EK-OKL10-TM-002

KL10-Based Technical Manual

digital

EK-OKL10-TM-002

KL10-Based Technical Manual

Prepared by Educational Services
of
Digital Equipment Corporation

1st Edition, August 1983
2nd Edition, July 1984

The material in this manual is for informational purposes and is subject to change without notice.

Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The manuscript for this book was created on a DIGITAL Word Processing System and, via a translation program, was automatically typeset on DIGITAL's DECset Integrated Publishing System. Book production was done by Educational Services Development and Publishing in Marlboro, MA.

The following are trademarks of Digital Equipment Corporation:

digital	MASSBUS	TOPS-20
DEC	PDP	UNIBUS
DECmate	P/OS	VAX
DECsystem-10	Professional	VMS
DECSYSTEM-20	Rainbow	VT
DECUS	RSTS	Work Processor
DECwriter	RSX	
DIBOL	TOPS-10	

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

TABLES

FIGURES

PREFACE

This manual contains the system-level technical description of the DECsystem-10/DECSYSTEM-20 computer, the KL10-R-based machine. The manual provides an integrated hardware/software system display with appropriate overviews of timesharing/batch operation and system features. The chapters of this system-level description are as follows.

1. Introduction
2. System Features
3. The Hardware
4. The Software

CHAPTER 1
INTRODUCTION

1.1 DECsystem-10/DECSYSTEM-20 TECHNICAL INTRODUCTION

1.1.1 Purpose and Use

The KL10-R is a version of the present KL-based DECsystem-10/DECSYSTEM-20 system. It offers a complete system with large capacity memories and increased disk, tape, and user capability. The software, which supports extended addressing, provides service for 40 to 80 users. Ongoing support and development for current customers protect their investment in TOPS-20 and application software already developed. Figure 1-1 shows a typical DECsystem-10/DECSYSTEM-20 configuration.

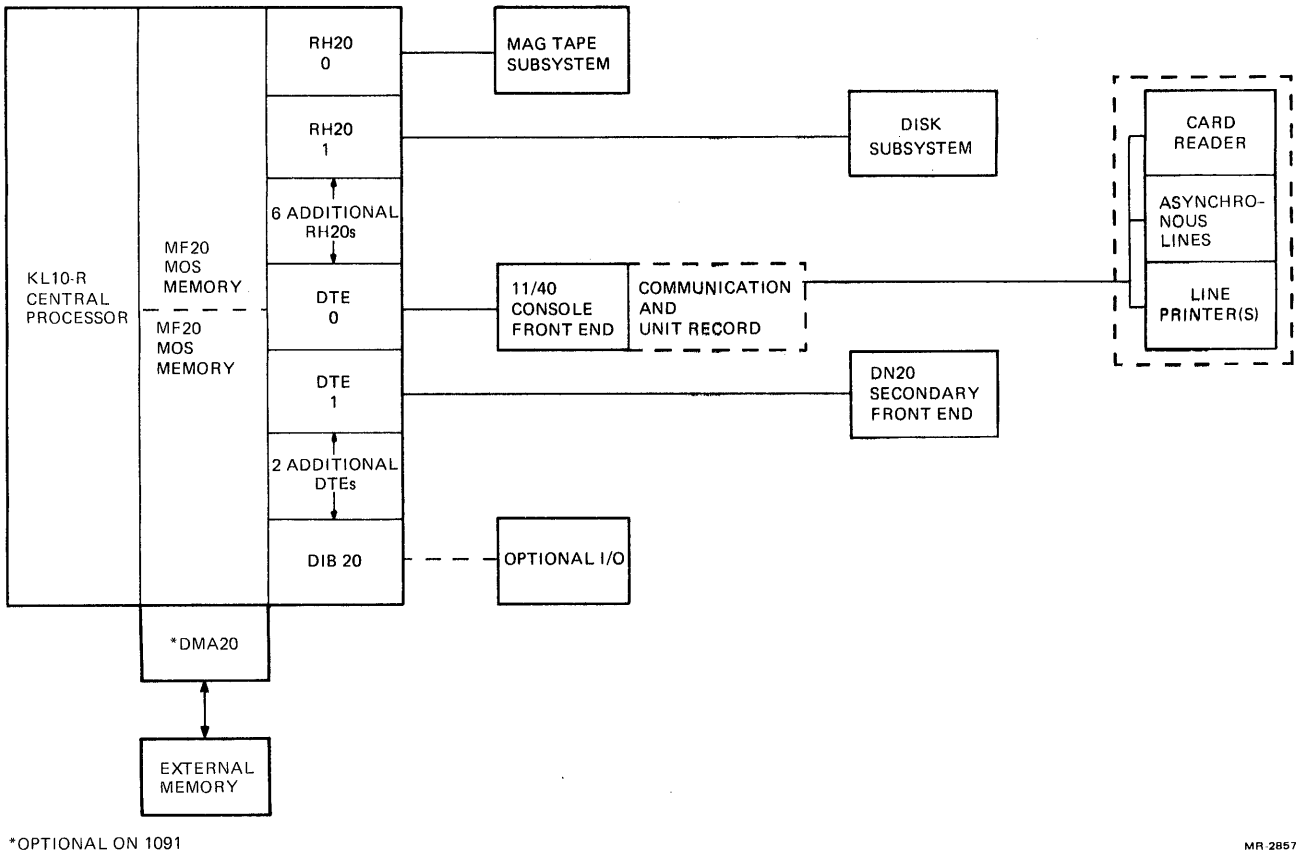


Figure 1-1 DECsystem-10/DECSYSTEM-20 Configuration

1.1.2 General Description

The DECsystem-10/DECSYSTEM-20 is a large-scale, high-performance system (DECsystem-10/DECSYSTEM-20 family) using a KL10-R central processing unit (see Figure 1-2). It contains at least 512 K words of memory, TOPS-20 extended features, and both hardware and software support for MOS memory.

The following features are included within the mainframe.

- Microprogrammed KL10-R CPU
- Minimum memory size of 512 K words MOS (except 1091)
- Internal data channels
- Internal Massbus controllers
- Integral PDP-11/40 front-end processor (supports asynchronous communication and unit-record equipment)
- Optional secondary front-end communications

1.1.3 Related Documents

1.1.3.1 Hardware Documentation – The following documents are available only on microfiche except where identified. Hard copy documents can be ordered from:

Digital Equipment Corporation
444 Whitney Street
Northboro, Massachusetts 01532
Attention: Printing and Circulating Services (NR2/M15)
Customer Services Section

Title	Document Number
AN10/AN20 ARPANET Interface Technical Manual	EK-AN1/2-TM
CD20 Maintenance Manual	ED-CD11-TM
DECsystem-10/DECSYSTEM-20 Front End (Console Channel Interface Description)	EK-FE-ID
DIA20 IBus Adapter Unit Description	EK-DIA20-TM
*DTE20 Unit Description**	EK-DTE20-UD
DN200 Remote Batch Station, Terminal Concentrator (DECSYSTEM-10, -20) Remote Station	EK-DN200-TM
DX20/DX20F Programmed Device Adapter Technical Manual	EK-ODX20-TM

*Microfiche and hard copy
†Hard copy only

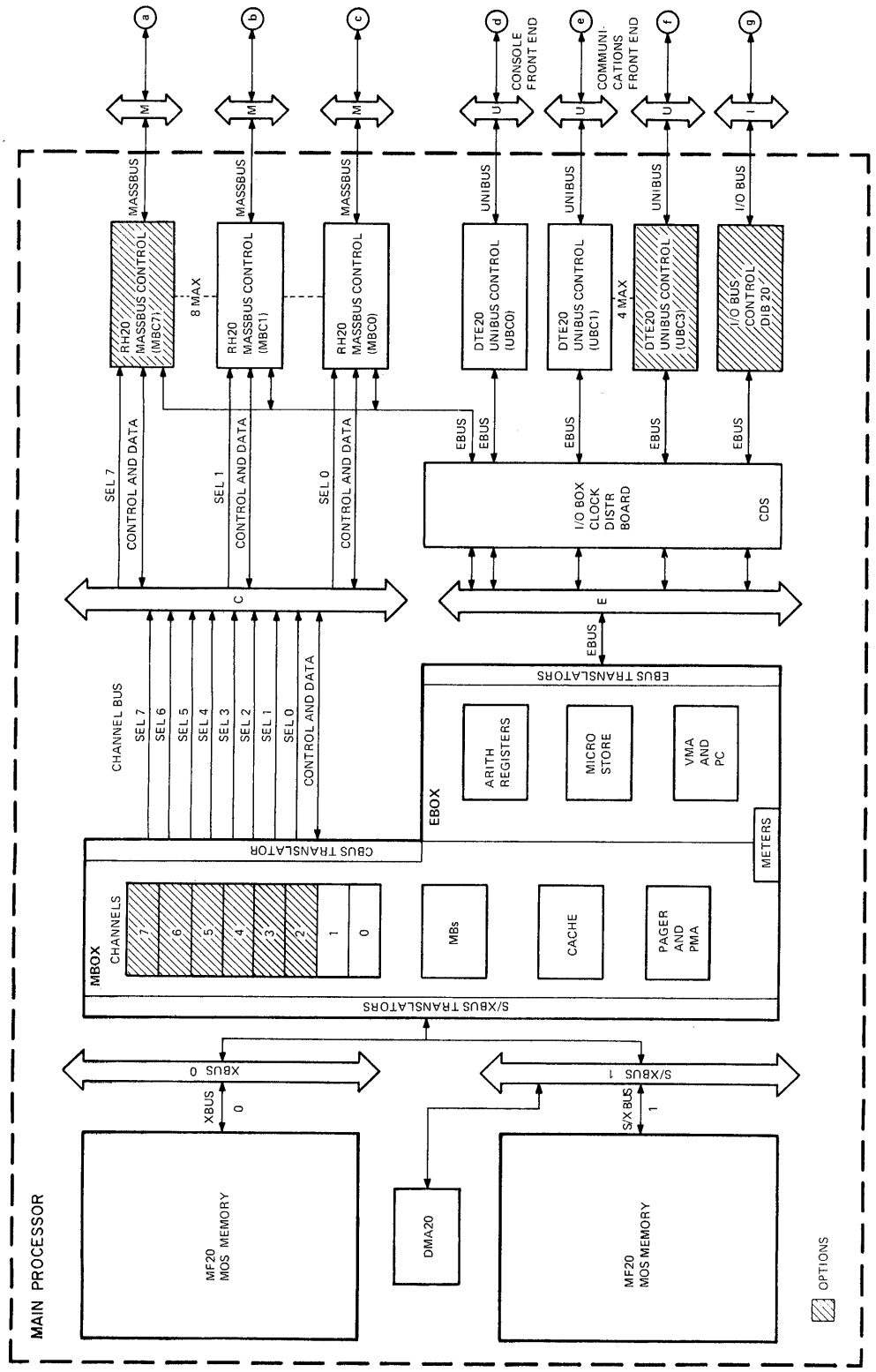


Figure 1-2 DECsystem-10/DECSYSTEM-20 Typical Block Diagram (Sheet 1 of 2)

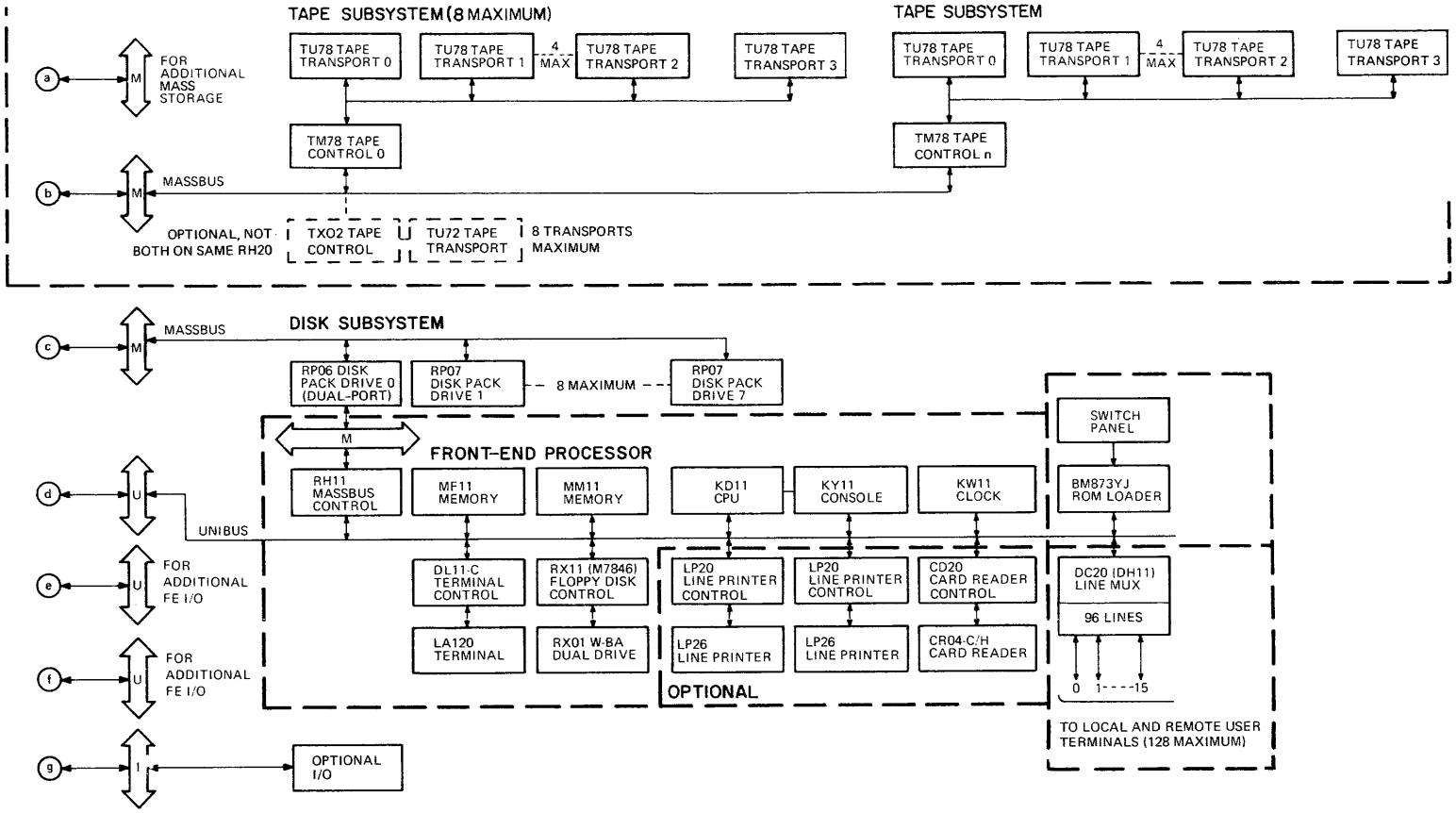


Figure 1-2 DECsystem-10/DECSYSTEM-20 Typical Block Diagram (Sheet 2 of 2)

*EBox Unit Description**	EK-EBOX-UD
Hardware Reference Manual†	AA-H391A-TK
KL10-Based Site Preparation, Power System, and Installation Manual†	EK-0KL10-SP
KL10 Maintenance Guide†	EK-0KL10-MG
KLINIK User's Guide for KL10-Based Systems	EK-KLIN-UG
*MBox Unit Description**	EK-MBOX-UD
*MF20/MF20F MOS Memory Subsystem Technical Manual**	EK-0MF20-TM
Packet Switching System Layout Kit†	EK-PACSS-LK
Packet Switching System Site Prep Guide†	EK-PACSS-SP
*RH20 Unit Description**	EK-RH20-UD

1.1.3.2 Software Documentation – Software documents can be ordered from:

Software Distribution Center
Digital Equipment Corporation
444 Whitney Street
Northboro, Massachusetts 01532

Title	Document Number
Operator's Guide	AA-4176C-TM
Operator's Guide Update	AD-4176C-TM
RSX-20F System Specifications	AA-H213A-TK
Software Installation Guide	AA-4195F-TM
SPEAR Manual	AA-J833A-TK
SPEAR Reference Card	EY-SPEAR-RC
System Manager's Guide	AA-4169E-TM
User Environment Test Package Reference Manual (UETP)	AA-D606A-TM

*Microfiche and hard copy
†Hard copy only

1.2 DECsystem-10/DECSYSTEM-20 SITE PREPARATION AND PLANNING

1.2.1 DECsystem-10/DECSYSTEM-20 Installation

The KL10-Based Site Preparation, Power System, and Installation Manual applies to this installation. The mechanical, environmental, and power requirements of various units that may be configured into a DECsystem-10/DECSYSTEM-20 are found in this manual. For information about equipment not included in these procedures, refer to the appropriate option manual, field service print set, and engineering installation/test specifications.

1.2.2 DECsystem-10/DECSYSTEM-20 Operation/Programming

Field service diagnostic software support for the DECsystem-10/DECSYSTEM-20 is provided by the KLAD-10/20 diagnostic pack. The version required for this system must contain the diagnostics that support minimum revision level 4.

For system-level diagnostic support of most peripheral devices, the normal exec mode diagnostic method is used in addition to user mode diagnostics. User mode diagnostics are restricted to memory, disk, and tape subsystems, and secondary communication front ends.

The TOPS-20 operating system provides entries in the system's SPEAR log file and the TGHA files. In addition to the previously explained devices, SPEAR supports error logging information relative to network environments.

1.2.3 DECsystem-10/DECSYSTEM-20 Service

The maintenance philosophy for DECsystem-10/DECSYSTEM-20s using the KL10-R processor is the same as for previous DECsystem-10/DECSYSTEM-20s. There have been no major logic changes.

Maintenance manuals for Digital manufactured equipment are available only on microfiche (except as identified in the list of related documents). Vendor manuals for purchased options will be sent with the system.

Field service engineering print sets are provided with the system. These print sets contain the unit assembly drawings for Digital manufactured subassemblies and show any features unique to the DECsystem-10/DECSYSTEM-20.

1.2.4 DECsystem-10/DECSYSTEM-20 Engineering Drawing Sets

Title	Print Number
KL10-R Field Maintenance Print Set	MP01708
DX20F Field Maintenance Print Set	MP01709
DC20F Field Maintenance Print Set	MP01710
MF20F Field Maintenance Print Set	MP01711
DN20F Field Maintenance Print Set	MP01712

CHAPTER 2
SYSTEM FEATURES

2.1 INTRODUCTION

The central processing unit (CPU) uses high-speed ECL logic, a microprogrammed instruction set, an automatic paging buffer, an automatic data buffer, and up to eight high-speed integral data channel processors. The system also features up to four front-end processing channels and up to 1.5 megawords of internal MOS memory. All of these functional elements are housed in three cabinets.

Besides the state-of-the-art hardware, the DECsystem-10/DECSYSTEM-20 features an advanced timesharing/batch operating system that allows both timesharing and batch users to have the benefit of system resources as if it were a system dedicated to each user. The software system also features advanced high-level languages, data base management facilities, and utilities.

2.2 INSTRUCTION SET

The DECsystem-10/DECSYSTEM-20 offers 398 instructions, a very large set that provides the flexibility required for special computing problems. The instruction set is microprogrammed; that is, each instruction is actually a series of microinstructions that perform various logical functions such as processor state control, data path control, and the actual execution of each instruction. The microcode is loaded into a 2 K words by 84-bit RAM (random-access memory) through the PDP-11 front-end processor. Since the set provides so many instructions to select from, fewer instructions are needed to perform a given function. Assembly language programs are, therefore, shorter than with other computers; and the instruction set simplifies the monitor, language processors, and utility programs. For example, compiled programs on a DECsystem-10/DECSYSTEM-20 are often 30 percent to 50 percent shorter, require less memory, and execute faster than those of corresponding computers.

In addition to the instructions, the DECsystem-10/DECSYSTEM-20 features 64 programmable operators, 33 of which trap to the monitor (monitor calls) and 31 of which trap to the user's area. An attempt to execute one of these unimplemented instructions results in a trap to the monitor.

The instruction set, regardless of its size, is easy to learn. It is logically arranged into families of instructions and the mnemonic code is constructed modularly (Figure 2-1). All instructions are capable of directly addressing a full 256 K words (36-bit) of memory without sorting again to base registers, displacement addressing, or indirect addressing. Instructions may, however, use indirect addressing with indexing to any level. Most instruction classes, including floating-point, allow immediate mode addressing, where the result of the effective address calculation is used directly as an operand in order to save storage and speed execution.

Although there are a large number of instructions, they may be broken into easily learned, logically completed groups (Figure 2-1). To show this, the group to move full words (36 bits) takes the form shown in Figure 2-2.

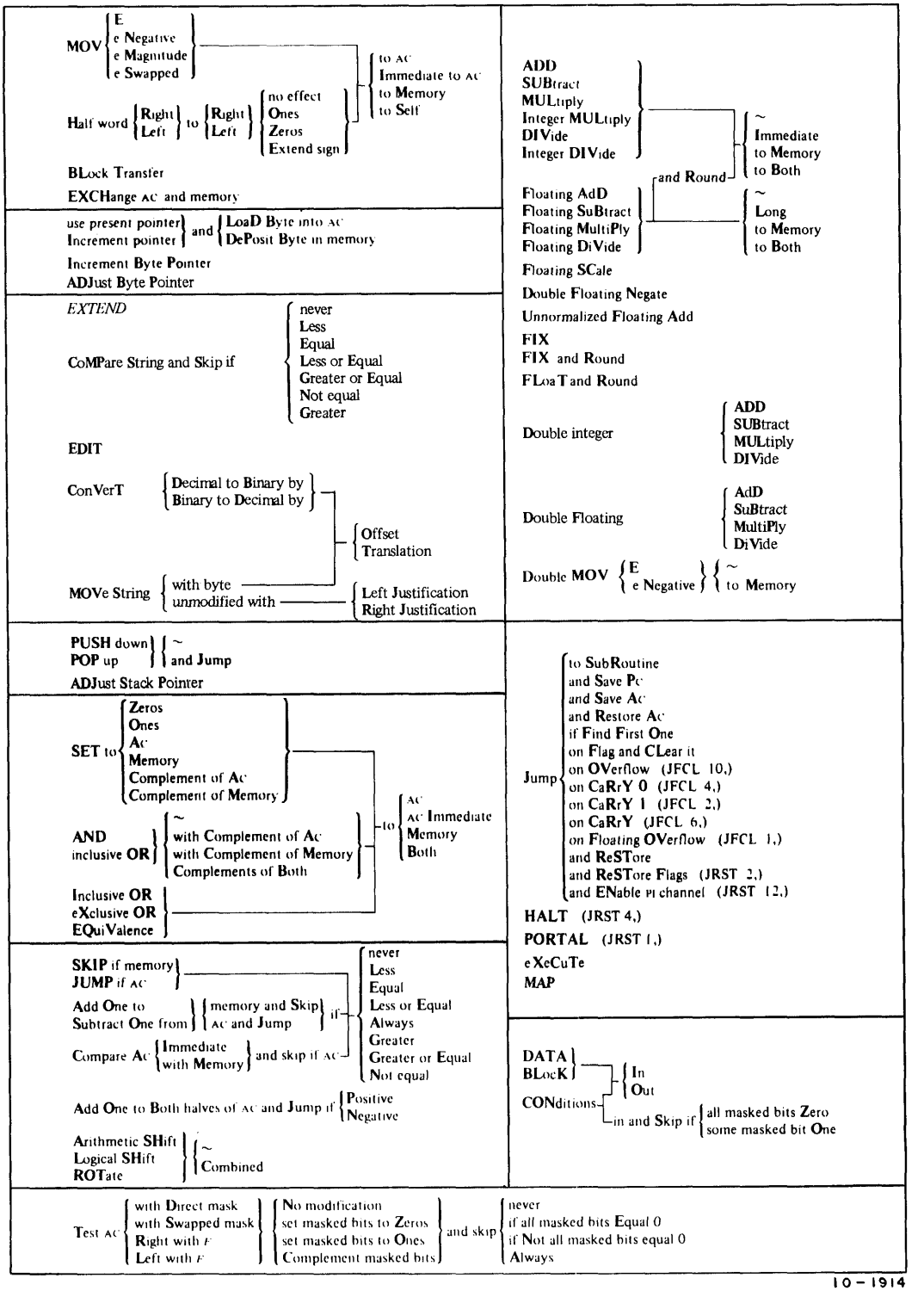


Figure 2-1 Instruction Set Constructs

To form a useful instruction, the following steps are taken.

1. The basic operator MOV is chosen. This specifies a full-word move.

2. One of four modifiers is chosen from the operator 1 group. This group specifies how the word is to be modified while being moved, where:

   - E = No modification
   - N = Take two's complement
   - M = Take magnitude
   - S = Swap left and right halves.

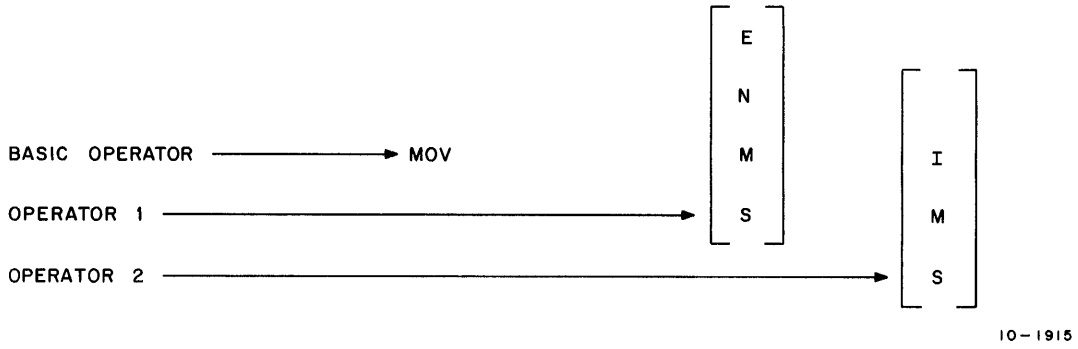


Figure 2-2 Move Instruction Construct

3. One of four modifiers is selected from the operator 2 group. This group specifies where the word is fetched from and where it is placed, therefore:

   - (Blank) = No memory to the accumulator
   - I = Take the 18-bit address as operand (immediate mode)
   - M = Accumulator to memory
   - S = Memory to memory (self).

This simple example illustrates the power and flexibility of the entire DECsystem-10/DECSYSTEM-20 instruction set. With one instruction class, a total of 16 instructions has been made. For example:

- MOVE AC, ADR moves the content of ADR to the specified AC.
- MOVEM AC, ADR moves the content of the specified AC to ADR.
- MOVEI AC, 5 moves the number 5 to AC.
- MOVNM AC, ADR moves the complemented content of AC or ADR.

A complete specification for the DECsystem-10/DECSYSTEM-20 instruction set is given in the *Hardware Reference Manual.*

2.2.1 Full-Word Data Transmission

The full-word data transmission instructions move one or more full words of data from one place to another. The instructions may perform minor arithmetic operations, such as making the negative (two's complement) or the magnitude of the word being processed.

2.2.2 Half-Word Data Transmission
The half-word data transmission instructions move a half word and may modify the contents of the other half of the destination location. There are 16 instructions that differ in the direction that they move the chosen half word and in the way in which they modify the other half of the destination location.

2.2.3 Block-Transfer Instruction
The block-transfer instruction facilitates the saving of accumulators or moving of blocks of memory from one set of contiguous locations to another. This instruction works for any block size, moving the block from any location to any other location.

2.2.4 Byte Manipulation
The five byte manipulation instructions pack or unpack bytes of any length anywhere within a word. In some systems, byte manipulation refers to 6-bit or 8-bit bytes. For the DECsystem-10/DECSYSTEM-20, byte manipulation refers to bytes of any size from 0 bits to a full word (36 bits). Note that ASCII is a 7-bit code, and on the DECsystem-10/DECSYSTEM-20, 7-bit bytes are efficiently stored 5 to a word. All the byte instructions on the DECsystem-10/DECSYSTEM-20 use a byte pointer that allows addressing of any size byte in any position in any of the 262,144 addressable words. Furthermore, both load and deposit byte instructions have a condition for automatic byte incrementation.

2.2.5 Business Instruction Set
Five instructions make up the business instruction set in the DECsystem-10/DECSYSTEM-20 central processor. Four of these are new arithmetic instructions to add, subtract, multiply, and divide using double-precision, fixed-point operands. The new extend (string) instruction is capable of performing nine separate functions.

These functions include an edit capability; decimal-to-binary and binary-to-decimal conversion in both offset and translated mode; move string in both offset and translated mode; and compare string in both offset and translated mode. Offset mode is byte modification by addition of the effective address of the string instruction to the source byte string; translated mode is byte modification by translation through a table of half words located at the effective address of the string instruction. This also occurs in edit. In addition to providing the translation function, those instructions that use translation can control the flags in ACs and can detect special characters in the source string.

This business instruction set provides faster processing since there are specific instructions for doing more comprehensive string operations. These instructions can be used on a series of code types including ASCII, EBCDIC, and so on.

2.2.6 Logic Instructions
The logic instructions provide for shifting and rotating, as well as performing the complete set of 16 Boolean functions on two variables.

2.2.7 Fixed-Point Arithmetic
Fixed-point arithmetic is handled in two's complement notation with 36-binary-bit accuracy (10 decimal digits). Mode options include immediate to accumulator, to memory, or to both with a result. Three classes of shifting include arithmetic, logical, and rotating operations to single- or double-word accumulators.

2.2.8 Floating-Point Arithmetic
The floating-point arithmetic instructions include instructions to perform scaling, negating (form two's complement), addition, subtraction, multiplication, and division upon numbers in single- and double-precision, floating-point format. In the single-precision, floating-point format, 1 bit is reserved for the sign, 8 bits are used for the exponent, and 27 bits are used for the fraction. In double-precision, floating-point format, 1 bit is used for the sign, 8 bits are used for the exponent, and 62 bits are used for the fraction.

2.2.9 Arithmetic Operation Modes

All of the DECsystem-10/DECSYSTEM-20 arithmetic operations — floating-point as well as fixed — and Boolean (logical) operations have options allowing the storage location for the result of the operation to be specified in the selected accumulator, in the addressed memory location, or in both. All may take their immediate address as an operand.

2.2.10 Fixed/Floating Conversions

Special instructions provide for converting fixed-point formats to or from floating-point formats. Two sets of instructions are provided to perform this function, one set optimized for FORTRAN and a second set optimized for ALGOL.

2.2.11 Compare and Modify

The compare and modify instruction set is large (128 instructions) and extremely flexible. Half of these are arithmetic compare and modify instructions, which may compare two numbers or compare the content of an accumulator or a memory word to zero and skip or jump accordingly. It is also possible to increment or decrement the word being tested and copy the modified word into an accumulator, all in a single instruction. In all cases of arithmetic comparisons, any one of the eight possible ordering relations on two variables may be specified, namely, if X and Y are the variables, $X = Y$, $X \neq Y$, $X > Y$, $X \geq Y$, $X < Y$, $X \leq Y$, true, and false.

The remaining 64 codes are logical compare and modify instructions that allow a variety of choices governing the way in which a bit selection mask is to be obtained, what the test condition is to be, and what modification is to be made on the selected bits.

2.2.12 Program Control

Program control instructions include several types of jump instructions and the subroutine control PUSHJ and POPJ instructions.

Pushdown stacks are handled by the PUSH and POP instructions, which, through a stack pointer, process data on a first-in, last-out basis. Subroutine entry and return is accomplished by jump instructions (PUSHJ and POPJ) that insert return addresses on a pushdown stack. These instructions are vital to the efficient operation of the timeshared monitor and all of the reentrant system's programs.

2.2.13 Input/Output

Input/output over the EBus is handled by eight straightforward instructions. Each instruction may reference one of 126 devices. In addition to reading status, writing status, reading data, and writing data, there are block-in and block-out instructions to handle blocks of data to and from memory and to a device in an efficient manner.

2.2.14 Unimplemented User Operations (UUOs)

Many of the codes not assigned as specific instructions are executed as unimplemented user operations, where the word given as an instruction is trapped and must be interpreted by a routine included for this purpose by the programmer. Those UUOs reserved for use by the monitor are called monitor UUOs (MUUOs), while user UUOs are called local UUOs (LUUOs). Instructions that are illegal in user mode also trap in the same manner as MUUOs.

2.2.15 Trap Handling

The DECSYSTEM-20 provides facilities for handling arithmetic overflow and underflow conditions, pushdown list overflow conditions, and page failures directly by the execution of programmed trap instructions. This trap capability avoids recourse to the program interrupt system. A trap instruction is executed in the same address space as the instruction that caused the trap.

2.3 INSTRUCTION FORMAT

In all the non-input/output instructions, the nine high-order bits (0–8) specify the operation, and bits 9–12 usually address an accumulator but are sometimes used for special control purposes such as addressing flags (Figure 2-3). The rest of the instruction word always supplies information for calculating the effective address, which is used for immediate mode data or is the actual address used to fetch the operand or alter program flow. Bit 13 specifies the type of addressing (direct or indirect), bits 14–17 specify an index register for use in address modification (zero indicates no indexing), and the remaining eighteen bits (18–35) contain a memory address.

The instruction codes that are not assigned as specific instructions are performed by the processor as so-called "unimplemented operations."

An input/output instruction is designated by three 1s in bits 0–2. Bits 3–9 address the input/output device to be used in executing the instruction, and bits 10–12 specify the operation. The rest of the word is the same as in non-input/output instructions.

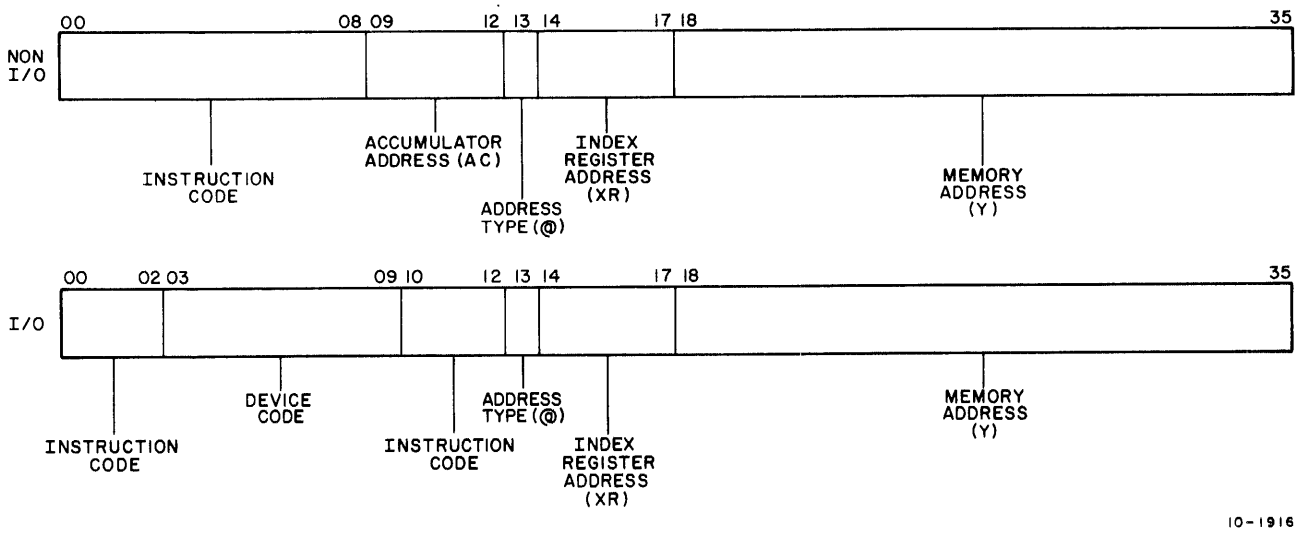


Figure 2-3 Instruction Format

2.4 NUMBER SYSTEM

The standard arithmetic instructions in the DECsystem-10/DECSYSTEM-20 use two's complement, fixed-point conventions to do binary arithmetic. In a word used as a number, bit 0 (the leftmost bit) represents the sign; 0 for positive, 1 for negative. In a positive number, the rest of the 35 bits represent the magnitude in typical binary notation. The negative of a number is found by taking its two's complement. Zero is represented by a word containing all 0s.

2.4.1 Fixed-Point Arithmetic Conventions

Two common conventions are to consider a number as an integer (binary point at the right) or as a proper fraction (binary point at the left); in these two cases, the range of numbers represented by a single word is -2^{35} to $2^{35}-1$ or -1 to $1-2^{-35}$. Since multiplication and division make use of double-length numbers, there are special instructions for performing these operations to get results that can be represented by a single word.

The format for double-length, fixed-point numbers is an extension of the single-length format. The magnitude (or its two's complement) is the 70-bit string in bits 1–35 of the high- and low-order words. Bit 0 of the high-order word is the sign, and bit 0 of the low-order word is ignored. The range for double-length integer and proper fractions is, therefore, -2^{70} to $2^{70} - 1$ or -1 to $1 - 2^{-70}$.

2.4.2 Floating-Point Arithmetic Conventions

The DECsystem-10/DECSYSTEM-20 has firmware for processing both single- and double-precision, floating-point numbers.

Included in the arithmetic instruction set are eight double-precision instructions and three fixed/floating conversion instructions. A double-precision word consists of the sign, an 8-bit exponent, and a 62-bit fraction. This gives a precision in the fraction of 1 part in 4.6×10^{18} and an exponent of 2 to a power of from -128 to $+127$.

The same format is used for a single-precision number and the high-order word of a double-precision number. A single-precision, floating-point instruction interprets bit 0 as the sign, but interprets the rest of the word as an 8-bit exponent and a 27-bit fraction. Normalized single-precision, floating-point numbers have a fraction that ranges in magnitude from $1/2$ to $1 - 2^{-27}$. Increasing the length of a number to two words does not change the range but instead increases the precision; in any format, the magnitude range of the normalized fraction is from $1/2$ to 1, decreased by the value of the least significant bit. In all formats, the exponent range is from -128 to $+127$.

2.5 EFFECTIVE ADDRESS CALCULATION

All instructions in the DECsystem-10/DECSYSTEM-20, without exception, calculate an effective address using bits 13–35 in exactly the same way. The steps are as follows.

1. Get the number in address field Y, bits 18–35. Any one of 262,144 locations can be specified.

2. If index field X, bits 14–17, is not zero, add the contents of the specified index register to the number found in step 1.

3. Get the indirect bit, I, bit 13. If it is 0, the calculation is done and the result of steps 1 and 2 is the effective address. If it is 1, then go to step 4.

4. Use the address calculated by steps 1 and 2 to get a new word from memory, and go back to step 1.

The effective address calculation continues until a word is found with a 0 in bit 13. At that point, the result of steps 1 and 2 is taken as the effective address for the instruction.

The calculation is done for all instructions, including those specifying immediate mode. As an example, it is possible in one immediate mode instruction to load an accumulator with the address of a particular entry within an indexed table for use as a subroutine argument.

2.6 GENERAL-PURPOSE REGISTER BLOCKS

General-purpose registers are another DECsystem-10/DECSYSTEM-20 feature that help improve program execution. These sets of fast integrated circuit registers can be used as accumulators, as index registers, and as the first 16 locations in memory. Since the registers can be addressed as memory locations, they do not require special handling instructions.

Eight sets of 16 fast registers are included. Program switching time between register stacks is 500 nanoseconds.

Different register blocks can be used for the operating system and individual users. This eliminates the need for storing register contents when switching from user mode to executive mode. Also, a critical real-time program is able to maintain its own register block for handling data and interrupt sequences at maximum speed.

2.7 MEMORY SYSTEM

2.7.1 MOS Memory

The DECsystem-10/DECSYSTEM-20 memory system can provide storage for up to 3,145,728 44-bit words (36 data bits, 6 ECC bits, 1 ECC parity bit, and 1 spare bit) in increments of 256 K (262,144) words, 1.5 megawords internal, 1.5 megawords external.

2.7.2 DMA

The DMA memory bus adapter adapts the KL10 storage bus (SBus) to the external memory bus structure. The DMA also separates the SBus into four separate buses (KBus 0–3) to allow 4-word transfers and 4-way interleaving of storage modules.

2.7.3 Cache Memory

A DECsystem-10/DECSYSTEM-20 can also be equipped with a 150 nanosecond access time data cache or buffer memory. Data being read from memory is usually found already in the cache 90 percent to 95 percent of the time, therefore giving the DECsystem-10/DECSYSTEM-20 an effective memory access time of approximately 300 nanoseconds. Another feature of the state-of-the-art design cache memory is that unlike contemporary designs, it does not require write-through to memory. Instead, words to be written are written into the cache memory. This eliminates, for example, the necessity of writing back into main memory each value of an index in a loop made up of only a few instructions.

The cache is paged and words from one or more pages are written back to main memory from the cache only when it is necessary to make room for words from new pages. A cache sweep feature allows main memory to be updated with all or selected pages of the cache.

2.8 PROCESSOR MODES

Instructions are executed in one of two modes depending on the state of a mode bit. Programs operate in either user mode or executive mode. In executive mode operations, all implemented instructions are legal. The monitor operates in executive mode and is able to control all system resources and the state of the processor. In user mode operations, certain instructions such as direct I/O are illegal, causing a trap to the monitor. Users are required to issue monitor calls for system services such as I/O.

Executive and user mode operations are each divided into two submodes. User mode is divided into public and concealed submodes and executive mode into supervisor and kernel submodes. For each 512 word page in the system, information is stored in a table (page map) maintained by the operating system that specifies whether or not a page can be accessed or changed and if it is defined to be public or concealed. The executive and user modes divide according to whether the active program is running in a public or concealed area.

If a program is running in public submode, pages within the user's addressing space are accessible only if they are listed in the user's page map and are defined to be accessible from public mode. Pages defined as public are, by definition, accessible. Pages defined as concealed may be accessed only at specific entry points; that is, portals that permit entry from public submode programs. In concealed submode operations, programs can access all of the virtual addressing space. However, if a program running in concealed submode executes an instruction from an area defined as public, the state of the processor transfers over into public submode. Typical user programs operate in public submode. Concealed areas can be used for proprietary coding that can be executed but not changed or examined by users operating in public mode.

The supervisor and kernel submodes are similar but not identical to the public and concealed submodes. Supervisor submode programs can access but cannot modify areas defined as concealed. Also, any instruction executed out of a public area from either supervisor or kernel submode returns the processor to supervisor submode. In kernel submode operations, all of memory is accessible and can be modified. Programs operating in kernel submode can address portions of memory directly, without paging; and it is through the kernel submode program that page restrictions are made. Functions that go to supervisor submode generally include those affecting individual users as opposed to the whole system management of input/output, priority interrupts, page map accounting, and so on, which are handled by kernel submode programs. The ability of kernel submode programs to supply information that supervisor submode programs can read but not change, allows portions of the operating system to be hardware-protected from other portions having adjustments or design changes.

2.9 PROCESS TABLES

There are two types of process tables in memory that are used by the hardware for both system and user management. These are the Exec Process Table (EPT) and the User Process Table (UPT). Each is one page long (Figures 2-4 and 2-5). One EPT is used for the monitor and one UPT for each user process in the system. The following is a partial list of the types of information that are maintained in these tables.

1. User Process Table

    a. Arithmetic overflow vector address. Overflow affects only the user and not the system so it is handled in the user address space.

    b. Memory and instruction processor convention accounting clocks. This information is kept for each user to help in exact user resource accounting.

2. Exec Process Table

    a. Channel recording area. Information about integrated channel status, which affects the entire system, is maintained here.

    b. Front-end processor communications area. Used in communications between the central processor and the front-end processor.

While the monitor is providing services for a user, such as I/O requests, it must be able to reference the user address space. There are two mechanisms provided to do this.

1. A special instruction called PXCT, which has bit settings to indicate which of the specified addresses are in the current address space and which are in the previous address space.

2. The Exec Page Table contains "indirect pointers" through the User Page Table. These are used to provide a per process area for each process in the system.

2.10 MEMORY ADDRESS MAPPING

The memory address mapping hardware has been developed in conjunction with the software so that memory management is transparent to the user. Physical memory is divided into 512 word segments called pages. All addresses — both monitor and user — are translated from the program's address space (referred to as the virtual address) to the physical memory address space. This facilitates protection of the monitor and allows efficient use of physical memory. For example, only a portion of the monitor need be permanently memory-resident, resulting in more available user memory.

