

decsystem -1080/1090
SYSTEM DESCRIPTION

decsystem -1080/1090
SYSTEM DESCRIPTION

1st Edition, June 1975
2nd Printing, April 1976
3rd Edition, (Rev) January 1977

The drawings and specifications herein are the property of Digital Equipment Corporation and shall not be reproduced or copied or used in whole or in part as the basis for the manufacture or sale of equipment described herein without written permission.

Copyright © 1975, 1976, 1977 by Digital Equipment Corporation

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 which may appear in this manual.

Printed in U.S.A.

This document was set on DIGITAL's DECset-8000 computerized typesetting system.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DEC	DECtape	PDP
DECCOMM	DECUS	RSTS
DECsystem-10	DIGITAL	TYPESET-8
DECSYSTEM-20	MASSBUS	TYPESET-11
		UNIBUS

CONTENTS

	Page
SECTION 1	PRINCIPLES OF TIME-SHARING
1.1	INTRODUCTION SYS10/1-1
1.2	OPERATION SYS10/1-2
1.2.1	Dynamic Scheduling SYS10/1-3
1.2.2	Software Sharing SYS10/1-4
1.2.3	Communications SYS10/1-5
1.2.4	Control of Input/Output SYS10/1-6
1.2.5	File Handling SYS10/1-6
1.2.6	Slow Peripherals SYS10/1-7
1.3	RELIABILITY SYS10/1-7
SECTION 2	DECsystem-10 PRIMER
2.1	INTRODUCTION SYS10/2-1
2.2	HARDWARE SYS10/2-2
2.3	OPERATING SYSTEM SYS10/2-3
2.4	NONRESIDENT SOFTWARE SYS10/2-4
2.5	MULTIPROCESSING SYS10/2-4
2.6	MULTIMODE COMPUTING SYS10/2-4
2.6.1	Time-Sharing SYS10/2-4
2.6.1.1	Command Language SYS10/2-5
2.6.1.2	Peripheral Devices SYS10/2-5
2.6.1.3	Spooling SYS10/2-5
2.6.1.4	Mass Storage File System SYS10/2-5
2.6.1.5	Core Utilization SYS10/2-5
2.6.1.6	General-Purpose Time-Sharing SYS10/2-6
2.6.2	Batch SYS10/2-6
2.6.2.1	Components SYS10/2-6
2.6.2.2	Use of System Features SYS10/2-8
2.6.2.3	Flexibility SYS10/2-8
2.6.2.4	Job Dependency SYS10/2-8
2.6.2.5	Error Recovery SYS10/2-8
2.6.2.6	Operator Intervention SYS10/2-8
2.6.3	Real-Time SYS10/2-9
2.6.3.1	Locking Jobs SYS10/2-9
2.6.3.2	Real-Time Devices SYS10/2-9
2.6.3.3	High-Priority Run Queues SYS10/2-10
2.6.3.4	Job Communication SYS10/2-10

CONTENTS (CONT)

	Page
SECTION 3	SYSTEM FEATURES
3.1	CENTRAL PROCESSOR SYS10/3-1
3.2	INSTRUCTION SET SYS10/3-1
3.2.1	Full-Word Data Transmission SYS10/3-4
3.2.2	Half-Word Data Transmission SYS10/3-4
3.2.3	Block-Transfer Instruction SYS10/3-4
3.2.4	Byte Manipulation SYS10/3-4
3.2.5	Business Instruction Set SYS10/3-4
3.2.6	Logic Instructions SYS10/3-5
3.2.7	Fixed-Point Arithmetic SYS10/3-5
3.2.8	Floating-Point Arithmetic SYS10/3-5
3.2.9	Arithmetic Operation Modes SYS10/3-5
3.2.10	Fixed/Floating Conversions SYS10/3-5
3.2.11	Compare and Modify SYS10/3-5
3.2.12	Program Control SYS10/3-6
3.2.13	Input/Output SYS10/3-6
3.2.14	Unimplemented User Operations (UUs) SYS10/3-6
3.2.15	Trap Handling SYS10/3-6
3.3	INSTRUCTION FORMAT SYS10/3-6
3.4	NUMBER SYSTEM SYS10/3-6
3.4.1	Fixed-Point Arithmetic SYS10/3-7
3.4.2	Floating-Point Arithmetic SYS10/3-7
3.5	EFFECTIVE ADDRESS CALCULATION SYS10/3-8
3.6	GENERAL-PURPOSE REGISTER BLOCKS SYS10/3-8
3.7	MEMORY SYSTEM SYS10/3-8
3.7.1	Core Memory SYS10/3-8
3.7.2	Cache Memory SYS10/3-9
3.8	PROCESSOR MODES SYS10/3-9
3.9	PROCESS TABLES SYS10/3-10
3.10	MEMORY PROTECTION AND RELOCATION SYS10/3-10
3.11	DIRECT I/O SYS10/3-13
3.12	CHANNEL I/O SYS10/3-13
3.12.1	External Data Channels SYS10/3-14
3.12.2	Integrated Data Channels SYS10/3-14
3.12.2.1	Massbus Controller SYS10/3-15
3.12.2.2	Channel Controllers SYS10/3-15
3.13	PRIORITY INTERRUPT SYSTEM SYS10/3-16
3.14	TRAP FACILITY SYS10/3-17
3.15	PROGRAMMABLE CLOCKS SYS10/3-17
3.16	CONSOLE/DIAGNOSTIC COMPUTER SYS10/3-18
3.17	SYSTEM INTEGRITY FEATURES SYS10/3-18

CONTENTS (CONT)

		Page
SECTION 4	THE HARDWARE	
4.1	CONFIGURATIONS	SYS10/4-1
4.2	CENTRAL PROCESSOR SUBSYSTEM	SYS10/4-1
4.2.1	EBox	SYS10/4-9a
4.2.1.1	Hardware	SYS10/4-9a
4.2.1.2	Firmware	SYS10/4-25
4.2.2	MBox	SYS10/4-27
4.2.3	Meter Board	SYS10/4-28
4.2.4	E/M Interface	SYS10/4-29
4.2.5	SBus and External Memory	SYS10/4-29
4.2.5.1	Read Operations	SYS10/4-29
4.2.5.2	Write Operations	SYS10/4-30
4.2.5.3	Read-Pause-Write Operation	SYS10/4-30
4.2.6	DMA20	SYS10/4-31
4.2.7	EBus	SYS10/4-31
4.2.7.1	Output Operations	SYS10/4-32
4.2.7.2	Input Operations	SYS10/4-33
4.2.7.3	Programmable Interrupt (PI1–7) Operation	SYS10/4-34
4.2.7.4	High Priority (PI0) Interrupt Operation	SYS10/4-35
4.2.8	CBus	SYS10/4-35
4.2.9	DTE20	SYS10/4-36
4.2.10	RH20	SYS10/4-38
4.2.11	DIA20	SYS10/4-39
4.2.12	Interrupt Facility	SYS10/4-39
4.2.13	Trap Facility	SYS10/4-40
4.2.14	Internal Devices	SYS10/4-40
4.2.14.1	APR	SYS10/4-40
4.2.14.2	PI	SYS10/4-40
4.2.14.3	PAG	SYS10/4-40
4.2.14.4	CCA	SYS10/4-40
4.2.14.5	TIM and MTR	SYS10/4-40
4.2.15	External and Internal I/O Controllers and Devices (Typical)	SYS10/4-41
4.2.16	Machine Instructions	SYS10/4-41
4.3	CONSOLE PROCESSOR SUBSYSTEM	SYS10/4-41
4.3.1	Devices	SYS10/4-42
4.3.1.1	KD11-A Central Processor	SYS10/4-42
4.3.1.2	KY11-D Programmer's Console	SYS10/4-42
4.3.1.3	KW11-L Line Clock Option	SYS10/4-42
4.3.1.4	MF11-UP Memory	SYS10/4-42
4.3.1.5	MM11-UP Memory	SYS10/4-42
4.3.1.6	BM873-YD/YG ROM Loader Module	SYS10/4-42
4.3.1.7	DL11-C Asynchronous Line Interface	SYS10/4-43
4.3.1.8	DL11-E Asynchronous Line Interface	SYS10/4-43

CONTENTS (CONT)

		Page
4.3.1.9	LA36 Keyboard Terminal	SYS10/4-43
4.3.1.10	RJP04/06 Disk File System	SYS10/4-43
4.3.1.11	TC11-G Magnetic Tape System	SYS10/4-43
4.3.1.12	BC11-A Unibus	SYS10/4-43
4.3.1.13	DTE20 Ten-Eleven (Console/Front-End Processor) Interface	SYS10/4-43
4.3.2	Interdevice Transfers	SYS10/4-44
4.3.3	Functions	SYS10/4-44
4.3.3.1	Examine/Deposit Operations	SYS10/4-44
4.3.3.2	TO10/TO11 Byte Transfer Operations	SYS10/4-45
4.3.3.3	Interprocessor Interrupts	SYS10/4-45
4.3.3.4	Diagnostic and Miscellaneous Console Functions	SYS10/4-46
4.3.3.5	System Bootstrap Function	SYS10/4-46
4.3.4	Modes	SYS10/4-46
4.3.5	Interprocessor Communications	SYS10/4-47
4.3.5.1	Communication Areas	SYS10/4-47
4.3.5.2	Queue Processing/Messages	SYS10/4-47
4.4	MAIN MEMORY SUBSYSTEM	SYS10/4-48
4.5	SECONDARY MEMORY SUBSYSTEMS	SYS10/4-49
4.5.1	Disk Subsystems	SYS10/4-50
4.5.1.1	RHP04/06 and RTP04/06 Disk Subsystem	SYS10/4-50
4.5.1.2	RHS04 Disk Subsystem	SYS10/4-51
4.5.2	Magnetic Tape Subsystem	SYS10/4-52
4.5.2.1	TU70/71/72 Magnetic Tape Subsystem	SYS10/4-52
4.5.2.2	THU16/TTU16 Magnetic Tape Subsystem	SYS10/4-53
4.5.2.3	TU56 DECTape Subsystem	SYS10/4-53
4.6	BA10 UNIT RECORD (HARD COPY) EQUIPMENT	SYS10/4-53
4.6.1	CR10 Card Reader	SYS10/4-54
4.6.2	LP10 Line Printer	SYS10/4-54
4.6.3	CP10-D Card Punch	SYS10/4-55
4.6.4	XY10 Plotter	SYS10/4-55
4.7	LP100 LINE PRINTER SUBSYSTEM	SYS10/4-55
4.8	COMMUNICATIONS SUBSYSTEMS	SYS10/4-56
4.8.1	Communication Primer	SYS10/4-57
4.8.2	DC76 Asynchronous Communication Subsystem	SYS10/4-59
4.8.3	DC75-NP Synchronous Communications Subsystem	SYS10/4-60
4.8.4	DN87/DN87S Universal Synchronous/Asynchronous Front-End Subsystems	SYS10/4-61
4.8.4.1	Asynchronous Interfaces	SYS10/4-61
4.8.4.2	Synchronous Interfaces	SYS10/4-62
4.8.5	Communication Subsystem Components	SYS10/4-62
4.8.5.1	KG11-A XOR and CRC Unit (DC75-NP, DN87, and DN87S)	SYS10/4-62
4.8.5.2	DS11 Multiple Line Synchronous Interface (DC75-NP)	SYS10/4-64

CONTENTS (CONT)

		Page
4.8.5.3	DH11 Asynchronous 16-Line Multiplexer (DC76, DN87, and DN87S)	SYS10/4-64
4.8.5.4	DM11-BB Modem Control Unit (DC76, DN87, and DN87S)	SYS10/4-65
4.8.5.5	DQ11 Synchronous Line Interface (DN87 and DN87S)	SYS10/4-65
4.8.6	DC72-NP Remote Station	SYS10/4-66
4.8.7	DN80 (DAS80)-Series Remote Station	SYS10/4-67
4.8.8	DAS92 Remote Station	SYS10/4-69
SECTION 5	THE SOFTWARE	
5.1	RESIDENT OPERATING SYSTEM	SYS10/5-1
5.1.1	Command Decoder	SYS10/5-3
5.1.2	Scheduler	SYS10/5-3
5.1.3	Swapper	SYS10/5-4
5.1.4	Control Routine	SYS10/5-5
5.1.5	UUO Handler	SYS10/5-5
5.1.6	Device Service Routines	SYS10/5-5
5.1.7	Summary	SYS10/5-7
5.2	COMMAND CONTROL LANGUAGE	SYS10/5-7
5.3	FILE SYSTEM	SYS10/5-7
5.3.1	File Handler	SYS10/5-8
5.3.2	File Structures	SYS10/5-9
5.3.3	File Protection	SYS10/5-9
5.3.4	Disk Quotas	SYS10/5-9
5.3.5	File Operations	SYS10/5-10
5.3.6	Disk Storage Management	SYS10/5-10
5.4	INPUT/OUTPUT	SYS10/5-10
5.4.1	Peripheral Device Assignment	SYS10/5-10
5.4.2	Spooling	SYS10/5-10
5.5	MEMORY MANAGEMENT	SYS10/5-11
5.5.1	Secondary Memory	SYS10/5-11
5.5.2	Reentrant Software	SYS10/5-11
5.5.3	Virtual Memory	SYS10/5-11
5.6	MULTIPROCESSING SYSTEMS	SYS10/5-12
5.7	INTERJOB COMMUNICATIONS	SYS10/5-12
5.7.1	Shared Data Areas	SYS10/5-12
5.7.2	Interprocess Communication Facility	SYS10/5-12
5.8	NONRESIDENT SYSTEM SOFTWARE	SYS10/5-13
5.8.1	MACRO Assembler	SYS10/5-14
5.8.2	Compilers	SYS10/5-14
5.8.2.1	ALGOL	SYS10/5-14
5.8.2.2	BASIC	SYS10/5-15
5.8.2.3	COBOL	SYS10/5-15
5.8.2.4	FORTRAN	SYS10/5-16

CONTENTS (CONT)

		Page
5.8.3	Interpreters	SYS10/5-17
5.8.3.1	AID	SYS10/5-17
5.8.3.2	APL	SYS10/5-17
5.8.3.3	CPL	SYS10/5-18
5.8.4	Editors	SYS10/5-18
5.8.4.1	LINED	SYS10/5-18
5.8.4.2	TECO	SYS10/5-18
5.8.4.3	SOUP	SYS10/5-19
5.8.4.4	RUNOFF	SYS10/5-19
5.8.5	Utilities	SYS10/5-19
5.8.5.1	CREF	SYS10/5-19
5.8.5.2	DBMS-10	SYS10/5-20
5.8.5.3	DDT	SYS10/5-20
5.8.5.4	FAILSAFE/BACKUP	SYS10/5-20
5.8.5.5	FILEX	SYS10/5-20
5.8.5.6	ITPS-10	SYS10/5-21
5.8.5.7	LINK-10	SYS10/5-21
5.8.5.8	PIP	SYS10/5-22
5.8.5.9	MACY11 and LNKX11	SYS10/5-22
5.8.5.10	MCS-10	SYS10/5-22
5.8.6	Monitor Support Programs	SYS10/5-23
5.8.6.1	MONGEN	SYS10/5-23
5.8.6.2	OPSER	SYS10/5-23
5.8.6.3	LOGIN	SYS10/5-23
5.8.6.4	KJOB-LOGOUT	SYS10/5-23
5.9	CONSOLE SOFTWARE	SYS10/5-23
5.9.1	Basic Command Facility Description	SYS10/5-24
5.9.2	General System Bootstrap	SYS10/5-25
5.9.3	Diagnostic Environments	SYS10/5-25
5.9.4	Diagnostic Options	SYS10/5-25
5.9.5	System Failure Procedures	SYS10/5-26
5.10	COMMUNICATION SOFTWARE	SYS10/5-26
5.11	DIAGNOSTIC SOFTWARE	SYS10/5-27
5.11.1	Operating System Features	SYS10/5-27
5.11.2	Integrated Diagnostic Logic	SYS10/5-27
5.11.3	On-Line Preventive Maintenance	SYS10/5-28
5.11.4	Remote Diagnosis	SYS10/5-28
5.11.5	Monitor Error Reporting Programs	SYS10/5-28
5.11.5.1	Overview of the Monitor Error Reporting System	SYS10/5-28
5.11.5.2	Hardware Error Information	SYS10/5-28
5.11.5.3	Reporting Programs	SYS10/5-29
5.11.5.4	Testing Programs	SYS10/5-30
5.11.6	Diagnostic Programs	SYS10/5-30

APPENDIX A ABBREVIATIONS AND MNEMONICS

FIGURES

Figure No.	Title	Page
1-1	Swapping of User Programs	SYS10/1-3
1-2	Memory Structure	SYS10/1-3
1-3	Software Types	SYS10/1-4
1-4	User/Computer Communications	SYS10/1-5
1-5	File Structure	SYS10/1-7
2-1	DECsystem-10 Components	SYS10/2-1
2-2	A Typical DECsystem-1080	SYS10/2-2
2-3	A Typical DECsystem-1090	SYS10/2-3
2-4	Programs in the Batch System	SYS10/2-7
3-1	Instruction Set Constructs	SYS10/3-3
3-2	Move Instruction Construct	SYS10/3-4
3-3	Instruction Format	SYS10/3-7
3-4	User Process Table	SYS10/3-11
3-5	Executive Process Table	SYS10/3-12
4-1	DECsystem-1080 (Typical) Block Diagram	SYS10/4-3
4-2	DECsystem-1090 (Typical) Block Diagram	SYS10/4-7
4-3	DECsystem-1080 Central Processor Subsystem Block Diagram	SYS10/4-11
4-4	DECsystem-1090 Central Processor Subsystem Block Diagram	SYS10/4-17
4-5	Instruction, Dispatch, and Control Formats	SYS10/4-26
4-6	IBM Compatible Message Format	SYS10/4-63
4-7	BCC Computation for Transparent Transmission	SYS10/4-63
4-8	DC72-NP Remote Station	SYS10/4-66
4-9	DN80-Series Remote Station	SYS10/4-68
4-10	DAS92 Remote Station (Typical Configuration)	SYS10/4-69
5-1	The Resident Operating System	SYS10/5-2

TABLES

Table No.	Title	Page
4-1	Interleaving Configurations	SYS10/4-29
5-1	File Protection Scheme	SYS10/5-8



M0607

DECsystem 1080/1090

PREFACE

This manual contains the system level technical description of the DECsystem-1080/1090, the KL10-A/B-based machines. The companion interface and unit level technical descriptions and other related manuals are identified in the document titled *Introduction to KL10-Based System Technical Description* (EK-KL10-TD-XXX). There are five sections in this manual. It provides an integrated hardware/software systems presentation with appropriate conceptual overviews of time-sharing/batch operation, DECsystem-10 principles, and system features. The five sections comprising this system level presentation are:

1. Principles of Time-Sharing
2. DECsystem-10 Primer
3. System Features
4. The Hardware
5. The Software

The purpose of this manual is to provide a comprehensive overview of the KL10-based DECsystem-10 machines. Intended target population is field service, manufacturing, and customer training personnel and students.

SECTION 1 PRINCIPLES OF TIME-SHARING

1.1 INTRODUCTION

Early computers were the province of the mathematician. Used mainly to solve differential equations, the systems were narrow in scope and poorly used. Because few persons were knowledgeable enough to employ the enormous processors, one individual could monopolize computer time – sit at the console and solve problems in step-by-step fashion.

As more people discovered computing techniques, it was no longer practical to let a few persons monopolize computer time. To increase machine efficiency, **batch processing** was introduced. In this mode of operation, no time was wasted between jobs. Programs were punched on cards and the cards stacked and fed to the computer in batches. Operation of each program was governed by control cards that took the place of the human operator.

Since card reading is a relatively slow process, some early systems employed a small computer to read the cards and transfer program information to magnetic tape that was then input to the large computer. As a further refinement, programs were **assigned priorities**, with short jobs being executed first to minimize job turnaround.

But what about the computer user? As computer utilization improved, program development took more time. To develop a new program, a user performed the following procedure:

After writing the program on paper, he carried it to a keypunch operator to have the cards punched and verified. A day or so later, when the program was returned, the user checked for punching errors, then returned the cards to the keypunch operator for corrections.

Next, he sent the cards to the computer center for compilation. The compilation, which might not be returned for a half day or more, could reveal spelling or syntactical errors. The cards then had to be changed and resubmitted – another half day's wait. If the next compilation was successful and the program was run, program logic errors might be discovered, meaning new cards, new compilation, etc. In addition, the user often studied reams of computer listings to find the errors. Using these inefficient methods, even simple programs might take weeks to develop.

Batch processing maximizes machine efficiency in routine data processing operations where turnaround is not critical. But for program development and modification, the user requires another mode of operation. The user needs a way to *“interact”* with the computer – to feed his program to the system, line-by-line, and continuously check the results.

In fact, the user may want to develop **interactive programs**. These programs, which are extremely productive tools, ask the user questions and perform an analysis based on his answers. Electronic circuit design programs are a prime example. The computer actually designs the circuit by asking the engineer questions and formulating and manipulating data based on the answers. In addition, interaction provides a new dimension in management information reporting. Via an interactive terminal, a manager can request summaries, plot trends in plant operation and sales, and select special data for use in decision making.

If the user had unlimited funds, he might be tempted to buy or lease a large computer system he could **dedicate** to his work that would provide sufficient power, many peripherals, and a large variety of software. With such a system, the user could develop programs interactively or utilize batch processing for routine tasks. However, costs normally preclude the dedication of a large system to a single user.

By using **time-sharing**, the user has most of the benefits of a dedicated system at a small fraction of the cost. Time-sharing, with today's technology, allows a large powerful computer to handle 10, 50, 100, or more users simultaneously. Through a choice of terminals, the user can interface with the system or initiate batch processing which runs concurrently. The user also has access to a choice of mass storage devices and other peripherals as well as a selection of languages and application programs. Since response is fast, the user appears to have a dedicated system; yet costs are shared. Each user pays only for the time and facilities that he requires and does not pay for the time the machine is idle.

1.2 OPERATION

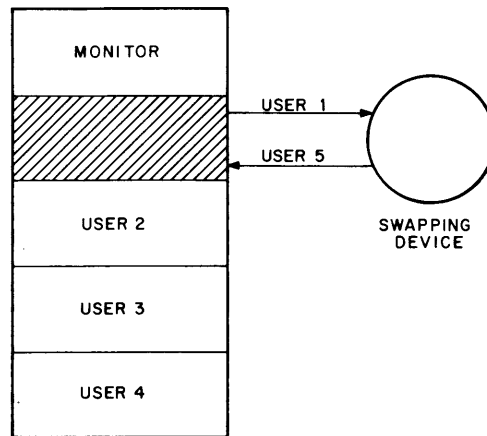
A time-sharing system is not just any computer with some additional hardware and software. It is a system such as DECsystem-10 designed specifically for time-sharing in an interactive computing environment. Otherwise, facilities are limited, fewer users can be handled efficiently, and the economics are unattractive. At a minimum, a time-sharing system requires a central processor with sufficient speed and power, input/output terminals, and an amount of **main memory** (core or semiconductor) adequate to hold several users.

In a simple time-sharing system, each program is assigned a fixed **time slice** (time quantum) and operation is switched from one program to another in **round robin** fashion until each program is completed. Essentially, if each user receives 1/60 of a second and 12 users are "on" the system, each user will receive service every 1/5 of a second.

The time-sharing system performs interactive **multiprogramming**; that is, it allows several programs to reside in main memory simultaneously and to operate sequentially. The switching between programs, called **context switching**, is initiated by a **clock** which interrupts the central processor to signal that a certain time period has elapsed. The interrupt function is provided by a **priority interrupt system**. A monitor, also called an **operating system** or **executive program**, directs the execution of these tasks and performs other housekeeping duties.

The monitor is also involved in keeping the actions of a user within his assigned memory space. An address mapping scheme where a page table containing the address and access keys is set up by the monitor, limits the core area that a particular user can access. Any attempt by the program to read or change information outside that limit will automatically stop the program and notify the monitor.

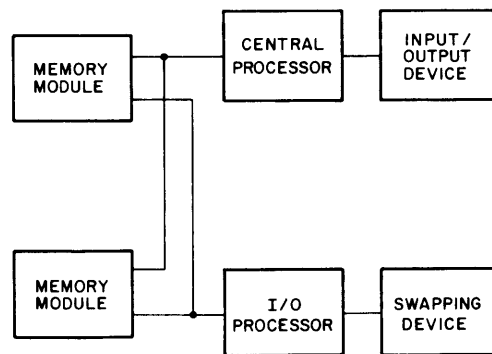
The system discussed so far services a number of users sequentially in round robin fashion. To increase the number of users serviced, more **main memory** is required. However, since core is expensive, a **secondary memory** is employed. This memory – usually magnetic disk or drum – is slower than core or main memory but provides greatly increased capacity at reasonable cost. User programs can be located in secondary memory and moved into main memory for execution. Programs entering main memory exchange places with a program (or programs) that has just been serviced by the central processor. This operation is called **swapping** (Figure 1-1).



10-1906

Figure 1-1 Swapping of User Programs

In operation, main memory is divided into separate **memory blocks** (modules). Secondary memory is connected to these modules through a high-speed **input/output processor** – a hardware device that allows the disk or drum to swap a program into any one of the main memory modules without any aid from the central processor. This structure allows the central processor to be operating a user program in one module of memory while programs are being swapped to and from another module. This independent overlapped operation, which greatly improves efficiency and processing power, is characteristic of an **asynchronous** system design philosophy (Figure 1-2).



10-1907

Figure 1-2 Memory Structure

1.2.1 Dynamic Scheduling

Round robin scheduling, in which each program operates in sequence and receives a fixed amount of time, is effective only if all programs have similar requirements. Such is not the case, however. At any particular time, a time-sharing system will be handling some programs which require extensive amounts of computing time (and are said to be **compute bound**) and other programs that must stop frequently for input or output (**I/O bound**).

To serve programs at and between these two extremes, the **scheduling algorithm** must provide frequent service to I/O bound programs and must give compute bound jobs longer time quanta (time slice) to prevent wasteful swapping. A simple dynamic scheme could provide two **queues** – one for each type of job. When a user first logs onto the system, he is placed in an I/O bound queue (waiting line) where he receives frequent service and small time quanta. If the program is not completed or does not request input or output during the time allotted, the job needs more computing time and is placed in the compute bound queue. Thus, the scheduling algorithm optimizes system efficiency by automatically adjusting to program requirements.

In the present state of scheduling art, algorithms are constantly being changed and improved. Current algorithms are extremely sophisticated, providing excellent service for most time-sharing job mixes. They also allow fine tuning, if such modifications are necessary. The ability of the algorithm to match processing to program requirements ensures the best service possible for all user programs.

In an efficient time-sharing system, monitor functions (referred to as **monitor overhead**) take 5 to 10 percent of central processor time, making 90 to 95 percent of the time available to users.

