a "logout" area in main memory where it puts information for the use of the monitor?

The Mbox microcode may implement error retry for selected fault conditions (other than correctable ECC errors).

CHAPTER 9

MEMORY

9.1 INTRODUCTION

The 2080 memory is composed of three principal functional units; the memory arrays,the memory control and interface, and the memory bus repeater.

9.2 TECHNOLOGY

The 2080 memory array will occupy two to sixteen Extended Hex multilayer modules. It will use 100K ECL logic to interface to the memory bus and 64K MOS rams for storage.

9.3 OVERVIEW

The 2080 main memory achieves a low cost, simple fast, pipelined design utilizing micro-programmed controllers and existing technology; +5V 64K NMOS RAM, 100K ECL and 74S IC's. The pipelined design is necessary to support the high memory bandwidth required by the 2080 Ibox and I/O system. The main memory is tightly coupled to the CPU due to the close physical proximity of the array boards and the memory controller which is integrated into the Mbox design.

9.4 FUNCTIONALITY

9.4.1 Organization

The two major components of the memory are the controller and the array modules. The controller is integrated into the Mbox and consists of 3 functional units; the address queue unit which queues up requests and starts cycles, two independent timing networks which provide RAS, CAS, etc. to the arrays, , and the Data Mover unit which handles data transfers to and from memory. The data mover unit also performs data transfers for the I/O interface and the cache. Each array module is organized as 512K x 22 bits and each pair of array boards constitutes a memory bank. A bank is subdivided into a lower and upper section of 256K words each which can be cycled independently. A bank can service two independent 4 word cycles which are staggered by 240 ns. A bank contains a four word data buffer, two independent timing buffer registers and refresh logic for use during battery backup.

7

9.4.2 Cycles

The memory executes 3 types of cycles; refresh, read and write. Refresh cycles are covered in 13.2 . A read or write cycle ties up a section of 256K words for 400ns, the cycle time of the 64K RAMS. To improve the memory bandwidth, the controller is able to overlap cycles in different 256K sections by starting a cycle every 240ns. Of the 4 combinations of "in progress" and "next" read or write cycle, all can be overlapped except the case of "read in progress" followed by "next write". This is due to a potential conflict of read and write data on the common set of data lines on the memory array backpanel.

9.5 ARRAY ADDRESSING

In order to provide for flexibility in dealing with larger memories in the future, which will be provided thru a memory bus repeater and or eventual use of the 256k MOS RAM, the memory controller incorporates a mapping RAM. This RAM maps address bit 11-16 of a memory address into one of 16 banks. In addition to the bank number, the mapping RAM also selects upper or lower section in the bank and an interleave mode bit. The memory backpanel in the basic system contains 3 banks. The system is designed to accommodate 1 bus repeater whose backpanel accommodates 8 more banks. This mapping RAM allows 64K or 256K RAM modules to be plugged into any bank and interleaving may be set up individually on each bank.

9.6 REFRESH

The array board operates in two refresh modes, local and system. Local mode is in effect during battery backup operation or occurs whenever a refresh cycle signal has not been received for greater than 25 micro- sec. In local refresh mode, each array board generates all of its refresh signals; refresh request, RAS timing, and refresh address. Refresh address is always supplied by the array board for either a local or system refresh. System refresh mode is in effect during normal operation when DC power is ok. The memory controller generates a refresh cycle request every 16 U sec and provides a RAS signal to the array. In the event that Mbox clocks are stopped, the array would switch over to local refresh mode in about 32 U sec. When Mbox clocks are turned back on, the system must wait 32 U sec before attempting to use memory in order for the refresh circuitry to synchronize.

9.7 QUEUES

The Mbox contains an 8 word address queue and a 16 word data queue in order to keep track of pipelined operations. Each queue has a pointer to the head and a pointer to the tail. New information is added to the tail of the respective queue and the pointer is incremented by one. As a request is serviced the head of the queue pointer is incremented by one. These queues could provide some amount of past history in the event of a machine error. Since old requests in the queues are never cleared but only overwritten when the location is reused, the console could examine all the entries in the queues and retrieve the past n requests which had not been overwritten. In the case the queues are full with pending requests there may be no usefull information of past requests.

9.8 ARRAY INTERFACE

The 2080 storage array interface consists of 127 signals.

```
MOS     DATA     <00:42> H
MOS     ADR      <16:33> L
MOS     RRT      <0> EN H
MOS     ARRAY    <0:7> EN H
MOS     SLOT DATA    <0:1> H
POWER   OK           <0> H
MOS     CLK          <0:8> H

MOS TIM A            RAS0 H
                     RAS1 H
                     CAS0 H
                     CAS1 H
                     WR 0 H
                     WR 1 H
                     BNK SEL A0
                     BNK SEL A1
                     BUS S1 A0
                     BUS S1 A1
                     BUS CLK A0
                     BUS CLK A1
MOS TIM B            RAS0 H
                     RAS1 H
                     CAS0 H
                     CAS1 H
                     WR0  H
                     WR1  H
                     BNK SEL A0
                     BNK SEL A1
                     BUS S1 A0
                     BUS S1 A1
                     BUS CLK A0
                     BUS CLK A1
```

9.9 PERFORMANCE

9.9.1 Access Time

The read access time seen by the Ibox measured from the time it clocks an address to the Mbox until it clocks data back into batch is 420ns for the 1st word and 560ns for the 4th word. The read access time seen by the I/O box measured from the time it clocks a Command Address word to the Mbox until it clocks data back into is 480ns for the 1st word and 620ns for the 4th word.

9.9.2 Cycle Time

The cycle time for either a read or write cycle is 400ns for successive cycles to the same bank. However, cycles to different banks may be started every 240ns.

9.9.3 Bandwidth

Successive cycles of the same type can be started every 240ns which results in a transfer rate of 16 megawords/sec . Successive cycles of alternate type can be started every 320ns which results in a transfer rate of 12 megawords/sec .

9.10 SIZE

9.10.1 64K Part

The basic memory system, using 64K RAM parts, provides memory sizes from 512K to 4096K words, in increments of 512K words. Use of a memory bus repeater would double the capacity given above to 8192K words.

9.10.2 256K Part

The advent of the 256K RAM part will quadruple the above capacity. A basic system would provide 2048K to 16384K words in increments of 2048K words. The repeater doubles this to a maximum of 32768K which is the limit of implemented physical memory address space. The memory bus repeater and 256K RAM are not committed products but are means of extending memory capacity at a future date if required.

9.10.3 16K Part

The memory system can also accommodate 16k parts in the event there is a problem with availability of 64K parts. In this case the memory sizes would be 128K to 1024K words in increments of 128K words. Use of a memory bus repeater would double the capacity above to 2048K words.

9.10.4 Mixed Parts

Systems which have a mixture of 64K modules and 256K modules are allowed with no restrictions of which banks contain 64K or 256K modules.

9.11 INITIALIZATION

Each time the machine powers up, the console must go thru an initialization routine to determine which banks contain memory modules , what size the arrays are, and ensure that bank 0 contains modules. Bank 0 module slots are wired up to provide far end memory bus terminators. Each memory array module asserts a true high backpanel signal when it is selected. Different backpanel pins are used for the 64K and 256K part. All of the left half word (00-17) banks have the 64K size pins bused together, the 256K size pins bused together and the corresponding thing is done for the right halfword (18-35) banks. The memory controller then sees 4 signals; left half 64K, left half 256K, right half 64K, right half 256K. The console must cause the memory control to address each bank and record the number and types of arrays present.

9.12 RECONFIGURATION

9.13 DIAGNOSTIC FEATURES

9.13.1 Arrays

The selected array checks address parity and returns an error indication to the memory controller. Arrays do not check data parity.

9.13.2 Controller

9.14 POWER REQUIREMENTS

9.15 BATTERY BACKUP

9.16 ERROR HANDLING

CHAPTER 10

IO SUBSYSTEM

10.1 DEFINITION OF TERMS

The following list attempts to concisely define some of the terms used in this specification.

"COMMAND" - A block of data, generally used to supply control functions to either the PORTS or the PORT DRIVERS.

"RESPONSE" - A PORT STATUS message to the 2080 PORT DRIVER which consists of several 2080 words assembled and chained into a RESPONSE QUEUE" by the 2080 PORT.

"PACKET" - The form in which data is passed over to the PORT. This is basicly a string of bytes.

"CONTROL MESSAGE PACKET" - A PACKET which contains a "COMMAND" directed to either the 2080 PORT or to a remote PORT.

"MESSAGE PACKET" - A PACKET which contains a "COMMAND" directed to a HOST CPU.

"DATA PACKET" - A PACKET which contains data to be routed to a named memory buffer in a HOST CPU.

"2080 PORT" - A microprocessor based hardware port that interfaces to the IOBUS and to BUS ADAPTERS.

"PORT DRIVER" - A software package in a HOST CPU which exercises control over both local and remote PORTS.

"QUEUE ENTRY" - A fixed length memory field (12 2080 words long) which is linked into a QUEUE CHAIN.

"QUEUE CHAIN" - A chain of QUEUE ENTRIES.

"HOST" - An intelligent MEMORY/CPU combination.

"CI BUS" - A high speed serial line bus which interconnects HOSTS.

"PLI" - Port to Link Interface.

"PCB" - PORT CONTROL BLOCK

"DMA" - DIRECT MEMORY ACCESS

"BDT" - BUFFER DESCRIPTOR TABLE

"BD" - BUFFER DESCRIPTOR

"RCB" - REGISTER CONTROL BLOCK

"ISW" - INTERRUPT STATUS WORD

"IVW" - INTERRUPT VECTOR WORD

"IOBUS" - bus that connects IOBUS PORTS to 2080 memory.

"BUS ADAPTER" - Hardware which interfaces 2080 PORTS to corporate and industry wide buses (ICCS, IBM CHAN, UNIBUS).

10.2 GOALS AND STRATEGY

10.2.1 GOALS

We perceive the goals of the I/O Subsystem to be as follows:

1. Provide communications,hardcopy,disc and tape functionality appropriate to a system with 3.5 to 10 times KL10 throughput.

2. Provide the ability to connect new devices and interconnect which are part of the corporate interconnect program (e.g. CI bus, NI bus, HSC50, MERCURY) as well as some current 2060 peripherals (e.g. DN20s, RP06s, TU72s).

3. Provide sufficient mass storage bandwidth to handle peak demands on the I/O subsystem.

4. Provide the necessary console functionality to support the system maintenance philosophy.

5. Design the I/O subsystem so that it is extensible. That is, provide the capability of adding future device interfaces in a cost/effective way.

6. Meet all of the above goals on schedule in the most cost/effective way.

10.2.2 STRATEGY

"PRIMARY NEEDS" are those needs which need cost/effective solutions (i.e.; design center needs). ""SECONDARY NEEDS" are those which need solutions but not necessarily the most cost/effective solutions. An example of a primary need that was recognized on the KL-10 project, was the need for a cheap and fast primary memory system. The SBUS/MA20 was the solution to this need. A secondary need recognized on the KL-10 project, was the need to interface to existing memories (i.e.; ME10's). An SBUS - KI10 mem bus adapter called the DMA20 was the solution to this secondary need. Listed below are the PRIMARY and SECONDARY NEEDS are for the 2080 I/O subsystem.

PRIMARY NEEDS

1. Provide a MASSBUS based tape and disk subsystem to allow a customer to retain some of his investment in MASSBUS peripherals and allow Digital to leverage our development by using existing MASSBUS software.

2. Allow for connection of DN20 hardware to enable a smooth transition to MERCURY based communications subsystems

3. Provide a connection to the HSC50 disc subsystem. The HSC50 is the only way we have to talk to the latest dec discs (i.e.; R80, R81, RL04).

4. Provide a high speed inter-computer communications link. This is needed to support the full corporate multi-processing scheme.

5. Provide a connection to the MERCURY communications subsystem. MERCURY is the next generation DN20. In many configurations it should be more cost/effective than DN20.

6. Provide a connection to high performance tapes, and high density disc drives. Systems over 3 mips will need 2400 Mbytes and greater disc storage along with fast reliable tape systems to do back-up.

SECONDARY NEEDS

1. Growth path for KL-10 customers. Our customers realize that our next generation computers must be optimized around new technologies in order for DEC to compete. It is therefore reasonable to characterize this need as a secondary need (i.e.; the DMA20 example above).

2. A back-up to the HSC-50 and MERCURY, in case there is a serious schedule slip. Back-up strategies are SECONDARY NEEDS by definition.

Listed below are the options we are currently designing to meet the primary needs. These options are optimized to meet these needs.

1. A PORT/MBA combination that will enable us to talk to current MASSBUS devices.

2. A PORT/UBA combination that will enable us to interface to the DN20 hardware using protocols similar to those planned for MERCURY.

3. A PORT/LINK option that allows us to communicate over the ICCS. This connects us up to the HSC50 and MERCURY in the only way possible at the current time. ICCS also provides a high-speed channel to channel connection for loosely coupled multi processing.

4. A PORT/IBM BMUX CHANNEL option that allows us to talk to high performance tapes and 8650 high density disc drives (STC 8650 drives).

10.3 IOBUS PORTS AND BUS ADAPTERS

Refer to figure 5.10

10.3.1 I/O PORTS

10.3.1.1 I/O PORT Functionality - The I/O PORTS provide the following functionality:

1. Provides an IO micro processor optimized for block transfers.

2. Supports a software port driver whose architecture is similar to the VAX CI port architecture.

3. Supports the CI BUS via an CI LINK module which can connect to the front-end of the port.

4. Supports an IBM 360/370 CHANNEL BUS via an IBM CHANNEL BUS module which can connect to the front-end of the port.

5. Supports a UNIBUS via a UNIBUS ADAPTER module which can connect to the front-end of the port.

6. Supports the MASSBUS via a MASSBUS ADAPTER module which can connect to the front-end of the port.

7. Supports four different data packing modes during data transfers; INDUSTRY COMPATIBLE, CORE DUMP, and HIGH DENSITY, 7 BIT ASCII.

```
  TO EBOX/MBOX

       /|\
        |
        |
  ----------
 |          |---------------------------------------------------------
 | IOBOX    |           20-40 Megabyte/sec (RAW) IOBUS
 |----------|---------------------------------------------------------
 | CONSOLE  |      |              |              |              |
  ----------       |              |              |              |
   |     |         |              |              |              |
   |     |         |              |              |              |
 CTY  KLINIK   -------        ------         ------         ------
              |       |      |      |       |      |       |CUSTOM|
              | PORT  |      | PORT |       | PORT |       |      |
              |       |      |      |       |      |       |INTERF|
               -------        ------         ------         ------
                   |              |              |             |
           LINK    |      LINK    |              |             |
           BUS     |      BUS     |      LINK    |             |
                   |              |      BUS     |             |
               ------         ------          --- ---
              |CI    |       | IBM  |       | UBA    |
              |      |       |      |       |        |
              | LINK |       | CHAN |       |        |
               ------         ------         -------
                   |              |              |
             CI    |      IBM     |   UNIBUS     |
             BUS   |      CHAN    |              |
                   |      BUS     |              |
                   |              |              |
                   |              |              !
             6 MB/SEC      2MB/SEC        2-3MB/SEC
```

Figure 5.10

10.3.1.2 I/O PORT Performance - Each I/O PORT can sustain a transfer rate of up to 6 MB/SEC.

1Ø.3.1.3 I/O PORT Physical Characteristics - Each I/O PORT will fit on a single extended hex module.

1Ø.3.1.4 IOBUS PORT CONTROL STATUS RECISTER - This register controls the software operation of the port. The register contains enables and valid bits for the queues and event flags for interrupt generation. All bits except PHE are read/write.

```
 Ø Ø Ø Ø Ø Ø 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
 4 5 6 7 8 9 Ø 1 2 3 4 5 6 7 8 9 Ø 1 2 3 4 5 6 7 8 9 Ø 1 2 3 4 5
-----------------------------------------------------------------
!P!P!P!P!P!P!P!P! !I!R!F!F!M!P! !C!C!C!C!Q!Q!Q!Q! ! ! ! ! !I!I!P!
!N!N!N!N!N!N!N!N! !T!Q!Q!Q!S!H! !Q!Q!Q!Q!V!V!V!V! ! ! ! ! !E!N!F!
!7!6!5!4!3!2!1!Ø! !C!A!A!E!E!E! !A!A!A!A!3!2!1!Ø! ! ! ! ! ! !I! !
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !3!2!1!Ø! ! ! ! ! ! ! ! ! ! !T! !
!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!-!
                     * * * * * * *
```

All bits marked with an asterisk (*) are in the INTERRUPT SOURCE WORD (ISW).

BITS <4:11>: PORT NUMBER (PN<7:Ø>) = The ICCS node number of this Port. The number is a function of dip the position of dip switches on the module. These bits are read only.

Bit <13>: INIT COMPLETE (ITC) = A Port initialization has been completed.

BIT <14>: RESPONSE QUEUE AVAILABLE (RQA) = Indicates that the Port has inserted an entry on an empty Response Queue.

BIT <15>: FREE QUEUE AVAILABLE (FQA) = Indicates that the port has placed an entry on an empty Free Queue.

BIT <16>: FREE QUEUE EMPTY (FQE) = A free queue entry was required for processing and was not available. Processing of incoming packets is halted until entries become available. Command queue processing continues.

BIT <17>: MEMORY SYSTEM ERROR (MSE) = This is a flag to indicate that a memory error was detected on a 2Ø8Ø memory access. The errors are Nonexistent Memory (NXM) and Memory Error (ME).

BIT <18>: PORT HARDWARE ERROR (PHE) = INclusive "OR" of the LSPE and CSPE bits of the MCSR. This bit is cleared by hardware INIT (MCSR<Ø9>=1. BIT <2Ø-23>: COMMAND QUEUE AVAILABLE (CQA<3:Ø> = Indicates that an entry has been inserted in the specified Command queue by the Port or the Host. CQA is set by the Port or the

Host at the time of insertion of the command and cleared by the Port during the interlocked Remque operation if the queue is found to be empty.

BIT <24:27>: QUEUE VALID (QV<3:0>) = Indicates that the specified queue is valid to use. This is the mechanism used to halt processing on a command queue. QV is set by the Host to allow processing and is cleared by the host to halt processing of the queue. QV is also cleared by the Port to halt command queue processing if an error in command execution occurs with the H bit set in the command.

BIT <33>: INTERRUPT ENABLE (IE) = Enable interrupts on flags and errors. This bit causes the 2080 to be interrupted.

BIT <34>: PORT ENABLE (PE) = Disables the processing of incoming messages. Set and cleared by the Host.

10.3.2 BUS ADAPTERS

There are four types of bus adapters that interface to the I/O BUS PORTS; an CI LINK, A MASSBUS ADAPTER, an IBM CHANNEL BUS, and a UNIBUS ADAPTER.

10.3.2.1 CI LINK

10.3.2.1 CI LINK - The CI LINK provides the interface to the Corporate CI BUS. The CI is a contention-arbitrated, bit serial (70 mbits/sec), packet switched bus. Contention and protocol overhead reduce the effective data bandwidth to approx 6 mbytes/sec. The CI Link consists of two modules. One module is a packet buffer. The other module is called the LINK module. The LINK module is being designed by Distributed Systems in Tewksbury.

10.3.2.2 IBM CHANNEL BUS - The IBM CHANNEL BUS ADAPTER provides the interface to a IBM 360/370 Block Multiplexor Channel Bus. Transfer rates of 2 Mbytes/Sec should be achievable across two subchannels. The IBM CHANNEL BUS ADAPTER will be packaged on a single extended hex module.

10.3.2.3 UNIBUS ADAPTER - The UNIBUS ADAPTER provides the interface to the UNIBUS. Transfer rates of 1.25 Mbytes/sec should be achievable. The UBA can be programmed as the UNIBUS master (i.e.; 2020 UBA) or UNIBUS slave (i.e.; DTE20). Current plans support all devices using a front-end PDP-11. If there are no applications for the UNIBUS adapter to operate as the UNIBUS master, this feature may be deleted.

The UNIBUS ADAPTER will be packaged on a single extended hex module.

10.4 I/O SUBSYSTEM INTERFACE TO CPU AND MEMORY

10.4.1 I/O SUBSYSTEM INTERFACE TO THE CPU

Control information is passed between the I/O PORTS and the EBOX via cached memory locations called "mailboxes" in the IO page. A mechanism will exist that will enable the EBOX to signal the I/O BUS PORT (ring doorbell). Upon seeing the doorbell an I/O PORT will read its' mailbox to pick up a command word and other control information. Control registers in the I/O PORT are read and written via the mailbox. This eliminates the need for READ I/O and WRITE I/O operations and thus simplifies the I/O PORT interface to the rest of the system. I/O registers existing in I/O CONTROLLERS (i.e.; on the UNIBUS, in an IBM CONTROL UNIT) can also be accessed via command packets placed on an I/O PORT command queue.

Each I/O PORT will have a connection to 7 interrupt lines (one line for each of seven interrupt levels) going to the EBOX. Upon receiving an I/O PORT interrupt on one of these seven lines, the EBOX will broadcast a PI Level to all interrupting ports. That port which is interrupting on the broadcasted level will request the IOBUS and write a word (or words) called the Interrupt Source Word into the IO page which will identify the port (i.e.; slot number), the reason for the interrupt, and an interrupt vector.

A total of 14 lines connect the EBOX with the I/O PORT.

PI LINES <1:7>	Each port connects to each one of these seven lines. When a port wishes to interrupt on a particular PI Level it will yank on the appropriate PI LINE.
START SELECT <8:10>	These three lines are used to select a port or are used to broadcast a PI Level.
RING <11>	When this line is asserted, the port selected by the PORT SELECT lines will read a location in the IO page. This is a "DOORBELL" operation.
RESET <12>	When this line is asserted, the port selected by the PORT SELECT lines will reset itself.

BROADCAST <13> When this line is asserted, all ports that are interrupting will look at the number on the PORT SELECT lines. If that number is equal to the PI Level that the port is interrupting on it will do a memory write to a specified IO page location and write its ID and interrupt status.

BUSY <14> The port will assert BUSY in response to a DOORBELL or BROADCAST with a match on the START SELECT lines. BUSY will become unasserted when the port has completed all memory references required for the DOORBELL or BROADCAST.

The "REGISTER ACCESS BLOCK" is an 8 word block which provides an I/O port with the control and data needed to do register reads and writes. In addition, there is room in the eight word block for the port to store relevant status when it interrupts the cpu. there is a "REGISTER ACCESS BLOCK" for each I/O PORT in the machine (total of eight).

word	name	description

0	OPCODE/IOADR	Bits <0:4> contain a command. format is:
		D0=0 nop
		D0=1 valid command
		
		D1=1 read
		D2=1 write
		
		D3=1 bitset
		D4=1 bitclr
		
		Bits <18:35> contain the i/o address of the register to be read/written.

1	DATA	For a register write, this word contains the data to be written. For a register bit modification, this word contains a mask. For a register read, this word will be written by the I/O PORT.
2	OPCODE/IOADR	Same format as word 0
3	DATA	Same format as word 1
4	ISW0	This word is an "interrupt status word". This word is written by the port when it interrupts the cpu. The ISW contains all the information necessary to determine the reason for the port interrupt. Since the port can interrupt on four different pi levels at one time there are four ISW's.An "INTERRUPT VECTOR WORD" (also written by the port when it interrupts) points to the appropriate ISW.
5	ISW1	same format as ISW0
6	ISW2	same format as ISW0
7	ISW3	same format as ISW0

Note that the port will pick up two OPCODE/IOADR words after the doorbell rings. This allows reading and writting two registers at one time. The format of the OPCODE/IOADR word is the same as the

2020 OPCODE/IOADR word. This will make the IOBUS more compatible with the 2020 UBA. During register write operations, the port will be able to write local registers as the data words come off the IOBUS.

Typical sequence for "WRITE I/O" operation:

1. CPU writes OPCODE/IOADR word(s) and data in the REGISTER ACCESS BLOCK.

2. CPU rings doorbell and waits for BUSY to assert and go away.

3. Port sets BUSY.

4. Port reads the first four words (one mem ref) from the REGISTER ACCESS BLOCK, and writes the data into the appropriate register(s).

5. Port clears BUSY.

Typical sequence for "READ I/O":

1. CPU writes OPCODE/IOADR word(s) into the REGISTER ACCESS BLOCK

2. CPU rings doorbell and waits for BUSY to assert and go away.

3. Port sets BUSY.

4. Port reads first four words of the REGISTER ACCESS BLOCK.

5. Port writes data word(s).

6. Port clears BUSY.

10.4.2 I/O SUBSYSTEM INTERFACE TO MEMORY

The I/O Subsystem will be built around a synchronous interlocked TTL bus (IOBUS) which will provide a DMA path between I/O PORTS on the bus and the 2080 main memory (the IOBUS will support up to a maximum of eight I/O PORTS). Memory references by the I/O BUS PORTS over the IOBUS can be either one word or four words per bus transaction.

The IOBUS is granted to requesting I/O PORTS by a centrally located arbitrator located in the console. The IOBUS PORT in slot 0 of the IOBUS will have highest priority.

THE RAW bandwidth of the IOBUS WILL BE 20-40 MBytes/sec. Raw bandwidth is defined is as the upper bound controlled by the bus clock frequency and protocol limits. The EFFECTIVE bandwidth of the bus will be greater than 16 MBytes/sec but less than 40 MBytes/sec.

Effective bandwidth factors are memory access times, memory interleaving and whether we are doing reads or writes. Refer to figure 5.20 which compares the RAW bandwidth of the IOBUS with other systems.

```
MAXIMUM RAW I/O
    BANDWIDTH
(MBYTES/SEC)

128     !
        !
        !
        !
        !
64      !
        !
        !                                               * 168-3MP
        !
        !
32      !
        !                                          * 3033      * 3033-AP
        !
        !            * SEL 32/75              *        *   * 470V/7
        !                                   168-3    2080
16      !
        !
        !                                      * 470V/5   * 470V/6
        !
        !
8       !                * 11/780
        !
        !
        !
        !
4       !          *    *     * 158
        !
        !       145-3  HARRIS S/6
        !
        !
2       !
        !
        !
        !
        !
1       !
        !
        !
        !
        !
       ------------------------------------------------------------

          .25      .5       1        2       4        8         16
```

ROUGH PROCESSOR SPEED (MIPS)

Figure 5.20

The IOBUS interfaces to the 2080 memory through the I/O BOX. The I/O BOX provides the necessary TTL to ECL translation and buffering in order to match the speed of the TTL IOBUS with the faster ECL MBOX BUS. In addition, the I/O BOX will contain the system console. The I/O BOX will be packaged on a single extended hex module.

10.4.3 CONSOLE

The SYSTEM CONSOLE will be the interface for the operator, field engineer, and development engineer (during machine debug). It will boot the system, support the diagnostic philosophy and sense the environment. Console hardware characteristics are summarized below:

1. F11 microprocessor.
2. LA34 (or equivalent) hardcopy terminal.
3. KLINIK and APT communications lines (at least 19.2 KB).
4. 48K - 64K bytes of RAM memory. Sufficient PROM to bootstrap the F11 from system devices.
5. Floppy disc as console load device.
6. Interrupt source for event timming.
7. Diagnostic visibility and control logic.
8. Battery powered calendar chip.
9. System clock control. Start, stop and burst.

10.5 I/O SUBSYSTEM INTERFACE TO THE OPERATING SYSTEM

10.5.1 I/O PORT COMMAND/RESPONSE PROCESSING

The SOFTWARE PORT DRIVER in the 2080 assembles COMMANDS and stacks them as QUEUE ENTRIES on the tail of one of four prioritized COMMAND QUEUE CHAINS.

The I/O PORT asynchronously removes the COMMAND QUEUE ENTRIES from the head of the highest priority COMMAND QUEUE CHAIN, decodes the COMMAND and takes the appropriate action. This action may be entirely internal to the I/O PORT, or it may require that the I/O

| PORT transmit the COMMAND over the PLI to a remote node on the CI
| BUS, an IBM CONTROL UNIT on the IBM CHANNEL BUS, The MASSBUS adapter
| or the UNIBUS adapter. Commands to the UNIBUS or MASSBUS adapter may
| be processed directly by the adapter or may result in one or more bus
| transactions.

The I/O PORT then completes the process by modifying the appropriate status bits in the COMMAND and placing it either on the tail of a RESPONSE QUEUE CHAIN or the TAIL of a FREE QUEUE CHAIN, both of which reside in 2080 memory.

If the I/O PORT places the modified COMMAND on the RESPONSE QUEUE CHAIN; if the RESPONSE QUEUE CHAIN is empty; and if the INTERRUPT ENABLE bit in the I/O PORT'S CONFIGURATION REGISTER is set; the I/O PORT will generate an INTERRUPT to inform the PORT DRIVER of the QUEUE ENTRY.

In this mode of operation the RESPONSE QUEUE entries are used to send status information to the PORT DRIVER. If the modified COMMAND was returned to the RESPONSE QUEUE CHAIN the PORT DRIVER may examine the resulting RESPONSE QUEUE ENTRY to determine the status of its associated COMMAND. (I.E. was the COMMAND successfully executed?). The I/O PORT always returns the modified COMMAND QUEUE ENTRY to the TAIL of the RESPONSE QUEUE CHAIN if an error was encountered during execution of the COMMAND.

The FREE QUEUE CHAIN is simply a list of blank queue entries residing in 2080 memory that both the PORT DRIVER and the I/O PORT may use whenever they need free memory to generate a new COMMAND or RESPONSE QUEUE entry.

The software PORT DRIVER of a remote HOST (CI SPECIFIC) also assembles COMMANDS directed to the 2080 PORT or to the 2080 PORT DRIVER. The remote PORT transmits the COMMANDS as CONTROL MESSAGE PACKETS or MESSAGE PACKETS, whichever is applicable, across the CI BUS. The 2080 PORT stacks the COMMANDS which were transmitted by CONTROL MESSAGE PACKETS as QUEUE ENTRIES on the tail of one of the four prioritized COMMAND QUEUE CHAINS. The 2080 PORT stacks the COMMANDS which were transmitted by MESSAGE PACKETS on the TAIL of the RESPONSE QUEUE CHAIN.

In this mode of operation the RESPONSE QUEUE entries are used to transfer short fixed length messages from a remote HOST CPU to the 2080 PORT DRIVER.

Each of the four prioritized COMMAND QUEUE CHAINS has an implicit priority assignment with QUEUE 0 being the highest priority and QUEUE 3 being the lowest. The various priority levels are used to sequence the execution of the different types of COMMANDS in the most expeditious manner.

COMMANDS fall into three general categories:

1. COMMANDS requested BY THE local 2080 HOST. These are COMMANDS stacked on the TAIL of a COMMAND QUEUE CHAIN directly by the local 2080 PORT DRIVER. The COMMAND is executed when it reaches the HEAD of the COMMAND QUEUE CHAIN.

2. Data transfers initiated by the local 2080 HOST. This is accomplished by a single COMMAND stacked on the TAIL of a COMMAND QUEUE CHAIN by the local 2080 PORT DRIVER. When the COMMAND reaches the HEAD of the COMMAND QUEUE CHAIN it causes a block data transfer over the TTL by a DMA mechanism.

3. Data transfers initiated by a remote HOST. This is accomplished by a single COMMAND issued by the remote HOST. The COMMAND is not processed immediately, but is stacked onto the TAIL of a COMMAND QUEUE CHAIN by the 2080 PORT. When the COMMAND reaches the HEAD of the COMMAND QUEUE CHAIN it causes a block data transfer over the TTL IOBUS by a DMA mechanism.

   NOTE: The CI adapter is the only kind of port which supports data transfers initiated by the remote host.

There is only one RESPONSE QUEUE CHAIN and one FREE QUEUE CHAIN.

All queue entries are fixed in size to 12 2080 words and reside in the PHYSICAL ADDRESS space of the 2080 OPERATING SYSTEM.

The 2080 PORT accesses the COMMAND, RESPONSE and FREE QUEUE ENTRIES by executing read interlock and write interlock instructions. During normal operation the 2080 PORT is permitted to:

1. Remove COMMAND QUEUE ENTRIES from the HEAD of the four COMMAND QUEUE CHAINS.

2. Insert COMMAND QUEUE ENTRIES which originate from a remote HOST on the TAIL of one of the four COMMAND QUEUE CHAINS. (CI ports only)

3. Insert RESPONSE QUEUE ENTRIES on the TAIL of the RESPONSE QUEUE CHAIN.

4. Insert FREE QUEUE ENTRIES on the TAIL of the FREE QUEUE CHAIN.

5. Remove FREE QUEUE ENTRIES from the HEAD of the FREE QUEUE CHAIN.

10.5.2 PORT CONTROL BLOCK (PCB)

The PCB is a 19 word table residing in 2080 memory. The first word of the PCB supplies a pointer to the base of the BUFFER DESCRIPTOR TABLE. The remaining 18 words of the PCB are used to control the linking of the COMMAND, RESPONSE and FREE QUEUE entries.

The format of the PCB is illustrated by FIGURE 5.30. The illustration shows a COMMAND QUEUE CHAIN with 4 entries. The other QUEUE CHAINS, however, have an identical format.