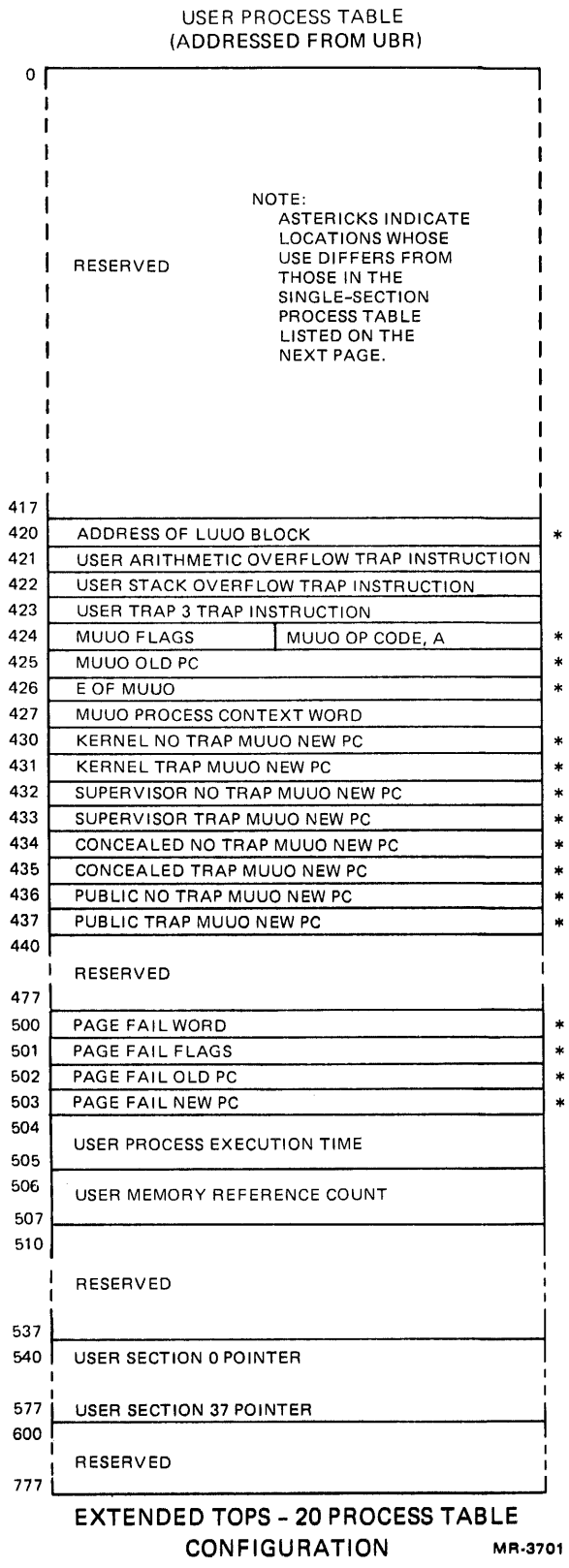


Figure 2-4 User Process Table

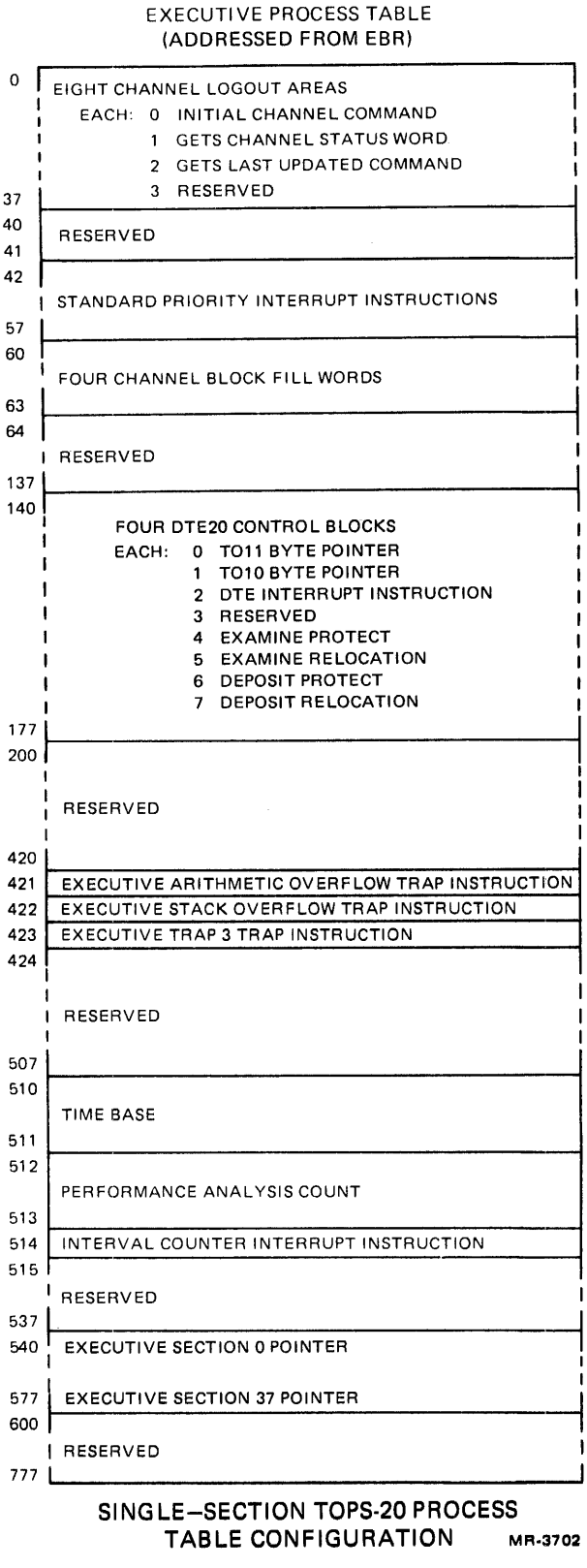


Figure 2-5 Exec Process Table

The high-order nine bits of an 18-bit virtual address make up the virtual page number and are used to index into a hardware page map. If the 13-bit physical page number is found in the hardware page map, then the low-order nine bits of the virtual address are appended to the 13-bit physical page number to form the complete physical address. If the entry in the hardware page map is not there, the memory processor gets the entry from the individual page map in memory and updates the hardware page map.

There is a page map in memory for the monitor and one for each user. These maps contain storage address pointers that identify either a page in memory or on disk. They may be of three basic types.

1. Private – The page is owned by one user.

2. Shared – The page is shared by more than one user.

3. Indirect – Points to another page map page where the pointer may again be one of the three types.

In addition to the physical page numbers in the page map, control bits are present to serve various functions. For example, one bit is used to indicate that the page is read-only; another to indicate that no physical page at that time agrees to this virtual address slot.

Page level sharing among users is supported by a hardware Shared Page Table, which holds the physical address of the shared page. This table, which is addressed through a hardware register, enables the software memory management routines to maintain only one pointer to a page no matter how many processes are sharing it.

Information about how long a page has been in memory and the number of processes sharing it is also stored by the hardware in a Core Status Table to help the monitor in core management.

2.11 DIRECT I/O

The EBus is used as a control and data path to/from a large number of low-speed I/O devices. Transfers are performed for 36-bit words at speeds of 370 K words/s. Therefore, each I/O instruction moves one word of data between memory and the buffer of the device controller (DTE20 or RH20). When block input or output instructions are used, entire blocks of data are moved to or from the device with a single instruction.

To initiate high-speed data channel transfers directly between memory and a device connected to the Massbus, a control word is first transferred over the EBus to the command register of a Massbus controller (RH20). Then, entire data blocks are moved between a drive and memory under the control of a channel command list.

2.12 CHANNEL I/O

2.12.1 Integrated Massbus Controller

The integrated Massbus controllers are high-speed mass storage controllers that interface the RP06, RP07, and RP20 disk drives and the TU72, TU77, and TU78 magnetic tape drives to the integrated data channels in the memory controller. The controllers have been designed to provide high throughput by features such as the following.

1. All controllers can transfer data simultaneously since each controller is connected to the memory system with its own channel.

2. While one device on a controller is transferring data, control operations such as seek or rewind

may be issued to another device on the same controller. The operation can be initiated and an interrupt generated when it is complete.

3. Each controller has a lookahead command register, which enables the software to preload the next transfer request during the current transfer. Therefore, the next transfer can start with the next sector on the same device with no rotational delay.

4. When the controller has completed an operation it interrupts the central processor via a vectored interrupt so the central processor does not have to poll a series of devices to determine which device caused the interrupt.

Error checking is provided for both the channel and device data paths. The controller will terminate a command if certain errors are detected.

The connection between the controller and its devices is called the Massbus and contains parallel data and control paths. These parallel paths permit simultaneous data transfer and control operations.

2.12.2 Integrated Channels

Within the storage controller there is one channel control for each device controller. Each channel control has a 15 word data buffer, a channel command word register, and a control word location pointer; or each has a program counter. A channel consists of a channel control, a device controller, and its associated drives.

The channels perform data transfer by executing channel programs, which are loaded into memory by the device service routines. The channel loads a channel command word into the command register and then executes it. The direction of transfer is specified in the instruction sent to each controller.

Channel instructions contain a memory address, a word count, and function control bits to facilitate efficient I/O programming. Each time a word is transferred, the word count is decremented.

If the initial address is 0 and the operation is a read, the number of words indicated by the word count is skipped. If the operation is a write, the channel transmits software-specified block-fill words for the required count.

If the initial word count is 0, the channel will jump to the address indicated in the instruction, where a new command word will be selected and decoded.

System I/O programs can select between two ways of halting, depending on which is more efficient for the device. They may use a channel program chaining technique in which a halt instruction, which contains the address of the next channel program, is the last command word. Alternatively, if the last data transfer command word has the halt control bit set, the channel will halt when the word count is decremented to 0. This saves time because it saves a halt command memory fetch.

If a channel detects an error, it stores two channel status words in the appropriate controller location in the Exec Process Table. The first status word contains bits to indicate what type of error occurred and the control word location pointer. The second status word contains the command word that was being executed with the up-to-date word count and address.

2.13 PRIORITY INTERRUPT SYSTEM

The DECsystem-10/DECSYSTEM-20 interrupt facility does not require devices to be hardwired to a particular level. Instead, devices are assigned under program control to any one of seven priority levels through the dynamic loading of device registers. Therefore, the monitor can change the priority level of any device or disconnect the device from the system and later reinstate it at another level. This is an

advantage over permanently hardwired systems, which require a large number of levels, often operate at very high overhead, and cannot change device priorities without system shutdown and rewiring.

Priority level 0 is above the seven programmable levels and is reserved for the front-end processor, which is therefore able to interrupt the system at any time for console or diagnostic operations.

2.14 TRAP FACILITY

The system also provides a trapping mechanism to handle certain conditions that affect a single job. Conditions that are detected by the trapping hardware include:

- Address violation
- Arithmetic overflow
- Pushdown overflow
- Illegal instruction
- Monitor calls
- Page faults.

2.15 ACCOUNTING AND PERFORMANCE METERS

The meters built into the DECsystem-10/DECSYSTEM-20 provide a number of timing and counting functions, as follows.

1. Interval timer – A programmable source of interrupts with a range from 10 microseconds to 40.96 milliseconds.

2. Time base – A one microsecond relative time-of-day clock that is used by the monitor for system accounting.

3. Performance analysis counter – Designed as a tool for testing and evaluating the system, this counter monitors either duration or rate of occurrence of various hardware conditions and events.

4. Two accounting meters – The first is an instruction processor meter, which measures the amount of the instruction processor time used. This information is stored in the User Process Table. While the default count is user time only, the monitor may also include interrupt time and/or exec mode time in this accounting. The second is a memory reference meter, which measures user program accesses to core memory. This information is also stored in the User Process Table.

2.16 FRONT-END PROCESSOR

2.16.1 Functions

One of the significant features of the DECsystem-10/DECSYSTEM-20 architecture is the PDP-11 front-end processor. It has the following uses.

1. Control of low-speed peripherals such as card readers, line printers, and asynchronous communications devices

2. Console operations for the central processor including push-button functions such as load, start, and stop

3. On-line and remote diagnostic analysis

4. Microcode loading and system startup

The front-end processor communicates with the central processor through an interface that permits concurrent two-way data transfers.

As a diagnostic computer, the front-end processor can examine the data paths and control logic of the central processor even if that unit is completely unable to operate. A diagnostic bus linkage permits automated testing procedures that allow a larger number of diagnostic tests to be run than would be possible with conventional techniques. Accuracy is increased since not only is there less chance for human error, but maintenance engineers do not have to rely upon a malfunctioning central processor to help diagnose itself. Other capabilities include remote and timesharing diagnosis.

If system power fails, a detection circuit senses the condition and causes an interrupt. The interrupt can trigger the operation of a program that saves changing registers and provides an orderly system shutdown so that the system can be restarted with a minimum amount of lost time.

All three phases of ac power are monitored. Low voltage on any phase will initiate the power-down sequence. A program-selectable automatic restart capability is provided to allow resumption of operation when power returns. Alternatively, a manual restart may be used.

Temperature sensors strategically placed within the equipment detect high temperature conditions and cause power shutdown. This, in turn, initiates the power failure interrupt.

The front end boots the DECsystem-10/DECSYSTEM-20 by reading in an initial program from the front-end disk (via the RH11) attached to the Unibus. This in turn brings the main bootstrap program from the dual-ported disk, which boots the system into operation.

2.16.2 Components

2.16.2.1 DTE20 – The DTE20 console processor interface connects the central processor and the front-end processor to provide capabilities to interrupt, examine, deposit, and transfer data during timesharing and diagnostic operations.

2.16.2.2 Console – The console provides the operator with a direct system interface by simulating the control switch and display indicator functions for the central processor. Therefore, the operator is provided with control of normal program and diagnostic operations, which allows him to start, stop, load, modify, or continue a program. The console is connected for simultaneous two-way transmission to the PDP-11 Unibus, through an asynchronous line interface.

2.16.2.3 PDP-11 – The heart of the PDP-11 system is the Unibus, which is an asynchronous bidirectional bus interconnecting the front-end processor components. The front-end processor is connected to the Unibus as a subsystem and controls time allocation of the Unibus for peripheral devices. In addition, the front-end processor performs arithmetic and logic operations through instruction decoding and execution. The instruction set is implemented through a group of hardware subroutines stored within a read-only memory (ROM). Memory is read/write, random-access magnetic core with a maximum cycle time of 980 nanoseconds and a maximum access time of 425 nanoseconds. Storage capacity is 28 K words having an 18-bit word length (16 data bits plus two parity bits).

CHAPTER 3
THE HARDWARE

3.1 INTRODUCTION

The DECsystem-10/DECSYSTEM-20 includes the following subsystems.

1. A KL10-based main processor subsystem with up to 1.5 megawords of internal MOS memory and 1.5 megawords of external MOS memory (except with the 1091)

2. A PDP-11-based front-end processor subsystem

3. A mass storage subsystem

3.2 THE MAIN PROCESSOR SUBSYSTEM

The main processor subsystem of the DECsystem-10/DECSYSTEM-20 contains the following units (Figure 3-1).

1. EBox
2. MBox
3. Meter board
4. MF20 MOS memory
5. DTE20 ten-eleven data interface (up to four)
6. RH20 Massbus controller (two at least, but there can be up to eight)

The EBox, MBox, and meter board are designed with high-speed, nonsaturating emitter-coupled logic (ECL) and are housed on the same assembly. These functional units make up the central processing unit (CPU). The DTE20 and RH20 are designed with TTL logic and serve as the interface controllers to the front-end processor and Massbus-compatible storage devices, respectively. An internal X/SBus serves as the control and data path between the CPU and its memory. An internal EBus serves as the control and data path between the CPU and the controllers (DTE20 and RH20). Since both ECL and TTL logic are used in the main processor, logic level translators are included in the CPU assembly to convert from one logic level to the other.

3.2.1 The EBox

3.2.1.1 Hardware – The EBox is the instruction execution unit of the central processor. Basically, the instruction execution unit consists of the following logic.

1. Memory request logic
2. 72-bit arithmetic logic
3. 23-bit address logic
4. 10-bit arithmetic logic
5. Eight general register blocks

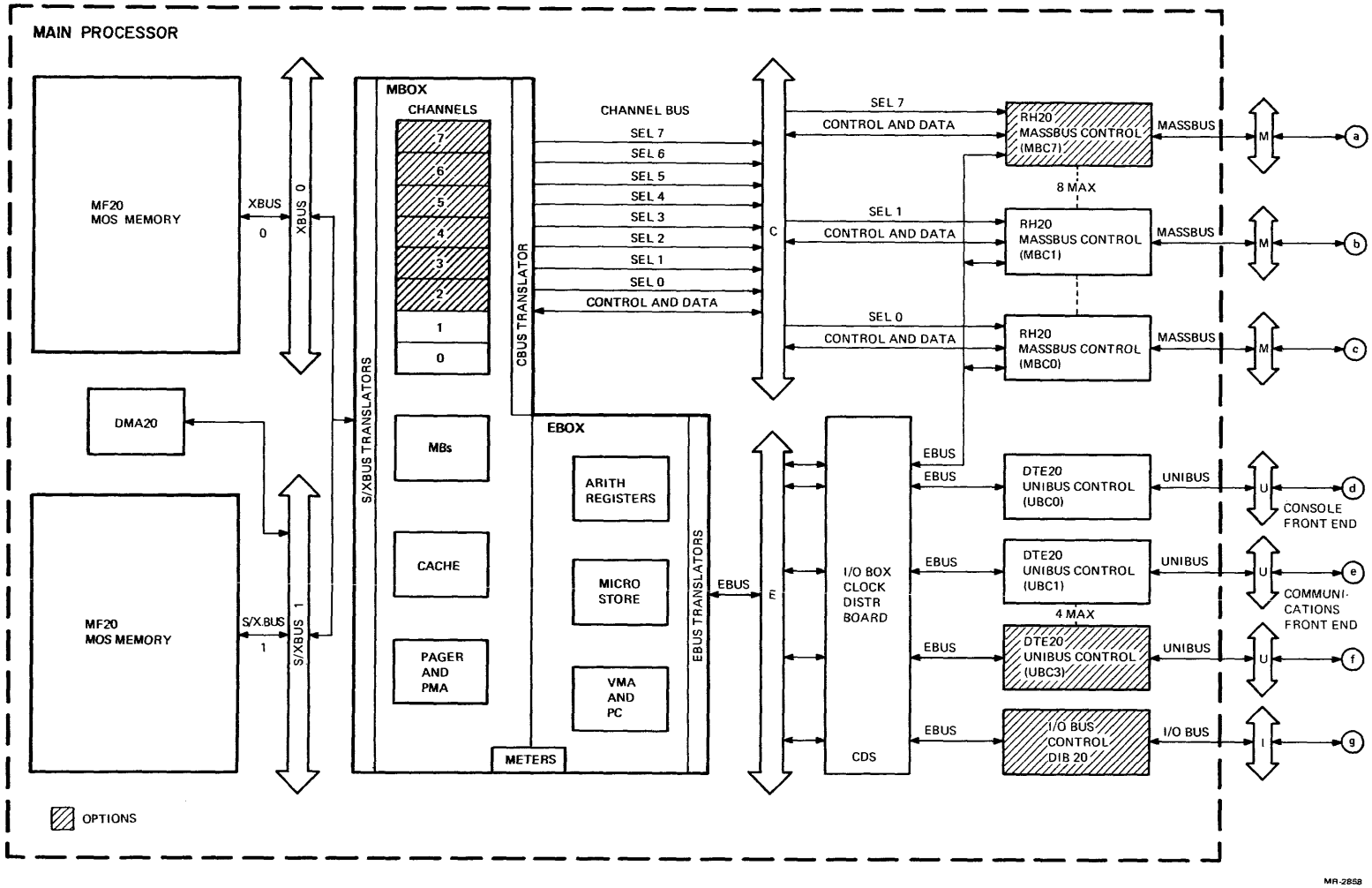


Figure 3-1 DECsystem-10/DECSYSTEM-20 Typical Block Diagram (Sheet 1 of 2)

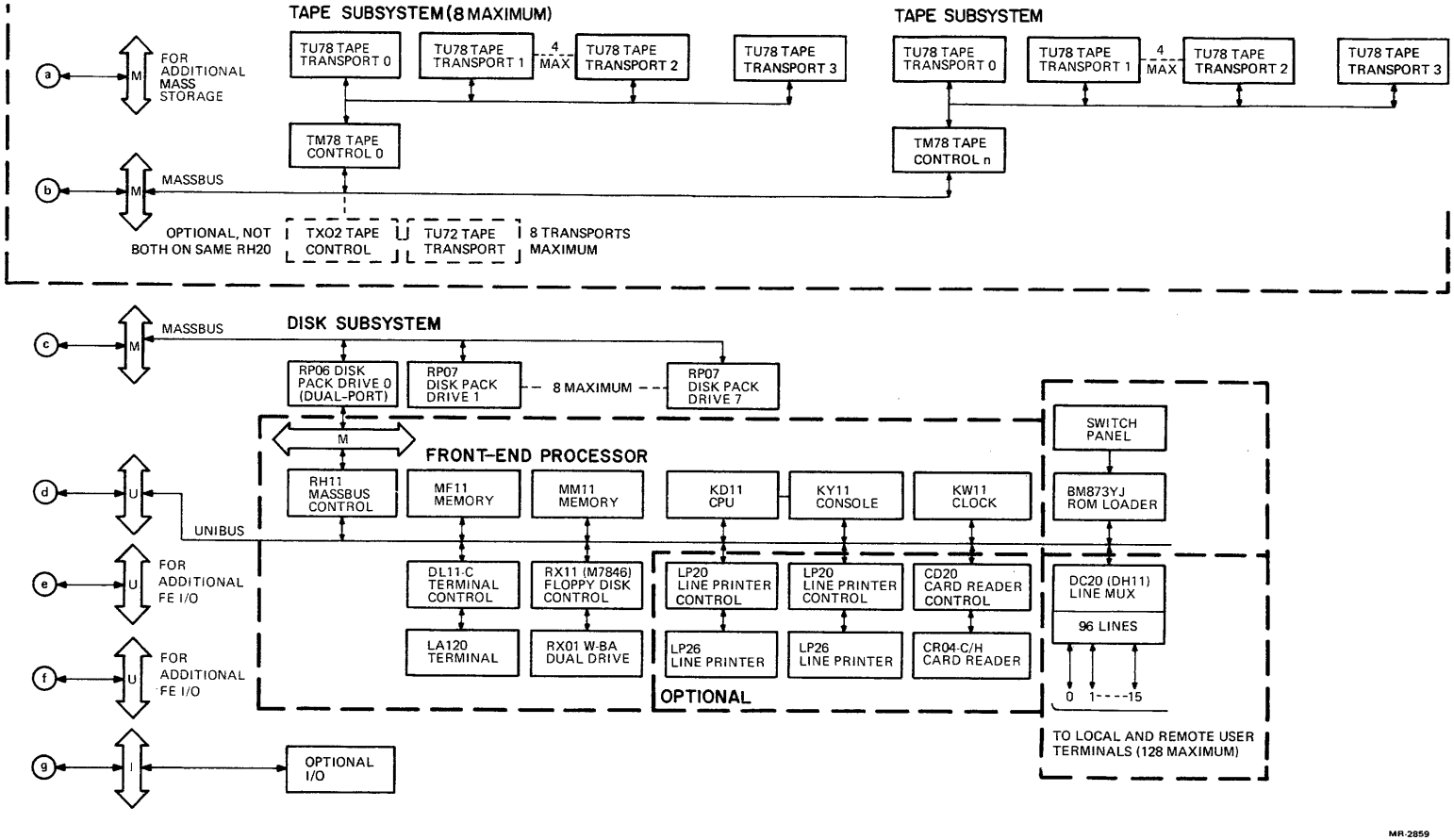


Figure 3-1 DECsystem-10/DECSYSTEM-20 Typical Block Diagram (Sheet 2 of 2)

6. EBus control logic
7. Microprogrammed instruction dispatch and control store

In addition, the EBox contains the master clock, meters, a processor status register, and the diagnostic control logic.

All operations in the DECsystem-10/DECSYSTEM-20 are synchronized to the master clock, which runs at 50 MHz. The master clock can be started, stopped, single-stepped, and otherwise controlled by the front-end processor via the diagnostic control logic. This logic is distributed between the EBox and the DTE. Besides being able to control the master clock, the diagnostic control logic provides a means for monitoring the processor status and diagnostic registers in both the EBox and the MBox. The master clock supplies a 30 MHz clock to the MBox, a 12.5 MHz clock for the EBox control store, and a 6.25 MHz clock to the EBus and SBus.

The program counter (PC), virtual memory address adder (VMA AD), virtual memory address register (VMA), and arithmetic adder (AD) form the basic address manipulation path in the EBox. This path is 23 bits wide to accommodate a virtual address space of 8 million words. (Only 18 bits of the virtual address are implemented for DECsystem-10/DECSYSTEM-20 to provide a virtual address space of 256 thousand words.) During the course of calculating the effective address for an instruction, AD will contain the value of Y, Y + XR, Y@, or (Y + XR)@ and pass it to the VMA. The source for Y, XR, and @ are specified by the corresponding fields of the instruction. The VMA can also receive PC + 1 and PC + 2 via the VMA AD. This is normally done for main line instruction fetches or for skip-type instructions.

The arithmetic register (AR) and arithmetic register extension (ARX), the buffer register (BR) and buffer register extension (BRX), and the adder (AD) and adder extension (ADX) form a 72-bit data path for manipulating data. This data path is implemented so that half words, full words, and double words can be manipulated with ease. The AD and ADX are implemented with arithmetic logic units (ALU), which are capable of performing 16 arithmetic and 16 logical operations. Words fetched from memory may be moved into the AR, ARX, and IR. Usually data (operands) is placed in the AR and ARX, while instructions are moved to the ARX and IR. Words to be moved to memory are placed in the AR when a memory request is made. Data transfers to and from the EBus are made via the AD and AR, respectively.

The shift count adder (SCAD), floating exponent register (FE), and shift count register (SC) form the 10-bit arithmetic logic, which is used in performing shift operations and operations on byte pointers and floating-point exponents.

The shifter (SH) is used in performing shift, rotate, and byte pointer operations. The shifter is also used in aligning a particular field of a word (such as the API function word) for dispatching into the control store as a function of the contents of that field.

The multiplier quotient register (MQ) is primarily used in performing floating-point and double-precision integer arithmetic operations. The MQ is also used as a temporary storage register for saving the API function word.

Each of the eight general register blocks (AC blocks 0–7) consists of a set of 16 general-purpose, high-speed registers. Block 0 is permanently assigned to the monitor and blocks 6 and 7 to the microcode. The monitor uses its AC block in the same way as the user program described in the following paragraphs. The microcode uses the assigned AC block when executing complex instruction algorithms. Of the remaining blocks, two can be assigned under program control (DATAO PAG) to the user as the current and previous context AC blocks. The current context AC block is used by the user program for indexing, for general storage as specified by the AC field of the instruction and/or by the effective virtual address (location 0–17), and for instruction execution if desired.

The previous context AC block is used by the monitor to allow the monitor to reference the previous user's address space to pass arguments, data, or status information between the previous user's program and the monitor. This is normally done when the user program executes a monitor call for some type of service.

The instruction register (IR), dispatch RAM (DRAM), and control RAM (CRAM) form the instruction dispatch control and execution logic. Essentially, the IR holds the instruction's operation code (000–777), which is used to address the DRAM. The DRAM contains a dispatch address into the CRAM, and the CRAM contains the microcode for executing the instruction. Along with the dispatch address, the DRAM also contains control bits for initiating an operand fetch, instruction prefetch, and the store operation. The dispatch address and associated control bits in the DRAM vary in accordance with the requirements of the dispatching instruction. The contents of both the DRAM and the CRAM are the assembled and formatted object code of the KL10 microprogram.

When an instruction is fetched from memory, it is normally placed into the ARX and the IR. The microcode detects that an instruction has been received and if no traps, interrupts, or errors are sensed, will start calculating the effective address. If indexing is specified by the instruction, the microprogram will access the assigned general register block at the location addressed by ARX bits 14–177 (the XR field). The initial address portion of the instruction word in ARX is the Y field, consisting of bits 18–35 of ARX. This is added to the contents of the addressed general register, and the result will enter VMA and AR. The effective address calculation continues; if the instruction specifies indirect addressing [ARX bit 13 (1)], a memory cycle is required. The fast memory is addressed by VMA bits 32–35 if VMA bits 18–31 = 0. If this is the case, then the indirect reference will access fast memory. As before, the word will pass via the adder into AR and VMA. If the indirect reference is not to fast memory, the microprogram generates a request via the memory request logic, which performs all the handshaking. When the word is available, it is passed via the MBox cache data lines and enters the AR and ARX. The DECsystem-10/DECSYSTEM-20 is capable of multilevel indirect addressing; indexing may be specified at each level. The process continues until the indirect bit in a word entering ARX is zero. At this point only one level of indexing is possible and having completed this operation the VMA and AR will contain the effective address (E).

After computing the effective address, the microprogram uses the op code of the instruction in IR to address the dispatch RAM. The word fetched from the dispatch RAM is loaded into a register (dispatch register) where it will be available while the instruction is processed. The dispatch register word contains the equivalent of a fetch field, a store field, and an address field. This equivalent points to the location in the microprogram where the execution portion of the cycle for the instruction starts. The fetch and store fields are sampled by the microprogram at the appropriate time to initiate fetch and store operations where required. The microprogram consists of many microinstructions, each of which is composed of discrete fields. Some of these fields control the data path; others control the microprogram branching mechanism; still others control the clock, and so on.

If an operand is to be fetched or if a write operation is to be performed, the address is page-checked after the effective address has been calculated. After issuing the appropriate request, the microprogram diverts to a point where it will wait for the operand to be loaded into AR and ARX if an operand was to be fetched, or it will wait for the page check to be verified.

When this phase is completed, the microprogram uses the address portion of the dispatch word to enter the microprogram at a point that will start the execution of the particular instruction.

Once the execution part of the cycle is entered, the branching mechanism is controlled dynamically by conditions given by the specific instruction. Some instructions cause a prefetch of the next instruction in the sequence; others do not; and some instructions, such as jumps, cause instruction fetches by their very nature. The last part of the microprogram cycle implements the storage of those operands developed during the execute part of the cycle. Remember that a write paging check was made previously and it is only necessary at this point to pass the data to the MBox via the E/M interface. The writing is done via the

AR register together with the correct request qualifier signals. This having been done, the microprogram branches back to a point where it will start the effective address calculation for the next instruction.

In general, an EBox request for memory requires that the EBox set up the correct effective virtual address in the VMA, set up the data path for accepting or providing the word, and set up the appropriate request qualifiers. Most memory requests are initiated by a 4-bit field (MEM field) in the microinstruction. This field may be used alone or in conjunction with the DRAM fetch or store fields. The contents of these fields control the operation of the memory request logic to initiate and execute the following types of request.

1. Fetch instruction
2. Fetch indirect address
3. Fetch operands
4. Store results

Besides these basic memory operations, the memory request logic can also initiate a request to write-check a page, map the virtual address, load and read internal MBox registers, and so on.

The EBus control logic consists of two sections. One section handles programmed I/O operations; the other handles priority interrupt (PI) I/O operations. To facilitate the transfer of control and data between the EBox and a specific controller, each controller on the EBus is permanently assigned a device code and a physical number. In addition, each controller can also be assigned to a priority channel under program control.

The section of the EBus control logic that handles programmed I/O operations is controlled by a field (SKIP/COND-EBUS CTRL) in the microinstruction to which the instruction dispatched. Specific patterns in this field, in conjunction with another field in the microinstruction, cause the EBus sequence for transferring control, status, or data between the EBox and the desired controller (or internal device) to be executed. Bits 03–09 of the IR (I/O instruction device code field) are used to select the controller, and IR bits 10–12 (I/O instruction function code) specify the type of I/O operation (CONO, CONI, DATAO, DATAI, and so on) to be executed. If the fetched I/O instruction specifies output operation to the device (CONO, DATAO, BLKO), the EBox issues a memory request to fetch the operand before executing the EBus dialog. If, however, the instruction specifies an input operation from the device (CONI, DATAI, CONSO, CONSZ, BLKI), the EBus dialog is executed to fetch the word first. After the word is received by the EBox, a store operation is initiated by the EBox.

The section of the EBus control logic that handles priority interrupt I/O operations (PI control) runs concurrently with the instruction execution logic to fetch the API function word in response to a PI request (P10–7) from the controller on the EBus. After the API function word is received, an INT request is issued by the PI control to dispatch to the execution microcode as a function of the API word. Included in the interrupt control are several control and status registers (PI) that can be loaded and read using the standard I/O instructions. These registers, therefore, make up an internal processor device. The purpose of these registers is to control and maintain the current status of the PI system.

Besides the PI internal device for controlling and maintaining the status of the PI system, the EBox also contains an arithmetic processor status register (APR) and a set of meter/timer registers (MTR and TIM). These registers are also internal processor devices. Standard I/O instructions are used to access these devices for setting up control function, for monitoring status, and for transferring data.

3.2.1.2 Firmware – The heart of the EBox is a high-speed 2 K word control random-access memory (CRAM). This memory is initialized to contain the microcode. The microcode is loaded into the EBox from the front-end disk subsystem or the RX11 floppy disk subsystem. These devices are used for booting, diagnosing, and dumping functions. They are not supported as system devices. The basic program modules of the microcode are shown in Figure 3-2. The modules are:

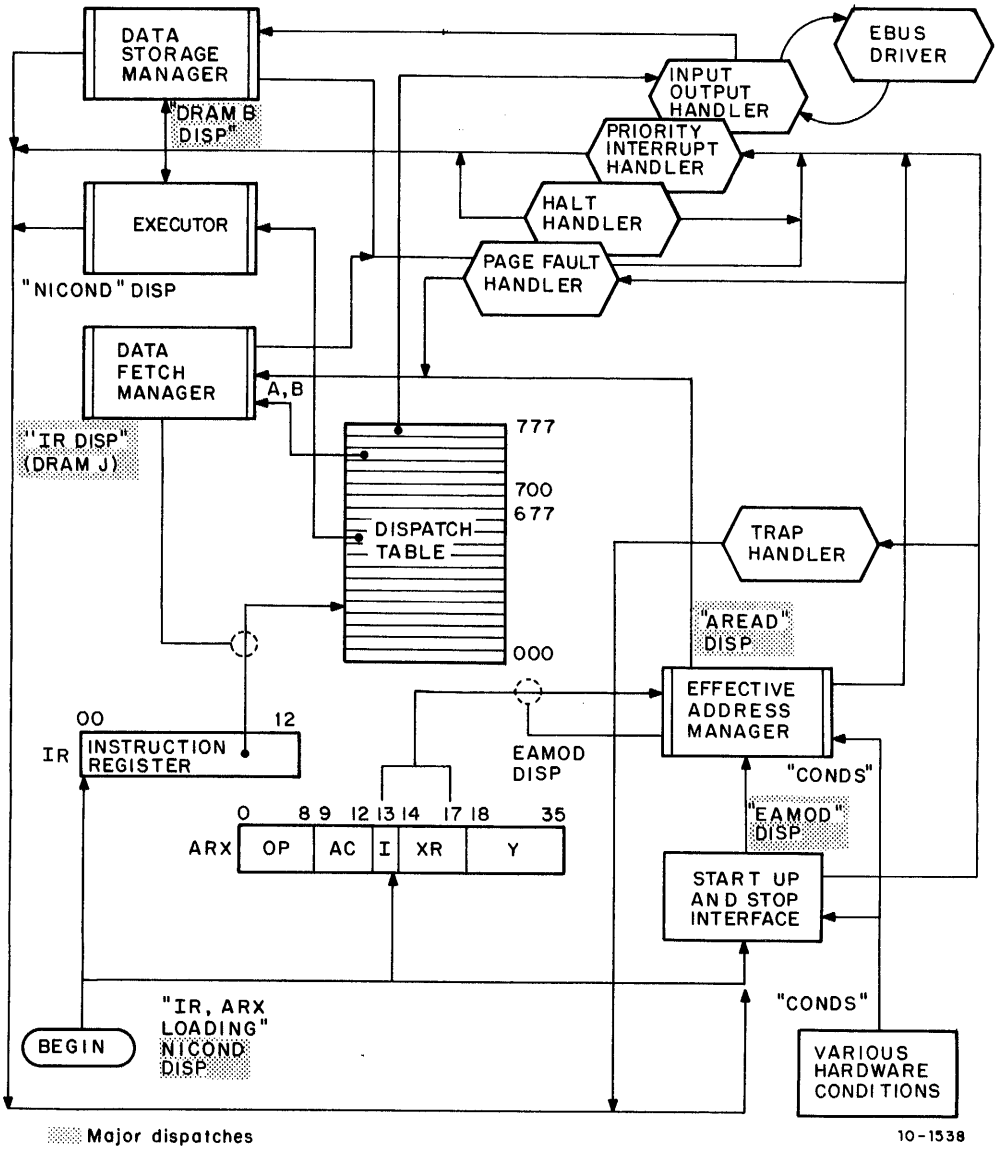


Figure 3-2 Microprogram Structure

1. Start-up and stop interface
2. Effective address manager
3. Data fetch manager
4. Executor
5. Data storage manager
6. Priority interrupt handler
7. Page fault handler
8. Halt handler
9. Input/output handler.

Each word of the microcode in the CRAM contains up to 84 bits of control information (Figure 3-3). Initial entry into the microcode for a given instruction is a function of the instruction op code via the

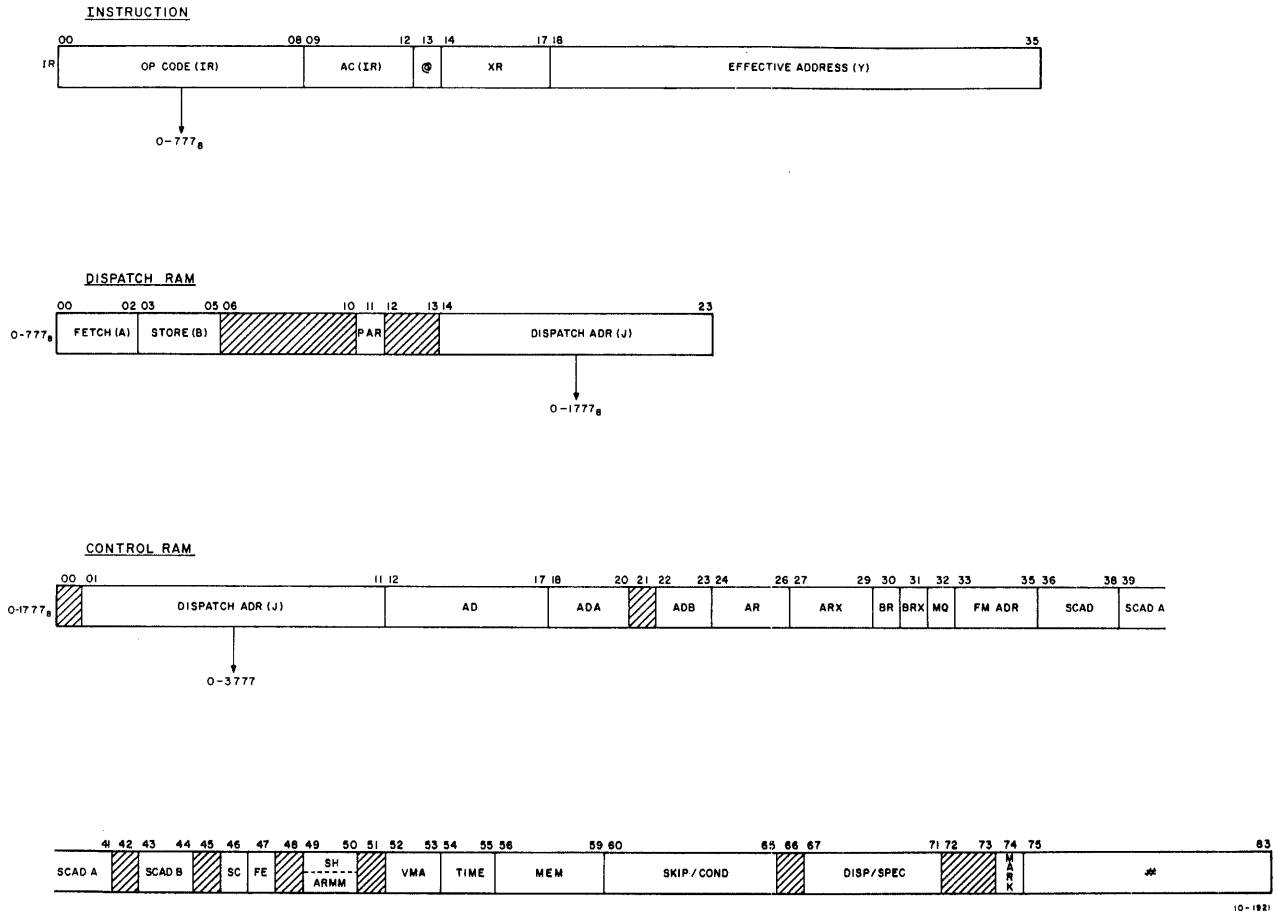


Figure 3-3 Instruction Dispatch and Control Formats

dispatch RAM (DRAM). Thereafter, the microcode sequences as a function of the J field of the microinstruction word.

The start-up and stop interface evaluates initial hardware conditions and dispatches to the appropriate handler. The nature of the condition could be a pending priority interrupt, a halt condition, and so on. Upon completion, all instructions must pass through this process.

The effective address manager evaluates the indirect address flag (bit 13) and the index field (bits 14–17) in the arithmetic register extension (which contains the current instruction) together with certain hardware conditions such as PIs or page failures. It either dispatches to the appropriate handler or calculates the effective address by requesting the necessary fast memory (index) cycles or MBox indirect (@) cycles.

The data fetch manager evaluates the 3-bit A (fetch) field (for the current instruction), which is in the dispatch table. The code in the 3-bit field defines the type of data fetch or write or combination operation (if any) required. The data fetch manager takes the proper action required; that is, it enables the EBox clock to stop as appropriate, dispatches directly to the executor, or initiates an instruction prefetch. Note the instruction register is used to address the proper location in the dispatch table (DRAM) based on the op code for the instruction.

The executor routine is the bulk of the microprogram. It contains a number of somewhat independent routines, which are used to execute the instruction-specific function; for example, move a half word from one register to another or push a word onto a subroutine stack and so on.

The data storage manager dispatches on the DRAM B field. In addition, when called from the executor as a subroutine only, (MEM/WRITE for example), it defines the appropriate MBox control signals and dialog and initiates the write operation. When the data storage manager is entered in the context of a store cycle, that control generally passes to the process from the executor. Then finally, control will pass to the start-up and stop interface.

The priority interrupt handler is dispatched to or from discrete points in the microprogram. Interrupts are looked at while computing the effective address and during certain longer instructions, such as BLT.

Control is passed to the page fault handler from the effective address manager or data storage manager when the MBox asserts PF HOLD and EBOX PF HANDLE prior to MBOX RESP during a memory request. The implication is that a memory address violation, such as an access failure, write protection violation, or some similar violation, occurred and that the paging address translation should be done by the microprogram. In addition, this handler is used for certain error conditions.

The halt handler routine is entered from the start-up and stop interface when the run flip-flop is found clear at the next instruction dispatch time. The run flip-flop can be cleared by various mechanisms. For example, when a halt instruction is executed, run is disabled. On power up, run must be set by a diagnostic function initiated from the DTE20.

The input/output handler is dispatched through IR dispatch from the dispatch table on DATAO, CONO after the data or status has already been fetched from memory, or directly on DATAI, CONI, CONSO, or CONSZ. The handler calls the EBus driver, which generates the necessary EBus dialog with the device. For BLKI or BLKO instructions, the pointer has been fetched but must be updated and stored back at E, and the required word must then be fetched. This is performed by the input/output handler first. When the data has been fetched, the EBus driver is called. On DATAI, CONI, the EBus driver is called to negotiate the transfer from the selected device over the EBus to the EBox. Then the input/output handler passes control to the data storage manager, which issues a request to store the data.

3.2.2 The MBox

The MBox is the storage controller of the DECsystem-10/DECSYSTEM-20. The MBox contains a pager, a physical memory address selector (PMA), and four memory buffer (MB) registers. These functional elements provide the EBox instruction execution unit access to physical memory. The physical memory address is created by the pager and the PMA, while the data path between main memory and the EBox is created via the MBs.

The MBox also contains an integral data channel I/O processor (a multiplexed channel controller). This I/O processor interfaces with the MBs to form a data path from the physical memory storage bus (SBus/XBus) to the channel bus (CBus). The CBus is multiplexed by the channel I/O processor to orderly select up to eight Massbus controllers (channels).

The pager is a high-speed, 512-word, set-associative automatic buffer memory where physical page addresses and page descriptor keys are stored. It serves as a high-speed extension of the page tables pointed to by entries in the User and Executive Process Tables. When the EBox issues a request for paged memory, the MBox automatically checks the contents of the pager to see if it contains a valid physical page address. If there is a valid address it simply concatenates the entry with the low order nine bits of the virtual address. This address is then used to issue a memory request. If the pager does not contain a valid physical page address, the MBox informs the EBox that a page refill operation is required.

When the EBox issues a memory request, the MBox fetches a single word from memory and transfers the word to the EBox. For write operations the MBox writes the word directly into memory.

The channel I/O processor is a multiplexed channel controller that can handle up to eight simultaneous high-speed block transfers without program intervention. After being started by a Massbus controller, the channel I/O processor executes the block transfer under the control of a channel command list, which is stored in physical memory. The channel I/O processor uses a set of RAMs for storing control and status bits, for maintaining the channel command list pointer and the channel command word, and for buffering the data.

For both CTOM (control to memory) and NOT CTOM operations, the channel controller will transfer blocks of four words to or from memory via the four MBs. The CBus transfers the data to or from the appropriate Massbus controller one word at a time.

3.2.3 Channel/Cache Interface Description

The channel/cache (CHAN/CSH) bus is an internal interface connecting the channel control logic and the cache/memory control logic portions of the MBox. This group of signals does not actually make up a bus; however, the signal-sequencing characteristics are similar to a bus. As such, this signal group is described as a bus. Table 3-1 describes each CHAN/CSH line. Figure 3-4 illustrates the CHAN/CSH bus configuration.

Table 3-1 CHAN/CSH Line Descriptions

Signal Line	Description
Control Commands	
Channel request (CCL3 CHAN REQ)	Asserted by the channels to request service.
Hold memory (CCL2 HOLD MEM)	Asserted by the channels if the channels have requests backed up. Asserting this signal assures the channel the next core cycle by preventing an EBox request from initiating a core cycle.
Channel cycle (CSH CHAN CYC)	Asserted when the cache cycle control starts processing the channel request. This signal informs the channel that it can start writing the memory buffers (MB) in the case of channel write operation or start looking for words ready to be taken from the MBs in the case of channel read operations.
Start memory (CCL START MEM)	Asserted by the channel during channel write operations after the first word is loaded into the MBs. During channel read operations, the cache cycle control starts the core cycle when it is ready.

Table 3-1 CHAN/CSH Line Descriptions (Cont)

Signal Line	Description
Memory buffer select 1–2 (CCL CH MB SEL 1–2)	The channel places a 2-bit code on these lines to select the correct MB to be loaded during channel write operations or read during channel read operations.
Load memory buffer (CCL CH LOAD MB)	Asserted by the channel to load the selected MB during channel write operations.
Hold in (MB0–3 HOLD IN)	Asserted by the cache cycle control and/or the core cycle control during a channel read operation to load the MBs and to inform the channel that the corresponding word is ready to be taken.
Memory buffer test parity (CCL CH MB TEST PAR)	Asserted by the channel to check parity of the selected word before it is taken from the MB during channel read operations.
Request Qualifiers	
Channel to memory (CCL CHAN TO MEM)	Asserted by the channel to specify an execution of a channel write operation. When negated, a channel read operation is executed.
Channel word 0–3 request (CCW CHAN WD0–3 RQ)	These four signals are asserted by the channel to specify the words to be read or written.
Exec Process Table (CCL CHAN EPT)	Asserted by the channel to read or write the Exec Process Table (EPT). The EPT is read to fetch the initial CCW and is written to store the channel status at the end of a transfer. The cache cycle control will automatically select the correct address for referencing the EPT.
Error Reporting Commands	
Channel parity error (CHAN PAR ERR)	Asserted for one clock period when the MB parity check fails during a channel read or channel write operation. During channel write operations, parity is checked when the channel asserts CCL CH MB TEST PAR. During channel read operations, parity is checked when the cache cycle or the core cycle control loads the word into the MB.

Table 3-1 CHAN/CSH Line Descriptions (Cont)

Signal Line	Description
Address parity error (CHAN ADR PAR ERR)	Asserted for one clock period when the SBus address parity check fails during channel read or channel write operations.
Nonexistent memory error (CHAN NXM ERR)	Asserted for one clock period on detection of a nonexistent memory error.
Address	
Physical memory address (CCN CHA 14–35)	Physical memory address from channel.
DATA	Data buffer and path is an integral part of the MB modules (MB CH BUF 00–35).
CLOCKS	Clocks are distributed to the channels from the EBox.