1.2.2 Software Sharing

Since users of large time-sharing systems have varying requirements, a good system provides a wide variety of software – interactive languages such as BASIC and AID for the computations of the engineer and scientist, FORTRAN for more complex calculations, COBOL for data processing functions. Therefore, many users can have compilers and other common programs in core at the same time.

To prevent excessive core usage that results when a program is duplicated for several users, **reentrant software** is employed. That is, the program is written in two parts. One part contains **pure code** (reentrant code) that is not modified during execution and can be used to simultaneously service any number of users. For example, the pure code portion of FORTRAN can service multiple FORTRAN users. A separate, second part of the program belongs strictly to each user and consists of the code and data that is developed during the compiling process (**impure code**). This section is stored in a separate area of core. A comparison of memory usage in the non-reentrant and reentrant systems is shown in Figure 1-3.

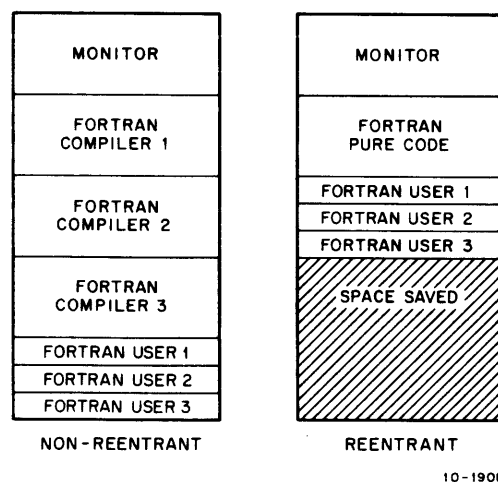


Figure 1-3 Software Types

What are the benefits of reentrant software? First, less core is required. For example, a reentrant system can service three FORTRAN users with one 8K compiler and three 2K user areas, a total of 14K. A non-reentrant system would require 30K for the three 8K compilers and three 2K user areas. Total savings in this case is 16K of core. Using less core means that more programs can fit into a given amount of space. The monitor then swaps less often and spends less time swapping the smaller impure sections.

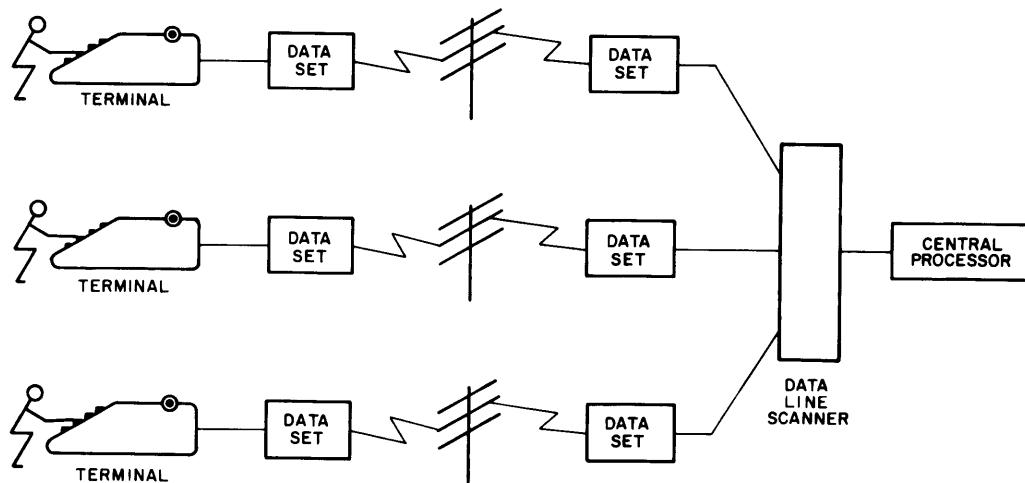
There are other savings too. Since the pure code never changes, it does not have to be returned to disk storage (swapped out). As long as a single copy is maintained on the disk, it can be called into core at any time. Programs can be swapped in or “overlaid” on top of the compiler to take its place in core whenever the compiler is not needed.

To protect the pure code from being modified, an address mapping feature, the pager, is implemented. This feature allows a program to execute as two separate segments where the pure segment is protected. User programs can also be written to make use of this protection. For example, a user might develop a reentrant information retrieval system written in COBOL.

1.2.3 Communications

Communication between the remote user and the computer passes over the conventional dial-up telephone network. User terminals can therefore be located anywhere that phone service is available and connected to any computer system; feasibility is limited only by long distance phone rates.

Each user terminal is connected to a **data set** or **modem** (modulator-demodulator) which converts user terminal output into a signal suitable for the telephone network. At the computer end of the phone lines, there is another data set which reconverts the signal and feeds it to a device called a **data line multiplexer** or **data line scanner**. This device, in turn, feeds the information from a number of terminals to the central processor (Figure 1-4).



10-1910

Figure 1-4 User/Computer Communications

The number of data sets employed at the user end of the system is unlimited. At the computer end of the communications network, however, the number of data sets is limited by the number of users that can be serviced simultaneously by the system.

In order to gain access to the system, the user dials the system phone number from his data set. The telephone network handles the call, scanning the data sets at the computer system. If all of the sets are busy, the user receives a busy signal, just as he would with normal phone service. If a set is available, the telephone network rings it, causing the data line scanner to interrupt the monitor. The computer answers the call, placing the user in communication with the monitor. The terminal is then **on-line** and ready for operation.

1.2.4 Control of Input/Output

A time-sharing system has performed its basic function if it allows a number of users simultaneous access to a central computer. However, to be fully useful, the system should also allow the users access to other system resources – storage devices for his programs and data, line printers, card readers, etc. For example, the user should be able to choose between magnetic tape and disk for program storage. If he has a 50-page report to produce, he should be able to employ a line printer instead of his terminal. If users controlled these devices, however, much confusion might result.

To prevent users from interfering with each other, the monitor coordinates input and output (I/O). The processor has an operating mode switch (user/executive mode) which the monitor sets before a user program is run. If the program attempts to perform input or output while in the user mode, the user program is stopped and the monitor takes over. Control thus diverted to the monitor is called **trapping**. When input/output is prevented or trapped, the computer is said to be in **user mode**; when I/O can be performed, the system is in **executive** or monitor mode.

User-to-monitor-mode switching occurs when the user requests I/O or other special functions to be performed by the monitor. The requests are made by using computer instructions referred to as **monitor calls** or **programmed operators**.

The monitor can also optimize throughput, keeping all devices busy simultaneously (**overlapping of I/O operations**) and executing jobs in the most efficient order. For example, it will start the read mechanisms on several disk packs in motion, simultaneously, to reduce the time required to find the desired data on each pack (**access time**). In addition, by means of the disk pack controller, the monitor can determine which of all needed data on a pack is closest to the read mechanism and can be obtained in the shortest amount of time (**latency optimization**).

1.2.5 File Handling

If a user does not require a fast device for his exclusive use (**private device**) he can elect to use a **public device** and, in effect, perform time-sharing with a disk or drum. Under these conditions, user programs and data coexist on the device. Therefore, a **filing system** is necessary if program and data segments are to be retrieved in proper order.

Data is transferred from memory to a peripheral device as a **block of words** or **record**. (A word is the number of binary digits or bits that the central processor can retrieve and “operate on” at one time.) Record length can be arbitrary or dictated by the physical device being used, e.g., the number of columns on an 80-column card or on a 132-column line printer. For DECsystem-10 disk files, the length is 128 words, so that blocks of 128 words are written at one time on a disk or other similar device.

For convenience each user's blocks are organized in groups called files which are listed in proper order in a special block on the disk called the **users' file directory** (UFD). A **master file directory** (MFD) is then required to maintain the locations of the UFDs and also keep track of the number of blocks of free storage that can be assigned to new files. The resulting hierarchy is shown in Figure 1-5.

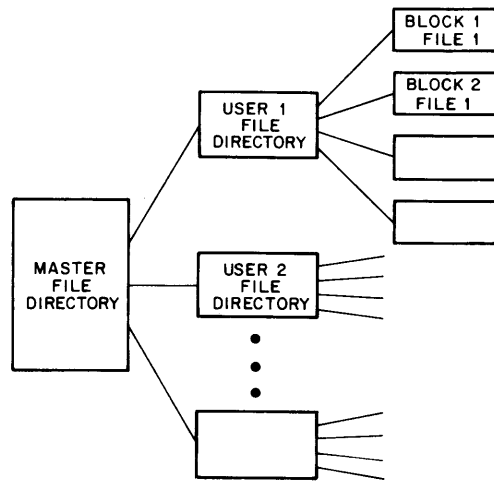


Figure 1-5 File Structure

Files, like memory, must be **protected** from access by unauthorized users. When a user creates a file, he can restrict it, specifying whether others can have access, and if access is permitted, whether the files can be modified or only read. With such an arrangement, programmers in various plant locations can use the same data to work simultaneously on the same project, but unauthorized personnel cannot modify or read the files.

1.2.6 Slow Peripherals

Fast peripherals can be time-shared. But what about the slow peripherals, such as the line printer and the card reader? Should other users be required to wait 20 minutes or so while one user ties up the line printer?

To eliminate conflicts, the user can request a slow device for his exclusive or private use. For example, he can request the line printer or card reader. Also available for private use are **removable storage devices** such as magnetic tape, DECTape (DIGITAL's low cost, high reliability magnetic tape), or disk packs. If the device is not already assigned to another user, the monitor, in cooperation with the operator, grants his request and the user has the device at his disposal until he releases it or logs off the system. For example, the user could request the use of multiple disk pack drives (exclusive use) to sort a payroll transaction file, or he could assign himself a DECTape drive and ask the system operator to mount the DECTape that contains his own personal library of programs.

Spooling is another method for handling data to or from slow peripherals. In this method, the slow device is simulated by a fast peripheral such as a disk. That is, all output for the line printer or card punch is deposited on the disk. The data on the disk is later "**unspooled**," with a special program transferring information to the slow device.

A program that has data for a slow device thus waits only milliseconds while the data is being deposited on a disk, instead of minutes or hours for a turn at the line printer. Input from slow devices can also be spooled, a particularly useful method for batch processing.

1.3 RELIABILITY

With a large number of users depending on its operation, the time-sharing system must be extremely reliable. A system with 99 percent reliability can be "down" 14 minutes during a 24-hour working day. If that 14 minutes affects only one user, reliability may be acceptable. However, if it affects a large number of users, the consequences are more serious.

The problem is also complicated by the fact that reliability is a function of both hardware and software. It may take years, for example, to experience all the events that could uncover an error in software as complex as a time-sharing monitor.

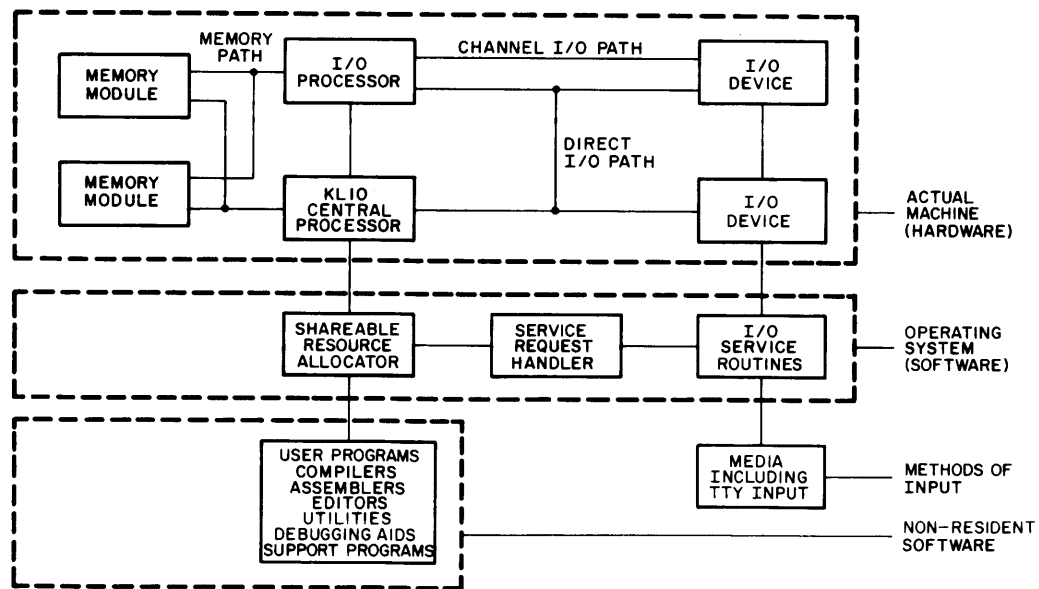
Today's hardware and software has reliability built in. Hardware is designed in **modular** fashion so that failed components can be removed and new replacements "plugged in." Some components also contain self-testing features that detect potential failures. Software is designed to be "defensive"; that is, it anticipates certain types of failures and helps to minimize their effects. For example, the software might note parity errors and limit their effect to the program being operated.

Diagnostic software can run routinely as one of the time-sharing users. Software can also maintain a log of failures, so that patterns can be established and problems remedied before serious damage occurs. Systems that employ these reliability techniques keep downtime at a minimum.

SECTION 2 DECsystem-10 PRIMER

2.1 INTRODUCTION

The DECsystem-10 is more than a processor and its associated peripheral devices. Because it is a system, there are many parts, or components, working together to achieve a goal in a manner that is both convenient for the user of the system and advantageous for the operation of the system. It is a machine designed to be utilized concurrently by many users who wish to perform various operations. There are three major components of the computing system, as shown in Figure 2-1: the actual machine, or hardware; the operating system, or monitor; and the languages and utilities, or non-resident software.

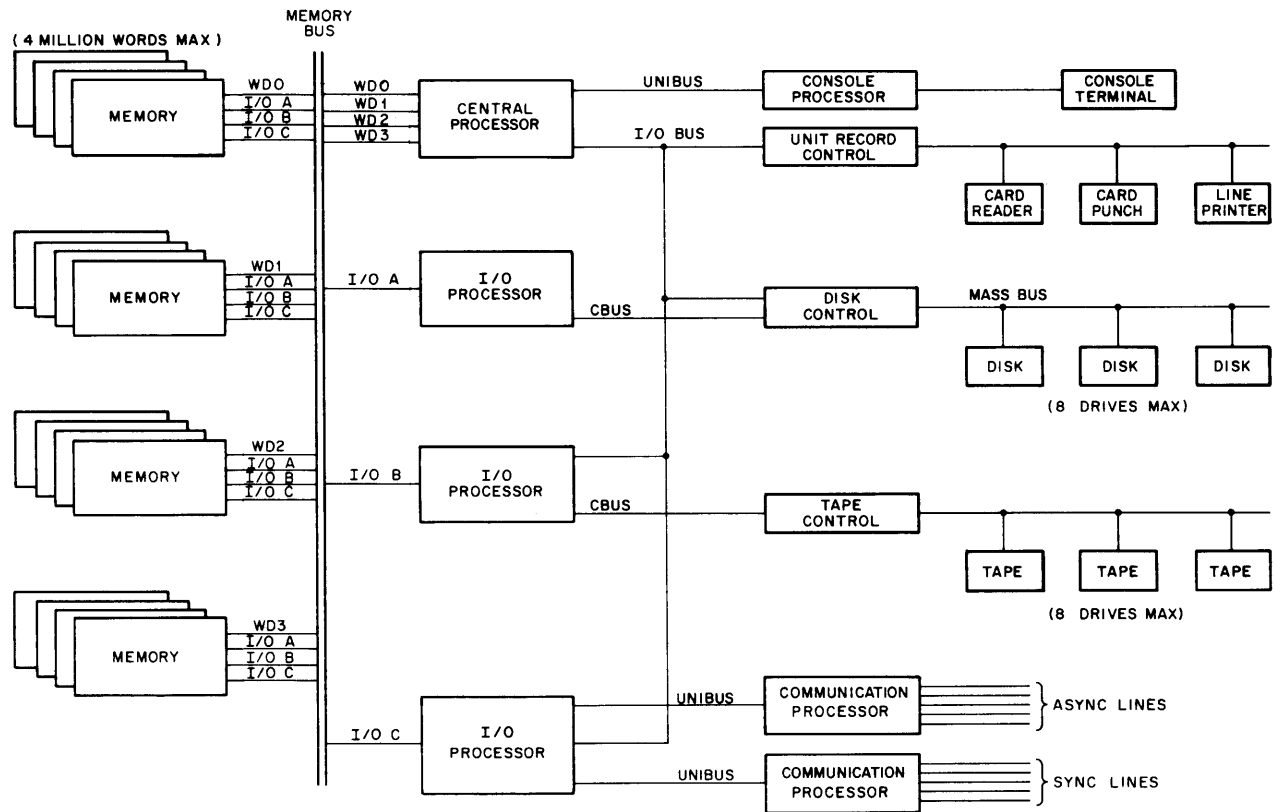


10-1911

Figure 2-1 DECsystem-10 Components

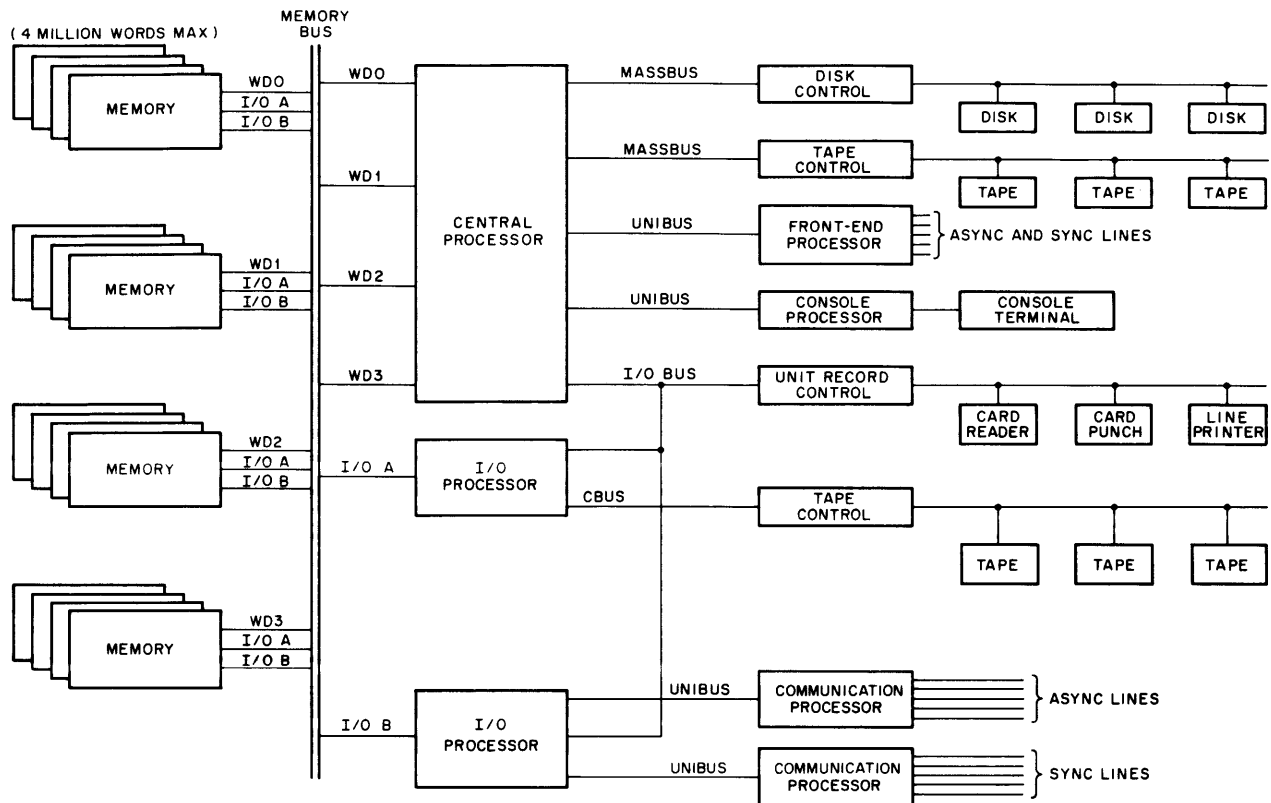
2.2 HARDWARE

The DECsystem-10 hardware consists of one or two central processors, various I/O processors, and various memories and input/output devices. There are now 11 basic system configurations included in the DECsystem-10 family; each configuration is distinguished by the hardware associated with the type and number of central processor and the type and number of I/O processors. By adding hardware to an individual system, additional performance is achieved. However, when adding hardware to expand from a small system to a larger system, no software changes are required in user programs. A single operating system and command control language can be used for all configurations of the DECsystem-10. Simplified block diagrams illustrating the 1080 and 1090 systems (the KL10-based systems) are shown in Figures 2-2 and 2-3.



10-1912

Figure 2-2 A Typical DECsystem-1080



10-2685

Figure 2-3 A Typical DECsystem-1090

2.3 OPERATING SYSTEM

The DECsystem-10 hardware has numerous capabilities: it is powerful, fast, and highly sophisticated. Because of its complexity, this machine is not usually manipulated directly by its users. The users communicate with an intermediary, the operating system, in order to direct their problems to the actual machine and to receive solutions back. With many users on the system, this second component of the DECsystem-10 must also keep track of what each user does and the devices and system resources that each user accesses. Though the operating system cannot be seen like the actual machine, the action of the operating system is the most important and noticeable part of the system to each user. It is true that the operating system can do nothing for the user if the actual machine does not exist, but the user normally does not think of this. If the operating system accomplishes for him what he wants the actual machine to do, he is satisfied. Therefore, it is important to the user that he depend on the same operating system regardless of the hardware that composes the actual machine.

The operating system is always resident in main memory of the actual machine and is composed of three parts (Figure 2-1). Because there are so many services that can be obtained from the operating system, including the allocation of core memory, processor time, and devices of the actual machine, one part, the service request handler, is responsible for accepting requests for these services. The service request handler passes the requests to another part, the sharable resource allocator, which is responsible for allocating the services requested. If the requested service is for use of a device, the I/O service routines are then notified to carry out the user's request.

2.4 NONRESIDENT SOFTWARE

The third component of the DECsystem-10 is the nonresident software, those programs necessary for the varied operation of the computing system. This software includes the compilers, assemblers, editors, debugging programs, and operating system support programs. These software programs reside on a high-speed mass storage device of the actual machine and are brought into memory when needed by a user. By utilizing the nonresident software, the user of the computing system can create programs, transfer them from one device to another, compile, edit, execute, and debug them, and then receive the results of execution on any specified device.

2.5 MULTIPROCESSING

The DECsystem-10 can be a single-processor system or a dual-processor system, composed of a primary processor and a secondary processor. Each processor in the dual-processor system runs user programs, schedules itself, fields instruction traps, and processes most requests to the operating system. In addition to these tasks, the primary processor also has control of all the input/output devices. This is because only the primary processor has access to the I/O bus. The primary processor completes any job that the secondary processor could not finish because of an I/O request to the operating system. Both processors are connected to the same memory and execute the same copy of the operating system, thereby saving core memory over a multiprocessing system in which each processor has its own copy.

The primary objective in the DECsystem-10 dual-processor environment is to provide more processing power than that found in the single-processor DECsystem-10. This means that with the addition of the second processor, more users can run at the same time. If more users are not allowed on the system, the addition of the second processor reduces the elapsed time required to complete the processing of most programs.

2.6 MULTIMODE COMPUTING

The DECsystem-10 is designed for the concurrent operations of time-sharing, multiprogram batch, real-time, and remote communications in either single- or dual-processor systems. In providing these multifunction capabilities, the DECsystem-10 services interactive users, operates local and remote batch stations, and performs data acquisition and control functions for on-line laboratories and other real-time projects. By dynamically adjusting system operation, the DECsystem-10 provides many features for each class of user and is therefore able to meet a large variety of computational requirements.

2.6.1 Time-Sharing

Time-sharing takes maximum advantage of the capabilities of the computing system by allowing many independent users to share the facilities of the DECsystem-10 simultaneously. Because of the interactive, conversational, rapid-response nature of time-sharing, a wide range of tasks – from solving simple mathematical problems to implementing complete and complex information gathering and processing networks – can be performed by the user. The number of users on the system at any one time depends on the system configuration and the job mix on the system. DECsystem-10 time-sharing is designed to allow for up to 150 command terminals. Interactive terminals can include CRTs, hard copy terminals, and other devices which operate at speeds of 110 to 9600 baud. Terminal users can be located at the computer center or at remote locations connected to the computer center by communication lines.

Time-sharing on the DECsystem-10 is general-purpose, i.e., the system is designed in such a way that the command language, input/output processing, file structures, and job scheduling are independent of the programming language being used. In addition, standard software interfaces make it easy for the user to develop his own special language or systems. This general-purpose approach is demonstrated by the many special-purpose programming languages implemented by DECsystem-10 users.

2.6.1.1 Command Language – By allowing resources to be shared among users, the time-sharing environment utilizes processor time and system resources that are wasted in single-user systems. Users are not restricted to a small set of system resources, but instead are provided with the full variety of facilities. By means of his terminal, the user has on-line access to most of the system's features. This on-line access is available through the operating system command language, which is the means by which the time-sharing user communicates with the computing system.

Through the command language, the user controls the running of his job, or task, to achieve the results he desires. He can create, edit, and delete his files; start, suspend, and terminate his job; compile, execute, and debug his program. In addition, since multiprogramming batch software accepts the same command language as the time-sharing software, any user can enter his program into the batch run queue. Thus, any time-sharing terminal can act as a remote job entry terminal.

2.6.1.2 Peripheral Devices – With the command language, the user can also request assignment of any peripheral device, e.g., magnetic tape, DECtape, and private disk pack, for his own exclusive use. When the request for assignment is received, the operator verifies that the device is available to this user, and the user is granted its private use until he relinquishes it. In this way, the user can also have complete control of devices such as card readers and punches, paper tape readers and punches, and line printers.

2.6.1.3 Spooling – When private assignment of a slow-speed device (e.g., card punch, line printer, paper tape punch, and plotter) is not required, the user can employ the spooling programs of the operating system. Spooling is a method by which output to a slow-speed device is placed on a high-speed disk or drum. This technique prevents the user from using unnecessary time and space in core while waiting for either a device to become available or output to be completed. It also allows the user to defer output until it is needed. In addition, the device is managed to a better degree because the users cannot tie it up indefinitely, and the demand fluctuations experienced by these devices are equalized.

2.6.1.4 Mass Storage File System – Mass storage devices, such as disks and drums, usually cannot be requested for a user's exclusive use, but must be shared among all users. Because many users share these devices, the operating system must ensure independence among the users; one user's actions must not affect the activities of another unless the users desire to work together. To guarantee such independence, the operating system provides a file system for disks, disk packs, drums, and DECtapes. Each user's data is organized into one or more 128-word blocks called files. The user gives a name to each of his files, and the list of these names is kept by the operating system for each user. The operating system is then responsible for protecting each user's file storage from intrusion by unauthorized users.