The 2080 PORT accesses a QUEUE ENTRY by first doing a read interlock. Once the 2080 PORT has successfully INTERLOCKED the desired QUEUE CHAIN it either connects a new QUEUE ENTRY to the TAIL of the QUEUE CHAIN or removes a QUEUE ENTRY from the HEAD of the QUEUE CHAIN (whichever is applicable). The I/O PORT accomplishes this by appropriately modifying the FLINK and the BLINK fields of the PORT CONTROL BLOCK and the adjacent QUEUE ENTRY. The I/O PORT prepares new QUEUE ENTRIES for the TAIL of the QUEUE CHAIN in advance by modifying previously removed FREE QUEUE ENTRIES.

```
                            +--------+    +--------+
                 +----------! FLINK  !<---! FLINK  !<--------+
                 !       +->!        ! +->!        !         !
                 !       !  +--------+ !  +--------+         !
                 V       !  ! BLINK  !-+  ! BLINK  !--+      !
             +--------+  !  +--------+    +--------+  !  +--------+
        +--->! FLINK  !  !  !COMMAND !    !COMMAND !  +->! FLINK  !<---+
        ! +--!        !  !  ! QUEUE  !    ! QUEUE  !     !        !    !
        ! !  +--------+  !  !   3    !    !   2    !     +--------+    !
        ! !  ! BLINK  !--+  +--------+    +--------+     ! BLINK  !--+ !
        ! !  !        !                                  !        !  ! !
        ! !  +--------+                                  +--------+  ! !
        ! !  !COMMAND !                                  !COMMAND !  ! !
        ! !  ! QUEUE  !                                  ! QUEUE  !  ! !
        ! !  ! ENTRY  !       PORT CONTROL BLOCK         ! ENTRY  !  ! !
        ! !  !   4    !    +-----------------------+     !   1    !  ! !
        ! !  +--------+  0 ! ADDRESS    OF    BDT  !     +--------+  ! !
        ! !    TAIL        +-----------------------+        HEAD     ! !
        ! !              1 ! FREE   QUEUE   FLINK  !                 ! !
        ! !                +-----------------------+                 ! !
        ! !              2 ! FREE   QUEUE   BLINK  !                 ! !
        ! !                +-----------------------+                 ! !
        ! !              3 !     RESERVED          !                 ! !
        ! !                +-----------------------+<----------------+ !
        ! +----------->  4 ! COMMAND QUEUE 0 FLINK !                   !
        !                  +-----------------------+-------------------+
        +-------------   5 ! COMMAND QUEUE 0 BLINK !
                           +-----------------------+
                         6 !     RESERVED          !
                           +-----------------------+
                         7 ! COMMAND QUEUE 1 FLINK !
                           +-----------------------+
                         8 ! COMMAND QUEUE 1 BLINK !
                           +-----------------------+
                         9 !     RESERVED          !
                           +-----------------------+
                        10 ! COMMAND QUEUE 2 FLINK !
                           +-----------------------+
                        11 ! COMMAND QUEUE 2 BLINK !
                           +-----------------------+
                        12 !     RESERVED          !
                           +-----------------------+
                        13 ! COMMAND QUEUE 3 FLINK !
                           +-----------------------+
                        14 ! COMMAND QUEUE 3 BLINK !
                           +-----------------------+
                        15 !     RESERVED          !
                           +-----------------------+
                        16 ! RESPONSE QUEUE FLINK  !
                           +-----------------------+
                        17 ! RESPONSE QUEUE BLINK  !
                           +-----------------------+
                        18 !     RESERVED          !
                           +-----------------------+
```

WORD OFFSET	DESCRIPTION
0	Physical address of the base of the BUFFER DESCRIPTOR TABLE.
1	Physical address of the head FREE QUEUE entry (FREE QUEUE FLINK)
2	Physical address of the tail FREE QUEUE entry (FREE QUEUE BLINK)
3	RESERVED
4	Physical address of the head COMMAND QUEUE 0 entry (COMMAND QUEUE FLINK)
5	Physical address of the tail COMMAND QUEUE 0 entry (COMMAND QUEUE BLINK)
6	RESERVED
7	Physical address of the head COMMAND QUEUE 1 entry (COMMAND QUEUE FLINK)
8	Physical address of the tail COMMAND QUEUE 1 entry (COMMAND QUEUE BLINK)
9	RESERVED
10	Physical address of the head COMMAND QUEUE 2 entry (COMMAND QUEUE FLINK)
11	Physical address of the tail COMMAND QUEUE 2 entry (COMMAND QUEUE BLINK)
12	RESERVED
13	Physical address of the head COMMAND QUEUE 3 entry (COMMAND QUEUE FLINK)
14	Physical address of the tail COMMAND QUEUE 3 entry (COMMAND QUEUE BLINK)
15	RESERVED
16	Physical address of the head RESPONSE QUEUE entry (RESPONSE QUEUE FLINK)
17	Physical address of the tail RESPONSE QUEUE entry (RESPONSE QUEUE BLINK)
18	RESERVED

10.5.3 BUFFER DESCRIPTOR TABLE AND BUFFER DESCRIPTORS

The I/O PORT determines the correct physical address of a DATA BLOCK in 2080 memory, during data transfers to or from a named 2080 memory buffer area, by extracting a series of routing parameters from three separate tables. These tables reside in the physical memory address space of the 2080 operating system.

A buffer descriptor (BD) consists of a linked list of descriptors each of which describes a region of physical memory. Each individual descriptor has the following format:

word	contents
0	address of next descriptor
1	mode(3), count(15), byte offset(4)
2	address of base of data buffer
3	port available storage and final status

The head of the BD chain (i.e.; the BDT itself) is located in the BDT and is as follows:

word	contents
0	valid bit(1), buffer key(16), port
1	address of current node

Each BDT identifier given out by IOPS20 will be the offset from the start of the BDT table in the PCB. Therefore the address of the BDT header is:

BDT+2*n

where
BDT is the base address of the start of the BDTs and n is the descriptor specified by the data request. For n to be valid it must be less than the number of BDT entries specified in the PCB.

10.5.4 DATA MODES

The I/O PORT supports the following 3 DATA FORMAT modes during high-speed data transfers between the PLI and 2080 memory.

1) INDUSTRY COMPATIBLE
2) CORE DUMP
3) HIGH DENSITY
4) 7-BIT ANSI-ASCII

The byte mappings for these modes are as follows:

10.5.4.1 INDUSTRY COMPATIBLE - Figure 5.40 illustrates the INDUSTRY COMPATIBLE mode for mapping 8 bit PLI bytes into 36 bit 2080 words.

```
                                2080 WORD
   0          7  8         15 16         23 24         31 32   35
+------------+------------+------------+------------+-------+
!            !            !            !            !       !
!  byte=4n+1 !  byte=4n+2 !  byte=4n+3 !  byte=4n+4 ! ZERO  !
!            !            !            !            !       !
+------------+------------+------------+------------+-------+
 7          0  7         0  7         0  7         0
                                PLI BYTE
```

n = Number of complete 36 bit 2080 words processed since the start of byte transfers.

FIGURE 5.40

10.5.4.2 CORE DUMP - Figure 5.50 illustrates the CORE DUMP mode for mapping 8 bit PLI bytes into 36 bit 2080 words.

```
                                2080 WORD
   0          7  8         15 16         23 24         31 32   35
+------------+------------+------------+------------+-------+
!            !            !            !            ! Byte  !
!  Byte=5n+1 !  byte=5n+2 !  Byte=5n+3 !  byte=5n+4 !   =   !
!            !            !            !            ! 5n+5  !
+------------+------------+------------+------------+-------+
 7          0  7         0  7         0  7         0 3     0
                                PLI BYTE
```

n = Number of complete 36 bit 2080 words processed since the start of byte transfers.

NOTE: Bits 4 through 7 of every 5th byte are discarded.

FIGURE 5.50

10.5.4.3 HIGH DENSITY - Figure 5.60 illustrates the HIGH DENSITY mode for mapping 8 bit PLI bytes into 36 bit 2080 words.

```
                              2080 WORD PAIR

                            FIRST WORD OF PAIR
   0          7  8         15 16         23 24         31 32   35
+------------+------------+------------+------------+-------+
!            !            !            !            ! BYTE  !
!  byte=9n+1 !  Byte=9n+2 !  Byte=9n+3 !  Byte=9n+4 !   =   !
!            !            !            !            ! 9n+5  !
+------------+------------+------------+------------+-------+
 7          0  7         0  7         0  7         0 7     4
                                PLI BYTE
```

```
                       SECOND WORD OF PAIR
  0    3   4          11  12          19  20          27  28          35
+------+-----------+-----------+-----------+-----------+
! Byte !           !           !           !           !
!   =  ! Byte=9n+6 ! byte=9n+7 ! Byte=9n+8 ! byte=9n+9 !
! 9n+5 !           !           !           !           !
+------+-----------+-----------+-----------+-----------+
 3    0 7         0 7         0 7         0 7         0
                          PLI BYTE
```

n = Number of complete 36 bit 2080 word pairs processed since the start of byte transfers.

FIGURE 5.60

10.5.4.4 ASCII -

```
                         2080 WORD
  0          6   7        13  14        20  21        27  28        34  35
+----------+----------+----------+----------+----------+---+
!          !          !          !          !          !   !
! byte=5n+1! byte=5n+2! byte=5n+3! byte=5n+4! byte=5n+5! 0 !
!          !          !          !          !          !   !
+----------+----------+----------+----------+----------+---+
 7        0 7        0 7        0 7        0 7        0
                         PLI BYTE
```

n = Number of complete 36 bit 2080 words processed

FIGURE 5.70

10.6 I/O SUBSYSTEM RAMP

To be supplied

CHAPTER 11

MASSBUS ADPATOR

11.1 GOALS

The goals for the MBA Channel Interface is to provide the following:

1. A cost/effective MBA Mass Bus Adapter for 2080 that uses a common I/O PORT architecture with the 2080 ICCS adapter.

2. Availability at 2080 CPU breadboard power-on.

3. Control Unit independent design. The MBA will be general purpose without any Control Unit specific micro-code. The software driver will deal with the Control Unit at the Control Unit command level.

4. Support up to 8 selector sub-channels or Units, (8 RP06 or 8 TM03, or 8 DX20's).

11.2 GENERAL DESCRIPTION

The MBA Mass Bus Adapter is physically partitioned into two logically functional pieces; a single extended hex board I/O Port and a single extended hex board MASS Bus Interface. See figure 11.1. The interface between the two functional pieces is a simple, general purpose path called the Port Link Interface. The adapter also connects to the 2080 TTL IO bus.

Functionally, The MBA Mass Bus Adapter consists of the following:

I/O PORT FUNCTIONS
----

1. Data Mover - This moves data to a peripheral controller from primary memory (write operation) or moves data to primary memory from a peripheral device (read operation). The Data Mover uses a byte count and starting address as control parameters. It is set up by the software via a CHANNEL COMMAND PACKET.

2. Bit-Slice Microprocessor - This fetches and processes CHANNEL COMMAND PACKETS ,and creates CHANNEL RESPONSE PACKETS.

CHANNEL INTERFACE FUNCTIONS

7. Channel Interface - This contains MASS Bus drivers and receivers, and a micro-sequencer to control the MBA bus protocol. It also contains byte buffering.

A Channel Program, consisting of a series of linked Channel Command Packet is used to set up the Channel Data, Data Mover, and send and receive Command and Status bytes to and from MBA Control Units.

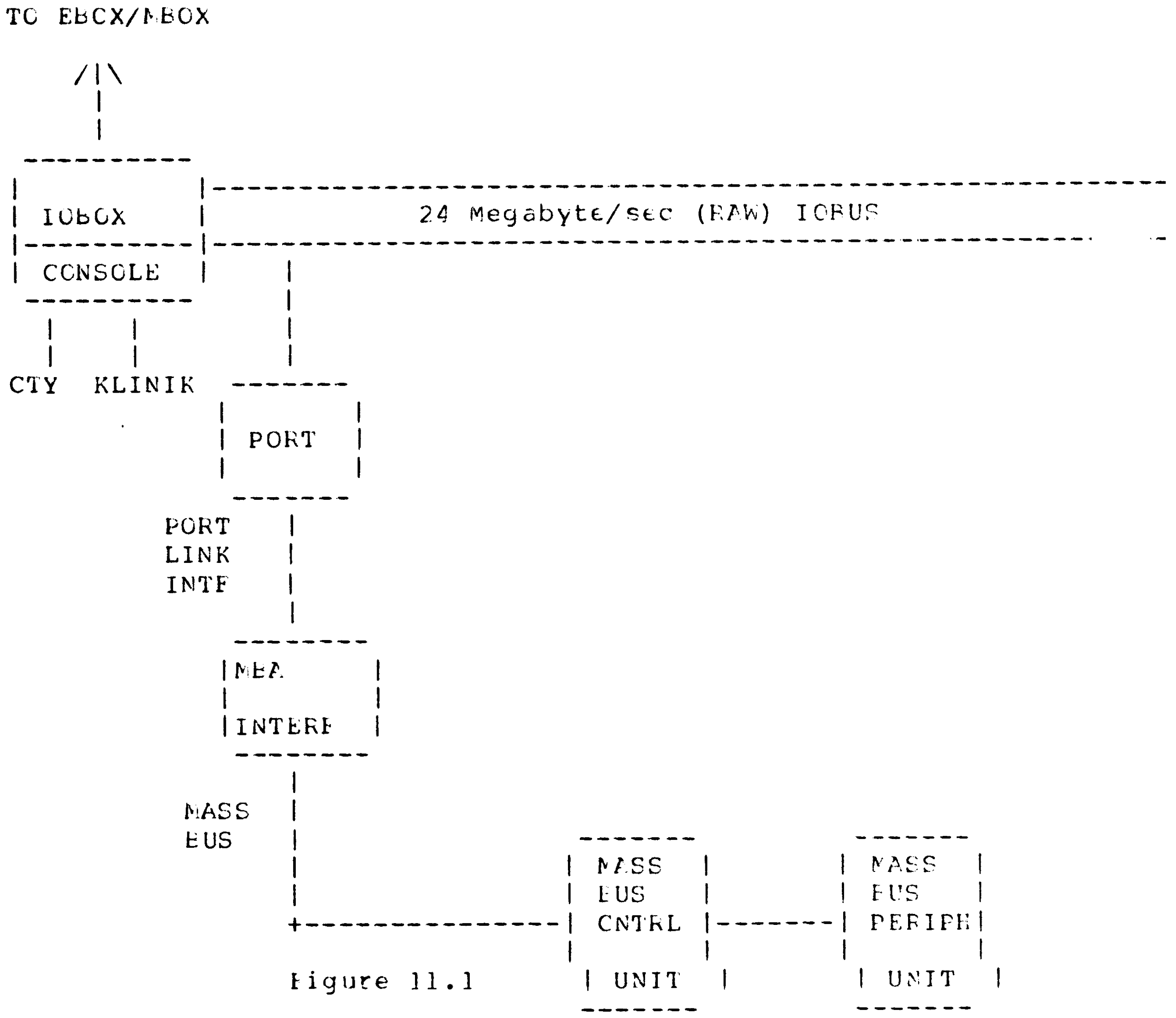

Figure 11.1

Single IO instructions can be used to talk to the MASS bus as in the traditional KL10 IO instructions. This mode of operation will be used for boot straps and initial monitor debug. In this mode of operation, software will clear the QUEUE VALID bit in the Port Control Status Register.

The second mode of operation uses a command queue. In this mode of operation,software will set the QUEUE VALID bit in the Port Control Status Register. Linkages within a command queue are created via FLINKS and BLINKS as described in the 2080 I/O Subsystem Specification (see chapter 10).

11.3 REGISTER ACCESS BLOCK

Register Access Block is an eight word block which provides an I/O port with the control and data needed to do register reads and writes. In addition, there is room in the eight word block for the port to store relevant status when it interrupts the cpu. There is one Register Access Block for each I/O port in the machine (total of eight).

Word	Name	Description
0	OPCODE/IOADR	Bit <0> :Error Bits <1:5> contain command. Format is: D1=0 nop D1=1 valid command D2=1 read D3=1 write D4=1 bitset D5=1 bitclr Bit <6> : Disable Register access error stop Bit <7> : Disable Transfer Error Stop Bit <8> : Response Enable Bits <25:27> : drive address Bits <30:35>: drive reg. address

1	DATA	For a register write, this word contains the data to be written. For a register read, this word will be written by the port. For a register bit modification, this word contains a mask. For a bitset command the mask will be logically ORed with the addressed register and the result will be in the addressed register. For a bitclr command, a logical AND of 1's complements of mask and the contents of the addressed register will be formed and the result will be stored in the addressed register.
2	OPCODE/IOADR	Same format as word 0
3	DATA	Same format as word 1

Words 4 through 7 are interrupt status words and are described in sec. 5.5 of 2080 functional specs.
OPCODE/IOADR Map:

```
 00  01  05  06   07   08   09  10    23 24   26  27   29  30   35
+-----------------------------------------------------------------+
|ERR |OPCODE|DIS |DIS |RESP|CHN |          |DEV SEL|       |REC SEL|
|    |      |RAEI|XES |ENA |BIT |          |CODE   |       |CODE   |
+-----------------------------------------------------------------+
```

DATA word map for read/write external register command:

```
 00-17   18      19    20                            35
+------------------------------------------------------+
|       |MASSBUS|MASSBUS|  EXTERNAL REGISTER DATA       |
|       |E P    |P      |                               |
|       |       |BIT    |                               |
+------------------------------------------------------+
```

Bit <18> - MASSBus Even parity

If this bit is set, the port will generate even parity for for the 16 control lines of the MASSBUS. otherwise odd parity is generated (even parity is provided for diagnostic purposes only. Normally this bit will be 0).

Bit <19> - parity bit sent by the drive.

Bits <20:35> - External Register Data

These bits contain data read from external register or the data to be written in an external register.

The Port will operate in two modes. In the Non Q Mode mode, all channel commands will be issued via the Register Access Block. In the QUEUED mode, the program will write in the MBA Status Register causing the port to process command packets from one of one of two Queues. Commands in the Queues have same format as in the Register Access Block.

11.4 COMMAND/RESPONSE PROCESSING

11.5 Q MODE

The SOFTWARE PORT DRIVER in the 2080 assembLes COMMANDS and stacks them as QUEUE ENTRIES on the tail of COMMAND QUEUE CHAIN.

The I/O PORT asynchronously removes the COMMAND QUEUE ENTRIES from the head of COMMAND QUEUE CHAIN, decodes the COMMAND and takes the appropriate action. This action may be entirely internal to the I/O PORT, or it may require that the I/O PORT transmit the COMMAND over the PLI to a MASSBUS adapter.

The I/O PORT then completes the process by modifying the appropriate status bits in the COMMAND and placing it either on the tail of a RESPONSE QUEUE CHAIN or the TAIL of a FREE QUEUE CHAIN, both of which reside in 2080 memory.

If the INTERRUPT ENABLE bit in the I/O PORT'S CONTROL STATUS REGISTER is set and if the RESPONSE QUEUE CHAIN is empty; the PORT will place the modified COMMAND on the RESPONSE QUEUE CHAIN. An interrupt will be generated to inform the PORT DRIVER of the QUEUE ENTRY.

In this mode of operation the RESPONSE QUEUE entries are used to send status information to the PORT DRIVER. If the modified COMMAND was returned to the RESPONSE QUEUE CHAIN the PORT DRIVER may examine the resulting RESPONSE QUEUE ENTRY to determine the status of its associated COMMAND. (I.E. was the COMMAND successfully executed?). The I/O PORT always returns the modified COMMAND QUEUE ENTRY to the TAIL of the RESPONSE QUEUE CHAIN if an error was encountered during execution of the COMMAND.

The FREE QUEUE CHAIN is simply a list of blank queue entries residing in 2080 memory that both the PORT DRIVER and the I/O PORT may use whenever they need free memory to generate a new COMMAND or RESPONSE QUEUE entry.

11.6 NON Q MODE

The software loads the Register Access Block and does a Ding instruction. The port then reads or writes the specified register.

11.7 CHANNEL COMMAND PACKET FORMAT

The Channel Command Packet (hereafter called CCP) is created by the CPU and placed on one of the command queues. A Command List consists of many CCP's linked together on a command queue via FLINKS and BLINKS. The format of the CCP is shown below.

```
                                                              Word
     +----------------------------------------------------+
     |                    FLINK                           |   0
     +----------------------------------------------------+
     |                    BLINK                           |   1
     +----------------------------------------------------+
     |                    OPCODE/IOADR                    |   2,4,6,8,
     |                                                    |   10,12,14
     +----------------------------------------------------+
     |                    DATA                            |   3,5,7,9
     |                                                    |   11,13,15
     +----------------------------------------------------+
```

WORD	BITS	NAME	DESCRIPTION
0	<00: 35>	FLINK	QUEUE FORWARD LINK. PHYSICAL ADDRESS of the first location of the successor QUEUE ENTRY.
1	<00:35>	BLINK	QUEUE BACKWARD LINK. PHYSICAL ADDRESS of the first location of the predecessor QUEUE ENTRY.
2,4,6,8, 10,12,14	<00:35>	OPCODE/IOADR	See sec. 10.4
3,5,7,9, 11,13,15	<00:35>	DATA	See sec. 10.4

11.8 Q MODE CHANNEL RESPONSE PACKET

The Channel Response Packet (hereafter called the CRP) is created by the channel and placed on the I/O Port Response Queue. The channel will write the CRP under the following conditions:

1. A CCP was executed successfully.

2. A channel error was detected during the execution of a CCP.

3. A Control Unit or device error was detected during the execution of a CCP.

The format of the CRP (if no error is detected) is shown below.

```
+-------------------------------------------------+
!                    FLINK                        !      0
+-------------------------------------------------+
!                    BLINK                        !      1
+-------------------------------------------------+
|                                                 |
|                  OPCODE/IOADR                   |      2,4,6,8,
|                                                 |      10,12,14
+-------------------------------------------------+
|                                                 |
|                    DATA                         |      3,5,7,9,
|                                                 |      11,13,15
+-------------------------------------------------+
```

WORD	BITS	NAME	DESCRIPTION
0	<00:35>	FLINK	QUEUE FORWARD LINK. PHYSICAL ADDRESS of the first location of the successor queue entry.
1	<00:35>	BLINK	QUEUE BACKWARD LINK. PHYSICAL ADDRESS of the first location of the predecessor queue entry.
4,6,8, 10,12,14	<00:35>	OPCODE/IOADR	This is a copy of the CCP OPCODE/IOADR word.
5,7,9, 11,13,15	<00:35>	DATA	For a read command, port writes this word. For all other commands, this is a copy of CCP data word.

If an error is detected in command processing, the port will set bit <0> of the first OPCODE/IOADR word and will read the first four registers of the drive and include them in the CRP. The format is as follows:

```
          +------------------------------+
          |        FLINK                 |      0
          +------------------------------+
          |        BLINK                 |      1
          +------------------------------+
          |        OPCODE/IOADR          |      2 ; Bit <0>=1
          +------------------------------+
          |        PORT STAUS REG        |      3
          +------------------------------+
          |        MBA STATUS REG        |      4
          +------------------------------+
          |        MASSBUS REG 0         |      5
          +------------------------------+
          |        MASSBUS REG 1         |      6
          +------------------------------+
          |        MASSBUS REG 2         |      7
          +------------------------------+
          |        MASSBUS REG 3         |      8
          +------------------------------+
          |        CURRENT BUFFER NAME   |      9
          +------------------------------+
          |        CURRENT WORD COUNT    |      10
          +------------------------------+
          |                              |
          |                              |      11 - 15
          |                              |
          +------------------------------+
```

11.9 BUFFER DESCRIPTORS

Buffer descriptors will not be implemented at FCS. A different version of port microcode will be loaded when Buffer Descriptors are implemented.

(See sec. 7 of 2080 Functional Specifications)

11.10 CHANNEL RESET AND LOGOUT AREA

Each of the eight data channels associated with a Massbus has a four word reset and logout area. 37 (octal) consecutive locations are assigned to Reset and Logout area.

Word	Contents
0	Initial channel command list pointer.
1	status word 1
2	status word 2
3	not used

At the end of data transfer, the port will load two status words in the logout area and notify the cpu of command completion by generating an interrupt.

The port will assign an internal register to hold the address of Reset and logout area. On powerup, this register will be initialize to a known value. Once the boot operation in complete, software will load the base address of the Logout area in this register. The port will compute the first location of the its own logout area by adding 4*Port Number to this base address.

```
        Status word 1

        0                       13 14                   35
        +------------------------+-----------------------+
        !      Status bits       !  Command List Pointer !
        +------------------------+-----------------------+
```

bit <0> this bit is always a one.

Bit <1>: Memory parity error

This bit is set if memory parity occurs whenever port references memory.

Bit <2>: not IO BUS Error:

This bit is reset if memory problems other than parity and nonexistent memory occurs.

Bit <3>: word count=0

This bit is set if the word count is zero when the status words are stored into memory.

Bit <4>: non existent memory

This bit is set if the Port references a non existent memory location.

Bits <5:7>: 0's

Bit <8:11>: (spare)

Bit <12>: Short Word Count Error

This bit is set if the Port has transferred all the data specified by the CCW and the device is still not done with the transfer.

Bit <13>: Overrun Error

This bit is set when

1. Device is inputing data and port buffer is full.

2. Device is demanding data and the port buffer is empty.

Bit <14:35>: Channel Command List Pointer

Points at current CCW+1.

2. Status Word 2

```
 00 01 02  03                 13  14                         35
+------------------------------------------------------------+
| OPCODE | CURRENT WORD COUNT | CURRENT DATA BUFFER ADDRESS |
+------------------------------------------------------------+
```

Contains the last CCW with up-to-date word count and address.

11.11 CHANNEL COMMAND WORD

A Channel Command Word (CCW) will use 3 bits for opcode, 22 bits to specify physical memory address to specify jump, halt addresses and the starting address of data transfer. A data transfer CCW will also use 10 bits to specify positive word count.

CCW Opcode

Code	Operation
000	Halt
001	Not used
010	Jump
011	Not used
100	Forward data transfer (do not halt)
101	Reverse data transfer (do not halt)
110	Forward data transfer (halt)
110	Reverse data transfer (halt)

11.12 CHANNEL COMMAND WORD FORMAT

11.12.1 Halt

```
00  01  02  03            13  14                                 35
+------------------------------------------------------------------+
| OPCODE     |                 | NEW CHANNEL COMMAND LIST POINTER  |
|0   0   0   |                 |                                   |
+------------------------------------------------------------------+
```

This CCW causes the channel operation to stop; i.e., the next sequential CCW in the command list is not fetched. This command also specifies a new channel command list pointer (bits <14:35>), thus providing a starting address for the next channel command list if the software driver does not reset the channel command list pointer at the start of next data transfer.

11.12.2 Jump

```
00  01  02  03            13  14                                 35
+------------------------------------------------------------------+
| OPCODE     |                 | NEXT CHANNEL COMMAND LIST POINTER |
|0   1   0   |                 |                                   |
+------------------------------------------------------------------+
```

This command loads a new address in the command list pointer, thus allowing branch during channel command list execution. For example, the software driver normally loads a Jump in location 0 of the channel's reset and logout area to point to the first location in the command list.

11.12.3 Data Transfer

```
00            03                  13  14                         35
+------------------------------------------------------------------+
| 1  |HALT|REV | POSITIVE WORD      | DATA BUFFER STARTING ADDRESS |
|    |XFR |XFR | COUNT              | (0 FOR SKIP OPERATION        |
+------------------------------------------------------------------+
```

Bit <01> = 1 : The port will halt after transferring the number of words specified in the positive count field.

Bit <01> = 0 : The port will fetch the next channel command from current channel command list pointer + 1.

Bit <02> = 1 : This allows read reverse operation on magtape systems.

bit <02> must be set to 0 for all write operations.

Positive Word Count (Bits <03:13>) : This field specifies the number of words to be transferred to/from memory.

Buffer Starting Address (Bits <14:35>) : If this field is 0 then the CCW does not move any data to/from memory. On reads, data coming from the Massbus is thrown away. On writes, the channel supplies block fill words (either all 0's or all 1's). On both reads and writes, positive word count is counted down each time a word is requested by the Massbus. After completing the data transfer, the port will either fetch next command (if bit <01>=0) from the current channel command list pointer + 1 or it will halt (if bit <01> = 1).

If the data buffer starting field is non zero then the positive word count is counted down each time a word is moved to/from memory. The data buffer address ,is upcounted (if bit <02> = 0> or downcounted (if bit <02> = 1) each time a word is moved to/from memory.

11.13 INTERRUPTS

Interrupts are generated by the I/O PORT under the following conditions:

1. Port initialization has been completed.

2. The PORT has inserted a CHANNEL RESPONSE PACKET on an empty RESPONSE QUEUE (QUEUED MODE).

3. The PORT has placed an entry on an empty FREE QUEUE (QUEUED MODE).

4. A FREE QUEUE entry was needed for processing and was not available (QUEUED MODE).

5. A memory error was detected on a 2080 memory reference.

6. An internal PORT hardware error occurred.

7. The port has completed data transfer command (Non Q mode)

8. Massbus ATTENTION line is asserted by a drive:

   In the Non Q Mode Mode, interrupt will be generated if Attention Interrupt Enable (bit < 30> of Status Fegister) is set.

   In the QUEUED mode, an entry will be placed in the REsponse Queue.

Refer to the 2080 I/O Subsystem Specification and the 2080 I/O Port Specification for further details.

11.14 PORT CONTROL STATUS REGISTER

Refer to the 2080 I/O Port Specification

11.15 REGISTERS ACCESSIBLE BY SOFTWARE

11.15.1 EXTERNAL REGISTERS

See DEC STD 159 for detailed description.

11.16 DESCRIPTION OF INTERNAL REGISTERS

11.16.1 Logout Address Register

This register contains the pointer to the word 0 of the channel logout area. Port will initialize this register to a known value on powerup. Software will load this register at the completion of boot operation. Port will compute the location of its logout area by adding 4*PORT Number to the contents of this register.

11.16.2 Port Control Block Address Register

This register contains the physical address of the 19 word port control block. The first word of the PCB supplies a pointer to the base of the buffer descriptor. The remaining 18 words of the PCB are used to control the linking of Command, Response and Free queue entries .

11.16.3 Transfer Control Register

This register contain the length (in 2's complement) of the data transfer (number of sectors or record) and the function code to specify a read/write operation .

11.16.4 Interrupt Vector Index Register

Bits <27:35> are specified by software as a relative address within EPT.

11.16.5 Read Register

```
+-----------------------------------------+
|                 |PARITY| DIAG. WRITE DATA |
+-----------------------------------------+
```

This register is used mainly by the diagnostic programmer. During a diagnostic write operation, data sent out by the controller (on the MASSbus Data Bus) can be made to loop back to the Read Register and can be read in for data verification.

11.16.6 Write Register

The register shall be loaded only by the diagnostic programmer. Contents of the Write register are used to simulate data inputs to the Controller (on the MASSbus Data Bus) during a diagnostic read operation. Bits <18:35> are data bits and bit <17> shall be the computed parity bit (by the programmer) for bits <18:35>.

11.16.7 Diagnostic Control Register

Most MASSbus operations can be simulated by the bits in this register.

Clock (bit 35)

This bit is used to simulate the MASSbus Syn Clock signal. The trailing edge of this bit will strobe in or send out data from/to the MASSbus Data Bus. The leading and trailing edges of this signal are generated by simply setting and then resetting bit 35.

Attention (bit 34)

This bit is used to simulate the MASSbus Attention signal. Interrupt will be generated when this bit is set provide that the Attention Interrupt Enable bit (MBA Status Reg-bit <30>) has been set.

Bit 33

Not used.

EBL (bit 32)

This bit will simulate the MASSbus End of Block signal. The Block Counter will be incremented by the leading edge of this bit.

Exception (bit 31)

This bit will simulate the MASSbus Exception signal. The setting of this signal along with EBL (bit 32) set will terminate the current Command provided that "Disable Transfer Error Stop" is not set.

Read/Write (bit 30)

This bit, when set, will enable output data (on the MASSbus Data Bus) to be looped back to become input data to enable testing of the data path and the drivers/receivers. This bit shall be set to "1" only for a read simulation and shall be cleared for a write simulation.

Even Parity Check (bit 29)

This bit, when set, will enable the Controller to check for even parity, instead of odd parity, on the MASSbus Control Bus. The parity network can be checked by this means.

Block Address Register Test (bit 28)

This bit, when set, will inhibit the Controller from sending out the content of the Secondary Transfer Control Register to a drive. Instead, the Secondary Block Address Register is sent out twice and can be checked by the diagnostic program.

Control Bus Test (bit 27)

This bit, when set, will loopback the MASSbus Control Bus to enable testing of the drives/receivers and related logic.

NOTE

The diagnostic software must select a drive number which is not present or is not powered up. This may not be possible in a configuration with 8 dual port drives without software intervention in both systems.

Transfer (bit 26)

This bit, when set, will simulate the MASSbus transfer signal to be generated whenever a drive register is being accessed.

11.16.8 MBA STATUS REGISTER

Bit <35> - Clear Massbus Controller

This bit, when set by the program, will initialize the Port and the Massbus drives to their powerup state.

This bit will be cleared by software.

Bit <33:34>

Not used.

Bit <32> - Command Done

This bit is set when the Command is terminated with or without transfer error. An interrupt will be generated. Transfer error conditions that will terminate a transfer command and will inhibit the execution of the next transfer command are:

1. Data Parity Error (bit <18>) or Drive Exception (bit <19>) is set and the "Disable Transfer Error Stop" bit in the channel command, which started current data transfer, is not set.

2. Long Word Count Error (bit <20>).

3. Short Word Count Error (bit <21>).

4. Channel Error (bit <22>).

5. Drive Response Error (bit <23>).

6. Data Overrun (bit <25>)

This bit will be cleared by software.

bit <31> - Stop Transfer

This bit , when set by the program , will terminate the current transfer. None of the status bits will be cleared and can be examined by reading the MBA status register.

This bit will be cleared by software.

Bit <30> - MASSBUS Attention Enable

This bit, when set by software, will enable the MASSbus "Attention" signal to generate an interrupt.

Bit <29> - Reset Command List Pointer

This bit , when set by the program, will cause the port to copy the contents of the Command List Address Register into the Secondary Command List Address Register. This bit should be set when there is no transfer in progress.

This bit will be cleared by software.

Bit <28> - MASSBUS Attention

This bit is set by any drive on the MASSbus whenever attention condition (seek complete etc.) occurs in the drive. The bit will generate an interrupt if Interrupt Enable in the Port Control Status Register (Bit <33>) is set.

MASSBUS Attention will be set even when MASSBUS Attention Interrupt Enable is not set; it just won't interrupt until MASSBUS Attention Enable becomes set. The MASSBUS Attention bit will be reset only after all the attention conditions in all drives are cleared. The program must select the appropriate drives and write the appropriate External (drive) registers.

Bit <27> - MASSbus Enable

This bit must be set by by the software driver to enable the Controller to interface to any MASSbus drive.

In systems where more than one Controllers are connected to the same MASSbus, the software shall make sure that only one Controller at any time is enabled (bit <27> =1). This feature is mainly intended for failsoft reconfiguration in single and multi-processing systems. The dual port option should be used where concurrent transfers are desired.

The other Controller must be disconnected from the MASSbus with bit <27> = Ø, else but errors will occur because of improper bus termination.