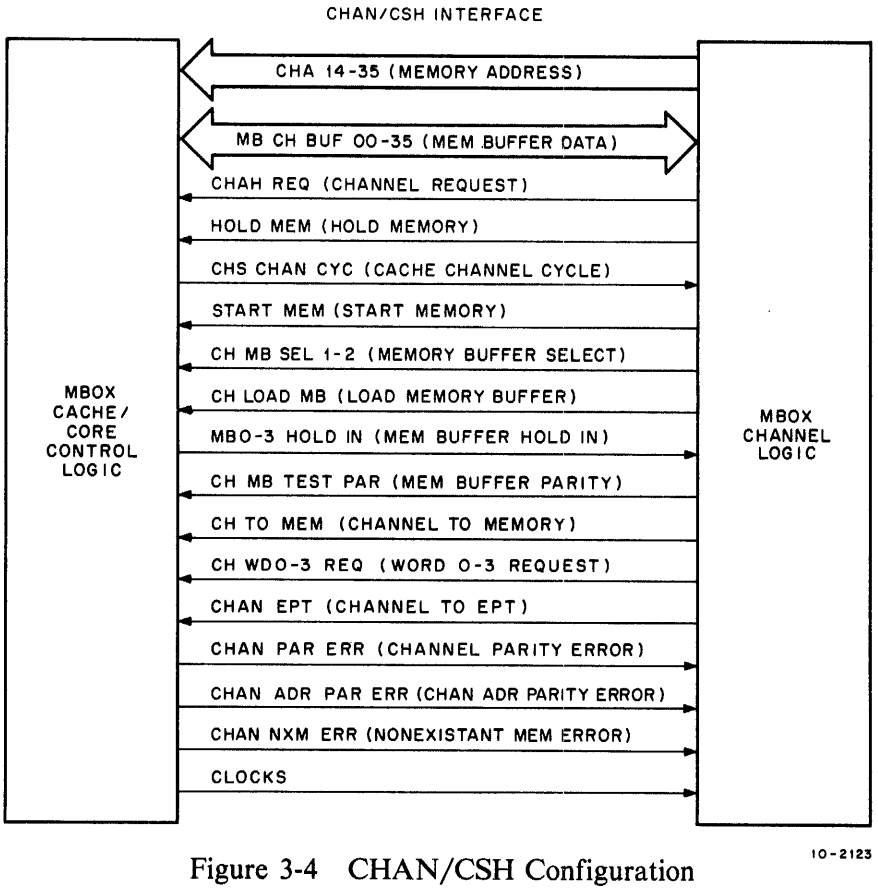


Figure 3-4 CHAN/CSH Configuration

3.2.3.1 Request Dialog – The channels issue requests to the cache cycle control for memory cycles by asserting CCL CHAN REQ and CCL HOLD MEM (see Figure 3-5). The channels also set up the channel address (CHA) and the following request qualifiers.

1. CCL CHAN TO MEM
2. CCL CHAN WD0–3RQ
3. CCL CHAN EPT

These signals stay valid until the request has been processed to completion. If another request is ready to be processed, CCL CHAN REQ and CCL HOLD MEM stay asserted while the address and request qualifiers are modified to specify the next request.

When the cache cycle control starts to process a request, the cache cycle control asserts CSH CHAN CYCLE. This signal informs the channel that it can start moving words from the CHAN to the MBs in the case of channel data write operations, or it can start looking for words that are ready to be moved out of the MBs into the channel buffer (CH BUF), in the case of channel data read operations.

3.2.3.2 Channel Read Operations – Two types of read requests can be issued by the channels.

1. Read a single word from the Executive Process Table (EPT). The EPT contains eight locations for storing the initial channel command words (CCW). One location is assigned to each channel.

2. Read one, two, three, or four words (instructions and data) from physical core memory.

To read the initial CCW from the EPT, the channel issues and qualifies the request as follows.

1. Assert CCL CHAN REQ, CCL CHAN EPT, CCW CHAN WD0 RQ.
2. Clear CCL CHAN TO MEM.

NOTE
Word 0 is requested because the initial CCW is stored in location 0 of a quadword group.

1. Set up CCL CH MB SEL 1–2 lines to point to MB0.
2. Assert CCL HOLD MEM.
3. Hold CHA 14–35.

The channel then waits for a cache cycle. When the cache cycle is started, the correct address is made by replacing CHA 14–26 with the contents of the EBR. This address is then used to look in the cache and if the word is not in the cache, to read the word from core. In either case, the word is moved into MB0. The channel recognizes that MB0 was loaded when MB0 HOLD IN is negated for one clock tick. The channel will then move the word from MB0 to the CCW BUF and cause MB parity to be checked.

To read data and instructions from physical core memory, the channel issues and qualifies the request as follows.

1. Assert CCL CHAN REQ and CCL HOLD MEM.

2. Clear CCL CHAN TO MEM and CCL CHAN EPT.

3. Set up CCW CHAN WD0–WD3 lines to indicate which words are to be read.

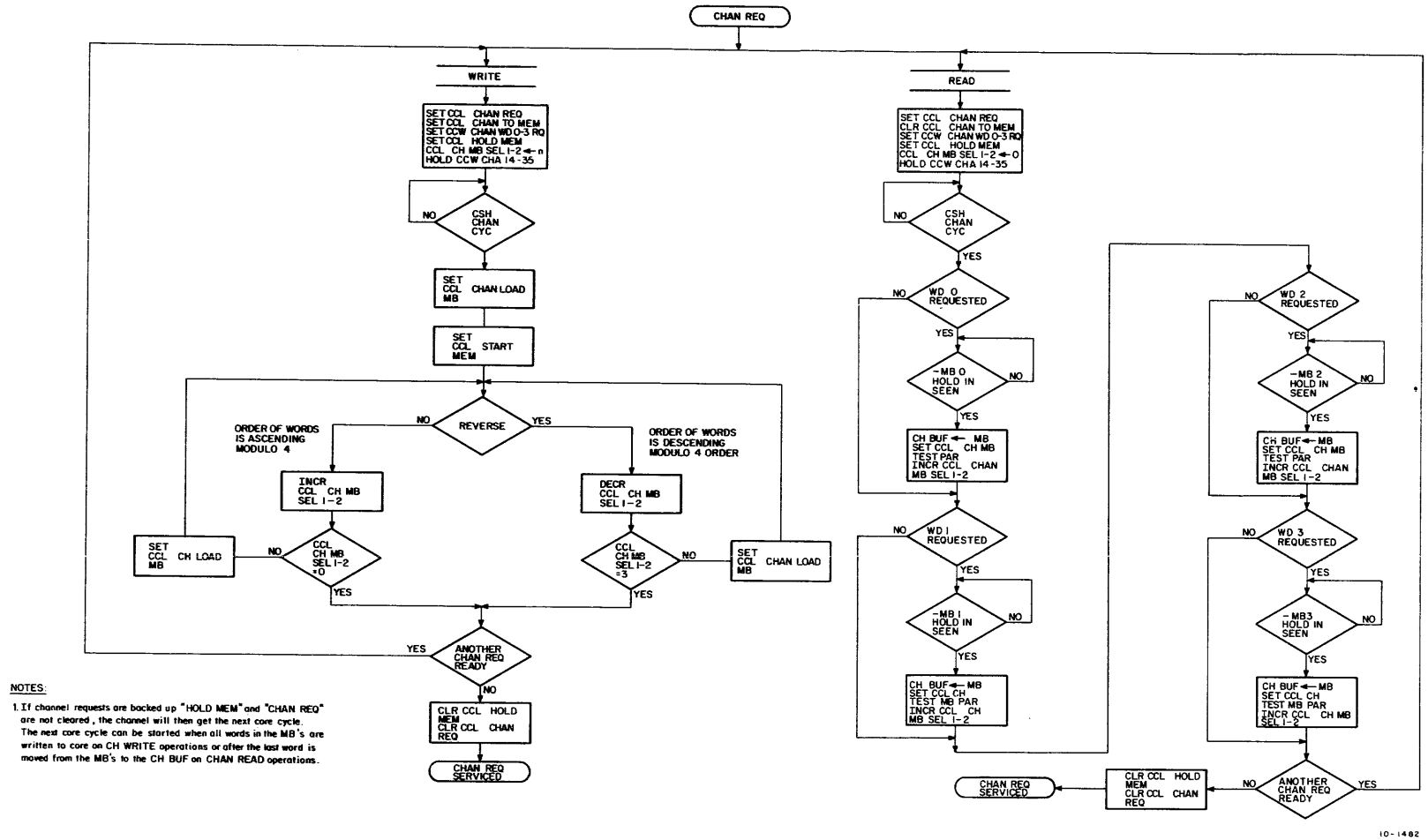


Figure 3-5 Channel Request Dialog

4. Set up CCL CH MB SEL 1-2 lines to point to the MB that corresponds to the lowest order word requested.

5. Hold CHA 14-35.

The channel then waits for a cache cycle. When the cache cycle is started, the channel address (CHA 14-35) is used to look in the cache and if all the requested words are not in the cache, to read those words from core. In either case, the requested words are moved into the MBs. The channel recognizes that an MB is loaded when MB0, 1, 2, or 3 HOLD IN is negated for one clock tick. The channel will start moving the words to the CH BUF as soon as the lowest order requested word is placed into the corresponding MB. Subsequent words are moved from the MBs to the CH BUF in ascending modulo four order. As each word is transferred, its parity is also checked in the MB.

3.2.3.3 Channel Write Operations – Two types of write requests can be issued by the channels.

1. Write two words into the Executive Process Table (EPT). The EPT contains 16 locations for storing channel status information. Two locations are assigned to each channel.

2. Write one, two, three, or four words (data and instruction) into physical core memory.

NOTE
These words may have been read from a magnetic tape drive that is capable of reading in forward and reverse mode.

To write the two status words into the EPT, the channel issues and qualifies the request as follows.

1. Assert CCL CHAN REQ and CCL CHAN EPT.

2. Assert CCL CHAN TO MEM and CCW CHAN WD1 and WE2 REQ.

NOTE
Words 1 and 2 are specified since the status words are stored in locations 1 and 2 of a quadword group.

3. Set up CCL CH MB SEL 1-2 lines to point to MB1.

4. Assert CCL HOLD MEM.

5. Hold CHA 14-35.

The channel then waits for a cache cycle. When the cache cycle is started, the correct address is made by replacing CHA 14-26 with the contents of the EBR. This address is then used to write the words to core after they are moved to the MBs. The cache is also checked to see if there is a copy of the referenced EPT locations in the cache, if EBOX CACHE LOOK EN is set. If there is, this copy is invalidated because it is assumed to be an old copy. After the first word is moved into the MB, the channel initiates a core write cycle to move the word to memory. The second word is moved into its MB 125 ns after the first word, in time for the memory control.

To write data and instructions into physical memory, the channel issues and qualifies the request as follows.

1. Assert CCL CHAN REQ and CCL CHAN TO MEM.

2. Clear CCL CHAN EPT.

3. Set up CCW CHAN WD0–WD3 RQ lines to indicate which words are to be written.

4. Set up CCL CH MB SEL 1 2 lines to point to the MB that corresponds to the first word to be transferred by the channel.

5. Assert CCL HOLD MEM.

6. Hold CHA 14–35.

NOTE

If the words were read from a magnetic tape drive operating in the forward mode, the words will be transferred in ascending modulo four order. However, if the drive was operating in the reverse mode, the words will be transferred in descending modulo four order.

The channel then waits for a cache channel cycle. When the cache cycle is started, the channel address (CHA 14–35) is used to write the words into core after they are moved into the MBs. The cache is also checked to see if there is a copy of the referenced memory locations in the cache. If there is, this copy is invalidated, since it is assumed that this must be an old copy. After the first word (lowest numbered word) is moved into the MB, the channel initiates a core write cycle to move the word to core. Subsequent words are moved into the MBs at four clock-tick intervals so that the words will be available for transfer to core. Once a core cycle is started, the core cycle control moves a word to core every six clock ticks.

3.2.4 Meter Board

This board contains the following programmable clocks, each of which provides a different timing or counting function.

1. Interval timer
2. Time base
3. Accounting meters
4. Performance analysis counter

This board uses ECL logic and is housed on the ECL CPU assembly along with the EBox and the MBox. The clocks are considered to be internal processor I/O devices (TIM and MTR) and are programmable using the standard I/O instructions.

The interval timer provides a programmable source of interrupts having 10 μs resolution and a choice of $2^{12}-1$ possible time intervals ranging from 10 μs to 40950 μs.

The readable time base is a long term clock for measuring elapsed time with 1 μs resolution. (Long term power line frequency time base is provided by the front-end processor.) It uses a 1.0 MHz (+0.005 percent) frequency source, derived and down-counted from the basic 50 MHz machine clock. The time base has less than 5 seconds of drift over a 24 hour period.

The accounting meters have an EBox busy meter, which counts when the EBox is executing microcode, and a memory cycle meter, which counts the number of EBox memory references. The two meters provide

a reproducible measure of the processor resources used by a program, and they can be used for billing users and for benchmark or comparison purposes.

The performance analysis counter serves as a tool for testing and evaluating the DECsystem-10/DECSYSTEM-20. It monitors either the duration or the rate of occurrence of several hardware signals from various parts of the main processor. The signals, chosen for their usefulness in evaluating system performance, define machine states and conditions that are not easy to measure by software techniques. The signals to be monitored are selected by means of a Boolean expression loaded by the program.

3.2.5 EBox/MBox (E/M) Interface Description

The EBox/MBox interface connects the EBox and MBox portions of the central processor. This group of signals does not actually make up a bus; however, the signal-sequencing characteristics are similar to a bus. As such, this signal group will be described as a bus.

The EBox asserts a set of interface signals (request qualifiers) along with EBox request to specify what type of service is required. Request qualification is required to show the difference between reads and writes and between memory and register references. In addition to these basic qualifications, each request is qualified by asserting other signals to identify the register of interest for a register reference, and to indicate the type of addressing to be used and whether the cache is to be used for memory references. After the MBox executes a cache cycle to process an EBox request, the MBox asserts MBox response to inform the EBox that the operation is complete.

The E/M interface lines are described in Table 3-2. Figure 3-6 shows the E/M interface configuration. Major bus line sequencing is described in the following paragraphs in the context of a specific operation.

Table 3-2 E/M Interface Line Descriptions

Signal Line	Description
Control Commands	
EBox request (EBox REQ)	Issued by the EBox to request service from the MBox.
MBox gate virtual memory address 27–35 (MBOX GATE VMA 27–35)	Asserted by the MBox when a cache EBox cycle is granted to service the EBOX REQ to enable VMA bits 27–35 for addressing the cache directory.
Cache EBox to in (CSH EBOX TO IN)	Asserted for one clock period when the cache cycle control starts processing an EBox request. This signal is used to clear EBOX REQ.
EBox accumulator reference (EBOX AC REF)	Asserted by the EBox when it finds that the reference is to one of the AC blocks (fast memory) to abort the MBox cache cycle if it was started. This is done to allow the MBox to start servicing a request earlier than would otherwise be possible.

Table 3-2 E/M Interface Line Descriptions (Cont)

Signal Line	Description
Page table public (PT PUBLIC)	Transferred to the EBox to allow the EBox to determine whether it should assert PAGE ILLEGAL ENTRY for the next reference or change its mode of operation from public to private.
EBox page fail handle (EBOX PF HANDLE)	Asserted by the MBox, this is a page test for a paged memory request failed, and KL paging mode is specified by the EBox.
MBox page fail hold (MBOX PAGE FAIL HOLD)	Asserted by the MBox, this is the page test for a paged memory reference request failed.
MBox response (MBOX RESP)	Asserted by the MBox after a request is processed.
EBox sync (EBOX SYNC)	Asserted by the EBox to inform the MBox the data will be taken.
Memory Reference Request Qualifiers	
EBox read (EBOX READ)	Read a word from memory. Read-check the page for paged references and assert MBOX PAGE FAIL HOLD if page test failed.
EBox write (EBOX WRITE)	Write a word into memory. Write-check the page for paged references and assert MBOX PAGE FAIL HOLD if page test failed.
EBox read and EBox write (EBOX READ and EBOX WRITE)	Read a word from memory, read- and write-check the page for paged references and assert MBOX PAGE FAIL HOLD if page test failed.
EBox read, EBox pause, and EBox write	Execute the read portion of the read-pause-write cycle. Read- and write-check the page for paged references and assert MBOX PAGE FAIL HOLD if page test failed. The write portion of the cycle is initiated by asserting EBOX REQ a second time.
EBox pause and EBox write	Write-check the page for paged references and assert MBOX PAGE FAIL HOLD if page test failed.

Table 3-2 E/M Interface Line Descriptions (Cont)

Signal Line	Description
EBox may be paged (EBOX MAY BE PAGED)	Asserted by the EBox indicating the reference may be paged. The MBox determines whether the reference is paged. In the KL paging mode, all core is paged; in KI paging mode, part of the executive address space is not paged.
EBox KI paging mode (EBOX KI PAGING MODE)	Indicates KI paging mode when asserted and KL paging mode when negated.
EBox user (EBOX USER)	Asserted by the EBox when the memory reference is to the user address space.
EBox user executive base register reference (EBOX UEBR REF)	Asserted by the EBox when the User or Exec Process Table is referenced to bypass the page check.
EBox User Process Table (EBOX UPT)	Asserted by the EBox when the reference is to the User Process Table to inform the MBox that the contents of the user base register must be used in forming the physical memory address (PMA).
EBox Exec Process Table (EBOX EPT)	Asserted by the EBox when the reference is to the Exec Process Table to inform the MBox that the contents of the executive base register must be used in forming the PMA.
Page illegal entry (PAGE ILL ENTRY)	Asserted by the EBox to force a page fail condition in the MBox to abort the current request. The EBox asserts PAGE ILL ENTRY if the previous instruction was fetched from a proprietary area and the instruction is not a portal instruction (JRSTO).
Page test private (PAGE TEST PRIVATE)	Asserted by the EBox for a noninstruction reference in the public mode to check whether the page is private. MBOX PAGE FAIL HOLD is asserted if the page is not public.
Page address condition (PAGE ADDRESS COND)	Asserted when the EBox detects an address break condition. The EBox also asserts PAGE ILL ENTRY at this time to force a page fail condition in the MBox and cause MBOX PAGE FAIL HOLD to be asserted.

Table 3-2 E/M Interface Line Descriptions (Cont)

Signal Line	Description
EBox cache (EBOX CACHE)	Asserted by the EBox for references to those instructions and data that may reside in the cache. Instructions and data that are shared by two processors cannot reside in the cache.
Cache look enable (CACHE LOOK EN)	Asserted by the EBox to take the word from the cache if it is found, even if EBOX CACHE is clear, or for paged references if PT CACHE is cleared.
Write even parity (CON WR EVEN PAR)	Asserted by the EBox to write even parity into the cache directory during a write request.
SBus diagnostic (SBUS DIAG)	Asserted by the EBox to initiate an SBus diagnostic cycle. All other request qualifiers must be clear for this request.
Register Reference Request Qualifiers	
EBox load register (EBOX LOAD REG)	Asserted by the EBox to load the UBR, EBR, or CCA in the MBox. The EBox also specifies which register is to be loaded by asserting the appropriate register signal.
EBox read register (EBOX READ REGISTER)	Asserted by the EBox to prepare to read a register (UBR, EBA, CCA, ERA) in the MBox. The EBox also specifies which register is to be read by asserting the appropriate register signal. After the READ REG request is executed by the MBox, the EBox can read the value of the register by asserting EBOX READ EBUS REG.
EBox user base register (EBOX UBR)	Asserted by the EBox when the UBR is to be loaded or read.
EBox executive base register (EBOX EBR)	Asserted by the EBox when the EBR is to be loaded or read.
EBox cache clearer register (EBOX CCA)	Asserted by the EBox when the CCA is to be loaded or read.

Table 3-2 E/M Interface Line Descriptions (Cont)

Signal Line	Description
EBox accumulator bit 10 (EBOX AC10)	Accumulator bit 10 (AC10) is set when only one page is to be cleared from the cache; it is reset when all of cache is to be cleared.
EBox accumulator bit 11 (EBOX AC11)	Accumulator bit 11 (AC11) is set when the cache entries are to be invalidated.
EBox accumulator bit 12 (EBOX AC12)	Accumulator bit 12 (AC12) is set when the written words in the cache are to be written back into core to validate core.
EBox error address register (EBOX ERA)	Asserted by the EBox when the ERA register is to be read. This register is a read-only registers.
EBox map (EBOX MAP)	Asserted by the EBox along with EBOX READ REG to transform the virtual address into the physical address.
DIA enable refill RAM write (DIA EN REFILL RAM WR)	Asserted by the EBox along with EBOX READ REG to load the cache refill RAM when the cache is initialized.
Error Reporting Commands	
MBox nonexistent memory error (MBOX NXM ERROR)	Asserted by the MBox when an NXM is detected.
MBox SBus error (MBOX SBUS ERR)	Asserted by the MBox when SBUS ERR is detected on SBus. SBUS ERROR is asserted by memory subsystem.
MBox memory buffer parity error (MBOX MB PAR ERR)	Asserted by the MBox when an MB parity error is detected.
MBox address parity error (MBOX ADR PAR ERR)	Asserted by the MBox when an address parity error is detected by the memory system.
Cache address parity error flag (CACHE DIR PAR ERR)	Asserted by the MBox when a cache directory parity error is detected.

Table 3-2 E/M Interface Line Descriptions (Cont)

Signal Line	Description
APR any EBox error flag (APR ANY EBOX ERR FLG)	Serves as an accumulative error flag for the APR register.
Direct Commands	
EBox page table directory write (EBOX PT DIR WRITE)	Asserted by the EBox during KL paging mode to write or clear a page table directory entry.
EBox page table write (EBOX PT WRITE)	Asserted by the EBox during KL paging mode to write or clear a page table entry.
Diagnostic read function (DIAG READ FUNCT 16X–17X)	Asserted by the EBox to read a diagnostic register. The diagnostic register to be read is specified by the code presented on DIA 04–06. Also asserted by the EBox to read the EBus register. Octal code seven must be presented on the DIAG 04–06 lines to read the EBus register. This register will contain the contents of the register specified with the EBox READ REG request or it will contain the PAGE FAIL WORD in the event the MBox pager sensed a page fail condition. The EBox is informed that a page fail condition was sensed by the MBox (PAGE FAIL HOLD is asserted by the MBox).
Diagnostic load function (DIAG LOAD FUNCT 071)	Asserted by the EBox to set up the MEM TO C mixer to read the contents of the memory data register (SBus) the MBs or the cache. The contents of the AR can also be looped back. The code presented on the EBus data bits 30–33 determines which data specified above will be read back on the cache data lines.
Diagnostic lines (DIAG 04–06)	These lines present a control code to the MBox for selecting diagnostic and EBus registers.
Address Lines	
Virtual memory address 13–35A (VMA 13–35A)	Register load data or virtual address from the EBox. Address parity is not propagated.
Virtual memory address 27–35G (VMA 27–35G)	Gated address from EBox. This address is gated by the MBox when a cache EBox cycle is started. Address parity is not propagated.

Table 3-2 E/M Interface Line Descriptions (Cont)

Signal Line	Description
Data Lines	
Arithmetic register 00–35 (AR00–35)	Transfers data from the EBox AR to the cache, memory buffers, or page table in the MBox.
Arithmetic register parity (AR PAR)	Transfers data parity from the AR parity generator, together with AR00–35.
Cache data 00–35B (CACHE DATA 00–35B)	Transfers data from the MBox cache to the EBox IR, AR, and ARX.
Cache data 00–35C (CACHE DATA 00–35C)	Transfers data from the MBox cache to the EBox IR.
EBus data 00–35 (EBUS D00–35)	Transfers data from the EBus register and mixer to the EBus.
Clock	13 clocks are generated on the EBox CLK module and distributed to the MBox boards.

3.2.5.1 Basic Request Dialog – The EBox issues requests to the MBox by asserting EBOX REQ (Figure 3-7). At the same time EBOX REQ is asserted, the request qualifiers become valid. These signals stay valid until the request has either been processed to completion or aborted. The request can be aborted by the EBox by asserting EBOX AC REF. If the EBox aborts the request, EBOX REQ is also cleared by the EBox (if the MBox has not started to process the request).

When the MBox starts to process the EBox request, the MBox asserts CSH EBOX TO IN. This signal causes EBOX REQ to be cleared. This occurs after the request is made if the MBox has no higher priority request pending. If the MBox is busy when the request is made, a number of clock ticks may transpire before the MBox asserts CSH EBOX TO IN. Therefore, EBOX REQ will stay asserted until the MBox starts processing the request.

After CSH EBOX TO IN is asserted, a number of clock periods may occur before the MBox completes processing the request. The MBox informs the EBox that it has completed processing the request by asserting MBOX RESP. This signal stays asserted until the EBox asserts EBOX SYNC. While MBOX RESP is asserted, the data or instruction requested by the MBox will be valid on the cache data lines. The MBox holds the data on the cache data lines until EBOX SYNC is asserted, since the EBox will take the data only when EBOX SYNC is asserted. One clock tick after EBOX SYNC is asserted, MBOX RESP is cleared.

3.2.5.2 Register References – The MBox contains a number of registers that can be loaded and read by the EBox. These registers are address registers for storing the address in the event of an error and

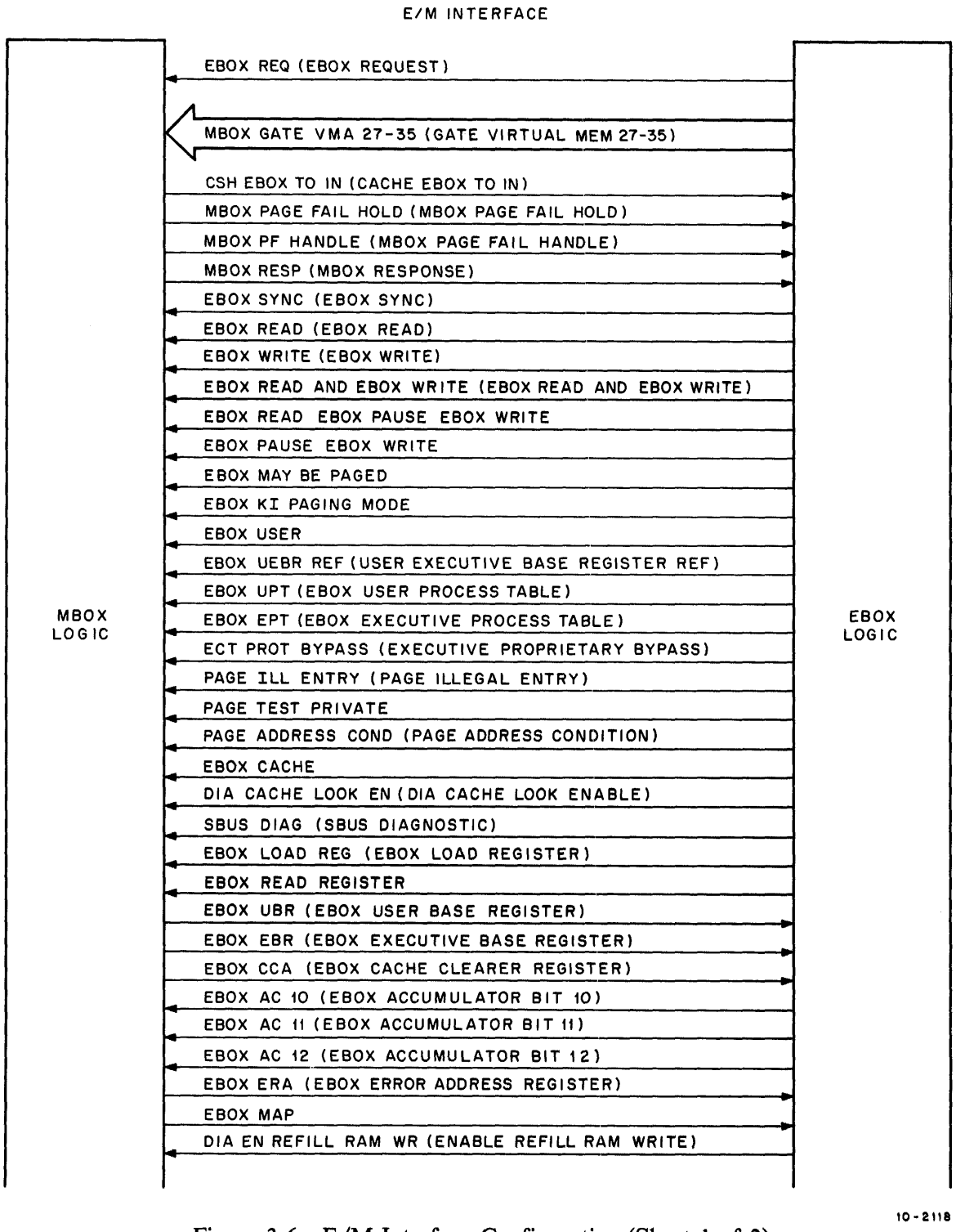


Figure 3-6 E/M Interface Configuration (Sheet 1 of 2)

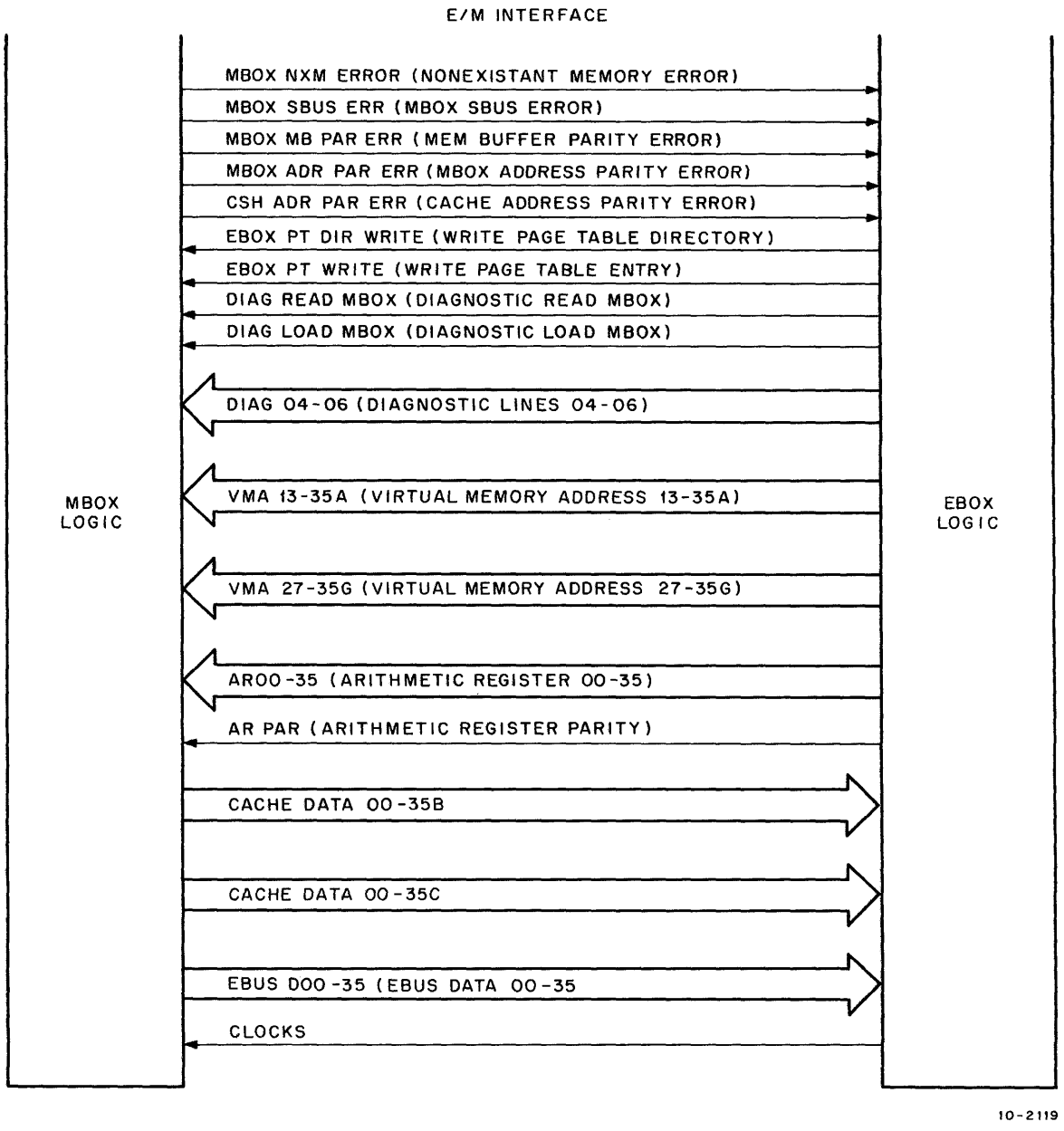


Figure 3-6 E/M Interface Configuration (Sheet 2 of 2)

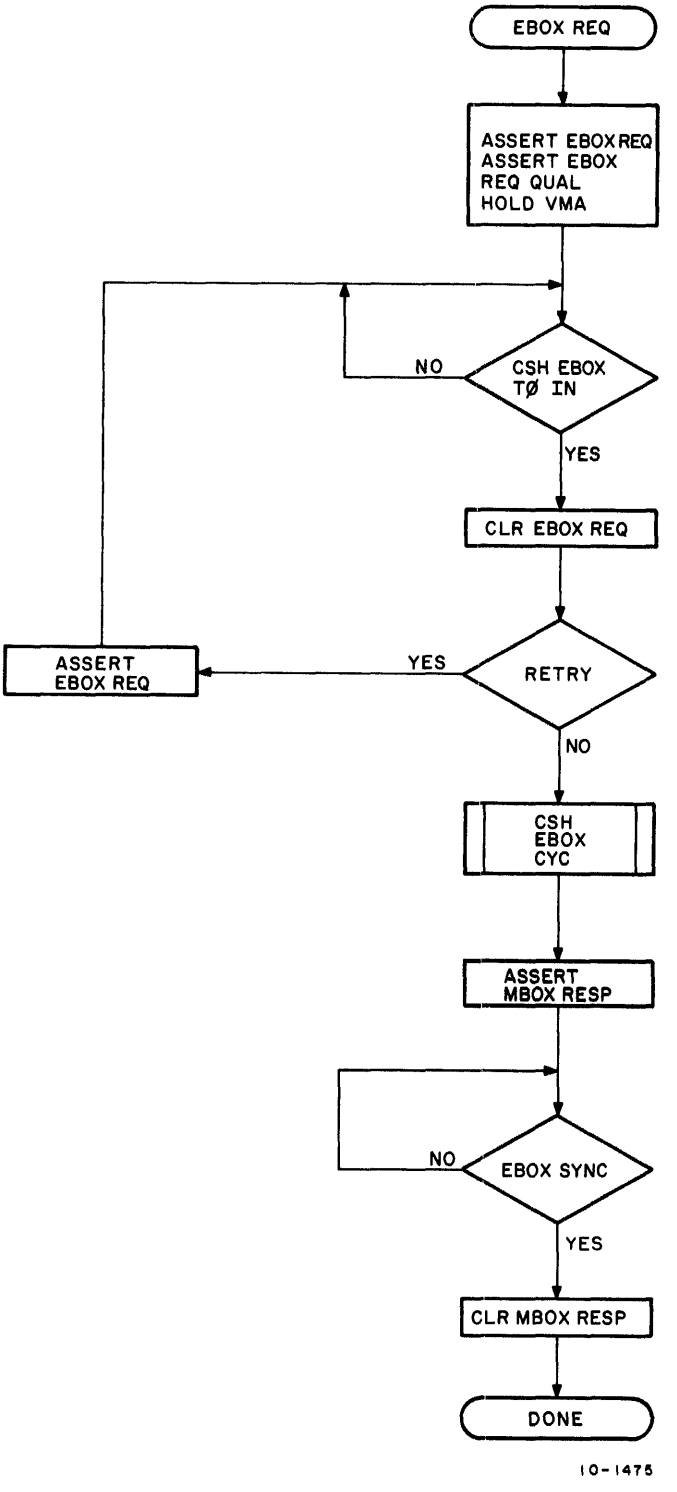


Figure 3-7 EBox Request Dialog

modifying the physical memory address in response to certain request qualifiers. The registers are:

1. User base registers – UBR
2. Executive base register – EBR
3. CCA cache clearer address – CCA
4. Error address – ERA.

NOTE
The ERA register can only be read by the EBox.

In addition, the EBox can also read the contents of the page table to map the virtual address to the physical address and load the cache refill RAM with the cache refill algorithm.

To read and load any of the registers and RAMs previously listed, the MBox must execute a cache cycle in response to the EBox request; this prevents potential conflicts with other pending requests. Some registers and RAMs can also be loaded and read directly by the EBox without executing a cache cycle. A conflict with another type of request (e.g., CHAN REQ) cannot occur with these registers and RAMs.

The MEM TO C diagnostic register and the page table can be loaded, and seven diagnostic registers plus the EBus register can be read directly from the EBox.

To read or write the registers and RAMs in the MBox, the EBox must assert a specific set of qualifier signals along with EBOX REQ for each type of reference. When loading registers, the EBox must also move the data to be loaded into the VMA before issuing the request.

3.2.5.3 Memory Reference – The EBox can issue requests to read and write memory. The EBox can request to read or write the Executive and User Process Tables and user or executive paged and unpaged memory. The EBox will also specify whether the cache is to be used in servicing the memory request. When the MBox starts processing a memory request, it automatically forms the correct physical memory address in response to the request qualifiers displayed with the request. If the EBox requested a reference to paged memory, it automatically read/write-checks the referenced page. If the page check fails, the MBox informs the EBox of this condition by asserting PAGE FAIL HOLD.

To read or write memory, the EBox must set up the address in the VMA and assert a specific set of qualifier signals along with EBOX REQ for each type of reference. When writing memory, the EBox must also move the data (or instruction) to be written into the AR before issuing the request.

3.2.5.4 Basic Cache Strategy – The MBox is the storage controller in the KL-based DECsystem-10/DECSYSTEM-20. The MBox contains a cache, four memory buffers (MBs), the pager, and the PMA. These functional elements provide the EBox access to core memory. The PMA and the pager form the physical memory address. The data path between core memory and the EBox is created by the MBs and the cache. The cache is a high-speed, 2048 word data buffer where instructions and data are stored and maintained as the EBox issues requests over the E/M interface for memory.

It is characteristic of programs executed by the EBox to execute the same instructions many times, as in iterative program loops. The cache serves its function in these cases. Once instructions and data have been moved from memory to cache, the EBox can fetch instructions and store results much faster on subsequent references, since a time-consuming memory cycle will not have to be executed. Therefore, the cache can be considered a high-speed, high-usage extension of memory.

As the EBox makes requests for instructions and data, memory cycles are granted by the MBox; and the cache is filled up with four words at a time (a quadword). Data is transferred from core to the cache via the four MBs. Considering that it is probable the EBox will request the next consecutive word in a string,

the word will already be in the cache. It will be available to the EBox sooner, since it will come from the cache instead of from memory.

When the EBox makes a request for a word that is not already in the cache, the MBox will grant another memory cycle to place another quadword (four words) in the cache. To identify each quadword group, the cache contains a directory that stores the physical page number of the quadword (ADR). The directory also contains locations for the purpose of identifying which words are valid (VAL bit) and which words were written by the EBox (WR bits). As the cache is filled with data and instructions, the associated locations of the directory are updated to specify the physical page address (ADR) of the quadword and to specify which words were fetched from core (VAL bits). This continues until the cache is filled. Therefore, the least recently used quadwords are replaced by new instructions and data as they are needed. Words that have been written into the cache by the EBox are identified by updating directory WR bits accordingly so that they can be moved back to core before they are replaced.

3.2.6 XBus Description

The XBus connects the MBox to the MF20 MOS memory system. It consists of a 36-bit wide data path with associated control and diagnostic lines. The basic memory system consists of two controllers and their associated storage modules to provide storage for up to 3,145,728 44-bit words in increments of 256 K words. Table 3-3 describes each XBus line and Figure 3-8 illustrates the XBus configuration.

During diagnostic cycle execution, all communication (including addressing) between the MBox and the MF20 is performed over the XBus data lines and associated diagnostic and error reporting lines. There are eleven diagnostic functions implemented in the MF20 hardware. The diagnostic functions permit the program to access the MF20 logic in a nonstandard way for one or more of the following purposes.

1. Loading a control RAM to configure the memory after power up
2. Reading error history flip-flops to log and analyze memory errors
3. Reading the state of key logic signals to diagnose memory faults
4. Setting up and starting the spare bit mechanism

The diagnostic functions are described under Paragraph 3.2.6.5. Examples are also provided to explain function use.

Table 3-3 XBus Signal Summary

Signal Name	Print	Function
ADR 14–35 ADR 14–21 ADR 22–33 ADR 34–35	 SYN0 ADT0 CTL0	These 22 lines transfer the physical address from the MBox to the MF20 MOS memory subsystem. Used to uniquely identify a specific block within a specific group in a specific MF20.
ADR PAR	ADT0	A single line that is controlled by the MBox whenever a memory reference is started. It is used to establish overall odd parity for the following lines. ADR 14–35 RD RQ WR RQ RQ 0–3 The MF20 checks the parity and asserts the XBus signal, ADR PAR ERR if an error is detected.

Table 3-3 XBus Signal Summary (Cont)

Signal Name	Print	Function
RQ 0–3	CTL0	Four lines that are asserted by the MBox to specify which words in a quadword group are to be accessed. RQ0 request word 0 RQ1 request word 1 RQ2 request word 2 RQ3 request word 3 If more than one word is requested, ADR 34–35 will specify the order in which the words will be processed.
RD RQ	CTL0	The MBox asserts this line to specify that the request is a read request.
WR RQ	CTL0	The MBox asserts this line to specify that the request is a write request. **Note** **During a read-modify-write both RD RQ and WR RQ are asserted by the MBox.**
START A–B	CTL0	The MBox asserts one of these two signals to signal the MF20 to start executing the read/write function specified by the state of the RD RQ, WR RQ, and RQ 0–3 lines.
ACKN A–B	CTL0	The selected MF20 responds to the MBOX START A–B signal by asserting the corresponding ACKN A or ACKN B signal to acknowledge the request. If no memory unit asserts ACKN within a specified time limit, the MBox will abort the request and set a nonexistent memory (NXM) flag.
D00–35	WRP0, 1	These 36 lines are used to transport a single KL10 data word either from the MF20 to the MBox during a read or from the MBox to the MF20 during a write.
DATA PAR	WRP1	This line is used to establish odd parity on the data bus, D00–35, during any read or write operation. Read – The MF20 generates the parity, and the MBox checks it. Write – The MBox generates the parity, and the MF20 checks it.
DATA VALID A–B	CTL0	Two lines asserted by either the MF20 (read) or the MBox (write portion of read-modify-write) to signal when data is valid on the data bus.

Table 3-3 XBus Signal Summary (Cont)

Signal Name	Print	Function
DIAG	ADT0	The MBox asserts this line to signal the MF20 selected by D00–04 that it is to perform the diagnostic function specified by D31–35.
ERROR	ADT0	This line is asserted by the selected MF20 to signal the MBox that one of the following types of errors has been detected. RD ERR WR PAR ERR ADR ERR TIM RAM ERR SUB RAM ERR BLK RAM ERR INC ERR The MBox can execute a subsequent diagnostic function 1 or 2 to determine which of the above errors has occurred.
ADR PAR ERR	ADT0	This line is asserted by the MF20 to signal that bad address parity was detected during a read-write operation.
MEM RESET	CTL0	The MBox can assert this signal to place the MF20 in a known initial state.
CROBAR	WRP0	This signal clears the MF20 control logic to its initial state during power-up or power-down.
CLK INT	CTL0	XBus clock used to synchronize all operations within the MF20 to the MBox timing.
VREF	CTL0	A single line that establishes a common voltage reference for the XBus transceivers at both ends of the XBus.

3.2.6.1 Read Dialog – When the MBox prepares to retrieve information from the MF20, it initiates the following XBus sequence.

1. It asserts the address lines, XBus ADR 14–35, to specify the location of the first word it wants to retrieve.
2. It asserts the XBUS RD RQ line to instruct the MF20 to initiate a read cycle.
3. It asserts up to four request lines, XBUS RQ 0–3, to specify which words within a quadword group it wants to retrieve.

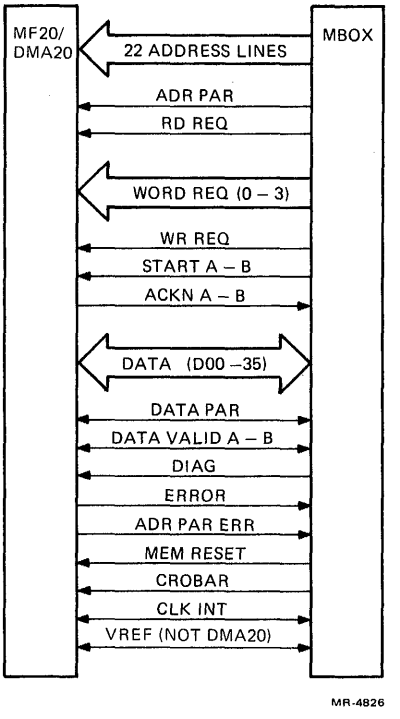


Figure 3-8 XBus Signals

4. It calculates odd parity based on the state of all address and request lines and either asserts or negates the XBUS ADR PAR signal to reflect this parity.

5. It then asserts either XBUS START A or XBUS START B to command the MF20 to execute the read request.

6. The selected MF20 uses XBUS START A/B to store the address and request information and initiates a MOS memory timing sequence to access and retrieve the information from the MOS storage array.

7. The MF20 responds immediately to the MBox START by asserting the signal XBUS ACKN A/B and keeps it asserted for N number of XBUS INT CLK periods, where N is the number of words requested. The MBox drops START and negates all other control lines after the last ACKN period.

8. As the words from the MOS array become available, the MF20 gates the information back on the XBUS D00–35, DATA PAR lines along with the control signal XBUS DATA VALID A,B.

9. The MBox uses DATA VALID to strobe the data lines into the appropriate memory buffer register where they are subsequently checked for parity and distributed to their destination (EBox, MBox, channel buffer, cache, and so on).

10. After all the words have been processed, the MF20 returns to its initial state (if a refresh request is not pending) and is ready to accept a new command.

3.2.6.2 Write Dialog – When the MBox prepares to store information in the MF20, it initiates the following XBus sequence.

1. It asserts the address lines, XBus ADR 14–35, to specify the location of the first word to be written.

2. It asserts the XBUS WR RQ line to instruct the MF20 to initiate a write cycle.

3. It asserts the XBUS RQ 0–3 lines to specify which words within the referenced quadword group are to be written.

4. It asserts XBUS ADR PAR to generate odd parity based on the state of all address and control lines (ADR 14–35, WR RQ, RQ 0–3).

5. It asserts the XBUS D00–35, DATA PARITY lines to reflect the first word to be written.

6. It asserts XBUS START A/B to command the MF20 to execute the write request.

7. Upon receiving START, the MF20 responds by asserting XBUS ACKN and maintains it asserted for N number of XBUS CLK INT periods, where N is equal to the number of request lines asserted. It also gates the XBUS D00–35, DATA PAR lines to store the first word to be written in the MF20's port data buffer.