In addition to allowing independent file storage for users, the operating system permits sharing of files among individual users. For example, programmers working on the same project can share the same data in order to complete a project without duplication of effort. The operating system lets the user specify protection codes, or rights, for his files. These codes designate if other users may read the file, and after access, if the files can be modified in any way.

The user of the DECSYSTEM-10 is not required to preallocate file storage; the operating system allocates and deallocates the file storage space dynamically on demand. Not only is this convenient for the user because he does not have to worry about allocation when he is creating files, but this feature also conserves storage by preventing large portions of storage from being unnecessarily tied up. However, a user can preallocate file storage to guarantee its availability.

2.6.1.5 Core Utilization – The DECSYSTEM-10 is a multiprogramming system, i.e., it allows multiple independent user programs to reside simultaneously in core and to run concurrently. This technique of sharing core and processor time enhances the efficient operation of the system by switching the processor from a program that is temporarily stopped because of I/O transmission to a program that is executable. When core and the processor are shared in this manner, each user's program has a memory area distinct from the area of the other users. Any attempt to read or change information outside the user access area immediately stops the program and notifies the operating system.

Because available core can contain only a finite number of programs at any one time, the computing system employs a secondary memory, usually disk or drum, to increase the number of users serviced. User programs exist on the secondary memory and move into core for execution. Programs in core change places with the programs being transferred from secondary memory for maximum use of available core. Because the transferring, or swapping, takes place directly between core and the secondary memory, the central processor can be operating on a user program in one part of core while swapping is taking place in another. This independent overlapped operation greatly improves system utilization by increasing the number of users that can be accommodated at the same time.

To further increase the utilization of core, the operating system allows the users to share the same copy of a program or data segment. This prevents the excessive core usage that results when a program is duplicated for several users. A program that can be shared is called a reentrant program and is divided into two parts or segments. One segment contains the code that is not modified during execution (e.g., compilers and assemblers) and can be used by any number of users. The other segment contains the user's code and data that are developed during the compiling process. The operating system invokes protection for shared segments to guarantee that they are not accidentally modified.

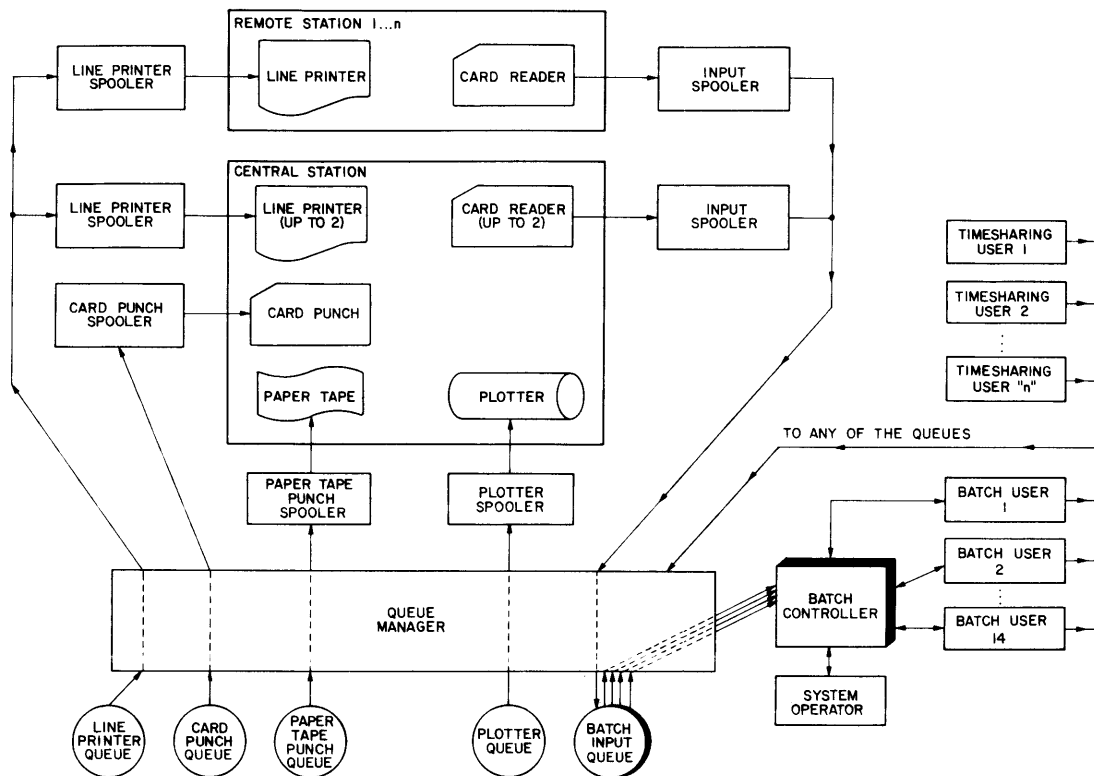
The virtual memory option permits a user program to execute with an address space greater than the physical memory actually allocated to that program during execution. User jobs are swapped as described above. However, the entire program may not necessarily be in core during execution. Programs are divided into pages, each of which is 512 words long. Some of these pages may remain on secondary storage while the program executes. When a virtual memory job attempts to access a page that is not in core, a page fault handler decides which page or pages to remove from core and which to bring in from secondary storage.

2.6.1.6 General-Purpose Time-Sharing – Time-sharing on the DECsystem-10 is general-purpose, i.e., the system is designed in such a way that the command language, input/output processing, file structures, and job scheduling are independent of the programming language being used. In addition, standard software interfaces make it easy for the user to develop his own special language or systems. The general-purpose approach is demonstrated by the many programming languages implemented by DECsystem-10 customers.

2.6.2 Batch

Batch software (GALAXY-10) enables the DECsystem-10 to execute up to 128 batch jobs concurrently with time-sharing jobs. Just as the time-sharing user communicates with the system by way of his terminal, the batch user normally communicates by way of the card reader. (However, he can enter his job from an interactive terminal.) Unlike the time-sharing user, the batch user can punch his job on cards, insert a few appropriate control cards, and leave his job for an operator to run. In addition, the user can debug his program in the time-sharing environment and then run it in batch mode without additional coding.

2.6.2.1 Components – The GALAXY-10 system (Figure 2-4) consists of a series of programs: the input spooler, SPRINT; the batch controller, BATCON; the system queue manager and scheduler, QUASAR; and the output spoolers, LPTSPL and SPROUT. The input spooler is responsible for reading the input from the input device and for entering the job into the batch controller's input queue. The input spooler sends a message to QUASAR to do the actual entering into the batch input queue. Although the input spooler is oriented toward card reader input, disk and magnetic tape also can be handled. The input information is then separated according to the control commands in the input deck and placed into disk files for subsequent processing. In addition, the input spooler creates the job's log file and enters a report of its job processing, along with a record of any operator intervention during its processing. The log file is part of the standard output that the user receives when his job terminates.



10-1913

Figure 2-4 Programs in the Batch System

After the input spooler reads the end-of-file and closes the disk files, it makes an entry in the batch controller's input queue. The batch controller processes batch jobs by reading the entries in its queue. The control file created by the input spooler is read by the batch controller, and data and nonresident software commands are passed directly to the user's job. Operating system commands are detected by the batch controller and passed to the operating system for action. Most operating system and nonresident software commands available to the time-sharing user are also available to the batch user. Therefore, only one control language need be learned for both time-sharing and batch. During the processing of the job and the control file, the batch controller adds information to the log file for later analysis by the user.

QUASAR 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 under the batch controller according to external priorities, processing time limits, and core requirements which are dynamically computed according to parameters specified by the user for his job, such as start and deadline time limits for program execution. QUASAR makes an entry for the job in the batch input queue based on the various priorities. After the job is completed, an entry is made in the output queues for the job's spooled output and the job's log file.

The output spooling programs improve 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 a particular output device. The log file and all job output are placed into one or more output queues to await processing. When the specified device is available, the output is then processed by the appropriate spooling program. These spooling programs may be utilized by all users of the computing system.

2.6.2.2 Use of System Features – The GALAXY-10 batch software employs many of the computing system's features in order to operate with maximum efficiency. Because core memory is not partitioned between batch and time-sharing jobs, batch jobs can occupy any available area of core. Fast throughput for high priority batch jobs is accomplished with the same swapping technique used for rapid response to interactive users. When available core is not large enough for a high priority batch job, the operating system transfers programs of lower priority to secondary memory in order to provide space for the job. This transfer is done at the same time that the processor is operating on another job. Thus, processing can be overlapped with swapping (and other I/O) to utilize time that would otherwise be wasted. Batch jobs can also share programs with time-sharing and other batch jobs. Only one copy of a sharable program need be in core to service any number of batch and time-sharing jobs at the same time.

2.6.2.3 Flexibility – GALAXY-10 batch allows the user great flexibility. The input spooler normally reads from the card reader, but can read from magnetic tape or disk in order to create a control file on a disk and to enter the job into the batch controller's input queue. However, a job can be entered from an interactive terminal, in which case the user bypasses the input spooler and creates a control file on the disk for the batch controller. The control file contains the operating system commands and non-resident software commands necessary to run the job. The user then enters the job into the batch controller's input queue by way of an operating system command string. In the command string, the user can include switches to define the operation and set the priorities and limits on core memory and processor time.

2.6.2.4 Job Dependency – Although jobs are entered sequentially into the batch system, they are not necessarily run in the order that they are read because of priorities, either set by the user in an input spooler control command or computed by the queue manager when determining the scheduling of jobs. Occasionally, the user may wish to submit jobs that must be executed in a particular order; in other words, the execution of one job is dependent on another. To ensure that jobs are executed in the proper order, the user must specify an initial dependency count in a control command of the dependent job. This dependency count is then part of the input queue entry. A control command in the job on which the dependent job depends decrements the count. When the count becomes zero, the dependent job is executed.

2.6.2.5 Error Recovery – The user can control system response to error conditions by including commands to the batch controller to aid in error recovery. These commands are copied into the control file by the input spooler. With error recovery commands, the user specifies the action to be taken when his program contains a fatal error, as for example, to skip to the next program or to transfer to a special user-written error handling routine. If an error occurs and the user did not include error recovery conditions in his job, the batch controller initiates a standard dump of the user's core area and terminates the job. This core dump provides the user with the means to debug his program.

Although the batch system allows a large number of parameters to be specified, it is capable of operating with very few user-specified values. If a parameter is missing, the batch system supplies a reasonable default value. These defaults can be modified by the individual installations.

2.6.2.6 Operator Intervention – Normal operating functions performed by the programs in the batch system require little or no operator intervention; however, the operator can exercise a great deal of control if necessary. He can specify the number of system resources to be dedicated to batch processing by limiting the number of programs and both 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 programs in the batch system can communicate information to the operator and record a disk log of all messages printed at the operator's console. All operator intervention during the running of the input spooler and the batch controller causes messages to be written in the user's log file, as well as in the operator's log file, for later analysis.

2.6.3 Real-Time

For a system to be satisfactory for real-time operations, two important requirements must be met. The more important requirement is fast response time. Because real-time devices cannot store their information until the computing system is ready to accept it, the system would be useless for real time if the response requirements of a real-time project could not be satisfied. The operating system must allocate system resources dynamically, in order to satisfy the response and computational requirements of real-time jobs.

The second requirement is protection. Each user of the computing system must be protected from other users, just as the system itself is protected from all user program errors. In addition, since real-time systems have special real-time devices associated with jobs, the computing system must be protected from hardware faults that could cause system breakdown. Because protection is part of the function of the operating system, the real-time software employs this feature to protect users, as well as itself, against hardware and software failures. Inherent in the operating system is the capability of real time, and it is through calls to the operating system that the user obtains real-time services. The services obtained by calls within the user's program include:

1. Locking a job in core
2. Connecting a real-time device to the priority interrupt system
3. Placing a job in a high-priority run queue
4. Initiating the execution of FORTRAN or machine language code on receipt of an interrupt
5. Disconnecting a real-time device from the priority interrupt system.

2.6.3.1 Locking Jobs – Memory space is occupied by the resident operating system and by a mix of real-time and nonreal-time jobs. The only fixed partition is between the resident operating system and the remainder of memory. Since a real-time job may need to be in memory to avoid losing information when its associated real-time device interrupts, the job can request that it be locked into core. This means that the job is not to be swapped into secondary memory and guarantees that the job is readily available when needed. Because memory is not divided into fixed partitions, it can be utilized to a better degree by dynamically allocating more space to real-time jobs when real-time demands are high. As real-time demands lessen, more memory can be made available to time-sharing and batch usage.

2.6.3.2 Real-Time Devices – The real-time user can connect real-time devices to the priority interrupt system, respond to these devices at interrupt level, remove the devices from the interrupt system, and/or change the priority interrupt level on which these devices are assigned. There is no requirement that these devices be connected at system generation time. The user specifies both the names of the devices generating the interrupts and the priority levels on which the devices function. The operating system then links the devices to the operating system.

The user can control the real-time device in one of two ways: single mode or block mode. In single mode, the user's interrupt program is run every time the real-time device interrupts. In block mode, the user's interrupt program is run after an entire block of data has been read from the real-time device. When the interrupt occurs from the device in single mode or at the end of a block of data in block mode, the operating system saves the current state of the machine and jumps to the user's interrupt routine. The user services his device and then returns control to the operating system to restore the previous state of the machine and to dismiss the interrupt. Any number of real-time devices may be placed on any available priority interrupt channel.

2.6.3.3 High-Priority Run Queues – The real-time user can receive faster response by placing jobs in high-priority run queues. These queues are examined before all other run queues in the computing system, and any runnable job in a high-priority queue is executed before jobs in other queues. In addition, jobs in high-priority queues are not swapped to secondary memory, until all other queues have been scanned. When jobs in a high-priority queue are to be swapped, the lowest priority job is swapped first and the highest priority job last. The highest priority job swapped to secondary memory is the first job to be brought into core for immediate execution. Therefore, in addition to being scanned before all other queues for job execution, the high-priority queues are examined after all other queues for swapping to secondary memory and before all other queues for swapping from secondary memory.

2.6.3.4 Job Communication – The DECsystem-10 operating system enables a real-time user to communicate with other jobs through the use of sharable data areas. This also enables a data analysis program, for example, to read or write an area in the real-time job's core space. Since the real-time job associated with the data acquisition would be locked in core, the data analysis program residing on secondary memory would become core resident only when the real-time job had filled a core buffer with data. Operating system calls can be used to allow the data analysis program to remain dormant on secondary memory until a specified event occurs in the real-time job, e.g., a buffer has been filled with data for the data analysis program to read. When the specified event occurs, the dormant program is then activated to process the data. The core space for the real-time job's buffer area or the space for the dormant job does not need to be reserved at system generation time. The hardware working in conjunction with the operating system's core management facilities provides optimum core usage.

SECTION 3 SYSTEM FEATURES

3.1 CENTRAL PROCESSORS

Within the family of 11 DECsystem-10 configurations, there are 3 central processor units. All operate under the same DECsystem-10 Total Operating System (TOPS-10), execute the same software, and share most models of DECsystem-10 peripheral equipment. The three processors differ in their speed, method of address mapping, memory capacity, program size, auto-diagnostic and restart features, and priority interrupt system.

The KA10 Central Processor is used in the DECsystem-1040, 1050, and 1055. The more powerful KI10 Central Processor is used in the DECsystem-1060, 1066, 1070, and 1077 configurations. The KL10 Central Processor is used in the DECsystem-1080, 1088, 1090, and 1099. The KL10 Central Processor departs from the traditional DECsystem-10 architecture in that it can be equipped with cache memory and internal high-speed data/communication channels. Because all processors operate with the same operating system, it is easy to field-upgrade a system with a KA10 and replace it with a KI10 processor or take a system with a KI10 and replace it with a KL10 processor, providing a significant increase in the computing power with no necessary changes in user programs.

The 1090 configuration differs from the 1040, 1060, or 1080 in that the high-speed data channels and communications processor channels are integral to the central processor. Up to eight high-speed buffered data channels can be utilized to support DIGITAL's Massbus peripherals. Because the data channels access four 36-bit words of memory in one channel request cycle, the total throughput is in excess of one million words per second or four megabytes per second. For non-Massbus peripherals, external channels which transfer directly to memory can be utilized. An important feature of the integrated high-speed data channels is that cache memory is automatically updated during I/O transfers. The integrated front-end processor channels will accommodate up to three PDP-11 front-end processors in addition to the KL10 console front-end. These channels utilize high-speed microcode to transfer and process data coming from the PDP-11 front-ends. All channel data transfers are parity checked. Other benefits of the 1090 internal channels are configuration simplicity, improved maintainability, and reduced power consumption.

3.2 INSTRUCTION SET

DECsystem-10 offers up to 398 instructions, an extremely large set which provides the flexibility required for specialized computing problems. The instruction sets of the KA10 and KI10 are hard-wired. By contrast, the instruction set of the KL10 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 1280-by 75-bit word RAM (random access memory) through the PDP-11 console front-end processor. Since the set provides so many instructions to choose from, fewer instructions are required 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 are often 30 to 50 percent shorter, require less memory, and execute faster than those of comparable computers.

In addition to these instructions, DECsystem-10 features 64 programmable operators, 33 of which “trap” to the monitor (monitor calls) and 31 of which trap to the user’s core area. The remaining instructions are unimplemented and reserved for future expansion. An attempt to execute one of these unimplemented instructions results in a trap to the monitor.

The instruction set, despite its size, is easy to learn. It is logically grouped into families of instructions and the mnemonic code is constructed modularly (Figure 3-1). All instructions are capable of directly addressing a full 256K (36-bit) words of memory without resorting 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 execution time.

Although there are a large number of instructions, they may be broken into easily learned, logically completed groups (Figure 3-1). To illustrate this, the group to move full words (36 bits) takes the form shown in Figure 3-2.

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 2’s complement
- M = take magnitude
- S = swap left and right halves.

3. One of four modifiers is chosen from the operator 2 group. This group specifies the location of the word to be fetched and where it is to be placed,

thus:

- (blank) = memory to accumulator
- I = take 18-bit address as operand (immediate mode)
- M = accumulator to memory
- S = memory to memory (self).

This simple example serves to show the power and flexibility built into the entire DECsystem-10 instruction set. With one broad instruction class, a total of 16 instructions has been formed.

For example:

MOVE AC, ADR = moves content of ADR to specified AC

MOVEM AC, ADR = moves content of specified AC to ADR

MOVEI AC, 5 = moves the number 5 to AC

MOVNM AC, ADR = moves the complemented content of AC to ADR.

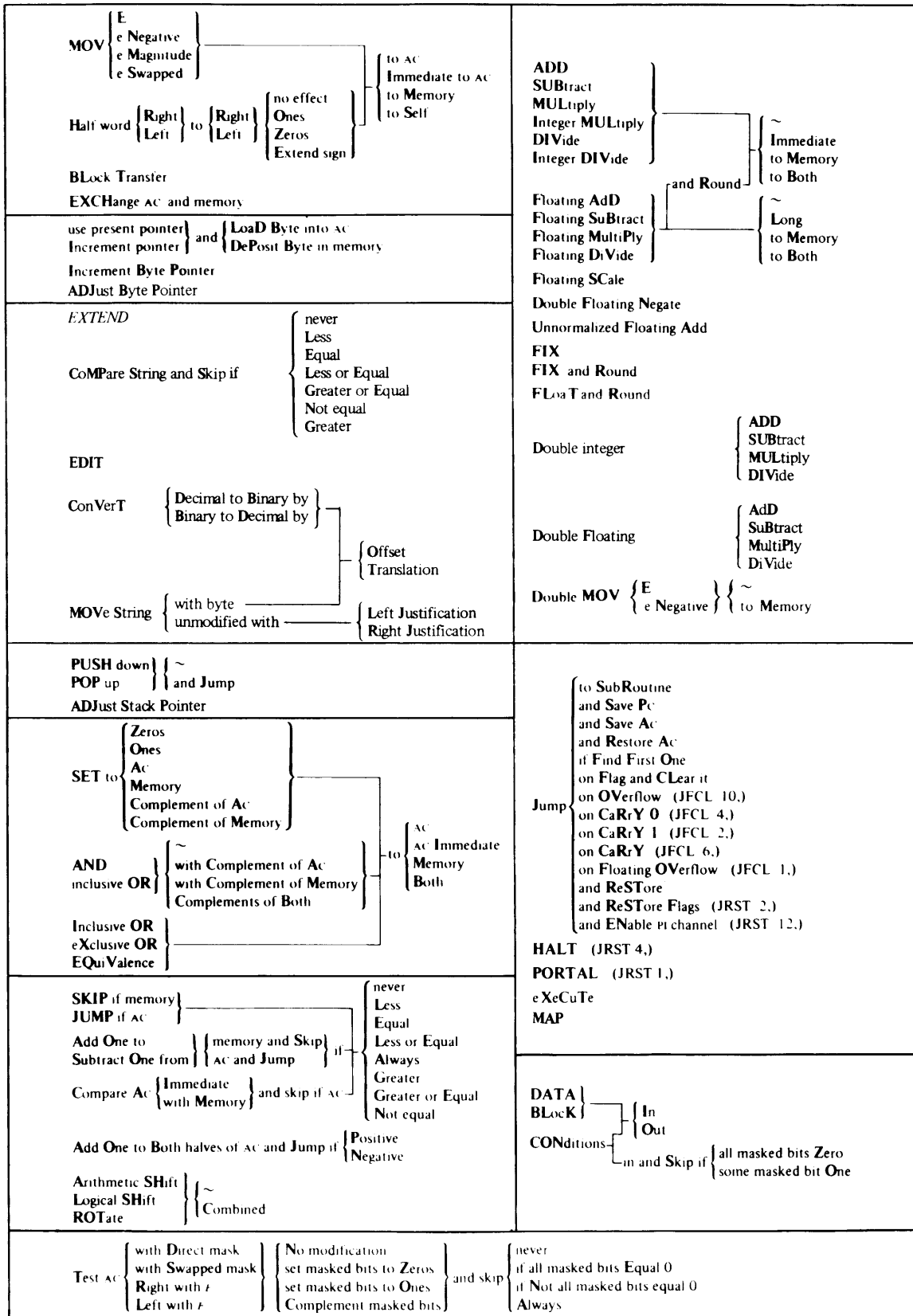


Figure 3-1 Instruction Set Constructs

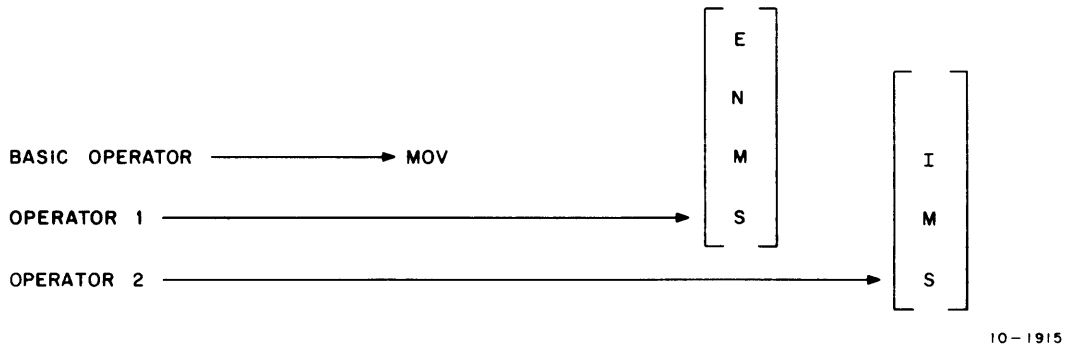


Figure 3-2 Move Instruction Construct

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

3.2.1 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 forming the negative (2's complement) or the magnitude of the word being processed.

3.2.2 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 which differ in the direction that they move the chosen half-word and in the manner in which they modify the other half of the destination location.

3.2.3 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, and for moving the block from any memory location to any other memory location.

3.2.4 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 DECsystem-10, 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 DECsystem-10 7-bit bytes are efficiently stored 5 to a word. All the byte instructions utilize a byte pointer which allows addressing of any size byte in any position in any of the 262,144 addressable words. Further, both load and deposit byte instructions have a provision for automatic byte incrementation.

3.2.5 Business Instruction Set

Five instructions comprise the new business instruction set implemented in the KL10-based DECsystem-10 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 which use translation can control the flags in accumulators 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 variety of code types including ASCII, EBCDIC, etc.

3.2.6 Logic Instructions

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

3.2.7 Fixed-Point Arithmetic

Fixed-point arithmetic is handled in 2's complement notation with 36 binary-bit accuracy (10 decimal digits). Mode options include immediate, to accumulator, to memory, or to both with result. Three classes of shifting include arithmetic, logical, and rotating operations to single- or double-word accumulators.

3.2.8 Floating-Point Arithmetic

The floating-point arithmetic instructions include instructions to perform scaling, negating (form 2'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.

3.2.9 Arithmetic Operation Modes

All of the DECsystem-10 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.

3.2.10 Fixed/Floating Conversions

Special instructions provide the capability of 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.

3.2.11 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 which 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.

3.2.12 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 time-shared monitor and all of the reentrant systems programs.

3.2.13 Input/Output

Input/output over the EBus and I/O bus is handled by eight straightforward instructions. Each instruction may reference 1 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 a device in an efficient manner.

3.2.14 Unimplemented User Operations (UOs)

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

3.2.15 Trap Handling

DECsystem-10 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 which caused the trap.

3.3 INSTRUCTION FORMAT

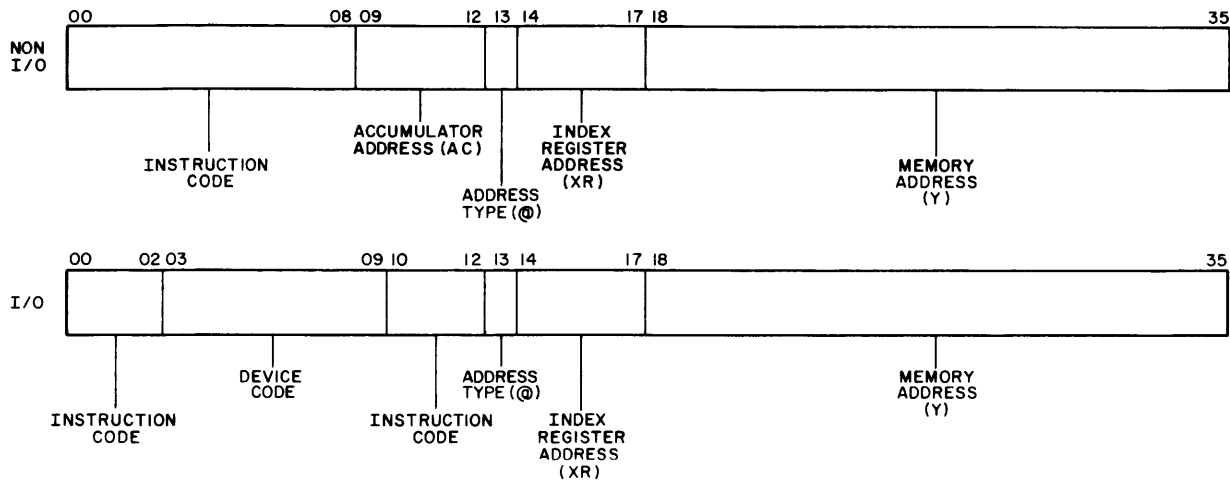
In all the non-input/output instructions, the nine high-order bits (0-8) specify the operation, and bits 9-12 usually address an accumulator but are sometimes used for special control purposes such as addressing flags (Figure 3-3). The rest of the instruction word always supplies information for calculating 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 18 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," as are the codes for floating-point and byte manipulation in any KA10 that does not have the hardware for these instructions.