Bit <26> - Data Overrun

This bit is set when:

1. A MASSbus drive is inputing data faster than the Port can transmit to memory.

2. A Massbus drive is demanding data faster than the Port can access data from memory.

This bit is cleared by software.

Bit <25> - Channel Not Ready

This status bit is controlled solely by the Port and can not be set or reset by the program. The Port sets this bit either when the system is initialized or after the Port has completed the previous transfer and has encountered a Halt command Word or Halt Bit in a command word and is ready to start a new transfer. The Port resets this bit when it is either actively doing data transfer or is in the process of finishing up the current data transfer operation (house-cleaning) and is not ready for the next data transfer operation.

Bit <25> - Clear MASSbus Controller

This bit, when set by the program, will initialize the Controller and the MASSbus drives to their powerup states.

This bit will be cleared by software.

Bit <24> - Register Access Error

This bit is set by the port when a drive register is accessed and one or more of the following error conditions occur:

1. Control Bus Time Out - This error condition occurs when a nonexistent drive is selected. If the Attention Summary Register is selected, the controller will expect a Control Bus Time Out and, therefore, will not set the error bit.

2. Control Bus Parity Error - This error condition occurs when a drive register is read and the Controller detects a parity error on the MASSbus Control Bus. If the Attention Summary Register is read, the Controller will inhibit parity checking and this error will not be set.

   When this error occurs, the port will re-try to access the the device register. If the error persists even after a pre-determined number of re-tries, this bit will be set.

The Controller will not allow further writing of any register if "Register Access Error" is set and the instruction that caused the error did not specify "Disable Register Access Error Stop".

This bit will be cleared by software.

Bit <23> - Drive Response Error

This bit is set by the port when a drive does not response with "Transfer" when the controller is loading a Read/Write Command into a drive. Drive Response Error will terminate the current transfer and Command Done will be set.

This bit will be cleared by software.

Bit <22> - Channel Error

This Status bit is set when the Port, during a data transfer operation, detects an error condition that will affect the validity of the data transfer. The Port upon sensing this bit, will terminate the current data transfer and Command Done will be set.

This bit will be cleared by software.

Bit <21> - Short Word Count

This bit will be set by the port if it has transferred all the data specified by the channel command word but the drive is still not done with the transfer. The port will terminate the current data transfer and will set Command Done and transfer error.

This bit is cleared by software.

Bit <20> - Long Word Count

This bit will be set if the port has transferred the specified number of sector (specified in the Transfer Control Register) but the device has not reached the final word count specified in the channel command word. The port will terminate the current data transfer and will set Command Done and transfer error.

This bit is cleared by software.

Bit <19> - Drive Exception

This bit is sent to the port by the drive specified in the current read/write command when a transfer error condition (data parity, etc.) is detected in the drive. The Controller will store the bit and will terminate the current transfer. Command Done and Transfer Error will be set. Drive Exception will not cause the Read/Write Command to be terminated if "Disable Transfer Error Stop" is specified.

This bit is cleared by software.

Bit <18> - Data Parity Error

This bit is set when the port detects a parity error on the MASSbus Data Bus during a Read and Write Command. In the case of a write command, the drive involved may also detect a parity error. The port will terminate the Read/Write Command and will set Command Done and Transfer Error. Data Parity Error will not terminate the Read/Write Command if "Disable Transfer Error Stop" is specified (see sec. 4.6).

This bit is cleared by software.

Bits 0 - 17

Not used.

11.17 MASSBUS PORT INTERFACE TO CPU

Control information is passed between the port and EBOX via mailbox (see sec 5.5 OF 2080 FUNCTIONAL SPEC.).

11.18 MASSBUS SIGNALS

Refer to DEC STANDARD 159 for a detailed description of Massbus signal.

CHAPTER 12

IBM COMPATIBLE BLOCK MUX CHANNEL

12.1 GOALS

The goals for the IBM Channel Interface is to provide the following:

1. A cost/effective IBM Block Multiplexer Channel for 2080 that uses a common I/O PORT architecture with the 2080 ICCS adapter and the KL10 ICCS adapter.
2. Availability at 2080 CPU breadboard power-on.
3. Control Unit independent design. The channel will be general purpose without any Control Unit specific micro-code. The software driver will deal with the Control Unit at the Control Unit command level.
4. Support up to 4 selector sub-channels.

12.2 GENERAL DESCRIPTION

The IBM Block Multiplexer Channel is physically partitioned into two logically functional pieces; a single extended hex board I/O Port and a single extended hex board Channel Bus Interface. See figure 7.1. The interface between the two functional pieces is a simple, general purpose path called the Port Link Interface.

Functionally, The IBM Block Multiplexer Channel consists of the following:

I/O PORT FUNCTIONS

1. Data Formatter - This packs and unpacks bytes in different formats. Format selection is specified in a BUFFER DESCRIPTOR that is pointed to by a CHANNEL COMMAND PACKET.

2. Data Mover - This moves data to a peripheral controller from primary memory (write operation) or moves data to primary memory from a peripheral device (read operation). The Data Mover uses a byte count and starting address as control parameters. It is set up by the software via a BUFFER DESCRIPTOR that is pointed to by a CHANNEL COMMAND PACKET.

3. Bit-Slice Microprocessor - This fetches and processes CHANNEL COMMAND PACKETS ,BUFFER DESCRIPTORS and creates CHANNEL RESPONSE PACKETS.

CHANNEL INTERFACE FUNCTIONS

4. Channel Interface - This contains Channel Bus drivers and receivers, and a micro-sequencer to control the IBM bus protocol. It also contains byte buffering.

A Channel Program, consisting of a series of linked Channel Command Packets is used to set up the Channel Data Formatter, Data Mover, and send and receive Command and Status bytes to and from IBM Control Units.

```
     TO EBOX/MBOX

        /|\
         |
         |
    ----------
   |          |-------------------------------------------------
   | IOBOX    |            24 Megabyte/sec (RAW) IOBUS
   |----------|-------------------------------------------------
   | CONSOLE  |      |
    ----------       |
     |     |         |
     |     |         |
    CTY  KLINIK  -------
                |       |
                | PORT  |
                |       |
                 -------
           PORT     |
           LINK     |
           INTF     |
                    |
                 ------
                |IBM   |
                |      |
                |INTERF|
                 ------
                    |
            IBM     |
            CHAN    |              -------         -------
            BUS     |             | IBM   |       | IBM   |
                    |             |       |       |       |
                    +-------------| CNTRL |-------| PERIPH|
                                  |       |       |       |
                    Figure 7.1    | UNIT  |       | UNIT  |
                                   -------         -------
```

Each one of the four sub-channels can control one IBM Control Unit at a time (i.e.; one data transfer per sub-channel). Therefore, four data transfers on four different IBM Control Units can be active simultaneously. Each of the sub- channels has a Channel Command List associated with it. These command lists will reside on the four I/O Port Command Queues (one command list per Command Queue). Linkages within a Command List are created via FLINKS and BLINKS as described in the 2080 I/O Subsystem Specification section 7.0 .

The four Command Lists have equal priority. The I/O Port will therefore interleave all active Command Lists. For example, if there are three active Command Lists, the I/O Port will execute a Channel Command Packet from Command Queue 0, then execute a Channel Command Packet from Command Queue 1, then execute a Channel Command Packet from Command Queue 2, and then execute a Channel Command Packet from Command Queue 0 again. The multiplexing point (i.e.; the point at which the I/O Port will switch from one command list to another) will be at the time when the current Channel/Control Unit signal exchange is completed. This is defined by the "Channel End" status byte.

Within a particular Command List, the next Channel Command Packet will be executed when the current Channel Command Packet has "expired". This is defined by the "Device End" status byte. When the I/O Port switches from one Command List to another (Channel End seen by the channel), relevant state information on the Channel Command Packet currently being executed will be saved by the I/O Port in a RAM File.

There may be circumstances when the software may wish to disable the multiplexing of Channel Command Packet processing (i.e.; there is a real time processing requirement for a whole Command List or part of a Command List). To facilitate this requirement, a Command Chain Bit will be present in each Channel Command Packet. If this bit is set then the next Channel Command Packet will be fetched from the same Command List as the one currently being executed. An example of this is the Search Header, Data Transfer sequence used for disc subsystems. After a Search Header, a Data Transfer command must be issued within a fixed time to prevent an additional rotational delay.

12.3 COMMAND/RESPONSE PROCESSING

The SOFTWARE PORT DRIVER in the 2080 assembles COMMANDS and stacks them as QUEUE ENTRIES on the tail of one of four COMMAND QUEUE CHAINS.

The I/O PORT asynchronously removes the COMMAND QUEUE ENTRIES from the head of COMMAND QUEUE CHAINS, decodes the COMMAND and takes the appropriate action. This action may be entirely internal to the I/O PORT, or it may require that the I/O PORT transmit the COMMAND over the PLI to a remote node on the CI BUS or to an IBM CONTROL UNIT on the IBM CHANNEL BUS.

The I/O PORT then completes the process by modifying the appropriate status bits in the COMMAND and placing it either on the tail of a RESPONSE QUEUE CHAIN or the TAIL of a FREE QUEUE CHAIN, both of which reside in 2080 memory.

If the I/O PORT places the modified COMMAND on the RESPONSE QUEUE CHAIN; if the RESPONSE QUEUE CHAIN is empty; and if the INTERRUPT ENABLE bit in the I/O PORT'S CONFIGURATION REGISTER is set; the I/O PORT will generate an INTERRUPT to inform the PORT DRIVER of the QUEUE ENTRY.

In this mode of operation the RESPONSE QUEUE entries are used to send status information to the PORT DRIVER. If the modified COMMAND was returned to the RESPONSE QUEUE CHAIN the PORT DRIVER may examine the resulting RESPONSE QUEUE ENTRY to determine the status of its associated COMMAND. (I.E. was the COMMAND successfully executed?). The I/O PORT always returns the modified COMMAND QUEUE ENTRY to the TAIL of the RESPONSE QUEUE CHAIN if an error was encountered during execution of the COMMAND.

The FREE QUEUE CHAIN is simply a list of blank queue entries residing in 2080 memory that both the PORT DRIVER and the I/O PORT may use whenever they need free memory to generate a new COMMAND or RESPONSE QUEUE entry.

12.4 CHANNEL COMMAND PACKET FORMAT

NOTE

> The 2080 port processor is a 32-bit computer. Bits 00 to 03 are ignored in all Channel Command Packets. Of course, the port will process all 36-bits on a data transfer.

The format of Channel Command Packets is the same as that used for ICCS Command Packets used on the 2080 ICCS Adapter and the KL10 ICCS Adapter.

The Channel Command Packet (hereafter called CCP) is created by the CPU and placed on one of four command queues. A Command List consists of many CCP's linked together on a command queue via FLINKS and BLINKS. The format of the CCP is shown below.

```
 4                                   35    WORD
+-------------------------------------+
!                FLINK                !     1
+-------------------------------------+
!                BLINK                !     2
+-------------------------------------+
! CPC    !  ZERO ! FLAGS !MASK-STATUS!     3
+-------------------------------------!
!         ZERO            ! DEV ADDR  !     4
+-------------------------------------+
!         COMMAND SPECIFIC            !     5 TO 16
!                                     !
+-------------------------------------+
```

WORD	BITS	NAME	DESCRIPTION
1	<04:35>	FLINK	QUEUE FORWARD LINK. PHYSICAL ADDRESS of the first location of the successor QUEUE ENTRY.

2	<Ø4:35>	BLINK	QUEUE BACKWARD LINK. PHYSICAL ADDRESS of the first location of the predecessor QUEUE ENTRY.
3	<Ø4:11>	OPC	OPERATION CODES defined as follows:

OPC (binary)	COMMAND
MMMM Ø1ØØ	SENSE
XXXX 11ØØ	SPARE
MMMM 11Ø1	READ-REVERSE
MMMM MMØ1	WRITE
MMMM MM1Ø	READ-FORWARD
MMMM MM11	CONTROL

NOTE: M indicates modifier
X indicates don't care

	<12:19>	ZERO	Must be zero.
	<2Ø:27>	FLAGS	Flags defined as follows:
	<2Ø>	CHAIN	If the CHAIN bit is set, than the next CCP will be fetched from the same command list as the CCP which just expired (i.e.; command list multiplexing will not occur).
	<21>	SUPPRESS LEN	If the SUPPRESS LENGTH bit is set, than length errors will be ignored.
	<22>	SKIP	If the SKIP bit is set in a READ CCP i data transfer to main storage will be suppressed. If the SKIP bit is set in a WRITE CCP, than zeros will be written instead of data from main storage.
	<23>	SENSE	If this bit is set, than a SENSE operation will occur if this CCP generates a CRF.

	<24>	RESPONSE ENA	If RESPONSE ENABLE is set than error free execution of this CCP will result in a CHANNEL RESPONSE PACKET being generated by the channel and then placed on the RESPONSE QUEUE. Execution of a CCP that results in an error always generates a response by the channel.
	<25>	REPEAT	If this bit is set, than the current CCP will be repeated if the DEVICE END status equals the MASK-STATUS. Each time the CCP is repeated the REPEAT COUNT is decremented. Repeated execution will cease and the next CCP will be fetched if the DEVICE END status does NOT equal the MASK-STATUS OR the repeat count is decremented to zero. This bit is used with CONTROL CCP,s only (will be ignored by all other CCP,s).
	<26>	IMMEAD-MODE	The IMMEDIATE MODE bit is used with CONTROL CCP's. If this bit is set than the channel will find the control bytes in the CCP. Otherwise, a BUFFER DESCRIPTOR will be used to find the control bytes in main storage.
	<27>	HALT	If the HALT bit is set and a device error or channel error occurs than a CRP will be written and execution of this Command List will halt.
	<28:35>	MASK-STATUS	Used with the REPEAT bit as described above.
4	<04:24>	N/A	Not used.
	<25:35>	DEV ADR	Address of selected device. The two most significant bits are the sub-channel number.
5->16	<04:35>		Command specific. See below.

COMMAND SPECIFIC WORDS 5 TO 16

OPC	WORD	BITS		DESCRIPTION
READ-REVERSE READ-FORWARD WRITE				
	5	<04:35>	LENGTH	Length of data transfer in bytes.
	6	<04:35>	NAME	This 32 bit word points to a buffer descriptor as follows:
		<04:19>	KEY	The KEY is a unique value stored in the buffer descriptor. KEYS must match, else ACCESS ERROR.
		<20:35>	INDEX	When added to the physical address of the base of the BUFFER DESCRIPTOR the resulting address will point to the first location of the BUFFER DESCRIPTOR HEADER.
SENSE	5	<04:35>	LENGTH	The number of sense bytes to be transfered from the Control Unit. The sense bytes will appear in a CHANNEL RESPONSE WORD.
	6->16	<04:39>	N/A	Not used.
CONTROL	5	<04:35>	LENGTH	Number of bytes required to be transfered for this command.

6	<04:35>	NAME	This 32 bit word points to a BUFFER DESCRIPTOR as in the READ, WRITE, READ-REVERSE CCP's. The NAME will not be used if the IMMEAD-MODE flag is on.
7	<04:35>	COUNT	This REPEAT COUNT is used in conjunction with the REPEAT flag. The count will be decremented each time a CCP is repeated.
8->10	<04:35>	COMMAND	Command bytes.
11->16	<04:35>	N/A	Not used.

12.5 CHANNEL RESPONSE PACKET

The format of the Channel Response Packet is the same format used for the ICCS Response Packet used on the 2080 ICCS adapter and the KL10 ICCS adapter. See the 2080 I/O Port Specification for further details.

The Channel Response Packet (hereafter called the CRP) is created by the channel and placed on the I/O Port Response Queue. The channel will write the CRP under the following conditions:

1. A CCP was executed with the RESPONSE ENA bit set in the flag field.

2. A channel error was detected during the execution of a CCP.

3. A Control Unit or device error was detected during the execution of a CCP.

The format of the CRP is shown below.

Bits marked with an "*" are conditions that will force a CRP to be written.

```
 04                           35
+-----------------------------------+
!        FLINK                      !   1
+-----------------------------------+
!        BLINK                      !   2
+-----------------------------------+
! CPC ! DEV-STATUS!FLAGS!MASK-STATUS!   3
+-----------------------------------!
```

```
!        ZERO             ! DEV ADDR  !    4
+-----------------------------------+
!        COMMAND SPECIFIC            !    5 TO 16
!                                    !
+-----------------------------------+
```

WORD	BITS	NAME	DESCRIPTION
1	<04:35>	FLINK	QUEUE FORWARD LINK. PHYSICAL ADDRESS of the first location of the successor queue entry.
2	<04:35>	BLINK	QUEUE BACKWARD LINK. PHYSICAL ADDRESS of the first location of the predecessor queue entry.
3	<04:11>	OPC	The OPC of the CRP is the same as the OPC of the CCP that generated the response.
	<12:19>	DEV-STATUS	The DEVICE STATUS bits describe the status of the selected device at the time the CCP that generated the response expired. The DEV-STATUS bits have the following meaning:
	<12> *	ATTENTION	Generated when a manual operation is initiated through such a device as a TTY.
	<13>	STAT-MOD	STATUS MODIFIER indicates that the I/O unit cannot execute the command.
	<14>	CU-END	CONTROL UNIT END indicates that the control unit is now free.
	<15> *	BUSY	BUSY indicates that the device is busy.

<16>		CHAN-END	CHANNEL END is always generated when the I/O control unit no longer needs the channel and is able to complete the I/O operation on its own.
<17>		DEV-END	DEVICE END is always generated when the device reaches its mechanical ending point or when the device is switched from the ready to not-ready status.
<18>	*	UNIT-CHK	UNIT CHECK is usually generated when the device has detected a a machine malfunction.
<19>	*	UNIT-EXCEP	UNIT EXCEPTION is generated when the I/O device detects a condition that does not normally occur (i.e., tape EOT).
<20:27>		FLAGS	FLAGS are the same as the flags of the CCP that generated the response.
<28:35>		CHAN-STATUS	The CHAN-STATUS bits have the following meaning:
<28>		SUCC	SUCCESS indicates successful completion of the CCP that generated the response.
<29>	*	ILLEGAL-CMD	ILLEGAL COMMAND indicates that the CCP that generated the response contained an illegal OPC.
<30>	*	STATUS ERR	STATUS ERROR indicates that an error occurred during a SENSE operation or during a status in operation.

	<31>	*	CHAN BUS ERR	Channel bus parity error or time out.
	<32>	*	BYTE COUNT LENGTH ERR	The device and channel did not agree regarding the number of bytes transfered.
	<33>	*	ACCESS ERR	This indicates that a buffer access violation occurred (key mismatch).
	<34>	*	OVERRUN	Indicates that a data overrun occurred.
	<35>	*	SEL ERROR	Indicates that the selected device did not respond.
4	<04:24>		ZERO	This field is zero
	<25:35>		DEV ADR	DEVICE ADR is the same DEVICE ADR of the CCP that generated the response.

COMMAND SPECIFIC WORDS 5 TO 16

OPC	WORD	BITS	NAME	DESCRIPTION
READ-REVERSE READ-FORWARD				
WRITE	5	<04:35>	LENGTH	Same as the LENGTH in the CCP that generated this response.
	6	<04:35>	NAME	Same as the CCP that generated the response.
	7->10	<04:35>	N/A	Not used.
	11->16	<04:35>	SENSE	SENSE bytes (if SENSE flag was set in CCP).

SENSE	5	<04:35>	LENGTH	The residual byte count after expiration of the CCP that generated the response.
	6	<04:35>	N/A	Not used.
	7->10	<04:35>	N/A	Not used.
	11->16	<04:35>	SENSE	Sense bytes received from device.
CONTROL	5	<04:35>	LENGTH	The residual byte count after expiration of the CCP that generated the response.
	6	<04:35>	COUNT	The residual REPEAT COUNT after expiration of the CCP that generated the response.
	7	<04:35>	NAME	NAME in the CCP that generated the response.
	8->10	<04:35>	COMMAND	COMMAND bytes specified in the CCP that generated the response.
	11->16	<04:35>	SENSE	SENSE bytes if SENSE flag was specified in CCP that generated the response.

12.6 BUFFER DESCRIPTORS

The format of BUFFER DESCRIPTORS is the same as that used for the 2080 ICCS Adapter and the KL10 ICCS Adapter.

Named memory buffers in the 2080's physical address space are described by a linked chain of BUFFER DESCRIPTORS. Each BUFFER DESCRIPTOR describes a buffer or a segment of a buffer. The BUFFER DESCRIPTORS are assembled and inserted into a BUFFER DESCRIPTOR TABLE by the 2080 I/O PORT DRIVER. The BUFFER DESCRIPTOR TABLE also resides in the 2080's physical address space.

A buffer name is a 32-bit quantity which is passed between PORTS via the NAME fields of the CHANNEL COMMAND PACKETS that transfer data. The 2080 I/O port uses the NAME fields to find the the BUFFER DESCRIPTORS. Buffer Descriptors inform the port what portion of memory to transfer to or from during a data transfer.

The format of a buffer name is illustrated below:

```
<----------------------->2080 WORD<------------->
 4                      19 20                   35
+-----------------------+-----------------------+
!          KEY          !          INDEX        !
+-----------------------+-----------------------+
```

1. KEY - The KEY is an additional mechanism to reduce the probability of an incorrect access to a buffer. The Key is generated by the port driver and must be the same for all BUFFER DESCRIPTORS that make up a buffer to be transfered. Name fields of all commands must have a Key that matches the key of the associated Buffer Descriptor. of the corresponding BUFFER DESCRIPTOR. The KEYS must match. A mis-match is flagged as an access error. The Keys of different data buffers must not be the same to prevent the port from linking them. The end of a buffer is flaged by a change in key.

2. INDEX - The INDEX field supplies a pointer to the first BUFFER DESCRIPTOR of the named buffer. When the INDEX field is added to the BUFFER DESCRIPTOR Base Register the resulting value will be the physical address of the first BUFFER DESCRIPTOR of the described buffer. To open a named buffer the 2080 PORT DRIVER fills in the appropriate fields and sets the VALID (V) bit of the BUFFER DESCRIPTORS. At this point the BUFFER DESCRIPTORS are under control of the 2080 port. The 2080 PORT DRIVER may not further modify the BUFFER DESCRIPTORS until they have been closed.

The 2080 PORT DRIVER may modify all BUFFER DESCRIPTORS associated with the BUFFER after the I/O Port has cleared the "V" bit.

```
                    BUFFER DESCRIPTOR HEADER

<--------------------2080 WORD------------------------->
                                                                  WORD
+-----------------------------------------------------+
|04<---------------->19|20|21|22<------------------->35|            1
|        KEY            |V |E |          RESERVED       |
|-----------------------------------------------------+
|04<---------->13|14<------------------------------->35|            2
|        RESERVED  |      PHYS ADR OF FIRST BUFF SEC DESC |
+-----------------------------------------------------+
|                    USED FOR ICCS                     |            3
+-----------------------------------------------------+
|                    USED FOR ICCS                     |            4
+-----------------------------------------------------+
|                    LENGTH                            |            5
+-----------------------------------------------------+


                    BUFFER SEGMENT DESCRIPTOR

+-----------------------------------------------------+
|04<->06|07|08<->13|14<--------------------------->35  |            1
| MODE  |L | RSVD  |PHYS BASE ADR OF BUFF SEGMENT      |
+-----------------------------------------------------+
|04<---------->13|14<----------------------------->35  |            2
|    RESERVED      |PHYS ADDR OF NEXT BUFF SEC DES     |
+-----------------------------------------------------+
|                    USED FOR ICCS                     |            3
+-----------------------------------------------------+
|                    SEGMENT LENGTH                    |            4
+-----------------------------------------------------+
|        SEGMENT BYTES TRANSFERED                      |            5
+-----------------------------------------------------+
```

BUFFER DESCRIPTOR HEADER

WORD	BITS	NAME	DESCRIPTION
1	<04:19>	KEY	Buffer Key. All named references to a Buffer must supply the correct KEY.
	<20>	V	Valid bit. If "V" is set, the buffer defined by this BUFFER DESCRIPTOR is opened and the remaining fields must contain valid information.
	<21>	E	ERROR. When set indicates that an access violation or length error occurred during the reading or writing of the buffer.
2	<14:35>	BSADR	Physical address of the first BUFFER SEGMENT DESCRIPTOR which describes contiguous region of this buffer.
3	<04:35>	N/A	Used by ICCS.
4	<04:35>	N/A	Used by ICCS.
5	<04:35>	LENGTH	The length of the buffer in bytes.

BUFFER SEGMENT DESCRIPTOR

WORD	BITS	NAME	DESCRIPTION
1	<04:06>	MODE	Mode Field. Defines byte packing mode to be used with this BUFFER SEGMENT DESCRIPTOR.
	<07>	L	Last BUFFER SEGMENT DESCRIPTOR.
	<04:13>		RESERVED
	<14:35>	DSBADR	Physical address of the base of the contiguous region of the data buffer defined by this BUFFER SEGMENT DESCRIPTOR.

2	<Ø4:13>	RESERVED
	<14:35> BSDADR	Physical address of the next BUFFER DESCRIPTOR SEGMENT.
3	<Ø4:35>	Used for ICCS
4	<Ø4:35> SEG-LEN	The length of the segment of buffer in bytes which is defined by this BUFFER SEGMENT DESCRIPTOR.
5	<Ø4:35> SBT	SEGMENT BYTES TRANSFERED. SBT is initially set to zero by the Port Driver. The I/O Port stores the number of actual bytes transfered each time it transfers data to or from this region of buffer.

12.7 INTERRUPTS

Interrupts are generated by the I/O PORT under the following conditions:

1. Port initialization has been completed.
2. The PORT has inserted a CHANNEL RESPONSE PACKET on an empty RESPONSE QUEUE.
3. The PORT has placed an entry on an empty FREE QUEUE.
4. A FREE QUEUE entry was needed for processing and was not available.
5. A memory error was detected on a 2Ø8Ø memory reference.
6. An internal PORT hardware error occurred.

Refer to the 2Ø8Ø I/O Subsystem Specification and the 2Ø8Ø I/O Port Specification for further details.

12.8 I/O PORT COMMAND STATUS REGISTER

Refer to the 2Ø8Ø I/O Port Specification

CHAPTER 13

PLI INTERFACE SPECIFICATION

13.1 GENERAL

The PORT, Packet Buffer and LINK combine to provide the interface and control mechanism for a device to the ICCS Bus. The PORT is the interface to the host device and is responsible for data mapping, data buffer handling and control of the Packet Buffer and LINK interface.

The Packet Buffer module provides dual 1K transmit buffers and dual 1K receive buffers. Some control for the LINK module is also provided by the Packet Buffer module. The LINK provides the actual interface to the ICCS Bus and is capable of servicing 2 ICCS Busses. The serialization and deserialization of data, 32 bit CRC generation and checking, ICCS protocol handling, and contention arbitration, are provided by the LINK.

The PLI Interface is the means through which data transfers and communications between the PORT Packet Buffers and LINK occur. The PORT has a master/slave relationship with the Packet Buffer Link combination. All data traffic and PLI functions are controlled by the PORT.

The PLI interface has been kept simple and general to allow flexibility. It is intended that the Packet Buffers LINK and PORT reside in the same backpanel.

13.2 PLI INTERFACE SIGNALS

SIGNAL	DIRECTION PORT LINK	LINES	LOGIC	MODULE PIN
DATA (7:0)	<---->	8	TRI-ST	TBS
SELECT	---->	1	TTL	TBS
RCVR BUFFER A FULL	<----	1	TTL	TBS
RCVR BUFFER B FULL	<----	1	TTL	TBS
XMTR ATTENTION	<----	1	TTL	TBS
PLI CONTROL (3:0)	---->	4	TTL	TBS
LOAD DATA PARITY	---->	1	TTL	TBS
RECEIVE DATA PARITY	<----	1	TTL	TBS
CLOCK	---->	1	TTL	TBS
INITIALIZE	---->	1	TTL	TBS
RECEIVER STATUS	<----	8	TTL	TBS
TRANSMIT STATUS	<----	8	TTL	TBS

13.3 DATA (7:0) (TRI-STATE, ASSERTED HIGH)

These lines are used to transfer data to and from the Packet Buffers and to pass status and control information. Status goes to and from both the Packet Buffers and the LINK. Control lines determine the direction and type of information being transferred.

13.4 SELECT (TTL, ASSERTED LOW)

The select line must be asserted by the PORT to execute all data transfers and control functions. This line acts as an enable for the PLI control lines. The LINK will provide a pullup resistor on this signal so that, if the PORT is not installed, the LINK will not respond to the floating control lines.

13.5 RCVR BUFFER A FULL (TTL, ASSERTED HIGH)

This line is asserted by the Packet Buffer when receive buffer A has stored a valid packet. The signal will remain asserted until buffer A has been released by the (Release

RCVR Buffer) function. Buffer A must have been selected by the (Select Read Buffer) function prior to the (Release RCVR Buffer) function.

13.6 RCVR BUFFER B FULL (TTL, ASSERTED HIGH)

This line is asserted by the Packet Buffer when receiver buffer B has stored a valid packet. The signal will remain asserted until buffer B has been released by the (Release RCVR Buffer) function. Buffer B must have been selected by the (Select Read Buffer) function prior to the (Release RCVR Buffer) function.

13.7 XMTR ATTENTION (TTL, ASSERTED HIGH)

XMTR ATTENTION is asserted by the LINK when there is a response to a (TRANSMIT) function . The response may be any of the following:

ACK NAK Transmit Buffer Parity Error Abort Transmission

The responses are described in the Transmit Status section (13.8.2).

The XMTR ATTENTION will remain asserted until the (Reset XMTR Status) function is executed by the PORT.

13.8 PLI CONTROL (3:0) (TTL, ASSERTED HIGH)

There are four PLI CONTROL lines originating at the PORT which are used to control the interface activities of the Packet Buffer and LINK. Control functions denoted by (*) utilize the DATA lines to pass auxiliary control information to the Packet Buffer and LINK. The SELECT Line must be asserted to make the CONTROL lines valid. The CONTROL lines are encoded as follows:

PLI CONTROL (3:0) (TTL, ASSERTED HIGH)

	PLI CONTROL BIT				FUNCTION
MAINT	3	2	1	0	
0 OR 1	0	0	0	0	TRANSMIT SET UP (*)
0 OR 1	0	0	0	1	LOAD TRANSMIT BUFFER (*)
0 OR 1	0	0	1	0	RESET TRANSMIT BUFFER
0 OR 1	0	0	1	1	TRANSMIT (*)
0 OR 1	0	1	0	0	RESET TRANSMIT STATUS
0 OR 1	0	1	0	1	ABORT TRANSMISSION
0 OR 1	0	1	1	0	ENABLE LINK CONTROL (*)
0 OR 1	0	1	1	1	DISABLE LINK CONTROL (*)
0	1	0	0	0	READ RECEIVER STATUS
0	1	0	0	1	READ TRANSMIT STATUS
0	1	0	1	0	READ RECEIVER BUFFER
0	1	0	1	1	READ NODE ADDRESS
0	1	1	0	0	RELEASE RECEIVER BUFFER
0	1	1	0	1	SELECT MAINT MODE (*)
0	1	1	1	0	SELECT READ BUFFER (*)
0	1	1	1	1	SELECT LOAD BUFFER (*)
1	1	0	0	0	READ TRANSMIT DATA
1	1	0	0	1	READ MAINT 1
1	1	0	1	0	READ MAINT 2
1	1	0	1	1	READ MAINT 3
1	1	1	0	0	READ MAINT 4
1	1	1	0	1	SELECT MAINT MODE (*)

13.8.1 Read Receiver Status

This function will enable the receiver status bits onto the DATA lines. These status bits also appear on module fingers for direct access. All receiver status bits are synchronized to the PLI interface clock.

The status bit allocation is as follows:

7	6	5	4	3	2	1	0
RCVR ENB	CRC ERR	- - -	BUFB BUS	BUFA BUS	FIRST BUF	BUFB FULL	BUFA FULL

BIT DESCRIPTION

---

7 Receiver Enable: This bit is set the as a result of an (Enable LINK Control) function with bit 7 set. When cleared, the LINK is shut off to all ICCS traffic.

Receiver Enable is cleared by INITIALIZE and by the (Disable LINK Control) function with bit 7 (Receiver Disable) set.

6 CRC Error: This bit is to be used in conjunction with Internal or External Loop mode. When set, it indicates that a CRC error was detected on the message or data packet received. CRC is not generated or checked on ACK packets while in loop mode. This bit is invalid while not in loop mode.

CRC Error is cleared by INITIALIZE and by the (Reset Transmit Status) function.

5 Undefined

4 Buffer B Bus: This bit indicates which ICCS Bus the last packet in Receiver Buffer B was received.

0 = ICCS Bus A
1 = ICCS Bus B

Buffer B Bus is cleared by INITIALIZE.

3 Buffer A Bus: This bit indicates which ICCS Bus the last packet in Receive Buffer A was received.

0	=	ICCS Bus A
1	=	ICCS Bus B

Buffer A Bus is cleared by INITIALIZE.

2 First Buffer: If both Receiver Buffer B Full and Receiver Buffer A Full are true when the Receive Status is read, this bit will indicate which of the buffers was filled first. Otherwise, the bit is invalid.

0	=	Buffer A
1	=	Buffer B

1 Receiver Buffer B Full: Set when a valid packet has been stored in Receive Buffer B.

Receiver Buffer B Full is cleared by INITIALIZE and by the (Release Receiver Buffer) function if buffer B is selected. The (Select Read Buffer) function is used to select the buffer.

0 Receiver Buffer A Full: Set when a valid packet has been stored in Receive Buffer A.

Receiver Buffer A Full is cleared by INITIALIZE and by the (Release Receiver Buffer) function if buffer A is selected. The (Select Read Buffer) function is used to select the buffer.

13.8.2 Read Transmit Status

This function will enable the transmit status onto the DATA lines. These bits also appear on module fingers for direct access. All Transmit Status bits are synchronized to the PLI interface clock.

The status bit allocation is as follows:

```
       7      6      5      4      3      2      1      0
   +------+------+------+------+------+------+------+------+
   | XBUF |  CDB |  CDA | XMIT |  ARE | XMTR |  NAK |  ACK |
   |  PE  |      |      | ABRT |      | BUSY |      |      |
   +------+------+------+------+------+------+------+------+
```

BIT DESCRIPTION
----

7 Transmit Buffer Parity Error: This bit is set if a parity error was detected on the data in the transmit buffer during a transmission. A parity error will cause the XMTR ATTENTION flag to be asserted and the transmission to be aborted.