8. During subsequent XBUS INT CLK periods, the MBox gates the rest of the words to be written onto the XBUS DATA lines where they are taken and stored by the MF20.

9. After receiving all the words (up to a maximum of four), the MF20 negates XBUS ACKN and initiates a MOS write timing sequence to store the words in the MOS storage array.

10. After writing the MOS, the MF20 returns to its initial state and is ready to accept another command.

3.2.6.3 Read-Pause-Write Dialog – When the MBox prepares to retrieve a single word from the MF20, modify it in the EBox, and then store the result back in the MF20, it initiates the following XBus sequence.

1. It asserts the XBUS ADR 14–35 lines to specify the location of the word to be modified.

2. It asserts both the XBus RD RQ and XBUS WR RQ lines to instruct the MF20 to execute both a read and a write cycle.

3. It asserts one of the XBUS RQ 0–3 lines to specify which word in the quadword group is to be modified.

4. It asserts or negates XBUS ADR PAR to generate odd parity for all the address and control lines.

5. It asserts XBUS START A/B to command the MF20 to execute the read-modify-write operation.

6. On receiving START, the selected MF20 responds by asserting XBUS ACKN A/B and initiates a MOS storage array timing sequence to retrieve the word from MOS.

7. When the word is available, it is gated out of the MF20 back to the MBox on XBUS DATA D00–35, DATA PAR along with the control signal XBUS DATA VALID A/B.

8. The MBox uses DATA VALID to gate the data lines into the appropriate memory buffer register (MBR) where it is sent to the EBox for manipulation.

9. The MF20 stops at this point and waits for the MBox to respond.

10. When the data is ready to be stored back in memory, it is loaded into the appropriate MBR and gated out onto the XBUS D00–35, DATA PAR lines.

11. At this time the MBox asserts XBUS DATA VALID A/B to signal the MF20 to continue.

12. When the MF20 receives DATA VALID, it gates in the data and initiates a MOS write sequence to store the word in the MOS storage array.

13. At this time, the cycle ends with the MF20 in its initial state ready to execute a new command.

3.2.6.4 Diagnostic Cycle Dialog – When the EBox decodes an XBus diagnostic function instruction (BLKO PI, E), it initiates the following XBus sequence.

1. It retrieves the contents of the location specified by E and instructs the MBox to gate this 36-bit word onto XBUS D00–35.

2. Then the MBox asserts the XBUS DIAG signal to initiate a diagnostic cycle.

3. On receiving the DIAG signal, the selected MF20 (specified by XBUS D00–04) stores the 36 bits of data in its port data buffer where the bits are used to start the control logic signals as specified by the particular function to be performed (XBUS D31–35).

4. After executing the specified function, the MF20's port data buffer is loaded with the pertinent status/error information and gated out onto the XBUS D00–35 lines.

5. After the MBox ends four XBUS CLK INT periods after asserting XBUS DIAG, the MBox strobes in the data and stores it at the address E+1.

6. At this point the diagnostic cycle ends and the MF20 returns to its initial state, ready to accept a new command.

3.2.6.5 Diagnostic Function Descriptions – In the following paragraphs, each diagnostic function is described in general, with MACRO 10 examples included to show how it could be used by the programmer. Figure 3-9 describes the general format of the diagnostic function instructions.

Tables 3-4 through 3-14 in the following paragraphs provide a detailed description of the bit encoding for each of the eleven functions. Each table is separated into two groups. The first group describes the bit encoding for E, the control information sent to the MF20; and the second group describes the E + 1 bit encoding, the status information returned from the MF20. Column 1 lists the bit position number, column 2 lists the logic print numbers where the function is implemented, and column 3 gives a short statement of the purpose of the bit. Since the diagnostic programs detect most hardware faults using diagnostic functions, the information in column 2 will be very valuable to the service engineer to key the software to the physical hardware function.

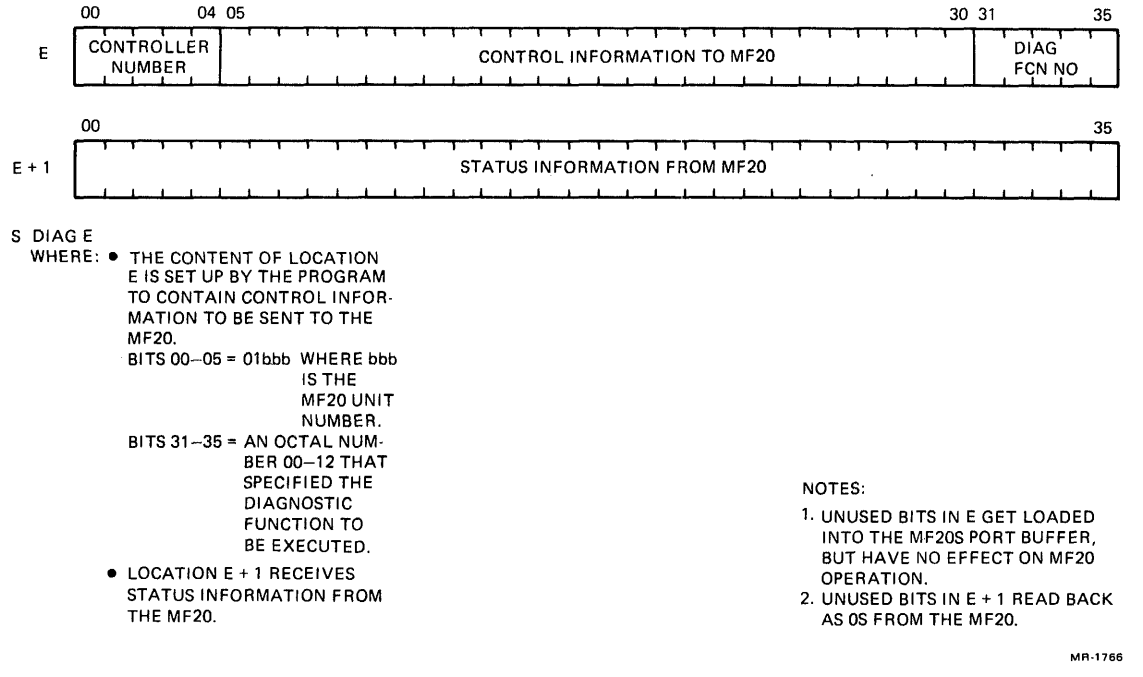


Figure 3-9 Diagnostic Function Format

Diagnostic Function 0 (Table 3-4)
This function permits starting an error-clear signal that clears all MF20 error flags and retrieving 36 bits of error/status information. The following example illustrates its use.

NOTE
All examples assume MF20 number 0.

```
HRLZI 0,210000          ;AC0 = 210000,,0
SETZM 1                 ;AC1 = 0
SBDIAG 0                ;execute the function 0
TLNE 1,770000           ;skip if all flags cleared
PUSHJ P,ERROR           ;go service error
continue
```

Table 3-4 Diagnostic Function 0

Bit	Print(s)	Description
[E] Control Information to MF20		
05	ADT9	Generates ADT9 CLR ERR that is used to clear MF20 error flags.
[E+1] Status Information from MF20		
00	ADT7,9	Asserted when one of the following RAM control errors occurs. BLK RAM ERR – Parity error when accessing the address response RAM.

Table 3-4 Diagnostic Function 0 (Cont)

Bit	Print(s)	Description
		TIM RAM ERR – Parity error when accessing the timing RAM. SUB RAM ERR – Parity error when accessing the spare bit RAM. A diagnostic function 2 is used to retrieve the individual error flag.
01	ADT7,9 SYN5	Asserted to indicate a RD PAR ERR that is a correctable error detected during a read cycle — this flag may be inhibited by setting the ICE bit in the spare bit RAM.
02		INCOMPLETE MEM CYCLE.
03–05	ADT7,9	A 3-bit field that indicates many types of data-related errors. 03 RD PAR ERR – Either a correctable read error or a double-bit error (DBE). 04 WR PAR ERR – A data parity error was detected during a write. 05 ADR ERR – Overall XBus address parity error — includes RQ 0–3 RD RQ, WR RQ.
06–07	CTL7	A 2-bit field hardwired to always read back as 11_2 to indicate 4-way interleave mode.
08–11	CTL4,7	ERR RQ 0–3 – A 4-bit field that indicates which words were being requested when an error was detected.
12–13	CTL4,7	A 2-bit field that indicates the type of memory request in progress when an error was detected. <12–13> = 01 Write cycle 10 Read cycle 11 Read-pause-write cycle
14–35	SYN7,9 CTL4 ADT1,9	These 22 bits indicate the XBus address being accessed when an error was detected.

Diagnostic Function 1 (Table 3-5)
Function 1 is used to enable looping back data for diagnostic testing of the MF20 data path. It also permits controlling a pair of status flip-flops that log the current state of the MF20. Ten bits of status are returned that specify controller type, loopback group enabled, and MF20 state. The following example illustrates how a program might use a function 1.

```
HRLZI 0,200000
ADDI 0,1                ;AC0 = 200000,,1
SETZM 1                 ;AC1 = 0
SBDIAG 0                ;execute the function 1
MOVS 2,1                ;set up ac2 to test it
ANDI 2,500
CAIE 2,500              ;skip if it is an MF20
JRST NOMF20             ;jump if not an MF20
continue
```

Table 3-5 Diagnostic Function 1

Bit	Print(s)	Description
[E] Control Information to MF20		
12–14	WRP4	Used to activate storage module loopback mode. <12> = 1 enable loopback <13–14> = n n = 0, 1, 2 – The group number to loopback
25–28	ADT7,9	<28> = 1 Enables bits 26–28 to set conditions within the MF20. <26–27> = 00 MF20 has powered up. 01 – All RAMS except the address response RAM have been loaded. 10 – Same as 01 but address response RAMs have been loaded to configure the MF20. 11 – Same as 10 except THGA has been run and completed its initialization tasks. <25> – Used to permit software to place the MF20 off-line. <25> = 0 Enable MF20 <25> = 1 Disable MF20
[E+1] Status Information from MF20		
08–11	WRP7	Hardwired to respond with a code of 05_8 to indicate the memory type is an MF20.
12–14	WRP7,4	Indicates the state of the loopback control signals set in the first half of the diagnostic cycle.

Diagnostic Function 2 (Table 3-6)
Function 2 permits reading the personality PROM and also provides special diagnostic control for hardware fault analysis. The following example illustrates its use to access the PROM in MF20 number 0, group 0, field 0, to test MOS chip size.

```
HRLZI 0,200030          ;AC0 = 200030,,2
ADDI 0,2
SETZM 1                 ;AC1 = 0
SBDIAG 0                ;execute the function 2
MOVS 2,1                ;mask chip size bits
ANDI 2,300
TRC 2,300               ;complement chip size bits
TRNN 2,300              ;4K chips
PUSHJ P,X4K             ;yes
TRNN 2,200              ;16K chips?
PUSHJ P,X16K            ;yes
TRNN 2,100              ;32K chips?
PUSHJ P,X32K            ;yes
X64K:                   ;must be 64K
```

Table 3-6 Diagnostic Function 2

Bit	Print(s)	Description
[E] Control Information to MF20		
09–12	CTL9 SM14	PROM select field used to select one of 12 possible PROM chips to be read. <9–10> Group select 0, 1, or 2 <11–12> Field select 00, 01, 10, or 11
13–14	CTL9 SM14	PROM word select – A 2-bit field used to select one of the first four locations in the PROM chip selected by <9–12>.
23–27	CTL4,7	When all 0s, this field indicates that the contents of bits D07–14 during the second half of the diagnostic cycle will be PROM data. When not all 0s, the bits are used to select which MF20 control signals will appear in D07–14 during the second half of the diagnostic cycle (diagnostic data).
[E+1] Status Information from MF20		
05–06	CTL4,7	ERR WD 2,1 – This 2-bit field specifies which word in the quadword group caused an error.
07–14	CTL4,7	This 8-bit field displays either PROM data or diagnostic data depending on the setting of D23–27 during the first half of the diagnostic cycle.
20–27	ADT2,9	This 8-bit field displays the contents of the MOS ADR 0–7 register during single-step diagnostic operations.

Table 3-6 Diagnostic Function 2 (Cont)

Bit	Print(s)	Description
28–30	ADT7,9	This 3-bit field displays the type of control RAM error flagged by bit 00 in a diagnostic function 0. 28 = 1 Timing RAM error 29 = 1 Spare bit RAM error 30 = 1 Address response RAM error

Diagnostic Function 3 (Table 3-7)
Function 3 is used to load the required bit patterns into the fixed value RAMs. The following example illustrates how a diagnostic might write and read back a RAM location for testing 1s.

```
HRLZI 0,200374          ;AC0 = 200374,,3
ADDI 0,3
SBDIAG 0                ;execute the function 3
HRLZI 0,200000          ;AC0 = 200000,,3
ADDI 0,3
SETZM 1                 ;AC 1 = 0
SBDIAG 0                ;read back loc. 000
TLC 1,360               ;complement RAM data
TLNE 1,360              ;skip if data all 1s
PUSHJ P,RAMERR          ;go service error
continue
```

Table 3-7 Diagnostic Function 3

Bit	Print(s)	Description
[E] Control Information to MF20		
10–15	CTL4,5	Used by the program to load or access the fixed value RAMs as follows. 14 = 1 Enable loading ACKN RAM (bit 10) 15 = 1 Enable loading DATA VALID RAM (11–13) 10 PACKN EN bit 11 Set DATA VALID bit 12–13 P RD ADR 34–35 bits
20–27	CTL4,5,7	Used by the program to establish the correct RAM addresses when loading or accessing the fixed value RAMs.
[E+1] Status Information from MF20		
10–13	CTL4,5,7	This 4-bit field displays the contents of the fixed value RAM address specified by D20–27 during the first half of the diagnostic cycle. 10 CTL4 PACK EN 1 11 CTL5 SET DATA VALID RAM H 12–13 CTL5 P RD ADR 34–35 H

Diagnostic Function 4 (Table 3-8)
Function 4 permits testing the MF20 control logic using single-step operations. It also allows the program to set up and monitor the refresh control logic. This is the function used by the diagnostic program to start port loopback mode. The following example would be used to set up the refresh interval.

```
HRLI 0,200000          ;AC0 = 200000,,021644
HRRI 0,021644
SBDIAG 0               ;execute function 4
                       ;set interval = 29
```

Table 3-8 Diagnostic Function 4

Bit	Print(s)	Description
[E] Control Information to MF20		
05	WRP0	Enables port data loopback mode of operation in the selected MF20.
09–12	CTL2,3 ADT5	Provides program control of MF20 clocking during single-step diagnostic operations. 09–CTL2 SIM A PHS COM 10–CTL2 SIM B PHS COM 11–CTL3 P DATA VALID IN 12–ADT5 REFRESH NOW
14–15	ADT6 CTL2	Used to enable single-step mode and gated clocks. 14 ADT6 CLK GO 15 CTL2 SINGLE STEP
20	ADT5	Enables activating the refresh logic timing chain by asserting the signal ADT5 REFRESH NOW L.
21	ADT3	Enables refresh logic.
22	ADT3	Enables loading refresh interval latches.
24–30	ADT3	This 7-bit field permits the program to establish the refresh interval. It is loaded into the refresh interval latches if bit 22 = 1.
[E+1] Status Information from MF20		
09–10	CTL7	Used to display the states of the simulated A and B phase clocks set up during the first half of the diagnostic cycle.
15	ADT9	Used to display the state of the single-step mode control signal. 1 = On, 0 = Off

Table 3-8 Diagnostic Function 4 (Cont)

Bit	Print(s)	Description
21	ADT9	Used to display the state of the refresh control logic. 1 = On, 0 = Off
23–30	ADT9	This 8-bit field displays the current contents of the refresh interval latches.

Diagnostic Function 5 (Table 3-9)
A function 5 would be used in conjunction with a function 4 to allow the diagnostic program to single-step the MF20 control logic. After stepping to the desired test point, the function 5 returns the state of the row address strobe (RAS) signals. The following example tests to verify that all RAS signals are inactive.

```
HRLZI 0,200000          ;AC0 = 200000,,5
ADDI 0,5
SETZM 1                 ;AC1 = 0
SBDIAG 0                ;execute function 5
TLNE 1,7400             ;skip if all RAS = 0
PUSHJ P,RASERR          ;jump if any are on
continue
```

Table 3-9 Diagnostic Function 5

Bit	Print(s)	Description
[E] Control Information to MF20		
06–13	CTL1	An 8-bit field used during diagnostic single-step operations to activate the MF20 control and timing logic. 06 CT01 DIAG START A 07 CT01 DIAG START B 08–11 CT01 P RQ 0–3 12 CT01 P RD RQ 13 CT01 P WR R0
15	CTL7	Enables generating CTL7 DIAG CYC RQ HLD to hold control information during single-step diagnostic mode.
20–27	SYN7	This 8-bit field is used to provide the address to the address response RAM during single-step diagnostic control operation.
29–30	CTL1	This 2-bit field selects which word in a quadword group is being referenced during the current step of a diagnostic single-step operation.
[E+1] Status Information from MF20		
24–27	ADT6,9	This 4-bit field is used to display the state of the row address strobe (RAS) signals ADT6 MOS RAS 0–3.

Diagnostic Function 6 (Table 3-10)
Like functions 4 and 5, this function is used strictly for hardware diagnosis. It provides seven diagnostic functions that permit many diagnostic operations to be executed under program control. The following example illustrates how the diagnostic might verify the ECC complement register.

```
HRLI 0,207774          ;AC0 = 203774,,2006
HRRI 0,2006
SETZM 1                ;AC1 = 0
SBDIAG 0               ;execute function 6
TLC 1,3770             ;complement ECC data
TLNE 1,377             ;skip if ECC = 1s
PUSHJ P,ECCERR         ;jump if not all 1s
continue
```

Table 3-10 Diagnostic Function 6

Bit	Print(s)	Description
[E] Control Information to MF20		
07–14	WRP7 SYN2	An 8-bit field used to set up the diagnostic data used to control SYN2 M TO CHK ECC 32, 16, 8, 4, 2, 1, PAR to be used as specified by the diagnostic subfunction specified in D25–27.
25–27	WRP7	Specifies one of eight subfunctions performed by a diagnostic function 6. 25–27 = 0 Read the ECC register on WRP7. 25–27 = 1 Read the syndrome buffer register on the SYN board. 25–27 = 2 Select diagnostic bits 07–13 in place of MOS bits 36–42, force 0s on 00–35 run a correction pass, and return 00–35. 25–27 = 3 XNU. 25–27 = 4 Write the ECC complement register. If D15 = 1, then read it back. 25–27 = 5 Write the ECC complement register, then enable it to be sent to memory in place of D36–42 on the next write cycle. 25–27 = 6 Read the output of the D36–42 mixer. 25–27 = 7 Enable latching of D36–42 mixer after next write.
[E+1] Status Information from MF20		
07–14	SYN9	This 8-bit field displays diagnostic data as a function of the subfunction specified by D25–27 in the first half of the diagnostic cycle.
24–27	ADT5,9	A 4-bit field that displays the state of the column address strobe (CAS) signals ADT5 MOS CAS 0–3.

Diagnostic Function 7 (Table 3-11)
This function permits controlling the bit substitution RAM. It also returns the state of the MOS write enable signals during single-step diagnostic operations. The following example loads location 021_8 in the bit substitution RAM to specify bit position 15_8 and inhibit reporting errors that can be corrected.

```
HRLI 0,200674          ;AC0 = 200674,,2407
HRRI 0,002407
SBDIAG 0               ;execute function 7
continue
```

Table 3-11 Diagnostic Function 7

Bit	Print(s)	Description
[E] Control Information to MF20		
07–14	WRP8	This 8-bit field specifies the information to be written into the spare bit RAM. 07–12 Bit position number. 13 Flag to inhibit reporting correctable errors (ICE bit). 14 Parity bit (odd).
15		When set, enables write to the spare bit RAM.
21–27		An 8-bit field that holds the address of the spare bit RAM location to be accessed.
[E+1] Status Information from MF20		
07–14	WRP8	An 8-bit field that displays the contents of the addressed spare bit RAM location.
24–27	ADT5,9	A 4-bit field that displays the current state of the write enable signals to the MOS chips ADT5 MOS WE 0–3.

Diagnostic Function 10 (Table 3-12)
Function 10 is primarily concerned with setting up and monitoring the power supply margin control logic. It is also used to clear the dc bad flip-flop on initial power-up, which lights a LED on the MF20 backplane. The following example tests the state of the power supply flag and clears it if it is set.

```
HRLZI 0,200000         ;AC0 = 200000,,10
ADDI 0,10
SETZM 1                ;AC1 = 0
SBDIAG 0               ;read flag
TLNE 1,400             ;skip if off
PUSHJ P,OK
IORI 0,400             ;AC0 = 200000,,410
SBDIAG 0               ;clear it - light LED
```

Table 3-12 Diagnostic Function 10

Bit	Print(s)	Description
[E] Control Information to MF20		
07–10	SYNA	A 4-bit field that specifies the direction of voltage margining. 1 = Margin high, 0 = Margin low 07 + 12 V MARG 08 + 5 V MARG 09 –2 V MARG 10 –5.2 V MARG
11–14	SYNA	A 4-bit field that enables voltage margining. 11 +12 MARG EN 12 +5 MARG EN 13 –2 MARG EN 14 –5.2 MARG EN
15	SYNA	Enables loading D07–14.
26	SYN9	Inhibits ECC correction when set.
27	SYNA	Enables clearing the DC CHK PWR BAD flip-flop, which also lights a LED on the backplane.
[E+1] Status Information from MF20		
07–10	SYNA	Displays the state of the margin control direction flip-flops set up during the first half.
11–14	SYNA	Displays the state of the margin control enable flip-flops set up during the first half.
26	SYN9	Displays the state of the ECC correction disable flip-flop.
	SYNA	Displays the state of the DC CHK PWR BAD flip-flop.

Diagnostic Function 11 (Table 3-13)
This function is used to set up and monitor the timing RAM. The following example illustrates how the diagnostic might use it to test the RAM.

```
HRLI 0,200000          ;AC0 = 200000,,777411
HRRI 0,777411
SBDIAG 0               ;execute function 11
HRRI 0,600011          ;AC0 = 200000,,600011
SETZM 1                ;AC1 = 0
SBDIAG 0               ;read back loc. 003
TLC 1,077400           ;complement RAM data
TLNE 1,77400           ;skip if data was 1s
PUSHJ P,RAMERR         ;jump if not
continue
```

Table 3-13 Diagnostic Function 11

Bit	Print(s)	Description
[E] Control Information to MF20		
13–19	ADT4	A 7-bit field used to specify a timing RAM address to be accessed.
20	ADT4	Enables writing D21-27 into the timing RAM location addressed by D13–19.
21–27	ADT4	A 7-bit field that contains the data to be written into the timing RAM. 21 RAS 22 CAS 23 RAM PAR 24 WE 25 ADR 2nd HALF 26 DATA RDY 27 RAM BUSY CLR The bit patterns loaded into the timing RAM control the MOS read/write cycle timing when accessing the MF20.
[E+1] Status Information from MF20		
21–27	ADT4	A 7-bit field that displays the contents of the addressed timing RAM location.
29	ADT2,9	This bit is activated by a backplane jumper to specify relative MOS chip size.
30	ADT5,9	This bit displays the state of the signal DESELECT CYC EN.

Diagnostic Function 12 (Table 3-14)
This function is used to set up and monitor the address response RAM. The following example illustrates how the program would set the deselect bit in RAM location 5.

```
HRLI 0,201014          ;AC0 = 201014,,2412
HRRI 0,2412
SBDIAG 0               ;execute function 12
```

Table 3-14 Diagnostic Function 12

Bit	Print(s)	Description
[E] Control Information to MF20		
08–14	SYN7	An 8-bit field that contains the data to be loaded into the address response RAM. 8 BLK ADR PAR set to establish odd parity on D8–14. 9 TYPE SELECT used to specify one of two possible MOS chip sizes (16 K words or 4 K words, 64 K words, or 16 K words and so on). 10–11 Group number 0–2 12–13 Block number 0–3 14 BOX SELECT set to inhibit response to specific XBus addresses.
20–27	SYN7	An 8-bit field that contains the address of the address response RAM location being accessed.
[E+1] Status Information from MF20		
08–14	SYN7	An 8-bit field that displays the contents of the location being accessed in the address response RAM.

3.2.7 The Storage Bus

The storage bus (SBus) connects the MA20 internal memory to the MBox. The memory system consists of one to four controllers interfaced to the SBus. Each controller connects to and controls two or four storage modules. The SBus consists of a 36-bit wide data path and its associated control and diagnostic lines. Table 3-15 describes each SBus line. Figure 3-10 illustrates the SBus configuration and Figure 3-11 shows a functional operation flow.

Table 3-15 SBus Line Descriptions

Signal Line	Description
SBus data (SBUS D00–35)	Transfers data words to the MBox during a read cycle and the read portion of a read/modify/write cycle. Also, the lines transfer data words to the MA20 during a write cycle and the write portion of a read/modify/write cycle.
SBus data parity (SBUS DATA PAR)	Transfers odd parity as computed on the associated data words during read/write cycles. The MA20 does not compute data parity, only stores it with the associated word.
SBus address (SBUS ADR 14–35)	Designates the specific address (physical location) to read/write in the selected storage module. Bits 14–33 address a quadword; bits 34–35 address the first word to be accessed within the quadword.
SBus address parity (SBUS ADR PAR)	Transfers odd parity with the read/write cycle address. The MA20 checks for odd parity. Should an address parity error be detected, the read/write currents in the storage module are inhibited.
SBus clock (SBUS CLK EXT)	This 125 ns square wave drives the internal MA20 clock generator, which in turn provides phased (deskewed) 125 ns and 62 ns clock signals. The phase A clocks are generated on the fall time of the external clock. Equivalent phase B clocks are generated in the control logic on the subsequent rise of the external clock.
SBus read request (SBUS RD RQ)	Asserted when the MBox is requesting a memory read cycle. The signal is applied to the MA20 request and address latches, which initiate a read request to the addressed storage module. With both SBUS RD RQ and SBUS WR RQ asserted, the MBox is requesting a read/modify/write cycle from the addressed storage module.
SBus write request (SBUS WR RQ)	Asserted when the MBox is requesting a memory write cycle. The signal is applied to the MA20 request and address latches, which initiate a write request to the addressed storage module. With both SBUS RD RQ and SBUS WR RQ asserted, the MBox is requesting a read/modify/write cycle from the addressed storage module.
SBus start (SBUS START A)	Asserted by the MBox on its phase A clock. START A is applied to the MA20 bus and cycle control logic, which initiates a request for a memory cycle from the addressed storage module. Memory cycle type is determined by the SBUS RD RQ/WR RQ line asserted.

Table 3-15 SBus Line Descriptions (Cont)

Signal Line	Description
SBus start B (SBUS START B)	Identical to SBUS START A except it is asserted by the MBox on its phase B clock.
SBus word requests (SBUS RQ 0–3)	The four request lines (0–3), in conjunction with the operational mode and starting address (SBUS ADR 34–35), determine MA20 bus selection and data word accessing.
SBus acknowledge A (SBUS ACKN A)	The address acknowledge as returned from the storage module, indicating that the storage module has accepted the memory cycle requested. ACKN A is returned to the MBox on the phase A clock.
SBus acknowledge B (SBUS ACKN B)	Identical to SBUS ACKN A, except that it is returned to the MBox coincident with the phase B clock.
SBus data valid A (SBUS DATA VALID A)	Transfers an equivalent memory-generated read restart signal from the MA20 in sync with the phase A clock. When true, it indicates that data and parity are valid and have been transferred to the MBox during a read cycle or read portion of a read/modify/write cycle. Only during the write portion of a read/modify/write cycle is DATA VALID sent to the MA20. In this case, it represents an equivalent write restart signal that initiates the write portion of the cycle in the MA20.
SBus data valid B (SBUS DATA VALID B)	DATA VALID B is functionally identical to SBUS DATA VALID A, except that it is in sync with the phase B clock.
SBus error (SBUS ERROR)	Indicates detection of an incomplete memory request.
SBus address error (SBUS ADR ERR)	Asserted on detection of an address parity error.
SBus diagnostic (SBUS DIAG)	DIAG is used only in diagnostic mode. When true, DIAG initiates a diagnostic operation in the MA20. The particular diagnostic function executed is determined by the diagnostic information placed on the SBus.

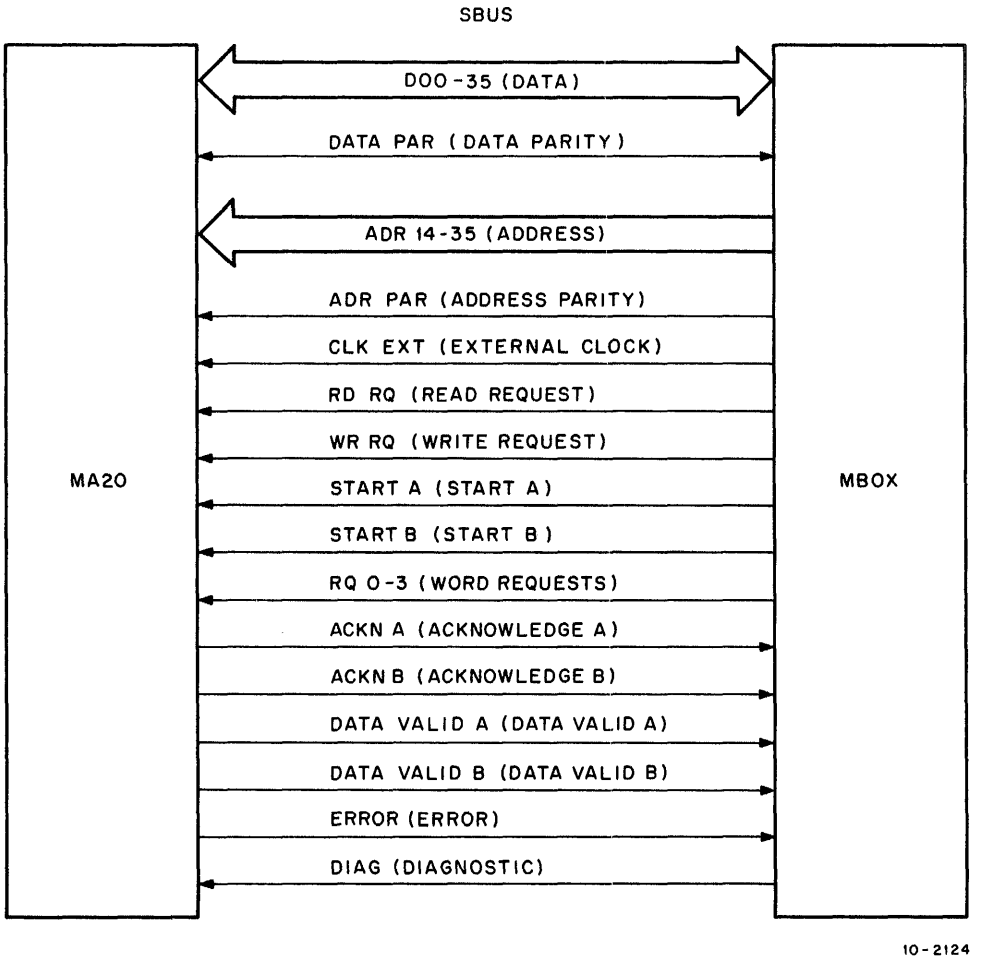


Figure 3-10 SBus Configuration

Core requests to read or write main memory are issued by the MBox core cycle control in response to a start signal and appropriate request qualifiers from the cache cycle control. All control signals, address, and data are transferred between the MBox and the memory system through the SBus. The cache cycle control can initiate a core cycle to read up to four words at a time. Once the core cycle control is set up by the cache cycle control, the core cycle control will execute the requested operation to completion, independently.

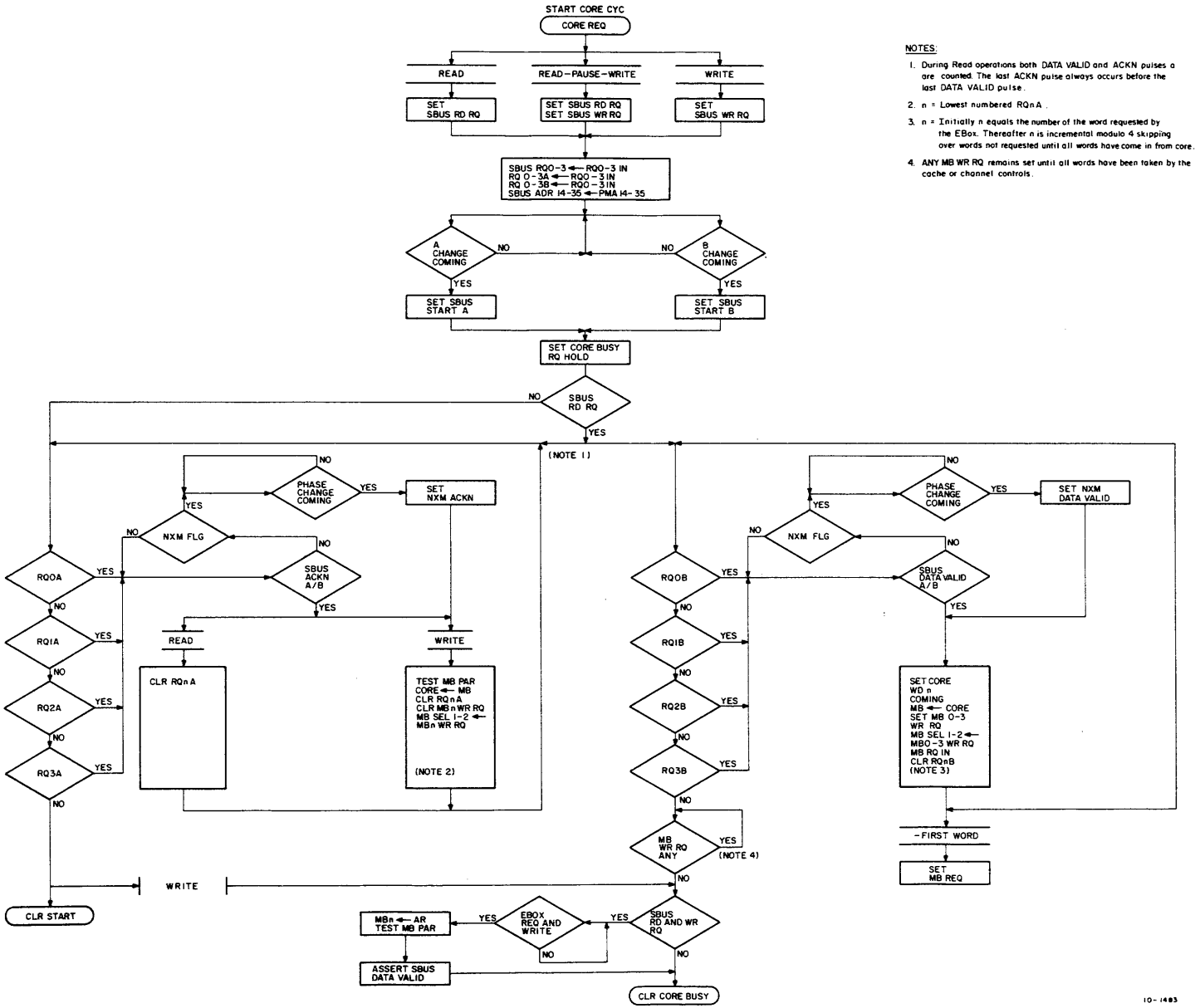


Figure 3-11 Core Cycle Control

3.2.8 The EBus

The EBus connects the EBox to the DECsystem-10/DECSYSTEM-20, KL-based front-end subsystem controller (DTE20), and the mass storage controllers (RH20). The EBus transfers both control information and data over the 36-bit wide data path and associated control lines (Figure 3-12). The EBus is controlled by the EBox during input/output instruction execution or by the priority interrupt system during interrupt handling service.

Each EBus controller is assigned a unique device select code used to address a particular controller. This device code is hardwired on the backplane. In addition, each controller is assigned a physical number that allows the EBox to identify an interrupting controller.

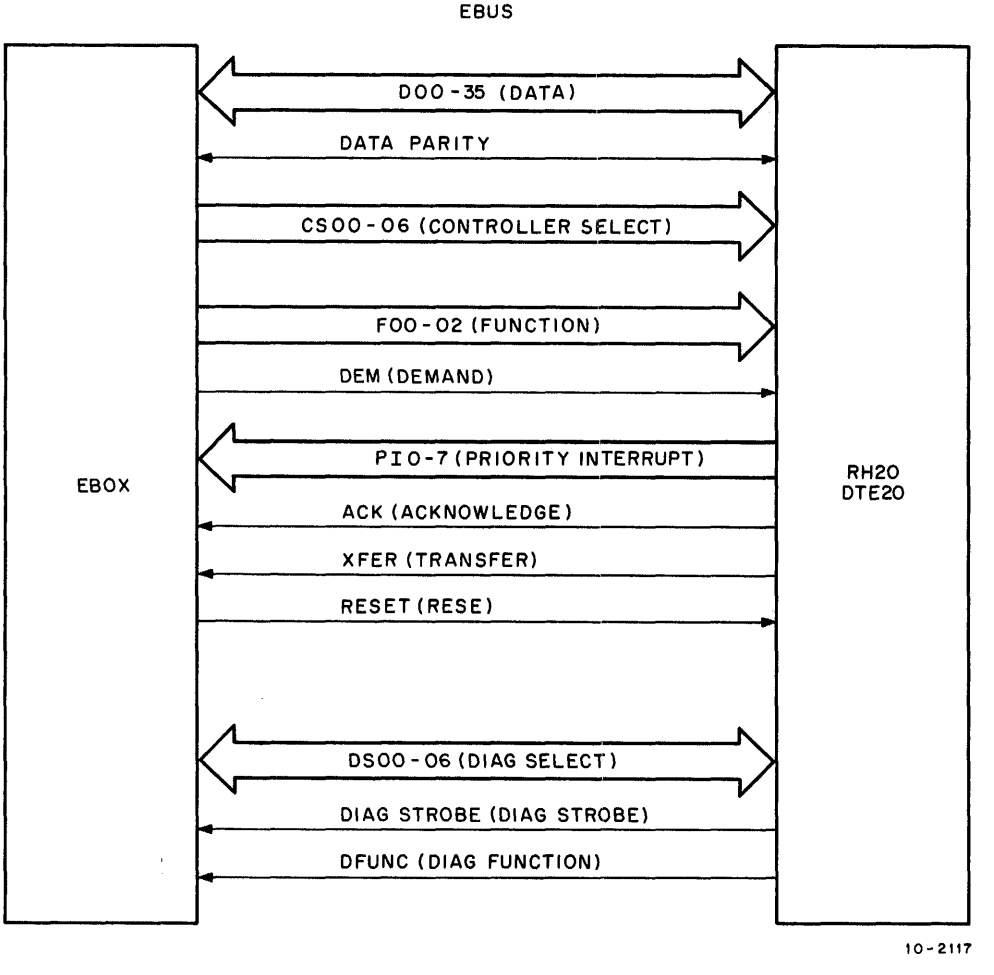


Figure 3-12 EBus Configuration

The DTE20 is the communication link between the main processor and the PDP-11 front-end processor. It is designed to facilitate implementation of the following console functions.

1. Deposit into memory
2. Examine memory
3. Direct memory access (DMA) type byte transfer operations between KL10 memory and PDP-11 memory or any PDP-11 NPR-type device
4. Diagnostic operations for getting processor status and diagnostic information and for controlling the processor from the console

The RH20 provides a programmable data link between the MBox channel controller and the secondary storage subsystem (mass storage drives). Its functions include:

1. Execute nondata transfer commands for setting up a mass storage drive
2. Execute data transfer commands to transfer data over the CBus in cooperation with the channel control logic of the MBox

The back panel in which the controllers are housed is hardwired in such a way that each controller is associated with its own physical number causing the controllers to be module-slot dependent. The RH20s are assigned physical numbers 0 and 1; the DTE20 is assigned physical number 10. This design facilitates replacement without having to rewire the physical number on the controller.

The following paragraphs summarize how output, input, and interrupt operations are performed over the EBus. Table 3-16 shows the EBus line descriptions.

Table 3-16 EBus Line Descriptions

Signal Line	Description
Data Transfers	
Data lines (D00-34)	Bidirectional lines used to transfer data control and diagnostic information between the EBox and the controllers. Controller select (CS00–06) selects the needed controller for a data transfer. Each controller has a unique select code hardwired on the backplane.
Function (F00–02)	Specifies the type of data or control transfer that is to take place. The command coding is listed below.

Data Transfer Command Coding

F00	F01	F02	Definition
0	0	0	Control out (CONO)
0	0	1	Control in (CONI)
0	1	0	Data out (DATAO)
0	1	1	Data in (DATAI)

Table 3-16 EBus Line Descriptions (Cont)

Signal Line	Description
Demand (DEM)	Causes the addressed controller to monitor and decode the CS and F lines. After implementing the specified function, TRANSFER and ACKNOWLEDGE are asserted as a response along with data placed on or removed from the EBus.
Acknowledge (ACK)	Required to inform the DIB20 not to respond to the current operation. If the DIB does not see ACK some time after DEM is asserted, it will try to execute the transfer.
Transfer (XFER)	Asserted by the selected controller when it is ready to execute the required function as specified in F00–02.
Priority Transfers	
00–06	During normal input/output operation these lines reflect the device code of a particular controller.
Controller select (CS04–06)	During interrupt arbitration, these lines represent the octal encoding of the interrupting channel with a range of 0–7.
Controller select (CS00–03)	Specifies the controller the EBox will honor during the interrupt sequence. The code corresponds to the hardwired physical device number of the interrupting controller.
Function (F00–02)	Supplies the two functions specified below during an interrupt sequence. The first is coded 4_8 and directs those controllers addressed by the channel number in CS04–06 to assert their physical controller number on the data lines on sensing DEM. The second is coded 5_8 and specifies that an interrupting controller has been selected as determined by the physical controller number coded in CS00–03.
Function (F00–02)	**Priority Transfer Command Coding** **F00 F01 F02 Definition** 1 0 0 Priority interrupt (PI) served 1 0 1 Priority interrupt (PI) address in
Acknowledge (ACK)	Identical to data transfer function.
Transfer (XFER)	Asserted to indicate to the EBox that the interrupting controller has placed a special function on the data lines.

3.2.8.1 CONO/DATAO Operation – This output operation transfers data (DATAO) or control (CONO) words from the EBox to the controllers. The EBox/EBus control places the appropriate device select code on CS00–06, the data or control word on D00–35, and encodes F00–02 as 0 or 2_8 (CONO/DATAO, respectively). The EBus control then allows a line settling and decode delay time and asserts DEM. The controllers compare the CS lines with their hardwired device code. If a true comparison results, the controller asserts ACK and simultaneously decodes the F lines. It then asserts XFER after strobing the data lines.

After receiving XFER, the EBus control resets DEM. The trailing edge of DEM causes resetting of ACK and XFER in the controller, terminating the output transfer. The EBus control must wait for a specified time prior to changing the select, function, and data lines. Figure 3-13 provides a functional timing diagram for the CONO/DATAO operation.

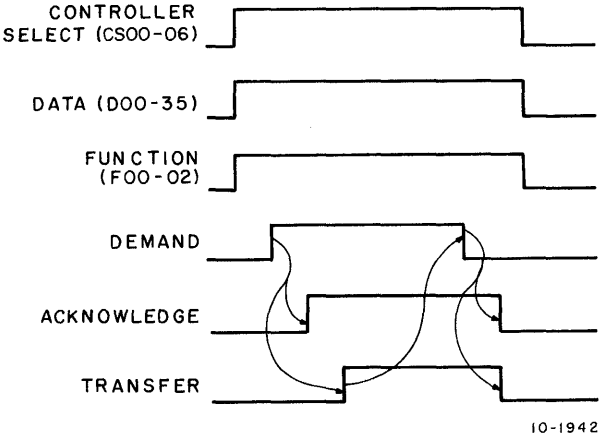


Figure 3-13 CONO/DATAO Operation Sequence

3.2.8.2 CONI/DATAI Operation – This input operation transfers control (CONI) and data (DATAI) words from a controller to the EBox. The EBus control places the appropriate device select code on CS00–06, allows a line settling and decode delay time, and asserts DEM. The controller compares the CS lines with the hardwired device code. If a true comparison results, the controller asserts ACK and simultaneously decodes the F lines. It then asserts XFER at the same time it places the input data on D00–35.

After detecting XFER, the EBus control allows a data line skew time, strobes the data lines, and resets DEM. The trailing edge of DEM causes resetting of ACK and XFER in the controller, terminating the output transfer. As in the output operation, the EBox must wait for a specified time prior to changing the select and function lines. Figure 3-14 provides a functional timing diagram of the CONI/DATAI operation.

3.2.8.3 Interrupt Operation – A controller requests an interrupt whenever it needs service from the EBox on one of the eight priority interrupt lines (PI0-7). When the conditions for initiating an interrupt request are met, the controller asserts its assigned interrupt line (PI0-7). The EBox PI request control detects the interrupt and determines the interrupt priority. When ready, the PA logic asserts the interrupt channel number or CS04–06 (e.g., CS04–06 = 110 for priority channel 6). It then encodes 4_8 (PI served) on F00–02 and after a time delay, asserts DEM.

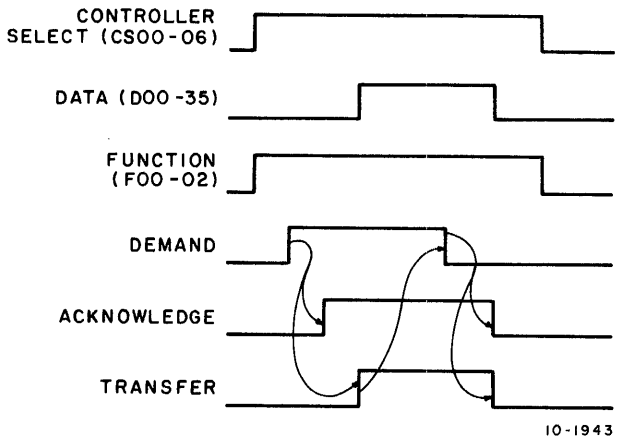


Figure 3-14 CONI/DATAI Operation Sequence

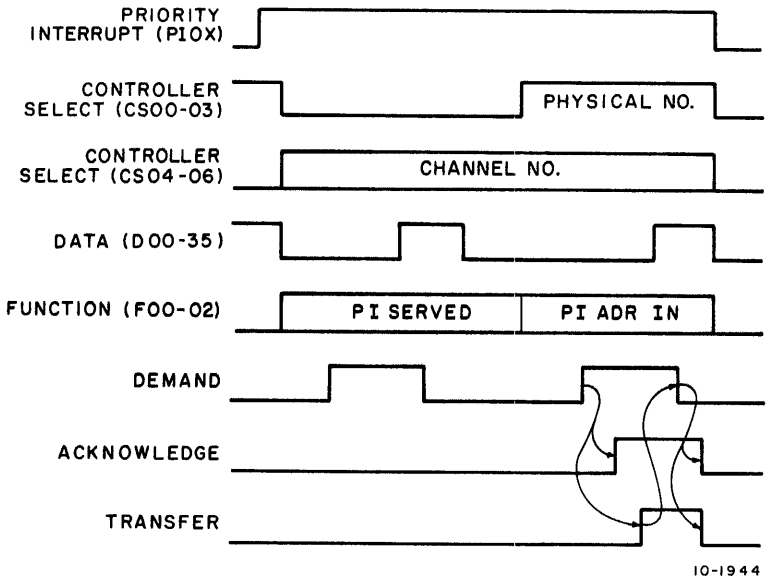


Figure 3-15 Priority Interrupt Operation Sequence

Each controller decodes the F lines and compares the CS lines with the channel on which it is interrupting. If a true comparison results (several controllers may have a true comparison), the controller will assert a data line corresponding to its physical controller number. For example, controller 0 and controller 15 (physical 0 and 15) will assert bits 1 and 15 on the data lines. No ACK or XFER is asserted by any controller. The PA logic waits at least 400 ns, strobes the data lines, and resets DEM. The data stays valid until DEM resets. The EBox will not change the CS or F lines for 150 ns after resetting DEM.