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.

3.4 NUMBER SYSTEM

The standard arithmetic instructions in the DECsystem-10 use 2'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 remaining 35 bits represent the magnitude in ordinary binary notation. The negative of a number is obtained by taking its 2's complement. Zero is represented by a word containing all 0s.



10-1916

Figure 3-3 Instruction Format

3.4.1 Fixed-Point Arithmetic

Two common conventions are to regard 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 yield results that can be represented by a single word.

The format for double-length fixed-point numbers is just an extension of the single-length format. The magnitude (or its 2'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 thus -2^{70} to $2^{70} - 1$ or -1 to $1 - 2^{-70}$.

3.4.2 Floating-Point Arithmetic

The KL10-based DECSYSTEM-10 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 which ranges in magnitude from $1/2$ to $1 - 2^{-27}$. Increasing the length of a number to two words does not significantly change the range but rather 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 $+127$.

3.5 EFFECTIVE ADDRESS CALCULATION

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

1. Obtain 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 non-zero, then add the contents of the specified index register to the number obtained in step 1.
3. Obtain 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 obtain a new word from memory, and go back to step 1.

The effective address calculation continues until a word is encountered 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 carried out 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.

3.6 GENERAL-PURPOSE REGISTER BLOCKS

General-purpose registers are another DECsystem-10 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 ns.

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.

3.7 MEMORY SYSTEM

3.7.1 Core Memory

To meet the requirements of large systems, DECsystem-10 core memory can be modularly expanded to 4096K words for KL10-based systems. Memory can consist of combinations in 16K, 32K, 64K, 128K, and 256K-word multiport modules. The structure of the memory bus gives the central processor and external high-speed data channels simultaneous access to separate memory modules and allows each to operate at its own speed.

Each MF10 memory module contains up to four ports and each port can be further expanded through the use of the MX10 memory bus multiplexer. MG10 and MH10 memory modules contain up to eight ports, reducing the need for the MX10 multiplexer. The multiplexer handles up to eight channels, interleaving data from the channels on a word-by-word priority basis. Such parallel operation yields many improvements over systems which provide only a single path to memory. The memory bus system allows each data channel to transmit full 36-bit words in parallel at a speed of one million words per second for KL10-based systems. Each memory module provides switches which allow that particular module to represent any module of its size in the addressable memory space. Thus, one module can replace another without rewiring. Switches are also provided for memory interleaving.

The memory bus architecture of the KL10 differs from both the KA10 and KI10 in that its width for the central processor unit (CPU) is four 36-bit words. That is, the KL10 CPU actually reads/writes four words from memory simultaneously. Combined with the effects of the cache memory, the KL10 has therefore an effective access time of approximately 300 ns when using 1 μ s memories such as the MF10 or MG10.

3.7.2 Cache Memory

The KL10-based DECsystem-10 features a high-speed cache or buffer memory. Data being read from memory is typically found in the cache 90 to 95 percent of the time, thus giving DECsystem-10 an effective memory access time of approximately 300 ns with 1 μ s memories such as the MF10, MG10, or MH10. Another feature of the "state-of-the-art" design cache memory is that unlike contemporary designs, it does not require write-through to memory. Rather, 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 composed 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.

3.8 PROCESSOR MODES

Instructions on the KL10 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, addresses are not relocated, and all core locations are accessible. The monitor operates in executive mode and is able to control all system resources and the state of the processor. In user mode operations, addresses are relocated, certain instructions are illegal, causing monitor traps when executed, and address references are confined within two logical program segments.

The KL10 further divides executive and user mode operation into two submodes each. User mode is subdivided 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 which specifies whether or not a page can be accessed or altered, and if it is defined to be public or concealed. The executive and user modes subdivide on the KL10 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 designated public are, by definition, accessible. Pages designated concealed may be accessed only at defined entry points, i.e., portals which 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 designated to be public, the state of the processor transfers over into public submode. Ordinary user programs operate in public submode. Concealed areas can be used for proprietary coding that can be executed but not altered 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 alter areas designated 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 altered. 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 established. Functions delegated to supervisor submode generally include those affecting individual users as opposed to overall system management of input/output, priority interrupts, page map accounting, etc., which are handled by kernel submode programs. The ability of kernel submode programs to supply information which supervisor submode programs can read but not alter allows portions of the operating system to be hardware-protected from other portions undergoing modifications or design changes.

3.9 PROCESS TABLES

The system maintains two types of process tables. They are termed the executive process table and the user process table. These tables contain address mapping information (page maps) and various status and control information required for the process. The executive process table is built and assigned to the hardware processor by the TOPS-10 monitor when the system is initialized. In multiprocessor systems, each hardware processor will be assigned its own executive process table. The user process tables are built and assigned to an individual user by the monitor when a user process is started (a user logs in on the system). The structure of the process tables is detailed in Figures 3-4 and 3-5.

3.10 MEMORY PROTECTION AND RELOCATION

The KL10 provides memory address mapping from the program's memory address space (referred to as the effective or virtual address) to the physical memory address space by substitution of the most significant bits (page address) of the memory address. This mapping provides access to the entire physical memory space which is 16 times larger than the maximum user address space. The user's effective address space is 256K words addressed with 18-bit addresses; the physical address space is 4096K words addressed with 22-bit addresses (where 4096K is equivalent to 4,194,304 decimal; K equals 1024).

The memory mapping process utilizes the most significant nine bits of the effective address as an index into the appropriate page map (user or executive) in memory. The data located by the index provides 13 bits which are appended to the least significant 9 bits of the effective address in order to form the 22-bit physical address. Also provided are four bits which indicate what type of memory requests are allowed to the page in question (accessible, private, writable, etc.).

If this scheme were implemented exactly as outlined above, every user memory reference would require two actual memory references; one to obtain the memory mapping data and one to obtain the user's mapped memory reference. In order to reduce the number of actual memory references to nearly the same number as required by the program, a memory mapping unit (the pager) is used in the KL10 hardware.

0	KI PAGING=1: PAGE TABLE ENTRIES FOR USER PAGES 000-777 KI PAGING=0: AVAILABLE TO SOFTWARE
377	
400	
417	
420	
421	USER ARITHMETIC TRAP OVERFLOW INSTRUCTION
422	USER PUSHDOWN OVERFLOW TRAP INSTRUCTION
423	USER TRAP-3 INSTRUCTION
424	MUO STORED HERE
425	PC WORD OF MUO STORED HERE
426	PROCESS CONTEXT WORD
427	UNUSED
430	KERNEL NO TRAP NEW MUO PC WORD
431	KERNEL TRAP NEW MUO PC WORD
432	SUPERVISOR NO TRAP NEW MUO PC WORD
433	SUPERVISOR TRAP NEW MUO PC WORD
434	CONCEALED NO TRAP NEW MUO PC WORD
435	CONCEALED TRAP NEW MUO PC WORD
436	PUBLIC NO TRAP NEW MUO PC WORD
437	PUBLIC TRAP NEW MUO PC WORD
440	KI PAGING=1: AVAILABLE TO SOFTWARE KI PAGING=0: SECTION TABLE ENTRIES FOR USER SECTIONS 00-37
477	
500	
501	PAGE FAIL OLD PC WORD
502	PAGE FAIL NEW PC WORD
503	UNUSED
504	EBOX CLOCK TICK METER HIGH-ORDER WORD
505	EBOX CLOCK TICK METER LOW-ORDER WORD
506	MBOX CYCLE METER HIGH-ORDER WORD
507	MBOX CYCLE METER LOW-ORDER WORD
510	RESERVED FOR FUTURE USE BY HARDWARE
577	
600	
777	

NOTE
 User LUUO's trap to User Virtual addresses 40 and 41.

10-1917

Figure 3-4 User Process Table

0	EIGHT 4-WORD DATA CHANNEL LOGOUT LOCATIONS
37	
40	RESERVED FOR FUTURE USE BY HARDWARE
41	
42	STANDARD PRIORITY INTERRUPT INSTRUCTIONS
57	
60	4 CHANNEL BLOCK FILL WORDS
63	
64	RESERVED FOR FUTURE USE BY HARDWARE
137	
140	FOUR 8-WORD DTE20 INTERRUPT AND BYTE POINTER LOCATIONS
177	
200	KI PAGING=1: PAGE TABLE ENTRIES FOR EXEC PAGES 400-777 KI PAGING=0: AVAILABLE TO SOFTWARE
377	
400	AVAILABLE TO SOFTWARE
417	
420	UNUSED
421	EXEC ARITHMETIC OVERFLOW TRAP INSTRUCTION
422	EXEC PUSHDOWN OVERFLOW TRAP INSTRUCTION
423	EXEC TRAP-3 TRAP INSTRUCTION
424	RESERVED FOR FUTURE USE BY HARDWARE
437	
440	KI PAGING = 1 : AVAILABLE TO SOFTWARE KI PAGING = 0 : SECTION TABLE FOR EXEC SECTIONS 00 - 37
477	
500	RESERVED FOR FUTURE USE BY HARDWARE
507	
510	TIME BASE HIGH-ORDER WORD
511	TIME BASE LOW-ORDER WORD
512	PERFORMANCE ANALYSIS COUNTER HIGH-ORDER WORD
513	PERFORMANCE ANALYSIS COUNTER LOW-ORDER WORD
514	INTERVAL TIMER VECTOR INTERRUPT LOCATION
515	RESERVED FOR FUTURE USE BY HARDWARE
577	
600	KI PAGING = 1 PAGE TABLE ENTRIES FOR EXEC PAGES 000-337 KI PAGING = 0 : AVAILABLE TO SOFTWARE
757	
760	AVAILABLE TO SOFTWARE FOR USE BY CHANNELS (DF10, DAS33) WITH PROGRAMMABLE LOGOUT AREA
777	

NOTE

Executive LUUO's trap to Executive Virtual addresses 40 and 41.

10-1918

Figure 3-5 Executive Process Table

If the address is in the range 0–17₈ inclusive, the hardware fast register blocks are referenced instead of the memory system. Otherwise, the user mode bit and the high-order nine bits of the virtual address are compared against the contents of the pager which is part of the memory mapping hardware. These ten bits will either match or not match with one of the entries in the pager. The user mode bit and the high-order nine bits of the virtual address are used to do a “table look-up” in the hardware page table.

If a page table entry exists, part of the entry may be used to form the 13-bit most significant portion of the physical memory address; the rest of the entry, the five descriptor bits, are used to check if the reference to the page is legal.

If the memory request is consistent with the page descriptor bits, the page number part of the entry is used as the 13 most significant bits of the physical address; and the nine least significant bits of the virtual address are used as the least significant nine bits of the physical address.

When the relocation data for a referenced page does not exist in the hardware page table (i.e., a no-match), the hardware reads the relocation data from the process table in core memory and stores it into the memory of the hardware page table.

The monitor assigns the core area for each user by loading the appropriate process tables, setting up the trap locations in the process table, and responding appropriately when a trap occurs. The monitor provides memory protection for itself and each user by filling the process tables only with those entries which are allowed to be accessed. A zero access bit in the entry will cause a reference to the associated page to initiate a page failure trap to the monitor.

The TOPS-10 Operating System utilizes the KL10 page maps in the process tables to create 1- or 2-segment programs in roughly the same fashion as it uses the protection and relocation registers of the KA10. The major benefits of the paging capability are a smaller unit of core allocation (512 words instead of 1024), the freedom to scatter the pages of a segment randomly through physical core (avoiding core fragmentation by eliminating unusable holes in core and the overhead of repacking core), and the opportunity to execute a program when all of its pages are not in physical core (i.e., a virtual memory capability).

3.11 DIRECT I/O

The DECsystem-10 EBus provides a 36-bit, full-word parallel path between memory (via the EBox) and an I/O device for purposes of control or low-speed data transfers. To initiate high-speed data channel transfers directly between memory and a device connected to the memory bus, a control word is first transferred over the EBus to the buffer of the high-speed device controller (RH10, RH20, or DX10). Then, on command, entire data blocks are moved directly to or from memory with a single instruction.

The EBus may also be used as a control and data path to/from a large number of low-speed I/O devices. Transfer is performed in 36-bit words in parallel at speeds of 370K words/second. Thus each data transmission instruction moves one word of data between memory and the buffer of the device controller. When block input or output instructions are used, entire blocks of data are moved to or from the device with a single instruction.

3.12 CHANNEL I/O

Channel I/O transfers are executed by the DF10 and DX10 external data channel processors or the integral data channel processor (channel control) of the MBox storage controller.

3.12.1 External Data Channels

The external data channel processors transmit 36-bit data words to and from the memory system up to the rated speed of the memories themselves. They accomplish these transfers in parallel with central processor memory references. The channel I/O processor is activated when it receives a starting address from a device controller. (In the case of the disk system, this is the RH10 Massbus controller.) It then proceeds to process a channel command list sequentially from the starting address. Four instructions are available for creating this list. They are:

1. **DATA TRANSFER** – A word count in the left half of the word (18 bits) and a memory address in the right half. This transfers the number of words specified in the word count starting at the specified memory address.
2. **JUMP** – Zero in the left half and an address in the right half. This causes the next instruction word to be fetched from the specified address rather than the next sequential location.
3. **SKIP/FILL** – A word count in the left half, zero in the right half. The data channel reads the specified number of words from the I/O device but does not put them in memory, or writes the specified number of words starting with memory location 1 to the I/O device.
4. **HALT** – A word containing all zeros halts the data channel, signals the controller, and puts it in a state ready for initialization.

High-speed external I/O can also be multiplexed in several ways on the DECsystem-10.

1. The data channel such as the DF10 and DX10 may be accessed by up to eight devices. Once a chain of transfers has been initialized, however, other devices must wait until completion before they may initialize the channel.
2. Additional data channels may be interfaced to their own memory bus and access any or all memories on this bus.
3. A multiplexer is available to allow multiple data channels including the DL10 communications channel to share one memory bus. This allows a true multiplex channel operation requiring only one bus.

3.12.2 Integrated Data Channels

The integrated data channels serve as high-speed data paths between main memory and secondary memory (disks and tapes) in a DECsystem-1090 as do the external data channels in a DECsystem-1080. In a DECsystem-1090 up to eight integrated data channels can be implemented along with a number of external channels. Each data channel is formed by main memory, a channel control in the MBox storage controller, and a Massbus controller and its drives. The channels are multiplexed to provide each implemented channel access to main memory when needed. Data is transferred to/from memory in 4-word blocks to extend the effective memory bandwidth and to improve channel throughput. The channels accomplish these transfers to/from memory by “stealing” memory cycles from the cache in the storage controller. To normalize the transfer speed between the memory and the devices, the channel control employs a 15-word buffer for each channel which will accumulate/supply the data when needed.

A given channel is activated when a controller and a drive receive a read or write command from the central processor. A synchronous data path is then automatically established between the drive and memory to execute the command under the control of a channel command list which is stored in main memory.

3.12.2.1 Massbus Controller – The Massbus controllers are high-speed, universal mass storage controllers which interface the RP04/06 disk drives and the TU16 magnetic tape drives to the integrated data channels in the MBox storage controller. The Massbus controllers have been designed to provide high throughput by having the following features implemented.

1. All controllers can transfer data simultaneously since each controller is connected to the memory system with its own built-in channel control.
2. While one device on a controller is transferring data, control operations such as seek or rewind may be issued by the CPU to another device on the same controller. After the operation is completed, the CPU is notified through the interrupt system.
3. Each controller has a lookahead command register, which enables the software to preload the next transfer request during the current transfer. Thus the next transfer can begin with the next sector on the same device with no rotational delay.
4. When the controller has finished 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 a synchronous data path and an asynchronous control path. These parallel paths permit simultaneous data transfer and control operations.

3.12.2.2 Channel Controllers – Within the MBox storage controller there is effectively one channel control for each Massbus device controller. Each channel control has a 15-word data buffer, a command list pointer register (essentially a program counter), and a channel command word register. The channel control transfers the data by executing a program (channel command list) which is loaded into memory by the device service routine. The first instruction of the channel program is stored in a fixed location of the executive process table (EPT). There are eight locations reserved in the EPT for the initial instructions of the channel programs (initial command word), one for each channel. After being started (by the Massbus controller), the channel control fetches the initial channel command word from the EPT, moves it into the channel command word register, and executes the specified function. The command list pointer register keeps track of the memory address of the channel command word. Thereafter, the remaining instructions are fetched and executed until a halt or a last data transfer instruction is encountered. The direction of the transfer is specified in the CPU instruction (read/write command) that is sent to the controller and drive to initiate the transfer.

Channel instructions contain an operation code field, a memory address field, and a word count field. Six basic instructions (operation codes) are implemented. They are:

1. HALT
2. JUMP
3. FORWARD DATA TRANSFER
4. REVERSE DATA TRANSFER
5. FORWARD LAST DATA TRANSFER
6. REVERSE LAST DATA TRANSFER.

Each of the four data transfer instructions can be further encoded to execute a Skip/Block Fill function. This function is specified if the value in the memory address field of the instruction is zero.

The jump instruction causes another instruction to be fetched from the location specified by the contents of the address field. The content of the word count field is not used. This instruction is normally placed in the EPT to serve as the initial channel command word. The content of the address field is automatically stored in the command list pointer register.

The four data transfer instructions are used to transfer the number of words specified in the word count field to/from memory, starting with the location specified in the address field. Each time a word is transferred, the word count is decremented and the address is incremented or decremented depending on whether the transfer is a forward or reverse transfer, respectively. When the word count reaches zero for a non-last data transfer instruction, the channel control fetches the next instruction from the location pointed to by the contents of the command list pointer register. If the instruction was a last data transfer instruction, the channel control terminates the transfer and may log out by moving two status words to the EPT. The channel control logs out only if the read/write command that was sent to the controller and drive specified a store operation.

System I/O programs can choose 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 is a time-saving feature since it saves a halt command memory fetch.

If a channel detects an error, it stores two channel status words in the appropriate channel locations in the executive process table. The first status word contains bits to indicate what kind of error occurred and the control word location pointer (CLP). The second status word contains the command word that was being executed with the up-to-date word count and address.

3.13 PRIORITY INTERRUPT SYSTEM

The DECsystem-10 priority interrupt system is one of the most flexible available today. Devices are assigned under program control to any one of seven priority levels through the dynamic loading of a 3-bit register within the device. Each interrupt level has any number of high-speed programmable sub-levels. Thus, a program can change the priority level of any device or disconnect the device from the system and later reinstate it at any other level. In the same manner, a program can set, enable, or disable any combination or all levels with a single instruction. In addition, the program can assign some or all devices to the same level.

A set of instructions (Block-In and Block-Out) allow blocks of information, using the interrupt system, to be transferred between a device and memory. These instructions identify the source of the interrupt, update a word count and data address which are stored in the EPT, transmit or receive a word of information, and dismiss the interrupt. This operation is repeated until the entire block is transferred.

The system can also generate interrupts through software. Real-time hardware can thus operate on a high-priority level while related computations, particularly if they are lengthy, can be performed on a lower level.

The DECsystem-10 program-assignable priority interrupt system provides much greater flexibility than permanently hard-wired systems. Hard-wired systems require a large number of levels, often operate at extremely high overhead, and cannot change device priorities without system shutdown and rewiring.

An interrupt can cause the processor and the interrupting device to immediately initiate one of several possible actions. In response to the “interrupt grant” signal from the processor, the device may supply a 33-bit word (API function word) which is decoded as 18 bits address, 12 bits data, and 3 bits function. The processor then does one of the following:

1. Executes the instruction found at the supplied 18-bit address
2. Transfers a word into or out of the addressed location
3. Adds or subtracts 1 to the addressed location.

3.14 TRAP FACILITY

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

1. Address Violation
2. Arithmetic Overflow
3. Pushdown Overflow
4. Illegal Instruction
5. Monitor Calls
6. Page Faults.

3.15 PROGRAMMABLE CLOCKS

The four clocks built into the KL10 provide a number of timing and counting functions including an interval timer, a time base, an accounting meter, and a performance analysis counter.

All of the operations on these clocks are accomplished by means of I/O instructions to internal devices. Many of these functions use a microsecond source of pulses which is counted down from the basic machine clock of 50 MHz. The machine clock has a tolerance of 0.005 percent which will provide less than 5 seconds drift over 25 hours.

The interval timer provides a programmable source of interrupts with a 1 μ s resolution and is similar to the DK10 real-time clock. It is used for real-time applications and for page management by the monitor. It is designed so that a real-time deadline schedule with varying deadlines can be implemented.

The time base provides a 60-bit μ s resolution source of elapsed time. This gives over a 9140-year maximum time before wraparound.

The accounting meters provide an accurate and reproducible measure of the amount of processor resources used by a job, interrupts, page failures, and cache activity.

The performance analysis counter provides a tool for studying hardware and software performance of the system. It may be used to point to hardware and/or software bottlenecks.

3.16 CONSOLE/DIAGNOSTIC COMPUTER

One of the most significant features of the KL10 is the PDP-11-based console/diagnostic computer.

The PDP-11 communicates with the KL10 through a Ten-Eleven Interface. The interface allows data transfers between the PDP-11 and the KL10 Central Processor to take place simultaneously in both directions. On the PDP-11 side of the interface, data has direct memory access over the PDP-11 Unibus in 8- or 16-bit bytes. On the KL10 Central Processor side of the interface, data may exist in any size byte up to 36 bits in length.

The PDP-11 serves as the console device for the KL10 Central Processor. An RP04 disk pack channel and a TU56 DECTape channel are implemented on the PDP-11 for reading the microcode into the KL10s microcode memory. An LA36 DECwriter-II is included for operator/monitor communication. The DECTape serves as the backup device.

As a diagnostic computer, the PDP-11 can examine the data paths and the control logic of the KL10 via a separate "diagnostic bus" (part of the EBus), even if the KL10 is completely inoperative. Other features of the diagnostic computer allow all data buses to be checked. Remote diagnostic checking can also be performed via the PDP-11 computer.

The console/diagnostic computer is a state-of-the-art concept to provide the DECsystem-1080 and DECsystem-1090 with greatly reduced mean-time-to-repair (MTTR).

3.17 SYSTEM INTEGRITY FEATURES

If system power fails, a power failure detection circuit senses the condition and causes an interrupt. The interrupt can trigger the operation of a program which saves all valuable registers so that the system can be restarted in a minimum amount of time.

On the KL10 (through the PDP-11 console computer) an automatic restart capability has been added to resume normal operations in the event of a power outage. All three phases of ac power are monitored. Low voltage on any phase will initiate a sequence of power-down operations. A program-selectable, automatic restart capability is provided to allow resumption of operations 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.

SECTION 4 THE HARDWARE

4.1 CONFIGURATIONS

The KL10-based system is offered in two basic DECsystem-10 configurations. They are designated DECsystem-1080 and DECsystem-1090 (Figures 4-1 and 4-2). Multiprocessor configurations are also offered (i.e., DECsystem-1088 and 1099). The major difference between the basic configurations is that the 1080 employs only external data and communication channels, whereas the 1090 uses both internal and external channels.

The DECsystem-1080 configuration may include the following subsystems:

1. Central Processor Subsystem Without Internal Channel Controllers
2. Console Processor Subsystem
3. Main Memory Subsystem (external)
4. RHP04/06 Disk Subsystem (external DF10 channel control)
5. RHS04 Disk Swapping Subsystem (external DF10 channel control)
6. THU16 Magnetic Tape Subsystem (external DF10 channel control)
7. TU70-72 Magnetic Tape Subsystem (external DX10 channel control)
8. Unit Record Subsystem
9. DC75/DC76/DN87 Communications Subsystem (external DL10 channel control).

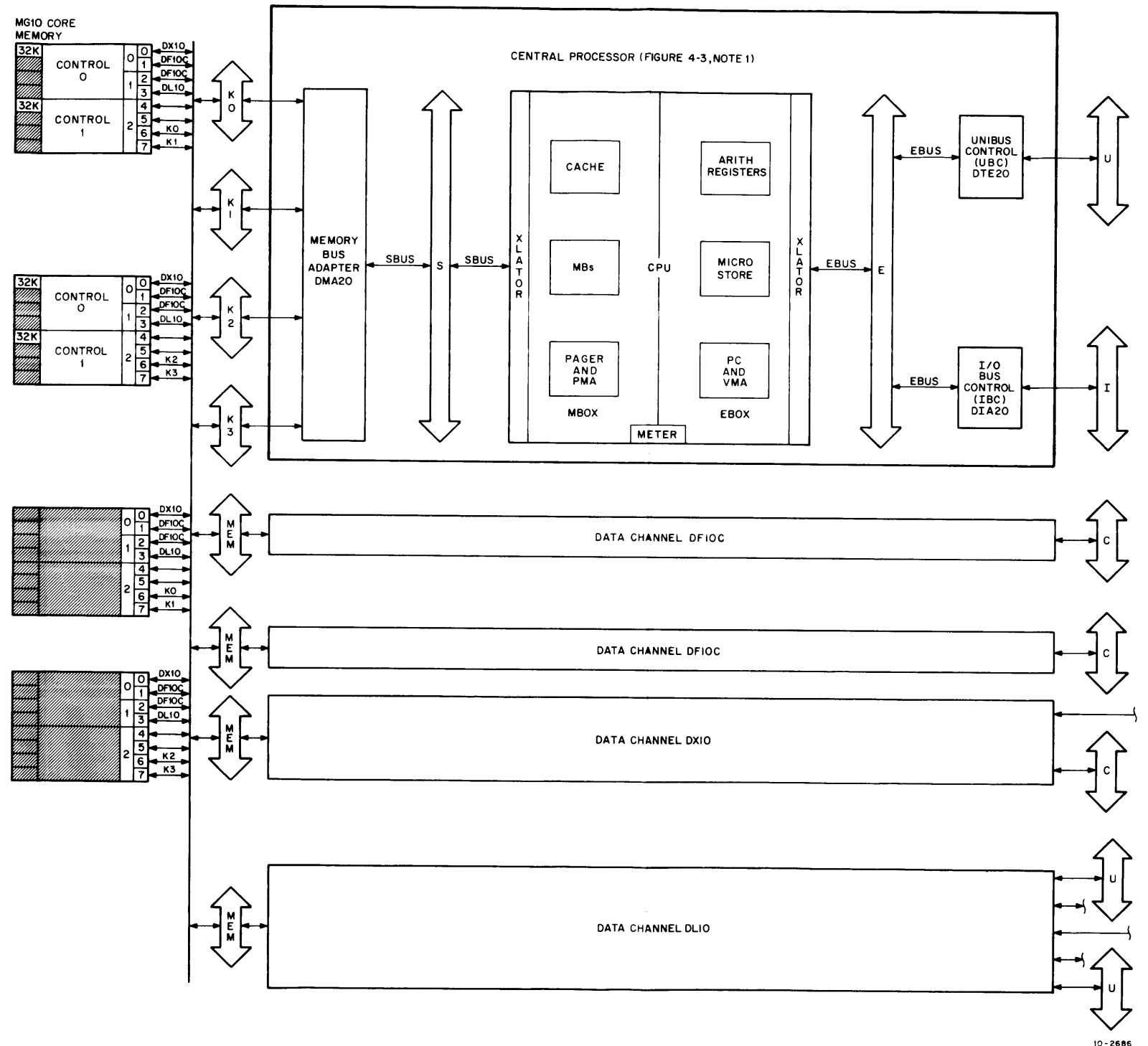
The DECsystem-1090 configuration may include the following subsystems:

1. Central Processor Subsystem with Internal Channel controllers
2. Console Processor Subsystem
3. Main Memory Subsystem (external)
4. RTP04/06 Disk Subsystem (internal channel control)
5. TTU16 Magnetic Tape Subsystem (internal channel control)
6. TU70-72 Magnetic Tape Subsystem (external DX10 channel control)
7. Unit Record Subsystem
8. TU56 DECtape Subsystem
9. DN87S Communications Subsystem (internal DTE20 channel control)
10. DN87 Communications Subsystem (external DL10 channel control).