This bit is cleared by INITIALIZE and by the (Reset Transmit Status) function.

6 ICCS B Carrier Detect (CDB): This bit indicates the state of the carrier on ICCS Bus B.

1 = Carrier on
0 = Carrier off

5 ICCS A Carrier Detect (CDA): This bit indicates the state of the carrier on ICCS Bus A.

1 = Carrier on
0 = Carrier off

4 Transmission Aborted: This bit is set when a transmission was aborted by the (Transmission Abort) function.

Transmission Aborted is cleared by INITIALIZE and by the (Reset Transmit Status) function.

3 Arbitration : This bit is set when the arbitration count has expired after a (Transmit) function. This does not mean that the transmition has taken place since rearbitration takes place under certain conditions. (see section 2.1.3 of ICCS LINK spec).

Arbitration is cleared by INITIALIZE and by the (Reset Transmit Status) function.

2 Transmitter Busy: This bit is set by the (Transmit) function and indicates that a Transmit sequence is in progress.

Transmitter Busy is cleared by INITIALIZE and by the (Reset Transmit Status) function.

1 NAK: This bit is set when a valid negative reply is received in response to a transmitted packet. The NAK is the response from the destination node that both of its receive buffers were unavailable. NAK will cause the XMTR ATTENTION flag to be asserted.

NAK is cleared by INITIALIZE and by the (Reset Transmit Status) function.

0 ACK: This bit is set when a valid positive acknowledgement is received in response to a transmitted packet. ACK will cause the XMTR ATTENTION flag to be asserted.

ACK is cleared by INITIALIZE and by the (Reset Transmit Status) function.

13.8.3 Read Receiver Buffer

This function enables the contents of the currently addressed location in the receive buffer, selected by the last (Select Read Buffer) function, onto the Data lines. The address counter for that buffer will be incremented at the end of each cycle in which this function is asserted. A byte count must be maintained by the PORT to determine when to stop reading the receive buffer. The parity bit for the read data will be passed to the PORT on the RECEIVE DATA PARITY line.

The PORT cannot load the address counter of the receiver buffers and, therefore, cannot randomly access the buffers. The data is available from the receiver buffers sequentially from the first byte received to the last byte received as shown below.

```
|---------------------------------------|
|  Body Byte n                          |
|---------------------------------------|          +
|  Body Byte 0                          |         +|+
|---------------------------------------|          |
|  Source Byte                          |          |
|---------------------------------------|      DIRECTION
|  Destination byte (complement)        |          of
|---------------------------------------|         READ
|  Destination Byte (true)              |          |
|---------------------------------------|          |
|  Packet Type/ (low)                   |          |
|---------------------------------------|          |
|  Packet Type/Length (high)            |  LOC 0   |
+---------------------------------------+
```

Receiver Buffer Data Organization

13.8.4 Load Transmit Buffer

When the PORT wishes to load one of the transmit buffers, it must first select the desired buffer via the (Select Load Buffer) function. Thereafter, the (Load Transmit Buffer) function will cause the data presented on the DATA lines and its associated parity bit to be written into the selected buffer. The buffer address counter will be incremented at the end of each cycle in which the load command is present. The (Load Transmit Buffer) command is necessary for each byte transfer to the Packet Buffers. The Packet Buffer will continue to load a new byte into the packet buffer for each PLI clock period that the Load Transmit Buffer Function is asserted with select. A transition is not required.

Loading Transmit Buffers

The transmit buffers are loaded from location 0 to location n where

n = the number of bytes loaded into the buffer by the PORT

The Transmit buffer is loaded starting at location Ø with the address counter incrementing after each (Load Transmit Buffer) function. The last location loaded is saved for use in determining the last location to transmit.
The transmit buffers must be loaded, starting at location 0, as shown below.

```
|----------------------------------------|
| Body Byte n                            |
|----------------------------------------|          +
| Body byte 0                            |         +|+
|----------------------------------------|          |
| Source Byte                            |          |
|----------------------------------------|      DIRECTION
| Destination Byte (complement)          |         of
|----------------------------------------|        LOAD
| Destination Byte (true)                |          |
|----------------------------------------|          |
| Packet Type/ (low)                     |          |
|----------------------------------------|          |
| Packet Type/Length (high)              | LOC 0    |
+----------------------------------------+
```

Transmit Buffer Data Organization

13.8.5 Release Receiver Buffer

This function will reset the receiver buffer currently selected, making it available to the ICCS for buffering of new packets. (Release Receiver Buffer) causes the buffer address pointer to reset to zero and resets the RCVR BUFFER FULL line of the buffer selected by the last (Select Read Buffer) function

13.8.6 Read Node Address

The (Read Node Address) function is used to read the 8 bit ICCS node address from the LINK's node address thumbwheel switch.

INITIALIZE will not disturb the LINK node address register.

The (load Node Address) function must not be issued while the receiver is enabled.

13.8.7 Select Read Buffer

The Packet Buffer has 2 1K byte receive buffers which are used to store inccomming ICCS packets. The (Select Read Buffer) function is used to select the receive buffer to be read via succeeding (Read Receive Buffer) functions. This function will also select which receive buffer will be reset by the (Release Receiver Buffer) function. The port also has the ability to read the data it has put into the transmit buffer. DATA bits 2-0, supplied by the PORT, are defined as follows during this function.

DATA BIT 2-0	DESCRIPTION
000	Select Receiver Buffer A
001	Select Receiver Buffer B
010	Select and offset Address of rec Buffer A
011	Select and offset Address of rec Buffer B
100	Select Transmit Buffer A
101	Select Transmit Buffer B
110	Select and offset transmit Buffer A
111	Select and offset transmit Buffer B

DATA BIT 6-3 =offset in bytes from address 0

Receiver Buffer A will be selected by INITIALIZE.

13.8.8 Select Load Buffer

The Packet Buffer has 2 independant 1K transmit buffers which are used to store outgoing ICCS packets. The (Select Load Buffer) function is used to select the transmit buffer to be loaded via succeeding (Load Transmit Buffer) functions. This function will also select the transmit buffer for the (Reset Transmit Buffer) function.

Data bit 0, supplied by the PORT, is defined as follows during this function.

DATA BIT 0	DESCRIPTION
0	Select Transmit Buffer A
1	Select Transmit Buffer B

Transmit Buffer A will be selected by INITIALIZE.

13.8.9 Transmit Setup

This function is used to select which transmit buffer and which ICCS Bus is to be used by the next (Transmit) function. Either buffer may be transmitted over either ICCS Bus. DATA Bits 1 and 0 are defined as follows during this function.

DATA BIT	STATE	DESCRIPTION
0	0	SELECTS TRANSMIT BUFFER A
	1	SELECTS TRANSMIT BUFFER B
1	0	SELECTS ICCS BUS A
	1	SELECTS ICCS BUS B

This function must not be executed while a transmit sequence is in progress.

INITIALIZE will select Buffer A and ICCS Bus A.

13.8.10 Transmit

The (Transmit) function will initiate ICCS arbitration on the ICCS bus selected by the (Transmit Setup) function. The selected buffer will be transmitted upon successful arbitration. The PORT must place the arbitration persistance number on the DATA lines when this function is issued. Parity is checked on the buffer data as it is transmitted and, if a parity error occurs, the transmission will be aborted.

The persistance number is a 4 bit number which is loaded into the LINK persistance counter Via DATA lines (3:0). This number represents the number of quiet slots the LINK must observe on the selected ICCS Bus before transmitting, unless that ICCS Bus was quiet when the (Transmit) function was issued. In the later case, the transmission will begin immediately.

It is possible to retransmit a buffer without reloading it. There are several retransmit situations. These situations and their respective procedures are as follows.

A. Retransmit immediately on the same ICCS Bus:
   1. Issue a new (Transmit) function.
B. Retransmit immediately on the alternate ICCS Bus:

1. Issue the (Select Load Buffer) function.
2. Issue the (Transmit Setup) function.
3. Issue the (Transmit) function.

C. Retransmit buffer A (B) if buffer B(A) has been transmitted since the original transmission.

1. Issue functions 1,2,3,4 the same as in case B above.

Only one (Transmit) function can be executed at a time. That is, the transmission must have been completed or aborted and the Transmit Status must be cleared before a second (Transmit) function can be issued.

13.8.11 Reset Transmit Status

This function must always be executed at the end of a transmission sequence, i.e., in response to a XMTR ATTENTION line after reading the transmit status. It will reset all transmit status bits, except bits 5 (CDA) and 6 (CDB).

13.8.12 Reset Transmit Buffer

This function will reset the address counter of the transmit buffer selected by the last (Select Load Buffer) function.

This function must be issued prior to loading a transmit buffer.

13.8.13 Select Maint Mode

The Select Maint Mode function allows diagnostics to examine internal logic in the Packet Buffer Board to provide fault isolation. The Data Bit format is as follows:

7	6	5	4	3	2	1	0
NOT USED	NOT USED	NOT USED	NOT USED	RCVR BUFB ENA	RCVR BUFA ENA	MAINT LOOP ENA	MAINT MODE ENA

BIT DESCRIPTION
----

0 MAINT MODE - Data bit 0=1 will cause Maint Mode to set. and data bit 0=0 will cause Maint Mode to clear. The Maint Mode bit redefines Read Receiver status, Read transmit status, read receiver buffer, read node address, and release receiver buffer to be maint functions. Reset clears MAINT MODE.

1 MAINT LOOP - Maint Loop allows the transmit data to be looped into the receive input register. This function should only be done while the link is disabled. Maint loop will have no affect if Maint Mode is not also set.

2 Receiver Buffer A: (Enable...) will enable receiver buffer A to be used to store incoming ICCS packets.

(Disable...) will cause buffer A to always appear full to the ICCS, thus, making it unavailable. The RECEIVER BUFFER A FULL line (Receiver Status bit 0) will not be set.

INITIALIZE will enable buffer A.

3 Receiver Buffer B: (Enable...)) will enable receiver buffer B to be used to store incoming ICCS packets.

(Disable...)) will cause buffer B to always appear full to the ICCS, thus, making it unavailable. The RECEIVER BUFFER B FULL line (Receiver status bit 1) will not be set.

INITIALIZE will enable Buffer B.

13.8.14 Abort Transmission

(Abort Transmission) will cause a currently active transmission to be aborted and will set bit 4 (Transmission Aborted) of the Transmit Status. XMTR ATTENTION will also be asserted. If XMTR ATTENTION is asserted when the (Abort Transmission) function is issued, the abort will be ignored.

13.8.15 Enable LINK Control/Disable LINK Control

These functions are used to enable and disable certain long term controls in the LINK. A particular control may be set by executing (Enable LINK Control) with a 1 in the DATA line bit position corresponding to that control. A control may be cleared by executing (Disable LINK Control) with a 1 in the proper bit position. Transfer of a 0 in any bit position will have no effect on the corresponding controls.

The DATA line bit layout is as follows:

7	6	5	4	3	2	1	0
RCVR B ENA	FORCE CDET ENA	VALID RPAR ENA	EXTM LOOP ENA	INTM LOOP ENA	FORCE ARB ENA	SWAP ADR ENA	RCVR A ENA

ENABLE LINK CONTROL

7	6	5	4	3	2	1	0
RCVR B DIS	FORCE CDET DIS	VALID RPAR DIS	EXTM LOOP DIS	INTM LOOP DIS	FORCE ARB DIS	SWAP ADR DIS	RCVR A DIS

DISABLE LINK CONTROL

BIT DESCRIPTION
----

7 Receiver B Enable/Disable: (Enable...) activates the LINK receiver B making the node accessable by remote nodes on this path, or CI bus.

(Receiver B Disable...) will cause this node to ignore all ICCS traffic on the B bus. Internal Loop mode will not function if the receiver is enabled.

INITIALIZE will disable the receiver.

6 Force Carrier Detect: This function must only be used in conjunction with Internal Maintenance Loop mode.

Enable.. causes the link to see a detected carrier. This forced carrier will start the header timeout logic and will stop the arbitration sequence.

5 Valid Receiver Parity: (Enable...) function will force the receiver to generate odd parity in the data buffer.

(Disable...) will cause the LINK receiver to generate even parity in the data buffer.

INITIALIZE will set the receiver to odd parity.

4 External Loop: (Enable...) will allow the LINK to receive its own transmission by looping on the selected ICCS bus. Internal and External Loop modes must never be invoked simultaneously.

The following requirements must be met:

1. The Destination field in the ICCS packet must contain the address of this node.

2. The PORT must load a pre-calculated CRC into the buffer immediately following the data. CRC will be checked by the receiver.

3. The packet byte count must not include the CRC characters.

4 Transmit buffer loading and transmit setup functions must occur the same as they do for normal operation. The normal ICCS arbitration sequence will occur when a (Transmit) function is issued while in External Loop mode.

If External Loop mode is invoked while on line in an operating system, packets may still be received from a remote node. There is, therefore, no guarantee that there will be a receiver buffer available when the local packet is transmitted. A real ACK response will not occur while in External Loop mode. However, the Transmit and Receive Status responses will be set as though a normal transmission had taken place.

(Disable...) will take the link out of External Loop mode.

INITIALIZE will disable this function.

3 Internal Loop: (Enable...) will cause the LINK to receive its own transmission by looping somewhere (TBD) inside the ICCS driver/receiver. This loop will not interfere with ICCS operation of other nodes. Traffic on both ICCS A and B will be ignored while in Internal Loop mode.

Internal and External Loop modes must never be invoked simultaneously.

The same requirements (1,2,3) stated above for External Loop apply to Internal Loop mode. Transmit buffer loading and transmit setup functions must occur the same as they do for a normal operation. ICCS arbitration will not occur in Internal Loop mode.

ACK responses will occur in this mode. However, CRC will not be calculated or checked on the ACK packet. Transmit and Receive status responses will be the same as they are during normal transmissions on the ICCS Bus.

(Disable...) will take the LINK out of Internal Loop mode. The receiver must be enabled to execute Internal Loop mode.

INITIALIZE will disable this function.

2 Force Arb: enable function will cause the LINK to force a successful arbitration and to transmit immediately upon receiving the Transmit function from the Port.

Disable, will allow the LINK to perform normal arbitration sequences in response to the Transmit function from the PORT.

1 Swap Address: Enable, will cause the true and complement address sources to swap, i.e., true node address becomes complement and complement node address becomes true. This will cause an address mismatch if a packet is sent in either of the maintenance loop modes and will result in a no response status.

If, while in Internal Maintenance Loop mode, the Port enables theis function and also complements the address in the packet to be transmitted, then address comparison will take place. This has given the PORT the ability to test all node address bits in both polarities.

Note that the later case should only be exercised while in Internal Maintenance Loop mode since the Link will be using an address which was not assigned to it during system configuration.

Disable, sets the node address with sources to their normal positions, i.e., true equals true ect.

Initialize also causes the switches to be set to their normal positions.

0 Receiver A Enable/Disable: (Enable...) activates the LINK receiver A making the node accessable by remote nodes on this path, or CI bus.

(Receiver A Disable...) will cause this node to ignore all ICCS traffic on the A bus. Internal Loop mode will not function if the receiver is enabled.

INITIALIZE will disable the receiver.

13.9 LOAD DATA PARITY (ODD) (TTL, ASSERTED HIGH)

Odd parity is calculated by the PORT on the data transferred to the Packet Buffer Transmit buffers. This parity bit is conveyed to the Packet Buffers via the LOAD DATA PARITY line.

The Packet Buffer will store the parity bit as supplied and the LINK will check parity when reading the buffer during a transmission.

13.10 RECEIVE DATA PARITY (ODD) (TTL, ASSERTED HIGH)

Data being read from a receive buffer includes a parity bit which was generated when the data was written into the buffer. This parity bit will be conveyed to the PORT via the RECEIVE DATA PARITY line and must be checked by the PORT. The LINK does not check receive data parity.

13.11 CLOCK (TTL)

The PLI interface requires a clock source from the PORT for its operation. The interface will operate with a minimum cycle period of 150 ns as shown in Figure 1. All PLI interface flags and status bits are synchronized to this clock.

13.11.1 Timing

Preliminary Packet Buffer timing requirements on the PLI are as shown in Figure 1. All times are shown referance to the finger pins going into the packet buffer card. These times seem to be realistic at this time, however, they may change as the LINK design procedes. All PLI interface signals must be asserted by the negative edge of the clock. The Packet Buffer and the LINK will strobe the data on the positive edge of the clock.

```
                       |<------150 ns min--->|

         --------------+              +------+
                       |              |      |
                       |<- 90 ns min->| 30 ns|<-
PORT CLOCK             |              |  min.|
                       +--------------+      +--------
                                      |<- BUFFER OR LINK
                                      |   TAKES DATA
                             |<-45 ns-|
                             |  min   |
                       +---------------------------------
                       |/ / /|                |/ / / / / / /
        Control +      | / / |                | / / / / / /
        Select         |/-/-/|----------------|/-/-/-/-/-/

                                      |
                                      |
                                      |
                     --+---------------------------------
                       |/ / /|        |       |/ / / / / / /
        Data + Parity  | / / |        |       | / / / / / /
(PORT-> Packet Buffer) -|/-/-/|--------|-------|/-/-/-/-/-/
                             |  45ns  |
                             |  min   |

       From Control +      ->|   |<- 40ns max
       Select                |   |
                       +---------------------------------
                       |/ / / / /|            |/ / / / / / /
        Data + Parity  | / / / / |            | / / / / / /
(Packet Buffer-> PORT) |/-/-/-/-/|------------|/-/-/-/-/-/

                       +---------------------------------
                       |/ / /|                |/ / / / / / /
        Attention      | / / |                | / / / / / /
          (any)        |/-/-/|----------------|/-/-/-/-/-/

                       30 ns |
                       |max  |
```

Figure 1
PLI Interface Timing

13.12 INITIALIZE (TTL, ASSERTED HIGH)

This signal from the PORT is used for system initialization. The PLI will provide a pullup resistor on this signal so that, if the PORT is not installed, the PLI will not interfere with or respond to the ICCS Bus. The INITIALIZE line may have to be asserted for up to 3 or 4 interface clock periods to allow internal state machines to reset themselves.

13.13 RECEIVER STATUS

The Receiver Status bits are described in section 13.8.1 (Read Receiver Status).

13.14 TRANSMIT STATUS

The Transmit Status bits are described in section 13.8.2 (Read Transmit Status).

13.15 PLI INTERFACE SEQUENCE FLOWS

The following flows are examples of the PLI interface sequence of events for a transmission and the sequence for servicing a receive buffer.

```
                    +------+
                    |START|
                    +------+
                       |
          +---------------------+
          | SELECT LOAD BUFFER |
          +---------------------+
                       |
              +---------------+
              | RESET BUFFER |
              +---------------+
                       |
          +---------------------+
          | LOAD BYTE TO BUFFER|
          +---------------------+
                       |
                                   NO
                  LAST BYTE
                       ?
                           YES
                       |
            +----------------+
            | TRANSMIT SETUP |
            +----------------+
                       |
                +-----------+
                | TRANSMIT |        (Loads Persistence _#)
                +-----------+
                       |
                       |<-----------------------------------
                                 NO                        |
             XMTR ATTENTION    ----------                  |
                                         |                 |
                       |   YES                      YES    |
                       |             WAIT LONGER  ------->
                       |                 ?                 |
                       |                 |   NO            |
       +----------------------+          |                 |
       | READ TRANSMIT STATUS |          V                 |
       +----------------------+  +-------------------+ |
                       |         |ABORT TRANSMISSION | |
                       |         +-------------------+ |
    +----------------------------+       |                 |
    | RESET TRANSMISSION STATUS |        |---------------|
    +----------------------------+
                       |
                   +-------+
                   | DONE |
                   +-------+
```

PLI Interface Transmit Sequence

```
            +--------+
            | START  |
            +--------+
                 |
                 |<---------------+
                 |                |
        +-----------------+       |NO
        | RCVR ATTENTION  |-------+
        |       ?         |
        +-----------------+
                 | YES
                 |
                 |
      +-----------------------+
      | READ RECEIVE STATUS   |
      +-----------------------+
                 |
                 |
       +-------------------+
       |   DECIDE WHICH    |
       |  RECEIVE BUFFER   |
       +-------------------+
                 |
     +------------------------+
     | SELECT RECEIVE BUFFER  |
     +------------------------+
                 |
                 |
                 |
         +---------------+
         |   READ BYTE   |
         +---------------+
                 |
                 |
                 |<--------+
                 |         |
           +------------+  |NO
           | LAST BYTE  |--+
           |    ?       |
           +------------+
                 | YES
                 |
                 |
         +-----------------+
         | RELEASE BUFFER  |
         +-----------------+
                 |
             +-------+
             | DONE  |
             +-------+
```

PLI Interface Receive Sequence

CHAPTER 14

2080 TTL BUS

14.1 INTRODUCTION

14.1.1 Definition

The 2080 TTL-Bus is an information path and communication protocol for data transfer among the elements of a data processing system. Data may be exchanged between I/O bus port and memory . The communication protocol allows the information path to be time-multiplexed such that an arbitrary number of data exchanges may be in progress simultaneously.

14.1.2 Nomenclature

The terms used in this specification are defined as follows:

Information - intelligence used to control and provide the basis for data processing including addresses, data and status.

Data - program generated information which is the object of processing.

Bus port - a physical connection to the 2080 TTL - BUS capable of any or all of the functions described below.

Commander - a bus port which transmits Command/Address information to the Mbox via the IO adapter.

IO adapter - an adapter which recognizes Command/Address information, converts it to ECL and transfers it to the Mbox. The IO bus adapter also performs the bus arbitration.

Mbox - a Cached memory controller.

Transmitter - a bus port which drives the signal lines.

Receiver - any bus port which samples and examines the signal lines.

14.1.3 Goals

Specifically designed to meet the requirements of the 2080 system, (2080 TTL-Bus) provides checked, parallel information transfer synchronous with a common system clock. In each clock period or cycle, interconnect arbitration, information transfer and transfer confirmation may occur in parallel. All 2080 TTL Bus transfers are monitored by a bus monitor which when enabled will cause the machine to freeze when it detects a bus error. The Console will be able to determine the source of all bus errors. Multiple failures and bus port control failures are not necessarily detected.

14.1.4 Maximum Transfer Rate

Using a 125 nanosecond clock period (the optimum speed for the components to be used), the 2080 TTL-Bus achieves a maximum data transfer rate of 6.4 million 36-bit words per second. The bus clock will be derived from the master oscillator for the cpu and will be 125 to 160 nanoseconds depending on the final clock rate of the 2080.

14.1.5 Information Path Usage

The same information transfer path is used to convey command, address words and data words. Each exchange consists of one, two or five information transfers: one command/address word and one data word. For write type commands, the bus port uses two or five cycles, transmitting command/address then one or four data. Read type commands also begin with a command/address transfer from the bus port, however, data emanates from the IO adapter and may be delayed by whatever access time is characteristic; IO adapter uses one to four cycles. Io adapter data transfers, like all others, are synchronous with the system clock.

14.1.6 Confirmation

The memory ACK signal confirms that the receiver has received a command address word. Any time a word is received that has bad parity the receiver asserts the parity error signal.

14.1.7 Control Signals

In addition to signals for arbitration, information transfer and confirmation, the 2080 TTL-Bus includes priority interrupt request signals and control signals to synchronize and provide notification of changes in system state including loss and resumption of power.

14.1.8 Information Path Width

The 2080 TTL-Bus is a 36 bit word oriented (rather than block or byte) The bus does not provide any byte write or masking capability.

14.1.9 Bus Monitor

The bus monitor will conditionally check all bus transfers, and cause all bus transfers to cease if any bus errors are detected. The bus monitor may be disabled by software.

14.2 SIGNAL DESCRIPTION

14.2.1 Signal Types

The 2080 TTL-bus consists of 78 signals. Divided by function into five classes, there are 16 arbitration lines, 36 information transfer lines, 9 control lines ,7 interrupt request lines and 2 clock lines. See figure 6.10.

```
+-+ +-+      +-+         +---------------------------------------+
!1! !1!      !1!         !                  36                   !
+-+ +-+      +-+         +---------------------------------------+
 P   C CYC    D CYC                       D<00:35>


+-+     +--+       +--+
!1!     ! 1!       ! 1!
+-+     +--+       +--+
FAULT   MEM ACK   UNCOR


+-+     +-+    +-+        +-+        +-----+    +---+
!1!     !1!    !1!        !1!        !  2  !    ! 3 !
+-+     +-+    +-+        +-+        +-----+    +---+
RESET   ACLO   DCLO    INTERLOCK      CLOCK      TAC


+--------+  +--------+  +--------+  +---+     +-+   +-+        +-+
!   8    !  !   8    !  !   7    !  ! 4 !     !1!   !1!        !1!
+--------+  +--------+  +--------+  +---+     +-+   +-+        +-+
BREQ<0:7>  GRANT<0:7>   IREQ<1:7>  SSEL<0:3>  IO   BROADCAST  RING
                                              BUSY
```

2080 TTL-Bus
SIGNALS

Figure 5.10

14.2.2 Arbitration

Arbitration logic, located on the TTL adapter determines which bus port of those requesting access to the interconnect in a particular cycle will perform an information transfer in the following cycle.

14.2.2.1 Arbitration Line Assignment - Arbitration lines, BREQ<0:7>, are assigned one per bus port to establish fixed priority access to the information path. Priority increases from REQ7 to REQ0.

14.2.2.2 Arbitration Line Use - To acquire control of the information path, a bus port asserts its assigned Request line (BREQ<0:7>) at the beginning of a cycle. The arbitration logic located in the TTL adapter determines which bus port if any should be granted the bus and sends a grant out to that bus port. At the beginning of the next cycle, the bus port can assert information on the bus if the Grant for this bus port has been returned. If higher priority REQ's are present, then the bus port's REQ remains asserted and the test is made again at the start of the next cycle.

14.2.2.3 Number Of Bus Ports Arbitrating - A total of 8 bus ports may be used operating on eight priority levels.

14.2.3 Information Transfer Lines

Three subfields comprise the information transfer path: parity check (P), information bits (D<00:35>), C CYC, D CYC. Parity check bit provides redundancy for detecting single bit (all EVEN numbers) errors in the information path. Transmitting bus port generate P as parity for D<00:35>. P is generated such that the sum of all one bits in the checked field including parity is odd.

14.2.4 C CYC, D CYC

C CYC, D CYC are asserted by the transmitting bus port to indicate the type of information being transmitted. The interpretation of P and D<ØØ:35> by a receiving bus port is determined by the C CYC, D CYC bits. Two information types are defined. See figure 6.20.

```
   -----------------------------------------
  !                                        !
  !!                                       !
  !!                                       ------------------------------------
  !!                                       0                                  3
  10      1          0        0                                               5
+--+  +---+ +---+ +---+   +--------------------------------------+
!  !  !   ! !   ! !   !   !                                      !
+--+  +---+ +---+ +---+   +--------------------------------------+
 P    C CYC  TAG  D CYC                 D<00:35>
```

2080 TTL-Bus
CODES AND FORMATS
Figure 6.20

C CYC asserted informs the TTL adapter that the information on D<ØØ:35> is an command address.
D CYC asserted informs the receiver that the information on D<ØØ:35> is data.

All bus ports check parity on all information that they transmit or receive from the bus.

The IO Adapter checks the parity of the latched information on each C CYC, the transmitter also checks parity on a C CYC. The transmitter and the receiver check parity on D CYC. Figure 6.30 shows the formats for each type of information.

C CYC or D CYC

```
 Ø                                     3
 Ø                                     5
+--------------------------------------+
!                                      !
+--------------------------------------+
```

READ DATA, READ DATA CORRECTED, READ DATA FAULTY

2080 TTL-Bus
INFORMATION FORMATS (D<ØØ:35>)
Figure 6.30

14.2.4.1 Command/Address C CYC - C CYC specifies that D<00:35> are a command/address word.

14.2.4.2 Data D CYC - D CYC indicates that D<00:35> contain data solicited by a previous read type command, or is the write data for a write command depending on the previous Command Address word.

14.2.4.3 TAG - TAG identifies one of eight bus ports that is transmitting or receiving information.

14.2.5 SSEL<0:3> - Start SELect <0:3>

Start SELect is used with the DING signal to indicate to the bus port who's number is encoded on SSEL <0:3> that there is a command to be executed. The bus port must fetch and execute from it's EPT block in memory whatever command is stored. Start SELect is also used with the Broadcast signal to indicate the interrupt level that is being serviced. The Broadcast signal is sent from the EBOX to the IO adapter where it is shaped to a pulse that is one bus tick wide. A bus port that is requesting on the interrupt level specified by the SSEL <0:3> when the Broadcast signal is sent, asserts IO BUSY, requests the bus and stores an interrupt vector word in the EPT. The interrupt vector word in the EPT tells which bus port interrupted. The bus port also stores a status word in a port dependent EPT location which contains enough information as to why the interrupt was requested. The bus port clears the IO BUSY line after storing the status word. There will be 8 EPT locations associated with each port.

14.2.6 IO BUSY

The IO BUSY signal is transmitted by a bus port that has received a DING with it's bus port number matching <SSEL 0:3> but has not finished processing the command from the EPT. IO BUSY is also set by a port when it receives Broadcast and the level that it is interrupting on matches the <SSEL 0:3>. The port keeps the IO BUSY line up until it has completed storing the interrupt vector and status word.

14.2.7 IREQ<1:7> Interrupt REQuest

Any bus port can request an interrupt on any one of the 7 interrupt levels IREQ<1:7>. The interrupt level must stay up until the interrupt condition is cleared.

14.2.8 Read Data UNCORRECTABLE

UNCOR (uncorrectable) denotes a transmission of FAULTY data solicited by a read type command. When ECC logic is unable to correct the requested data, UNCOR is used to indicate this condition. D<00:35> contain, if possible, the uncorrected data or other meaningful data.

14.2.9 Command/Address Format

Information bits D<00:35> carry the payload of the 2080 TTL-Bus. Information appears on these lines in command/address format, data format. In command/address format, information is grouped in two fields: Cycle type and a 24 bit physical address. See figure 5.40.

```
+----+---------------------------------+
!    !                                 !
+----+---------------------------------+
F<0:3>    A<12:35>
```

14.2.9.1 Commands -

F<0:3>	
0000	READ FOUR WORDS
0001	READ FOUR WORDS EPT
0010	READ ONE WORD
0011	READ ONE WORD EPT
0100	READ INTERLOCK FOUR WORDS
0101	READ INTERLOCK FOUR WORDS EPT
0110	READ INTERLOCK ONE WORD
0111	READ INTERLOCK ONE WORD EPT
1000	WRITE FOUR WORDS
1001	WRITE FOUR WORDS EPT
1010	WRITE ONE WORD
1011	WRITE ONE WORD EPT
1100	WRITE INTERLOCK RELEASE FOUR WORDS
1101	WRITE INTERLOCK RELEASE FOUR WORDS EPT
1110	WRITE INTERLOCK RELEASE ONE WORD
1111	WRITE INTERLOCK RELEASE ONE WORD EPT

2080 TTL-Bus
COMMAND CODES

Figure 5.40

Four functions are defined: READ , READ INTERLOCK, WRITE , WRITE RELEASE.

14.2.9.2 READ - Encoded into F<0:3> as 0010, READ instructs the Memory selected by A<12:35> to retrieve the addressed 36 bit data and transmit it to the bus port If this is impossible due to uncorrectable data error the addressed Mbox shall transmit UNCOR insteau. No fixed time is established for retrieval but the bus port, after an interval of 10 microseconds without response may assume catastrophic error.

14.2.9.3 Read INTERLOCK Function - F<0:3> = 0100 The READ INTERLOCK function used to interlock a queue. A bus port interlocks a queue to add something to it or take something off it. The interlock prevents other bus ports from accessing the queue while the modification is taking place. The READ INTERLOCK instructs the bus port selected by A<12:35> to retrieve the addressed 36 bit data and transmit it to the bus port. The READ INTERLOCK function also causes the selected bus port to raise the INTERLOCK SIGNAL. The read interlock cycle is done in the same way as the normal read cycle. The following happens if the bus is not already interlocked (interlock not asserted):

1. The bus port requests the TTL bus.

2. The Arbitrator grants the bus port the bus.

3. The bus port transmits a command address word that has D00 not asserted and D01 asserted (READ INTERLOCK command address word).

4. The adapter then requests the Mbox.

5. The Mbox sends a grant to the adapter.

6. The adapter transmits the READ INTERLOCK command address word to the MBOX.

7. The Mbox asserts the ECL interlock signal and returns the requested words to the TTL adapter.

8. The adapter returns the requested word or words to the bus port and asserts the interlock signal at the same time as the last word is transmitted.

The read interlock provides a mechanism for a bus port to interlock a QUEUE making it inaccessible to any other bus port that tries to access it with a READ interlock function. The read interlock does not lock up the memory or prevent any non READ INTERLOCK access to the memory. The Pbox can wait for the interlock to clear either by testing the line or by staying a page fail loop until the interlock is cleared.

The following happens if the Mbox becomes interlocked after the bus port on the TTL bus issues a command address word cycle but before the adapter is granted the Mbox.

1. A INTERLOCK SIGNAL will be transmitted to the bus port with no data cycle.

2. The adapter will grant the TTL bus to the next requester.

3. All devices on the TTL bus who's next request is for is for an interlock read must not request the bus while the interlock signal is asserted the interlock signal is asserted, so that noninterlocked transfers can take place at maximum speed.

4. Any bus port that receives the INTERLOCK signal with no data cycle in response to a READ INTERLOCK command address word, must request the bus again when the interlock is not asserted and retransmit the the READ INTERLOCK command address word.

14.2.9.4 Write - WRITE F<0:3> = 10x0, instructs the Memory to modify the data in the storage element or elements addressed by A<12-35>, using data transmitted in the next succeeding cycle or cycles.

14.2.9.5 WRITE RELEASE Function - F<0:3> = 1100, the Write Release function instructs the Memory selected by A<12:35> to first modify the word addressed using the data transmitted in the next succeeding cycle or cycles The Interlock line is dropped after the words are written. WRITE RELEASE is an illegal function if the interlock is is not asserted, The Mbox will flag this as a memory error.