After determining which physical controller's interrupt request to serve, the PA logic places the interrupt channel number on CS00–03 (for example, CS00–03 = 1010 for physical controller 10). It then encodes 5_8 (PI address in) on F00–02, waits 200 ns, and asserts DEM. Each controller decodes the F lines and compares the interrupt channel number and physical number with its own. If a true comparison results (only one controller will have a true comparison), the controller asserts ACK.

If the controller is a DTE20, it places the interrupt function index (if any) and interrupt address on the data lines and asserts XFER. The EBox strobes the data lines and resets DEM. The resetting of DEM causes resetting of ACK and XFER in the controller. Figure 3-15 provides a functional timing diagram of the interrupt operation.

3.2.8.4 Diagnostic Bus – All KL10 diagnostic functions are performed over the diagnostic bus portion of the EBus. The diagnostic lines are described in Table 3-17.

Table 3-17 Diagnostic Line Descriptions

Signal Line	Description
Diagnostic select (DS00–06)	Transfers encoded diagnostic functions to the KL10. These lines can be read by the PDP-11 at any time, even while the rest of the EBus is active for other devices.
Diagnostic strobe (DIAG STROBE) (Run flag, EBOX halted, ERR stop)	Asserted to indicate that the diagnostic select lines are CLOCK stable and that the indicated function should be performed.
Diagnostic function (DFUNC)	When true, causes the KL10 to disable the basic CPU status from the DS lines, switch the translation (only for DS lines) to convert TTL to ECL, and put the EBus translator under control of DS00–01.

3.2.9 Channel Bus Description

The channel bus (CBus) is a synchronous bus that connects the integral data channel logic of the MBox to a maximum of eight RH20 mass storage controllers. The CBus is composed of a 36-bit wide data path and its associated control lines and transfers high-speed data and control information between the channel logic and RH20 controllers. Each CBus line is described in Table 3-18. Figure 3-16 illustrates the CBus configuration.

Table 3-18 CBus Line Descriptions

Signal Line	Description
Data (D00–35)	Transfers high-speed data. The lines are valid only during a data cycle (see Paragraph 3.2.9.2). The MBox will place zeros on the data lines for an RH20 (during its data cycle) whenever there is no data transfer request from that RH20.
Par left/par right	Transfers the computed parity for the left and right half words of the data lines.
Select (SEL 0–7)	Continuously selects an RH20 each 138 ns. SELECT for an RH20 defines the beginning of its four data transfer cycles (see Paragraph 3.2.9.2).
Reset (RESET)	Asserted by an RH20 during its data cycle, causing the MBox to: • Clear the data buffers associated with the selected RH20. • Reset the command list pointer associated with the selected RH20 to the initial address. • Negate all status and data lines associated with the selected RH20 after a and b are complete.
Start (START)	Asserted by an RH20 to start a transfer. START is asserted once during its data cycle and only when READY is negated.
Controller to memory (CTOM)	Asserted by an RH20 for one data cycle to indicate the transfer direction to the MBox. For a controller-to-memory transfer (CTOM), CTOM is asserted; for a memory-to-controller (not CTOM), CTOM is negated.
Ready (READY)	Asserted by the MBox (during the data cycle) after it detects START from the RH20 and after the MBox is ready for a data transfer. The MBox is required to have at least two data words from memory in its buffer before asserting READY. READY will be negated only after sensing DONE and after the MBox is prepared to start another transfer.

Table 3-18 CBus Line Descriptions (Cont)

Signal Line	Description
Request (REQUEST)	Asserted by an RH20 during its request cycle when: • One of its data buffers is full, for a CTOM transfer. • One of its data buffers is empty, for a not CTOM transfer. An RH20 will not assert REQUEST if: • READY is not asserted by the MBox. • ERROR is asserted by the MBox during the current transfer. • LAST WORD is asserted by the MBox during the current transfer. • DONE is asserted by the RH20 during the current transfer. For an input data transfer, the RH20 places data (throughout its data cycle) on the data lines, and the MBox will strobe the lines at the trailing edge of the same data cycle. For an output transfer, the above operation is reversed.
Done (DONE)	Asserted by an RH20 during its data cycle to terminate a data transfer. No further data requests will be made after DONE is asserted. The MBox, after detecting DONE, will get ready for a new transfer (empty the input data buffers, and so on). ERROR can be used to inform the RH20 that an error has been detected in the current transfer as long as READY is not negated.
Store (STORE)	Asserted by an RH20 together with DONE when: • The current transfer is terminated due to errors detected in the RH20. • The current transfer command in the RH20 specifies that STORE be sent to the MBox at the conclusion of the transfer.

Table 3-18 CBus Line Descriptions (Cont)

Signal Line	Description
Last word (LAST WORD)	Asserted by the MBox during the data cycle (for not CTOM transfer only) at the same time the last data word is sent to a controller. No further data requests will be made by the RH20 after detecting LAST WORD.
Error (ERROR)	Asserted by the MBox (during the data cycle) to inform the RH20 that the current data transfer must terminate due to an error in the MBox. On sensing ERROR, the RH20 terminates the transfer by not issuing any further requests. The RH20 also asserts DONE during a subsequent data cycle. ERROR is negated before the MBox negates READY. If ERROR is detected after READY is negated, it may be interpreted by an RH20 as an error associated with the next transfer.

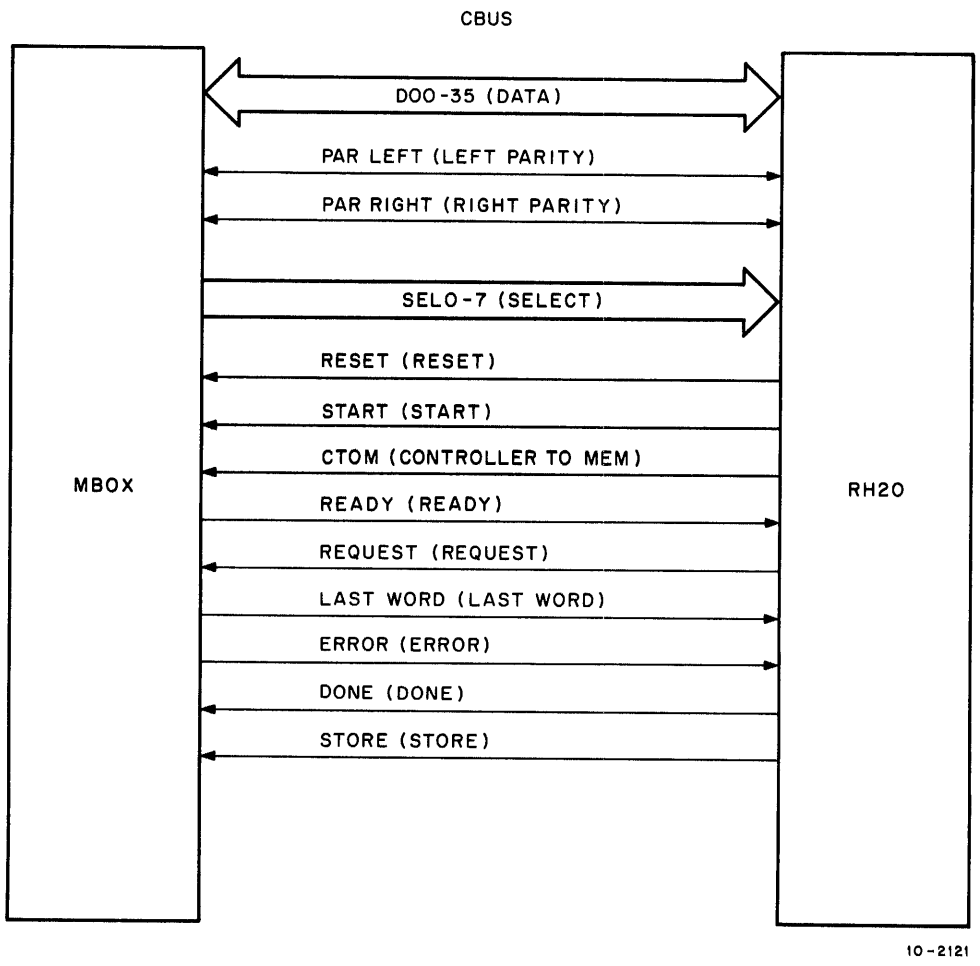


Figure 3-16 CBus Configuration

3.2.9.1 Controller Selection – The RH20 selection order is on a set priority basis using a time division multiplexing technique. Selection is of the following order: 0, 1, 2, 3, 4, 5; 0, 1, 2, 3, 6, 7. Each channel is assigned a specific time slot within a fixed repetition rate. A channel is selected during the same time slot in each of the following successive repetition cycles.

Time slot and repetition rate are determined by the 7.25 MHz EBox clock, which is distributed to the RH20 through the EBus, and to the MBox through the E/M interface. The controller select line sequence is stepped with the leading edge of the clock. Since only six out of a maximum of eight channels are selected during any one repetition cycle, the EBox clock provides a 138 μs time slot for each channel, with a repetition rate of 828 μs (that is, 138 μs time $\times$ 6 successively-selected channels).

3.2.9.2 Transfer Cycle Definitions – In addition to showing basic channel selection timing, Figure 3-17 shows timing for the four cycles (select, request, wait, and data) used by the MBox and a controller during a data transfer operation.

1. Select cycle – The select line for a particular controller is asserted through this cycle.

2. Request cycle – The selected controller will assert its request line (if data request is needed) during this cycle.

3. Wait cycle – This cycle is used by the MBox to prepare data and status for transmission. Neither data nor status is asserted during this cycle.

4. Data cycle – Data is placed on the data lines either by the MBox or the controller during this cycle. The receiver will strobe the data lines on the trailing edge of the data cycle. All control lines, except the request line, are allowed to be asserted only during this cycle.

3.2.10 DTE20

The DTE20 ten-eleven data interface serves as the interface between the KL10 main processor and the PDP-11 front-end processor. To the main processor, the DTE20 appears to be a standard EBus-compatible device controller; and to the front-end processor, the DTE20 appears to be a standard Unibus peripheral device. Therefore, either processor can access the DTE20 for transferring status, control, and data using the normal I/O instructions of the respective machines.

The DTE20 consists of bus control and interrupt logic for interfacing with both the EBus and the Unibus; a 16 $\times$ 16 RAM for storing status, control, and data information required in executing a transfer; and diagnostic control logic. The bus control logic responds to the processor-initiated dialog to load or read internal DTE registers. The processors initiate their respective bus sequences when they execute I/O instructions for the DTE. The interrupt logic is included in the DTE to allow the processors to interrupt each other in passing status information and in executing data transfers. The diagnostic control logic in the DTE operates in conjunction with the diagnostic control logic in the EBox to gain direct access to various registers and status and control bits in the EBox and the MBox. The EBus dialog is not initiated in executing diagnostic functions, but the EBus data lines may be used in transferring the data between the DTE and the EBox and MBox.

The DTE20 is capable of executing the following operations.

1. Ring -10 and -11 doorbell
2. Deposit
3. Examine
4. Byte transfer
5. Diagnostic
6. Reload 11

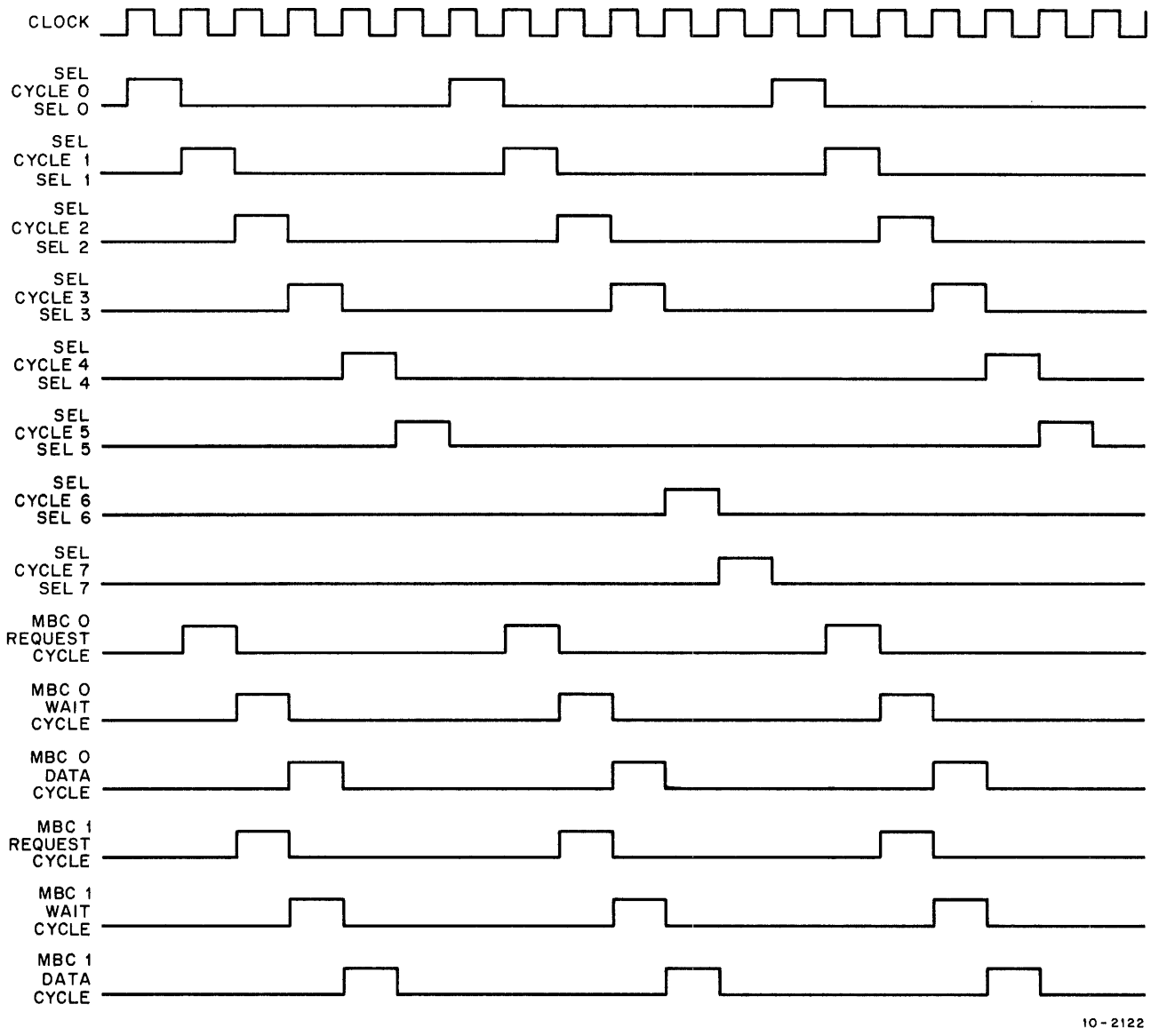


Figure 3-17 Basic RH20 Selection Timing

The doorbell function permits the processors connected to the DTE to interrupt each other on the assignable interrupt channels to report a change of status as in the case of reporting a power failure. This facility is also used to report that the byte transfer operation is done. The assignable channels are PI1–7 on the -10 side and BR4–7 on the -11 side.

The examine and deposit functions are included in the DTE20 so that the front-end processor can fetch or change any location in the physical memory (MF20) while the EBox is running or executing a halt instruction. After being initiated by the PDP-11 front-end processor, the examine and deposit functions are handled as a special PI request to the EBox. The PI request is referred to as PI0. This request will be honored and executed even when the PI system is turned off. To execute these functions, the front-end processor must assemble the address and assemble or disassemble the data in the DTE because of the difference in the length of the address and data words between the two machines.

The byte transfer function is included in the DTE20 so that the front-end and the main processors can initiate high-speed, DMA-type data transfers between each other's memories or between KL10 main memory and any NPR-type PDP-11 device. Once initiated, the DTE uses the processor's high priority interrupt facility without further intervention by the program to execute that transfer. On the -10 side, the special PI0 request is used to fetch or store a byte into -10 memory; and on the -11 side, the nonprocessor interrupt request (NPR) is used to fetch or store a byte in -11 memory or an NPR-type device. Interrupt requests PI0 and NPR are issued until the byte count is zero, at which time an interrupt request is issued on the assignable priority channel(s).

During the byte transfer operation, the DTE transfers fields of information between -11 memory and -10 memory via the EBox. On the -10 side, the fields are of variable lengths and are accessed through DTE byte pointers in the EPT. On the -11 side, fields are either 8 bits wide and are stored in consecutive bytes or are 16 bits wide and are stored in consecutive words. If the field into which the information is stored is narrower than the field from which it was read, as many of the rightmost bits as will fit are stored. If the field into which the information is being stored is wider than the field from which it was read, the information is right-justified and filled with zeros on the left.

The diagnostic facilities are included in the DTE20 so that the front-end processor can execute various diagnostic and console functions. The diagnostic part of the EBus is used in implementing these functions. In addition, if the function includes the transfer of data (as is the case for load and read functions) the EBus data lines are also used in executing the diagnostic functions. In general, the following diagnostic functions are implemented.

1. Clock control functions to start, stop, or single-step the clock; to initiate a clock burst of 1 to 255 clock ticks; or to select the desired clock source.

2. EBox control functions to start and stop (HALT and CONTINUE) the microcode and to control the decoding of some special op codes.

3. Various load functions to facilitate the following.

    a. Load DRAM in the EBox
    b. Load CRAM in the EBox
    c. Load AR in the EBox
    d. Load the MBox control functions
    e. Enable the EBus register
    f. Reset DMA
    g. Load the channel control functions

4. Various read functions to facilitate reading all diagnostic registers and RAMs of the EBox, MBox, and meters.

3.2.10.1 UNIBUS Description – The Unibus is a single, common set of signal wires that connect the PDP-11 processor, memory, and all its peripheral devices to the DTE20. Address, data, and control information are transferred over the 56 lines of the bus. Communication between two devices on the bus is in a master-slave relationship. The device in control of the bus is considered the master; the device being addressed is the slave. Communication on the bus is interlocked; that is, each control signal issued by the master device must be acknowledged by a corresponding response from the slave to complete the transfer. When no other device requires the bus, control automatically reverts to the CP.

Bus Request Levels

Each device uses one of two levels for requesting bus control: nonprocessor requests (NPR) and bus requests (BR). The NPR is used when a device requires a direct memory or device access data transfer (that is, a transfer not requiring CP intervention). Normally, NPR transfers are made between the memory and a mass storage device, such as a disk drive. Because NPRs are executed between bus cycles, the CP can give up bus control while an instruction is being executed. Two bus lines are associated with the NPR priority level. The device issues its request on the NPR line; the CP responds by issuing a grant on the NPR grant (NPG) lines.

The BR level is used when a device interrupts the CP for a service requiring CP intervention. The service required may be to inform the CPU to initiate a transfer or that an error condition has occurred. Unlike NPRs, BR level interrupts are not serviced until the processor has completed executing its current instruction. Two lines are associated with each BR level. The bus request is issued on a BR line (BR7-BR4); the bus grant is issued on the corresponding bus grant line (BG7-BR4).

When a device service program is to be run, the task being performed by the CP is interrupted and the device service routine is started. After the device request is satisfied, the processor returns to its former task. Note that interrupt requests can only be made if bus control is gained through a BR priority level. The NPR level is never used for an interrupt request.

Priority Structure and Chaining

When a device requests use of the bus, the handling of that request depends on the location of that device in a two-dimensional device priority structure. Priority is controlled by the priority arbitration (PA) logic of the KD11.

The device priority structure consists of five priority levels: NPR, BR7-4. Bus requests from external devices can be made on any one of the request lines. The NPR has highest priority, BR7 is the next highest priority, and BR4 is the lowest. The PA is structured such that if two devices on different BR levels issue simultaneous requests, the PA grants the bus to the device with the highest priority. However, the lower priority device keeps its request up and will gain bus control when the higher priority device is through with the bus (providing no other higher priority device issues a BR).

In addition, the KD11 has a programmable priority, which can be set to any one of eight levels (0-7). The CP priority is dynamic and can be raised/lowered by the program running at that time. For example, the CP priority can be raised from level 4 to level 6 when the CP completes service on a BR4 device and initiates service on a BR6 device. This programmable priority design permits masking of device requests. When the CP is set to a specific level, all bus requests on that level and below are ignored.

Since there are only five priority levels, more than one device is connected to a specific request level. If more than one device makes a request at the same level, the device closest (electrically) to the KD11 has highest priority. The grant bus for the NPR level is connected to all devices on that level in a daisy-chain arrangement (chaining). When an NPG is issued, it goes to the device closest to the CP. If that drive did not make the request, it permits the NPG to pass to the next closest device. When the NPG reaches the device making the request, that device captures the grant and prevents it from passing on to any subsequent device in the chain.

Functionally, BR chaining is identical to NPR chaining. However, each BR level has its own BG chain. Therefore, the grant chain for BR7 is the BG7 line, which is chained through all devices at the BR7 level.

Device Register Organization

The actual transfer of data and status information over the Unibus is done between status, control, and data buffer registers located within the peripheral devices and their control units. All device registers are assigned addresses similar to memory. Therefore, all -11 instructions that address memory locations can become I/O instructions.

Control and status functions are assigned to the individual bits within the corresponding addressable registers. Since the register content can be controlled, setting and clearing register bits can control service operations. For example, the command to make the paper-tape reader read a frame of tape is provided by setting the reader enable bit in the control register in the device control unit. Internal device status may be loaded into the appropriate register and retrieved when a program instruction addresses that register. Depending on the function, register bits may be read/write, read-only, or write-only. The number of addressable registers in a device (and control unit) change depending on the device's function.

Peripheral device registers are assigned an address within the bus address allocations from 760000_8–777777_8 (program addresses 160000_8–177777_8); however, only 16 bits are normally provided by programs as memory reference addresses. Whenever the processor tries to reference an address between 160000_8–177777_8 (when A15–13 = 1) address bits A17–16 are forced to ones. Therefore, the 16-bit address is converted to a full 18-bit address and references a device register. Device register addresses are always even, although byte operations may address either half of the register.

Unibus Line Definitions

The Unibus consists of 56 signal lines, which may be divided into three function groups: bus control, data transfer, and miscellaneous signals. The 13 lines of the bus control group make up those signals required to gain bus control through an NPR/BR or for a priority transfer to select the next bus master while the current bus master is still in control of the bus. The 40 bidirectional lines of the data transfer group are those signals required during data transfers to or from a slave device. The miscellaneous signals are the initialization and power-fail signals required on the bus. Table 3-19 describes the bus signals within each group. Figure 3-18 illustrates the Unibus configuration.

Table 3-19 Unibus Line Descriptions

Signal Line	Description
Bus Control Group	
Bus request (BR7–BR4)	Used by peripheral devices to request control of the bus as determined by the priority structure.
Bus grant (BG7–BG4)	Processor's response to a bus request. They are asserted only at the end of an instruction execution as determined by the priority structure.
Nonprocessor request (NPR)	Is a bus request from a device to the processor for an access not needing CP intervention (that is, direct memory access).
Nonprocessor grant (NPG)	Is the processor's response to an NPR. It can occur whenever the CP is not in control of the bus (that is, between bus cycles).

Table 3-19 Unibus Line Descriptions (Cont)

Signal Line	Description
Slave acknowledge (SACK)	Asserted by a bus-requesting device after having received a grant from the processor. Bus control passes to this device when the current bus master completes its operation.
Interrupt (INTR)	Asserted by the master to initiate a program interrupt in the processor. Bus busy (BBSY) is asserted by the new master device to indicate that the bus is being used. No other device can become bus master while BBSY is asserted.
Data Transfer Group	
Address lines (A17–00)	Used by the master device to select the slave (actually a unique memory or device register address). A17–01 specifies a unique 16-bit word; A00 specifies a byte within the word.
Data Lines (D15–00)	Used to transfer information between master and slave.
Control (C01–00)	Coded by the master device to control the slave in one of the four possible data transfer operations specified below. Note that the transfer direction is always defined with respect to the master device.

Data Transfer Designation

C1	C0	Description
0	0	Data in (DATI) – A data word or byte transferred into the master from the slave.
0	1	Data in, pause (DATIP) – Used with core memory, similar to DATI except data is not restored back into the addressed memory location.
1	0	Data out (DATO) – A data word is transferred out of the master to the slave.
1	1	Data out, byte (DATOB) – Identical to DATO except a byte is transferred instead of a full word.

Parity A-B (PA, PB)	Transfer Unibus parity information. PA indicates that the transferring device is generating parity. PB contains the computed odd parity. Bit configurations for PA and PB are specified below.

Table 3-19 Unibus Line Descriptions (Cont)

Signal Line	Description
	Parity Line Definitions
	PA **PG** **Definition**
	0 0 Sender not generating parity.
	1 1 Sender generating parity, parity bit = 1.
	1 0 Sender generating parity, parity bit = 0.
	0 1 Core memory parity error, no Unibus parity checking in effect.
Master synchronization (MSYN)	Is asserted by the master to indicate to the slave that valid address and control (data in DATI) information is present on the bus.
Slave synchronization (SSYN)	Is asserted by the slave in response to MSYN from the master.
Miscellaneous Group	
Initialize (INIT)	Is asserted by the processor when the START key on the console is depressed, when a RESET instruction is executed, or when the power-fail sequence occurs.
AC line low (AC LO)	Is an anticipatory signal that initiates the power-fail trap sequence and may also be issued in peripheral devices to terminate operations in preparation for power loss.
DC line low (DC LO)	Is available from each system power supply and stays clear as long as all dc voltages are within the specified limits. If an out-of-voltage condition occurs, DC LO is asserted.

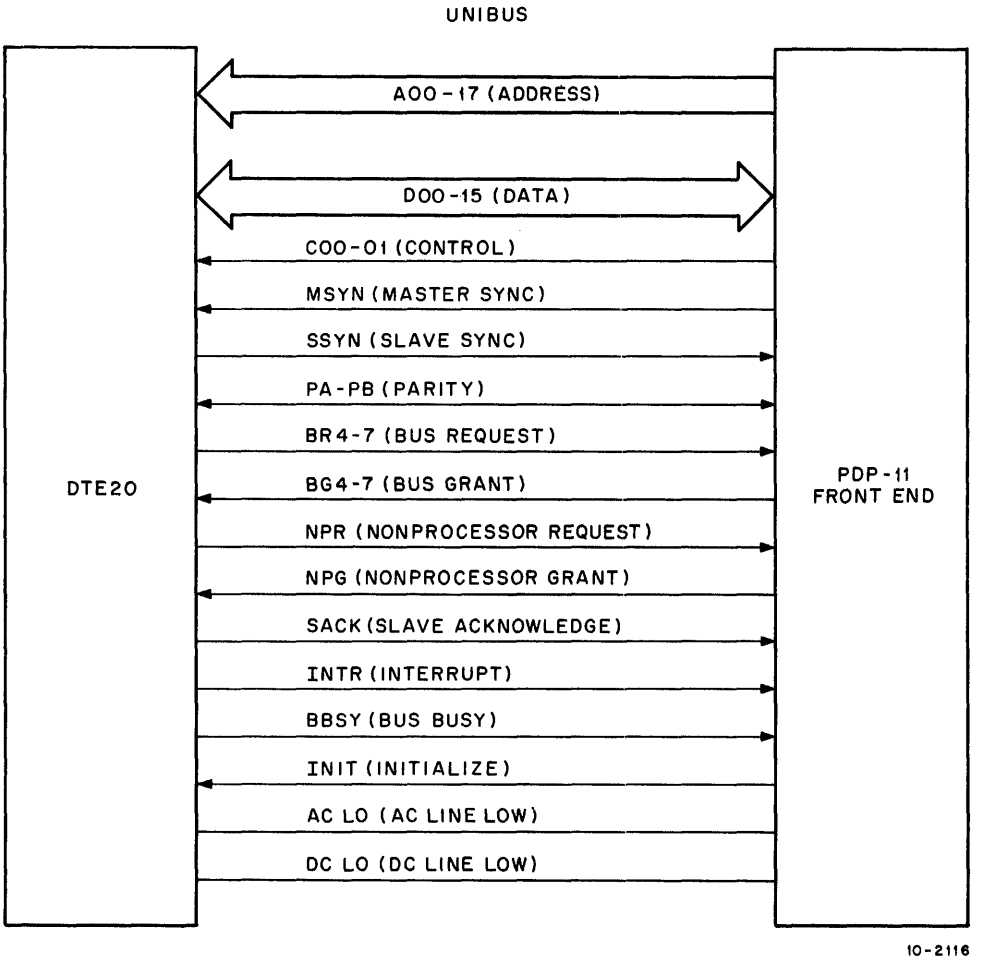


Figure 3-18 Unibus Configuration

Unibus Operation Sequencing

The following sections provide a summary description of the bus operation sequences. The timing diagrams provided in the descriptions do not reflect actual timing, only general signal relationships. If additional detail is required (such as, skew times, logic structures, and so on), refer to the appropriate -11 device maintenance manual or the Unibus section of the *Peripherals Handbook 1973-74.*

Priority Transfer – The priority transfer operation is the signal sequence that selects the next bus master. The operation does not actually transfer bus control but only selects the next bus master.

The device requiring service asserts its BR (or NPR) line. In practice, the processor may be receiving several simultaneous BR signals. These signals enter the PA of the KD11, which compares BR levels with the CP priority level and against the NPR. If the PA system determines that the request has highest priority and the SACK line is clear, the processor asserts the corresponding BG (or NPG) line. The grant is propagated through each device on the asserted BG line. The first device on the line having BR asserted acknowledges the grant by asserting SACK, blocks the grant from following devices, and clears its BR. The processor responds to SACK by clearing BG. If SACK is not received within 5–10 μs, a time-out occurs and BG is automatically cleared.

The device will keep SACK asserted until the current bus master gives up bus control by clearing its BBSY. SACK asserted prevents another device on the same priority level (or lower) from gaining bus control. Priority arbitration can be performed at the same time that a data transaction or interrupt is being serviced. While one device is using the bus, the PA is able to monitor other requests and to issue an appropriate grant.

Figure 3-19 provides a functional timing diagram for the priority transfer sequence.

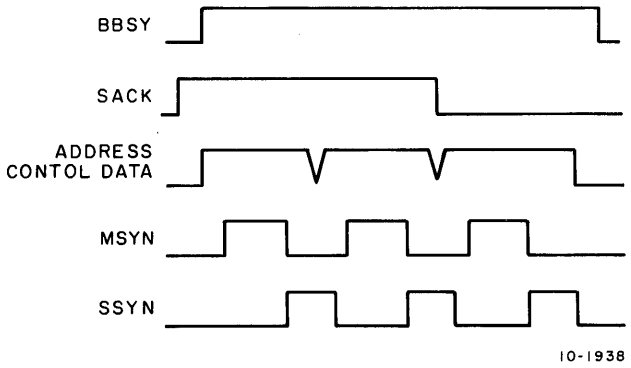


Figure 3-19 Priority Transfer Sequence

Data Transfer – The 40 bidirectional data transfer lines are used for all data transfers. In a data transfer, one device is the bus master and controls the transfer of data to or from a slave device.

Once the device has become bus master through an NPR or BR (asserting BBSY and clearing SACK), the master places the appropriate address and control information and data on the bus. The address selects the desired slave device register or memory location, the C1–0 control lines select the transfer type (in this case DATO). Following a delay to allow bus line settling and address decoding, the master asserts MSYN. MSYN is a strobe signal for the address and control lines and is always cleared before these lines are changed. The slave acknowledges MSYN by asserting SSYN, indicating that the address has been decoded and data has been strobed from the bus. The master responds to SSYN by clearing MSYN. Following a specific time delay, the master clears the address and control lines. With the clearing of MSYN the slave responds by clearing SSYN.

During multiple word (or byte) transfers the master again asserts the correctly coded address and control lines. Again, the master allows a settling and decoding time and asserts MSYN. The sequence continues as described in the previous paragraph. Note that during multiple transfers the master and slave have kept BBSY and SACK asserted. Prior to the last bus cycle, the slave clears SACK, allowing the PA to issue the next grant. When the slave has asserted SSYN (for the last bus cycle), the master again responds by clearing MSYN. As before, the master clears the address and control lines following the time delay. With the last bus transfer complete, the master clears BBSY, allowing a new master to assume bus control.

Figure 3-20 provides a functional timing diagram for the multiple DATAO/DATOB transfer.

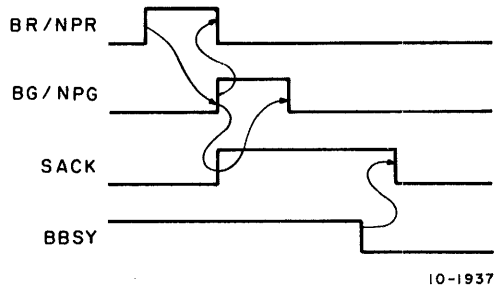


Figure 3-20 Multiple Data Transfer Sequence

The bus sequence for a DATI is essentially the same as a DATO, with the following exceptions.

1. The master asserts only address and control information.
2. The slave gates data onto the bus simultaneously with the assertion of SSYN.
3. The master allows a time delay (for skew between SSYN and the data) before clearing MSYN and strobing the data.
4. The slave clears the data when it clears SSYN.

Multiple transfers are functionally the same as the DATO (considering the exceptions). Figure 3-21 provides a functional timing diagram for the DATI bus sequence.

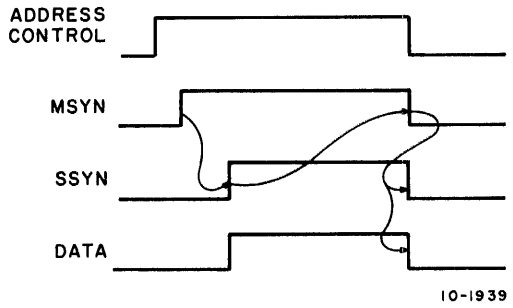


Figure 3-21 DATI Bus Sequence

Interrupt Sequence – A device may cause an interrupt operation each time it gains bus control on a BR priority level. It is usually done immediately upon becoming bus master; however, it may follow one or more data transactions on the bus.

If an interrupt operation is to be initiated and the device has control of the bus, the master (interrupting device) asserts BBSY and clears SACK. At the same time, the master asserts INTR and places a vector address on the data lines. The vector directs the CPU to a memory location that contains the starting address of the interrupt service routine.

The KD11 receives INTR and after a time delay to make sure that all bits of the interrupt vector address are valid, asserts SSYN when the address is strobed in. The bus master receives SSYN and releases the bus to the CP by clearing INTR, removing the vector from the data lines, and clearing BBSY. The CP acknowledges by clearing SSYN and stores the required breakpoint information to return to the interrupted program. The CP then enters the interrupt handling sequence. When the interrupt operation is completed, the CP retrieves the stored breakpoint information and continues the program at the point where it was interrupted.

Figure 3-22 provides a functional timing table for the interrupt sequence.

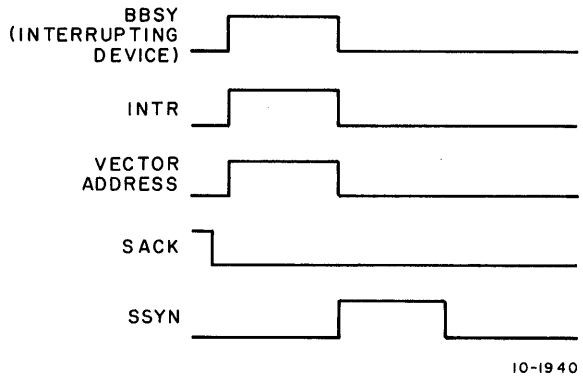


Figure 3-22 Interrupt Sequence

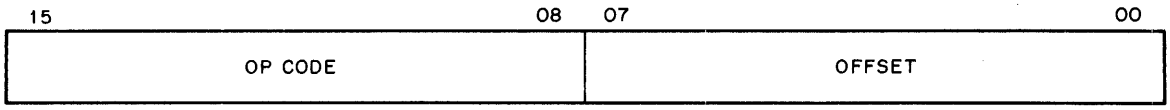
Breakpoint Information Handling – The interrupt vector from the interrupting device is a location whose content points to the first of two words in core. The first word is the program counter (PC), which is the starting address of the service routine. The second word is the processor status word (PSW), which contains the processor priority and condition codes for the service routine.

When the KD11 is interrupted for service, it retrieves the interrupt vector from the data lines, terminates the current program, and stores the breakpoint information (current PC and PSW words) in the hardware stack. The stack — located in core — allows the CP to return to the program when the interrupt has been serviced. KD11 general-purpose register R6 is the hardware stack pointer and contains the address of the latest stack entry.

The CP then retrieves the new PC and PSW words as directed by the interrupt vector and proceeds to execute the required service routine. When the service routine is terminated, the CP reloads the previous PC and PSW words from the stack (via R6) and continues the interrupted program.

Emulator Trap Handling – The emulator (EMT) trap instruction is a software-generated interrupt and is used for identification and tracking of variable input calls to a program. Operation codes 104000–104377 are EMT instructions and may be used to transmit information to the emulating routine. The format of the EMT, shown in Figure 3-23, allows a large number of variables to be assigned in the low-order byte.

When an EMT call is found, the program traps to location 30, which is a permanently assigned address in the trap vector area of core. The contents of the current PC and PSW are pushed onto the hardware stack and replaced by the content of the two-word EMT trap vector (locations 30-32) containing the new PC and PSW. This action causes a jump into the EMT handling routine.



OP CODE = 107
OFFSET = 000-377

10-2139

Figure 3-23 EMT Instruction Format

The handler locates and stores the EMT instruction through the stack pointer. It then retrieves the low-order byte of the instruction, which is used as an entry number into the EMT dispatch table. The low-order byte is added to the starting address of the EMT dispatch table. The resulting dispatch table location contains a pointer to a subroutine that agrees to the trap number. This pointer is used in a branch to the starting address of the required subroutine. At termination of the EMT subroutine, the previous PC and PSW are restored from the stack and the interrupted program is continued.

Register/Vector/Priority Assignments – Table 3-20 lists the device register, vector addresses, and priority levels assigned to the console processor options in a KL10 system. The assignments follow the standard -11 configuration rules for all options except the DL11-C and the DTE20.

The DL11-C is the console terminal controller and has the standard terminal controller address and vector assignments, since all standard software will access this terminal. The DTE20 is not a standard -11 option and as such, has been assigned to the floating vector area along with the DL11-E. The DTE register address assignments are those of the DP11, since this device will never appear in a configuration using the DTE.

Table 3-20 Register/Vector/Priority Assignments

Equipment Option	Register Address Assignment	Vector Address Assignment	Interrupt Priority Level	Physical Unibus Position
DL11-C	777566 (XBUF) 777564 (XCSR) 777562 (RBUF) 777560 (RCSR)	60 (XMIT) 64 (REC)	BR4	1
KW11-L	777546 (LKS)	100	BR6	Internal to KD11
LP20	777476 777460	200	BR4 NPR	14
RX11/RX01/RX02	777172 (RXCS) 777170 (RXDB)	264	BR5	6
CD20 (CD11)	777166 (CDDB) 777164 (CDDA) 777162 (CDCC) 777160 (CDST)	230	BR4 NPR	12

Table 3-20 Register/Vector/Priority Assignments (Cont)

Equipment Option	Register Address Assignment	Vector Address Assignment	Interrupt Priority Level	Physical Unibus Position
RH11/RP04/ RP06/RP07	776746 (RPEC2) 776744 (RPEC1) 776742 (RPER3) 776740 (RPER2) 776736 (RPCC) 776734 (RPDC) 776732 (RPOF) 776730 (RPSN) 776726 (RPDT) 776724 (RPMR) 776722 (RPDB) 776720 (RPLA) 776716 (RPAS) 776714 (RPER1) 776712 (RPDS) 776710 (RPCS2) 776706 (RPDA) 776704 (RPBA) 776702 (RPWC) 776700 (RPCS1)	254	BR5 NPR	3
DL11-E	775616 (XBUF) 775614 (XCSR) 775612 (RBUF) 775610 (RXCSR)	300 (XMIT) 304 (REC)	BR4	5
DTE20	774436 774400	774	BR6 NPR	7

3.2.11 RH20

The RH20 Massbus controller (MBC) is the mass storage interface (data channel) for the DECsystem-10/DECSYSTEM-20. Each MBC is capable of controlling up to eight Massbus-compatible devices (disks, drives, or magnetic tapes). The MBC can be interfaced with single- or dual-ported drives and will allow timeshared swapping and paging software to operate the system efficiently with minimum latency. Up to eight MBCs can be connected to a DECsystem-10/DECSYSTEM-20, and one or more MBCs can operate (reading or writing) simultaneously. However, only one drive per MBC can transfer data at a time. Nondata transfer commands (such as seek and rewind) can overlap and can be issued to any drive at any time as long as the drive is not included in executing a read or write command.

For addressing purposes, each controller is permanently assigned one unique controller select (CS) code. A total of eight controller select codes have been assigned, since up to eight controllers can be implemented in the DECsystem-10/DECSYSTEM-20.

Each controller is also assigned a physical number according to the physical slots in which the controller modules reside. Both the device code and the physical number of a controller are hardwired on the KL10 backplane. The device code is used to address the controller and the physical number is used to identify the interrupting controller.

Since each controller can accommodate up to eight drives, each drive is permanently assigned a unique drive select (DS) code for addressing purposes. The drive select code is permanently hardwired in the drive.

To permit any type of mass storage device (disk, drum, or magnetic tape) to be interfaced with the same controller, the working registers (status control, address, command, and data) are divided between the controller and the drive. Registers required for all drives reside in the controller; registers required to operate a given drive are implemented in that drive. Accordingly, registers that are implemented in the controller are referred to as internal registers, and registers that are implemented in the drive are referred to as external registers. Up to 32 internal registers can be implemented in the drive. The RH20 controller and available drives only use a subset of the available register address space.

Each controller has two command (internal) registers. They are the secondary transfer command and primary transfer command registers. A command in the primary command register will be executed immediately provided no transfer error condition is detected in the controller. The secondary command register serves as a command lookahead facility.

The command in the secondary command register will be executed as soon as the command in the primary command register is terminated (done) and no transfer error is detected in the controller.

CAUTION

Only data transfer (read/write) commands can be loaded into the command registers. The program should load nondata transfer (seek, rewind, and so on) commands directly into the drive's (internal) control register.

The secondary command register allows the software to specify the next command to be executed before the controller has completed the current command.

This means that the controller can start the next command immediately after the current command is done — instead of waiting for a software interrupt routine to supply the next command and miss the next sector. Without this lookahead feature, the software can only transfer every other block or page for different users.

The controller will interrupt the EBox when any of the following conditions occur.

1. A data transfer (read/write) command is done (with or without a transfer error).
2. An attention signal from the drive (caused by SEEK COMPLETE, and so on) is detected on the Massbus by the controller, provided that the attention interrupt enable control (CONO) bit is set.
3. A register access error is detected in the controller when a drive is loaded or read.

Commands are arranged into two groups: nondata transfer and data transfer. The nondata transfer commands do not cause data channel transfers. These commands are generally used to set up the drive for a subsequent read or write command. The drive will assert the MBus attention line to inform the MBC and the EBox that the setup is completed.

Data transfer commands are those that cause data channel transfers over the CBus. These commands are limited to read and write operations.

3.2.11.1 MASSBUS Description – The Massbus provides the interface between the RH20 controller and the disk pack and tape mass storage devices. The Massbus is composed of two separate, independent buses: the control bus and data bus. These independent buses allow synchronous and asynchronous communication between the RH20 and its drives. Each Massbus line is described in Table 3-21. Figure 3-24 illustrates the Massbus configuration.

Table 3-21 Massbus Line Descriptions

Signal Line	Description
Control Bus	
Control and status (C00–15)	Transfers 16 parallel control or status bits to or from the drive.
Control bus parity (CPA)	Transfers odd control bus parity to or from the drive. Parity is simultaneously transferred with control bus data.
Drive select (DS0–2)	Transfers a 3-bit binary code from the RH20 to select a drive. The drive responds when the (unit) select switch in the drive corresponds to the transmitted binary code.
Register select (RS0–4)	Transfers a 5-bit binary code from the RH20 to select a particular drive register.
Controller to drive (CTOD)	Indicates which direction information is to be transferred on the control bus. For a controller-to-drive transfer, the RH20 asserts CTOD; for a drive-to-controller transfer, the RH20 negates CTOD.
Demand (DEM)	Asserted by the RH20 to indicate a transfer is to take place on the control bus. For a controller-to-drive transfer, DEM is asserted by the RH20 when data is present. For a drive-to-controller transfer, DEM is asserted by the RH20 to request data and is negated when the data has been strobed from the control bus. In both cases, the RS, DS, and CTOD lines are asserted and allowed to settle before assertion of DEM.
Transfer (TRA)	Asserted by the drive in response to DEM. For a controller-to-drive transfer, TRA is asserted when the data is strobed and negated when DEM is removed. For a drive-to-controller transfer, TRA is asserted when the data is asserted on the bus and negated when the negation of DEM is received.