4.2 CENTRAL PROCESSOR SUBSYSTEM

The central processor subsystem of DECsystem-1080/1090 may include the following units:

1. EBox
2. MBox (with or without internal channel controllers)
3. Meter Board
4. DMA20 Memory Bus Adapter
5. DTE20 Ten-Eleven Data Interfaces (a maximum of 4)
6. RH20 Massbus Controllers (a maximum of 8)
7. DIA20 IBus Adapter Control.



10-2686

Figure 4-1 DECsystem-1080 (Typical)
Block Diagram (Sheet 1 of 2)

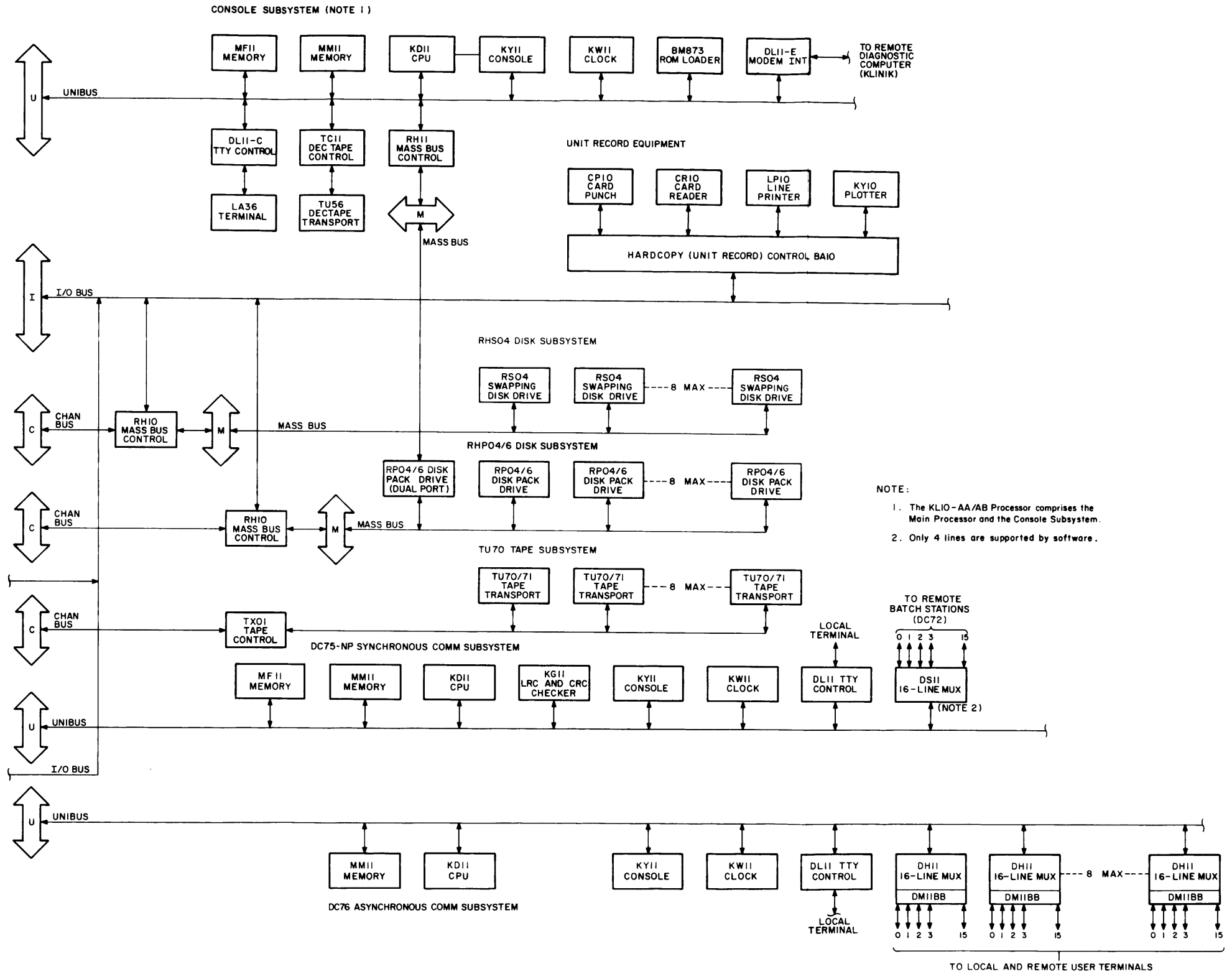


Figure 4-1 DECsystem-1080 (Typical)
Block Diagram (Sheet 2 of 2)

NOTES

- At least two MH10 memories are required to operate with four way interleaving in effect. Each MH10 memory can be equipped with 256K (8 × 32K) of core. The MG10 memory which may also be implemented can be equipped with 128K (8 × 16K) of core.
- The KL10-BA/BB Processor consists of the Central Processing Unit (CPU) and the Console Processor Subsystem.
- Up to 3 DX10 Data Channels can be implemented.
- Four basic DN87/DN87S maximum configurations are available as follows:

Config	ASYN		SYNC		Cabinets	
	Lines	DH11s	Lines	DQ11s	Basic	Expansion
1	128	8	0	0	1	2
2	64	4	4	4	1	1
3	32	2	8	8	1	2
4	0	0	12	12	1	1

- Up to 2 DL10 Communications Channels can be implemented. Each DL10 Channel can be equipped with 4 DN87 Communication Subsystems.

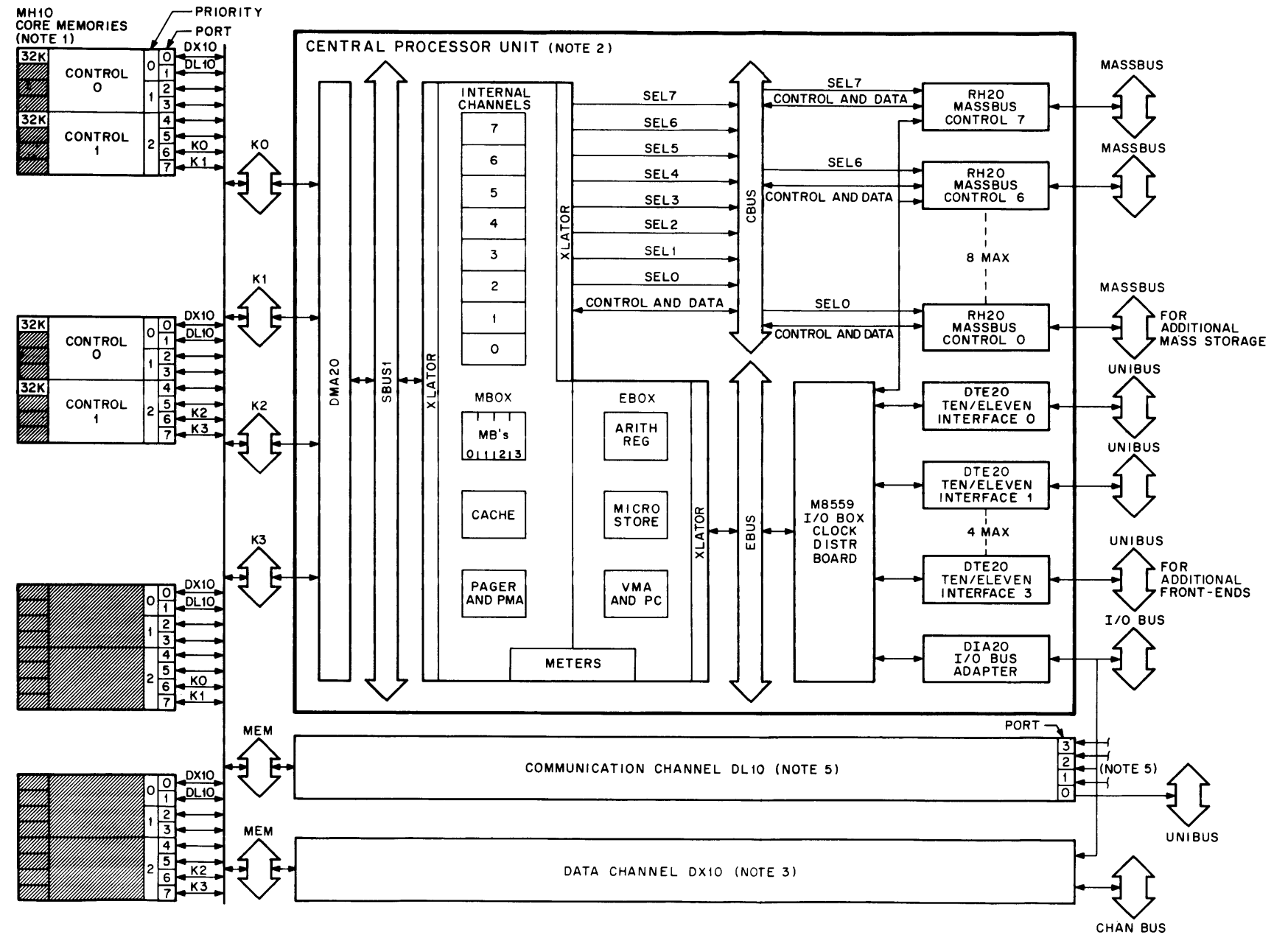


Figure 4-2 DECsystem-1090 (Typical) Block Diagram (Sheet 1 of 2)

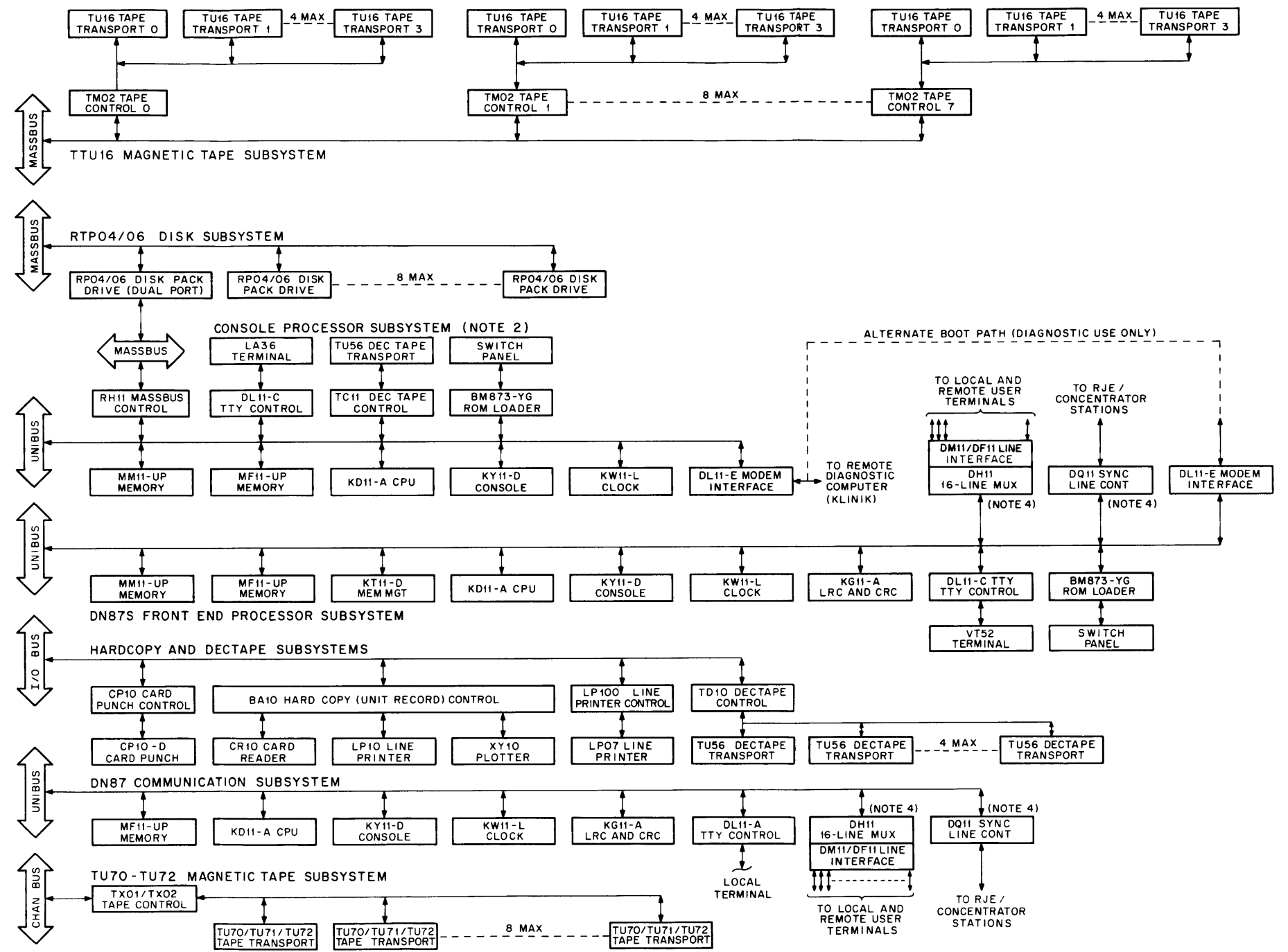


Figure 4-2 DECsystem-1090 (Typical) Block Diagram (Sheet 2 of 2)

The EBox, MBox and Meter Board are designed with high-speed, non-saturating emitter-coupled logic (ECL) and are housed on the same assembly. These functional units comprise the central processing unit (CPU). The DMA20, DTE20, RH20, and DIA20 are designed with high-speed TTL logic and serve as the interface controllers to core memory, the console processor and communication front-ends, the Massbus-compatible devices and I/O bus compatible devices, respectively (Figures 4-3 and 4-4). An internal storage bus (SBus) serves as the control and data path between the CPU and the DMA20 external memory bus adapter; an internal execution bus (EBus) serves as the control and data path between the CPU and the I/O controllers (DTE20, RH20, and DIA20). Since both ECL and TTL logic are employed in the central processor, logic level translators are incorporated into the CPU assembly to convert from one logic level to the other.

4.2.1 EBox

4.2.1.1 Hardware – The EBox is the instruction execution unit of the central processor. Essentially, the instruction execution unit consists of the following logic:

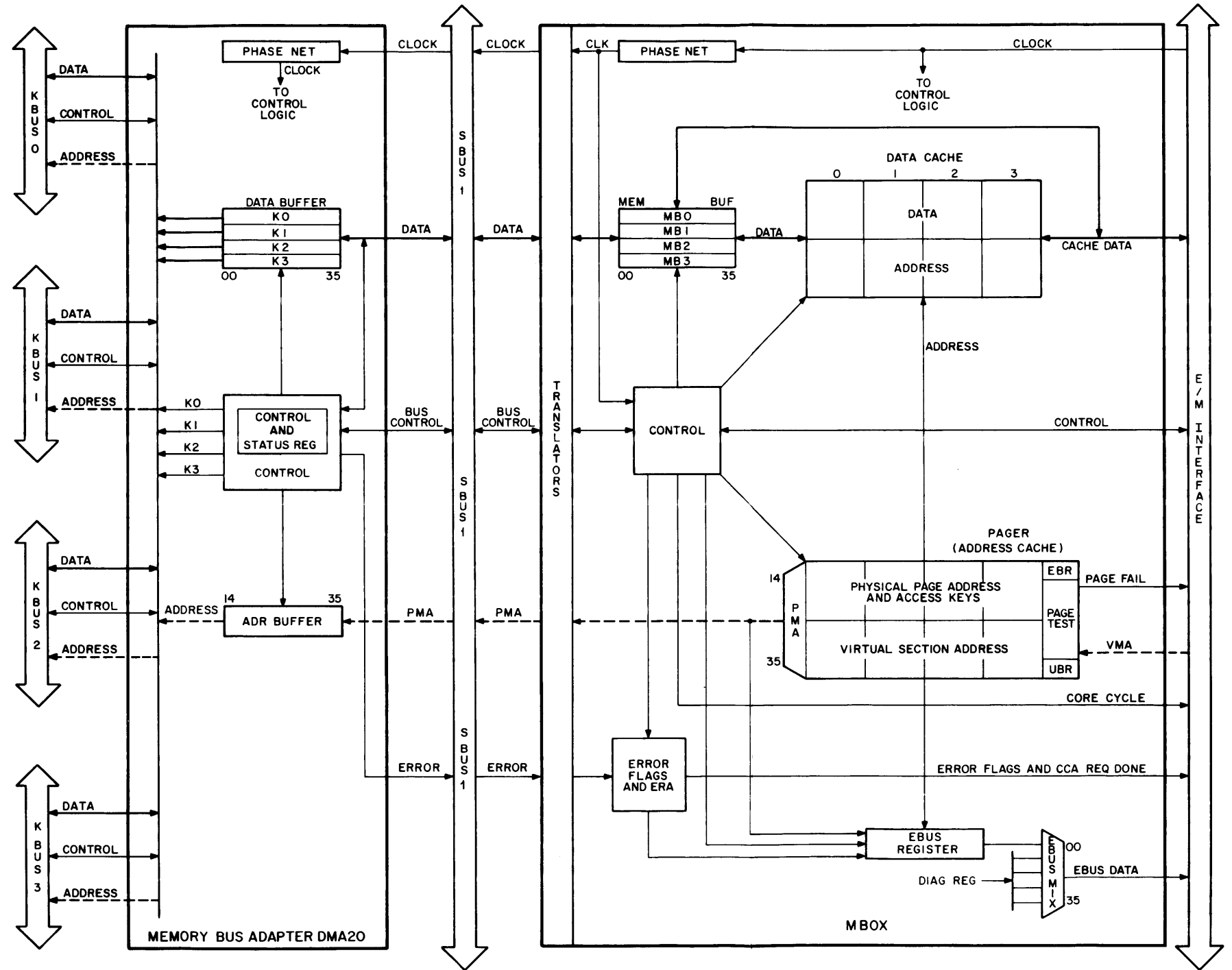
1. Memory Request Logic
2. 72-bit Arithmetic Logic
3. 23-bit Address Logic
4. 10-bit Arithmetic Logic
5. Eight General Register Blocks
6. EBus Control Logic
7. Microprogrammed Instruction Dispatch and Control Store.

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

All operations in the KL10-based DECsystem-10 are synchronized to the master clock which runs at 50 MHz in the model A CPU and 58 MHz in the model B CPU. The master clock can be started, stopped, single-stepped, and otherwise controlled by the console processor (privileged front-end) via the diagnostic control logic. This logic is distributed between the EBox and the DTE20. Besides being able to control the master clock, the diagnostic control logic provides a means for monitoring processor status and diagnostic registers in both the EBox and the MBox. The master clock of a model A CPU is divided down to supply a 25 MHz clock to the MBox, a slower variable clock for the EBox control store, and an 6.25 MHz clock to the EBus and SBus.

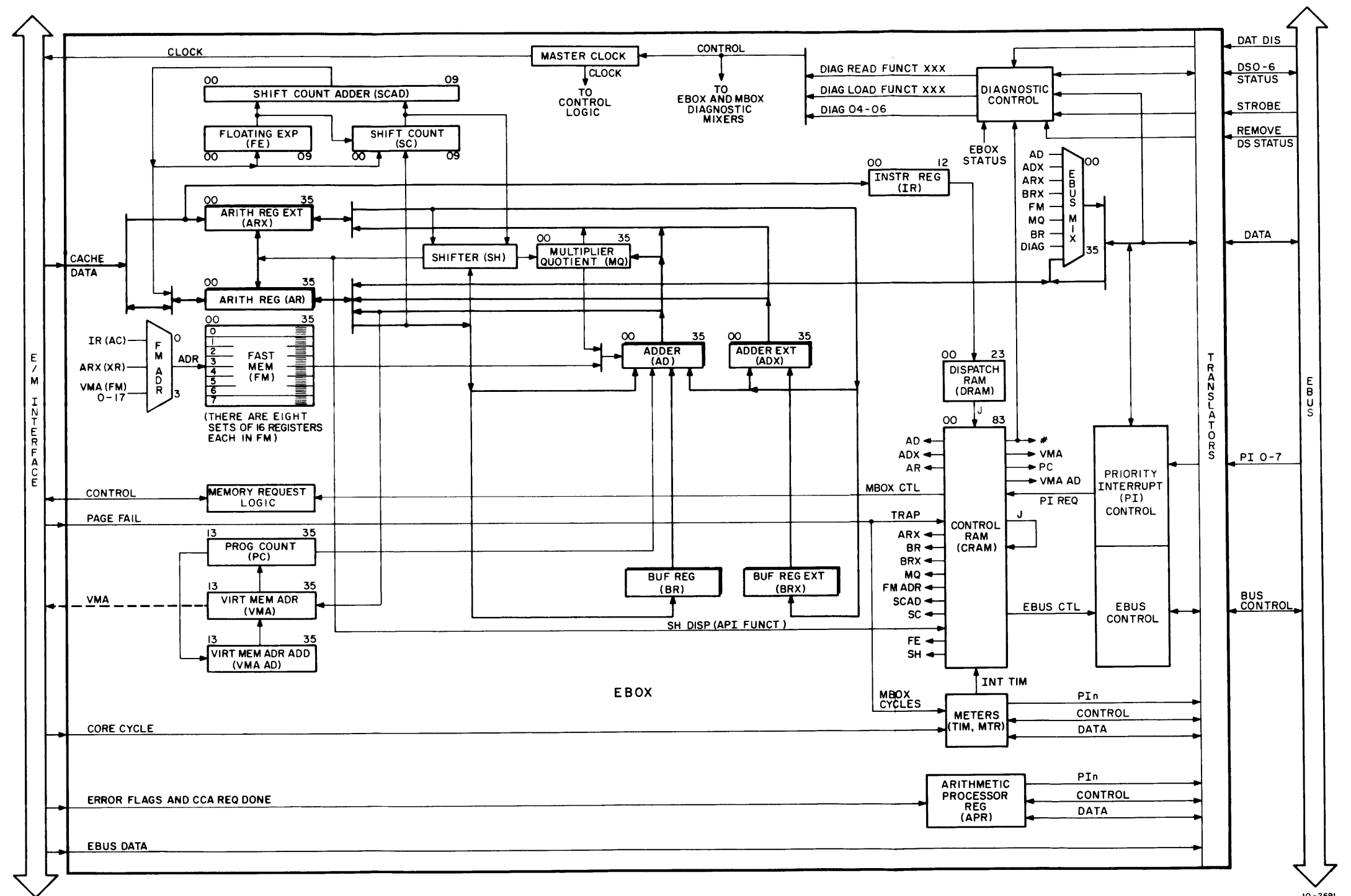
The program counter (PC), virtual memory address adder (VMA AD), virtual memory address register (VMA), and the arithmetic adder (AD) form the basic address manipulation path in the EBox. This path is 23 bits wide to accommodate a virtual address space of 8 million words. Only 18 bits of the virtual address are implemented for DECsystem-1080 and 1090 to provide a virtual address space of 256,000 words. During the course of calculating the effective address for an instruction, AD will contain the value of memory address 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 in such a way that half words, full words and double words can be manipulated easily. The AD and ADX are implemented with arithmetic logic units (ALU) which are capable of performing 16 arithmetic and 16 logic operations. Words fetched from memory may be moved into the AR, ARX, and IR. Typically, data (operands) is placed in the AR 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.