14.2.9.6 READ EPT Function - F<0:3> = 00x1, The READ EPT function instructs the bus port selected by A<12:35> to first take the READ EPT command address word and replace A<12:26> with the contents of the Executive Base Register and read the word or words requested.

14.2.9.7 Bus Port Not Implementing READ INTERLOCK - Bus ports that are not memory or an adapter need not assert the READ INTERLOCK bus signal. Bus ports that do not do Read Interlock or write release need not look at the interlock signal.

14.2.9.8 A CPU Bus Port Implementation Of Read Interlock - CPU bus port must implement the READ INTERLOCK and WRITE RELEASE functions and cooperate through the use of the INTERLOCK bus signal. A CPU must not request the bus for issuing a READ INTERLOCK function if the INTERLOCK bus signal is asserted by the Mbox.

14.2.9.9 Unused Function Codes - Function codes that are marked reserved must produce an error in any bus port that receives them.

14.2.10 Physical Address Space

The 24 physical address bits define a 16,777,216 word address space. Memory begins at address 0, the address space is r and consists only of storage elements.

14.2.11 Bus Port Configuration/Status Register

Each bus port must provide a configuration/status register at the lowest address assigned to it in the I/O address space. This register contains a code identifying the bus port type as well as FAULT status indicator.

14.2.12 Confirmation Lines

Mem ACK and parity error , provide a signal path from the receiver to transmitter two cycles after each information transmission. Confirmation is delayed to allow information path signals to propagate, to be received, checked and decoded by all receivers and to be generated by the responder. During each cycle, every bus port receiving or transmitting latches and makes judgements on the information transfer signals. See figure 6.50. The IO ADAPTER will Recognize command address information. The IO adapter will assert mem ACK or parity error appropriate. Only IO ADAPTER or transmitting bus port may assert parity error for any of several reasons; assertion of parity error indicates protocol or information path failure.

```
WRITE COMMANDS

  ARB.  ! BREQ N ! HOLD !       !       !       !       !       !
        ----------------------------------------------------------
  INF.  !        ! C/A  !  WD   !       !       !       !       !
        ----------------------------------------------------------
  CONF. !        !      !       ! ACK   ! ACK   !       !       !
              ->!      !<-
                2080 TTL-Bus CYCLE


WRITE RELEASE

  ARB.  ! BREQ N ! HOLD !       !       !       !       !       !
        ----------------------------------------------------------
  INF.  !        ! C/A  !  WD1  !       !       !       !       !
        ----------------------------------------------------------
  CONF. !        !   ^  !    ^  ! ACK ! ACK !       !       !
                                ! READ INTERLOCK DEASSERTED


READ COMMAND

  ARB.  ! BREQ N !      !       !       !       !       !       !
        ---------------------------\-------------------------------
  INF.  !        ! C/A  !       !  \    !D CYC  !       !       !
        -----------------------------\-----------------------------
  CONF. !        !      !       ! ACK\  !       !       ! ACK   !       !


INTERLOCK READ COMMAND

  ARB.  !BREQ N  !      !       !\      ! HOLD  !       !       !
        ---------------------------\-------------------------------
  INF.  !        ! C/A  !       !  \    !  RD1  ! RD2   !       !
        -----------------------------\-----------------------------
  CONF. !        !      !       ! ACK\  !       !       ! ACK   ! ACK   !
                                !               ! READ INTERLOCK
ASSERTED
```

2080 TTL-Bus
COMMAND SEQUENCES
Figure 6.50

IO adapter that decode a command address word and determine that the address is within it's address space return ACK. The addressed adapter does not transmit parity error provided that the parity of the command address word checks.

14.2.12.1 Use Of Confirmation - Transmitting bus ports sample the ACK and optionally interlock line at the beginning of the third cycle after transmission. ACK is the expected response indicating either that the command will be executed or the information has been correctly received. No response confirmation should be treated the same as ERR.

14.2.12.2 Successive Cycle Confirmation - Because WRITE and WRITE RELEASE operations consist of successive transfers, acknowledgement is somewhat more complex. If ACK is not received as confirmation for WRITE DATA then the transfer should be aborted and contingency mechanisms invoked.

14.2.12.3 Presumption Of Success - Some adapter may be unable to determine within two 2080 TTL-Bus cycles whether a function will be successfully completed. For these cases, the adapter should presume success and respond with ACK confirmation. If it is later determined that a READ type function cannot be completed, a READ DATA transfer of all zeros should be transmitted and an interrupt request signaled. If a WRITE type request cannot be completed, the command should be aborted and an interrupt request signaled. In either case, the cause of the interrupt should be indicated in a status register. Note that this technique is never used by a memory .

14.2.13 Fault Detection

Any of several conditions may cause a bus port to assert FAULT. FAULT[A], information path parity error may be generated by one or more bus port depending on the origin of the problem causing the calculated Parity to disagree with the received Parity. FAULT[B] results when the IO adapter which received a WRITE or WRITE RELEASE command in the immediately preceding cycle does not receive the anticipated WRITE DATA in the following cycle. FAULT[C] is indicated when a bus port which has not issued a READ or INTERLOCK READ command receives a response to a read type command. FAULT[D] arises when a bus port receives a WRITE RELEASE command and INTERLOCK READ has not been set by a INTERLOCK READ command.

14.2.13.1 Status Register FAULT Indicators - Each bus port maintains bits in its status register to indicate conditions which cause the assertion of FAULT. See figure 6.7C. These status indicators are examined by the Console to determine the source of a FAULT signal. The FAULT status bits are cleared when the FAULT signal is deasserted.

14.2.14 Interrupt Request Lines

Seven interrupt request lines, REQ<1:7>, are used by bus port which must invoke a CPU to service some condition such as transfer completion for an I/O bus port. Requesting bus port may assert one of REQ<1:7> synchronous with the 2080 TTL-Bus clock to signal a processor that attention is required. Any of the REQ lines may be asserted simultaneously by more than one bus port and any combination of lines may be asserted by the collection of requesting bus ports.

14.2.15 Clock Signals

The two clock signals, T clock and R clock, provide a universal time base for all bus port connected to the 2080 TTL-Bus. A standard circuit present in every bus port uses the two interconnect clock signals to produce the signals which actually synchronize bus port activity. As shown in figure 6.80, the derived signals define four instants in each cycle: T0, T1, T2 and T3. At the leading edge of T clock, the begining of the cycle, a transmitting bus port enables its drivers; at the leading edge of R clock, all bus port unlatch, allowing the 2080 TTL-Bus signals into there latches. The time from T clock to R clock is determined by maximum cable delay and driver-receiver propagation delay. T3 to T0 time, while bounded by receiver propagation delay, is fixed by the propagation delay of the logic required for 2080 TTL-Bus arbitration. The clock period is 125 to 160 nanoseconds with each time interval T0-T1, T1-T2, T2-T3, T3-T0 being centered on31.25 nanoseconds.

CLOCK SIGNALS

```
            T0      T1      T2      T3      T0      T1      T2      T3
            !       !       !       !       !       !       !       !
        ----+-------+-------+-------+-------+-------+-------+-------+---
            !       !       !       !       !       !       !       !


            +---------------+               +---------------+
TCLK H      !               !               !               !
        ----+               +---------------+               +-----------

        ------------+               +---------------+               +---------
RCLK L              !               !               !               !
                    +---------------+               +---------------+

                 2080 TTL-Bus
                            CLOCK SIGNALS
                              Figure 6.80
```

14.2.15.1 Use Of Clock Signals - TCLK H and RCLK L are used to clock the driver flip-flops and receiver latches respectively. As the loading of these signals affects signal skew, one buffered output of the clock derivation circuit for each of these signals is reserved for driver flip-flops and receiver latches.

14.2.16 ACLO Signal

The ACLO signal is asserted by bus ports whose existence in the system is necessary for the proper restart after a power failure (memory,bootstraps,etc.). The purpose of ACLO is to provide a restart signal to the CPU so that a system restart operation may be initiated.

14.2.16.1 Assertion Of ACLO - A bus port asserts ACLO asynchronous to the 2080 TTL-Bus clock during the assertion of the power supply AC LO at that bus port. The assertion of ACLO inhibits CPU'S from activating a power up routine.

14.2.16.2 Deassertion Of ACLO - ACLO is deasserted when all bus port required for the power up routine operation have detected that AC power and DC power are within specification. CPU'S sample the ACLO signal at an appropriate time after a power down routine (assertion of ACLO) to determine if a power up routine should be enabled.

14.2.17 DCLO Signal

The DCLO signal is asserted when an impending power failure to the clock circuit or bus terminating networks is detected. Bus port attached to the bus use this signal to prevent invalid data from being received while the bus is in an unstable state.

14.2.17.1 Assertion Of DCLO - The assertion of power supply DC LO to the clock circuit or terminating networks causes the assertion of DCLO. DCLO is asserted asynchronous to the 2080 TTL-Bus clock and occurs at least 2 microseconds before the clock becomes inoperable.

14.2.17.2 Deassertion Of DCLO - DCLO is deasserted when DC LO is deasserted at the clock circuit or terminating networks. The clock is operational for at least 2 microseconds before the deassertion of DCLO.

14.2.18 RESET Signal

Initialization of the system is accomplished via the RESET signal. This signal may only be asserted by a CPU or console and must be detected by all bus port attached to the 2080 TTL-Bus. The initialization operation should leave all devices in a known, well defined state. bus port which are not capable of performing an initialization in the period of the RESET pulse are required to not perform any commands until initialization is completed.

14.2.18.1 Assertion Of RESET - The CPU or console asserts RESET only when a console key (or sequence) has been selected or the program desires to master clear the system. The CPU or console which intends to assert RESET will first lock up the arbitration logic for a minimum of 15 2080 TTL-Bus cycles. The CPU or console continues to prevent any bus port from being granted the for the duration of RESET and for a minimum of 15 2080 TTL-Bus cycles after the deassertion of RESET. The lock up of arbitration insures that the 2080 TTL-Bus is inactive preceding, during and after the RESET condition. RESET is asserted at T0 time.

14.2.18.2 Deassertion Of RESET - The CPU or console will deassert the RESET signal at T0 time. The duration of the RESET pulse must be a minimum of 15 2080 TTL-Bus cycles.

14.2.18.3 Device Response To RESET - Each bus port must receive the RESET signal at PCLK time and begin an initialization sequence. Any current operation of short duration should not be aborted if that might leave the bus port in an undefined state. Bus Ports are forbidden to perform operations using the 2080 TTL-Bus during the assertion of RESET and must be in an idle state with respect to 2080 TTL-Bus activity at the conclusion of the RESET pulse. The initialization sequence should not cause a bus port to pass through any states which will inadvertently assert signals on the 2080 TTL-Bus. All READ commands issued before the RESET are cancelled and no data response should be expected. It is necessary to initialize the bus port from a read data expected state and to inhibit that particular FAULT conditions. All other FAULT status should be detected and when detected should cause FAULT to assert. In the event of a power failure during the RESET sequence, certain bus port are expected to assert the ACLO and/or DCLO signals. The initialization sequence should, however, cause the bus port to clear any existing ERROR status bits and to deassert the FAULT signal if it was asserted. Some of the bus port registers may contain new or old values. The contents of these registers after receipt of the deassertion of RESET must be defined in the appropriate bus port specification. It is recommended that bus port retain as much status data as possible to facilitate error analysis.

14.2.19 INTERLOCK Signal

The INTERLOCK signal provides coordination among CPU or other intelligent bus ports requesting INTERLOCK READ and WRITE RELEASE commands insuring exclusive access when required to data structures shared among CPU's and intelligent bus ports.

14.2.19.1 Assertion Of INTERLOCK - INTERLOCK is asserted synchronous with the 2080 TTL-Bus clock and in the same cycle as the acknowledgement of a INTERLOCK READ command. INTERLOCK remains asserted, reflecting the state of the INTERLOCK flip-flop, until the memory receives WRITE RELEASE command.

14.2.19.2 Deassertion Of INTERLOCK - Synchronous with the 2080 TTL-Bus clock and on the same cycle as the acknowledgement of a WRITE RELEASE command.

14.2.19.3 Bus Port Response To INTERLOCK - Memory bus port respond with (INTERLOCK) to any INTERLOCK READ. IO ADAPTER do not return data cycle for INTERLOCK READ commands if INTERLOCK is asserted before the IO ADAPTER is Granted the MBOX.

14.3 ELECTRICAL CHARACTERISTICS

14.3.1 Interface IC's

The 2080 TTL-Bus must be implemented using LSI components. The LSI implementation includes registers and parity networks common to all 2080 TTL-Bus port.

14.3.2 LSI Bus Interface

The LSI chip which interfaces the bus is the DEC 8646. The specifications of the 8646 are included in Appendix A.

14.3.3 Backpanel Characteristics

Backpanel impedance:	75 Ohms
Backpanel propagation delay:	2.0 nS/ft.
Backpanel crosstalk:	to be determined.

14.3.4 Connections

Module to backpanel: these connections should be as short as possible, in any case less than 10 centimeters. Not more than one IC may connect to any one bus line per interface. Backpanel to cable: this connection will be made as directly as possible. More complete specification awaits further work on packaging.

14.3.5 Terminations

The terminations will be physically at each end of the bus, and each will be active. For all but the clock signals, a voltage divider to +5 volts is used at both ends. For the clock signals, a voltage divider to -5.2 or 68 ohms to -2 volts is used at one end only.

14.3.6 Signal Voltages

For all signals except the four clocks, signal voltages will be nominally 2.9 volts and 0.7 volts for logical zero and logical one respectively. For the clocks, signal voltages will be nominally -1.6 volts and -0.8 volts for logical zero and logical one respectively.

14.3.7 Timing

The nominal clock period is 125 nanoseconds. T0 to T2 interval is centered on 93.65 nanoseconds and T2 to T0 interval is 31.25 nanoseconds.

14.3.8 Maximum Length

The maximum length of the 2080 TTL-Bus is 3 meters.

14.4 DIAGNOSTIC SIGNALS

To be provided.

CHAPTER 15

CONSOLE HARDWARE

15.1 SYSTEM CONSOLE - OVERVIEW

The system console is the point of interface between the operator, field engineer, or development engineer (during debug) and the 2080 computer system. Its function in life is to support the diagnostic philosophy, boot the system, and monitor the system environment of the DECSYSTEM-2080.

The console hardware consists of a DEC PDP-11 compatible instruction set microprocessor, 64k words of RAM, 4k words of ROM, four serial asynchronous I/O devices, an accurate interval timer that includes a time of century clock (with battery back-up), system clock control hooks (system clock start, stop, or half-step), diagnostic paths, some mass storage device interface path, environmental monitoring paths for monitoring both the power sub-system and the thermal environment, and at least 64 bytes of non-volatile CMOS RAM (with on board battery backup).

Coupled with the console specific software, the console subsystem functions as a virtual switch register and indicator panel. Numerous other functions are provided, these are described in the section 3.0 of this document.

This device also provides a path to the DECSYSTEM 2080 specific logic housed on the I/O Box module. The specific functions available are described in the I/O-Box subsystem specification but are described, briefly, here for ease of understanding. The I/O-Box houses a TTL-bus Monitor, Diagnostic Path Logic, and the Master Oscillator Control Logic.

THE FOLLOWING CONSTITUTES A PRELIMINARY BASIC BLOCK DIAGRAM:

```
                        +-----------------+
                        ! DIAGNOSTIC PATH !
                        ! THRU CPU/MEMORY !
                        +-----------------+
                              ^
+----------+            +----------+
!          !            !          !
!   M-BOX  :------------: I/O-BOX  :------------------------------
!          : MBUS (ECL) :          :      I/O-BUS (TTL)
!          :------------:          :------------------------------
+----------+            +----------+
                                           ^
                             ^             V
                             V             ^
                        +----------+   +---------+  "   "   "
                        ! CONSOLE  !   !  PORT   !
                        +----------+   +---------+
                         ^  ^  ^  ^        ^
                         ^  ^  ^  ^    +----------+
                         ^  ^  ^  ^    ! ADAPTER  !
                         ^  ^  ^  ^    +----------+
                         ^  ^  ^  ^
                         ^  ^  ^  L->+---------------------+
                         ^  ^  ^     ! CONSOLE LOAD DEVICE !
                         ^  ^  ^     +---------------------+
                         ^  ^  L---->+-----------------------+
                         ^  ^        ! ENVIRONMENTAL MONITOR !
                         ^  ^        +-----------------------+
                         ^  L------->+-----------------------+
                         ^           ! REMOTE DIAGNOSIS LINK !
                         ^           +-----------------------+
                         L----->+------------------+
                                ! CONSOLE TERMINAL !
                                ! LOCAL TERMINALS  !
                                ! CONTROL PANEL    !
                                +------------------+
        ---------------
        : F-11 PLUS    :
        :  MMU         :-------------------------
        :              :   INTERNAL CONSOLE BUS :
        :              :---------------V--------
        ---------------                V
                                       V
                                       V
                                       V
        --------------                 V
        :             :                V
        :  4 ASYNC    :----------------:
        :  LINES      :                :
        --------------                 :
                                       :       ---------------
                                       :-------:  I/O-BOX      :
                                       :       :  PATH LOGIC   :
                                       :       ---------------
        ------------                   :
        : MSU INTFC*:------------------:
        ------------                   :       ------------
                                       :       :  CONTROL    :
                                       :-------:  PANEL      :
                                       :       :  INTFC      :
       -------------------             :       ------------
       :    CONSOLE MEMORY :           :
       :    64K WORDS RAM  :-----------:
       :   6K WORDS ROM    :           :       +-----------+
       -------------------             :       ! ENVIRON-  !
                                       :-------! MENTAL    !
                                       :       ! INTFC     !
       -------------------             :       +-----------+
       : INTERVAL TIMER    :-----------:
       -------------------
```

* indicates DMA interface to the F-11

FIGURE 1 2080 SYSTEM CONSOLE BLOCK DIAGRAM

15.2 GOALS

The console subsystem goals are:

1. Meet DECSYSTEM-2080 RAMP goals.
2. Support Remote Diagnosis.
3. Allow for the implementation of a true, operator-less system.

4. Provide On-Line diagnosis capability.

5. Provide a console load device consistent with the system RAMP requirements and philosophy.

The following sections describe the specifications that must be met for the console subsystem to meet its goals.

15.3 POWER REQUIREMENTS

The console subsystem (the LC063 module) requires the following power:

+5 volts @ 5 amp. (or less)
+12 volts @ 1 amp. (or less)
-12 volts @ 1 amp. (or less)

The console subsystem load device (Mass Storage Unit) requires the following power:

((To be supplied))

15.4 ENVIRONMENTAL REQUIREMENTS

The console subsystem is designed as an integral part of the DECSystem 2080, and as such, is intended to operate in a DEC STD Class A environment. The console relies on the system packaging to assure that the applicable RFI/EMI standards are assured.

15.5 OPTIONS

There are NO Options available for the Console subsystem per se. The cables and terminal devices attached to the console communications ports are not part of the console subsystem.

15.6 PRE-REQUISITES

The following items are required for proper console subsystem operation:

* VT100 or equivalent video display terminal
* LA38 or equivalent hard-copy terminal
* Sufficient power and cooling

15.7 NEGATIVE SPECIFICATIONS

The following items are not part of the console subsystem goals/design criteria:

1. Software compatibility with existing PDP-11 based systems.

2. Compatibility with existing, off the shelf, pre-built operating systems.

Both of these items are precluded because of the optimization of hardware based addressing for the communications and other I/O devices used to achieve a single module console subsystem.

15.8 CONSOLE RAMP

The following data lists the RAMP goals for the 2080 console subsystem. These goals reflect the overall system RAMP goals.

1. Error detection and Error Logging

2. Remote Diagnostic

3. I/O-Bus monitor support (I/O Box resident transaction monitor with an historical ring buffer)

4. Console Memory Error Correction (soft error detection)

5. Console Memory Parity (classical PDP-11 parity)

6. Loopback Diagnostic capability for Console Communications lines

7. Temperature and Airflow sensing

8. Power Supply Monitoring

9. Module Isolation Diagnostics

10. Software support of the above listed features

11. Error Recovery

15.9 IMPLEMENTATION

The console subsystem RAMP goals are intimately tied to the overall 2080 System RAMP goals and RAS structure. In fact, the console is the focal point for implementing the various RAMP features that must be included in order to meet both customer and Digital Equipment

Corporation expectations.

15.9.1 CPU Error Detection And Error Logging

The internal structure of the 2080 CPU is a pipelined architecture. This structure lends itself to the detection of single bit errors through the use of odd parity on each nine bit portion of the data path. The partitioning of the data path is such that input has a parity check performed before an operation is accomplished. This method of partitioning allows the pipe to be stopped and the error occurrence to be recorded. Additionally, this allows the input to be properly recovered and retried.

The various logic functions within the 2080 CPU are tasked with informing the console subsystem of the occurrence of a parity error. A soft error, i.e. a transient parity error - one that does not occur every time data is fed into a function, can be corrected through the recovery of a correct copy of the erroneous input data. There are various intermediate level copies of Accumulator (AC) data including a Master AC copy, that are available for just this purpose.

The clock input for the failing logic function (perhaps the entire CPU - with the exception of the M-Box) is stopped in an orderly and proper fashion upon detection of a parity error. The console subsystem is tasked with restarting the clock in an orderly and proper manner.

The fact that an error has occurred is logged by the console subsystem, stored in the console load device, and, upon proper restart of the clocks, the console informs the CPU that there is error information that the operating system must forward to the system error log. This provides a redundant log of system error conditions but this is extremely beneficial, as will be shown in the next section.

15.9.2 2080 System Remote Diagnosis

The console subsystem provides an asynchronous communications channel that allows the use of remote system diagnosis. This communications channel allows Field Support to call the 2080 system and access the console. The remote diagnosis channel can be used to run device diagnostic programs that will determine if a module is in need of replacement. Through the use of this diagnostic tool, the remote diagnostic path, there will be a higher than previously experienced degree of certainty that the modules the Field Support Engineer brings to the ailing machine are the proper modules.

15.9.3 2080 I/O-Bus Monitor

To be Determined

15.9.4 Console Memory Parity

The console subsystem local memory has memory parity implemented. This allows the detection of any single bit error. Memory transfers automaticly generate and check parity - odd parity of course. This error detection scheme coupled with the soft error correction method detailed in 2.1.5, provide a console memory that is capable of single bit error detection and correction.

15.9.5 Console Memory Error Correction

The console subsystem local memory has an error correction mechanism built into the console resident software. This software algorithm consists of a series of check words that when coupled with a CRC type routine provide the isolation to the single bit error level. The method of backing up this error correction scheme is the console reboot from the console load device. The console reboot sequence does NOT require 2080 CPU intervention to be accomplished. This means that the console can recover from soft errors with out impacting the system performance. It is possible for the console to reload a bad block of memory (perhaps 256 word block) with out reloading the whole console memory.

15.9.6 Loopback Capability

All console devices have loopback capability. The disk interface, the communications channels, and the I/O-Box interface can all be looped back on them selves to turn typical Write Only or Read Only paths into Read/Write paths. This Loopback capability is extended in the I/O-Box to include the TTL-Bus, the Ports (under RAM loadable control) and the M-Box path.

15.9.7 Environmental Monitoring

The console subsystem can monitor the physical environment to determine the presence of over temperature or out of voltage specification conditions.

15.9.8 Diagnostics

The 2080 System diagnostics provide true module failure isolation. The basic architecture of the CPU is such that fault conditions can be detected and with some moderate support from the console subsystem, the failing unit can be determined.

All of the features described in 2.1 are supported by the console monitor software supplied by the diagnostic engineering group. This software is resident on the console load device.

15.10 SYSTEM CONSOLE DESCRIPTION - HARDWARE

The DECSYSTEM-2080 console subsystem is based on the Digital Equipment Corporation F-11 chip set. This chip set microprocessor implements the full PDP-11/34 instruction set. As implemented, the F-11 has a maximum addressing capability of 124K words of memory and 4K words of I/O space. This addressing capability is implemented as 64K words of RAM, 4K words of PROM, and 4K words of ROM-I/O. The subsystem includes a DMA disk type of mass storage unit (MSU) interface, four programmable asynchronous communications channels, an interval timer, a time of century clock with non-volatile RAM (with built in battery backup), a DEC MMU - memory management unit, and a parallel I/O channel to the I/O Box.

This subsystem can be viewed as a simple, bounded PDP-11 computer system. The console subsystem executes the PDP-11 instruction set, handles interrupts in the classical PDP-11 manner, addresses I/O in the classical PDP-11 manner, and has the typical PDP-11 programmable/ jumperable features hardwired. Interrupt vectors and priorities are hard wired in this subsystem because there are no variations to the standard console subsystem.

15.11 F-11 DESCRIPTION

The F-11 is the DEC nick-name for the second generation LSI-11 micro processor chip set that executes the Digital Equipment Corporation PDP-11 instruction set. This chip set includes a standalone ISP (Instruction Set Processor) chip, an MMU (Memory Management Unit) chip, and an FPP (Floating Point Processor) chip. The DECSYSTEM-2080 utilizes the ISP chip (known as the Dat/Ctl chip) and the MMU chip to implement the Console Subsystem's console processor.

15.11.1 CONSOLE PROCESSOR

The ISP or Dat/Ctl chip recognizes and executes the PDP-11/34 instruction set, supports four priority vectored interrupt levels, DMA, addresses up to 28K of memory, and supports the classical PDP-11 I/O page. The MMU or Memory Management Unit chip expands the addressing space in the same manner as the PDP-11/34 MMU but the addressing is limitted to the on board memory: 64K words of RAM, 4K words of PROM in the memory address area; 2K words of PROM in the I/O address area, and the I/O device addresses implemented in the upper 2K words of the I/O area.

15.11.2 CONSOLE MEMORY

The memory mapping of the 2080 Console Subsystem provides for accessing the 70K Words of on board memory (68K Words in the Memory Page, and 2K Words in the I/O Page) partitioned as shown in the following physical address map:

PAGE	F-11 PHYSICAL ADDRESS (OCTAL)		FUNCTION
M	000000-377776	(word boundary)	RAM (addressable)
E	400000-737776	(word boundary)	NON-EXISTENT
M	740000-757776	(word boundary)	4K ROM
	* * * * * * * * * * * * * * * * * * * *		
I	760000-767776	(word boundary)	2K ROM
O	770000-777776	(word boundary)	I/O Devices

Notice that the ROM memory overlaps the memory and I/O space so that the top 4K words of memory (4K ROM) are accessible only when the MMU is enabled. In this manner, the necessary power-up routines can always be accessed while the routines in the 4K ROM can be accessed only when the system MMU is enabled. Additionally, although the MMU provides the capabilities to address substantially more memory than is implemented on the console processor, there are no provisions for addressing memory that is not on-board.

The Console Memory includes memory parity and a memory parity error address latch. Parity is computed on data being written into memory and checked on data being read from memory. The parity circuit is intended to assure data integrity in the Console Memory. It is not intended to provide bus transaction parity between the F-11 CPU and I/O devices. The CMR1 (Console Memory Register1) is a read write register that allows the parity circuit to be enabled or tested and contains the the Error Flag and the high order memory address bits. The CMR2 is a read write register that is capable of latching the low order 16 bits of memory parity error address.

15.12 MSU INTERFACE DESCRIPTION

The MSU (mass storage unit) interface provides a DMA (Direct Memory Access) path between the MSU device (a disk) and the Console subsystem resident memory. The DMA operations are enabled by the console processor software. The Console subsystem MSU device contains all the required microcode, system boot code, diagnostics, etc. that is necessary to bring up the hardware portion of the system. System software is not available on the console MSU device.

The following information characterizes the console subsystem mass storage unit operational capabilities:

1. DMA path to the console memory

2. Full 18 bit memory address support.

3. Byte transfers only (word transfers not supported).

4. Byte count support for a maximum of 64K bytes per transfer (Hardware limitation).

5. Byte count and transfer address fully programmable.

The MSU disk interface consists of the following sixteen bit registers:

Mnemonic	Function
DCSW	- This is the disk Control/Status register Writable (WRITE ONLY).
DCSR	- This is the disk Control/Status register Readable (READ ONLY).
DBCR	- This is the disk Byte Counter register Readable (READ ONLY).
DACR	- This is the disk current Address Counter register Readable (READ ONLY)
DDTB	- This is the disk Data Transfer Buffer (Writable and Readable).
DCAW	- This is the disk Current Address holding register Writable(WRITE ONLY)
DBCW	- This is the disk Byte Count holding register Writable (WRITE ONLY).

Address	Register Mnemonic and Key
771100	DCSW - Control/Status Writable
771102	DCSR - Control/Status Readable
771104	DBCR - Byte Count Readable
771106	DACR - Address Counter Readable
771110	DDTB - Data Transfer Buffer
771112	DCAW - Current Address Writable
771114	DBCW - Byte Count Writable
771116	RESERVED

Register Bit descriptions:

Register	Bit	Function
DCSW/DCSR	0:2 8:10	Mode control for the DMA address, byte counter and done flag. Refer to the programming section for specific details on implementations.
	3	Interrupt Enable
	4:5	Address bits 16 and 17, Extended Memory Address
	6:7	reserved
	11:14	reserved
	15	Interrupt Flag
DCAW/DACR	15:0	These registers contain the address currently being used for the storage or retrieval of disk data. This represents the byte address.
DBCW/DBCR	15:0	These registers contain the byte count for data transfers.
DDTB	11:0	This is the register used for the actual transfer of data to or from the disk. Data is transfered in byte mode.

15.13 SLU - SERIAL LINE UNIT - DESCRIPTION

The ZILOG Z80-SIO chip is used to implement the Serial Line Units for the 2080. The 2080 Console subsystem implements four serial line units. These SLU's are serial asynchronous EIA RS232C lines. The line speed is programmable for speeds between 50 baud and 19.2 kilobaud. The character format for the transmitter and receiver must be the same. The functional allocation is as follows (The addresses below are restricted to the low byte address exactly as shown):

1. CTY - Bus Address 776000 (channel A of SIO 0)

   Interrupt Priority 5

   Interrupt Vector 300

   This is used as the local copy console terminal. The line speed is programmable for both the transmitter and receiver. Split line speed is supported (i.e. the transmitter and receiver can be programmed to run independently).

2. REMOTE LINE - Bus Address 776010 (channel B of SIO 0)

   Interrupt Priority 5

   Interrupt Vector 300

   This line is used for KLINIK/APT, line speed is programmable and split line speed operation is supported.

3. LOCAL LINE 1 - Bus Address 776020 (channel A of SIO 1)

   Interrupt Priority 5

   Interrupt Vector 310

   This line is used for local applications. The line speed is programmable and split speed operation is supported.

4. LOCAL LINE 2 - Bus Address 776030 (channel B of SIO 1)

   Interrupt Priority 5

   Interrupt Vector 310

   This line is used for local applications. The line speed is programmable and split speed operation is supported.

The following table details the programmable baud rate generator addresses:

Bus Address	Function
776004	CTY TX/RX BAUD RATE SEL (SIO0 A) - Split baud rate (8 bit register)
776014	REMOTE LINE TX/RX BAUD RATE SEL (SIO0 B) - Split baud rate (8 bit register)
776024	LOCAL LINE 1 TX/RX BAUD RATE SEL (SIO1 A) - Split baud rate (8 bit register)
776034	LOCAL LINE 2 TX/RX BAUD RATE SEL (SIO1 B) - Split baud rate (8 bit register)

The Remote Line supports the following modem control signals:

1. Protective ground
2. Transmit Data (Output)
3. Receive Data (Input)
4. Request To Send (Output)
5. Clear To Send (Input)
6. Data Terminal Ready (Output)
7. Data Carrier Detect (Input)
8. Signal Ground
9. Data Set Ready (Input)

15.14 INTERVAL TIMER DESCRIPTION

The console Interval Timer is a programmable interrupt timer and Time of Day clock. This logic circuitry has an on board (or, on module, to be exact) battery backup. The function consists of an oscillator, an interval timer chip, 256 bytes of ram, and an interval timer output, all done in CMOS logic.

The device appears to be xx bytes of memory in the I/O space. This is a byte addressable device only. The first 12 bytes are used for the clock bookkeeping while the remaining 244 bytes are available for the user. The RAM addresses are:

Address	Function
770000	Seconds
770001	Seconds alarm
770002	Minutes
770003	Minutes alarm
770004	Hours
770005	Hours alarm
770006	Day of week
770007	Date of Month
770010	Month
770011	Year
770012	Control Reg 1
770013	Control Reg 2
770014 To 770377	User area

The clock circuit with its internal registers is found in addresses:
770400 To 770477 Reserved for the Clock chip itself.
The TCC circuit has an interrupt priority level of 6 with an interrupt vector of 100.

15.15 I/O BOX INTERFACE DESCRIPTION

This interface consists of a series of 16-bit registers (a maximum of 192) that reside on the I/O-Box. The register definitions are as described in the 2080 I/O-Box specification. The register addresses listed below are reserved for the I/O-Box use:

770500-770576	I/O-Box internal registers
770600-770676	System Clock controller Registers
770700-770776	Diagnostic control registers

15.15.1 INTERNAL REGISTERS

The I/O-Box Internal Registers are used to communicate with the 2080 system CPU. The registers consist of the translation of the CSL data bits into the appropriate 2080 bit as shown below:

CSL BIT	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
I/O-BOX																
REG Right	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
REG Middle	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
REG Left	xx	xx	xx	MT	MT	MT	MT	MT	xx	C	D	P	00	01	02	03

The addresses for these registers are:

ADDRESS MNEMONIC	FUNCTION
770500 - CREGR	This is the right most 16 bits of the 2080 MB interface register. The MB interface register is the means for translating the 16 bit console data into the 36 plus bits necessary to talk to the 2080.
770502 - CREGM	This is the middle 16 bits of the 2080 MB interface register.
770504 - CREGL	This is the left most 16 bits of the 2080 MB interface register.
770506 - DREGR1	Right most 16 bits of the Data REGister 1
770510 - DREGM1	Middle 16 bits of the Data REGister 1
770512 - DREGL1	Left most 16 bits of the Data REGister 1
770514 - DREGR2	Right most 16 bits of the Data REGister 2
770516 - DREGM2	Middle 16 bits of the Data REGister 2
770520 - DREGL2	Left most 16 bits of the Data Register 2
770522 - DREGR3	Right most 16 bits of the Data REGister 3
770524 - DREGM3	Middle 16 bits of the Data REGister 3
770526 - DREGL3	Left most 16 bits of the Data REGister 3
770530 - DREGR4	Right most 16 bits of the Data REGister 4
770532 - DREGM4	Middle 16 bits of the Data REGister 4
770534 - DREGL4	Left most 16 bits of the Data REGister 4

A word of data being sent to the 2080 CPU would require the console to write CREG_, and then DREG__. The DREGL_ must be written last. A four word write would require the writing of the CREG_ followed by the writing of the twelve DREG__'s.