Table 3-21 Massbus Line Descriptions (Cont)

Signal Line	Description
Attention (ATTN)	The drive asserts this line to signal the RH20 of any change in drive status or an abnormal condition. ATTN is asserted any time a drive's ATA status bit is set. ATTN is common to all drives and may be asserted by more than one drive at a time.
Initialize (INIT)	Asserted by the RH20 to initialize all drives on the bus. This signal is transmitted at system startup or whenever the RH20 issues an initialize command.
Fail (FAIL)	When asserted, this line indicates a power-up fail condition has occurred in the RH20.
Data Bus	
Data (D00–17)	These bidirectional lines transfer 18 parallel data bits between the RH20 and drives.
Data bus parity (DPA)	Transfers an odd parity bit to or from the drive. Parity is simultaneously transferred with bits on the data bus.
Sync clock (SCLK)	Asserted by the drive during a read operation to indicate when data on the data bus is to be strobed by the RH20. During a write operation SCLK is asserted to the RH20 to indicate the rate at which data should be presented on the data bus.
Write clock (WCLK)	Asserted by the RH20 to indicate when data to be written is to be strobed.
Run (RUN)	Asserted by the RH20 to initiate data transfer command execution. During a data transfer, the drive samples run at the end of each sector. If run is still asserted, the drive continues the transfer into the next sector; if run is negated the drive terminates the transfer.
End-of-block (EBL)	Asserted by the drive at the end of each sector. For certain error conditions where it is necessary to terminate operations immediately, EBL is asserted preceding to the normal time. In this case, the transfer is terminated before the end of the sector.
Exception (EXC)	Asserted by the drive to indicate an error condition during a data transfer command. EXC stays asserted until the trailing edge of the last EBL pulse.
Occupied (OCC)	Asserted by the drive to indicate that the drive has accepted a valid data transfer command.

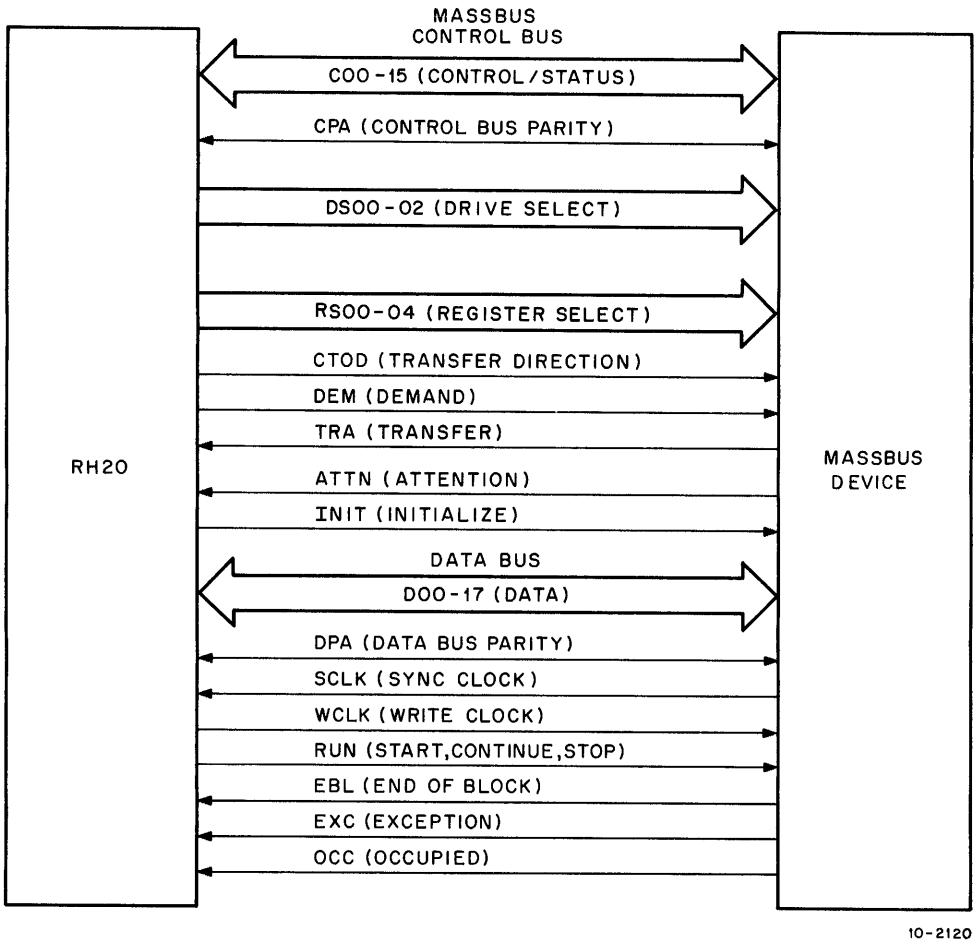


Figure 3-24 Massbus Configuration

Command Initiation

Commands are of two types: data transfer commands (read, write) and nondata transfer commands (drive, clear, search). Data transfer commands are initiated through the RH20 control register. All nondata transfer commands to this register are illegal and will result in an error condition.

Nondata transfer commands affect only the state of the drive. The RH20 writes the command word into the drive control register. At completion of command execution, the drive asserts ATTN, indicating command completion. In addition, if the nondata transfer command is not recognized as a valid command, the drive will immediately signal an error by asserting ATTN.

To initiate a data transfer command in a drive, the RH20's control register is loaded through a DATAO instruction, which causes the command word to be written into the drive's control register. If the command is valid, the addressed drive executes the command. With the command loaded, the channel is started and a control bus cycle is initiated. The RH20 asserts DEM; the drive returns TRA. The RH20 then asserts RUN, and the drive responds with OCC on the data bus portion of the Massbus. It is over this bus that the data transfer will occur.

Control Bus Write Operation
For a write operation to a drive register, the RH20 asserts:

1. The C lines with the register data to be transferred
2. The DS lines, to specify a particular drive
3. The RS lines, to specify the register to which the data must be transferred
4. The CTOD line, to indicate that transfer direction is to the drive.

After these signals have settled on the bus, the RH20 asserts DEM to cause the drive to load the data on the C lines into the addressed register. The drive asserts TRA, indicating that the register has been loaded. The RH20 then negates DEM. At the same time, it negates the C, RS, and CTOD lines, terminating the operation (see Figure 3-25).

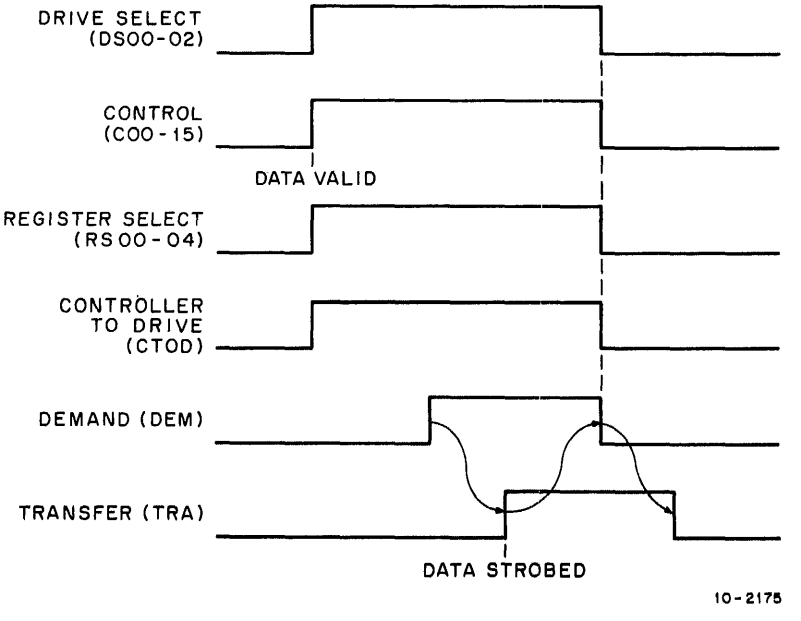


Figure 3-25 Control Bus Write Operation

Control Bus Read Operation
The register read operation sequence is similar to the write sequence. The DS and RS lines are asserted by the RH20; the CTOD line is negated, indicating that the transfer direction is from the drive. The RH20 then asserts DEM, causing the drive to place the register data on the C lines. When the C lines have settled, the drive asserts TRA causing the RH20 to strobe the data from the C lines. The RH20 then negates DEM, causing the drive to negate TRA and the C lines and terminating the operation (see Figure 3-26).

Data Bus Write Operation
In executing a write command, the desired address and write function are set in the drive. The RH20 asserting RUN initiates an address search operation. When the desired sector is found, the drive starts issuing SCLKs. The SCLKs are echoed back to the drive by the RH20 as WCLKs. On the leading edge of WCLK, the drive strobes D00–17 into its data buffer. On the trailing edge, the RH20 displays another word on D00–17.

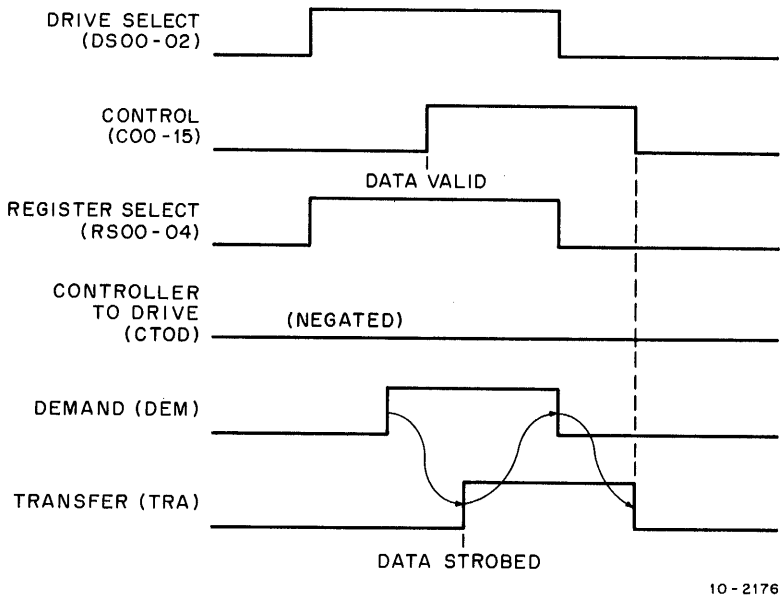


Figure 3-26 Control Bus Read Operation

During write (and read) operations, the drive asserts EBL after the last word of each sector is transferred. At the end of each sector (EBL time) the drive samples RUN; if it is still asserted, it continues the transfer into the next sector. If RUN is negated, the drive terminates the transfer (see Figure 3-27).

Data Bus Read Operation

To execute a read command, the RH20 sets the desired address and read function to the drive. It then asserts RUN, causing the drive to search for the desired address. Once the address is found, the data transfer starts.

The drive places the first data word on the D lines with the assertion of SCLK. The RH20 interprets the leading edge of SCLK to indicate that data has been placed on D00–17. It uses the trailing edge of SCLK to strobe the data into its data buffer. As in the write operation, the drive samples RUN at the end of each sector, and continues transferring sectors until RUN is negated (see Figure 3-28).

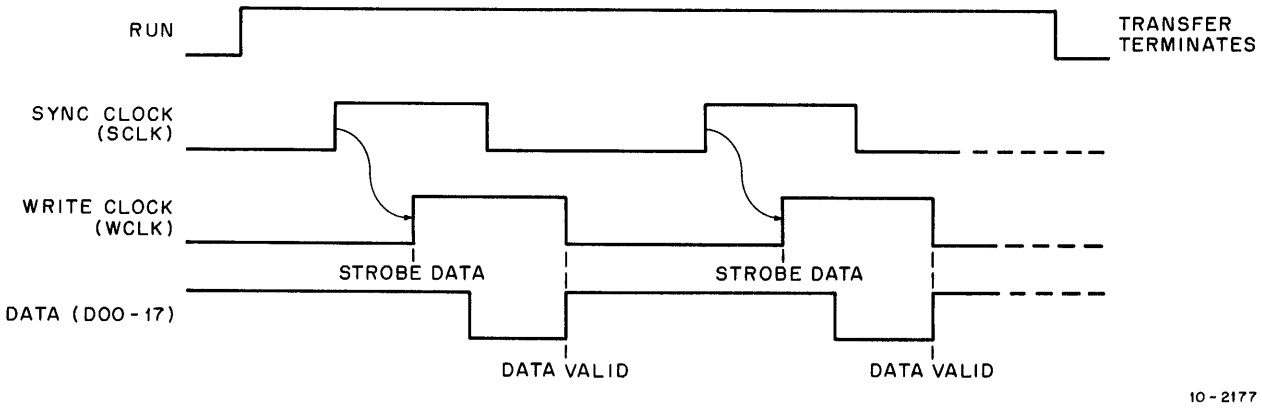


Figure 3-27 Data Bus Write Operation

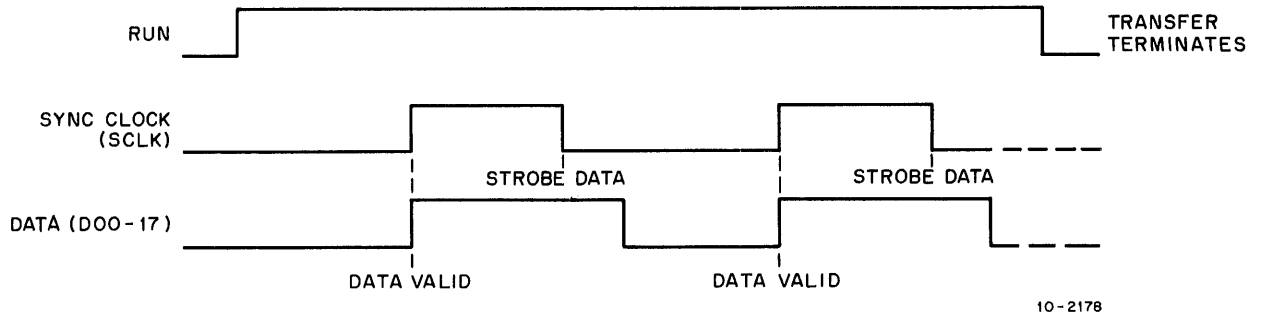


Figure 3-28 Data Bus Read Operation

Data Transfer Condition
Under certain abnormal conditions, the RH20 may require termination of a data transfer prior to the end of the current sector. In such a case, the RH20 asserts EXC (which is recognized only by the drive transferring data) causing that drive to immediately assert EBL and terminate the transfer. EXC is asserted by the drive to indicate to the RH20 that an error condition has occurred during a data transfer.

In addition, the ATTN line is used to indicate an error condition or status change in the drive. ATTN may be asserted under any of the following conditions.

1. An error detected while no data transfer is taking place (asserted immediately)
2. Completion of a data transfer command if an error occurred during the data transfer
3. Completion of a nondata transfer command (such as search)

3.2.12 Interrupt Facility
The KL10 main processor has an eight level priority interrupt facility. Seven levels (levels 1–7) are programmable and one level (level 0) is permanently assigned to the DTE. The order of priority is from 0 to 7.

Each I/O device on the EBus (including internal processor devices) can be assigned one of the programmable priority levels (channel) and can then interrupt the processor on that channel when it requires service or when it has completed a programmed operation. When a device issues an interrupt on level n, the next instruction is taken from location 40 + 2*n of the Exec Process Table. This instruction is then executed in the exec mode. After the interrupt is serviced, control is restored to the interrupted program. All processor flags, PC, and ACs are saved and restored when servicing an interrupt.

Priority level 0, the highest priority level, is not assignable but is reserved for the front-end processor interface (DTE) in executing deposit, examine, and byte transfer operations. The DTE can also be assigned a programmable interrupt level for reporting status information and for requesting service.

3.2.13 Trap Facility
The direct I/O and priority interrupt facilities permit the processor to maintain system status and to effect control. To add to these facilities, the main processor also includes a trapping mechanism. This mechanism allows certain conditions resulting from an executing program to interrupt the program sequence without having to go to the direct I/O or priority interrupt facilities, which would be time consuming. The conditions that are sensed by the trapping mechanism are:

1. Address violation
2. Arithmetic overflow

3. Pushdown overflow
4. Page faults
5. Illegal instructions in user mode (I/O)
6. Monitor calls (UUO and MUUO).

3.2.14 Internal Devices

The main processor contains several internal devices that can be accessed under program control using the appropriate I/O instructions. These devices provide the means for initiating internal processor functions and for providing ready access to processor status information. The internal devices are:

1. APR – Arithmetic processor registers
2. PI – Priority interrupt registers
3. PAG – Pager registers
4. CCA – Cache clearer address and control registers
5. TIM – Timer registers
6. MTR – Meter registers.

These registers are considered to be internal I/O devices because they can be accessed under program control in the same way that conventional I/O devices are accessed using the standard DECsystem-10/DECSYSTEM-20 I/O instructions.

3.2.14.1 APR – The APR device facilitates programmable access and control of processor identification information, the address break facility, the cache refill RAM, and the processor status and error flags.

3.2.14.2 PI – The PI device facilitates programmable access and control of the error address (ERA) register, the SBus diagnostic cycle control, and the priority interrupt facility status and control bits.

3.2.14.3 PAG – The PAG device facilitates programmable access to the pager in the MBox for setting up the executive and user base registers and for invalidating desired entries in the pager. This device also provides the means for selecting the desired mapping mode (KI or KL paging), for selecting the desired cache use strategy and for context switching.

3.2.14.4 CCA – The CCA device provides programmable access to the cache clearer control in the MBox for clearing the cache.

3.2.14.5 TIM and MTR – The meter (TIM and MTR) device facilitates access to the following clocks.

1. The 1 microsecond interval timer, which is a source of programmable interrupts with a maximum period of 32 milliseconds

2. The 1 microsecond readable time base

3. The accounting meter, which counts EBox clock ticks and MBox references with two separate counters

4. The performance analysis counter, which is used in evaluating the performance of the system

3.2.15 External and Internal I/O Controllers and Devices (Typical) – Every device has a 7-bit device selection network; a priority interrupt assignment; and at least two flags, busy and done, or some equivalent. The selection network decodes bits 03–09 of the instruction so that only the addressed device responds to signals sent by the KL10 main processor over the EBus. To use the device with the priority interrupt facility, the program must assign a channel to it. Then whenever an appropriate event occurs in the device, it requests an interrupt on the assigned channel.

The busy and done flags together indicate the basic state of the device. When both are clear, the device is idle. To place the device in operation, a CONO or DATAO sets busy. If the device will be used for output, the program must give a DATAO that sends the first unit of data — a word or character depending on how the device handles information. When the device has processed a unit of data, it clears busy and sets done to indicate that it is ready to receive new data for output or that it has data ready for input. In the former case, the program would respond with a DATAO to send more data; in the latter, with a DATAI to bring in the data that is ready. If an interrupt channel has been assigned to the device, the setting of done signals the program by requesting an interrupt; otherwise the program must keep testing done to determine when the device is ready.

3.3 FRONT-END PROCESSOR SUBSYSTEM

The front-end processor is a hardware/software subsystem that replaces typical control and indicator panels. To replace this hardware, the front-end processor provides an automatic bootstrap mechanism and a console command facility. Besides these functions, the front-end processor also handles low-speed, asynchronous I/O operations for unit-record and communications equipment using the byte transfer facility. The front-end processor, which includes a PDP-11 minicomputer system, interfaces with the main processor through the DTE ten-eleven interface.

The bootstrap facility automatically initializes the dispatch and control RAMs in the EBox, places a small sequence of code into the memory of the main processor, and starts its execution to load the system. Several bootstrap options, including an operator/software dialog, are available. The console command facilities are available through the console terminal (CTY) to permit the operator to examine or deposit main processor memory locations and otherwise control and monitor the processor in the same way given through the typical operator console and indicator panels.

The PDP-11 used as the front-end processor is a 16-bit, general-purpose, microprogrammed minicomputer that uses single- and double-operand instructions and two's complement arithmetic. Included in the front-end processor are the associated -11 controllers and peripheral devices.

The instruction word format is such that the processor can directly address up to 32 K words (64 K bytes) of core memory. Only 28 K words are available for program storage. The continuing 4 K words are reserved for peripheral and register addresses. All communication among system components (including processor, core memory, and peripherals) is performed over the Unibus. Because of the Unibus concept, all peripherals are compatible, and device to device transfers can be done at the rate of 2.5 million 16-bit words or 8-bit bytes per second. All system components and peripherals are linked by the Unibus, and all peripherals are in the basic system address space. Most instructions applied to data in memory can also be applied to data in peripheral device registers, enabling peripheral device registers to be manipulated as flexibly as memory.

3.3.1 Devices

The following paragraphs provide a short description of each component that is part of the front-end processor subsystem.

3.3.1.1 KD11-A Central Processor – This device is the basic component of the front-end processor. The KD11 is connected to the Unibus as a subsystem. It controls time allocation of the Unibus for peripheral devices and performs arithmetic and logic operations through instruction decoding and execution. The instruction set is implemented through a group of hardware subroutines stored within the 256 by 56-bit read-only memory (ROM).

3.3.1.2 KY11-D Programmers Console – This device is an important part of the PDP-11 system. It provides the operator a direct system interface through the control switches and display indicators. The control switches provide the operator with real-time control of normal program and diagnostic operations, therefore allowing the operation to start, stop, load, modify, or continue a program.

3.3.1.3 KW11-L Line Clock Option – This device provides a method of referencing real-time intervals by generating a repetitious interrupt request to the KD11. The rate of interrupt is derived from the ac line frequency.

3.3.1.4 MF11-UP Memory – This device is a read/write, random-access, coincident current, magnetic core type memory with a maximum cycle time of 980 ns and a maximum access time of 425 ns. It is organized in a 3D, 3-wire planer configuration. Storage capacity is 16,384 words (32,768 bytes) with an 18-bit word length (16 data bits plus two parity bits).

3.3.1.5 MM11-UP Memory – This device is a 16 K word expansion memory having characteristics identical to the MF11-UP.

3.3.1.6 BM873-YJ ROM Loader Module – This module is an essential part of the front-end processor. It provides 256 words of read-only memory, which are blasted to contain a number of routines to facilitate bootstrap, power-fail, and dump operations.

In addition to these routines, the ROM module contains four locations that serve as entry points for these routines. The routines in the ROM facilitate bootstrapping and dump operations; you initiate these from the -10 via the DTE 10/11 interface and by pressing a physical load push button on the switch panel.

Either the -11 based floppy disk or the -11 based RP06 disk may be specified for the bootstrap or dump procedure.

NOTE
These storage devices are not supported as system devices.

3.3.1.7 DL11-C Asynchronous Line Interface – This device provides simultaneous two-way transmission between the console terminal and the Unibus. The DL11-C is a character-buffered interface that controls and translates serial bit flow data from the terminal to parallel character data for transfer over the Unibus. The interface also provides parallel to serial translation from the Unibus to the terminal.

3.3.1.8 DL11-E Asynchronous Line Interface – This device has identical characteristics as the DL11-C and in addition, provides control functions for a communication modem (such as, Bell Model 103) that interfaces with the KLINIK diagnostic facility.

3.3.1.9 LA120 Keyboard Terminal – This is a high-speed data communications terminal. The terminal consists of a bidirectional, 180 character/second printer with up to 9,600 baud serial data communication capability and an operator's console that includes a typewriter-type keyboard.

3.3.1.10 DC20F Asynchronous 16-Line Multiplexer – The DC20 asynchronous 16-line programmable multiplexer (similar to the DH11) connects the PDP-11 with up to 16 asynchronous serial communications lines operating with the following individually programmable parameters.

1. Character length: 5-, 6-, 7-, or 8-bit
2. Number of stop bits: 1 or 2 for 6-, 7-, 8-bit characters
   1 or 1.5 for 5-bit characters
3. Parity generation and detection: odd, even, or none
4. Operating mode: half-duplex or full-duplex

5. Transmitter speed and receiver speed: 0, 50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2400, 4800, or 9600 baud plus Ext A, Ext B

6. Breaks: May be detected or generated on each line

The DC20 multiplexer uses 16 double-buffered MOS/LSI receivers to assemble the incoming characters. An automatic scanner takes each received character and the line number and deposits that information in a first-in, first-out buffer memory referred to as the silo. The bottom of the silo is a register that is addressable from the Unibus.

The transmitter in the DC20 also uses double-buffered MOS/LSI units. They are loaded directly from message tables in the PDP-11 memory by means of single-cycle direct memory transfers (NPR). The current addresses and byte counts for each line's message table are stored in semiconductor memories located in the DC20. This decreases the Unibus time taken for the NPR transfers to one NPR cycle per character transmitted. The NPR cycle used is extended slightly.

3.3.1.11 CD20 Card Readers – The CD20 card readers are rated and designed to meet changing throughput requirements and provide reliable, quiet, trouble-free operation. These low-cost card readers accept 80-column EIA/ANSI standard cards.

For fast throughput, the user can select the console model CD20-B, which processes 1200 cards/minute, or the table model CD20-A, which processes 300 cards/minute and has a smaller hopper while still including the same basic features as the higher speed model.

CD20 card readers are designed to prevent card jams and keep card wear to a minimum. The readers have a high tolerance to cards that have been subjected to high humidity or to rough handling and are worn, scratched, warped, bent, or otherwise damaged.

To keep cards from sticking together, the readers use a special "riffle air" feature. The bottom half-inch of cards in the input hopper are subject to a current of air that separates the cards and air cushions them from the deck and from each other. This action unsticks those cards attached electrostatically and loosens those cards attached through torn webs or hole locking. It also separates cards that are swollen and stuck from high humidity.

Cards entering the reader are selected through an advanced design vacuum picker. The picker and its associated throat block prevent the unit from double selecting, so that cards that have been stapled or taped together (unless such taping is on the leading edge) will not enter the card track. To lower the chances of jamming, the card track is short (less than four inches) so that only one card at a time is in motion.

The "riffle air" and vacuum picker features greatly extend card life. Stoppages are also decreased since the reader automatically tries six times before it determines that a card cannot be selected.

The read station of the CD20 card readers uses infrared light emitting diodes as its light source and phototransistors as its sensors to provide complete dependability. No adjustments are needed during the ten years of life expectancy of the diodes. Typical incandescent sources, on the other hand, need continuous adjustments with age.

3.3.1.12 LP26 Line Printer – The LP26 line printer produces hard copy with a maximum line length of 132 columns. It prints at a speed of 600 lines/min using the ASCII 64-character set or at a speed of 445 lines/min using the AXCII 96-character set.

The LP26 includes the direct-access vertical format unit (DAVFU) and is driven by the LP20 line printer controller through the long line interface (LLI).

3.3.1.13 RP06 Disk File System – This system consists of an RP06 disk drive and an RH11 device controller. The RP06 is a dual Massbus port, moveable head, disk pack drive used as the -11 diagnostic and bootstrap load routines. The RH11 provides the control and parallel data path interface between the Unibus and RP06 through the Massbus. In addition to communicating with the KD11, the controller has access to -11 memory to fetch and store data.

NOTE

The drive is not supported as a system device from the -11 side but is supported as a system storage device from the -10 side.

3.3.1.14 RX11 Floppy Disk Subsystem – The RX11 floppy disk subsystem is a very dependable, low-cost, mass storage system capable of storing up to 256,256 8-bit bytes per drive in an industry-compatible format. The RX11 provides a data interchange and software distribution medium for critical I/O applications. In addition, the RX11's random-access capability allows configuring very low-cost, disk-based systems with small PDP-11 processors. Such systems can meet the needs of applications that could never before afford random-access storage.

The RX11 floppy disk system consists of an RX01W-BA (RX02) model of a floppy disk drive unit and a PDP-11 quad interface module that requires a single SPC slot. The RX11 includes either one or two drives, a microprogrammed controller module, and a read/write electronics module, all housed in a 10½ inch, rack-mountable chassis. Up to two drives can be supported by each controller for a total storage capacity of 512,512 bytes.

Given an absolute sector address, the RX01W-BA (RX02) locates the desired sector and performs the indicated function. It automatically verifies head position and generates and verifies the cyclic redundancy check (CRC) character.

Track-to-track moves require 10 milliseconds for the move plus 20 milliseconds for settling time if the head is loaded for a read or write. The rotational speed of the diskette is 360 r/min, which results in an average latency time of 83 milliseconds. The track-to-track move, head settling, and latency time produce an average access time of 483 milliseconds. During a sequential access, the complete diskette can be read in about 30 seconds.

The RX01W-BA (RX02) floppy disk uses the industry-standard "diskette" or "floppy" media. These are thin, flexible, oxide-coated disks similar in size to a 45 r/min phonograph record. The disk is recorded on one side only and is permanently contained in an square, 8 inch, flexible envelope.

The envelope has a large center hole for the drive spindle, a small hole for track index sensing, and a large slit for the read/write head. A solenoid contact load pad is located on the opposite side of the envelope. The inside of the envelope is covered with a soft material, designed to wipe the disk surface clean right before reading.

The diskette contains 77 tracks and 26 sectors per track. Each sector can store 128 8-bit bytes for a total formatted capacity of 256,256 8-bit bytes.

The diskette is a good storage, interchange, and software distribution medium. Compared to disk cartridges or disk packs, it is less expensive. Because it is flat and thin, the diskette is compact, enabling large amounts of data to be stored in a small space. Diskettes can also be transported with ease in a briefcase or in a manila envelope.

Because the diskette is preformatted in the industry-standard format, it conforms to industry compatibility and drive-to-drive interchangeability. The RX01W-BA (RX02) can read diskettes written on other standard floppy disk equipment and vice versa. Preformatted diskettes also decrease hardware costs by deleting the circuitry needed to generate the correct format.

NOTE
The floppy disk is not supported as a system device.

3.3.1.15 BC11-A Unibus – The Unibus is a 120-conductor ribbon cable connecting the front-end processor system components. The Unibus consists of 56 signal and 64 ground lines assembled alternately within the cable to minimize crosstalk.

On the EBus, the DTE appears in a way as a DECsystem-10/DECSYSTEM-20 device controller. On the Unibus it connects as a standard -11 peripheral device, using the direct memory access and vector interrupt features of the -11.

3.3.2 Interdevice Transfers

Communication between two devices on the Unibus is done in a master-slave relationship. During any bus operation only one device has control of the bus. This device (master) controls the bus when communicating with another device on the bus (slave). For example, the KD11, as the master, transfers data to the MF11 or MM11, which operate as the slave. Master-slave relationships are dynamic. The KD11, for example, can pass bus control to a disk. The disk, as master, may then communicate with the slave memory.

Since the Unibus is used by the KD11 and all its I/O devices, a priority structure determines which device gets control of the bus. Therefore, every device capable of becoming bus master has an assigned priority level. When two devices having identical priority levels simultaneously request use of the bus, the device electrically closest to the KD11 receives control. The KD11 performs priority arbitration and when no other device has bus control, assumes bus control.

Full 16-bit words or 8-bit bytes can be transferred over the bus between master and slave. The data in (DATI) and data in pause (DATIP) operations transfer data into the master; data out (DATO) and data out byte (DATOB) operations transfer data out of the master. When a device requests control of the bus, it is for one of two purposes.

1. To make a direct memory access (DMA) transfer of data directly to/from another device or memory without processor intervention

2. To interrupt program execution and force the processor to branch to an interrupt service routine

Bus control is found under a nonprocessor request (NPR) for the direct memory access or under a bus request (BR) for an interrupt.

Requests for the bus can be made any time on the BR and NPR lines. Transfer of bus control from one device to another is made by the KD11 priority arbitration logic, which grants control of the bus to the device having the highest priority. NPRs are granted higher priority than BRs. The NPRs are serviced between bus cycles, in addition to specific times during wait or trap sequences. BRs are serviced on completion of the current instruction if the requesting priority exceeds that of the processor.

3.3.3 Functions

One of the major functions of the front-end processor is to provide the typical console functions for the KL10 main processor. The front-end processor is programmed to accept console commands to display and change locations in the KL10 memory, to start and stop the main processor, and to affect many other

operations. The functions, initiated through the terminal connected to the front-end processor, are implemented by the following hardware implemented operations.

1. Examine
2. Deposit
3. TO10 transfer
4. TO11 transfer
5. System bootstrap
6. Interprocessor interrupts
7. Diagnostic and miscellaneous console functions

3.3.3.1 Examine/Deposit Operations – The examine and deposit functions allow the front-end processor to fetch or modify any location in KL10 memory while the main processor is running or executing a halt. Examine and deposit are handled as a priority interrupt (PI) request with a priority higher than any programmed PI level. Note that these functions are completed even when the PI system is off and the main processor is halted.

The deposit operation accesses and writes a 36-bit data word into a -10 memory location. Both the data word and 23-bit addresses are entered through the console terminal. The examine operation accesses and retrieves a 36-bit data word from a -10 memory location for display on the console terminal. As with the deposit, the examined memory location is specified through the console terminal.

An examine or deposit starts when the -11 program writes the -10 address into the DTE20. No program interrupts are generated on the -10 or -11 side to indicate completion of the operation. Therefore, the -11 program must check for completion by monitoring the appropriate flag in the status register of the DTE. The DTE logic is structured such that once the address and data is written into the DTE it stays the same after an operation. The -11 may now perform repeated examines and deposits by changing the -10 address each time.

3.3.3.2 TO10/TO11 Byte Transfer Operations – TO10/TO11 transfers are multiple bus operations (including both the EBus and Unibus) transferring fields of information between the -10 and -11. Multiple transfers are executed for both byte transfers, only the source and destination differ. In the TO11 transfer the source of information is the -10 and the destination is the -11. In the TO10 transfer source and destination are reversed; that is, the source is the -11 and destination is the -10.

The fields of information that are transferred between the processors differ at the 10/11 interface (DTE20). At the -10 side of the DTE, the fields are of variable length and are accessed through a -10 byte pointer. At the -11 side of the DTE, the fields are either 8 bits wide and stored in consecutive byte locations in -11 memory, or 16 bits wide and stored in consecutive (even) word locations in -11 memory. If the field into which the information is being stored is narrower than the field from which it was read, as many rightmost bits as will fit are stored. If the field into which the information is being stored is wider than the field from which it was read, the information is right-aligned and padded with zeros in the high-order bits.

Prior to the actual transfer, several parameters are provided by both processors to the DTE. First (possibly at system startup) the -11 determines the transfer rate and the Unibus address bits 17 and 16. The transmitting processor specifies the source address. The receiving processor specifies the destination address for the data and a byte count equivalent to the length of the data string. In addition, the receiving processor determines whether it alone, or both processors will receive the normal termination interrupt. When the transfer is initiated the receiving processor's byte count is decremented as each word (or byte) is transferred. At zero byte count the transfer is complete and the receiving processor (and transmitting processor, if specified) is interrupted. Note that in operation, the actual interrupt is issued from the DTE.

3.3.3.3 System Bootstrap Function – First both systems must be loaded by the bootstrap function in order to start operation. Bootstrapping can be initiated in several ways.

1. A power-fail restart
2. The operator pressing a bootstrap button (floppy disk, disk pack, or switch register) on the system switch panel
3. The KL10 initiating a bootstrap of the -11
4. The operator entering appropriate commands to the console command facility via the console terminal

Generally the bootstrap is initiated from the -11 ROM loader module, which provides a minimum number of instructions to load the absolute loader program from a selected storage media. The absolute loader in turn loads the initialization, handling, and device support routines required by the -11. The bootstrap then loads the -10 control RAM and dispatch RAM with enough code to start the -10 running. As more code is transferred, the -10 will configure its memory, load the resident monitor, set up communications with the front-end processor, and relinquish control to the resident DECsystem-10/DECSYSTEM-20 monitor.

3.3.3.4 Interprocessor Interrupts – These interrupt operations provide the interprocessor communication function; that is, the capability of either processor to interrupt the other. The interprocessor interrupts (doorbell feature) allows the -10 to interrupt the -11 as well as the -11 to interrupt the -10. The doorbell consists of a programmable interrupt and status flags located in the -10 and -11 status registers of the DTE20.

For the -11 to interrupt an interfaced -10, the -11 sets an interrupt flag in its associated DTE status register using a DATO. With the flag set, the DTE generates an interrupt to the -10. The -10 then executes a CONI, which informs the -10 that the -11 has programmed an interrupt to the -10 for it to initiate appropriate action.

The procedure is executed in a reversed but similar way for the -10 interrupting the -11. The -10 sets an interrupt flag by executing a CONO to the DTE, which in turn generates an interrupt to the -11. The -11 finds the cause for the interrupt by monitoring the flag in the DTE status register and initiating appropriate action.

3.3.3.5 Diagnostic and Miscellaneous Console Functions – Other console functions, such as displaying the contents of certain -10 registers or memory locations, require the cooperation of the operating systems. The operating system must regularly store the quantities to be displayed in the communication areas. Displayed information may include the PI system state, current job being run, number of active jobs, program counter on the last clock interrupt, and so on. It is also possible to simulate all of the typical console indicators used while the system is in normal operation.

Major -10 CPU state information is continuously available on the diagnostic bus while the system is running. In addition, for the case of a -10 crash, the -11 can use the diagnostic bus to determine additional hardware status information. However, the front-end processor is not allowed to use the diagnostic bus for data transfers during normal system operation since this would interfere with normal traffic on the EBus.

3.3.4 Modes

The front-end processor system has two operating modes: privileged and restricted. They are switch-selectable from the DTE20.

The privileged processor has access to the diagnostic bus and the capability to execute unprotected examines and deposits. Unprotected examines and deposits are unique in that they require special software and may address any of the following areas in -10 memory: Exec Process Table and executive virtual address spaces; User Process Table and user virtual address spaces; and the actual physical address space. Although a privileged processor normally executes protected examines and deposits, it does have the capability to override the normal protection checks defined in the Exec Process Table (EPT). Because of the relatively unlimited access allowed, a privileged front-end processor may degrade system operation or crash the associated KL10 main processor.

The restricted front-end processor can only access -10 memory provided the -10 has executed a CONO instruction and enabled the associated PI0 level. After PI0 is enabled, the restricted processor can then only examine in a -10 owned region and deposit in its own -11 owned region. In addition, the processor is prohibited from using the diagnostic bus. Since the restricted processor cannot violate the -10's system security, it has no more privileges or capabilities than a user program. Because of its limited access, it does not have the capability to degrade the -10's operation.

3.3.5 Interprocessor Communication

Interprocessor communication is necessary to allow -11 and -10 to execute those functions required during timesharing, bootstrapping and diagnostics. This communication is implemented by special communication areas allocated in -10 memory. These areas are used to coordinate status, prepare byte transfer operations, and process limited amounts of data. Communication areas are allocated to each processor in the system such that each processor can read or write its own allocated area (that is, -10 owned region, -11 owned region) but only read the other area owned by the system. These areas reflect the hardware and software states of the owning processor to its associated processor.

In addition to the communication area functions, the majority of control information and data transferred between a -10 and -11 is through software processing queues. A TO11 queue is maintained in -10 memory by the monitor. The -11 will access this queue using byte transfers through the DTE20. The TO10 queue is maintained in the -11 memory and is accessed by the -10 in a similar way. Since the processing queues are not part of the communication areas and are accessed only by byte transfer operations, they are protected from modification by any processor other than the making processor. When a queue is made and ready for transfer operations, the transmitting processor will interrupt its associated processor. At this point the interrupted processor starts processing the queue.

The -10 is also able to communicate with a console terminal directly through the communication areas, independent of queue processing. Normally, it is only used during bootstrap, diagnostic operations, or when the monitor finds it inconvenient to output an error message using queue processing.

3.3.5.1 Communication Areas – First the -10 will set up the communication areas at load time with each processor responsible for protecting itself from the other. Since interprocessor communication is through the DTE20, a pair of communication areas is associated with the DTE (that is, -10 owned and -11 owned areas).

The -11 owned area is defined in -10 memory by the deposit relocation and protection word in the EPT (EPT DPW). The area is written by the -11 using protected deposits and read by the -10. Each -11 in the system has a separate area that it alone can modify. The -10 owned area is defined purely in software and is separate from the -11 owned area. It is written by the -10 and read by the -11 using protected examines.

Each processor's communication area is divided into two zones. The first zone contains 16 (36-bit) words with identification and hardware and software status information specific to the owning processor. The second zone contains an additional eight words of communication status information for each processor that is in communication with the owning processor. Therefore, the size of each communication area is variable depending on the number of processors in the system.

3.3.5.2 Queue Processing/Messages – Information transferred between processors is stored in variable length queues and accessed using byte transfer operations. Each processor maintains a queue of messages waiting for transmission to the associated processor. Each queue has an associated word in the transmitting processor's communication area indicating to the receiving processor the size of the collected queue.

After a transmitting processor places information in its queue, it interrupts the receiving processor informing it to start processing queue entries using byte transfers. With the DTE20 hardware, executing byte transfers in either direction requires cooperation of both processors. For example, to perform a TO10 byte transfer the following general parameters are required.

1. The -11 provides the DTE with the source address of the queue to be transferred.

2. The -11 specifies how the queue is stored in -10 memory (that is, 8-bit bytes or 16-bit words).

3. The -10 sets the EPT byte pointer word to a byte pointer location where the queue is to be stored.

Queue content is varied, containing control information as well as data. Each queue content (or message) contains the length of that message (in bytes) in its first entry. Most messages will contain information indicating the -11 device inputting data, or the -11 device for which the data is destined. Message content may contain information about queue processing. For example, it may contain information indicating the end of a queue and resetting the DTE to start processing the next queue. In some cases, the queue message may be in the form of a pointer that defines data not in the queue but located elsewhere in the transmitting processor's memory. This type of messages is used when it is more efficient to change the byte pointers to point to another area of memory than to copy that memory area in the queue.

3.4 MASS STORAGE SUBSYSTEMS

Several types of mass storage subsystems are included in the DECsystem-10/DECSYSTEM-20.

The system supports mass storage subsystems to serve as large file storage and swapping areas. Both disk and magnetic tape storage drives can be attached to a channel I/O processor, which is an integral part of the main processor.

The channel I/O processor (channel control), is time-division multiplexed to provide service for up to eight separate synchronous channel paths simultaneously. A typical disk channel consists of main memory, the channel control in the MBox, one RH20 Massbus controller, and up to eight mass storage drives. Each mass storage drive implemented on a given channel, is connected to the same RH20 Massbus controller. The controller is connected to the EBox via the asynchronous EBus, which allows the EBox to issue control and data transfer commands to the controller and the associated drives. The controller is also connected to the MBox via the CBus. This path is the synchronous data path that allows the controller to access memory via the MBox channel control without having to use the EBus and the EBox. This configuration lets the EBox perform computation and execute direct I/O operations to other controllers and devices while the channels are executing a data block transfer. Memory fetch and store operations can also be performed by the EBox while the channels are busy executing a block transfer, provided the cache is implemented. Otherwise, the EBox must compete with the channels for core cycles.

Each block transfer between main memory and a mass storage drive must be initiated by the EBox. This is done by the EBox by setting up the channel command list in main memory and by executing DATAO instructions to transfer one or more command words and other control information to a specific controller. The channel command list serves as a control program for executing the block transfer to/from a series of contiguous segments of main memory. The control information and commands specify one particular drive of those connected to the controller, a physical starting block address, a block count, a command function (read or write) code, and other control bits, such as reset command list pointer and/or store status, if required.

As soon as the block address and command are transferred to the drive, which is done automatically as soon as the drive is not busy, the controller informs both the channel control and the drive to start the block transfer. To get ready, the channel control fetches the first word in the channel command list. If the block transfer is a channel read operation (NOT CTOM) that is specified by the RH20, the channel control also fetches at least two words of data from the locations specified by the address field of the CCW. This is done because the controller has a two word data buffer for which words will be requested as soon as the channel control is ready. The drive, on the other hand, will stay dormant until it reaches the specified block address. When the block is reached, the drive, the controller, and the channel control will operate together under the control of the channel command list and the block counter to transfer the block(s) of data. Both the controller and the channel control contain data buffers to normalize the transfer speeds of the different components in the channel path. As the buffers are filled or emptied, additional requests will be made via the buses and interfaces in the path to keep the data moving until the entire block transfer is done. The transfer is done when the channel control fetches a HALT CCW, when it is executing a LAST DATA XFER CCW and the WC field of that CCW has reached zero, or when the block counter in the controller overflows.

The RH20 controller maintains and updates the block count as the block transfer is executed. Up to 1024 blocks can be specified when the read/write command is issued by the EBox. When the block count overflows, the RH20 interrupts the EBox to inform it that the transfer is done. The RH20 also informs the channel control that the transfer is done.

The channel control logic maintains a status and command list pointer (CLP) word and a channel command word (CCW). These two words are kept in the CCW BUF. To keep track of these words for all the channels, the CCW BUF contains two locations for each of the eight possible channels. The status/CLP word (relative location 1 in the CCW BUF) contains the status of the channel and the address (program counter or CLP) of the next channel command word to be executed. The initial CCW is kept in the EPT. The status bits of word 1 are updated by the channel control when the channel logs out, which occurs on an error condition or when the block transfer is completed if a store operation was specified when the transfer was initiated by the EBox. The channel control logs out by writing the appropriate status/CLP and CCW words into the preassigned EPT locations. The CCW (relative location 0 in the CCW BUF) contains the current channel command word. This word specifies the operation (instruction) the channel control is to perform. The word contains a 3-bit op code field that specifies one of the following six operations.

1. Op code 0 specifies a halt operation.
2. Op code 2_8 specifies a jump operation.
3. Op code 4_8 specifies a forward data transfer operation.
4. Op code 5_8 specifies a reverse data transfer operation.
5. Op code 6_8 specifies a forward last data transfer operation.
6. Op code 7_8 specifies a reverse last data transfer operation.

After being started, the channel control will continue to fetch CCW until it gets a HALT CCW or a DATA TRANSFER CCW. In response to a HALT CCW, the channel control will simply halt; and it may cause the channel control to log out, if so specified, when the transfer was initiated. In response to a JUMP, the channel control will simply fetch another CCW. The location of the next CCW is specified by the contents of the ADR field of the JUMP CCW. In response to a DATA TRANSFER, the channel control will transfer the number of words specified by the WC field from/to the starting address specified by the ADR field.

3.4.1 TX02 Tape Control Unit

The TX02 connects to the Massbus via the DX20f. The TX02 tape control unit (TCU) contains the logic required to operate the TU72-E series tape units. The basic TCU can perform tape operations at 6250 or 1600 bits/inch.

During 6250 bits/inch write operations, data is assembled in data groups, translated, formatted into storage groups, and recorded. The TCU will, at specified intervals, insert control characters for error correction, record detection, data check, and read circuit resynchronization. A complete readback check of the write data is performed to verify proper recording.

The 6250 bits/inch read operations convert GCR data to standard characters, which are then transmitted as bytes to the channel interface. Errors that can be corrected are determined and corrected while running.

Control commands are included for rewinding, unloading, spacing, erasing, tape marking, status recording, and diagnostic assistance.

The features available on the TX02 tape control unit are described in the following paragraphs.

3.4.1.1 Nine-Track NRZI – Nine-track NRZI is like the nine-track features of the TX02, with tapes written and read in 800 bits/inch NRZI. For proper operation, a TU70 tape unit and dual density is required.

3.4.1.2 Seven-Track NRZI – The seven-track NRZI feature for the TX02 compares with the seven-track feature of the TX01 TCU. Seven-track operation can be at 800, 556, or 200 bits/inch on a seven-track TU70 tape unit.

The seven-track feature also includes the data convert and data translate functions. Data convert causes four tape characters (24 bits) to be written for every three characters (24 bits) transmitted across the channel. Data translate causes the translation of each character to a 6-bit binary character. There are 64 possible character combinations in translate mode.

NOTE

Data convert, data translate, density, and parity functions are started by mode set 1 commands. Mode set 1 commands can be used at any time; therefore, they must be issued under strict control. Incorrect use may result in tapes being written at various settings for density, parity, and so on within the same tape. Use of data convert decreases the operating data rate by 25 percent.