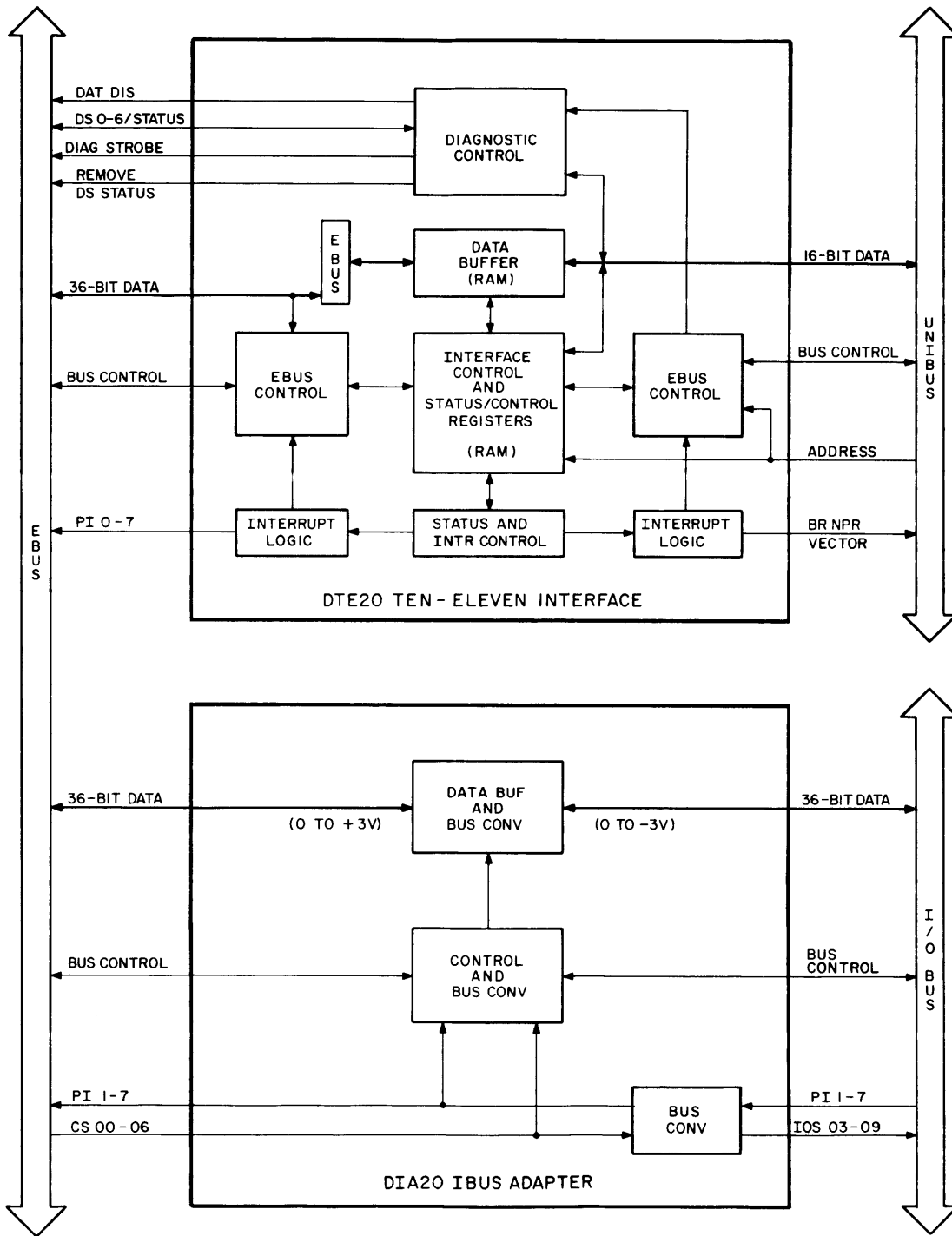
10-2690

Figure 4-3 DECsystem-1080 Central Processor Subsystem Block Diagram (Sheet 1 of 3)



10-2691

Figure 4-3 DECsystem-1080 Central Processor Subsystem Block Diagram (Sheet 2 of 3)



10-2692

Figure 4-3 DECsystem-1080 Central Processor Subsystem Block Diagram (Sheet 3 of 3)

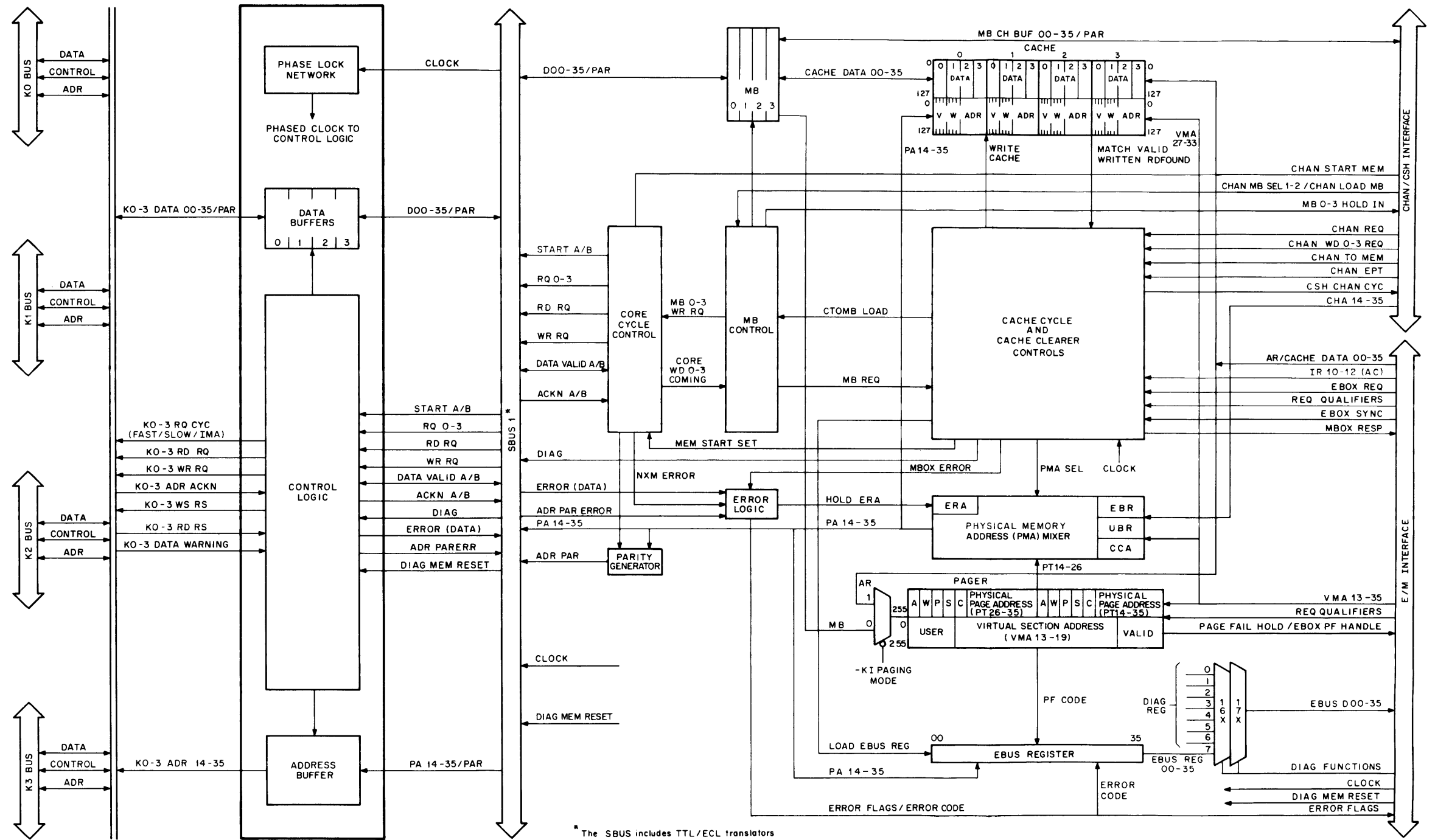


Figure 4-4 DECsystem-1090 Central Processor Subsystem Block Diagram (Sheet 1 of 3)

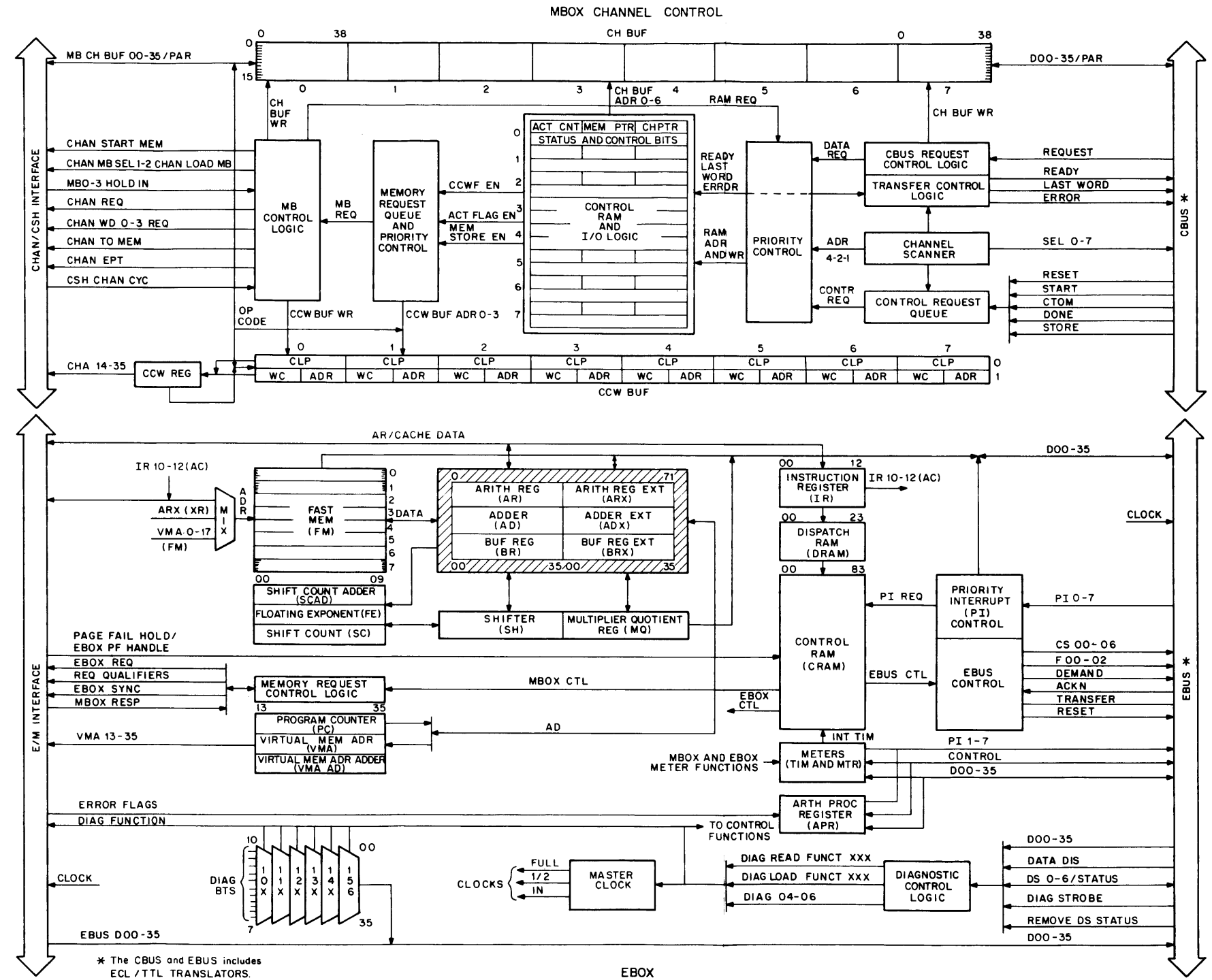
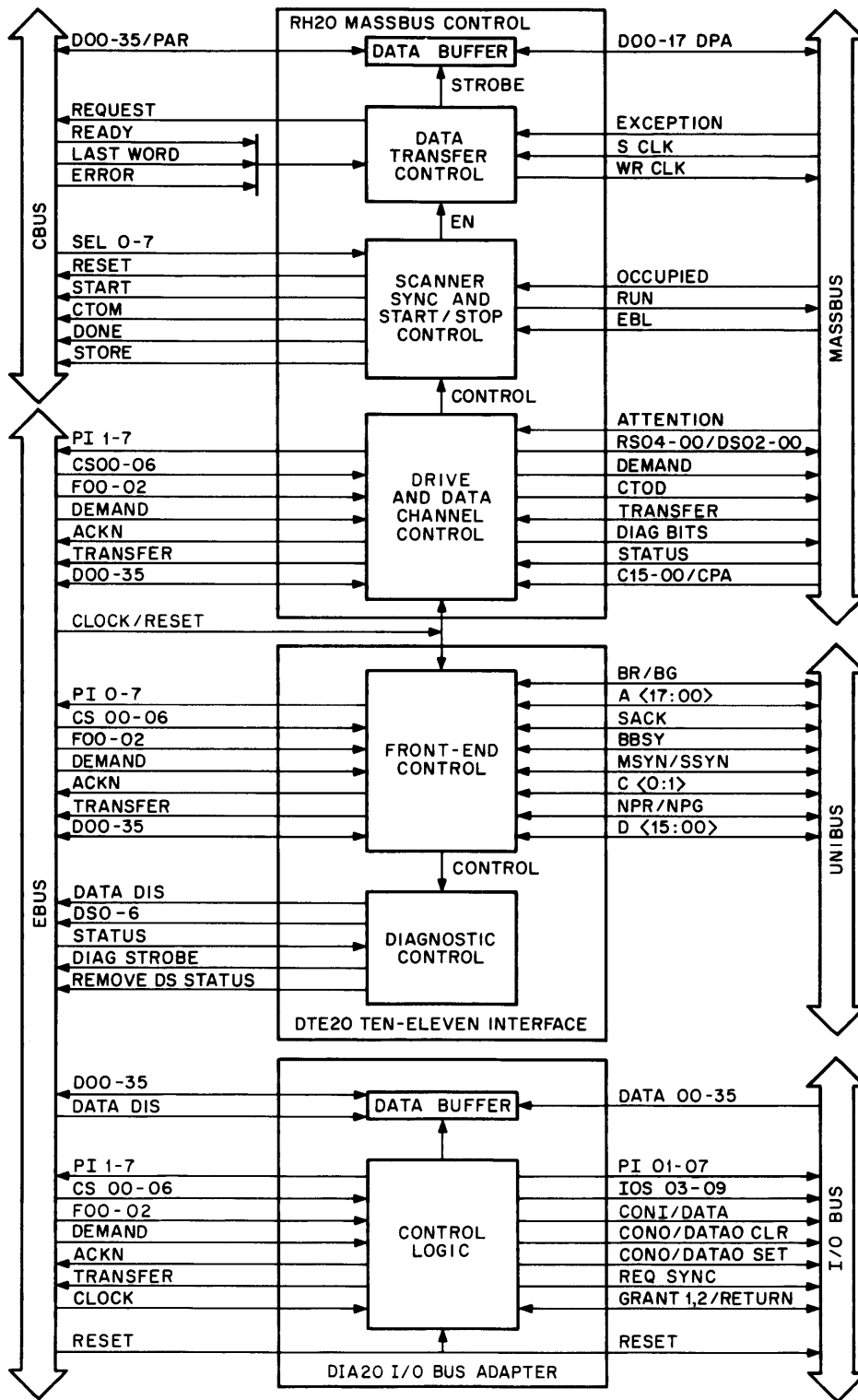


Figure 4-4 DECsystem-1090 Central Processor Subsystem Block Diagram (Sheet 2 of 3)



10-2694

Figure 4-4 DECsystem-1090 Central Processor Subsystem Block Diagram (Sheet 3 of 3)

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) consist of a set of 16 general-purpose, high-speed registers. Blocks 0 and 7 are permanently assigned to the monitor and the microcode, respectively. From the remaining blocks two can be assigned under program control (DATAO PAG) to the user as the current and previous context AC blocks.

The monitor uses its assigned AC block in the same way as the user program uses its current context AC block.

The microcode uses the assigned AC block when executing complex instruction algorithms.

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 (locations 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 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 for ultimately initiating 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 (Subsection 4.2.1.2).

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; if no traps, interrupts, or errors are sensed, the microcode will begin 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 14–17 (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. 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 KL10 CPU is capable of multilevel indirect addressing and 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 which points to the location in the microprogram where the execution portion of the cycle for the instruction begins. 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, etc.

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 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 which will begin the execution of the particular instruction.

Once the execution portion of the cycle is entered, the branching mechanism is controlled dynamically by conditions dictated by the particular 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 portion of the microprogram cycle implements the storage of those operands developed during the execute portion 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 begin 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 supplying 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, etc.

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 dialogue 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) specifies the type of I/O operation (CONO, CONI, DATAO, DATAI, etc.) to be executed. If the fetched I/O instruction specifies output operation to the device (CONO, DATAO, CONSO, BLKO), then the EBox issues a memory request to fetch the operand before executing the EBus dialogue. If, however, the instruction specifies an input operation from the device (CONI, DATAI, CONSI, BLKI), the EBus dialogue is executed to fetch the word first, and 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 (PI0–7) from the controller on the EBus. After the API function word is received, an interrupt 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 are, therefore, considered to 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 considered to be internal processor devices. Standard I/O instructions are used to access these devices for setting up control functions, for monitoring status, and for transferring data.

4.2.1.2 Firmware – The heart of the EBox is a high-speed, 1280-word control random access memory (CRAM). This memory is initialized to contain the microcode. The microcode is loaded into the EBox by the console processor from the RP04 disk subsystem or the TC11-G DECTape subsystem. These devices are used for booting, diagnosing, and dumping functions. The console processor-based TC11-G DECTape subsystem is not supported as a system device. The microcode includes the following functional process elements:

1. Startup 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 75 bits of actual control information in an 84-bit field (Figure 4-5). Initial entry into the microcode for a given instruction is a function of the instruction op code via the dispatch RAM (DRAM). Thereafter, the microcode sequences as a function of the J dispatch address 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, etc. Upon completion, all instructions must pass through this process.

The effective address manager evaluates indirect address flag bit 13, 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, i.e., enabling the EBox clock to stop, as appropriate, or dispatching directly to the executor, or initiating an instruction prefetch. Note the instruction register is used to address the proper location in the dispatch table (DRAM) based upon the op code for the instruction.

The executor routine is the bulk of the microprogram. It contains a number of somewhat autonomous routines which are used to execute the instruction specific function, e.g., move a half word from one register to another, 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, e.g., MEM/WRITE, the data storage manager defines the appropriate MBox control signals and dialogue and initiates the write operation. When the data storage manager is entered in the context of a store cycle, that control generally passes to that process from the executor. Then, finally, control will pass to the start-up and stop interface.

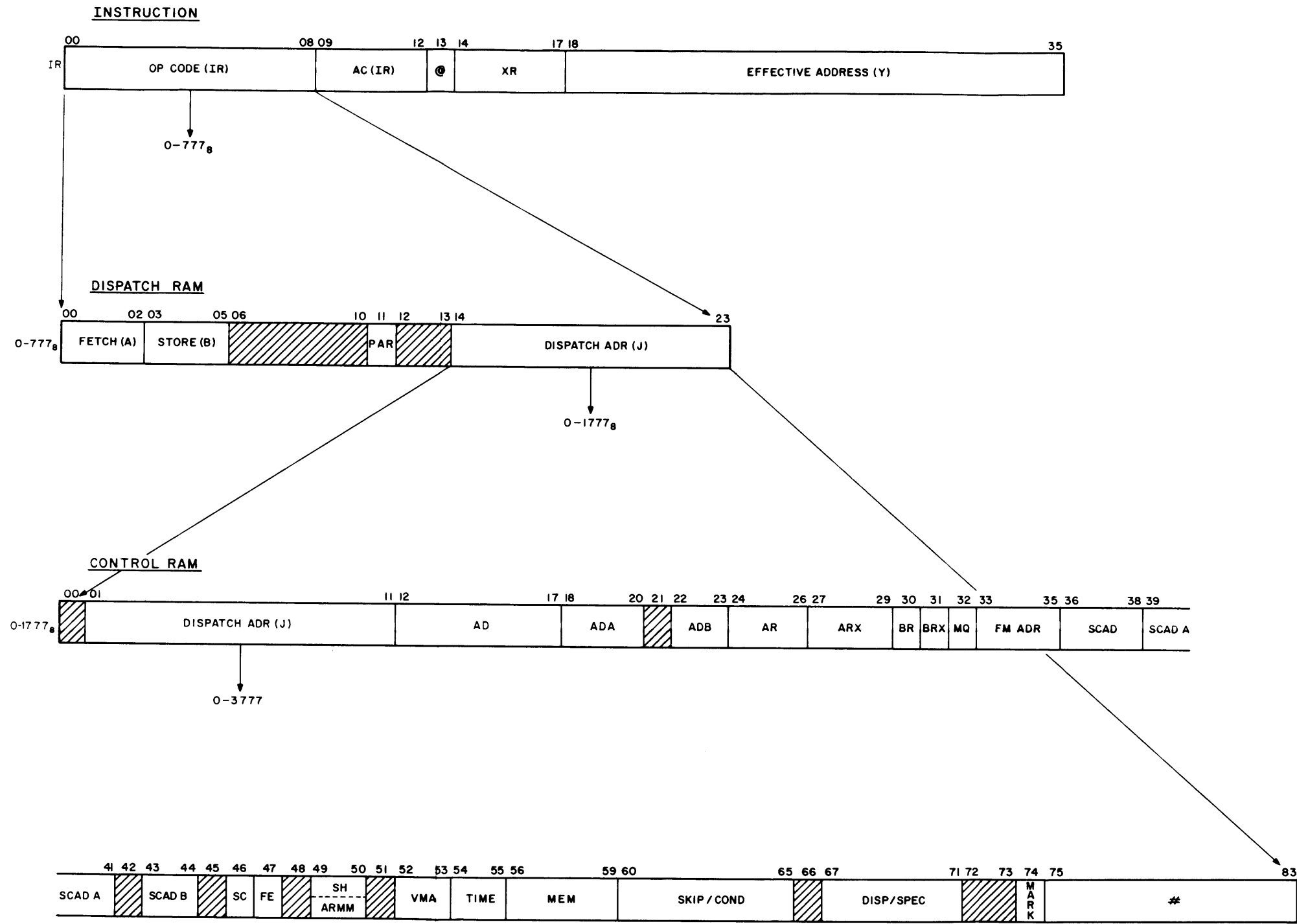


Figure 4-5 Instruction, Dispatch, and Control Formats

The priority interrupt handler is entered from discrete points in the microprogram. Interrupt is granted 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 prior to MBOX RESP during a memory request. The implication is that a memory address violation (page failure) occurred, i.e., an access failure, write protection violation, or some similar violation. 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 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 to via IR dispatch from the dispatch table on DATAO and CONO instructions after the data or status has already been fetched from memory, or directly on DATAI, CONI, CONSO, or CONSZ instructions. The handler calls the EBus driver which generates the necessary EBus dialogue with the device. For BLKI or BLKO instructions, the pointer word must be fetched from the EPT, updated, and stored back at E before the required word is fetched. This is performed by the input/output handler first. When the data has been fetched, the EBus driver is called. On DATAI and CONI instructions, 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.

4.2.2 MBox

The MBox is the storage controller of the KL10-based DECsystem-10. The MBox contains a pager, a physical memory address selector (PMA), a 4-segment data cache, and four memory buffer (MB) registers. These functional elements provide the EBox instruction execution unit access to physical memory. The physical memory address is formed by the pager and the PMA while the data path between main memory and the EBox is created via the MBs and the data cache.

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) 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 channel I/O processor interacts with the data cache to maintain the integrity of the flow of data between physical memory and mass storage.

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 table portions of 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 (LINE NO.). This address is then used to look in the cache and, if necessary, issue a core request. If the pager does not contain a valid physical page address, the MBox automatically issues a core read cycle to refill the hardware page table from the user or executive process table. Since four words are typically fetched at a time and since the process table contains two physical page address entries per word, eight page table entries will be fetched and moved to the pager at a time. Consequently, a page refill cycle will be required only when the program addresses pass through the boundary of every eighth page.

The cache is a high-speed, 2048-word, set-associative automatic data buffer memory where instructions and operands are stored and maintained as the EBox issues requests for memory. It serves as a high-speed extension of core memory. When the EBox issues a memory request, the MBox fetches a 4-word block (quadword) from core, transfers the requested word to the EBox, and stores the words in the cache (refills the cache). Once instructions and operands have been moved from core to the cache, the EBox can fetch instructions much faster via the cache on subsequent references since a time-consuming core cycle will not have to be executed. Fetching 4-word blocks instead of single words from memory (and, due to the principle that the program may need the next sequential word or words in the program) results in what is referred to as "lookahead." Another characteristic of programs is to execute the same instructions many times as in iterative loops. In this situation the cache is particularly effective because once the instructions and data are resident in the cache, further references to core will not be required in executing the code composing the loop.

For write operations, the MBox writes the word directly into the cache instead of core. Write operations to core are initiated only when core needs to be updated. This feature has the effect of conserving core cycles while a user program is running.

The channel I/O processor is a multiplexed channel controller that can handle up to eight simultaneous, high-speed block transfers without program intervention. Up to eight internal channels can be implemented on a KL10-based DECsystem-10. 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 employs 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 and NOT CROM operations, the channel controller will transfer blocks of four words to/from memory via the four MBs. The CBus transfers the data to/from the appropriate Massbus controller at a time.

4.2.3 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 employs ECL logic and is, therefore, 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 instruction set.

The interval timer provides a programmable source of interrupts having 10 μ s resolution and a choice of $2^{12} - 1$ possible time intervals ranging from 10 μ s to 40.95 ms.

The readable time base is a long-term clock for measuring elapsed time with 1 μ s resolution. (Long-term power line frequency time base is provided by the console 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 consist of 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 KL10-based systems. 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 not easily measured by software techniques. The signals to be monitored are selected by means of a Boolean expression loaded by the program.

4.2.4 E/M Interface

The interface between the EBox and the MBox is essentially asynchronous in terms of the request/response dialogue. However, the two units run synchronously in step with the same master clock (50 or 58 MHz). The EBox runs at a variable clock rate that is governed by the time field of the microcoded instruction word and the response time of memory (MBox) and/or the I/O device; and the MBox runs at 25 or 29 MHz. The response time between EBox request and MBox response varies with the type of request made and whether a memory cycle is required. The EBox declares the type of request by asserting a specific set of request qualifiers.

4.2.5 SBus and External Memory

The SBus is used to connect the MBox to the external core memory subsystem via the DMA20 memory bus adapter. In the 1080 and 1090 systems, the core memory subsystem may include a mixture of the following types of memory modules:

1. MF10 Memory
2. MG10 Memory
3. MH10 Memory.

The MF10 memory is a 4-port memory and the MG10 and MH10 memories are 8-port memories. When mixing these memories, configurations are limited to those that can be implemented on the MF10. Besides having eight ports, the MG10 has twice and the MH10 has four times as much memory capacity in the same space as the MF10 memory. All three memories must be manually configured (setting up appropriate address switches) to complement the desired interleaving mode. In addition, the DMA must be configured by the system initializing software to operate in 1-bus, 2-bus, or 4-bus mode to complement the designed interleaving mode (Table 4-1). To set up the DMA, the system initializing software executes a series of SBDIAG (BLKI PI) instructions.

**Table 4-1
Interleaving Configurations**

Interleaving	DMA Bus Mode
None	1
2 way	2
4 way	4

The following paragraphs summarize how read, write, and read-pause-write operations are performed over the SBus.

4.2.5.1 Read Operations – Read operations are initiated by the MBox to refill the pager, refill the cache, fetch the words the internal channel control requested, and fetch the word the EBox requested. To start a read operation, the MBox forms and places the physical memory address (PMA) on the SBus ADDRESS lines, specifies the number of words needed (1, 2, 3, or 4) by asserting the appropriate SBus RQ lines, specifies the type of cycle by asserting the SBus RD line, and asserts SBus START.

After decoding the incoming SBus request, the DMA enables the appropriate KBus(es), issues a memory cycle request, and presents the memory address to those modules that are connected to the enabled KBus(es). If the addressed storage module is not busy, it reads the address into its address buffer and responds with an ACKNOWLEDGE signal. The ACKNOWLEDGE signal is gated through the DMA to the MBox to indicate the acceptance of the memory request.