15.16 2080 SYSTEM CONTROL PANEL DESCRIPTION

The 2080 system Control Panel consists of five switches, two LED indicators and an alphanumeric LED display. The control panel is used to select which of the local terminals is the CTY, enable KLINIK, control system On/Off, and provide control for various other functions that will be specified at a later date. The LEDs are used to inform the operator that the system is running by displaying the time of day and any other information that may be appropriate. The following addresses are reserved for the control panel:

771100 - 771377

The following switches are resident on the control panel:

1. CONTROL -This is a four position key switch.
   Position 1: TBD
   Position 2: Disable KLINIK Link
   Position 3: Enable User KLINIK Link
   Position 4: Enable Cty KLINIK Link
2. TBD
3. TBD
4. TBD
5. TBD

These were decided on months ago!

The following LED's are resident on the control panel:

1. TTL ON - This LED indicates that there is TTL power available to this LED, and hopefully, to the Console.
2. KLINIK ENABLE - This LED indicates that the CONTROL switch is in key position 3 or 4.

15.17 ENVIRONMENTAL MONITOR

The system environmental monitor consists of various thermal sensing mechanisms located in the machine (not on the actual 2080 logic modules) and various sensors located in the power subsystem. The address range 771500 - 771777 is reserved for the environmental monitoring interface.

15.18 SYSTEM CONSOLE - PROGRAMMING INFORMATION

The system console interfaces to the 2080-TTL bus as an invisible type of port, that is, it uses the TTL bus as the path from the console to the CPU. Using what is, essentially, a subset of the basic I/O port protocol, the console can request interrupt service, pass information to the CPU via the EPT, access memory at the discretion of the memory controller, etc.

The console can also use the TTL bus to pass information to, and receive information from, any of the ports. The console also has access to the I/O-bus arbitration logic, and, with such access it can initiate a memory to port transfer. This can be done through the simulation of the doorbell function provided by the E-BOX. The console can ring the bell of a given port after the console has loaded the 2080 memory and EPT with the proper transaction data. The E-BOX path to the ports is through the I/O-Box and so is the console path to the rest of the system. This allows the I/O-Box to be designed in such a manner as to allow the console to access all of the functions available to a port, E-BOX, or M-BOX. A certain amount of caution will be exercised in the design of this area of the console and I/O-Box in order to preclude the possibility of a paranoid console (console has access to the machine - so - take over the busses).

The console is notified of the fact that the 2080 CPU has data for it by the use of specific EPT locations. The actual software protocol used to read and write the EPT locations used for the console will be detailed in the 2080 console diagnostic specification and will be excerpted and included in this specification as Appendix A.

15.19 F-11 - CONSOLE PROCESSOR

The F-11 implementation used on the console subsysten does not have the capability of running existing DEC software without modification. The standard processor tests, memory tests and so on will run but the I/O routines for passing information from the operator to the cpu or vice versa will not work because of the fact that there is no DL-11 type device present at the classical DL-11 address.

15.20 MSU - MASS STORAGE UNIT

Upon determination of the type of device used this section will be completed.

15.21 SLU - SERIAL LINE UNITS

The SLU's are implemented using the ZILOG Z80-SIO dual channel communications device. The Z80-SIO's are implemented as word addressable devices. Any attempt to write into the high byte of the SLU addresses is ignored while the low byte is accepted in word or byte mode operations. The SIO device is similar to some typical communications devices available on the Unibus (DQ-11, DMC-11, or KMC-11) in that there are actually registers hidden inside the device. The device appears to have two registers resident on the bus while in actuality the first address or CSR (Control and Status Register) is used as a pointer register.

The SIO register addressing scheme allows the internal registers to be selected by setting up the CSR register with the internal register pointer. The SIO internal register is then read by accessing the SLU Data register.

Each SLU's CSR (register addresses 776002, 776012, 776022, and 776032) or Control and Status Register has the following bit definitions:
D<2:0> REGISTER SELECT POINTER address (0 to 7 octal) bits

D<5:3> Command Select bits

D<7:6> Extended Command bits

The read and write functions appear to access different combinations of status information lumped together in a register. Write commands access up to 8 registers (including Reg 0) while the read commands access up to 3 registers.

15.21.1 Register Descriptions

The registers are defined as follows:

Write Reg	Function
0	Pointer and command
1	Interrupt control register
2	(REMLIN and LOCLIN2 only) Interrupt Vector register
3	Receive Mode Reg
4	Clock Mode Plus reg
5	Transmit Mode reg
6	Channel A sync reg (CTY or LOCLIN1 - not used)
7	Channel B sync reg (REMLIN or LOCLIN2 - not used)

Read Register	Function
0	Flag Register
1	Extended Flag Register
2	(REMLIN and LOCLIN2 only) Interrupt Vector register

The following information describes the internal SLU registers:

Write Reg 0

Bit	Title	Function
0 1 2	PNT 0 PNT 1 PNT 2	These bits are used to select the destination register for the next action: for a write, the next byte is written into the register selected by the pointer; for a read, the next read access is to the register selected by the pointer.
3 4 5	CMD 0 CMD 1 CMD 2	These bits are encoded to provide 8 commands to the SIO: 0 - null command, 1 - Send Abort (SDLC mode only not used by console), 2 - Reset external/status Interrupts. 3 - channel reset, 4 - reset receive interrupt on first receive character, 5 - reset transmitter interrupt pending, 6 - error reset, 7 - return from interrupt.
6 7	CRC 0 CRC 1	These bits are used to encode four CRC logic reset commands: 0 - Null command, 1 - reset RX CRC checker, 2 - reset TX CRC generator, 3 - reset Sending CRC/Sync latch. Only the null command is used by the console.

WRITE REG 1

0	EIE	External Interrupt Enable allows modem control status changes to cause an interrupt.
1	TIE	Transmit Interrupt Enable allows TBMT interrupts
2	SAV	Status Affects Vector allows the Interrupt Status to be encoded into the interrupt vector. This allows the various interrupt conditions to point to the proper service routine through the use of a dispatch table.
3	RIM 0	Rx Interrupt Mode bits provide and encoded command that
4	RIM 1	controls the action taken on a Receiver Interrupt: 0 - Rx Interrupts Disabled. 1 - Rx interrupt on first character only. 2 - Interrupt on all received characters, Parity error affects vector. 3 - Interrupt on all received characters, Parity Error does not affect Vector.
5	WROT	?
6	REN	?
7	W/RE	?

WRITE REGISTER 2

This is the internal SIC interrupt vector register. This is a programmable register that can be used hold a basic pointer for a dispatch table. Coupled with the assertion of the SAV bit (bit 2 of Write Reg 1), the various interrupt causing conditions can be directed to the proper subroutine. For example, if a dispatch table were constructed with the following information:

```
XXX X00 Base address - pointer for Remlin TBMT service routine
XXX X02 Base address +2 - Pointer for Remlin Modem control service routine
XXX X04 Base address +4 - pointer for Remlin Receive data available routine
XXX X06 Base address +6 - pointer for Remlin special receive condition
                              routine
XXX X10 Base address +10 - pointer for CTY TBMT service routine
XXX X12 Base address +12 - pointer for CTY Modem control routine (not used)
XXX X14 Base address +14 - pointer for CTY receive data avail routine
XXX X16 Base address +16 - pointer for CTY special receive condition routine
```

The same scenario would apply to LOCLIN1 and LOCLIN2. This is a programmable 8 bit register that does not generate the PDP-11 type interrupt vector but rather provides a handy dandy register to store the low byte pointer for the dispatch table (a programmers imagination can be counted on to achieve other uses for this mechanism).

WRITE REGISTER 3

0	RECEN	Receiver Enable.
1	SYNLDIN	Not used
2	ASM	Not used
3	RCRCE	Not used
4	EHM	Not used
5	AUTO	Automatic mode enable. DCD and CTS become the enable signals for RX and TX respectively.
6 7	REC 0 REC 1	Receiver Bits per Char select: 0-5 bits, 1-7 bits, 2 - 6 bits, 3 - 8 bits per character.

WRITE REGISTER 4

0	PAREN	Parity Enable
1	PE/-O	Parity Even (when set)/ Parity Odd (when clear)
2 3	SB 0 SB 1	Stop bit select: 0 - not used. 1 - 1 stop bit per character. 2 - 1.5 stop bits per character. 3 - 2 stop bits per character.
4 5	SYMC 0 SYMC 1	Not used.
6 7	CLCK 0 CLCK 1	Clock divisor select. Always set to 1 (01) for 2080 use. 0 - X1 clock, 1 - X16 clock, 2 - X32 clock, 3 - X64 clock.

WRITE REGISTER 5

0	TCRCE	Tx CRC Enable - not used. Set to 0.
1	RTS	Request To Send. Set = ON, Clear = OFF.
2	-SDLC	Not Used. Setting does not matter
3	TX EN	Transmitter Enable.
4	BREAK	Send BREAK character. The Break character is sent for as long as this bit is true.
5 6	TBC 0 TBC 1	Tx Bit Count: 0 - 5 bits (or less), 1 - 7 bits, 2 - 6 bits, 3 - 8 bits per character.
7	DTR	Data Terminal Ready. Set = ON, Clear = OFF.

WRITE REGISTER 6 Not used.

WRITE REGISTER 7 Not used.

READ REGISTER 0

0	RCA	Rx Character Available. Commonly called Receive

		Data Available.
1	IP	Interrupt Pending. Available only on CTY and LOCLIN1. Always set to 0 in REMLIN and LOCLIN2.
2	TBMT	Transmitter buffer empty.
3	DCD	Shows the state of DCD signal at time of last external interrupt or following execution of command 2.
4	S/H	Not used in 2080. State indeterminate, must be ignored.
5	CTS	Shows state of CTS as in DCD above.
6	SC/S	Not used in 2080. State indeterminate, must be ignored.
7	B/A	Break character detected.

READ REGISTER 1

0	ALSENT	Used for diagnostic purposes to determine that the Transmit shift register is empty. (TBMT true coupled with this bit indicates that the Transmitter is DONE).
1	RC 0	
2	RC 1	
3	RC 2	
4	PE	
5	ROE	
6	C/FE	
7	EOF	

READ REGISTER 2
((More Data will be provided))

15.22 TOC - TIME OF CENTURY CLOCK

((To be provided upon selection of TOC chip))

15.23 SYSTEM CONSOLE - I/O BOX INTERFACE PROGRAMMING

The I/O-BOX interface consists of up to 256 16-bit registers that can be read or written by the console processor. The actual register descriptions and addresses are described in section 3.5. The I/O-Box includes the 2080 Diagnostic paths, the console to CPU path through the I/O box, the console to TTL-Bus PORT path, and the Environmental monitoring logic. The interrupt request level presented to the console sub-system from the I/O Box is:

Level 4 - CPU to Console interrupt request.

15.23.1 System Console - IOBCX Transaction Protocol

((THE FOLLOWING DATA REPRESENTS A PROJECTED SOFTWARE IMPLEMENTATION OF THE SUB-FUNCTIONS NECESSARY TO ALLOW THE CONSOLE TO COMMUNICATE WITH THE REST OF THE 2080 SYSTEM. THERE ARE NO HARDWARE FEATURES -CROCKS- THAT PRECLUDE THE IMPLEMENTATION OF THE FULL TTL-BUS PROTOCOL!!))

The following is the subset of the basic COMMAND/ADDRESS formats that can be used by the system console (note that this is preliminary data and subject to change upon mastication). There is absolutely no hardware restriction that precludes the expansion of the following protocol into the full TTL-BUS protocol defined in the the 2080-TTL bus specification.

```
  COMMAND/ADDRESS FORMAT

+----+--------------------------------+
!    !                                !
+----+--------------------------------+
F<0:3>    A<12:35>

  COMMANDS
F<0:3>
0000 MEMORY READ - FOUR WORDS
0001 EPT READ - FOUR WORDS
0010 MEMORY READ - ONE WORD
0011 EPT READ - ONE WORD
1000 MEMORY WRITE - FOUR WORDS
1001 EPT WRITE - FOUR WORDS
1010 MEMORY WRITE - ONE WORD
1011 EPT WRITE - ONE WORD
```

COMMAND CODES
FIGURE 2

The 2080 I/O bus defines four command/address functions. of these four functions, only two are supported by the system console: READ and WRITE. The READ and WRITE functions have subfunctions that are supported by the system console: READ one or four words from memory, READ one or four words EPT, WRITE one or four words to memory, WRITE one or four words to EPT. These functions are detailed below. The READ functions are indicated by the state of bit <0> of the command function - when bit <0> is zero, the function is a READ. The subfunction is indicated by the state of bits <2:3>. Bit <3> set indicates an EPT subfunction while bit <3> clear indicates a memory subfunction. bit <2> clear indicates a four word transfer operation while bit <2> set indicates a one word transfer operation.

15.23.1.1 MEMORY READ FUNCTION - Encoded into f<0:3> as 0000/0010, the Memory READ function requests that the M-BOX retrieve the 36 bit data in the memory location specified by the address given in bits <12:35> of the Command/Address word. The M-BOX decodes the command in bits <0:3> and transfers either one or four words to the I/O-BOX. The I/O-BOX buffers the words and transfers them to the console device. The memory address specified in bits <12:35> of the Command/Address word specifies the starting address of a four word transfer if a four word function has been selected. In the event the selected address can not be read because of a non-recoverable data error, that is one that can not be corrected, the UNCOR signal is sent to the console.

15.23.1.2 EPT READ FUNCTION - Encoded into F<0:3> as 0001/0011, the EPT READ function requests that the M-BOX retrieve the 36 bit data in the EPT location specified in the address bits <12:35> of the Command/Address word. Note that bits <12:26> are translated by the M-BOX to reflect the current EBR contents and thus access the proper EPT location. The M-BOX decodes the command in bits <0:3> and transfers either one or four words from the EPT location specified in Command/ Address word to the I/O-BOX. The I/O-BOX buffers the word or words of data and then transfers them to the console. in the event of a transaction error, the M-BOX informs the console through the assertion of the UNCOR signal line.

15.23.2 WRITE FUNCTIONS

The WRITE functions are indicated by the state of bit <0> of the command function - when bit <0> is a one, the function is a WRITE. The subfunction is indicated by the state of bits <2:3>. Bit <3> set indicates an EPT subfunction while bit <3> clear indicates a memory subfunction. Bit <2> clear indicates a four word transfer operation while bit <2> set indicates a one word transfer operation. The 36 bit data plus parity is buffered by the I/O-BOX before it is transferred to the M-BOX (memory or ept) locations.

15.23.2.1 MEMORY WRITE FUNCTION - Encoded into F<0:3> as 1000/1010, the memory WRITE function requests that the M-BOX allow the subsequent 36 bit word or words to be jam transferred from the console to the memory address specified by the address given in bits <12:35> of the Command/Address word. The I/O-BOX buffers the 36 bit data word or words (along with the parity bits) until the M-Box generates a grant that allows the transfer to take place.

15.23.2.2 EPT WRITE FUNCTION - Encoded into F<0:3> as 1001/1011, the EPT write function requests that the M-BOX allow the jam transfer of 36 bit data from the console to the EPT location/locations specified in bits <12:35> of the Command/ Address word. Note that bits <12:26> of the Command/Address word are translated by the M-BOX to reflect the current EBR contents and thus allow the data to be written into the proper EPT address.

15.24 REFERENCE CHART

The following data attempts to provide a quick reference for the addressing, vector address and interrupt priority of each device on the internal console subsystem bus.

BUS ADDRESS	FUNCTION (READ/WRITE)	COMMON MNEMONIC	INTERRUPT PRIORITY	INTERRUPT VECTOR
777 776	PROCESSOR STATUS WORD			
777 774	STACK LIMIT			
777 772	PROGRAM INTERRUPT REQUEST			
777 770	:			
. .	} DEC RESERVED			
777 720	:			
777 717	USER	R6 (SP)		
777 716	SUPERVISOR	R6 (SP)		
777 707	:	R7 (PC)		
777 706	KERNEL	R6 (SP)		
777 705	}	R5		
777 704	}	R4		
777 703	} GENERAL	R3		
777 702	} REGISTER	R2		
777 701	} SET 0	R1		
777 700	}	R0		
777 656	|			
777 654	|			
777 652	|			
777 650	} USER ACTIVE PAGE			
777 646	} ADDRESS REGISTERS			
777 644	|			
777 642	|			
777 640	|			
777 616	|			
777 614	|			
777 612	|			
777 610	} USER ACTIVE PAGE			
777 606	} DESCRIPTOR REGISTERS			
777 604	|			
777 602	|			
777 600	|			
777 576	} STATUS REGISTER 2			
777 574	} STATUS REGISTER 1			
777 572	} STATUS REGISTER 0			

776 Ø36	reserved					
776 Ø34	Baud Rate Gen	R/W				
776 Ø32	Cont/Stat Register	R/W				
776 Ø3Ø	Data Register	R/W	LOCLIN2	5	31Ø	(SLU 1 Chan B)
776 Ø26	reserved					
776 Ø24	Baud Rate Gen	R/W				
776 Ø22	Cont/Stat Register	R/W				
776 Ø2Ø	Data Register	R/W	LOCLIN1	5	31Ø	(SLU 1 Chan A)
776 Ø16	reserved					
776 Ø14	Baud Rate Gen	R/W				
776 Ø12	Cont/Stat Register	R/W	REMLIN			
776 Ø1Ø	Data Register	R/W	REMLIN	5	30Ø	(SLU Ø Chan B)
776 ØØ6	reserved					
776 ØØ4	Baud Rate Gen	R/W				
776 ØØ2	Cont/Stat Register	R/W	CTY			
776 ØØØ	Data Register	R/W	CTY	5	30Ø	(SLU Ø Chan A)
772 356						
772 354						
772 352						
772 35Ø	USER ACTIVE PAGE					
772 346	ADDRESS REGISTERS					
772 344						
772 342						
772 34Ø						
772 316						
772 314						
772 312						
772 31Ø	USER ACTIVE PAGE					
772 3Ø6	DESCRIPTOR REGISTERS					
772 3Ø4						
772 3Ø2						
772 3ØØ						
771 777	Environmental					
: :	Monitor					
771 4ØØ	Interface	R/W	ENMO	7	35Ø	
771 377	Control					
: :	Panel					
771 1ØØ	Interface	R/W	CPI	6	34Ø	
771 Ø77	Mass Storage					
: :	Unit					
771 ØØØ	Interface	R/W	DSK	4	33Ø	
77Ø 776	I/O Box					
: :	Interface					
77Ø 5ØØ	Registers	R/W	IOB	4	32Ø	

770 477	} Time of				
: :	} Century				
770 400	} Interface	R/W	TOC	6	100
770 377	} Non-Volatile				
: :	} Battery-Backed				
770 000	} RAM	R/W	NVRAM	none	none
767 776	} I/O Page				
: :	} Resident				
760 000	} ROM	READ ONLY		none	none
------	} Power Fail	READ ONLY		?	024
------	} Bus Error				004

CHAPTER 16

CONSOLE SOFTWARE

16.1 PRODUCT OVERVIEW

16.1.1 Product Abstract

The 2080 console subsystem serves as:

1. The operating system's controlling terminal(s). When timesharing is running, the console provides the system CTY link, monitors environmental conditions, and provides support for soft error recovery and error logging.

2. The computer system's hardware console. It can be used for the operation and control of the hardware in the system. Functions include system bootstrapping, power fail restart, system initialization, and acting as the lights and switches of the machine.

3. The system's diagnostic console. It controls loading, scripting and execution of diagnostic programs, and serves as a hardware and software debugging tool.

This document details the functionality of the 2080 console subsystem and the associated software tasks necessary to implement that functionality.

The console subsystem software will consist of three segments of code:

1. The "hard-core" console (KCROM) is blasted in EPROM. This may be up to 12KB of memory, always available. This segment of software contains:

   * routines to test the basic hard-core hardware in the console.

* routines to load the "loadable" console or console diagnostics into console RAM (see (2) below).
* an ASCII console terminal handler and command parser.
* routines to handle the required KLINIK/APT functionality.

When the "hard-core" console is in control, the prompt 'ROM>' will be typed on the console terminal(s).

2. The "loadable" console (KCCON) is the next 38KB of F11 program loaded into the F11 RAM. This 38KB "loadable" console plus the 12KB of "hard-core" console make up what is referred to as the "resident" console. This "loadable" console provides:

   * an extensive set of 2080 CPU specific functions.
   * a more sophisticated (TOPS-20 like) terminal handler and command parser than the "hard-core" console.
   * a general purpose subroutine package for use by the console-based diagnostics and the Run-time Support Program (see (3) below).
   * routines for bootstrapping of the entire 2080 system.

   The memory segments containing the "loadable" console are mapped as read-only using the F11 memory management option. When the "loadable" console is in control, the prompt 'CON>' will be typed on the console terminal(s).

3. The Run-time Support Program (KCRSP, called the RSP hereafter), is loaded into a portion of the rest of the available console RAM. It forms the topmost (third) level of software in the console subsystem. The RSP will recognize all commands recognized by the "loadable" console plus some additional ones. It will provide the same sophisticated terminal handler and command parser as the "loadable" console. The RSP implements the additional functions and communication protocols necessary to run an operating system or any stand alone exec mode program. When the RSP is in control, the prompt 'RSP>' will be typed on the console terminal(s).

During hardware debug, or the manufacturing process, the RSP may be replaced by a console-based 2080 diagnostic. Any such diagnostic would act as the top most level of the console software, and command parsing would be the same as described for the RSP. In general, command parsing shall be consistent for all levels of command decoding except for the "hard-core" console.

The software shall basically be an upgrade of the KL10 front end KLDCP. Code for new and different front end devices will be added to the program. Code for obsolete and non-existent devices will be removed. New functions dealing with the 2080 CPU will be provided and a new command parsing scheme will be provided. As an upgrade of KLDCP, the 2080 Console Program will be written in PDP11 assembly language and assembled under MACY11.

16.1.2 Product Audience

The console software will be used by hardware engineers, field service personnel, manufacturing personnel, system programmers, diagnostic programmers, and end-user operators, administrators, and programming personnel. In short, anyone who uses the 2080 console terminal(s).

16.2 PRODUCT GOALS

16.2.1 Performance

1. The console processor will drive the APT line at 19.2Kb. A VT100 or hardcopy terminal may be attached to 19.2Kb lines (Max speed, programmable by the console), and will be driven at the maximum rate whenever possible.

2. The console command language will be easy to learn and use. It will conform to the DEC Standard Command Language guidelines and will implement full recognition and prompting.

3. System booting and power-fail restart will occur without operator intervention.

4. The time to perform the console hardcore tests will be less than 30 seconds.

5. The console will act as the light and switches of the 2080 system.

16.2.2 Environments

The console software assumes the following console hardware configuration:

1. A FCNZ11 (F11) with memory management for the console MPU.

2. Two console terminals - a hardcopy terminal (an LA34, LA120 or equivalent) and a video terminal (a VT100 or equivalent ANSII standard terminal).

3. Eight asynchronous lines. One of these lines must provide full modem control and variable baud rates to at least 19.2KB in order to meet the KLINIK/APT requirements. One of these lines must implement EIA standard RS-422 and the other seven must implement EIA standard RS-232-C.

4. 128KB of RAM.

5. 12KB of EPROM.

6. An RX04 Floppy Disk for the console load device. The powering on and off of this device must be programmable by the console software.

7. A programmable event timer.

8. A parity computing network to allow faster computation of microcode parity.

9. A Time-of-Century clock with battery backup.

10. 256 words of non-volatile read-write memory to support proper power-fail restarts of the system.

11. A diagnostic visibility/control mechanism to support the microdiagnostics.

12. System clock control - start, stop, half-step and burst.

13. An 16-character alphanumeric display on which the console can display error information and other functions.

14. A console panel containing:

    1. a power-on switch.
    2. a "System Start" switch.
    3. a "Console Start" switch.
    4. a Console Lock key switch. This switch must provide a hardware lockout of the two "Start" switches and must be readable by the console software.
    5. a four-position key switch: power-on disable, power-on enable, KLINIK user enabled, and KLINIK CTY enabled. The status of this switch must be readable by the console software.

16.2.3 Reliability Goals

1. To allow the operating system to continue running if the console crashes.

2. To provide enough hard-core testing in PROM to verify that the console MPU can talk to the CTY and the load device.

16.2.3.1 Failsoft Goals -

1. The 2080 console software shall attempt to recover from soft parity errors which occur in its instruction space.

2. The console shall also handle console subsystem programming errors and operator typein errors in a reasonable fashion. During timesharing, any commands that would cause timesharing to stop in an unorderly fashion will require confirmation. This will not stop the malicious user from crashing a system, but it will help prevent the clumsy user from accidentally crashing the system. There will be a command available to turn off confirmation.

3. The Console Lock switch will prevent a user from inadvertently crashing a running system. With the switch enabled, depressing a console switch or typing Control-\ during timesharing will have no affect.

4. System operation will continue if one of the two console terminals goes away.

16.2.4 Non-Goals

1. To have the loadable 2080 console software implement a VAX compatible console command language. For those commands which have the exact same function, parsing shall be consistent.

2. To be capable of accessing the operating system disk file structure, except for the code necessary to find a disk "bootstrap" area. The bootstrap area shall be a contiguous number of blocks on the disk, and pointed to by an entry in the disk's home block. This scheme is like that used on the KS10.

3. To support any devices on the console other than those listed in the "Environments" section.

4. To provide the "hard-core" console capability to boot the "loadable" console from a system disk or tape. This implies that if the console floppy is down, the only means to boot the "loadable" console is the via the KLINIK/APT line.

16.3 FUNCTIONAL DEFINITION

16.3.1 Operational Description

The console software covered by this specification consists of a "hard-core" console (KCROM) which is blasted in PROM, a "loadable" console (KCCCN) which is located in the first 28KB of RAM, and the Run-time Support Program (KCRSP) which occupies a portion of the rest of the RAM space. The functionality contained in each of these three programs is described briefly in the next three sections. More detailed descriptions of individual functions are provided in subsequent sections.

16.3.1.1 "hard-core" Console -

The 12KB of PROM space contains the "hard-core" console code (program name KCROM) which performs those software functions which must be permanently available. Those functions include:

1. Fielding and attempting recovery on console memory parity errors.

2. Performing a console hardware hardcore test.

3. Booting the "loadable" console from the console floppy or downline over the KLINIK/APT line.

4. Performing very basic terminal handling and command parsing. No TOPS20 style prompting or recognition will be provided.

5. Providing console ODT (Octal Debugging Technique) functionality plus commands to select the boot device and to boot the "loadable" console.

6. Driving the KLINIK line as a remote diagnosis port.

7. Initiating the Start or Restart procedures up to the point where the "loadable" console can take over.

When the "hard-core" console is in control, the prompt 'ROM>' will be typed on the console terminal(s).

16.3.1.2 "Loadable" Console -

The first 38KB of loadable RAM is referred to as the "loadable" console (program name KCCCN). Together with the "hard-core" console, it comprises the "resident" console. It is loaded by the "hard-core" console and provides the following functionality:

1. An extensive set of commands to access and control the 2080 System hardware.

2. A text editor.

3. Routines to communicate with the power controller and monitor the environmental conditions.

4. A more sophisticated terminal handler and command parser than the "hard-core" console. The parser provides a consistent subset of TOPS20-like prompting and recognition.

5. A general purpose subroutine package for use by console-based diagnostic programs, the RSP, and console utility programs. The routines and their interfaces are described in Appendix C.

6. Routines for bootstrapping the 2080 system. This includes loading microcode into the CPU and I/O Ports.

7. An ODT-like debugging package.

The memory segments containing the "loadable" console are mapped as read-only using the F11 memory management option. This protects against higher level console programs overwriting parts of the "loadable" console.

When the "loadable" console is in control, the prompt 'CCN>' will be typed on the console terminal(s).

16.3.1.2 Run-time Support Program (RSP) -

The Run-time Support Program is loaded somewhere in the remaining console RAM which is not occupied by the "resident" console. The only time that the RSP will not be in RAM is when console-based diagnostics or utilities (such as the Trace/Debugging Package) are being run. The functionality provided by the RSP includes:

1. Executing its part of the Start or Restart sequence.
2. Supporting USER MODE KLINIK between the KLINIK USART and the 2080 Program.
3. Supporting 2080 Program to/from Console floppy data transfers. This features will be used only for software development and release engineering.
4. Communicating with a 2080 program (Operating System or diagnostic monitor). This includes routing message packets between the 2080 program and the console devices.
5. Providing module callout for the CPU cluster when the CPU error detection hardware detects a fault. This assumes that adequate error scan-out logic is provide to do this isolation. Approximately 70-80% of the possible CPU errors should be isolatible to the module.
6. Providing error logging for soft CPU errors.
7. Providing routines to initiate instruction retry after a 2080 CPU hardware error.
8. Capability to run two tasks in a foreground/background mode. This will be used to do fault insertion to test CPU error detection and microcode retry algorithms.

When the RSP is in control, the prompt 'RSP>' will be typed on the console terminal(s).

16.3.1.4 Software Configurations -

There are several possible configurations of software running in the console and the 2080 CPU. Obviously, the "hard-core" console is always present. The "loadable" console is always present except when a console diagnostic is being run or a console hard-core hardware problem exists which prevents its being loaded. Normally, when a program is running in the 2080 CPU, the RSP program must be running in the console. Otherwise, the RSP may be replaced by console-based 2080 diagnostics or other console utility programs. The table below lists the possible configurations.

Console software	2080 CPU software
1. KCROM *	-
2. Console Diag * KCROM	-
3. KCCON * KCROM	-
4. Utility Program * KCCON KCROM	-
5. Console-based 2080 Diag * KCDSP KCCON KCROM	-
6. KCRSP * KCCON KCROM	-
7. KCRSP * KCCON KCROM	Operating System *
8. KCRSP * KCCON KCROM	Diagnostic Monitor (D80MON) *

* = program in control

In configurations 1-6 no program is running in the 2080 CPU.

1. In configuration 1, the user is at 'ROM>' prompt level talking to the basic "hard-core" console command parser.

2. In configuration 2, the user is running a diagnostic of the console itself. These diagnostics will be reworks of standard PDP11 CPU diagnostics. Prompting and dialogue is under control of the diagnostic.

3. In configuration 3, the user is at 'CON>' prompt level talking to the TOPS-20 like command parser.

4. In configuration 4, the user is running a console utility program (such as the Trace Package) as a third level of software in addition to KCROM and KCCON. The functionality of KCCON is available to the utility as subroutines. Prompting and dialogue are under control of the utility.

5. In configuration 5, the user is running a console-based 2080 diagnostic as a fourth level of software in addition to KCROM, KCCON and KCDSP. KCDSP is the Diagnostic Support Package which runs transparent to the user as a third level of software. It provides test dispatching and subroutines for the diagnostic. Prompting and dialogue are under control of the diagnostic and KCDSP.

6. In configuration 6, the user is running the Run-time Support Program as a third level of software in addition to KCROM and KCCON. The functionality of KCCON is available to RSP as subroutines. Prompting and dialogue are under control of the RSP.

7. In configuration 7, the user is running an operating system in the 2080. The RSP is actively communicating with the operating system. CTY input and output are under control of the operating system, with the RSP merely serving as a buffer.

8. In configuration 8, the user is running 2080-based diagnostics under control of the 2080 Diagnostic Monitor (D80MON). The RSP is actively communicating with D80MON. CTY input and output are under control of D80MON, with the RSP merely serving as a buffer.

Note that when a console-based 2080 diagnostic is running (configuration 5), no program can run in the 2080 since the diagnostic will be manipulating the 2080 hardware at the microcode level.

16.3.1.5 Console Terminal Handling -

16.3.1.5.1 Terminal Modes -

The two contexts in which the console terminals operate are "Console Mode" and "System Mode". In "Console Mode", the controlling console program handles all terminal input and output. The console program's types it prompt (e.g., 'CON>') and all characters typed at the terminal are interpreted as input for the console program. "System Mode" is the normal mode when a program is running in the 2080 CPU and the RSP is running in the console. The RSP program is transparent to the user and acts mainly as an intelligent terminal buffer. All prompting and character echoing must be provided by the 2080 program. Characters typed at the terminal are passed directly to the 2080 program. Up to 128 input characters will be buffered. If an overflow occurs in "Console Mode", the character is thrown away (not echoed) and a bell is given. In "System Mode", the character is written into the input buffer over the previous last character received. When this overflow character is finally passed to the 2080 program it is accompanied by an error status. The capability will be provided for the 2080 program to request the flushing of the console input buffer.

There will be commands provided to switch the console context. To change context from "System Mode" to "Console Mode", the user must type a CONTROL-BACKSLASH(^\). This is the one character which is not passed to the 2080 program in "System" mode. If a single Control Backslash is typed, the mode switch will occur after all the preceding characters in the input buffer have been sent to the 2080 program or after a timeout has occurred. Two Control Backslashes cause immediate transfer to "Console" mode. All characters in the input buffer are flushed. This is analogous to the operation of Control-C in TOPS-20. This mode switch will result in the 'RSP>' prompt being typed and the console entering into a foreground/background mode. At this point, the terminal input is processed by the RSP program. The RSP will continue to maintain keep-alive and other essential communication with the 2080 program. A command to change the escape character will be provided. To return to "System Mode", the user must type the command 'QUIT' to the RSP (see Appendix B). Any command to the RSP which executes a 2080 instruction or starts a 2080 program will automatically cause a switch to "System Mode".

NOTE:If the Console Lock switch is in the lock position, typing a Control-\ will have no affect.

16.3.1.5.2 Terminal Usage -

The 2080 will have both a hardcopy and a video terminal attached to the console at all times. The use of these terminals and how they are controlled by the console software varies depending on the situation.