3.4.1.3 Two-Channel Switch – TX03 – The addition of this feature permits a second channel to access the TX02. The two channels may be from the same or separate CPUs, and use of the two-channel switch can be under manual or program control.

3.4.1.4 2 × 8 Tape Switch – TX05 – This feature permits two tape control units to access any one of sixteen tape drives. The tape switch feature allows tape units to be dynamically switched among the control units. Any or all units may be rendered inaccessible to a given control unit or control units by switches located on the TCU operator panel.

Those TCUs marked "Switch" contain the 2x, 3x, or 4x switching feature. The "Remotes" contain communication paths to the switching circuitry and therefore do not need the switch feature. In a 4x configuration, if four drives are selected via four different control units, data may be transmitted simultaneously on all four paths.

3.4.2 TU72 Tape Units

The TU72 tape units are attached to the TX02 tape control units in configurations of from one to sixteen. Tape unit models may be intermixed within the same string provided that the TCUs are properly featured. TU70 series tape units may also be intermixed with TU72 series tape units in the subsystem. Each TCU

has power and signal connections for eight tape units. The address of the individual tape unit is determined by the port to which the tape unit signal cable is attached and by pluggable jumpers within the control unit.

The TU72 tape unit is a self-loading, single capstan unit. The features that are standard on all TU72 model tape units are described in the following paragraphs.

3.4.2.1 Self-Loading – Tape mounts in the file reel position will be automatically threaded to the machine reel, then loaded in the columns, and the beginning-of-tape (BOT) label brought to ready position.

3.4.2.2 Automatic Reel Hub – Operator action is not necessary to secure the file reel to the reel hub. The reel hub is automatically started by pressing the LOAD/REWIND push button on the operator panel.

3.4.2.3 Capstan – All in-column tape motion is controlled by a single capstan. Contact between tape and capstan is restricted to a nonoxide surface, thereby minimizing recording surface damage.

3.4.2.4 Dynamic Amplitude Control – During 6250/1600 read operations, the read bus is continuously monitored to ensure correct signal amplitude. Decreases or increases beyond defined limits will start the dynamic amplitude control (DAC) circuitry, which will adjust read amplifier gain accordingly. Continuous monitoring and adjustment of individual records by the DAC feature improves the unit's ability to handle variations in signal strength because of differences in media.

3.4.2.5 Backside Tape Cleaner – The backside tape cleaner removes loose particles from the nonoxide tape surface to prevent contamination of the recording surface when the tape is rewound. Cleaners are located at the top of each vacuum column and are operational whenever tape is in the columns.

3.4.2.6 Tape Storage Pocket – The pocket is located to the right of the vacuum column door and provides storage for two cartridges or three open reels.

3.4.2.7 Power Window – Window operation is automatic during load and unload operations. The window may be opened at any time by putting the tape unit in the not ready condition and pressing the HUB/WINDOW-UP push button.

3.4.2.8 Data Density Option – Any TU72 model tape unit can be equipped to perform either 1600 or 6250 bits/inch operations.

3.4.3 TU77 Magnetic Tape

The TU77 is a nine-track, freestanding, magnetic tape storage system that connects to the processor via the Massbus controller/adapter. The TU77 was designed to operate at high speeds in high usage environments in a wide range of applications, such as fast disk-to-tape backup, data acquisition, heavy transaction processing/journaling, and tape interchange.

The TU77 has the following features.

- Program-selectable recording at 1600 bits/inch (PE) or 800 bits/inch CNRZI)
- Read/write speed of 125 inches/s
- Automatic tape threading of 10.5 inch tape reels as well as IBM Easy Load No. 1 and No. 2™ type tape cartridges
- Automatic density select on a read operation (1600 bits/inch or 800 bits/inch)
- Expandable up to a total of four tape transports per subsystem
- Direct memory access (DMA) data transfers
- Vacuum column tape buffering

3.4.4 TU78 Magnetic Tape

The TU78 magnetic tape subsystem consists of a master TU78 tape transport, up to three slave TU78 transports, and an RH20 Massbus controller. The master transport consists of a TU78 transport and a TM78 formatter contained in a single cabinet. The RH20 Massbus controller is ordered separately to avoid redundancy for systems previously configured with the necessary RH20 channels.

Up to two master TU78 transports may be connected to a single RH20 controller. The master TU78 may only be connected to a high-speed RH20 (controller number 0, 1, 2, or 3).

The TU78 subsystem is supported by the TOPS-20 operating system, which can support up to 16 tape transports.

The TU78 has the following features.

- Program-selectable recording at 6250 bits/inch (GCR) or 1600 bits/inch (PE)
- Storage capacity of 145 megabytes (8 K bytes blocks at 6250 bits/inch on a 2400 ft reel)
- Transfer rate of 781 K bytes/s (at 6250 bits/inch)
- Dual-access capability
- 125 inches/s read and write tape speed
- Automatic density select on a read operation (6250 or 11600 bits/inch)
- Automatic tape threading
- Two-track "on-the-fly" error correction at 6250 bits/inch
- Internal microprocessor-controlled diagnostics
- Front access to all modules

3.4.5 RP06 Disk Pack Drive

The RP06 disk pack drive is a direct-access, single-head-per-surface drive, with 815 cylinders, 18 read/write heads, and 1 servo head. The RP06 has a removable pack, and has dual-ports as configured on the KL10 system. These features fill two requirements for diagnosing and operating the KL10 system.

First, the front-end processor performs CPU diagnostic testing, which requires a mounted diagnostic pack called the KLAD pack (KL advanced diagnosis).

The KL10 system needs the front-end processor because it brings up the operating system and runs front-end task files used for the CPU. To expedite information needed to perform these operations, one RP06 port is connected to the RH11 Massbus controller, controlled by the front-end processor over the Unibus. This port is used only by the front-end processor and not as a system device.

The other port on the RP06 drive is connected to an RH20 Massbus and is supported by the system.

The front-end processor files are not accessible by the CPU, and the CPU files are not accessible by the front-end processor.

3.4.6 RP07 Disk Drive

The RP07 disk drive is a fixed-media, direct-access, mass-storage device attached to an RH20 Massbus controller. The drive has a moving carriage that positions 32 read/write and 1 servo head over 17 disk surfaces. These surfaces have 630 customer-usable cylinders and 2 FE cylinders. The Massbus can have a maximum of eight RP07s attached to it. There is one exception to this: one Massbus requires a dual-port RP06.

3.4.7 RP20 Disk Subsystem

The RP20 is a fixed-media, dual-spindle disk subsystem with a capacity of 1200 megabytes unformatted. It consists of a disk control unit (DCU) and a master drive. Up to three slave drives may be added to each master drive. The RP20 is interfaced to a DECsystem-10 or DECSYSTEM-20 (except for 2020) through a DX20 Massbus-to-channel adapter.

The disk subsystem can be configured as either single-port, dual-port, or dual-port-dual-CPU. When formatted for TOPS-10, the RP20 has a capacity of 107.5 megawords (36-bit). Under TOPS-20, the capacity is 103.2 megawords.

3.4.8 DX20 MASSBUS-To-IBM Adapter

The DX20 is a microprocessor-based I/O adapter that interfaces the Massbus-to-IBM channel pulse code modulation (PCM) bus. Microcode allows the DX20 to emulate selector channel operation for PCM disk units. In response to commands issued by the host CPU over the control portion of the Massbus, the DX20 executes the appropriate PCM channel protocol to effect the requested action. A high-speed data path with formatting capabilities facilitates the transport of data between the buses. Tapes and disks cannot be connected to the same DX20 because of the differences between tape and disk functional microcode.

CHAPTER 4
THE SOFTWARE

4.1 INTRODUCTION

The DECsystem-10/DECSYSTEM-20 provides the following features.

1. A powerful batch system that is completely compatible with the interactive timesharing mode. The batch system offers an extensive complement of languages including a comprehensive COBOL. This allows commercial applications to be run efficiently and smoothly with a minimum of personnel training.

2. Many kinds of people can use a system with interactive timesharing. The system features a flexible command language that includes many tutorials to help the beginning user; but it also allows the expert to use it quickly and efficiently. It supports a full range of languages and interactive on-line editing and debugging facilities.

3. A secure file system that needs no preallocation or special formatting, that encourages file transport between languages, and that permits controlled sharing of data and programs.

4. A state-of-the-art central processor that supports a demand paged virtual memory system, therefore providing all users with the power and dependability needed for their applications.

5. Extensive language capabilities including ALGOL, APL, BASIC, COBOL, FORTRAN, and MACRO, with such support systems as the Data Base Management System (DBMS) and a powerful sort facility.

4.2 TOPS-10/20 OPERATING SYSTEM

Certain common problems are faced by most computer users. Data must be organized, up-to-date, and accessible but secure. Monthly, quarterly, and yearly reports must be generated, so the data base must be simple to maintain but flexible enough to meet unforeseen needs.

Consider the following inventory control example. A small manufacturing company maintains an inventory of finished goods. A combination of actual and forecast orders determines the quantity on hand at any one time. Control of this quantity is critical because too much means working capital is tied up unnecessarily. Too little means lost customer orders. The key to tight inventory management is accurate, timely information. Order entries, status inquiries, customer and vendor lists, forecast data — these are but a few of the types of information that must be handled by the inventory management system if it is to be effective.

The TOPS-10/20 operating system has been designed to handle this range in a simple, flexible, and efficient way. It includes:

1. A flexible file system
2. A common command language and processor for both timesharing and batch modes
3. The monitor
4. Separate front-end software
5. System reporting and control features.

4.2.1 The File System

The TOPS-10/20 file system provides controlled, secure, flexible access to large quantities of data. Major design features are:

1. File security
2. Automatic but controlled sharing of data
3. Convenient read and write access methods — by characters, strings, and blocks.

4.2.1.1 Files – Files are collections of data that are identified by a file name and a qualifier such as SALES.DAT, where SALES is the file name and .DAT is the qualifier. Because programs are also files, applications may be grouped by giving the same name to a data file and the program that processes it, while distinguishing between them by the qualifier. For example, SALES.DAT is a data file for the program SALES.EXE.

Device independence is provided by allowing the user to specify a logical device name within his program. At execution time he may specify the physical device or allow the system to supply a default. Normally the default device is a disk.

All files are assigned protection codes for each of three classes of users: the owner, users within a common group or project, and all other system users. Members of each class may be granted none, some, or all of the following privileges.

1. Read access
2. Write access
3. Authorization to append data to the end of the file
4. Acknowledgement that the file appears when listing the directory

When a file is created, the owner may either assign a protection code or take the system default protection. The owner is permitted to change the protection code at any time.

A file descriptor block that completely describes the file's characteristics to the system is associated with each file. The information given includes the file's protection code, who the owner is, who last wrote in it, the date created, the date last changed, and its size. This information block points to an address table that contains pointers to the location of each block of the file.

4.2.1.2 PNNs – The PPN (project-programmer number) identifies the user's file storage area on the structure. It is made up of two octal numbers separated by a comma. The first number is the project number and the second number is the programmer number. The project number identifies a specific group of users whose programmer numbers all are part of that group.

4.2.1.3 Directory Files for TOPS-10 –

Master File Directory
A structure contains two types of files: your data files and the directory files, which tell the operating system where to find the data files. The directory files are the master file directory (MFD), user file directories (UFDs), and subfile directories (SFDs).

Each structure has one MFD. When a disk pack is mounted, TOPS-10 reads the home blocks and finds the MFD, which makes the complete file system on the structure available. Home blocks exist on every disk pack. They identify the structure to which the unit belongs, they contain parameters about the unit and structure, and they contain a pointer to a retrieval information block (RIB) for the MFD. The RIB points to the MFD.

The entries contained in the MFD are the names of all the UFDs on the structure. The MFD points to a RIB, which points to the UFD.

User File Directory
Each user with access to a structure has a UFD that contains a list of all the files in that user's directory. The UFD is synonymous with the PPN. The UFD points to the first RIB for each file.

Subfile Directory
An SFD is a subgroup of files within a specific UFD. Subfile directories are logically separated from each other in the way that PPN UFDs are separated from each other. You may have up to five nested levels of SFDs per UFD. When you are working in a subdirectory, all defaults and commands act upon the files in that subdirectory unless you state otherwise with a different file specification.

4.2.1.4 Directories for TOPS-20 – The system maintains a file directory for each user. It contains the names of all the user's files, pointers to the user's file information blocks, and general information about the user. This information includes the user's password, his privileges, his disk space quota, how much disk space he has used, and the system default protection code. All directories are themselves named files, and information about them is contained in a master directory called the root directory. There are two copies of this root directory to increase file system integrity.

As files in the system are protected by specific access codes, so each directory has its own protection code. In addition, each directory has a password that controls who may get or gain owner access rights. This permits a flexible but protected system where a user who has password privileges may become the temporary owner of other directories.

4.2.1.5 Groups on TOPS-20 – A group is a set of cooperating users created by the system administrator. It is the second class of users in the file protection design previously discussed. The directory for each user contains two group membership lists: the list of groups that may have group level access to the files and the list of groups to which the owner of this file belongs.

If files in a directory are needed by a user other than the owner, you can assign a directory group number to it. Each directory on the system may be a member of one or more directory groups.

Once directory groups are created, you select the users who may need to access specific directories. You assign them one or more numbers, user group numbers, that indicate the directory groups they can access. (Group privileges apply to disk files only; they do not apply to tape files.)

For example, if you are a member of user groups 200 and 300 you have group privileges to directories with directory group numbers 200 and 300. If your directory is in group 299, then only users with a user group number of 200 have group privileges to your directory.

4.2.1.6 File Usage – A prime concern for users of a file system is the mechanism that permits controlled sharing of programs and data. The design of TOPS-20 ensures that two users with access to the same program will share it automatically. When either user modifies a page of the program, the system will give him his own copy of the page (unless the two users specify that they also want to share such pages). This makes memory use far more efficient because many users may share a single physical copy of any program. For example, the DECSYSTEM-20 COBOL compiler has common code that is shared by all users, but each user has his own data area for his specific program statements.

Although program sharing is important, data sharing is even more important when it comes to building effective multiuser data base systems, specifically when on-line interaction is needed. When more than one user is accessing a specific file at any one time, there is a chance of confusion if one user is changing the file while another is reading it. For example, imagine the possible effects if one user were updating a file at the same time it was being used to produce a quarterly report.

To prevent this confusion the system offers several methods to share access.

1. Shared reading (one or more users reading, none writing)

2. Writing (one user writing, none reading)

3. Shared updating (one or more users reading and writing at the same time, using the DECSYSTEM-20 queueing facility for coordination)

4. Restricted updating (one user writing, any number reading)

For more protection, if a user has opened a file for reading only or is executing a shared program and tries to write in a page that other users believe is not changing, the user who wants to write is given a private copy of the page and the write proceeds. This copy-on-write facility is automatic.

The system also helps the programmer by providing a large and flexible group of file access calls. Through these calls, the user can address characters, strings of data, complete pages, or preedited input text. Sequential and random-access calls enable the programmer to build any access method he wants; for example, COBOL's powerful application-oriented indexed sequential access method (ISAM).

4.2.2 The Command Processor

The DECSYSTEM-20 command processor makes the system resources available to the terminal user. It runs in user mode, which has the advantage of protecting the monitor and allowing installations to write special application-tailored command processors.

Simple English verbs are used to perform system functions. Each command starts with a keyword and may be followed by parameters to completely describe the function requested by the user. Prompting is available to make the DECSYSTEM-20 easy for new users and to provide a readable typescript for later reference.

If only a part of a command is typed followed by the escape key (ESC), a nonprinting character, the command processor will try to recognize the abbreviation and print the rest of the command. This will be followed by a guide word, which requests the next input parameter needed. For example, the command interaction to make a copy of a file would be:

COPY $ (from) **OLD.FIL** $ (to) **NEW.FIL**

In the example above, the user typed the uppercase, boldface letters and the escape key (represented by $). The command processor responds by completing the remaining part of the command name, followed by a

guide word (shown in parentheses), which logically leads to the next field needed to complete the command. As can be seen, the combination of command completion and the automatic typing of prompting makes the command language very easy to use.

If a command is typed erroneously, the command processor responds with an error message that starts with a question mark. The user must then enter the command again. If the user needs more help, he can type ?, which results in the typing of a list of all commands beginning with the characters he has typed. More information that describes command functions is also available interactively, providing available on-line documentation.

The user may select to type only enough of a command to uniquely distinguish it from any other command, followed by a space. For most commands only the first three letters are enough to uniquely identify one command from another. The abbreviated mode of the copy command as described above would be:

```
COP OLD.FIL NEW.FIL
```

Additional flexibility is available by allowing use of both prompting and abbreviations in the same command line.

In addition to prompting and help information, there is a pair of commands to save a current program address space and start another. For example, a user starts a program and later finds that he forgot to supply a necessary file. Instead of having to start over, he can return to command level and save the current program address space. Next he creates the file, restores the saved address space, and continues the original program.

TOPS-20 has one user command language that is used in both batch and timesharing modes. Therefore a timesharing user can make a job from the same statements he would use under timesharing, although with different options, and then submit the job to run under batch. Concurrent with the processing of that job, he may then continue interactive work under timesharing.

4.2.2.1 Timesharing – The DECsystem-10/DECSYSTEM-20 takes maximum advantage of system throughput capabilities by allowing many independent users to share system facilities at the same time. Because of the conversational, rapid-response nature of DECsystem-10/DECSYSTEM-20 timesharing, it is well suited for tasks ranging from student homework problems to data processing applications such as order entry and status inquiries and general data base management. A wide range of CRT and hard copy terminals are supported and are capable of operating at speeds of 10 to 240 characters per second.

Terminal users can be found at the computer center or at remote locations connected to the computer center by communications. Timesharing on the DECsystem-10/DECSYSTEM-20 is designed in such a way that the command language, input/output processing, file processing, and process scheduling are independent of the programming language being used. This allows timesharing users to concurrently compile and execute programs using a wide range of languages such as COBOL, FORTRAN, BASIC, APL, and MACRO.

Extensive terminal handling capability provides the terminal user with access to all system resources. From his terminal, the user controls the running of a program; creates, edits and deletes files; and compiles, executes, and debugs programs.

In addition, the user can request assignment of a peripheral device, such as magnetic tape, for exclusive use. When the request for assignment is received, the operating system verifies that the device is available to this user and if so, the user is granted its private use until he relinquishes it. The operator can control and terminate such assignments as needed.

4.2.2.2 Batch – Batch software enables the DECSYSTEM-20 to execute batch jobs concurrently with timesharing jobs. The batch user has two ways in which he enters his job into the batch subsystem: card mode, wherein he punches his job on cards, inserts necessary control cards, and then leaves the job for an operator to run; terminal mode, in which he creates a batch control file containing normal terminal commands and then submits it to the batch subsystem from his terminal. Therefore a user can debug a program while timesharing and then submit it to batch either from the terminal or from cards.

The batch software has a series of programs: the input spooler, the batch controller, the centralized queue manager and task scheduler, and the output spoolers.

The input spooler is responsible for reading from the input device and for requesting the queue manager to enter jobs into the batch controller's input queue. Although the input spooler is oriented toward card reader input, disk and magnetic tape can also be handled. The input data is then separated according to the control commands in the input deck and placed into disk files, either user data files or the batch controller's control file, for subsequent processing. In addition, the input spooler creates the job's log file and enters a report of its processing of the job, with a record of any operator intervention during its processing. The log file is part of the standard output that a user receives when his job terminates.

The batch controller processes a batch job when the queue manager passes it a request from the batch queue. The control file previously created by the input spooler is read by the batch controller, and commands from this file are passed to the command language processor for action. During the processing of the job and the control file, the batch controller records job processing history in the log file as a record for the user.

The queue manager is responsible for scheduling jobs and maintaining both the batch controller's input queue and the output spooling queues. A job is scheduled to run according to external priorities, processing time limits, and parameters specified by the user for his job, such as start and deadline time limits for program execution. The queue manager makes an entry for the job in the batch input queue based on its priorities. After the job is completed, the queue manager schedules it for output by placing an entry in an output queue. When the output is finished, the job's entry in the output queue is deleted by the queue manager.

The output spooling program improves system throughput by allowing the output from a job to be written temporarily on the disk for later transfer instead of being written immediately on the printer. The log file and all job output are placed by the queue manager into one or more output queues to wait for printing. When the printer is available, the output is then processed by the line printer spooler. The line printer spooler also includes such facilities as special forms control and character sets, multiple copies, and accounting information. The system administrator may request a guaranteed percentage of CPU time for batch jobs.

Normal operating functions performed by programs in the batch system need little or no operator intervention; however, the operator can exercise a large amount of control if necessary. He can specify the system resources to be dedicated to batch processing. He can also limit the number of programs as well as the core and processor time for individual programs. He can stop a job at any point, requeue it, and then change its priorities. By examining the system queues, he can determine the status of all batch jobs. In addition, the batch system can communicate information to the operator and record a disk log of all messages printed on the operator's console. All operator intervention during the running of the input spooler, the batch controller, and the output spooler causes a message to be written in the user's log file, as well as in the operator's log file, for later analysis.

Although jobs are entered sequentially into the batch system, they are not necessarily run in the order in which they are read because of priorities either set by the user in a batch control command or computed

by the queue manager when determining the scheduling of jobs. Sometimes, the user may want to submit jobs that must be executed in a specific order; in other words, the execution of one job may depend on another. To make sure that such jobs are executed in the correct order, the user may specify many initial dependency counts in control command of the dependent jobs. Control commands in each job on which the dependent jobs depend must in turn decrement the count. When the count for a dependent job becomes zero, it becomes eligible to be executed.

The batch system allows the user great flexibility. The input spooler normally reads from the card reader but can read from magnetic tape or disk. In the command string, the user can include switches to define the operation and set priorities and limits on core memory and processor time.

The user can control handling of error conditions by including special commands in his job. These commands, copied into the control file by the input spooler, specify the action to be taken when a program gets a fatal error; for example, skip to the next program or transfer to a special user-written error handling routine.

Although the batch system allows a large number of parameters to be specified, it is capable of operating with very few user-defined values. Therefore, if a user wants to specify the normal choice for the installation, he may do so by omitting the parameters. These defaults can be modified by the individual installations.

The batch system may run multiple batch jobs concurrently, a feature that enhances both system and user throughput because the user can break his job up into several parallel job steps. Periodic reports, such as year-to-date sales figures, may be efficiently run under batch at the same time order inquiries are being processed under timesharing.

4.2.3 The Monitor

The TOPS-20 Monitor is divided into two sections: the resident section and the nonresident section.

4.2.3.1 Resident – The resident section contains the system control modules including:

1. Scheduler-processor scheduler. Defines storage for all system tables and variables for processes and the balance set. Contains routines for balance set and process management.

2. Disk routines necessary for paging. Includes initialization for the Core Status Table and Shared Pages Table; routines for setting, changing, and reading all process tables and the index block and core management routines.

3. Basic interrupt and system call handlers. Contains routines for dispatching device interrupts, central processor interrupts, and system accounting meters and the initial dispatch and return code for the JSYS monitor calls.

4. Peripheral device error reporting and recovery routines. These increase system availability by recording exactly what error conditions occurred, then trying specific recovery procedures in order to continue processing. Errors recovered from in this way are completely transparent to the users. Complex on-line diagnostic capabilities are provided in order to diagnose problems while the system is in normal operation.

5. Monitor tables.

These routines are mapped through the operating system process table and the code is write-protected.

4.2.3.2 Nonresident – The swappable section of the operating system is demand-paged into available memory (similar to user programs) so that only those functions in use need memory. This allows a high capability operating system to operate in constrained memory environments, which was not possible before. The swappable or nonresident section of the operating system contains:

1. The operating system call service routines (called JSYSs for Jump to SYStem) provide the system support programs with an extensive sharable library of functions to simplify device I/O, interprocess communication, cooperation, and control, terminal interaction. They generally have an extensive, flexible, easy-to-use set of functions.

2. The job and process status in pageable tables. The job status block (JSB) and process status block (PBS) contain data needed in order to run a process. Other data pertaining to the files a user is accessing also is paged with the process. Finally, the process map is itself a page necessary to run a process.

3. The file service user interface is a named, multiuser file system with extensive user security features, cooperative sharing techniques, extreme dependability, and efficient operation.

4.2.3.3 Communication with the Monitor for TOPS-20 – The JSYS handler is responsible for accepting requests for services provided by the operating system. Support programs, such as the COBOL compiler or user programs, issue these requests through monitor calls known as JSYSs. All JSYSs are reentrant in order to maximize system dependability by isolating JSYS processing from operating system processing. The operating system services include communicating with I/O devices, including terminals; getting or changing status information about either the computing system as it applies to the user process, such as controlling execution by creating a new process or interrupting the user program when a predefined hardware or software event occurs; and communicating and transferring control between processes. There are also special privileged capabilities that are limited to authorized users. Authorization is granted by the system installation manager, and the operating system makes sure that only the appropriate privileged users can execute special JSYSs.

Contained in the calling program is a JSYS operation code that, when executed, causes the hardware to transfer control to the JSYS handler for processing. The processing routine gets arguments from the program that completely define the operation requested. After the JSYS request has been processed, control is returned to the calling program along with any indication of error conditions.

4.2.4 The Front-End Software (RSX20-F)

4.2.4.1 Introduction – The purpose of the front-end computer is to increase the amount of work that the central processor can do for the user by moving certain overhead tasks to the front-end processor. To do this, it has four major processors.

1. Console processor
2. Communication processor
3. Peripheral processor
4. Diagnostic/maintenance processor

4.2.4.2 Console Processor – Two major functions of the console processor are system initialization and system operator communications. The operator need only push a button to start the following automatic program. The front-end processor loads and verifies the microcode, configures and interleaves memory, and loads and starts execution of the central processor bootstrap program from a device specified by the operator.

There is a hardware confidence check of not only the microcode but also disk and memory. Any problems that are found are reported to the operator in clear English error messages.

The operator may request a memory configuration list indicating which memory controller is on-line, what is the highest memory address configured, and how the memory is interleaved.

All normal console capabilities, such as lights, start, stop, reset, and load, are provided by the front-end processor. In addition, there is a straightforward English operator command language that allows the beginner to communicate with the system.

System protection is extended by separating the privileges needed by the operator for normal system operations from the full-scale privileges needed by the system administrator.

4.2.4.3 Communication Processor – The front-end processor operates as an intelligent data link and buffer for the interactive terminals, thereby relieving the central processor of this overhead.

The front-end processor buffers the characters received and interrupts the central processor only when it has a group of characters to send. The central processor also sends groups of characters for terminal output to the front-end processor, increasing the efficiency of both processors. Another feature is the programmable terminal speed setting capability, which allows terminal speeds to be changed dynamically at command or program level.

4.2.4.4 Peripheral Processor – The unit-record spooling programs in TOPS-10/20 communicate with the front-end processor by the same buffering mechanism described for the communication processor. Both the central processor and the front-end processor maintain buffers for data and interrupt only once per buffer transmission, therefore greatly increasing the amount of useful work each processor can do.

4.2.4.5 Diagnostic/Maintenance Processor – An important part of the DECsystem-10/DECSYSTEM-20 is a remote diagnosis capability (KLINIK) that lowers mean time to repair (MTTR) and increases system availability to the customer.

This capability provides the facility for Field Service to isolate most system failures to the correct option or subsystem — maybe even to the failing component — before leaving the local office. By running his diagnostic tests using telephone dial-up facilities, the Field Service representative selects which spares, test equipment, and so on should go with him on the call. These tests may be run during either general timesharing or standalone operation even when the main CPU is completely inoperative.

Also, this facility provides help in system reconfiguration to remove defective devices from the available stock of resources, thereby allowing less time for repair. Daily remote access to the system performance and error file provides an automatic preventive maintenance technique, which decreases the requirement for standalone preventive maintenance periods.

Total system security is maintained because the on-site system operator must enable the communication link, and only the on-site personnel may specify the specific password to the system each time remote diagnosis is used. Time limits for system access may also be imposed, and the remote link may never operate at a higher privilege level for commands other than the local console terminal. Also, all input and output to the diagnostic link is copied to the local terminal so that on-site personnel have a record of all steps taken.

4.2.5 The System Reporting and Control Facilities

One of the major system design goals for the DECSYSTEM-20 has been to provide a high degree of control and a simple to operate system. Areas of specific interest include the following.

4.2.5.1 Accounting – Files are maintained by the system to provide information on resource usage (CPU, disk space, and so on). This provides the basis for charging individual users as well as providing a year-to-date summary and usage statistics.

4.2.5.2 System Control – The system administrator can control access rights and privileges to the system in general, as well as batch system usage. One important feature here is the hierarchical level of privileges whereby the operator can function at a lower level of system access than the system administrator. This helps protect the system from an inexperienced operator. In addition, the separation of the highest level privileges from normal operation privileges gives a higher degree of system production.

The system administrator sets resource allocation rules: disk quotas for each individual user to control disk usage, the amount of resources for batch and timesharing, and the maximum size of spooled output files.

4.2.5.3 System Generation – There is no complex system generation procedure necessary. Digital provides the monitor, which can simply be initialized and started as soon as the hardware has been installed. In addition, initialization routines will dynamically reconfigure the system should any device be unavailable. There is a parameter file provided to allow the user to change such things as maximum number of users allowed to access the system at the same time. Normally, no change will be necessary.

4.2.5.4 Error Reporting – The TOPS-10/20 operating system includes an extensive error detection and recovery package to ensure maximum system availability to the user. It also includes complete error recording facilities for use by Digital maintenance engineers and support personnel and the customer operating department.

When an error is detected, the TOPS-10/20 monitor collects all pertinent hardware and software information including whether the error is recoverable or not, and the recovery procedure is invoked and adds this information to a disk file for storage. Later, a user mode program may be run to read this file and generate reports about the complete system or individual items.

Also, significant operational events such as system reloads and changes in system configuration are recorded to assist the operations department in monitoring system performance.

On a periodic basis, maintenance engineers collect summary information from this error file and can detect early indications of possible problems. Detailed reports about specific errors often allow diagnosis of a problem without having to run exhaustive diagnostics standalone. In other cases, user-mode diagnostics will identify failing components after the error reports have pointed out the failing option. In this way, a system is efficiently maintained with minimum interruption to the customer's operating schedule.

The error file can be saved on magnetic tape to provide a complete history of system operation and can identify slowly degrading parts of a system long before serious problems are caused.

This facility of TOPS-10/20 increases system availability by decreasing mean time to repair.

4.2.5.5 Backup Facilities – User's data and programs become the most valuable part of any installation; it is essential to protect them against unforeseen human errors as well as hardware or software errors. This protection is provided by a facility that can be used to save all files or selected files and that can later be used to reload any files that are needed.

4.3 THE PROCESS

4.3.1 Structure

A process is an independent task or job that is capable of being run by the monitor. By this definition we can see that the terms program, task, and process may be used interchangeably. In the TOPS-20 operating system, however, processes have attributes that are very useful in solving complex problems.

A process has its own environment including a 256 K word virtual address space that is divided into 512 pages of 512 words each. Core management is totally automatic, and only active process pages need be in memory for execution.

A process may create other processes; the new processes are said to be inferior to the creating process. Two processes that have been created by another process are called parallel processes; such processes can be very useful when there are tasks that can run at the same time as in a reservation system when multiple requests must be handled at the same time. The superior, or creating process has control over its inferior processes and may give or withhold privileges and suspend, continue, or terminate them.

The simplest example of a process is a single task such as a program to solve an engineering problem. It reads in some data, performs calculations on it, then prints an answer and exits. Other applications can be much more complex. For example, an inventory control system may have one task to receive requests from terminals and send them to subtasks for processing. These processes in turn may want to start other tasks to write update reports as a data base is updated. Figure 4-1 shows such an application.

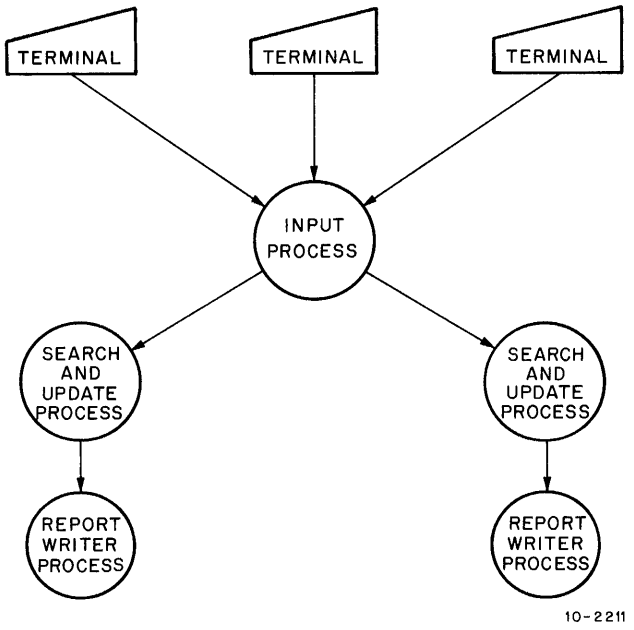


Figure 4-1 Parallel and Inferior Processes

It is important to stress that all five of these processes can be run at the same time (although they do not have to be), thereby greatly increasing throughput. Response time to the input terminals may be enhanced because the top level process must only receive input requests.

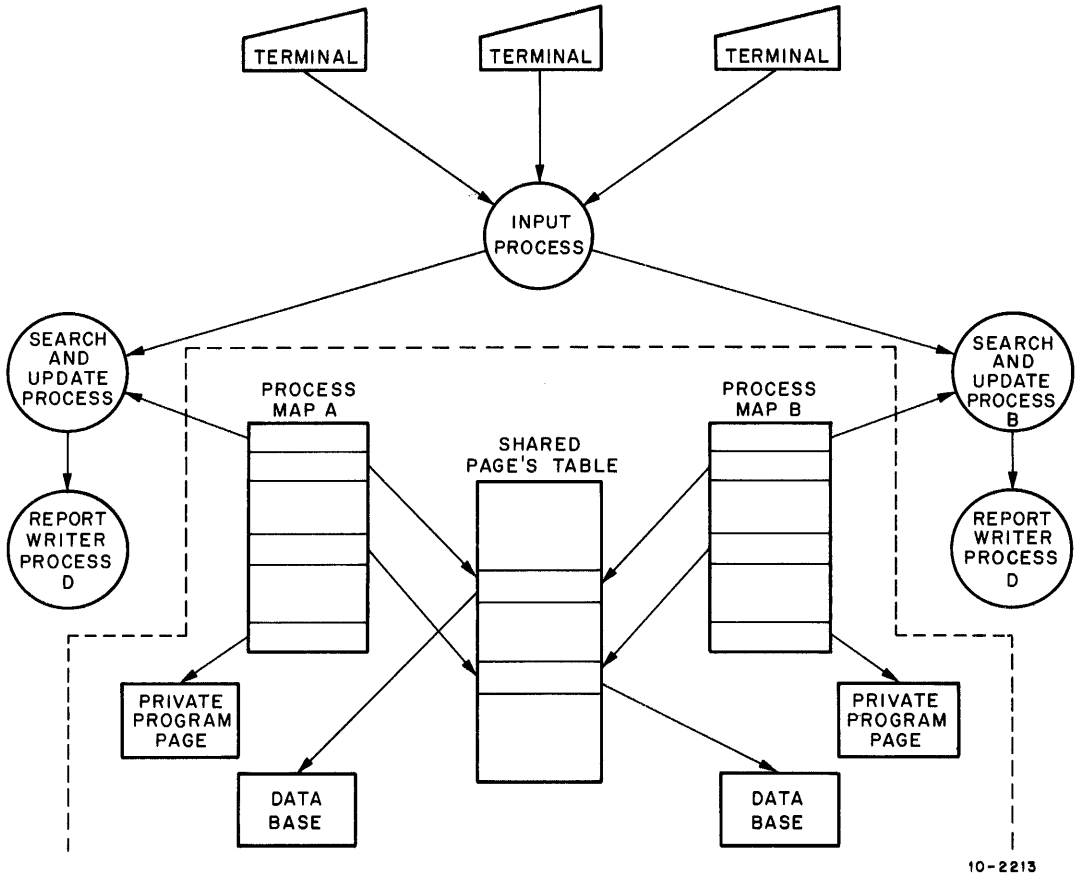


Figure 4-2 Process Mapping

Each process has a one-page process map (Figure 4-2) containing 512 map slots, one for each possible process page in use. Each map slot provides the information needed for the central processor to access instructions or data in the corresponding page of the user's address space. Although there are potentially 512 pages in each process address space, in practice fewer pages are usually used. The process map may be sparse (some slots null).

The Shared Pages Table (SPT) helps page sharing for programs and data by allowing the software to maintain only one pointer to a page that is shared by multiple processes. Both the process map and SPT are interpreted by the hardware as a part of the memory mapping process. In the example shown in Figure 4-2, while only the search and update process maps are shown, each process will have its own map. As the example shows, direct sharing allows equal sharing of data or programs. This sharing is generalized to more than two processes.

4.3.2 Interprocess Communication Facility (IPCF)

Although each process is basically independent, the advantage of a multiprocess job structure is that output of some processes can be taken as input to others. The Interprocess Communication Facility (IPCF) allows such communication among processes. This communication occurs when processes send and receive information in the form of packets (messages). Each sender and receiver has a unique process ID (PID) assigned to it to allow this communication to work.

When one process sends information to another process, it is placed into the receiver's input queue, where it stays until the receiver retrieves it. Instead of regularly checking its input queue, the receiver can enable the software interrupt system to generate an interrupt when information arrives in its queue.

In order to use IPCF, the system administrator must assign a user two quotas that define the number of sends and receives his process may have left at any time. For example, if a user has a send quota of two and has sent two messages, he cannot send any more until at least one has been retrieved by its receiver. A user cannot use the facility if his quotas are zero.

An example of the use of this facility could be the inventory data base update process communication with the report writer process.

4.3.3 Enqueue/Dequeue (ENQ/DEQ)

Although TOPS-20 makes program and data sharing very easy while preventing any user from changing a file that others are using in read-only mode, some more complex applications, such as reservation or inventory systems, need updating of the data base.

Consider the inventory data base used by different departments of a company. When order status inquiries are being processed, users may access the file at the same time because no one is changing any part of the data. However, when someone wants to modify or replace an individual record, that portion of the file should be accessed exclusively by that person. No one wants to access records that are being changed until after the changes are complete.

By using the ENQ/DEQ facility, cooperating processes can make sure that such resources are shared correctly and that one user's changes do not interfere with another user's. Examples of resources that can be controlled by this facility are devices, files, operations on files (READ, WRITE), records, and memory pages.

4.3.4 Software Interrupt System

The software interrupt system provides a mechanism for a process to respond to several types of interrupts generated by other processes, hardware and error conditions, and terminal interactions. It may be enabled by both the user and the system. The system provides 36 software interrupt channels of which 18 have reserved functions. The primary types of interrupts are:

1. Terminal character interrupts
2. IPCF interrupts
3. ENQ/DEQ interrupts
4. Hardware error condition interrupts
5. Interrupt error condition interrupts
6. Program errors such as arithmetic overflow.

There are three interrupt priority levels assignable to each software interrupt channel, therefore providing priority ordering. A call is provided for interrupt dismissal to continue the interrupted code.

Terminal character interrupts are used in the implementation of the command processor and by interactive debugging tools. IPCF interrupts are used in the batch system. ENQ/DEQ interrupts are used by the COBOL system to implement simultaneous file updating by cooperating processes.

4.3.5 Virtual Memory

The TOPS-10/20 software system has been designed for efficient memory allocation and takes into account well-known characteristics of typical programs. One such characteristic is locality; that is, that at any time only certain areas of a process address space are active — those instructions near the current locus of execution and the data referenced by those instructions. Therefore, only a few process pages need to be in memory at any time, and sections not used do not tie up memory.

The system is responsible for balancing the requirements for both memory and central processor time among all users. To do this efficiently, two key concepts are used.

4.3.5.1 Working Set – The working set for a process is the set of pages referenced in a recent interval of time. This interval is a constant and is selected to provide a high chance that memory references in the next interval will be to the same pages and not to pages outside the current set (which causes page faults). However, this constant should minimize the number of pages needed. Often some compromise between all and no pages are selected.

4.3.5.2 Balance Set – The balance set is the set of processes that are most eligible to run and whose working sets fit in core at the same time. This setting of the eligibility of a process is event driven and occurs when the state of the process changes. To ensure good response, interactive processes tend to get a higher eligibility than compute-bound processes. However, if a compute-bound process has been blocked by interactive processes for a long period, its eligibility is changed to allow it to run.

The scheduler regularly defines the balance set. Working set sizes are modified dynamically as a process runs and are regularly monitored by the scheduler to adjust balance set membership. Conditions exist to guarantee a percentage of the processor to a specific user. This guarantee operates both as a ceiling (the maximum if the system is loaded) and a floor (guaranteed minimum) and is controlled by the system administrator.

Paging is on a demand basis. A recently started process may start with no pages in core. As it makes memory references, page faults occur (temporary failures caused by referencing pages that are not in core), and a working set is built up. File input and output is done via demand paging as well; the desired page is mapped into the system or user address space and is then referenced.

The DECsystem-10/DECSYSTEM-20 supports memory management in several important ways through hardware and microcode. Memory mapping and page level access protection are provided, including sharing of pages between processes and efficient context switching. Page status and age information are automatically updated for each core page by the microcode from information that has been set up by the software. This minimizes overhead while maximizing the amount of accuracy of information available to the system for decision making. Therefore intelligent decisions can be made at a very low overhead cost.

4.4 LANGUAGES AND UTILITIES

4.4.1 ALGOL

The algorithmic language, ALGOL, is a scientific language designed for describing computational processes, or algorithms. It is a problem-solving language in which the problem is shown as complete and exact statements of a procedure.

The DECSYSTEM-20 ALGOL system is based on the ALGOL-60 specifications. It has the ALGOL compiler and the ALGOL object time system. The compiler is responsible for reading programs written in the ALGOL language and converting these programs into machine language. The user may specify a parameter at compile time, which produces a sequence numbered listing with cross-referenced symbol tables.

The ALGOL object time system provides special services, including the input/output service for the compiled ALGOL program. Part of the object time system ALGOL library is a set of routines that the user's program can call in order to perform different functions including mathematical functions and string and data transmission routines. These routines are loaded with the user's program when needed; the user need only make a call to them. The remainder of the object time system is responsible for the running of the program and providing services for system resources, such as core allocation and management and assignment of peripheral devices.

Source level, interactive debugging is provided through ALGDDT. The user may stop his program at any point, may examine and change the contents of data locations, may insert connected code, and then continue execution to test his changes.

4.4.2 APL

APL (a programming language) is a concise programming language made for numeric and character data that can be collected into lists and arrays. The conciseness with which APL expressions can be written, greatly increases programmer productivity and allows for very compact and readable code that can be executed in a very efficient way. APL finds applications not only in mathematics and engineering but also in financial modeling and text handling situations.

APL is a completely interactive system with both immediate (desk calculator) and function (program) modes of operation. It includes its own editor as well as debugging tools, tracing of function execution, type out of intermediate values, setting break points, and so on. APL for the DECSYSTEM-20 includes many extensions not normally found in other APL implementations. Some of these features include a flexible file system that supports both ASCII sequential and binary direct-access files, the dyadic format operator, expanded command formations that allow the user to take advantage of the DECSYSTEM-20 file system, the execute operator, and tools for error analysis and recovery. In addition, work space is dynamically variable in size up to 512 K bytes, and the system supports double-precision calculations that allow up to 18 decimal digits of precision.

Finally, APL on the DECSYSTEM-20 is available to all users at all times without the need to run a separate subsystem. This provides extra terminal handling, work space swapping, and so on. These tasks are all done by the TOPS-20 operating system.

4.4.3 BASIC

BASIC is a conversational problem solving language that is well suited for timesharing and easy to learn. It has wide application in the scientific, business, and educational markets and can be used to solve both simple and complex mathematical problems from the user's terminal.

The BASIC user types in computational procedures as a series of numbered statements that have common English terms and standard mathematical notation. After the statements are entered, a run-type command starts the execution of the program and returns the results.

If there are errors during execution, the user, from his terminal, simply changes the line or lines in error by deleting, modifying, or inserting lines.

The beginning user has many facilities at his disposal to help in program creation.

1. Program editing facilities – An existing program or data file can be edited by adding or deleting lines, by renaming it, or by resequencing the line numbers. The user can combine two programs or data files into one and request either a listing of all or part of it on the terminal or a listing of all of it on the high-speed line printer.

2. Help in documentation – Documenting programs by the insertion of statements within procedures helps the user remember needed information at some later date and is invaluable in situations in which the program is shared by other users.

3. As a BASIC user, you type in a computational procedure as a series of numbered statements by using simple common English syntax and familiar mathematical notation. You can solve any problem by taking an hour or so learning the necessary basic commands.

4. Functions – Sometimes, you may want to calculate a function, (for example, the square of a number). Instead of writing a program to calculate this function, BASIC provides functions as part of the language.

More advanced users will want to take advantage of some of the more complex computation and data handling tools which BASIC provides.

- File input – The user may create a named data file using the BASIC editing facilities. This file may now be referenced within a program.
- Output formatting – The user can control the format of his output to the terminal or printer.
- Data access – Data files may be read and written either sequentially or randomly.
- String manipulation – Alphanumeric strings may be read, printed, concatenated, and searched.

4.4.4 COBOL

The common business oriented language, COBOL, is an industry-wide data processing language that is designed for business applications, such as payroll, inventory control, and accounts receivable.

Because COBOL programs are written in terms that are familiar to the business user, he can describe the formats of his data and the processing to be performed in simple English-like statements. Therefore, programmer training is minimal, COBOL programs are self-documenting, and programming of needed applications is done quickly and with ease.

4.4.4.1 Features – Major features include:

1. On-line editing and debugging – The programmer may create and modify the source program from his terminal using an easily learned editing facility. After the program has been submitted to the compiler — again from the terminal — any syntax errors may be immediately corrected using the editor and the program resubmitted. The program listing is sequence numbered with symbols cross-referenced to show when they are defined and where used. COBDDT, the on-line, interactive COBOL debugging package, allows the programmer to check out the program:

   a. By selectively displaying a paragraph name
   b. By causing the program to pause at any needed step during execution
   c. By allowing the program to examine and modify data at will before continuing execution.

   Easy program development and debugging increases programmer productivity by deleting tedious waiting periods.

2. Batch – Using the same commands he used for timesharing, the programmer may submit the program via a control file to the batch system. This increases his efficiency because other terminal work may be done concurrent with the batch processing.

3. Access methods – The programmer has a choice of three access methods: sequential, indexed sequential, and random.

4.4.4.2 Indexed Sequential Access Mode (ISAM) – ISAM needs a minimum amount of programming while providing a large data file handling capacity. It is supported by the COBOL object time system, which automatically handles all of the searching and movement of data.