NOTE

A storage module is busy if it is being accessed through another port.

The DMA then controls the core read cycle to transfer the data between core memory and the MBox and to restore the addressed memory location as follows:

1. During the read portion of the core read cycle, the memory module reads the data and its parity from the addressed location and places it into the memory data buffer. Then the data along with its parity, is transferred to the associated data buffer in the DMA. Providing that an ACKNOWLEDGE signal was received, the associated DMA data buffer is enabled, causing the data to be parity checked and then transferred from the buffer to the associated MBox MB via the SBus DATA lines. A memory-generated restart signal which is gated through the DMA to the MBox (DATA VALID) indicates that the requested data has been transferred and can be placed into the appropriate MB.
2. During the restore portion of the core read cycle, the storage module is disconnected from the KBus to write the data and parity held in the memory data buffer back into the addressed core location.

4.2.5.2 Write Operations – Write operations are initiated by the MBox to write-back written words from the cache, store a word in response to a specific EBox request, and store words in response to an internal channel control request. To start a write operation, the MBox forms and places the physical memory address (PMA) on the SBus ADDRESS lines, places the data to be written into the MBs whose outputs are connected to the SBus DATA lines, specifies the number of words to be written (1, 2, 3, or 4) by asserting the appropriate SBus RQ lines, specifies the type of cycle by asserting the SBus WR line, and asserts SBus START.

The DMA then responds to the incoming request as described for the read operation. If the memory is not busy the DMA continues the operation by controlling the core read cycle to clear the addressed location(s) and to transfer the data from the MBox MBs to core memory as follows:

1. During the clear portion of the core write cycle, the memory module reads the data and its parity from the addressed location and discards it. This operation clears the memory location.
2. During the write portion of the core write cycle, the DMA places the data and parity that is held on the SBus DATA lines into the associated DMA data buffer, checks parity and then gates the data and parity to the memory module via the selected KBus. The memory module completes the cycle by writing the data and parity into the addressed core location.

4.2.5.3 Read-Pause-Write Operation – Read-pause-write operations are initiated by the MBox providing this operation is specified by the EBox and the cache is not to be used (refer to *MBox Unit Description*). Normally this type of memory operation is requested by the EBox only to interlock data bases of multiple processor systems. To start a read-pause-write operation, the MBox forms and places the physical memory address (PMA) on the SBus ADDRESS lines, specifies the word to be transferred by asserting the appropriate SBus RQ line, specifies the type of core cycle by asserting both the SBus RD and WR lines, and asserts SBus START.

The DMA, in cooperation with the addressed memory module, then responds by transferring the data and its parity bit from the addressed memory location to the MBox. The MBox, in turn, transfers the data along with its parity bit to the EBox, which then operates on the data to modify it. After the data is modified, the data is returned to the core memory module via the MBox, SBus, and the DMA where it is then written into the same memory location. The core memory module, the DMA and the MBox will all remain busy for the duration of this operation, preventing access to the memory module. As described for both read and write operations previously, parity is checked by the DMA in both directions.

4.2.6 DMA20

The DMA20 memory bus adapter adapts the KL10 storage bus (SBus) to the external memory bus structure. Besides serving the bus adapting function, the DMA20 also splits the SBus into four separate buses (KBus 0-3) to facilitate 4-word transfers and 4-way interleaving of storage modules thereby speeding up memory transfers. The bus adapter contains the bus control logic, an address buffer register, and four data buffer registers.

The DMA20 is capable of operating in one of three modes. The modes are 1-bus, 2-bus, and 4-bus and are directly associated with the type of interleaving to be used. The desired mode is selected by executing an SBus diagnostic cycle (BLKI PI) when the system is initialized during the bootstrapping operation. Besides executing the SBus diagnostic cycle, the memory modules connected to the memory buses (KBus 0-3) must be manually set up to exhibit the correct physical memory address and interleave mode.

4.2.7 EBus

The EBus is used to connect all KL10 I/O controllers in parallel with the EBus control logic and priority interrupt control logic of the EBox. In the 1080 and 1090 systems, three types of controllers can be connected to the EBus; they are:

1. DTE20 Ten-Eleven Data Interface
2. DIA20 IBus Adapter Control
3. RH20 Massbus Controller.

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

1. Deposit into memory
2. Examine memory
3. DMA type byte transfer operations between KL10 and PDP-11 memory
4. Interprocessor interrupt facility (doorbell)
5. Diagnostic operations for obtaining processor status and diagnostic information and for controlling the processor from the console.

Up to four DTE20s can be connected to the EBus. Only the privileged DTE20 which interfaces to the console front-end is permitted to exercise the diagnostic functions.

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 non-data transfer commands for setting up a mass storage drive in preparation for a data transfer operation.

2. Execute data transfer commands to transfer data over the Massbus and CBus in cooperation with the channel control logic of the MBox.

Up to eight RH20 controllers can be connected to the EBus.

The DIA20 is used in the 1080- and 1090-based KL10 processor to communicate with all I/O bus type peripheral controllers and devices. Its main functions are to adapt the EBus to the I/O bus and level conversion. Only one DIA20 can be connected to the EBus.

The backplane in which the controllers are housed are 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 0-7; the DTE20s are assigned physical numbers 8-11; and the DIA20 is assigned physical number 15. This scheme facilitates replacement without having to require the physical number on the controller.

The DTE20s and RH20s are also each assigned a unique device code for addressing purposes in the same way that device codes are assigned to the I/O bus type controllers. The device code is hardwired on the backpanel just like the physical number.

NOTE

The DIA20 is not assigned a device code but is assigned a physical number.

The following paragraphs summarize how output, input and interrupt operations are performed over the EBus.

4.2.7.1 Output Operations – Output operations via the EBus are executed by the DATAO and CONO instructions. In addition to a unique operation code reflecting the operation to be performed, the instructions specify the destination device code and the source memory location of the data to be transferred.

NOTE

The BLKO instruction causes a DATAO operation to be executed after the word pointer is updated. The pointer is used as the source address for the data to be transferred.

The EBox places the device code on the EBus CONTROLLER SELECT lines, the data to be transferred on the EBus DATA lines, the operation code on the EBus FUNCTION lines, and asserts DEMAND.

All controllers except the DIA20 compare the code on the EBus CONTROLLER SELECT lines with the hardwired device code when they detect the leading edge of EBus DEMAND. If a true comparison results in a controller, that controller asserts EBus ACKNOWLEDGE. At the same time, the addressed controller decodes the code on the EBus FUNCTION lines (DATAO/CONO) and strobes the data on the EBus DATA lines into the appropriate register. After the data is strobed into the register the controller asserts EBus TRANSFER.

Because the DIA20 does not have an assigned device code and has no way of knowing what devices and device codes are implemented on the I/O bus, it does not perform the device code comparison but simply simulates the I/O bus dialogue if a DTE20 or an RH20 controller does not recognize the device code. The DIA20 will:

1. Simulate the I/O bus timing and control dialogue if an ACKNOWLEDGE signal is not generated by a controller after detecting the leading edge of DEMAND; when the output operation to the selected controller on the I/O bus is done the DIA asserts EBus TRANSFER. However, ACKNOWLEDGE is not asserted by the DIA20.
2. Remains idle if a controller asserts ACKNOWLEDGE after the DIA20 detects the leading edge of DEMAND.

The EBox clears DEMAND to terminate the output operation when one of the following conditions occur:

1. The EBox receives TRANSFER.
2. The EBox does not receive TRANSFER within a specific period of time after it asserts DEMAND. This condition implies an error condition in that the device code was not recognized by a controller and the DIA20 also did not respond.

4.2.7.2 Input Operations – Input operations via the EBus are executed by the DATAI and CONI instructions. In addition to a unique operation code reflecting the operation to be performed, the instructions specify the source device code and the destination memory location of the data to be transferred.

NOTE

The BLKI instruction causes a DATAI operation to be executed after the word pointer is updated. The word pointer is used as the destination address for the data to be transferred. The CONSO and CONSZ instructions cause a CONI operation to be executed. A test (for a one) is then made on the input data to determine whether the skip condition is satisfied.

The EBox places the device code on the EBus CONTROLLER SELECT lines, the operation code on the EBus FUNCTION lines, and asserts DEMAND.

All controllers, except the DIA20, compare the code on the EBus CONTROLLER SELECT lines with the hardwired device code when they detect the leading edge of EBus DEMAND. If a true comparison results, the addressed controller asserts EBus ACKNOWLEDGE. The controller then places the selected data on the EBus DATA lines and asserts EBus TRANSFER.

The DIA20 simulates the I/O bus dialogue to get the data from the selected controller if the DTE20 or an RH20 controller does not respond by asserting EBus ACKNOWLEDGE. This is done for the same reasons given in the description for output operations (Subsection 4.2.7.1). When the data is received from the selected controller the DIA20 places the data on the EBus DATA lines and asserts EBus TRANSFER.

The EBox strobbs the data on the EBus DATA lines into the AR and clears EBus DEMAND after it senses the leading edge of TRANSFER, provided that TRANSFER is detected within a specific period of time after DEMAND is asserted. If the EBus does not receive TRANSFER within a specific period of time after it asserts DEMAND, it clears DEMAND also. This condition implies an error condition in that the device code was not recognized by a controller and the DIA20 did not respond.

4.2.7.3 Programmable Interrupt (PI1-7) Operation – Part of the control information sent from the EBox to a controller during system initialization is a 3-bit interrupt channel number. When the conditions for initiating an interrupt request are met in a controller, the controller will apply a signal to one of the seven PRIORITY INTERRUPT lines. The EBox detects the priority interrupt and resolves the interrupt priority. When the EBox is ready to serve one of the seven interrupt requests, it places the interrupt channel on the CONTROLLER SELECT lines, encoded PI served on the FUNCTION lines, and asserts DEMAND.

Every controller (including the DIA20), upon detecting the leading edge of DEMAND, decodes the function code to be PI served and compares the low order three bits of the CONTROLLER SELECT lines with the channel on which it is interrupting. If a true comparison results (one or more controllers may have true comparison), the controller will place a “1” bit on the DATA line according to its physical controller number. Controller 0 and controller 1 (physical no. 0 and 1) will put a “1” bit on bits 0 and 1 of the DATA lines, etc. ACKNOWLEDGE and TRANSFER are not asserted by any controller.

The EBox, after asserting DEMAND, waits a specific period of time, strobbs the DATA lines, and negates DEMAND. The data will remain valid until DEMAND is negated.

The EBox, after determining which physical controller’s interrupt request it wants to serve, places the interrupt channel number, in binary, on the low order three bits of the CONTROLLER SELECT lines; the controller’s physical number, in binary, on the next four bits of the CONTROLLER SELECT lines (CS00, CS01, CS02, CS03 = 1010 for physical controller 10 etc.); and encoded PI address in on the FUNCTION lines. It then asserts DEMAND.

Every controller (including the DIA20 adapter) decodes the function code to be PI address in, and compares the interrupt channel number and physical number with its own. If true comparison results (only one controller will have true comparison), the controller asserts ACKNOWLEDGE.

If the controller is a DTE20 or RH20, it places the interrupt function, the index (if any), and the interrupt address on the DATA lines and asserts TRANSFER.

If the controller is a DIA20 it simulates the KA10/KI10 I/O bus interrupt sequence and places the interrupt function, the index (if any), and the interrupt address (if any) on the DATA lines and asserts TRANSFER at the end of the interrupt sequence.

The EBox strobes the data from the DATA lines and negates DEMAND:

1. A specific period of time after DEMAND is asserted if TRANSFER is not detected. The EBox should note that this is an error condition.
2. A specific period of time after TRANSFER is detected, provided TRANSFER is detected within a specific period of time after DEMAND is asserted. This is the normal condition.

4.2.7.4 High Priority (PI0) Interrupt Operation – This priority level is permanently assigned to the DTE20 to facilitate deposit, examine and byte transfer operations. The EBus dialogue to service this interrupt is similar to that described in Subsection 4.2.7.3, with the exception that it preempts all other pending priority requests.

4.2.8 CBus

The CBus is a synchronous bus system that connects the integral channel control logic of the MBox to a maximum of eight RH20 Massbus controllers. These controllers are selected (scanned) in such a way that the first four controllers (0–3) will handle a data transfer rate of approximately one 36-bit word per microsecond while the second four controllers (4–7) handle a data transfer rate of half that speed.

The MBox is a logical unit which provides the path to the main memory subsystem for both the integral data channels and the EBox. Each Massbus controller can control up to eight mass storage disk drives (fixed-head disks or moving-head disks) or up to eight TM02 magtape controllers with each controller having up to four TU16 or TU45 drives. The purpose of the CBus is to provide a high-speed path between the MBox channel control logic and up to eight controllers for control and data information.

A clock-time-division multiplexing technique is used to control the CBus operations. A free-running clock exists in the EBox and is sent to the MBox by internal connections. One delay line per Massbus controller is used to synchronize (deskew) the signals between each Massbus controller and the channel control logic of the MBox.

The channel control continuously selects one of the eight controllers by generating eight selection lines in the following sequence: 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 6, 7; 0, 1, 2, 3, 4, 5. . . . The sequence is stepped with the leading edge of the clock signal.

A Massbus controller is allowed to transmit or receive data and control information only after it has been selected by the channel control. Four major cycles are used by the channel control and a Massbus controller during a data transfer operation. Each cycle is asserted by the leading edge of a clock pulse and is negated by the leading edge of the next clock pulse.

1. **SELECT cycle** – The SELECT line of a particular Massbus controller is asserted throughout this cycle.
2. **REQUEST cycle** – The selected Massbus controller will assert the REQUEST line (if data request is needed) throughout this cycle.
3. **WAIT cycle** – This cycle is used by the channel control to prepare data and status for transmission. Neither data nor status is asserted during this cycle.

4. DATA cycle – Data are placed on the DATA lines either by the MBox (output data transfer) or by the Massbus controller (input data transfer) during this cycle. The recipient of the data will strobe the data lines at the trailing edge of the data cycle. All CBus control lines (ERROR, READY, LAST WORD, CTOM, START, RESET, DONE, and STORE), except the REQUEST line, are allowed to be asserted during this cycle only.

Controllers 0, 1, 2, and 3 are selected twice as often by the channel control's selection sequence as controllers 4, 5, 6, and 7.

1. The maximum transfer rate of Massbus controllers 0, 1, 2, 3 is 1 μ s/word.
2. The maximum transfer rate of Massbus controllers 4, 5, 6, and 7 is 2 μ s/word.

4.2.9 DTE20

The DTE20 Ten-Eleven Data Interface serves as the interface between the KL10 Central Processor and the PDP-11 console or front-end processor(s). To the central processor, the DTE20 appears as a standard EBus-compatible device controller and to the console/front-end processor the DTE20 appears as a standard Unibus device controller. Consequently, 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 dialogue to load or read internal DTE20 registers. The processors initiate their respective bus sequences when they execute I/O instructions for the DTE20. The interrupt logic is included in the DTE20 to allow the processors to interrupt each other in passing status information and in executing data transfers. The diagnostic control logic in the DTE20 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 dialogue is not initiated in executing diagnostic functions but the EBus data lines may be used in transferring the data between the DTE20 and the EBox and MBox.

The DTE20 is capable of executing the following operations:

1. Deposit
2. Examine
3. Byte Transfer
4. Interprocessor Interrupts
5. Diagnostic

The examine and deposit functions are included in the DTE20 so that the console processor or a front-end processor can fetch or change any location in the KL10 physical memory while the EBox is either running or just executing a halt instruction. After being initiated by the PDP-11 console/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 console/front-end processor must assemble the address and assemble or disassemble the data in the DTE20 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 console/front-end and the central processors can initiate high-speed DMA type data transfers between each other's memories or between KL10 main memory and an NPR-type PDP-11 device. Once initiated the DTE20 uses the processors high priority interrupt facility without further intervention by the program to execute that transfer. On the KL10 side, the special PIO request is used to fetch or store a byte in KL10 memory and on the PDP-11 side the non-processor interrupt request (NPR) is used to fetch or store a byte in PDP-11 memory or an NPR-type device. Interrupt requests PIO and NPR are issued until the byte count maintained by the DTE20 is zero, at which time an interrupt request is issued on the assignable priority channel(s).

During the byte transfer operation the DTE20 transfers fields of information between PDP-11 memory and KL10 memory via the EBox. On the KL10 side, the fields are of variable lengths and are accessed through DTE20 byte pointers in the EPT. On the PDP-11 side, fields are either 8 bits wide and are stored in consecutive bytes or 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 interprocessor interrupt facility (doorbell function) permits the processors connected to the DTE20 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 KL10 side and BR4-7 on the PDP-11 side.

The diagnostic facilities are included in the DTE20 so that the console processor (the privileged 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 involves 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 EBox
 - b. Load CRAM in EBox
 - c. Load AR in EBox
 - d. Load MBox control functions
 - e. Enable EBus register
 - f. Reset DMA
 - g. Load channel control functions
4. Various Read functions to facilitate reading all diagnostic registers and RAMs of the EBox, MBox, and meters.

4.2.10 RH20

The RH20 Massbus controller (MBC) is the internal channel mass storage interface for a KL10-based DECsystem-10. Each MBC is capable of controlling up to eight Massbus-compatible devices (disks, drums, or magtapes). The MBC can be interfaced with single- or dual-ported drives and will allow time-shared swapping and paging software to operate the system efficiently with minimum latency. Up to eight MBCs can be connected to a DECsystem-10 and one or more MBCs can operate (reading or writing) simultaneously. However, only one drive per MBC can transfer data at a time. Non-data 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 involved 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 a DECsystem-10.

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 magtape) 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 currently 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 non-data transfer (seek, rewind, etc.) 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 finished 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, etc.) 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 classified into two categories: non-data transfer and data transfer.

The non-data 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. Typically, the drive will assert the MBus ATTENTION line to inform the MBC and the EBox that the setup is complete.

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

4.2.11 DIA20

The DIA20 IBus adapter control adapts the KL10 execution bus (EBus) to the external I/O bus structure. The DIA20 is the default controller on the EBus. If another controller does not respond to the EBus dialogue, the DIA20 will pass on the dialogue to the I/O bus to facilitate communication with I/O bus type devices.

4.2.12 Interrupt Facility

The central processor has an 8-level priority interrupt facility. Seven levels (levels 1–7) are programmable and one level (level 0) is permanently assigned to the DTE20(s). The order of priority is from 0 to 7, with level 0 having the highest priority.

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

Priority level 0, the highest priority level, is not assignable but is reserved for the console/front-end processor interface (DTE20) in executing deposit, examine, and byte transfer operations. The DTE20 can also be assigned a programmable interrupt level for reporting status information and for requesting service.

4.2.13 Trap Facility

The direct I/O and priority interrupt facilities permit the processor to maintain system status and to effect control. To supplement these facilities the central processor also incorporates a trapping mechanism. This mechanism allows certain conditions resulting from an executing program to interrupt the program sequence without having to resort 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
6. Monitor Calls (UUO and MUUO)

4.2.14 Internal Devices

The central processor contains several internal devices which can be accessed under executive mode 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.

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 I/O instructions.

4.2.14.1 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.

4.2.14.2 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.

4.2.14.3 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.

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

4.2.14.5 TIM and MTR – The meter (TIM and MTR) device facilitates access to the following built-in clocks:

1. The 1 μ s interval timer which is a source of programmable interrupts with a maximum period of 32 ms
2. The 1 μ s 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.

4.2.15 External and Internal I/O Controllers and Devices (Typical)

Every device controller on the EBus and I/O bus 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 3-9 of the instruction so that only the addressed device responds to signals sent by the KL10 Central Processor over the EBus. To use the device with the priority interrupt system, 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 denote 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.

4.2.16 Machine Instructions

Two basic types of instruction are implemented for the KL10 Central Processor. They are those that reference memory and execute a particular operation such as data transmission, logical operations, arithmetic operations, or program control operations and those that reference memory and perform I/O operations for internal devices and for devices connected to the EBus (refer to Section 3, System Features).

4.3 CONSOLE PROCESSOR SUBSYSTEM

The console processor is a unified hardware/software subsystem that replaces the KA10/KI10 based standard console I/O equipment (which is used for read-in operations), the computer operator console, and the three arithmetic processor indicator panels. To replace this hardware, the console processor provides an automatic bootstrap mechanism and a console command facility. The console processor which incorporates a PDP-11 minicomputer system interfaces with the central processor through the DTE20 Ten-Eleven Interface which is set up to operate in its privileged mode.

The bootstrap facility automatically initializes the dispatch and control RAMs in the EBox, places a small sequence of KL10 code into KL10 memory, and starts its execution to load the system. Several bootstrap options including an operator/software dialogue are available. The console command facilities are available through the console terminal (CTY) to permit the operator to examine or deposit KL10 memory locations and otherwise control and monitor the processor in the same way previously provided by the replaced operator console and indicator panels.

The PDP-11, used as the console processor, is a 16-bit, general purpose, microprogrammed minicomputer which uses single- and double-operand instructions and 2's complement arithmetic. Included in the console processor are the associated PDP-11 controllers and peripheral devices.

The instruction word format is such that the processor can directly address up to 32K words (64K bytes) of core memory. Only 28K are available for program storage. The remaining 4K 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 accomplished 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.

4.3.1 Devices

The following paragraphs provide a brief description of each component that is part of the console processor subsystem.

4.3.1.1 KD11-A Central Processor – This device is the basic component of the console processor. The KD11 is connected to the Unibus as a subsystem and 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).

4.3.1.2 KY11-D Programmer's Console – This device is an integral part of PDP-11 system and provides the operator with 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, thus allowing the operator to start, stop, load, modify, or continue a program.

4.3.1.3 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.

4.3.1.4 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 planar configuration. Storage capacity is 16,384 words (32,768 bytes), having an 18-bit word length (16 data bits plus two parity bits).

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

4.3.1.6 BM873-YD/YG ROM Loader Module – This module is an essential part of the console 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 which serve as entry points for these routines. The routines in the ROM facilitate bootstrapping and dump operations to be initiated from the KL10 via the DTE20 10/11 Interface and by pressing a physical LOAD pushbutton on the margin check panel (refer to *Power System Description Manual*).

Either the PDP-11 based DEctape or the PDP-11 based RJP04/06 disk may be specified for the bootstrap or dump procedure.

NOTE

These storage devices are not supported as system devices.

4.3.1.7 DL11-C Asynchronous Line Interface – This device provides simultaneous 2-way transmission between the console terminal and the Unibus. The DL11-C is a character-buffered interface which controls and translates serial bit stream 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.

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

4.3.1.9 LA36 Keyboard Terminal – This device is a 30-character per second, serial upper/lowercase character printer with a 132-column format. The terminal interfaces to the Unibus through the DL11-C and is used as the main console terminal (CTY).

4.3.1.10 RJP04/06 Disk File System – This system consists of an RP04/06 disk drive and an RH11 device controller. The RP04/06 is a dual Massbus port, moveable head, disk pack drive used as the prime storage media for KL10 and PDP-11 diagnostic and bootstrap load routines. The RH11 provides the control and parallel data path interface between the Unibus and RP04/06 through the Massbus. In addition to communicating with the KD11, the controller has access to PDP-11 memory to fetch and store data.

NOTE

The drive is not supported as a system device from the PDP-11 side but is supported as a system storage device from the KL10 side. (Refer to Secondary Memory Subsystem Description.)

4.3.1.11 TC11-G Magnetic Tape System – This system consists of a TC11 control unit and TU56 tape transport. The TC11 controls transport selection and tape motion and direction. In addition it controls transport read and write operations, and buffers data transferred in either direction. The TU56 is a dual unit, bidirectional tape transport which handles 3/4-inch magnetic DECtape. The transport uses the Manchester-phase recording technique with redundant recording. The tape serves as an auxiliary storage media for KL10 and PDP-11 diagnostic and bootstrap load routines; it is not supported as a KL10 system device.

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

4.3.1.13 DTE20 Ten-Eleven (Console/Front-End Processor) Interface – This device connects the KL10 central processor with the PDP-11 console/front-end processor to provide the system with interprocessor interrupt, examine, deposit, byte transfers, diagnostic, and boot facilities.

On the EBus, the DTE20 appears in some respect as a KL10-based DECsystem-10 device controller. On the Unibus it connects as a standard PDP-11 peripheral device, using the direct memory access and vector interrupt features of the PDP-11.

4.3.2 Interdevice Transfers

Communication between two devices on the Unibus is 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 act 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 CPU and all its I/O devices, a priority structure determines which device obtains control of the bus. Consequently, 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 obtained 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 accorded 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.

4.3.3 Functions

One of the major functions of the console processor is to provide the traditional (KA10/KI10) console functions. The console processor is programmed to accept console commands to display and change locations in KL10 memory, to start and stop the central processor, and to affect numerous other operations. The functions, initiated through the terminal connected to the console processor, are implemented by the following hardware-implemented operations.

1. Examine
2. Deposit
3. Byte Transfer
4. Interprocessor Interrupts
5. Diagnostic and Miscellaneous Console Functions
6. System Bootstrap.

4.3.3.1 Examine/Deposit Operations – The examine and deposit functions allow the console processor to fetch or modify any locations in KL10 memory while the central processor is running or executing a halt instruction. 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 central processor is halted.

The deposit operation accesses and writes a 36-bit data word into a KL10 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 KL10 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 PDP-11 program writes the KL10 address into the DTE20. No program interrupts are generated on the KL10 or PDP-11 side to indicate completion of the operation. Therefore, the PDP-11 program must check for completion by monitoring the appropriate flag in the status register of the DTE20. The DTE20 logic is structured such that once the address and data is written into the DTE20 it remains intact after an operation. The PDP-11 may now perform repeated examines and deposits by changing the KL10 address each time.

4.3.3.2 TO10/TO11 Byte Transfer Operations – TO10/TO11 transfers are multiple bus operations (involving both the EBus and Unibus) transferring fields of information between the KL10 and PDP-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 KL10 and the destination is the PDP-11. In the TO10 transfer, the source and destination are reversed, i.e., the source is the PDP-11 and destination is the KL10.

The fields of information transferred between the processors differ at the 10/11 interface (DTE20). At the KL10 side of the DTE20, the fields are of variable length and are accessed through a KL10 byte pointer. At the PDP-11 side of the DTE20 the fields are either 8 bits wide and stored in consecutive byte locations in PDP-11 memory, or 16 bits wide and stored in consecutive (even) word locations in PDP-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 filled with zeros in the high-order bits.

Prior to the actual transfer, several parameters are provided by both processors to the DTE20. Initially (possibly at system startup), the PDP-11 determines the transfer rate and 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 DTE20.