The "hard-core" console will treat both terminals identically. All output will be sent to both terminals and input will be accepted from both terminals. Also, if the console panel key switch is in the KLINIK CTY position, the KLINIK line will be treated the same as the two terminals.

The "loadable" console and the console utilities treat the terminals differently.

- Input from the hardcopy terminal will be echoed at both terminals and all output resulting from the execution of a command at the hardcopy terminal will be sent to both terminals. If the KLINIK line is being accessed, it will also receive echo and command execution output.

- Input from the video terminal will echo on the video terminal and on the KLINIK line if it is being accessed. Command execution output also goes to the video terminal and the KLINIK line. Echo and command execution output will be sent to the hardcopy terminal only if it is enabled as a logging device. There will be a commands to enable and disable this feature.

- Input from the KLINIK line will echo on the KLINIK line and at both terminals. Command execution output will also be sent to all three lines. By command, the KLINIK user can inhibit all typein on the console terminals. The local operator can override this by ???.

The console-based 2080 diagnostics will treat the three lines the same as the "loadable" console does except that the hardcopy terminal will always be enabled as a logging device.

If no program is running in the 2080 CPU, the RSP treats the three lines the same as the "loadable" console does. If there is a 2080 program running, the RSP and the 2080 program work together to provide the following functionality. The video terminal is operated in a split screen mode. The first four lines are used for a hardware real-time display controlled by the RSP. The remaining twenty lines are used for the operator's terminal under the control of the 2080 program. The hardcopy terminal is used as a logging device for important system messages and information which may or may not be also displayed on the operator's terminal. The output to each is determined by the 2080 program. A Control-\ typed at the video terminal will put both the video and hardcopy terminals into "Console Mode" at

RSP prompt level. A Control-\ typed at the hardcopy terminal will put the hardcopy terminal into "Console Mode" but will not affect the video terminal. A Control-\ typed over the KLINIK line (assuming that KLINIK CTY is enabled) will put the KLINIK line and the hardcopy terminal into "Console Mode" but will not affect the video terminal.

16.3.1.6 Transitions Between Prompt Levels -

The table below describes the actions necessary to move between command levels:

From Level	To Level	Action Required
ROM>	CON>	type 'R KCCON.xxx'
ROM>	RSP>	not directly possible
CON>	ROM>	depress 'Console Start' switch on console panel
CON>	RSP>	type 'RUN RSP'
RSP>	ROM>	depress 'Console Start' switch on console panel
RSP>	CON>	type 'QUIT'
user prog prompt	CON>	type 'QUIT'

16.3.1.7 "Hard-core" Console Commands -

The "hard-core" console will implement commands to perform the following functions.

1. Examine and deposit console memory and general purpose registers.
2. Examine and deposit the console Processor Status Word (PSW).
3. Start a console program at a specified location.
4. Proceed from the location specified in the console PC (R7).
5. Change the default Boot device.
6. Load console RAM from the default Boot device or the KLINIK/APT line.
7. Set one non-deleting breakpoint.
8. Change USART line speeds for the hardcopy terminal and the KLINIK/APT line.

NOTE: Those commands which reference memory are restricted to Kernel Virtual addresses 0-32K.

16.3.1.8 "Loadable" Console Commands -

The "loadable" console will implement commands to perform the following functions:

1. Examine and deposit console memory and general purpose registers.
2. Load, verify, start and run in programs the console.
3. Dump the contents of console memory onto the console floppy. Dump file will be in binary format (.BIN).
4. Take a directory of the console floppy contents.
5. Delete or rename a file on the console floppy.
6. Edit a command file. For a description of the editor functionality, see the section 'Console Editor'.

7. Execute the contents of a command file from the console floppy as if the information were being typed at the CTY. This is similar to a Batch control file on TOPS-20.

8. Enable and disable echoing during execution of command files.

9. Perform a hardware reset on the entire 2080 system or on selective sections of it, the TTL Bus, MBCX, etc.

10. Examine and deposit I/O Port microcode.

11. Load and verify Port microcode from the console floppy or the KLINIK/APT line.

12. Examine and deposit 2080 MCS memory, using physical addressing.

13. Examine and deposit 2080 CPU accumulators (all AC blocks).

14. Read and Write 2080 CPU internal registers.

15. Examine and deposit 2080 CPU microcode control RAMs and lookup RAMs.

16. Load and verify 2080 CPU microcode from the console floppy or the KLINIK/APT line.

17. Examine and deposit Next and Previous locations, on commands where applicable (console memory, MCS memory, I/O Port microcode).

18. Repeat command.

19. Perform clock control commands - burst, cycle, clock start, clock stop, select clock rates and margin the clocks.

20. Single Step the 2080 CPU micromachine.

21. Breakpoint the micromachine (if the feature is implemented in the hardware).

22. Load and start the system or diagnostic "pre-Boot" programs from the console floppy or the KLINIK/APT line into 2080 memory.

23. Load the system. This includes all microcode.

24. Boot the system. This includes loading all microcode and performing a cold start, warm start, or soft start, as specified by the command.

25. Zero MOS memory, either by direct deposit into the memory, or by the quiet execution of the appropriate 2080 instruction.

26. Define a logical command. This will be used for the situation where a sequence of console commands is used frequently. This command will allow the user to specify a sequence of commands which can be executed by typing the logical command name as a command or as the argument top a "REPEAT" command. There will be either 4 or 8 logical commands which can be defined. The user will also be able to define a sequence to be executed upon hitting specified terminal keys.

27. Set parameters to be used by the console for input/output to the four asynchronous lines or as defaults in certain situations. Some of these parameters will be maintained in non-volatile RAM and therefore will remain in effect until changed by another command. (See the section 'Non-volatile RAM Contents' for a complete list.) Some currently envisioned parameters are:

    1. Terminal characteristics:

        (a) Page length
        (b) Line width
        (c) Terminal type
        (d) Line speed
        (e) Tabs
        (f) XON/XOFF enabled

    2. Confirmation (if you would like "fatal to 2080 program" commands confirmed)

    3. Auto boot device (which device should be used in start and restart situations)

    4. Suppress leading zeroes (Software people prefer leading zeroes be suppressed. Hardware and service people prefer full printout. This option will be in effect only for responses to console commands. Console-based diagnostics will ignore this parameter for error printouts).

    5. Auto Reload. When this parameter is set, the console will attempt to automatically boot and start the 2080 after Keep-Alive failure.

    6. KLINIK password

    7. Default load device. This will be used to specify that either the console floppy or the KLINIK/APT line be used by the console during normal operation. For

example, it specifies the load device to be used when a program running in the 2080 requests a read or write of the default load device. Note that this may be different from the Auto-boot load device and is not maintained in non-volatile memory.

28. Set voltage margins.

29. Get information about various parameters and status.

30. Enter the console ODT package. (see section 'Console ODT' for details.)

31. Terminate a KLINIK access.

32. Use the hardcopy terminal as a line printer.

16.3.1.9 RSP Commands -

The RSP will implement commands to perform the following functions:

1. Enter "System Mode". This will be used mostly in conjunction with the Control-Backslash to facilitate transferring between "System Mode" and "Console Mode".

2. Change the escape character used to transfer from "System Mode" to "Console Mode". The default is Control Backslash.

3. Select one or both of the console terminals to be the CTY for operating system communications.

4. Shutdown the 2080 Program. This command will do the equivalent of what is done for the KS10 Shutdown command, namely deposit a -1 into memory location 30.

5. Continue the 2080 program.

6. Halt the 2080 CPU.

7. Start the 2080 at a specified address. This will always be an LXEC mode start with no RESET done.

8. Resync the RSP with the 2080 program. This will re-establish communication and Keep-alive.

9. Execute a specified 2080 instruction in the 2080 CPU.

10. Single instruct the 2080 CPU.

11. All commands supported at the "loadable" console command level will also be available at RSP command level. Commands which cause the RSP to be overwritten, such as RUN and LOAD CONSOLE, will require confirmation.

16.3.1.10 Console Memory Parity Errors -

The console will attempt to recover from parity errors in its RAM space. The console RAM code will be divided into 256 word segments. For each segment, the "hard-core" console will keep the total XOR of all locations within that segment. It will also keep a map of which segments contain instructions and which contain data buffers. If there is a RAM parity error, the address of the faulty word is latched in a special hardware register and a hardware trap to a "hard-core" console recovery routine occurs. The routine turns off all interrupts.

If the error occurred within a data segment, recovery is not possible. The "hard-core" console types an error message on the CTY and in the panel display and logs the error information in the non-volatile RAM. It then does a console init and restarts the "loadable" console.

If the error occurred in an instruction segment, the recovery routine computes the total XOR of all locations within that segment and compares the result to the "kept" XOR value. This comparison will identify the bad bit. The contents of the latched address is read, the bad bit is complemented, and the result is written back to the address. The contents are read back to insure that the bit got corrected. If the error is non-recoverable and the Console Lock switch is not enabled, the console goes to prompt level ('ROM>') and waits for input. If the Console Lock switch is enabled, terminal input is ignored. If the operating system was running at the time of error, it will time-out the console keep-alive and poke its electronic finger at the console. This will cause the console to reboot and resync with the operating system. The logged error information will then be passed to the operating system for SPEAR reporting.

During this recovery period all communication, including keep-alive, with the 2080 program is suspended. Thus the keep-alive period must be defined to be at least twice the expected recovery time.

16.3.1.11 Console Hardcore Test -

When the console is powered up or the 'System Start' switch is depressed, a console hardcore test is performed. Testing includes a basic PDP-11 instruction test, a checksum of the ROM code, a complete console RAM test, a UART loop-back test and a DMA controller loop-back test. As it progresses through this test sequence, it prints an incremental message in the console alphanumeric display, on the console hardcopy and video terminals, and over the KLINIK/APT line. Should the console fail anywhere in its self-test, the displayed code will indicate exactly how far along in the sequence it got. This correspondence will be documented and available to the user. This technique will aid the user in distinguishing between hardcore console failures and failures in the console terminal in the case where nothing gets typed on the terminal after power-up. If possible, the program will enter an error loop. To escape the loop and enter "hard-core" console CDT level, the user must type a Control-C.

For complete details see the "2080 System Console PROM Program" functional specification by Tony Berardi.

16.3.1.12 Terminal Handler And Command Parser -

The console terminal handler and command parser will be implemented in the "loadable" console. It will contain the following features:

1. The fields of a command can be given in abbreviated, recognition, or full input mode. Recognition is the completion of a partially typed field when the user types an ESCAPE.

2. Input can be edited using a DELETE, Control-U, Control-W, and Control-R.

   - Typing a DELETE will delete the last character in the current command line. This may be either the last character typed by the user or the last character printed by the parser in response to an ESCAPE. On a hardcopy terminal the deleted character(s) will be printed enclosed by BACKSLASHs. On a video terminal the cursor is backed-up to the deleted character's position and the character is blanked. For a TAB, the cursor is backed-up to the first position following the previous character and the rest of the line is blanked.

   - Typing a Control-U will delete the entire command line back to the prompt. On a hardcopy terminal a space followed by XXX is printed on the current command line. Then a CR-LF and a new prompt is printed. On a video terminal the cursor is moved back to the first position after the prompt and the current command line is blanked.

- Typing a Control-W deletes the last field in the current command line. On a hardcopy terminal an UNDERSCORE is printed. On a video terminal the cursor is moved back to the first position of the deleted filed and the field is blanked.

- Typing a Control-R retypes the current command line. On a hardcopy terminal the command line is retyped on the next line. On a video terminal the command line is retyped on the same line.

3. Certain fields of certain commands can be defaulted if an ESCAPE is typed at the beginning of the field or if the field is omitted entirely. (see details of the EXAMINE command in Appendix B.)

4. A help message is printed if a question mark (?) is typed at the beginning of any field.

5. The correct portion of the last command line (i.e., up to the point where an error was detected) will be retyped if the next command line begins with a Control-H. The user can continue typing from that point.

6. Comments may be appear on a command line. An exclamation mark (!) or a semi-colon (;) signals the beginning of a comment field. All input from that point until the end of the line is considered a comment and is not parsed.

7. A minus sign (-) can be typed preceding an number field to input a negative number. The two's complement of the number will be generated as program input.

8. Spaces and Tabs are field delimiters and as such are thrown away during parsing.

9. Parsing is not done until an break character is typed, either a CR, an ESCAPE, or a ??.

10. No execution is performed on the command until the CR is typed.

11. Where applicable, numeric arguments can be typed as a range of values, with the values separated by a colon (:). For example, EXAMINE 1000:10007.

12. Multiple arguments of the same class can be typed on the same command line separated by commas. For example, DELETE foo,dum,diddle.

13. A parse EMT is provided for user programs wishing to receive and parse input. Unlike the COMND JSYS, a single EMT call parses the entire command line. Since the parser is table driven, this requires the user to provide complete parsing tables for all implemented commands.

Programs running as a third (or higher) level of console software can implement their own set of commands plus retain those of lower levels. This is done by passing parameters to the parser telling it to use multiple levels of parsing tables.

16.3.1.13 Console ODT -

To facilitate the debugging of console-based programs, an Octal Debugging Technique (ODT) package will be provided. It will reside in the "loadable" console and as such will not have to be loaded with or linked to user programs. The functionality is based on the RT-11 ODT package. Some of the features are:

1. Print the contents of any location for examination or alteration. This includes the general purpose registers and the right half of the processor status word.

2. Data printouts are user controlled. Addresses can be either absolute or relative to a relocation register. The contents of words are printable in base 2, 8, or 10. The contents of a location opened in byte mode are printable as ascii or octal.

3. The movement between locations is accomplished with the following commands:

   (a) <cr> - close a location with or without modification.
   (b) <lf> - same as <cr>, and then open next location.
   (c) <^> - same as <cr>, and then open previous location.
   (d) <@> or tab - close the current location, using its contents as the address of the next location to open.
   (e) </> - open the specified location as a word.
   (f) <\> - open the specified location as a byte.
   (g) <_> - open the specified location as a 36 bit word.

4. Resume execution of the program at any point, after exiting a breakpoint trap. This includes single step mode of program execution and the ability to bypass a breakpoint an arbitrary number of times before control is returned to the user.

5. Automatic calculation of offsets (for Branch-type instructions) will be provided.

6. Establish the priority level at which debug is to operate while in command mode. if set at '4', for instance, the real-time clock could continue to operate without losing time.

7. Typing a Control-C will exit the CDT package and return the user to the previous command level. ODT will be re-enterable to allow debugging to continue.

8. The CDT parsing will understand +, -, and . (current location) operators.

9. Memory management support - to be supplied.

10. Eight non-deleting breakpoints.

16.3.1.14 Console Editor -

A text editing program will be invoked from the console floppy when the user types EDIT. The editor to be used is EDIT/SOS.

16.3.1.15 Power Control And Environmental Monitoring -

In the 2080 system, the console has the capability to monitor environmental conditions and control system power. The console can read the status of:

1. each of the system's thermal sensors - normal or over temperature

2. the voltage level from each of the power supply's modules - normal, over tolerance or under tolerance

3. battery backup - valid or not

4. the AC and DC low signals - true or false

5. the ground fault detector - current flowing or not

There are five environmental conditions which require immediate console action:

1. Over temperature. One or more thermal sensors detects a temperature rise above acceptable limits.

2. Voltage over tolerance. Circuitry in the power supply detects that the voltage from one or more power supply modules is above the acceptable threshold.

3. Voltage under tolerance. Circuitry in the power supply detects that the voltage from one or more power supply modules is below the acceptable threshold.

4. AC low. Circuitry in the power supply detects that the AC voltage into the system is below the acceptable threshold.

5. Ground fault. Circuitry in the power supply detects that current is flowing in system ground.

All of the above conditions will produce a Level 5 interrupt to the console. The "loadable" console will field the interrupt and determine which condition caused it.

If the interrupt was caused by AC low, the console will store the Time-of-Century and environmental status. An interrupt to the 2080 CPU will also occur so that the operating system (if running) can shutdown before power is lost.

If the interrupt was caused by an over temperature condition or a ground fault, the console will print a warning message on the CTY and in the console display. If a 2080 program is running, the RSP will generate an interrupt to the 2080 CPU and pass the warning information to the 2080 program.

If the interrupt was caused by a voltage over or under tolerance, the console will print a warning message on the CTY and in the console display and will turn off ECL power to the system. If a 2080 program is running, the RSP will generate an interrupt to the 2080 CPU and pass an ABORT command to the 2080 program.

There will be "loadable" console commands to turn ECL power on and off. However, these commands will not be executable from the KLINIK line.

16.3.1.16 Non-volatile RAM Contents -

The 256 words of non-volatile RAM will be used to store information which must be maintained over powered down periods. This information currently includes:

1. The time-of-century of last power down.
2. The terminal characteristics for the console terminal lines.
3. The baud rates for 7 of the 8 console USART lines, excluding the video terminal line.
4. The KLINIK password.
5. The environmental conditions before power down (AC low) or whenever an abnormal environmental condition is detected.
6. System configuration information. This includes:
    1. The names and versions of CPU microcode to be loaded on autoload.
    2. Which type of microcode to load into each I/O Port.
    3. Which 2080 pre-boot program is to be loaded during auto reload - system or diagnostic preboot or none.
    4. The program name to be passed to the Boot program. This will be a limited character, complete filespec which is settable by command or by the Boot program itself.
    5. 2080 memory configuration, including Cache.
7. Default parameters, such as:
    1. the autoboot load device.
    2. the Autoreload flag.
8. Error logging information.

16.3.1.17 Console Floppy Control -

In order to increase the perceived reliability of the console floppy disk, it will not be made available to timesharing users. Instead, it will be powered down by the console software whenever the floppy is idle for 5 minutes. This should increase the mean time between failure of the floppy drive.

16.3.1.18 Start/Restart Procedures -

There are four ways that the console can be started:

1. Power-up. This can be a cold start initiated by depressing the console panel power on switch or a power-fail restart.

2. Depressing the console panel "Console Start" switch.

3. Depressing the console panel "System Start" switch.

4. The 2080 CPU asserting its electronic finger to the console.

These stimuli, together with the parameters stored in the non-volatile memory, determine the exact start or restart sequence to be performed.

16.3.1.18.1 Cold Start Or Power-fail Restart -

1. The "hard-core" console is powered up.

2. The "hard-core" console performs its hardcore test. If an error is detected, the power-on and bootstrapping procedures will be aborted. To successfully power-up the user must repair or replace the console module.

3. The "hard-core" console prints its name and version.

4. The "hard-core" console checks non-volatile memory for the default boot load device. This may be either the console floppy or the KLINIK/APT line.

5. If the "hard-core" console cannot access the specified load device, the automatic boot process is aborted. The console prints an error message on hardcopy. It then prints its prompt and waits for a command.

6. If the "hard-core" console is successful in accessing the boot load device, it begins bootstrapping itself by transferring the "loadable" console from the device into its RAM.

7. The "hard-core" console transfers control to the "loadable" console, passing it parameters relating to the boot process. The "loadable" console prints its name and version.

8. The "loadable" console then checks environmental conditions. If not OK, a message will be printed on hardcopy. The power-on and bootstrapping procedures will be aborted and the prompt 'CON>' will be typed.

9. If environmental conditions are OK, a message to that effect will be displayed in the console display. The "loadable" console turns power on to the rest of the 2080 system via the power controller then executes a 2080 CPU init.

10. The "loadable" console reads the status of battery backup. If backup is valid, then the console will print a message telling the user that it is beginning a power-fail restart. Otherwise, it prints a message telling the user that it is starting the Automatic Boot process. In either case a Control-C will abort the process. If aborted, the "loadable" console prints its prompt and waits for a command.

11. If the console is performing a cold start, then it checks a parameter in non-volatile memory which specifies whether the 2080 operating system, the 2080 Diagnostic Monitor or neither of the two is supposed to be booted. If neither, then the "loadable" console prints its prompt and waits for a command.

12. The "loadable" console next reads the I/O Port microcodes from the console boot load device and loads the Ports via the diagnostic scan path. Which type of microcode to load into each Port is determined by the configuration information in the non-volatile RAM. If an error is detected it will be reported on both console terminals and the "loadable" console will go to prompt level.

13. If the console is to boot the operating system or the 2080 Diagnostic Monitor or if it is performing a power-fail restart, it loads the RSP program from the boot load device. It then loads in all CPU microcodes and lookup RAM data from the boot load device. Before the microcode and data are loaded, an address test of the RAMs being loaded is performed. The loading process includes typing the version number of each microcode as it is loaded and also verifying each location as it is loaded. If an error is detected it will be reported on both console terminals and the "loadable" console will go to prompt level.

14. If the console is to boot the operating system or the 2080 Diagnostic Monitor, it zeros all of 2080 memory and performs a double-bit error scan. The resulting map is later passed to the pre-Boot program for use by the Boot program.

15. If a power-fail restart is being done, the "loadable" console turns control over to the RSP which prints it name and version and restarts the 2080 program at physical location 70. The RSP then switches to "System Mode". It enters a timing loop waiting for the operating system keep-alive to become active. If keep-alive does not become active in 15 seconds, an error message is printed on the console terminals. If the Auto-reload parameter is set, a cold start is initiated.

16. If a cold start is being done, the "loadable" console loads a console-based diagnostic which tests the 2080 CPU error detection logic. If an fault is detected, an error message is printed and the boot process is aborted. The "loadable" console types it prompt, 'CON>'.

17. If the diagnostic passes, the "loadable" console reads in the appropriate pre-boot program, either for the operating system or the 2080 Diagnostic Monitor, from the boot load device.

18. The "loadable" console passes control to the Run-time Support Program which prints its name and version. The RSP starts the pre-boot program and switches to "System Mode". It enters a timing loop waiting for the operating system keep-alive to become active. If keep-alive does not become active in 15 seconds, an error message is printed on the console terminals. The RSP types its prompt and enters "Console Mode".

16.3.1.18.2 Console Start Switch -

When the "Console Start" switch is depressed, the console goes directly to the PROM command level, ROM>. No testing or bootstrapping is done.

16.3.1.18.3 System Start Switch -

When the "System Start" switch is depressed, the console vectors to the PROM start code and performs the equivalent of a cold start of the operating system.

16.3.1.18.4 Electronic Finger -

When the 2080 CPU asserts its electronic finger, the console vectors to the PROM start code. It proceeds to run the hardcore test, boot the "loadable" console and the RSP, and attempts to resync itself with the 2080 program.

16.3.1.18.5 2080 Program Keep-alive Failure -

If a program running in the 2080 fails to maintain its keep-alive counter, the console will report the failure on the CTY. A non-interfering hardware snapshot is taken and stored. It will then check the Auto-reload parameter in non-volatile memory. If not set, the console will simply wait at RSP prompt level. If Auto-reload is set, the console will initiate a system restart procedure. It executes location 71 in EXEC mode and waits for an acknowledgement that the 2080 is alive. If no acknowledgement comes, first a system reset is , then the appropriate pre-boot program is loaded and started in 2080 memory.

16.3.1.18.6 Non-Automatic Start -

On applying power to the console, automatic bootstrapping will not be performed if a control-C is typed or if the Auto-reload parameter is not set in the non-volatile memory. It these instances, the power-up process stops at "hard-core" prompt level ('ROM>'). A system boot can then be performed by first typing a 'Load' command, which will cause only the "loadable" console to be loaded. When the load is complete, the operator will be talking to the "resident" console command parser. He may then type a 'Boot System' command to boot the system.

16.3.1.19 KLINIK -

16.3.1.19.1 KLINIK Access -

Access to the 2080 system over the KLINIK line is controlled by the key switch on the console panel and the KLINIK password. With the key switch in the KLINIK User or KLINIK CTY positions, a remote user is allowed access to the system with the capabilities described below. With the key switch in any other position, no remote access is allowed. Once a user is allowed access, the state of the key switch is polled. If a KLINIK CTY access is in progress and the switch is moved to the KLINIK User position, no status change is made. However, if the switch is moved to the POWER ENABLE or DISABLE positions, the KLINIK connection is broken (line hung up). The same will happen to a KLINIK User access. There will also be a command to the "loadable" console to terminate a KLINIK access.

The CONSOLE LOCK switch will not affect the functionality of Control-Backslash when typed on the KLINIK line.

16.3.1.19.2 KLINIK Password -

When Data Set Ready (DSR) and carrier are detected on the KLINIK line and remote access is allowed, the user will be asked for the KLINIK password with the prompt 'password:'. If he types the incorrect password, the message '?Invalid' is typed and a reprompt is given. The user is given three tries to provide the correct password. After the third incorrect try, the line is hung up.

The password is set using the 'Set KLINIK Password' console command and is stored in the non-volatile memory. It may be up to six alphanumeric characters.

16.3.1.19.3 KLINIK Modes -

A person using the KLINIK line can be in any one of three modes, depending on the panel switch settings and previous actions. Appendix D contains diagrams illustrating the three modes described below.

KLINIK Console Mode - All KLINIK input is OR'ed with CTY input and processed by the console software as CTY input. Output to the CTY is also sent over the KLINIK line.

KLINIK System Mode - All KLINIK input is OR'ed with the CTY input and sent to the 2080 program via the Terminal Input Word for the CTY. Output from the 2080 program via the CTY Terminal Output Word is sent to both the CTY and the KLINIK line.

KLINIK User Mode - All KLINIK input is sent to the 2080 program via the KLINIK Input Word in the reserved communication page in 2080 memory. All 2080 program output to the KLINIK line is sent via the KLINIK Output Word in the communication page. The 2080 program can request hang-up of the KLINIK line at any time by sending the appropriate code in the KLINIK Output Word. If KLINIK connection is broken (see below Modem Control for details), the 2080 program is notified via the KLINIK Input Word.

16.3.1.19.4 KLINIK Situations -

The following list described the possible situations in which the user could attempt to gain remote access and the resulting actions by the console software.

Key switch in a position other than KLINIK User or KLINIK CTY.

Situation - State of 2080 software is a "don't care". User is not allowed access.
Action - No password prompting is done. Instead, the message "KLINIK access attempted - switch disabled" is returned over the KLINIK line and also typed on the CTY. This informs the user that his KLINIK connection is okay but the key switch is not positioned correctly. It also signals the local operator that a remote access was attempted. The operator can then turn the key switch to one of the enable positions if KLINIK access was scheduled to be allowed. If the switch is enabled, then the user will be prompted for password and one of the scenarios described below will occur.

Key switch is in the KLINIK User position:

Situation - No program is running in 2080 CPU and the console is in "Console" mode.
Action - After password verification, the message "DECSystem not running" is returned. Any input received from the user is thrown away until someone at the CTY starts a program running in the 2080. When the 2080 program-RSP communication is established, the 2080 program is notified that the KLINIK line is active. The user is then in KLINIK User Mode. A message that a KLINIK User access was made is typed on the hardcopy terminal.

Situation - A program is running in 2080 CPU and the console is in "Console" or "System" mode.

Action - After password verification, the 2080 program is notified that the KLINIK line is active and the user enters KLINIK User Mode. A message that a KLINIK User access was made is typed on the hardcopy terminal.

Key switch is in the KLINIK CTY position.

Situation - No program is running in the 2080 CPU and the console is in "Console" mode.
Action - After password verification, the user enters KLINIK Console Mode. A program Control-C is executed to abort any "Console" mode activity which may be in progress. The following message is typed on the CTY and sent to the KLINIK line:

```
[KLINIK activated - Console Mode]
[To continue aborted activity, type CONTINUE]

'prompt'>
```

the 'prompt' will be based on which console level the CTY was in when the KLINIK user dialed in.

Situation - A program is running in the 2080 CPU and the console is in "Console" mode.
Action - After password verification, the user is placed into KLINIK User Mode. By typing a Control-\, the user can enter KLINIK Console Mode. At that time the same action is taken as described in the previous situation.

Situation - A program is running in the 2080 CPU and the console is in "System" mode.
Action - After password verification, the user is placed into KLINIK User Mode. By typing a Control-\, the user can enter KLINIK System Mode. At that time the following message is typed on the CTY and sent to the KLINIK line:

```
[KLINIK activated - System Mode]
```

By typing a Control-\ while in KLINIK System Mode, the user can enter KLINIK Console Mode. When in KLINIK Console Mode, if the user types a command, such as 'Continue', he will reenter KLINIK System Mode.

16.3.1.19.5 Modem Control -

The console control of the KLINIK Modem will conform to the pending standard written by A. Kent (see Bibliography). The important aspects of that control are:

1. The KLINIK UART will be programmed to produce even parity on KLINIK data.

2. The presence of Data Set Ready (DSR) will indicate the existence of a connection.

3. A Carrier loss for more than ?? seconds will cause a hangup.

4. The absence of DSR for more than ?? milliseconds will cause a hangup.

16.3.1.20 2080 Errors -

Certain hardware error conditions require the console to intervene and provide fault isolation, fault data collection and retry. The classes of errors and the actions to be taken by the console will be provided at a later date.

16.4 COMPATIBILITY

1. Modem control will conform to the standards described in the "Operational Requirements for Asynchronous, Full Duplex, Serial Terminals and System Interfaces Operating as DTEs Connected to EIA RS-232-c or CCITT V.28 Point-to-Point DCEs" by A. Kent.

2. The error messages typed by the console software will conform to the DEC standard - % preceding a warning, ? preceding a fatal message, and square brackets [] enclosing information.

16.5 EXTERNAL INTERACTIONS AND IMPACT

16.5.1 2080 System User To Console Interface

The user of the 2080 system will communicate to the console (and system) via a set of Console commands. The actual command list is located in Appendix F of this document. Radix of arguments for console commands requiring arguments will be either octal or decimal depending on context, as detailed in the Appendix E command list.

Arguments to commands dealing with 36-bit data paths will be in 12 Octal digits (or less), or in half words, each half word having 6 Octal digits and the half words separated by a pair of commas.

On type-out leading zeroes will not be suppressed (default case). There will be a command to set the suppression of leading zeroes for users who desire the feature, and for minimizing the character traffic and character translation of zeroes for the APT and perhaps KLINIK environments.

Arguments to commands involving the depositing of microcode will be in strings of octal digits. The number of octal digits required will depend on the number of microcode bits. Leading zeroes may be left off for these commands. Field break-down in the typeout of microcodes will be left to the remote host program and console utilities.

On type-in to the above commands, a "-"(minus) sign may be used and the two's complement of the number will be generated as the argument to the command.

16.5.2 2080 Program To RSP Interface

When a program is running in the 2080 CPU, the console acts as an intelligent subsystem controller with the following devices and pseudo-devices attached:

1. The KLINIK/APT line.
2. A floppy disk drive.
3. An environmental monitor.
4. A pseudo-TTY.
5. A pseudo-Error device.
6. Two asynchronous terminal lines.
7. One asynchronous multi-point line connected to CI nodes.

The RSP buffers and controls communication between these devices and the 2080 program. The communication scheme is similar to that used between the I/O Ports and the 2080 CPU. It involves a reserved communication area in 2080 memory and doorbell interrupts. When the 2080 program wishes to send data to a console device, it puts that data into the reserved area and interrupts the console. The RSP processes the interrupt and the associated data. When data from a device becomes available, the RSP puts it into the reserved area and interrupts the 2080 program. The format of the reserved area is defined in Appendix A.

16.5.2.1 General Characteristics -

Some of the features of the RSP-2080 program interface are:

1. The RSP shall exchange KEEP ALIVE counts with the 2080 Program. If the 2080 Program should not update its KEEP ALIVE counter for a period of ??? seconds, the operating system will be halted, and the instruction in physical location ??? will be executed. The 2080 Program will then be continued.

2. The RSP itself shall be reloadable by the 2080 Program. In the case where the console has not updated its keep alive count for a period of ??? seconds, the 2080 program will first request a reload using the 2080 program-RSP protocol. If the RSP does not resume its keep-alive after a specified time (to be determined later), the 2080 program will then initiate a console reload directly by asserting the console reload signal

via the I/O box. This will vector directly to the "hard-core" console which will initiate console boot procedure.

3. The RSP shall issue a warning and require confirmation of any console command that would abort or crash a running 2080 Program.

4. The Console will buffer input from any of the console input devices up to 128 characters. In "System Mode", when all CTY input is meant for the 2080 program, the Console will not echo characters as they are input. It will be the responsibility of the running 2080 program to echo typein. The 2080 program will have the ability to request the flushing of the console input or output buffers. Also, the RSP will handle XON/XOFF from the console devices.

5. The 2080 Program shall be able to execute just about any console command by sending ASCII strings representing the commands to the RSP pseudo-TTY device.

6. The 2080 program can request that the RSP sleep. This means that the RSP will make no references to 2080 memory until the 2080 program interrupts it. This will cause suspension of keep-alive. This functionality is needed for the instances when the 2080 program wishes to change the reserved communication area. It must, of course, notify the RSP of the new location before putting it to sleep.

16.5.2.2 2080 Program-RSP Communication -

Note: Refer to Appendix A for understanding of terms used in the following sections.

The general procedure for 2080 program initiated communication is:

1. The 2080 program checks the 2080 program Control Word. If clear, the RSP is ready to accept another interrupt. If not, wait.

2. The 2080 program writes the information to be sent in the appropriate Output Word.

3. The 2080 program writes the corresponding Interrupt Code in the Control Word and interrupts the RSP.

4. The RSP reads the Interrupt Code from the 2080 program Control Word and clears it. This signals the 2080 program that it is ready to accept another interrupt.

5. Based on the Interrupt Code, the RSP reads the appropriate Output Word and performs the specified function. If no 'Done' interrupt is required, the communication is finished. Else, continue.

6. The RSP reads the RSP Status Word. If not clear, wait for it to clear. If clear, the RSP writes the Device Number, sets the Done bit, and interrupts the 2080 program.

7. The 2080 program reads the RSP Status Word and clears it.

8. The 2080 program processes the 'Done' interrupt.

The general procedure for RSP initiated communication is:

1. The RSP reads the RSP Status Word. If clear, continue. If not clear, wait for it to clear.

2. The RSP writes the appropriate Input Word with the information to be sent.

3. The RSP writes the corresponding Device Number or Interrupt Code in the RSP Status Word and interrupts the 2080 program.