All reading and writing of an index file (ten levels of indexing) are performed by the run-time operating system (LIBOL). This does not include the user. When using the indexed sequential files, the programmer need only specify which record is to be read, written, or deleted.

When records are added to the file, the index is automatically updated. Additions to the file will not lower the file as with other computers. The common way of using overflow areas for added records has been prevented. When records have been deleted from the file, the empty space is used again for later additions. The net effect of the addition and deletion methods increases the time between major "repairs" of the data files because the time needed to access a file is independent of the number of changes made to it.

1. Sorting – The SORT package permits a user to rearrange records or data according to a set of user-specified keys. The keys may be ascending or descending, alpha or numeric, and any size or location within the record. Multireel file devices may be specified.

2. Source library maintenance system – This system lists file entries or adds, replaces, and/or deletes source language data on a file. It can also add, replace, or get a complete file from the library.

3. Device independence – The operating system allows the programmer to reference a device with a user-assigned logical name as well as its physical name, therefore allowing dynamic assignment of peripheral devices at run time.

4. Data base management system interface – The programmer may call on the data base management system facilities from within his program. This system, which follows the CODASYL specifications, allows data files to be consolidated into one or more data bases. Application programs are then permitted to access the data in the way best suited to their needs.

4.4.5 EDIT

Edit allows on-line creation and modification of programs and data files. The user may insert, delete, or print lines, as well as modify lines without having to type them over. Advanced methods include string searches, substitutions, and text manipulations.

The beginner finds the system easy to use due to the user-oriented syntax and extensive helping facilities. At the same time, the more advanced user may perform complex text manipulation with equal ease.

4.4.6 MACRO

MACRO is the symbolic assembly language on the DECSYSTEM-20. It makes machine language programming easier and faster for the user by:

1. Translating symbolic operation codes in the source program into the binary codes needed in machine language instructions

2. Relating symbols specified by the user to stored addresses or numeric values

3. Assigning relative core addresses to symbolic addresses of program instructions and data

4. Providing a sequence numbered listing with symbols cross-referenced to show where they are defined and where used.

MACRO programs have a series of free format statements that can be prepared on the user's terminal with a system editing program. The elements in each statement can be entered in free format. The assembler interprets and processes these statements, generates binary instructions or data words, and

processes a listing that may contain cross-reference symbols for ease in debugging. MACRO is a device-independent program; it allows the user to select, at run time, standard peripheral devices for input and output files. For example, input of the source program can come from the user's terminal, output of the assembled binary program can go to a magnetic tape, and output of the program listing can go to the line printer. Usually, the source program input and the binary output are disk files.

The MACRO assembler contains powerful macro capabilities that allow the user to create new language elements. This capability is useful when a sequence of code is used several times with only certain arguments changed. The code sequence is defined with dummy arguments as a macro instruction. Therefore, a single statement in the source program referring to the macro by name, along with a list of the actual arguments, generates the complete sequence needed. This capability allows for the expansion and adjustment of the assembler in order to perform special functions for each programming job.

4.4.7 DDT

The on-line symbolic debugging tool, DDT (dynamic debugging technique), enables the user to perform rapid checkout of a new program by making a change resulting from an error detected using DDT and then immediately executing that section of the program for testing and loading. The user may enter DDT at any time, either before execution or after an error has occurred. After the source program has been assembled, the binary object program, with its table of defined symbols, is loaded with DDT. Through command strings to DDT, the user can specify locations in his program, called breakpoints, where DDT is to suspend execution in order to accept more commands. In this way, the user can check out his program section-by-section and if an error occurs, insert the corrected code immediately. Either before DDT starts execution or at breakpoints, the user can examine and modify the contents of any location. Insertions or deletions can be in source language code or in different numbered text modes. DDT also performs searches, gives conditional dumps, and calls user-coded debugging subroutines at breakpoint locations. The important feature of DDT is that user communication with DDT is in terms of the original symbolic location names; instead of the machine-assigned locations.

4.4.8 Dumper/Backup

Dumper/Backup provides file backup for the disk file structure. Installations may use it to dump and restore all files in all directories, to dump and restore only those files that have been changed since last dump, and to dump and restore individual user directories.

4.4.9 FORTRAN

The formula translator language, FORTRAN, is a widely-used and procedure-oriented programming language. It is designed for solving scientific problems and is therefore made up mathematical-like statements made in accordance with precisely formulated rules. Therefore, programs written in FORTRAN have meaningful sequences of these statements that are designed to direct the computer to perform the specified computations.

FORTRAN has a different use in every segment of the computer market. Universities find that FORTRAN is a good language with which to instruct students how to solve problems via the computer. The scientific market relies on FORTRAN because of the ease with which scientific problems can be shown. In addition, FORTRAN is used as the primary data processing language by many timesharing utilities.

Because of this wide market, DECSYSTEM-20 FORTRAN is designed to meet the needs of all users. FORTRAN is a superset of ANSI standard. FORTRAN also provides many extensions and additions to this standard that greatly enhance its usefulness and increase its compatibility with other FORTRAN language implementations. The compiler produces optimized object code to decrease execution time.

FOROTS, the FORTRAN object time system, implements all program data file functions and provides the user with an extensive run-time error reporting system. Device independence is provided to allow the programmer or operator to determine the physical device at run time.

The FORTRAN system is easy to use in both timesharing and batch processing environments. Under timesharing, the user operates in an interactive editing and debugging environment. FORDDT, an interactive program that is used to help in debugging FORTRAN programs, uses the language constructions and variable names of the program itself — thereby eliminating the need to learn another language in order to do on-line debugging. Under batch processing, the user submits his program through the batch software in order to have the compiling, loading, and executing phases performed without his intervention.

FORTRAN programs can be entered into the FORTRAN system from a number of devices: disk, magnetic tape, user terminal, and card reader. In addition to data files created by FORTRAN, the user can submit data files or FORTRAN source files created by the system editor. The data files contain the data needed by the user's object program during execution. The source files contain the FORTRAN source text to be compiled by the FORTRAN compiler. Output may be received on the user's terminal, disk, or magnetic tape. The source listing cross references all symbols to show where they are defined and where used.

4.4.10 Data Base Management System

4.4.10.1 Introduction – Certain data, such as that within commercial, accounting, inventory control, and administrative systems, is used in computer applications that have common relationships and processing requirements with data in other applications. This can be a problem because as file organizations are defined in one application or program, restructuring, redundant format, and even repetitive processing of the same data are often needed in another application.

To complicate the problem, on-line processes (for example, customer order entry, shipment planning, and student information retrieval systems) use different data structures than the processes used to create and maintain primary data files. This means that as new applications for existing data are determined and additional data is defined, program development personnel must change existing data file forms and programs or create and maintain redundant copies of previously recorded data.

Digital offers a solution to the problem in the Data Base Management System, DBMS. It enables DECSYSTEM-20 users to organize and maintain data in forms more acceptable to the integration of a number of related but separate processes and applications. DBMS is ideal in situations where data processing control and program development functions need structures and methods not satisfied by typical data management facilities.

4.4.10.2 DBMS Features – The DECSYSTEM-20 Data Base Management System is based on the proposals of the CODASYL Data Base Task Group (DBTG), which appear in their report of April, 1971. Digital's goal for DBMS is to provide features that help users in getting the most significant objectives stated in the CODASYL DBTG report. These features include:

1. Hierarchical data structures – In addition to sequential structures, simple tree structures, and more complex network structures can be created and maintained. Data items can be related within and between different levels of the structure created. Figure 4-3 shows these basic structures.

2. Nonredundant data occurrence – Data items may appear in a number of different structural relationships without needing multiple copies of the data. Data structures may be created and modified in a way most suitable to a given application without changing the occurrence of the data in structures maintained for other applications.

3. Variety of access and search designs – Data records can be maintained in chains, as shown in Figure 4-4. Access may be through DIRECT, CALCULATED, or VIA set location modes. Sort keys, in addition to the normal storage key, may be defined.

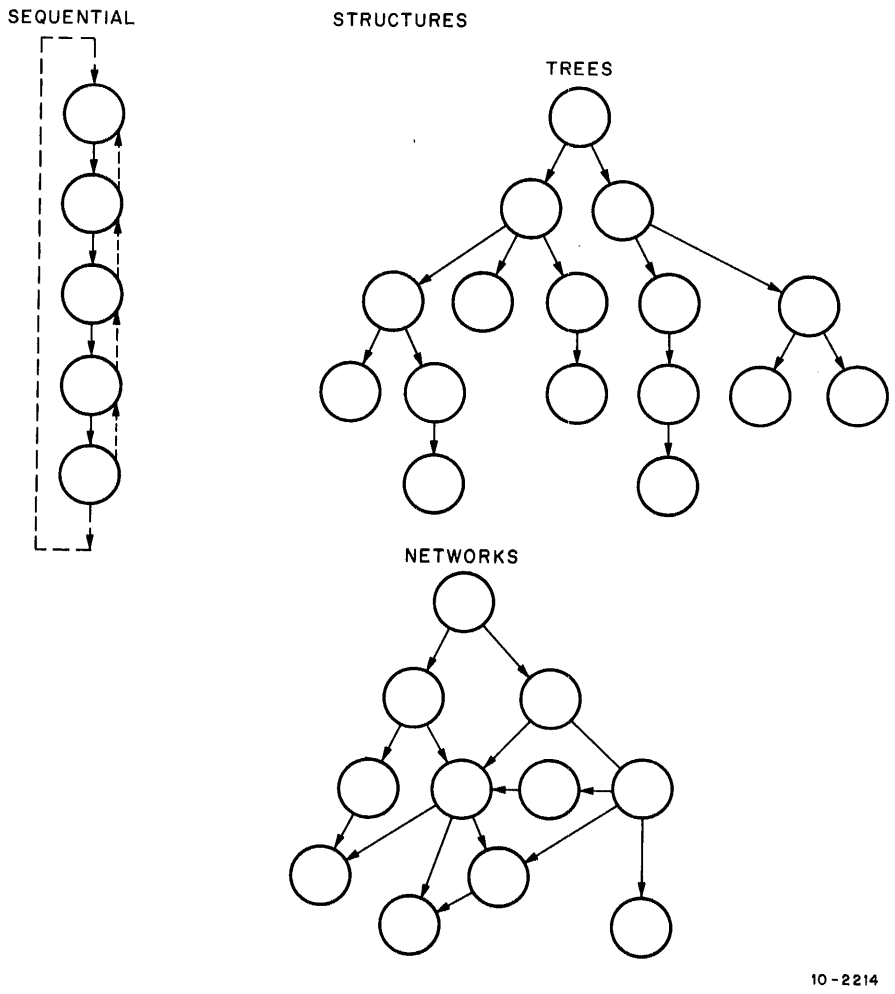


Figure 4-3 Basic Data Structures

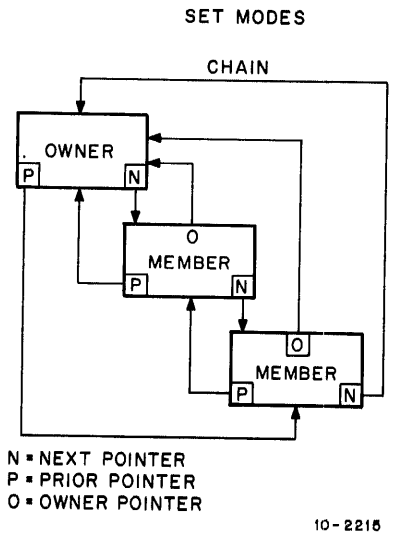


Figure 4-4 Chaining of Data Records

4. Concurrent access – Multiple run units (or programs) that use the same reentrant code module add to the DECSYSTEM-20 monitor's extensive centralized file handling capabilities. Any number of concurrent retrievals to the same data areas can be handled. Concurrent updates to the same area can be handled by a multiple update queueing mechanism.

5. Protection and centralized control – Usage of a common data definition language creates data base structures maintained on direct-access storage devices that are selectively referenced by individual application programs through privacy lock and key mechanisms. Physical placement of data is specified by a centrally-controlled data definition processor.

6. Device independence – The common input/output and control conventions of the DECSYSTEM-20 monitor provide basic device independence. Applications programs have logical areas instead of physical devices. Data base areas may reside on the same or different direct-access storage devices as nondata base files.

7. Program independence of data – Significant steps toward program independence are achieved by using the SCHEMA and SUB-SCHEMA concepts of data definition. Individual programs reference only user-selected data elements instead of complete record formats. Program changes caused by adding new data and relationships are minimized. Changes of the original form (for example, binary, display) and element sizes need program recompilations.

8. User interaction without structural maintenance responsibilities – Individual users, applications, and programs may access data structures without the responsibility of maintaining detailed linkage mechanisms that are internal to the data base software. Common and centralized authorization error recovery, and building techniques decrease individual activity to maintain data structures.

9. Multiple language usage – The DBMS data manipulation language is available to both COBOL and FORTRAN as host languages.

10. DBMS software modules – Consistent with the CODASYL Data Base Task Group report, the body of DECSYSTEM-20 data base management software includes:

    a. DDL – Data description language and its processor
    b. DML – Data manipulation language for COBOL and FORTRAN programs
    c. DCBS – Data base manager module of reentrant run-time routines
    d. DBU – Group of data base system support utilities.

4.4.11 LINK

LINK, the DECSYSTEM-20 linking loader, merges independently translated modules of the user's program and any necessary system modules into a single module that can be executed by the operating system.

The primary output of LINK is the executable version of the user's program. The user can also request auxiliary output in the form of map, log, save, symbol, overlay plot, and expanded core image files by including appropriate instructions in his command strings to LINK. The user can also gain exact control over the loading process by setting different loading parameters and by controlling the loading of symbols and modules. Also, by setting parameters in his command strings to LINK, the user may specify the core sizes and starting addresses of modules, the size of the symbol table, the segment into which the symbol table is placed, the messages he will see on his terminal or in his log file, and the severity and detail of any error messages. Finally, he can accept the LINK defaults for items in a file specification or he can set his own defaults that will be used automatically when he omits an item from his command string.

4.4.12 RUNOFF

RUNOFF is the DECSYSTEM-20 documentation preparation program. It provides line justification, page numbering, titling, indexing, formatting, and case shifting. The user creates a file that includes text and information for formatting and case shifting, with an editor. RUNOFF processes the file and produces the final formatted file to be output to the terminal, the line printer, or to another file.

With RUNOFF, large amounts of material can be inserted into or deleted from the file without retyping the text that is unchanged. After a group of changes have been added to a file, RUNOFF produces a new copy of the file that is correctly paged and formatted.

4.4.13 SORT

The TOPS-20 SORT arranges the records of one or more files according to a user-specified sequence. The user names the keys on which the records are sorted from one or more fields within a record. The keys can be in either ascending or descending order. SORT compares the key fields values of all records. Then it arranges the records in the specified sequence and merges them into a single output file. SORT may be used under timesharing and batch and may be called from within a COBOL program.

APPENDIX A
MCA25 KL CACHE/PAGING UPGRADE

A.1 INTRODUCTION

The KL Cache/Paging Upgrade provides performance improvement for the KL10 CPU. The KL Cache Upgrade improves the cache hit ratio by doubling the cache size. There are currently four caches in the KL10 CPU.

The KL Paging Upgrade improves the performance of TOPS-20 paging in three ways.

1. Expands the hardware page table from 512 to 1024 entries.

2. Makes the page table two-way associative (two 512-entry page tables).

3. Adds the KEEP bit, allowing the translation of certain pages to be retained in the hardware page table during context-switch.

A.2 CURRENT KL10 TOPS-20 PAGING

All memory, physical and virtual, is divided into pages of 512 words each. Physical memory can contain 8,192 pages. The locations in physical memory are specified by 22-bit addresses, where the left 13 bits (14–26) specify the page, and the right 9 bits (27–35) specify the location within the page. The virtual memory space addressable by a program is 16,384 pages and requires 23-bit addresses, where the left 14 bits (13–26) are the extended page number. However, the virtual space is usually regarded as 32 sections, 512 pages each. With this view, the extended page number has two parts: the left 4 bits (13–17) specify the section, and the right 9 bits (18–26) specify the page number. The hardware maps each section of the virtual address space into a part of the physical address space by transforming the 18-bit addresses into 22-bit addresses. In this transformation, the right 9 bits of the virtual page is the same location in the corresponding physical page. The translation maps a virtual page into a physical page by substituting a 13-bit physical page number for the 9-bit virtual page number. The mappings are different for each section by virtue of each section having a separate page map. The procedure is carried out automatically by the pager, but the maps that supply the necessary substitutions are set up by the monitor. The formats for these maps are shown in Figures A-1 and A-2.

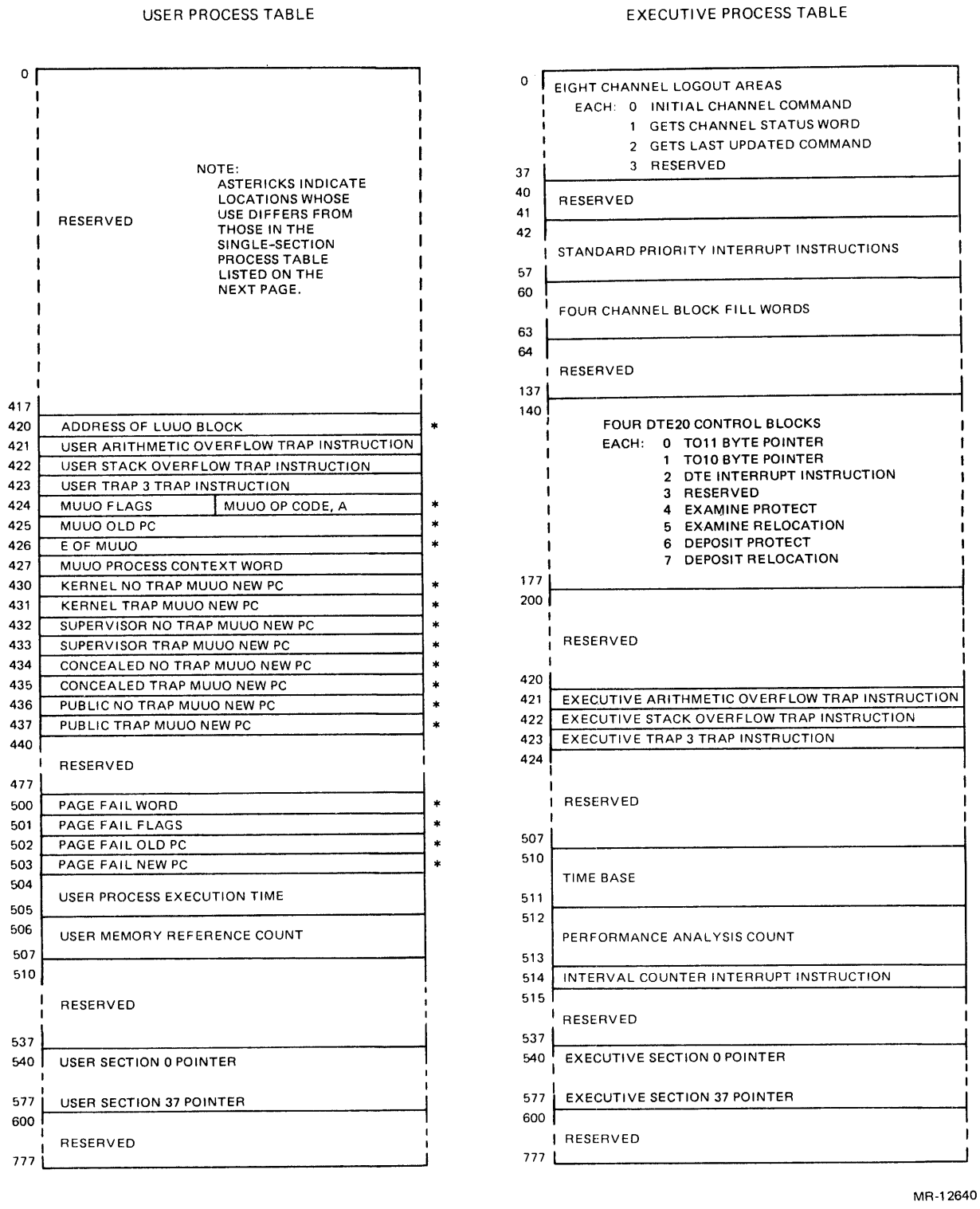


Figure A-1 Extended TOPS-20 Process Table Configuration

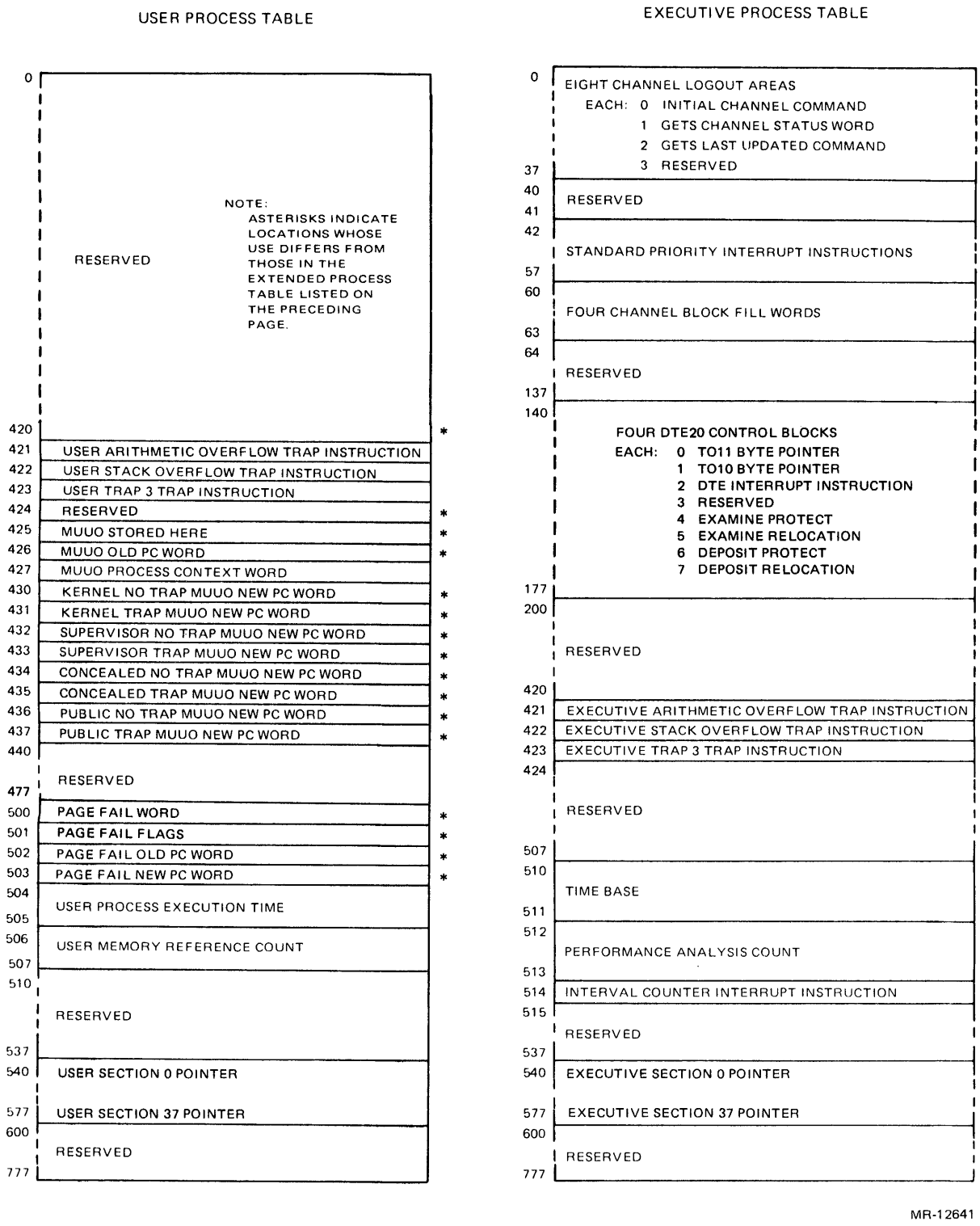


Figure A-2 Single-Section TOPS-20 Process Table Configuration

A.3 TOPS-20 PAGING MICROCODE

To determine mapping for a virtual page, the microcode carries out a pointer-evaluation procedure that starts at the appropriate entry in the section table. If it is discovered during this procedure that any of the following conditions exist, then microcode traps to the monitor to handle the situation.

1. The section or page table is inaccessible.
2. The page map or referenced page is not in memory.
3. The program is attempting to write in a write-protected page.

A trap to the monitor for a reason of this sort is produced by generating a "soft-page failure." If no problems arise, the procedure is carried out entirely by the microcode with no need to call the software, and mapping is generated for the specified virtual page. The procedure requires access to:

1. Both the section table and page map.

2. A memory status table in which the microcode keeps track of the page map.

3. The program-referenced page, and perhaps to other predefined or software-defined tables as well.

If the complete procedure were to be carried out in every instance, the processor would require at least five memory references for every one done by the program. To avoid this, each mapping generated by the procedure is placed in a hardware page table. Then the pager makes its virtual-to-physical translations from the mappings held in the table. Therefore, it is only necessary to perform the evaluation procedure when mapping is not available in the hardware page table. Since the object of the procedure is to place mapping in the table, it is referred to as a "page refill."

A.4 KEEP BIT

Currently the KL10 hardware forces the monitor to sweep all entries of the entire page table each time it changes users. The sweep is done when a DATO page is executed to set up the user base register for the new user's pages. Sweeping the entire page table clears the user's pages, and the executive page table entries. The result is that the executive pages must be refetched by the microcode even though they have not changed. The KL10 needlessly spends an estimated 10 percent of the average program execution time evaluating and refetching executive page table entries.

A.4.1 Maps With KEEP Bit

The upgrade adds the KEEP bit and eliminates the need to refetch the executive page table entries. Pages with the KEEP bit set are protected when a page table sweep is done.

To set the KEEP bit in the hardware page table, the monitor must set the KEEP bit (bit 5) of all of the section and map pointers used to evaluate the translation of a virtual reference. Refer to Figures A-3 through A-8.

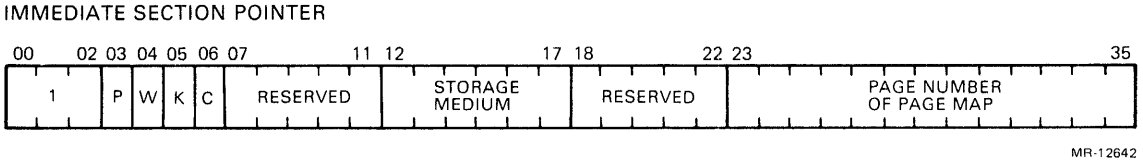


Figure A-3 Immediate Section Pointer

SHARED SECTION POINTER

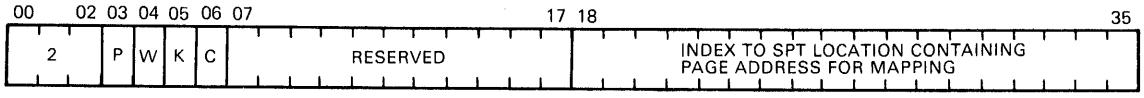


Figure A-4 Shared Section Pointer

INDIRECT SECTION POINT

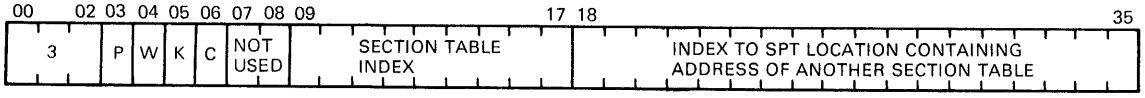


Figure A-5 Indirect Section Pointer

IMMEDIATE MAP POINTER

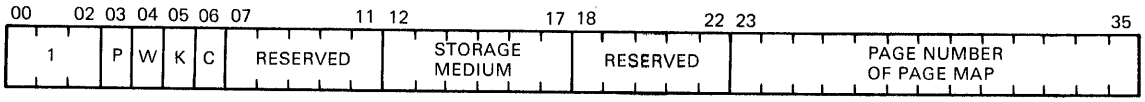


Figure A-6 Immediate Map Pointer

SHARED MAP POINTER

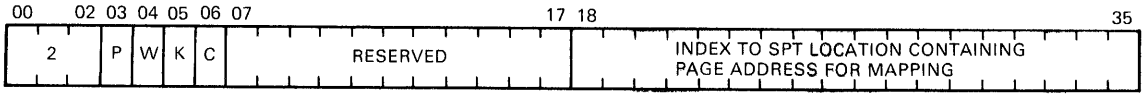


Figure A-7 Shared Map Pointer

INDIRECT MAP POINTER

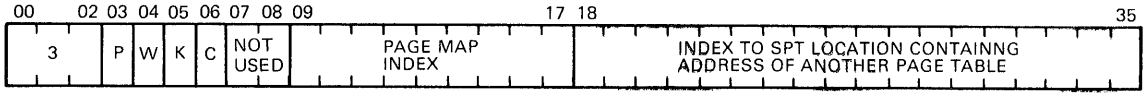


Figure A-8 Indirect Map Pointer

A.5 TWO-WAY ASSOCIATIVE

Each new release of the monitor requires more virtual address space and more extended sections. As more sections are used, the chance of the virtual addresses requiring the same hardware page table space increases. Since the KL10 has only one hardware page table, any time two references require the same page but a different section, a page refill must be done. Some programs or even instructions can cause the pager to constantly refill back and forth between two sections. This is called "page table thrashing." To reduce the number of references that require the same page table reference, the KL Page Table Upgrade expands the number of page tables in the KL10 from one to two. Two page tables, or a two-way associative page table, reduces "thrashing."

A.6 DATA OUT PAGER (DATAO PAG)

The DATAO PAG instruction has been modified to allow the page table to be swept while retaining entries with the KEEP bit. See Figure A-9.

Set up the process-oriented elements of the pager according to the content of location E. The format of word E is shown in Table A-1.

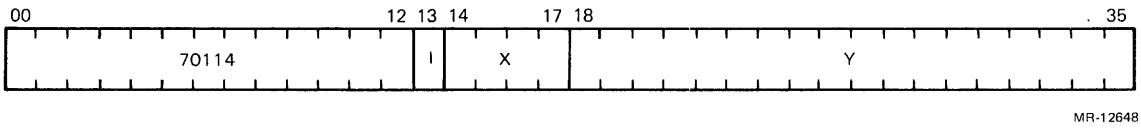


Figure A-9 DATAO Page Instruction

Table A-1 Word E Format

Bit	Description
00	Select the current and previous context AC blocks specified by bits 6–8 and 9–11 of E, respectively.
01	Select previous context section specified by bits 13–17 of E.
02	Load User Base Register (UBR).
03	Invalidate entire page table if bit 3 is reset; invalidate only non-keep entries in page table if bit 3 is set.
06–08	Current AC block number.
09–11	Previous context AC block number.
13–17	Previous context section number.
18	Do not update user's accounts, loc 504–507 of UPT.
23–35	User base register.

A.7 THE MAP INSTRUCTION

If the pager is on and the processor is in kernel or user I/O mode, map the page number of the virtual effective address E and place the resulting physical address and other map data in AC. See Figure A-10.

The information loaded into AC for a true mapping is shown in Table A-2.

Failure of the instruction to generate a valid mapping is indicated by the AC receiving a page-fail word. (Refer to Paragraph A.8.)

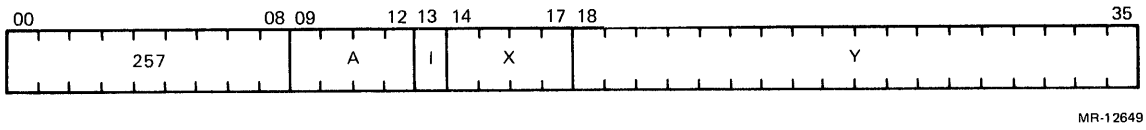


Figure A-10 Map Instruction

Table A-2 AC Format

Bit	Description
00	USER's page=1, EXEC's page=0
01	0
02	1
03	MODIFIED (indicates page has been written)
04	WRITABLE (indicates page is writable)
05	0
06	PUBLIC
07	CACHEABLE
08	KEEP bit (page will not be invalidated on DATAO PAGE)
09–13	0
14–35	PHYSICAL ADDRESS

A.8 PAGE FAILURE

When the pager is unable to make a desired reference, a "page failure" occurs. The pager terminates the instruction immediately without disturbing the PC or storing any results in memory or the accumulators. It then executes a page fail trap. The trap operation uses four locations in the user processor table.

KL-Style Paging	KI-Style Paging
500 page fail word	page fail word
501 page fail flags	page fail old PC word
502 page fail old PC	page fail new PC word
503 page fail new PC	

The processor then resumes operation in the new state at the location now addressed by the PC. The page-fail word supplies the information.

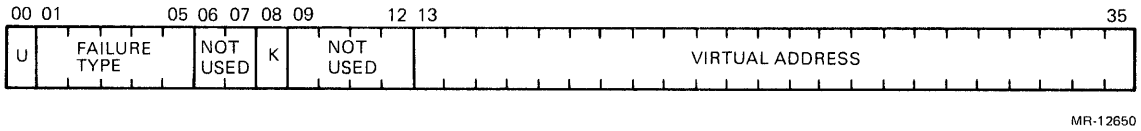


Figure A-11 Page Fail Word

Table A-3 Page Fail Word Format

Bit	Description
00	USER's page=1, EXEC's page=0
01–05	Failure type (see *DECSYSTEM-20 Processor Reference Manual*)*
06–07	Not used
08	KEEP bit
09–13	0
14–35	Virtual address

*<01:05> Page Fail Codes

21 Proprietary violation	36 EXEC Mode AR data parity error
23 Address Failure	37 EXEC Mode ARX data parity error
24 Illegal/Indirect	76 User Mode AR data parity error
25 Page Table Parity Error	77 User Mode ARX data parity error
27 Illegal Address - Section>37	

A.9 CONDITIONS OUT, PAGER (CONO PAG)

The CONO PAG instruction executes exactly the same as the current KL10. Entire page tables will be invalidated, including KEEP entries.

A.10 MICROCODE

The KL10 microcode will be changed for DATAO PAG to sweep the entire page table and the KEEP bits. Changes to the page-refill microcode are also required to move the KEEP bit from bit 5 of the map pointer to bit 23 of AR in the hardware. The following is an estimate of the number of additional microcode words.

1. Three words to move the KEEP bit in the refill routine.
2. Three words to allow DATAO PAG to clear the KEEP bits.
3. One word to honor the KEEP bit.
4. No additional words required to cause CONO PAGE to clear the page table.

The Mbox hardware sweeps the page table. Mbox hardware invalidates the page table according to the code in the number field and the assertion of COND/MBOX CTL. The number field is decoded as shown in Table A-4.

Table A-4 Number Field Format

Bit	Description
00	Normal
01	Enable/invalidate entire page table
02	Enable right-half and left-half of page table
10	Write translation buffer
20	Write page table directory, set valid bit
31	Invalidate entire page table
33	Write page table directory, set valid bits for even and odd page
41	Enable/invalidate non-KEEP entries of page table
61	Invalidate non-KEEP entries of page table
70*	Select both page table 0 and page table 1
71*	Select page table 0 only
73*	Select page table 1 only
100	Set I/O PF ERR
200	Set PAGE FAIL

* Indicates that these codes are not currently used in KL. Code 70, 71, and 73 are used by diagnostics and KLINIT only.

A.11 PAGE REFILL ALGORITHM

Prior to the O.S. executing CONO PAG to turn on paging, the page table is invalidated by clearing all VALID bits in the translation buffers.

When the Ebox issues a request to the Mbox, the pager checks the USER bit and section number in both page table directories, and the VALID bits in the translation buffers. A mismatch in both page table directories or a non-valid translation buffer causes a page-refill trap.

Microcode then evaluates the section pointers and map pointers. Each access bit (P,W,K,C) of each pointer is ANDed with the same access bit of all the pointers evaluated. The pager then writes the resulting section information into the page table directory. Two translation buffer entries are then cleared in the least-recently refilled (LRR) page table associated with the current page table directory. Only two entries are cleared, rather than four as in the current KL10, because there are twice as many directory entries per directory, or four times as much directory space as in the current KL10.

Microcode places paging information in the Arithmetic Register (AR), ACCESS bit (A,P,W,S,C) in AR<00:04>, physical page number in AR<05:17>, and KEEP bit in AR<23>. The rest of the bits in AR should be zero to generate correct parity in the hardware page table. Microcode then writes the paging information into the translation buffer of the LRR page table. The hardware updates the LRR bit when the translation buffer is written.

On a context-switch, the monitor may choose to sweep both page tables except those entries that have the KEEP bit set. DATAO PAG with bit 3 set clears the VALID bits in both translation buffers for those entries that do not have the KEEP bit. However, both page table directories and LRR information remain as they were before the context-switch.

After the sweep, the entries in the translation buffers with the KEEP bit are still valid. References to the kept pages will not cause refills. If after the sweep, one page table translation buffer has the KEEP bit set, and the other page table is not valid, the LRR bit is forced to indicate that the empty page table is least-recently refilled. The next refill then goes into the unused translation buffer. After both page tables are valid, the next refill goes into the page table that is least-recently refilled.

A.12 APRID WORD

Bit 23 of the APRID word is used to identify the hardware option of the keep feature in the paging board.

A.13 CACHE STRUCTURE

The cache consists of a data buffer for storing instructions and operands, and a directory buffer for storing the physical memory address and status (VALID and WRITTEN bits) information (Figure A-12). The contents of the directory buffer identify the contents of the data buffer. The cache data buffer contains 4096 locations, each of which is associated with a valid and a written bit location in the directory. The 4096-word data and status bit locations are divided into 1024 sets of four, which are directly associated with corresponding address locations in the directory. In addition, the 1024 sets of data and directory locations are divided further into sets of 256, resulting in four cache quarters. If a copy of a block is made from main memory, it is always and only stored in one of the four corresponding (addressed) blocks of the data buffer. The actual block to be used is specified by the contents of a use table. The use table maintains a record of the order in which the four addressed cache blocks are used and maintains one entry for each of the 256 lines in the cache. The contents of the use table are used to select the block that contains the least-recently used (LRU) data for storing the new data.

Besides writing a block of four words into the cache data buffer, the associated directory locations are also updated to specify the valid words and the physical address of the data block. The written bits in the cache directory are not set when data is moved from memory to the cache, but set only when the EBox writes into the cache. When words are written into the cache by the EBox, the address and the valid bits in the directory are also updated.

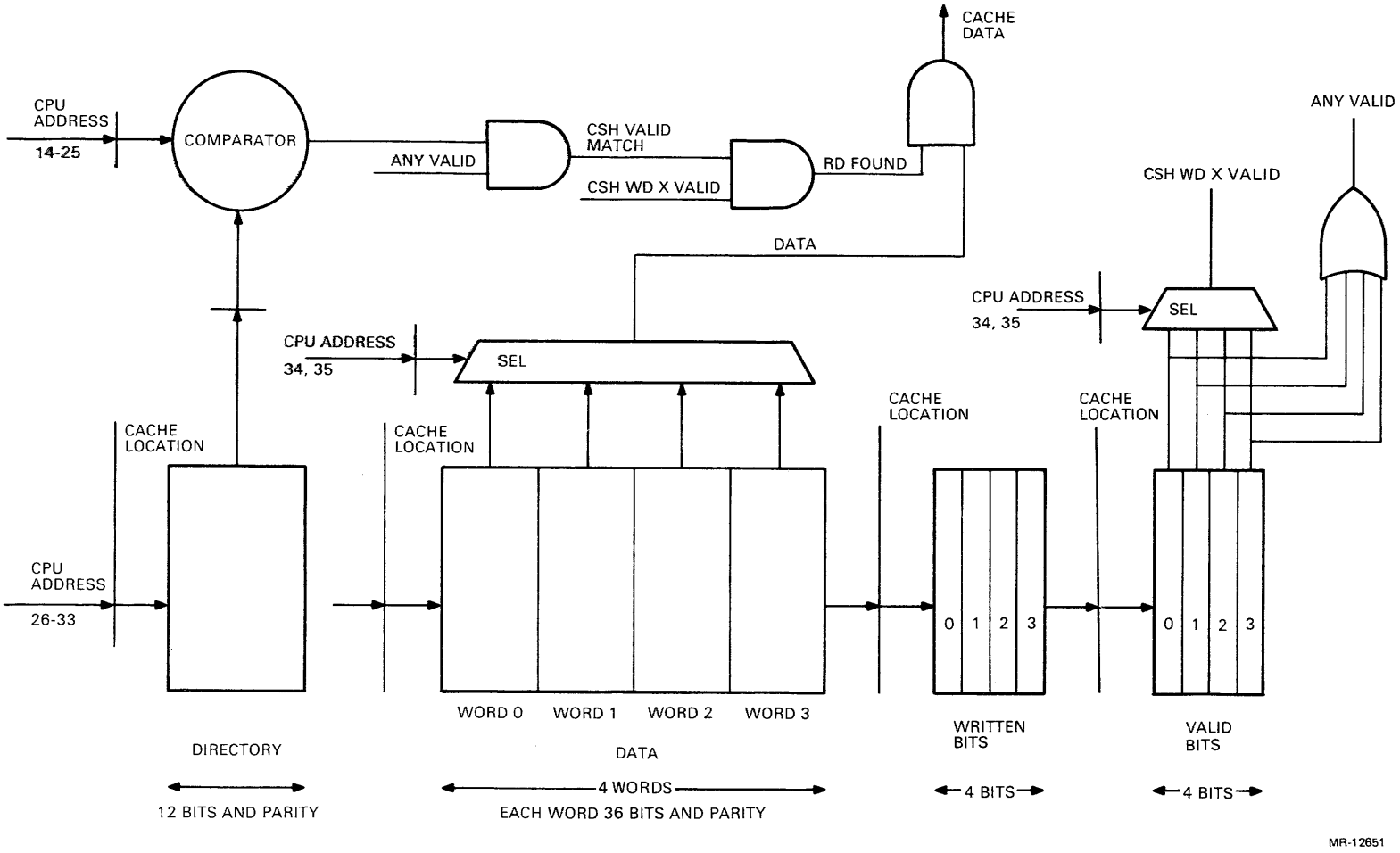


Figure A-12 Example of Basic Cache Structure

The convention that a block from main memory is always stored in the LRU block of the corresponding data buffer line ensures that a given line in the data buffer will never contain more than one quadword from a given page. Therefore, a conflict (more than one address in one line matching) will not occur when comparing the address with the contents of the directory to determine if the desired word is in the data buffer. This feature of refilling the cache also tends to keep frequently used instructions and operands stored in the cache for a longer period of time.

At any given time, the cache may contain up to 1024 quadwords (4096 words). The distribution may range from eight complete pages, from anywhere in core, to four words from every page of any section of core. When the EBox makes a paged or unpaged request to read or write a word (for which the page test has passed), the cache directory is checked to see if a record exists for the quadword in which the requested word is located. If an address matches and at least one valid bit in that block is set, then the cache has a record of the quadword.

A.13.1 Cache Control

The cache control executes requests initiated by the EBox and the channel control. Both the EBox and the channel control can issue data-read and data-write requests to the cache control. The EBox can also request to load or read internal MBox registers, check if a given page is writable, map the virtual address, and sweep the cache.

Data-read and data-write requests from the EBox, and from the channel control, cause the cache control to enter a specific cache cycle and step through a set of time states. (The relevant time-state-set varies with the cycle.) The cache control can execute four major and two minor cache cycles (Table A-5).

Table A-5 Cache Cycle Types

Cycle	Major	Minor
CSH EBOX	X	
CSH PAGE REFILL		X
CSH WRITEBACK		X
CSH MB	X	
CSH CCA	X	
CSH CHAN	X	

All EBox requests are serviced by the MBox by starting a cache EBox cycle. As the cache control advances through the relevant states in response to an EBox request, the page table (if paged reference) and cache directory are checked for valid entries. Page table entries are valid when the USER bit and section address matches the EBOX USER signal and virtual section address presented by the EBox and the INVAL bit in the table is cleared. Cache entries are valid if the address of the requested word is found and the VALID bit is set in the cache directory. If a valid entry is found for an EBox request, the data is simply transferred between the cache and the AR. If a valid entry is not found and the EBox requested to read a word, the cache control initiates a core read cycle to fetch the desired word along with adjacent words of the quadword group. For EBox write requests, the cache control writes the word into the cache

block that has a record of one quadword or into the least-recently used cache block; no core cycle is started. Words coming in from core are placed into the memory buffers (MBs) by the core and MB controls and then are individually moved into the cache by the cache and MB controls. The first word, which will be the word the EBox requested, is placed on cache data lines so that the EBox can take it. Words are written back into core only when the EBox makes a request to read or write a word (except for cache sweep) and a valid entry is not found but the written bit is set (Table A-5).

Having the written bit set means that the corresponding data is more up to date than the core copy and, therefore, core must be validated before that cache location can be used for the pending request. To write words back to core, the cache and MB controls move the words into the MBs and start a core write cycle after the first word is placed into an MB.

The channel control does not write into the cache, but moves the words to be written from the channel buffer to the MBs and causes the cache control to invalidate any valid entries in the cache. On channel writes, the valid entries in the cache (if any) are invalidated because it is defined that data coming in from mass storage is more up to date (or is another process) than any data that may still be in core or in cache. Therefore, on channel-write requests, the cache control always initiates a core-write cycle. On channel reads, any valid entries in the cache will be moved into the MBs and a core-read cycle will be initiated for the remaining words requested, if any. The channel control then moves the words from the MBs to the channel buffer.

A.13.2 Cache Sweep Instruction

A cache data sweep for a one-page instruction (SWPIO, SWPVO, and SWPUO) will now sweep two pages. Both even and odd pages are swept when this instruction is executed.

KL10-Based Technical Manual
EK-OKL10-TM-002

Reader's Comments

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our publications.

What is your general reaction to this manual? In your judgment is it complete, accurate, well organized, well written, etc.? Is it easy to use? _____

_____

_____

_____

What features are most useful? _____

_____

_____

_____

What faults or errors have you found in the manual? _____

_____

_____

_____

Does this manual satisfy the need you think it was intended to satisfy? _____

Does it satisfy *your* needs? _____ Why? _____

_____

_____

_____

☐ Please send me the current copy of the *Technical Documentation Catalog*, which contains information on the remainder of DIGITAL's technical documentation.

Name _____ Street _____

Title _____ City _____

Company _____ State/Country _____

Department _____ Zip _____

Additional copies of this document are available from:

Digital Equipment Corporation
444 Whitney Street
Northboro, MA 01532
Attention: Printing and Circulating Service (NRO2/M15)
Customer Services Section

Order No. EK-OKL10-TM

MRO

Do Not Tear — Fold Here and Staple

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD, MA.

POSTAGE WILL BE PAID BY ADDRESSEE

Digital Equipment Corporation
Educational Services/Quality Assurance
12 Crosby Drive (BUO/E08)
Bedford, MA 01730

Digital Equipment Corporation • Bedford, MA 01730