4. The 2080 program reads the RSP Status Word.

5. Based on the Device Number or Interrupt Code, the 2080 program reads the appropriate Input Word and clears it.

6. The 2080 program clears the RSP Status Word to signal the RSP that it is ready to accept another interrupt.

16.5.2.2.1 Terminal Input/Output -

1. 2080 program interrupts the RSP to output either one ASCII character or a string of ASCII characters packed 5 to a word left-justified. A 'Done' interrupt is returned by the RSP when the character(s) have been copied into the console's I/O buffers.

2. When the RSP receives a character from the terminal UART, it checks that the corresponding Terminal Input Word is cleared. If it is, then the RSP stores the word and interrupts the 2080 program to input the ASCII character. There is no 'Done' interrupt returned by the 2080 program. Instead, the 2080 program clears the Terminal Input Word, indicating that it has processed the character and is ready to receive another. If the Terminal Input Word is not cleared, the RSP will buffer up to 128 input characters.

3. The RSP will report UART parity, framing or overrun errors through the corresponding Terminal Input Word.

4. When the RSP Input Buffer overflows, the overflow character will be written into the last position in the buffer, thus overwriting the previous last character. This overwriting will continue until the 2080 program resumes accepting characters. The 2080 program will be notified of the overflow when the character from the overwritten buffer position is sent to it.

5. The 2080 program can request that a 'break' be executed by the RSP on the specified terminal line.

16.5.2.2.2 KLINIK/APT Line Input/Output -

1. 2080 program interrupts the RSP to output either one ASCII character or a string of ASCII characters packed 5 to a word left-justified. A 'Done' interrupt is returned by the RSP when the character(s) have been copied into the console's I/O buffers.

2. When the RSP receives a character from the KLINIK/APT UART, it checks that the KLINIK/APT Input Word is cleared. If it is, then the RSP interrupts the 2080 program to input the ASCII character. There is no 'Done' interrupt returned by the 2080 program. Instead, the 2080 program clears the KLINIK/APT Input Word, indicating that it has processed the character and is ready to receive another.

3. The RSP will report UART parity, framing or overrun errors through the corresponding KLINIK/APT Input Word.

4. The RSP will notify the 2080 program of carrier detect or carrier lost conditions on the KLINIK/APT line.

5. When the RSP Input Buffer overflows, the overflow character will be written into the last position in the buffer, thus overwriting the previous last character. This overwriting will continue until the 2080 program resumes accepting characters. The 2080 program will be notified of the overflow when the character from the overwritten buffer position is sent to it.

6. The 2080 program can request that a 'break' be executed by the RSP on the KLINIK/APT line.

7. The 2080 program can request a 'hang-up' of the KLINIK/APT line.

8. The RSP will notify the 2080 program when the KLINIK line is activated or deactivated.

9. The console code will handle proper connect/disconnect procedures according to the available U.S./European specifications.

16.5.2.2.3 Pseudo-TTY Input/Output -

1. 2080 program interrupts the RSP to output either one ASCII character or a string of ASCII characters packed 5 to a word left-justified. A 'Done' interrupt is returned by the RSP when the character(s) have been copied into the console's I/O buffers.

2. The RSP responds to the Pseudo-TTY commands one character at a time. It first checks that the Pseudo-TTY Input Word is cleared. If it is, then the RSP interrupts the 2080 program to input the ASCII character. There is no 'Done' interrupt returned by the 2080 program. Instead, the 2080 program clears the Pseudo-TTY Input Word, indicating that it has processed the character and is ready to receive another.

3. The RSP sends error-logging information to the 2080 program either one character at a time or in a string of ASCII characters packed 5 to a word left-justified.

16.5.2.2.4 Console Load Device Input/Output -

NOTE: Console load device files will be in DOS format.

1. The 2080 program can request the opening of a specified 'filnam.ext' for either reading or writing.

2. The 2080 program interrupts the RSP with data to be written onto the Console load device. The request can be either (1) to write a previously opened file or (2) to write a specified number of words starting at a specified block. The data can be a single ASCII character, a string of ASCII characters packed 5 to a word left-justified, or 'Packed 11 mode' (data is packed with 16 bits right adjusted in the right and left halfwords). The 2080 program also specifies the type of Done interrupt it wants returned by the RSP. This is done using the "Last Transfer Bit". To increase speed, with the "Last Transfer Bit" zero, the RSP will interrupt as soon as it has copied the data to be written into its output buffer. With the "Last Transfer Bit" one, the RSP will wait until the data has been written (successfully or not) before returning a Done interrupt. This mode would be necessary when writing the last block of a file.

3. The 2080 program interrupts the RSP with a request to read data from the Floppy disk. The request can be either (1) to read a previously opened file or (2) to read a specified number of words starting at a specified block. In both cases the 2080 program must specify the starting address in 2080 memory where the data is to be stored by the RSP. The RSP interrupts the 2080 program with the requested data in one of two formats - a single ASCII character at a time or 'Packed 11 mode' (data is packed with 16 bits right adjusted in the right and left halfwords) - depending on the what the 2080 program requested. The 2080 program can also specify that the read be done from the default console load device, which might not be the floppy.

16.5.2.2.5 Status -

1. The 2080 program notifies the RSP when its 'keep-alive' becomes active or inactive.

2. The 2080 program can request that the RSP reload the operating system.

3. The 2080 program can request that the RSP reload itself.

4. The 2080 program can request that the RSP sleep. This means that the RSP will make no references to 2080 memory until the 2080 program interrupts it. This will cause suspension of keep-alive.

5. The 2080 program can request that the RSP set the Time of Day. The RSP would then write the T.O.D. into the assigned communication page location and interrupt the 2080 program.

6. The RSP notifies the 2080 program of any environmental problems. There are three levels of notification - Caution, Please Stop, and Immediate Abort.

7. The RSP notifies the 2080 program when its 'keep-alive' becomes active or inactive.

In addition to the interrupt notification of status changes, both the 2080 program and RSP will maintain current status information in the 2080 program Status word and Console Status word, respectively. This will permit either program to poll the status of the other.

16.5.3 Console Subroutine Package Interface

All "calls" to basic Console subroutines will use the macros and macro package provided. There will be a Pseudo-terminal implemented internally in the basic Console software which will enable any console-based program to "execute" any console command and receive the results of that command. Arguments to calls requiring arguments will be passed on a data stack pointed to by R5. Calls returning information will return the information on the data stack.

All calls to EMTs and TRAPs will be via a MACRO.

16.5.4 APT Host System To 2080 Console Interface

An enhanced version of the APT protocol used in the manufacture of KS10s will be implemented for the 2080.

16.6 RELIABILITY/AVAILABILITY/SERVICEABILITY (RAS)

16.7 PACKAGING AND SYSTEM GENERATION

16.8 DOCUMENTATION

16.9 REFERENCES

"Dolphin Console Subsystem Specification", Bob Petty, March, 1979

"2080 Console Subsystem Hardware Specification" Rev 3, R.M. Smith, 30-Nov-79

"2080 Engineering Functional Specification" Second Edition, Don Lewine, 30-Nov-79

"2080 System Console PROM Program", Tony Berardi

"Operational Requirements for Asynchronous, Full Duplex, Serial Terminals and System Interfaces Operating as DTEs Connected to EIA RS-232-c or CCITT V.28 Point-to-Point DCEs", A. Kent

16.10 GLOSSARY

APA	Arithmetic Processing Accelerator
APT	Automatic Product Test - this is the manufacturing process which consists of a central host computer controlling the testing of systems or subsystems.
Console	The console-based subsystem used for diagnostic and runtime support. The term refers to the hardware and, sometimes, the software/firmware which runs in the hardware.
CPU	Central Processing Unit
C-RAM	Control RAM, which contains CPU microcode.
CTY	Console Terminal. The master controlling terminal connected to a running timesharing system.
D-RAM	Dispatch RAM
EBOX	Execution BOX. The major portion of the 2080 CPU concerned with instruction execution.
EPROM	Erasable (by ultraviolet light) Programmable Read Only Memory
F11	Fonz 11 - a three chip PDP-11 microprocessor
Hard-core Console	That portion of the console subsystem software which resides in the console EPROM.
IBOX	Instruction BOX. The major portion of the 2080 CPU logic which is concerned with instruction fetching and decoding.
2080 program	A program running in the 2080 CPU.
I/O	Input/Output
Kb	Kilobaud - one thousand bits per second.
KB	Kilobytes - one thousand bytes
KLDCP	The KL10 Diagnostic Control Program which runs in the front end.

KLINIK
The remote diagnosis technique used by Field Service.

Loadable Console
That portion of the console subsystem software which resides in the first 38KB of console RAM. It provides all the console functionality except that which is required to run and communicate with an 2080 program.

MACY11
The PDP-11 assembler.

MBOX
Memory BOX. The major portion of the 2080 CPU logic concerned with interfacing to main memory and the I/O Bus.

MOS Memory
A volatile memory using Metal Oxide Semiconductor technology.

ODT
Octal Debugging Technique.

PROM
Programmable Read Only Memory. EPROMs are often referred to as simply PROMs.

RAM
Random Access Memory.

Resident Console
That portion of the console subsystem software which comprises the hard-core console and the loadable console.

RSP
The Run-time Support Program. This is that portion of the console subsystem software which is specific to running and communicating with an 2080 program.

SPEAR
The system error reporting program.

VAX
Virtual Address eXtension. The new family of DEC 32-bit machines.

16.11 APPENDIX

16.11.1 2080 Program/RSP Communication Page

Communication page location definitions:

100	2080 program-to-RSP Control Word (2080 program writes this word)
101	2080 Keep Alive Word (2080 program writes this word)
102	Klinik/APT Output Word (to RSP from 2080 program)
103	Terminal 1 Output Word (to RSP from 2080 program)
104	Terminal 2 Output Word (to RSP from 2080 program)
105	Terminal 3 Output Word (to RSP from 2080 program)
106	Pseudo-TTY/Pseudo-ERR Device Output Word (to RSP from 2080 program)
107	2080 program Status Word
110	RSP Status Word (RSP writes this word)
111	RSP Keep Alive Word (RSP writes this word)
112	Klinik/APT Input Word (to 2080 program from RSP)
113	Terminal 1 Input Word (to 2080 program from RSP)
114	Terminal 2 Input Word (to 2080 program from RSP)
115	Terminal 2 Input Word (to 2080 program from RSP)
116	Pseudo-TTY/Pseudo-ERR Device Input Word (to 2080 program from RSP)
117	Console Status Word
125	Floppy Disk Control/Status Word
126	Floppy Disk Control Word
132	Time of Day [T.O.D.] (written by RSP on request of 2080 program)

2080 program-to-RSP Control Word Format (Word 100)

Bits	0-31	Zeros
Bits	32-35	Interrupt Code

0	RSP ready to accept next control function (always set to zero by RSP)
1	Output for Console Floppy Disk
2	Output for Console KLINIK Line
3	Output for Console Terminal 1
4	Output for Console Terminal 2
5	Output for Console Terminal 3
6	Output for Console pseudo-TTY
7	Request for Console to set T.O.D.
10	Request for Console to Reload Monitor
11	Request for Console to reload itself
12	2080 program Keep Alive is Active
13	2080 program Keep Alive is Inactive

KLINIK/APT Output Word Format (Word 102)

Bits	1-3	Interrupt Code

001	"1 Byte Mode" ;a single ASCII byte is being sent
010	Break request
100	"7-Bit ASCII Mode" ; Five 7-bit ASCII characters are packed left-adjusted.
111	Hang Up request

Bits	9-35	"Physical Address" ;Used in Mode 100 above
Bits	29-35	"ASCII character" ;Used in Mode 001 above

Terminal 1,2 and 3 Output Word Format (Words 103, 104, and 105)

Bits	1-3	Interrupt Code
	001	"1 Byte Mode" ;a single ASCII byte is being sent
	010	Break request
	100	"7-Bit ASCII Mode" ; Five 7-bit ASCII characters are packed left-adjusted.
Bits	9-35	"Physical Address" ;Used in Mode 100 above
Bits	29-35	"ASCII character" ;Used in Mode 001 above

Pseudo-TTY/Pseudo-ERR Device Output Word Format (Word 106)

Bits	1-3	Interrupt Code
	001	"1 Byte Mode" ;a single ASCII byte is being sent
	100	"7-bit ASCII Mode" ; Five 7-bit ASCII characters are packed left-adjusted.
Bits	9-35	"Physical Address" ;Used in Mode 100 above
Bits	29-35	"ASCII character" ;Used in Mode 001 above

2080 2080 program Status Word Format (Word 107)

Bit 35	2080 program 'Keep-Alive' - 1 = Active, 0 = Inactive

```
RSP Status Word Format (Word 110)

     Bit      35       Done     0 = Status is input to 2080 program (such as CT
                                    input char)
                                1 = Status is response to previous 2080 program
                                    Output Request (such as CTY typeout)

     Bits     31-34    Device Number/Interrupt Code

              1        Console Floppy Disk
              2        Console KLINIK Line
              3        Console Terminal 1
              4        Console Terminal 2
              5        Console Terminal 3
              6        Console Pseudo-TTY
              7        Immediate Abort - Environmental Problem
              10       Please Stop - Environmental Problem
              11       RSP Keep Alive is Active
              12       RSP Keep Alive is Inactive
              13       Console Status Change
                                - new status is in Console Status word




KLINIK/APT Input Word Format (Word 112)

     Bit      0        Error - error code is in bits 22-28

     Bits     1-3      Interrupt Code

              001      "1 Byte Mode" ;a single ASCII byte is being sent
              111      "Status change" ;code indicating new status found
                                in bits 29-35

     Bits     22-28    "Error Code"; used if bit 0 is set

              001      Input Buffer Overflow
              002      UART Framing Error
              004      UART Parity Error
              010      UART Overrun Error

     Bits     29-35    "ASCII character" ;Used in Mode 001 above

     Bits     29-35    "Status change" ;Used in Mode 111 above

              001      Carrier detect
              002      Carrier loss
              004      KLINIK active
              010      KLINIK inactive
```

Terminal 1,2 and 3 Input Word Format (Word 113, 114 and 115)

Bit	0	Error - error code is in bits 22-28
Bits	22-28	"Error Code"; used if bit 0 is set
	001	Input Buffer Overflow
	002	UART Framing Error
	004	UART Parity Error
	010	UART Overrun Error
Bits	29-35	"ASCII character"

Pseudo-TTY/Pseudo-ERR Device Input Word (Word 116)

Bit	0	1 if error logging information is being sent 0 if pseudo-TTY characters are being sent
Bits	1-3	Interrupt Code
	001	"1 Byte Mode"; a single ASCII byte is being sent
	100	"7-bit ASCII Mode"; five 7-bit ASCII characters are packed left-adjusted.
Bits	9-35	Physical Address; used in Mode 100 above
Bits	29-35	ASCII character; used in Mode 001 above

Console Status Word Format (Word 127)

Bit	35	RSP "Keep-Alive"; 1 = Active, 0 = Inactive
Bit	34	KLINIK status; 1 = Active, 0 = Inactive
Bit	33	Environmental status; 1 = Caution, 0 = No caution

Floppy Disk Control/Status Word Format (Word 125)
To be supplied

16.11.2 Console Commands

"hard-core" Console Commands

(#) <CR> - modify the open location with (#) and close the open location.

(#) <LF> - modify the open location or general purpose register (GPR) with (#) and close the open location or GPR. Then open location+2 or GPR+1.

"/" - open the location specified by # and type out the 'contents'.

"$" # "/" - open the general purpose register specified by # and type out the 'contents'.

"$" "S" "/" - open the processor status word (PSW) and type out the 'contents'.

(#) "^" - modify the open location or GPR with (#) and close the open location or GPR. Then open location-2 or GPR-1.

(#) "@" - modify the open location or GPR with (#) and close the open location or GPR. Then open the location pointed to by the original 'contents' of the opened location or GPR. The "@" command will wrap around to location 0 if 'contents' was 177777 or to GR0 if 'contents' was 7.

(#) "<" - modify the open location or GPR with (#) and close the open location or GPR. Then open the relative location pointed to by the original location + 'contents' + 2. This command will not perform the relative mode operation on GPR or PSW registers.

"G" <CR> - start program execution at memory location specified by #. PS register is cleared.

"B" <CR> - set the non-deleting breakpoint at the address specified by #.

"B" <CR> - list the address of the set breakpoint. If breakpoint is not set, ODT will respond with a ?.

0"B" <CR> - delete the set breakpoint.

"P" <CR> - proceed from set breakpoint. If a breakpoint is not set, ODT will respond with a ?.

10"S" <CR> - set default load device to be the console floppy disk.

20"S" <CR> - set default load device to be the KLINIK/APT line.

"S" <CR> - type out the current default load device.

"L" "filename.ext" <CR> - load console ram from the default device with filename.ext.

"R" "filename.ext" <CR> - load console ram from the default device with filename.ext and run the loaded program.

"RUBOUT" - deletes the previous inputted character and echos a \ (backslash). "RUBOUT" will not delete commands <CR>, <LF>, /.

Symbols:

- represents last 6 inputted octal digits < or = 177777. For example, if 72320675 is typed in, then # = 20675.

(#) - optional octal number.

" " - specifies some odt command.

'contents' - octal data pointed to by some octal number.

filename.ext - specifies a loadable file from the default device. it is 10 characters long, filename=6 chr. and ext=3 chr.

? - is typed out whenever an invalid command is typed on the console terminal.

"Loadable" Console Commands

Guide words are indicated by parentheses and optional fields by square brackets.

```
SET CLOCK NORMAL
          HIGH
          HALF
```

This command provides for the setting of the system clock rate - normal, margined high, or one-half normal rate.

```
SET TERMINAL TYPE type
             WIDTH w
             LENGTH l
             SPEED (INPUT) i (OUTPUT) o
```

This commands provides for the setting of terminal characteristics.

```
SET PASSWORD (FOR KLINIK) password
```

This command provides for the setting of the KLINIK password. The password is limited to six alphanumeric characters.

```
SET BOOT (DEVICE TO BE) FLOPPY
                        KLINIK/APT
```

This command provides for the selecting of the load device to be used for autobooting the console and 2020 system.

```
SET LOAD (DEVICE TO BE) FLOPPY
                        KLINIK/APT
```

This command provides for the selecting of the load device to be used in default situations. For example, when a user types the command 'LOAD filename, the console will look for filename on the default load device.

```
SET TIME (OF CENTURY) dd-mm-yy hh:mm:ss
```

This command provides for the setting of the time of century. Since there is a battery backed-up time of century clock in the hardware, this command should rarely be needed. Confirmation will always be required before the command will be executed.

```
SET [NO] CONFIRMATION
```

| This command provides for enabling or disabling the
| requirement for confirmation on those commands which affect a
| running 2080.

SET [NO] AUTORELOAD

This command provides for the enabling or disabling of the automatic reloading of the system upon powerup.

```
SET VOLTAGE (MARGIN) NORMAL
                     HIGH
                     LOW
```

This command provides for the margining of the system ECL power.

SET LINE n SPEED (INPUT) i (OUTPUT) o

This command provides for the setting of the line speeds for any of the four asynchronous lines attached to the console. One use of it will be to select the speeds for the KLINIK line.

```
ENABLE  HARDCOPY (AS LOGGING DEVICE)
DISABLE
```

This commands enables or disables the hardcopy as a logging device for all input and output at the video terminal or over the KLINIK line.

```
READ (CPU REGISTER) CPU
                    PC
                    VMA
                     .
                     .
                     .
```

This command provides for the reading of the CPU registers which are contained in the diagnostic scan paths. Specifying the argument CPU will cause the most commonly read registers to be read and displayed, such as the PC and VMA. The complete list of readable registers will be defined later.

```
WRITE (CPU REGISTER) ???
                      .
                      .
                      .
```

This command provides for the writing of the CPU registers which are contained in the diagnostic scan paths. The complete list of writable registers will be defined later.

EXAMINE [address-type] [address]

This commands provides for the examining of memory in all parts of the 2080 CPU cluster. This includes the main MOS memory, all control RAMs, console memory and Port RAMs.

'address-type' specifies the memory space to be examined and can be any one of:

```
CONSOLE BYTE
        WORD
        REGISTER
AC
FAST-EBOX
SLOW-EBOX
IBOX
FPA
MBOX
DIVIDE-LOOKUP
EARLY-DECODE
FRU-LOOKUP
PORT n
MAIN-MEMORY PHYSICAL
```

'address' can be any one of:

```
NEXT
PREVIOUS
adr [ (COUNT) n ]
adr1:adr2
```

The NEXT argument will cause examination of the next location following the last location referenced in the last EXAMINE command executed. The PREVIOUS argument causes examination of the location before the last location referenced in the last EXAMINE command executed. Specifying 'adr' will cause that single location to be examined. By adding the COUNT field, the user can examine the location 'adr' plus the contents of the next n-1 locations. By specifying 'adr1:adr2' the user can examine the range of locations between and including 'adr1' and 'adr2'.

Both the 'address-type' and 'address' fields are optional. The value for each field is saved from the last executed EXAMINE command. Therefore, a user can type 'EXAMINE <CR>' to examine the same location in the same memory space that was examined by the last EXAMINE command. To examine a different location in the same memory space as the last EXAMINE command, the user includes only the 'address' field, e.g., 'EXAMINE NEXT'.

DEPOSIT [address-type] address data

This commands provides for the depositing of memory in all parts of the 2080 CPU cluster. This includes the main MOS memory, all control RAMs, console memory and Port RAMs.

'address-type' specifies the memory space to be examined and can be any one of:

```
CONSOLE BYTE
        WORD
        REGISTER
AC
FAST-EBOX
SLOW-EBOX
IBOX
FPA
MBOX
DIVIDE-LOOKUP
EARLY-DECODE
FRU-LOOKUP
PORT n
MAIN-MEMORY PHYSICAL
```

'address' can be any one of:

```
NEXT
LAST
PREVIOUS
adr [ (COUNT) n ]
adr1:adr2
```

The NEXT argument will cause the depositing of the next location following the last location referenced in the last DEPOSIT command executed. The LAST argument will cause the depositing of the same location that was referenced in the last DEPOSIT command executed. The PREVIOUS argument causes the depositing of the location before the last location referenced in the last DEPOSIT command executed. Specifying 'adr' will cause that single location to be deposited. By adding the COUNT field, the user can deposit the location 'adr' plus the contents of the next n-1 locations. By specifying 'adr1:adr2' the user can deposit the range of locations between and including 'adr1' and 'adr2'.

The 'address-type' field is optional. The value for that field is saved from the last executed DEPOSIT command. Therefore, a user can type 'DEPOSIT 1234 -1<CR>' to deposit the value -1 into the location 1234 in the same memory space that was deposited by the last DEPOSIT command.

```
LOAD memory-space filename.ext
     SYSTEM
```

This command provides for the loading of various system memory spaces with the contents of a file located on the console floppy. Specifying SYSTEM will cause all system microcodes to be loaded.

'memory-space' can be any one of:

```
MBOX
IBOX
FAST-EBOX
SLOW-EBOX
DIVIDE-LOOKUP
EARLY-DECODE
FRU-LOOKUP
PORT n
CONSOLE
PREBOOT
```

```
VERIFY memory-space filename.ext
       SYSTEM
```

This command provides for the verifying of various system memory spaces against the contents of a file located on the console floppy. Specifying SYSTEM will cause all system microcodes to be verified.

'memory-space' can be any one of:

```
MBOX
IBOX
FAST-EBOX
SLOW-EBOX
DIVIDE-LOOKUP
EARLY-DECODE
FRU-LOOKUP
PORT n
CONSOLE
PREBOOT
```

Doing a 'VERIFY SYSTEM' will cause a verification of all the system memory spaces. In this case, the contents of 'filnam.ext' must be an ASCII file containing the names of all the files to be verified. Doing a 'VERIFY PREBOOT' will cause the contents of the specified file to be verified against page zero of 2080 main memory.

```
START CONSOLE       address
      2080-PROGRAM
      IBCX
      SLOW-EBCX
      MBCX
      EPA
```

This command provides for the starting programs or microcodes at specified addresses. Typing 'START address' will default to 2080-PROGRAM.

```
START CLOCK
      PORT n
```

This command provides for the starting of the system or Port clocks.

```
STOP CLOCK
     PORT n
```

This command provides for the stopping of the system or Port clocks.

```
CYCLE (CLOCK) cycle-count
```

This command provides for the cycling of the system clock a specified number of cycles.

```
BURST (CLOCK) phase-count
```

This command provides for the bursting of the system clock a specified number of clock phases.

```
SHOW (ABOUT) TERMINAL n
             NON-VOLATILE RAM
             LOGICAL-COMMAND name
             DEFAULTS
             TIME-OF-CENTURY
             VERSIONS
```

This command provides the ability to obtain information about certain console and system parameters.

```
RUN filename.ext
```

This command provides the ability to load a program from the console default load device into console memory and start it.

```
DEFINE (LOGICAL COMMAND) logical-name
        (type Control-Z to terminate command sequence)
        command
        command
            .
            .
            .
        control-z
```

This command provides the ability to define a unique logical name to be a specified sequence of commands. This command sequence can then be executed by using the logical name in the REPEAT command or by typing the logical name directly as a command. When the <CR> is typed, the user enters an input mode in which he can specify the command sequence to be defined. The format for the sequence is one command per line, each followed by a <CR>. The input mode is terminated by a Control-Z.

```
REPEAT [ repeat-count ] (COMMAND SEQUENCE)
        (type Control-Z to terminate command sequence)
        command
        command
            .
            .
            .
        control-z

        CR

REPEAT [ repeat-count ] logical-name
```

This command provides the ability to repeatedly execute a specified sequence of commands. A repeat count may be specified. If not, the sequence is repeated indefinitely until a Control-C is typed. If a predefined logical command (see DEFINE command) is specified, it will be executed in a repeat mode. If a <CR> is typed, the user enters an input mode in which he can specify the command sequence to be repeated. The format for the sequence is one command per line, each followed by a <CR>. The input mode is terminated by a Control-Z.

```
RESET SYSTEM
      IO
      IBOX-EBOX
      MBOX
      ERROR-FLOPS
      MEMORY-CONTROL
```

This command provides the ability to reset the entire 2080 system or selected parts of it.

```
ZERO (MAIN MEMORY) DIRECT
                   INDIRECT
```

This command provides the ability to zero all of the 2080 main memory. Specifying direct will cause the zeroing to be done via console DMA depositing into memory. Specifying INDIRECT will cause the zeroing to be done via the execution of a 2080 instruction.

ODT

This command causes the console to enter its ODT package.

EDIT filename.ext

This command invokes the console editor and loads the specified file into the editing buffer.

TAKE filename.CMD

This command executes the contents of the specified file as though it were input coming from the CTY.

```
ECHO (OF COMMAND FILE) ENABLED
                       DISABLED
```

This command enables and disables the echoing on the terminal of the execution of a command file (see TAKE command).

COPY (FILE) filename.ext (TO) filename.ext

This command copies a floppy disk file.

TYPE (FILE) filename.ext

This commands types the specified file on the user's terminal.

DIRECTORY (OF FILE) [filename.ext]

This command provides the ability to take a full directory of the console floppy or of a specified file.

RENAME (FILE) filename.ext (TO) filename.ext

This command renames a file on the console floppy.

DELETE (FILE) filename.ext

This command deletes the specified file from the console floppy.

PRINT (FILE) filnam.ext

This command prints the specified file on the console hardcopy terminal.

```
SAVE (CONSOLE RAM) adr:adr (TO FILE) filnam.ext
                   ALL
```

This command dumps the contents of a specified range or all of console RAM onto the specified floppy file.

```
START SYSTEM COLD
             WARM
             SOFT
```

This command will load and start the 2080 system in the specified manner. A COLD start will simulate the depressing of the console panel System Start switch except for the booting of the console. A WARM start will simulate a power-fail restart procedure except for booting of the console. A SOFT start will ?? All three options will cause the automatic loading and running of the FSP.

TERMINATE (KLINIK ACCESS)

This command will terminate a KLINIK access which is in progress. This means sending a warning message to the KLINIK user and hanging up of the KLINIK connection.

```
POWER ON
      OFF
```

This command will turn ECL power on or off to the 2080 system.

RSP Commands

KCCCN

This command leaves RSP and puts the user back to "loadable" console command level, CCN>. Any timesharing support activity in progress, such as keep-alive, will be gracefully aborted.

QUIT

This command put the user back to the last command level he was in before going to RSP command level. This may be either 'CCN>' prompt level or "System Mode" talking to a 2080 program.

HALT (2080 CPU)

This command halts the 2080 CPU by putting the 2080 microcodes back to their idle loops.

CONTINUE (2080 PROGRAM)

This command continues a halted 2080 program.

START 2080-PROGRAM address

This command starts a previously loaded 2080 program at the specified address.

EXECUTE (2080 INSTRUCTION) instr

This command will load and execute the specified instruction in the 2080 CPU. An accepted abbreviation for this command is 'XCT'.

SHUTDOWN (2080 PROGRAM)

This command will shutdown the running 2080 program by depositing a -1 into 2080 memory location 30.

CHANGE (ESCAPE CHARACTER TO) char

This command changes the escape character used to go from "System Mode" to "Console Mode".

SYSTEM (MODE)

This command causes the RSP to enter "System Mode". This includes attempting to establish communication (e.g., keep-alive) with the 2080 Program.

SINGLE (INSTRUCT 2080 CPU)

This command will single instruct the 2080 CPU.

SHOW

This command will display the status of all the available global parameters in use by the FSP.

16.11.3 EMTs and Subroutines

(to be supplied)

16.11.4 KLINIK Modes

```
                     +-------------+
                     ! CTY Output  !
                     +-------------+
          +---+
          ! U !                           COMM PAGE
          ! S !                      +----------------+
CTY ----->+ A +----------+           !  CTY Input     !
          ! R !          !           !     Word       !
          ! T !          !           +----------------+       +----------+
          +---+          !           !  CTY Output    !       !          !
                         !           !     Word       !       !   2020   !
                         !           +----------------+       ! program  !
          +---+          !           ! KLINIK Input   !       !          !
          ! U !          !           !     Word       !       !          !
          ! S !          V           +----------------+       +----------+
KLINIK -->+ A +--------->+           ! KLINIK Output  !
 line     ! R !          !           !     Word       !
          ! T !          !           +----------------+
          +---+          V
                     +-------------+
                     ! CTY Input   !
                     +-------------+

--------------------------------------------------------------------------

                     +-------------+
                     ! CTY Output  !
                     +-------------+
          +---+          !
          ! U !          !                COMM PAGE
          ! S !          V           +----------------+
CTY <-----+ A +<---------+           !  CTY Input     !
          ! R !          !           !     Word       !
          ! T !          !           +----------------+       +----------+
          +---+          !           !  CTY Output    !       !          !
                         !           !     Word       !       !   2020   !
                         !           +----------------+       ! program  !
          +---+          !           ! KLINIK Input   !       !          !
          ! U !          !           !     Word       !       !          !
          ! S !          !           +----------------+       +----------+
KLINIK <--+ A +<---------+           ! KLINIK Output  !
 line     ! R !                      !     Word       !
          ! T !                      +----------------+
          +---+
                     +-------------+
                     ! CTY Input   !
                     +-------------+
```

KLINIK Console Mode

```
                   +------------+
                   ! CTY Output !
                   +------------+
          +---+
          ! U !                            COMM PAGE
          ! S !                    +---------------+
CTY ----->+ A +--------->+------->! CTY Input     !---+
          ! R !          ^         !   Word        !   !
          ! T !          !         +---------------+   !   +----------+
          +---+          !         ! CTY Output    !   !   !          !
                         !         !   Word        !   !   !  2080    !
                         !         +---------------+   +-->! program  !
          +---+          !         ! KLINIK Input  !       !          !
          ! U !          !         !   Word        !       !          !
          ! S !          !         +---------------+       +----------+
KLINIK -->+ A +----------+         ! KLINIK Output !
 line     ! R !                    !   Word        !
          ! T !                    +---------------+
          +---+
                   +------------+
                   ! CTY Input  !
                   +------------+

------------------------------------------------------------------------

                   +------------+
                   ! CTY Output !
                   +------------+
          +---+
          ! U !                            COMM PAGE
          ! S !                    +---------------+
CTY <-----+ A +<---------+         ! CTY Input     !
          ! R !          !         !   Word        !
          ! T !          !         +---------------+       +----------+
          +---+          +<--------! CTY Output    !<------!          !
                         !         !   Word        !       !  2080    !
                         !         +---------------+       ! program  !
          +---+          !         ! KLINIK Input  !       !          !
          ! U !          !         !   Word        !       !          !
          ! S !          !         +---------------+       +----------+
KLINIK <--+ A +<---------+         ! KLINIK Output !
 line     ! R !                    !   Word        !
          ! T !                    +---------------+
          +---+
                   +------------+
                   ! CTY Input  !
                   +------------+
```

KLINIK System Mode

```
                 +------------+
                 ! CTY Output !
                 +------------+
          +---+
          ! U !                     COMM PAGE
          ! S !                 +---------------+
CTY ------+ A +                 !  CTY Input    !
          ! R !                 !     Word      !
          ! T !                 +---------------+       +----------+
          +---+                 !  CTY Output   !       !          !
                                !     Word      !       !  2080    !
                                +---------------+       ! program  !
          +---+         +------>! KLINIK Input  !------>!          !
          ! U !         !       !     Word      !       !          !
          ! S !         !       +---------------+       +----------+
KLINIK -->+ A +---------+       ! KLINIK Output !
 line     ! R !                 !     Word      !
          ! T !                 +---------------+
          +---+
                 +------------+
                 ! CTY Input  !
                 +------------+

----------------------------------------------------------------------

                 +------------+
                 ! CTY Output !
                 +------------+
          +---+
          ! U !                     COMM PAGE
          ! S !                 +---------------+
CTY ------+ A +                 !  CTY Input    !
          ! R !                 !     Word      !
          ! T !                 +---------------+       +----------+
          +---+                 !  CTY Output   !       !          !
                                !     Word      !       !  2080    !
                                +---------------+   +---! program  !
          +---+                 ! KLINIK Input  !   !   !          !
          ! U !                 !     Word      !   !   !          !
          ! S !                 +---------------+   !   +----------+
KLINIK <--+ A +<----------------! KLINIK Output !<--+
 line     ! R !                 !     Word      !
          ! T !                 +---------------+
          +---+
                 +------------+
                 ! CTY Input  !
                 +------------+
```

KLINIK User Mode