4.3.3.3 Interprocessor Interrupts – These interrupt operations provide the interprocessor communication function, i.e., the capability of either processor to interrupt the other. The interprocessor interrupts (doorbell feature) allow the KL10 to interrupt the PDP-11, as well as the PDP-11 to interrupt the KL10. The doorbell consists of a programmable interrupt and status flags located in the KL10 and PDP-11 status registers of the DTE20.

For the PDP-11 to interrupt an interfaced KL10, the PDP-11 sets an interrupt flag in its associated DTE20 status register using a DATO. With the flag set, the DTE20 generates an interrupt to the KL10. The KL10 then executes a CONI which informs the KL10 that the PDP-11 has programmed an interrupt to the KL10 for it to initiate appropriate action.

The procedure is executed in a reversed but similar manner for the KL10 interrupting the PDP-11. In this case, the KL10 sets an interrupt flag by executing a CONO to the DTE20 which in turn generates an interrupt to the PDP-11. The PDP-11 discovers the cause for the interrupt by monitoring the flag in the DTE20 status register and initiating appropriate action.

4.3.3.4 Diagnostic and Miscellaneous Console Functions – Other console functions, such as displaying the contents of certain KL10 registers or memory locations require the cooperation of the operating systems. The operating system must periodically store the quantities to be displayed in the communications areas (Subsection 4.3.5). Displayed information may include the PI system state, current job being run, number of active jobs, program counter on the last clock interrupt, etc. It is also possible to simulate all of the traditional console indicators used while the system is in normal operation.

Major KL10 CPU state information is continuously available on the diagnostic bus while the system is running. In addition, for the case of a KL10 crash, the PDP-11 can use the diagnostic bus to determine additional hardware status information. However, the console 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.

4.3.3.5 System Bootstrap Function – Initially both systems must be loaded by the bootstrap handler in order to begin operation. Bootstrapping can be initiated in several ways:

1. By a power fail restart
2. Operator depressing a bootstrap button (DECtape, disk pack, or switch register) on the system margin check/switch panel
3. KL10 initiating a bootstrap of the PDP-11
4. Operator entering appropriate commands to the console command handler via the console terminal.

Generally, the bootstrap is initiated from the BM873 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 PDP-11. The bootstrap then loads the KL10 control RAM and dispatch RAM with sufficient code to start the KL10 running. As more code is transferred, the PDP-11 will configure KL10's memory, load the resident monitor, set up communications with the console and front-end processors, and eventually relinquish control to the resident TOPS-10 monitor.

4.3.4 Modes

The DTE20 Ten-Eleven Interface has two switch-selectable operating modes: privileged and restricted.

A processor that is connected to a privileged DTE20 has access to the diagnostic bus and the capability to execute unprotected examines and deposits. The console processor must operate in this mode. Unprotected examines and deposits are unique in that they require special software and may address any of the following areas in KL10 memory: executive 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 executive process table (EPT). Because of the relatively unlimited access allowed, a privileged console processor may seriously degrade system operation or crash the associated KL10 Central Processor.

Non-console front-end processors must connect to a DTE20 that is set up in the restricted mode. A restricted processor can only access KL10 memory provided the KL10 has executed a CONO instruction and enabled the associated PIO level. After PIO is enabled the restricted processor can then only examine in a KL10-owned region and deposit in its own PDP-11-owned region. In addition, the processor is prohibited from using the diagnostic bus. Since the restricted processor cannot violate the KL10'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 seriously degrade the KL10's operation.

4.3.5 Interprocessor Communications

Interprocessor communication is necessary to allow the PDP-11 and KL10 to execute those functions required during time-sharing, bootstrapping, and diagnostics. This communication is implemented by special communication areas allocated in KL10 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 (i.e., KL10-owned region, PDP-11-owned region), but only read the other system-owned area. These areas effectively 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 KL10 and PDP-11 is through software processing queues. A TO11 queue is maintained in KL10 memory by the monitor. The PDP-11 will access this queue using byte transfers through the DTE20. The TO10 queue is maintained in the PDP-11 memory and is accessed by the KL10 in a similar manner. 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 constructing processor. When a queue is constructed and ready for transfer operations, the transmitting processor will interrupt its associated processor. At this point the interrupted processor begins processing the queue.

The KL10 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.

4.3.5.1 Communication Areas – Initially the KL10 will set up the communications areas at load time with each processor responsible for protecting itself from the other. Since interprocessor communication is through the DTE20, a pair of communications areas is associated with the DTE20 (i.e., KL10-owned and PDP-11-owned areas).

The PDP-11-owned area is defined in KL10 memory by the deposit relocation and protection word in the EPT (EPT DPW). The area is written by the PDP-11 using protected deposits and read by the KL10. Each PDP-11 in the system has a separate area which it alone can modify. The KL10-owned area is defined purely in software and separate from the PDP-11-owned area. It is written by the KL10 and read by the PDP-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. Thus, the size of each communication area is variable, depending on the number of processors in the system.

4.3.5.2 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 accumulated queue.

After a transmitting processor places information in its queue, it interrupts the receiving processor notifying it to start processing queue entries using byte transfers. The DTE20 hardware is such that 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 PDP-11 provides the DTE20 with the source address of the queue to be transferred.

2. The PDP-11 specifies how the queue is stored in KL10 memory, i.e., 8-bit bytes of 16-bit words.
3. The KL10 sets the EPT byte pointer word to a byte pointer locating 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 identifying the PDP-11 device inputting data, or the PDP-11 device for which the data is destined. Message content may contain information concerning queue processing. For example, it may contain information indicating the end of a queue and resetting the DTE20 to begin 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. These type messages are used when it is more efficient to change the byte pointers to point to another area of memory than to duplicate that memory area in the queue.

4.4 MAIN MEMORY SUBSYSTEM

In a KL10-based system such as the 1080 and 1090 systems, main memory includes core, cache, and fast memories.

Three types of core memories may be used. They are designated MF10, MG10, and MH10 memories. All of these memories can be expanded in 32K increments. The MF10 incorporates four access ports and the MG10 and MH10 memories have eight access ports. These ports facilitate memory module interleaving and sharing of memory by several processors (including external high-speed communications and data channel processors).

The cache memory is an integral part of the MBox in the central processor. Essentially, the cache is a high-speed multiple set associative extension (buffer) of core. Each time the EBox requests a word from memory, the MBox fetches the requested word and the three adjacent words if the requested word is not already in the cache. When the first word comes in it is given to the EBox and placed into the cache. The EBox then continues executing the programmed instructions. Meanwhile, the MBox will wait for the other three words to come in from core. As each word comes in it is automatically placed into the cache. While this is going on, the EBox cannot cause another core cycle to be started but can get an additional data or instructions from the cache if these words are already in the cache. Simulations have shown that after the cache is initially refilled, a hit ratio of over 90 percent can be achieved. The term "hit" applies to the condition where the requested word is in the cache. When the EBox issues a request to write memory, the MBox writes directly into the cache but does not write through to core. This characteristic of the MBox requires that core be updated at various times. A new instruction (SWEEP) is therefore implemented to facilitate this operation.

Fast memory is an integral part of the EBox. It contains 16 locations which are addressed in place of the first 16 core locations. These fast memory locations can also be used as index registers and as accumulator registers when specified in the instruction AC or XR fields.

The core memory subsystem is not dependent on the processor for its timing. It establishes its own timing in sync with the master clock in the EBox and, therefore, operates at maximum rate in an asynchronous manner. A memory module has three modes of operation: the read mode, the write mode, and the read-modify-write mode. The mode is established for each cycle by a signal from the processor.

1. **Read Mode** – In a read mode cycle the processor remains connected to the memory module only long enough to access the operand or the next instruction. The processor can then continue its own operations, and can, in fact, access a location in another memory module while the first memory independently completes its read-write cycle.
2. **Write Mode** – In a write mode cycle, the processor remains connected to the memory module only long enough to transfer the contents of the processor's memory buffer (MB) register into the module's buffer register. Each device then continues independently.
3. **Read-Modify-Write Mode** – In a read-modify-write cycle, the processor remains connected to the memory module for a length of time equal to the access time, plus the time required to execute the algorithm specified by the instruction.

A bus system runs between all memories so that each processor must be interfaced to its own distinct bus. Each MF10 memory unit has the facility for four separate buses, allowing up to four processors to access any one memory module. The MG10 and MH10 memory units have facilities for implementing eight separate buses. Each bus has a priority associated with it, allowing synchronous devices, such as displays, and high-speed storage or communication channels to have priority over the central processor. Bank selection may be done on either high-order or low-order address bits. Thus, an installation may select address interleaving between two or four memory modules for added memory efficiency. Each memory location contains 36 data bits and 1 bit for parity check. Parity is checked and generated at the processor to protect against both memory and bus transmission errors.

Memory interleaving and assignment of bus priorities may be specified for most efficient use of the system.

4.5 SECONDARY MEMORY SUBSYSTEMS

Several types of secondary memory subsystems may be included in the 1080 and 1090 systems. They are:

1. Console processor-based DEC magnetic tape, TC11/TU56 (not supported as a system device)
2. Console processor-based random access disk, RJP04/06 (not supported as a system device)
3. Central processor-based random access disk, RHP04/06 and RTP04/06
4. Central processor-based swapping disk, RHS04
5. Central processor-based standard magnetic tape, THU16 and TTU16
6. Central processor-based standard magnetic tape, TU70/72
7. Central processor-based DECtape, TD10/TU56.

The system-supported secondary memory subsystem serves as a large file storage and swapping system. Both disk and magnetic tape storage drives can be attached to either an internal or an external data channel I/O processor (also referred to as a channel controller). The internal channel I/O processor is an integral part of the MBox in the central processor. This I/O processor interfaces with main memory and an integral RH20 Massbus controller which is a universal storage controller that can be interfaced with either disk or magnetic tape drives. The external channel I/O processor is an older stand-alone channel controller (DF10-C or DAS33) which interfaces with main memory and stand-alone RH10 Massbus controller. The RH10 is also a universal storage controller that can be interfaced with either disk or magnetic tape drives.

Physically, the internal and external channels are quite different. The external channels employ stand-alone channel and Massbus controllers while the internal channels are an integral part of one CPU assembly. Functionally, the internal and external channels are similar with only minor differences in programming requirements.

NOTE

DECtape cannot be interfaced with the high-speed data channels. Data can only be transferred to/from a DECTape under direct program control.

4.5.1 Disk Subsystems

A disk (or a drum) is generally the largest random access storage device in a computer system, and it also provides the fastest storage outside of core. These devices are exceptionally desirable as backup storage for memory, especially for storage of large files and for swapping in time-sharing systems – while the currently active user programs are in core, inactive programs are stored on a disk or drum. Unlike magnetic tape, a disk or drum is constantly in motion and has a predetermined format with data blocks of fixed length. Hence, individual blocks are addressable and at the simplest level, reading and writing may be the only functions the system need perform.

In a drum unit, the data is recorded in tracks that are circles on the surface of a cylinder. In this case the tracks are all the same length, so both timing and density are constant. As with a disk, the drum is divided into sectors, and the basic data blocks are recorded in the sectors of the tracks.

In a disk pack unit the storage medium is a removable stack of disks. Hence, not only is the storage capacity much greater, but the data can literally be stored on the shelf like magnetic tape while the drive is being used with another pack. Each disk surface has tracks and sectors. However, there are many surfaces, and the set of identically numbered tracks on the various surfaces constitutes a cylinder. (Logically a disk pack is equivalent to a drum pack.) As before, the basic data block is a sector of a track, which is addressed as the intersection of a cylinder and a surface. In terms of the addressed scheme used in continuous data processing, the disk pack is treated as though it were a drum pack; the hardware counts through all the tracks (surfaces) one cylinder at a time.

If there is a separate read-write head for each track (fixed-head disk), the average random access time is a little over half a revolution; otherwise additional time may be required for head positioning. Since the storage medium is continuous, has a fixed format, and is in constant motion (both in speed and direction), functions can be limited simply to read and write, with an automatic search for an initially specified sector. However, in more complex systems, there may be a separate search function and even special functions for handling non-data parts of a sector.

A disk or drum system always consists of a control and a number of disk, drum, or disk pack units. In all cases, the program communicates with the control, which in turn governs all the units but effectively communicates with only one at a time. Data transfers between the control and device are governed by the control. The control is always connected to the EBus, but only for the transfer of initial conditions and status. For controls on the I/O bus, a DIA20 I/O bus adapter is required. Once the program sets up the system for reading or writing, data transmission between control and memory is handled automatically via a separate channel/memory bus (i.e., bypassing the central processor). To accomplish this, the control is connected to the memory bus through a data channel controller (internal or external) or contains the necessary hardware for direct connection to the bus.

4.5.1.1 RHP04/06 and RTP04/06 Disk Subsystem – The RHP04 and RTP04 disk subsystems provide high-performance, random access disk drives, with a 20 million (36-bit) word storage capacity. The RHP06 and RTP06 disk subsystems provide twice this capacity with the same high performance.

The RP04/06 disk drive is initialized through a universal controller (RH10 or RH20) and transfers data directly to and from memory at $5.6 \mu\text{s}$ per word via the RH10 or RH20 Massbus controller and the DF10-C or internal data channel controller. Since the Massbus controller provides overlapped positioner operation, the TOPS-10 operating software system will simultaneously position two or more disk drives, shortening the effective access time and increasing throughput. (Up to eight drives may be connected to a single controller.) This feature, in combination with the rotational position sensing (RPS) capability, is utilized by the TOPS-10 operating system to optimize disk access across all jobs active in the system.

The RP04/06 disk drive offers a high level of data reliability. A phase-locked loop clock system and state-of-the-art recording provide the latest in reliable reading and recording techniques. In addition, error detection and correction hardware within the RHP04/06 disk system provides information as to the position and pattern of any error burst up to 11 bits within the data field. Correction of any data field errors is then accomplished under TOPS-10 software control. The ability to offset heads also facilitates read recovery and increases data reliability. Moreover, if the sector on a disk pack becomes defective so that all attempts by the hardware and the software to recover data fail, the operating system dynamically marks the sector as bad so that no future attempts will be made to use it again.

To support the 1080 and 1090 systems, there is also a dual-port RP04/06 disk drive which interfaces with both an RH10/RH20 and RH11 controller. This disk drive is needed to facilitate diagnostic and bootstrapping functions in the event of a crash and to facilitate system dump procedures.

NOTE

Access to the RP04/06 from the PDP-11 is not supported for system storage use. Access is available only for booting and diagnosing the system. However, from the KL10 side (via the RH10/RH20) the drive may also be used for system storage.

4.5.1.2 RHS04 Disk Subsystem – The RHS04 swapping disk subsystem is a fixed-head disk that provides a time-sharing or real-time DECsystem-1080/1090 with a fast access, high transfer rate swapping device which greatly enhances system performance and load handling capacity.

The basic RHS04-C disk provides 256K 36-bit words of fast access storage for swapping and for storage of program libraries. With zero positioning time, an average latency time of 8.5 ms and a transfer rate of $4 \mu\text{s}$ per 36-bit word, the RHS04 disk system can swap a 4K user job in or out of core memory in an average of 25 ms.

Control for the RHS04 disk system is provided by the RH10 universal controller which is interfaced to both the I/O bus and to a DF10/DF10-C data channel which is, in turn, interfaced directly to the memory bus of the DECsystem-10. A single control may handle up to eight swapping disks, making possible a total swapping capacity of 2 million words.

The DF10-C data channel allows data transfers between the RHS04 system and memory to take place simultaneously with central processor computation, as long as the channel and processor are not accessing the same memory module.

4.5.2 Magnetic Tape Subsystem

Two types of magnetic tape equipment are available for use in a DECsystem-10. One handles the large reels of half-inch tape that are standard throughout the industry; the other handles DECtape, of DIGITAL's own creation. DECtape trades off some of the speed and storage capacity of standard magnetic tape for the convenience of paper tape. Its small reels are easy to handle and carry about, making DECtape desirable for program input. In addition, the redundancy used in recording helps to maintain the information intact despite heavy handling. The format of the tape allows single blocks to be replaced without affecting the rest of the tape, thus making it especially convenient for holding a library of commonly used routines.

The industry-standard magnetic tape may be used as backup storage for memory and is especially convenient for storing large amounts of data that need not be available to the system continuously. (A disk or drum can store information much faster, but physical storage of the information storing medium separate from its handling equipment is less convenient, and often impossible.)

Both types of magnetic tape equipment automatically record error-checking information while recording data on tape. This provides a means of checking for possible data loss that results from using the tape as a storage medium.

4.5.2.1 TU70/71/72 Magnetic Tape Subsystem – The TU70-series of high-speed magnetic tape drives represents the latest state-of-the-art design in tape transport technology. The following three models are offered:

1. The TU70 is a 200 inch/second, 9-track transport with program-selectable recording densities of 800 (non-return-to-zero-inverted encoded, NRZI) and 1600 (phase encoded, PE) characters/inch.
2. The TU71 is a 200 inch/second, 7-track transport with program-selectable recording densities of 200, 556, and 800 NRZI encoded characters/inch.
3. The TU72 is a 125 inch/second, 9-track transport with program-selectable recording densities of 1600 (PE) and 6250 (group coded recording, GCR) characters/inch.

All drives have a rewind speed of 500 inches/second.

The TU70-series drives incorporate such features as automatic threading and loading of 5, 8-1/2, and 10-1/2 inch reels as well as industry-standard cartridges. Dynamic amplitude control during read operations reduces read errors which avoids the need for preamplifier adjustments and allows optimum performance with different tape brands on the same drive. All TU70-series tape drives are equipped with an automatic reel hub to speed and ease tape loading and eliminate reel slippage. An analog capstan control provides constant motor drive control during read/write and rewind for smoother operations and greater drive-to-drive compatibility, while a velocity feedback reel control provides for direction and velocity sensing of the tape loop in the vacuum columns to minimize reel over-speed. This prevents tape stretching resulting in longer tape life and greater machine reliability. A linear high-speed rewind mechanism reduces rewind time and lessens stress on the reel drive system, resulting in smoother operation. A tape storage pocket is provided for convenience.

The basic TX01 and TX02 controllers handle up to eight tape drives each and provide direct access to the DECsystem-10 memory via the external data channel (DX10). The controller includes such features as improved phase-encoded error correction techniques at 1600 characters/inch for fewer permanent and temporary read errors; a read-only memory control for implementation of microprogrammed diagnostics; monolithic MSI design for greater reliability; loadable in-line microprogrammed diagnostics; programmable maintenance memory; adjustment-free read-detection; and a radial interface which allows each individual drive to be switched out or physically removed without affecting the operation of other units.

4.5.2.2 THU16/TTU16 Magnetic Tape Subsystem – The TU16 is a low-cost, industry-compatible, 9-track magnetic tape transport. It uses standard recording formats, with densities of 200, 556, 800, and 1600 characters/inch, selectable under program control. Reading and writing are performed at 45 inches/second. The recording method is NRZI for 200, 556, and 800 characters/inch and PE for 1600 characters/inch.

The TU16 magnetic tape transport operates through an RH10 or an RH20 Massbus controller (MBC) which transfers data directly to and from memory via an external or internal channel controller respectively. The first TU16 on the MBC includes a control unit plus the master tape control electronics (TM02/TM03) in the same cabinet.

Parity is checked character-by-character when reading and writing on tape to verify the accuracy of data transfer. With NRZI, there is also a cyclic redundancy check (CRC) character generated or checked at the end of each record, plus a longitudinal parity check (LPC) character generated or checked at the end of each record. Data reliability of the TU16 is enhanced by the 1600 character/inch, phase-encoding, self-clocking feature which is not dependent on precise tape speed control.

Reading can be performed while tape is moving in the forward or reverse direction; however, writing occurs only in the forward direction. To enhance tape life, tape motion is controlled by vacuum columns and a servo controlled single capstan.

4.5.2.3 TU56 DECtape Subsystem – A DECtape system consists of a TD10 control and up to four TU56 dual DECtape transports. Any number of tapes can be in motion simultaneously, but the control can monitor only one at a time. Both reading and writing can be done in either direction of tape motion, with an average data transfer rate between processor and tape of 400 μ s per 36-bit word. Each transport has two reels, which function as supply or takeup on the direction of tape motion. A full reel has 260 feet of 3/4-inch, 1 mil magnetic tape and can store 2-1/2 million data bits, 3 bits per frame (73,984 36-bit words in standard KL10 DECtape format).

4.6 BA10 UNIT RECORD (HARD COPY) EQUIPMENT

The following subsections describe the line printer, card reader, card punch, and plotter. These devices are primarily used for communication between computer and operator using a paper or card medium.

The line printer provides text output at a relatively high rate. The program must effectively typeset each line; and then upon command the printer prints the entire line.

The card equipment processes standard 12-row, 80-column cards. Many programmers find cards a convenient medium for source program input and for supplying data that varies from one program run to another. Cards are convenient to prepare manually, and simple changes are easy to make: individual cards can be repunched, and cards can be added or removed from the deck.

The plotter is an incremental digital plotter that produces quality ink plots of computer-generated data.

The BA10 hard copy equipment cabinet houses two controllers for these devices. They are the BA10 line printer/card reader/plotter control and the CP10 card punch control. The TD10 DECTape control cabinet may also be equipped with a plotter control. In addition, the CP10 card punch control may be mounted in a stand-alone cabinet.

4.6.1 CR10 Card Reader

The card reader handles standard 12-row, 80-column cards at speeds of 300 to 1200 cards per minute depending on the model. Once started, an entire card is read column by column. The reader supplies each column to the processor as 12 bits corresponding to the column punch and also in a more compact form. The program can translate the column data in any way it wishes, but the monitor automatically translates the standard DIGITAL character representation into 7-bit ASCII. Of course, the data can simply be in binary at three columns per word. (A 7 and 9 punch in the first column is the traditional indication that the rest of the card contains binary data.)

4.6.2 LP10 Line Printer

These line printers output hard copy composed of lines 132 characters long at rates of 300 to 1250 lines per minute. Various character sets are available. Besides accepting printing characters, the printer responds to ten control characters, HT, CR, LF, VT, FF, DLE, and DC1-4. Only these control codes affect the printer, but the interface recognizes two others: NUL, which is ignored, and DEL (delete), which allows expansion of the character set by providing a means for distinguishing between control characters and printing characters with the same codes. All other codes are ignored.

Printers LP10A, C and F have a 64-character print drum whose print positions are selected by the figure and uppercase codes, 040-137. Lowercase codes (140-176) are also valid for these printers: when a lowercase code is given, the corresponding uppercase code is loaded into the buffer. Printing speeds of these printers are 300, 1000, and 1250 lines/minute, respectively.

Models D and H have a 96-character drum whose print positions are selected by the figure, uppercase, and lowercase codes, 040-176, and the delete code. A single delete code is ignored, but two consecutive 177s cause the code 177 to be loaded into the buffer. When a code for a printing character is the same as one for a nonprinting character and is loaded by giving it immediately after a delete, the printing character is said to be "hidden" under the nonprinting one. Printing speeds of these printers are 600 and 925 lines/minute, respectively.

Model E has a 128-character drum and uses the entire set of 7-bit codes for printing characters, with characters hidden under the ten control characters and also under null and delete. The printing speed of this model is 500 lines/minute.

The character sets in models A to E are fixed, but models F and H have removable drums. Two standard versions (designated respectively by E and F appended to the model number) of these drums are available with EDP and scientific character sets; in the latter, zero and Z are crossed.

The printer has a 132-character buffer that holds the image of a single line; the program must first load the buffer up to five characters at a time, and then give a control character to print the entire line. The buffer is loaded from left to right, and only the portion filled produces a printout. Hence, for each line the program need send out characters (including spaces) only as far as the rightmost nonspace character. The characters are printed in the order that they pass the print hammers, and a given character is printed simultaneously in all positions that require it. In other words the drum has a row of 132 Ms, a row of Ns, etc.; all Ms are printed together, all Ns together, and so forth. The first character printed depends only on the position of the drum when the print command is given.

4.6.3 CP10-D Card Punch

The card punch handles standard 12-row, 80-column cards at speeds up to 100 cards per minute if all 80 columns are punched and 285 cards per minute if only 1 column is punched. The processor must supply each column to the punch as 12 bits, and the program can generate this data by any procedure it wishes; the standard DEC character representations and the translation from ASCII made by the monitor are given in Appendix B of the *Hardware Reference Manual*. Of course the data can simply be in binary at three columns per word. (Punching rows 7 and 9 in the first column is the standard procedure for indicating that the rest of the card contains binary data.)

A card is taken from the hopper only when the program supplies data for the first column. In the interface is a 12-bit buffer to which the processor sends each column, but the punch has a 48-bit buffer, and it punches four columns at a time from each set of four 12-bit bytes sent through the interface. The program can send a card to the stacker after punching any number of columns.

4.6.4 XY10 Plotter

The XY10 plotter control logic of the BA10 control or the TD10 control interfaces the DECsystem-10 to various plotters that use Cartesian coordinates. The models most frequently used are the Calcomp 563, 565, and 836 known as the XY10-A, XY10-B, and XY10-C respectively. These are high accuracy incremental digital plotters that produce quality ink plots of computer-generated data. Bidirectional stepping motors provide individual increments of motion in either coordinate or both at once. The program draws a continuous sequence of line segments by controlling the relative motion of pen and paper with the pen lowered. It can also raise the pen for repositioning. Plotter step size is specified at the time of order.

Motion in "y" is movement of the pen carriage along a pair of rods. Motion of "x" is movement of the entire carriage-and-rod mechanism on a bed plotter or movement of the paper under the carriage on the drum type. On a bed plotter the coordinate directions are the standard ones when viewing the device from the front: positive "x" to the right, positive "y" to the back. The coordinate system on a drum is in the standard orientation when the viewer is standing at the right side, unrolling the paper from the drum with his left hand. In other words, positive "y" is movement of the pen from the right to left across the drum, positive "x" is drum rotation downward at the front.

4.7 LP100 LINE PRINTER SUBSYSTEM

The LP100 line printer subsystem is composed of two hardware components: an LP07 line printer and an LP100 line printer controller and data source interface. Each component is in a free-standing cabinet and is equipped with the necessary power supplies and interconnecting cabling.

The LP07 line printer is a high-performance, horizontal font motion line printer. It is designed for use in data processing environments that require high-grade print quality, heavy print volumes, and high reliability. The LP07 is an impact type, shaped (whole) character, 132-column line printer. It will produce printed output at 1220 or 990 lines per minute using a 64-character set or 905 or 715 lines per minute using a 96-character set; the print speed is operator-selectable at the printer control panel. It performs a single forms (paper) step in 12.5 ms and slews forms at up to 152.4 cm (60 inches) per second when formatting vertically.

The LP07 line printer stores a stream of up to 132 characters in a print line buffer and, upon command from the data source, it prints the contents of the buffer and advances the forms as specified by the command. It signals the data source when it is ready for the next line of print data or forms motion command.

The LP07 line printer uses a Charaband™ as the horizontal font carrier, which is in front of the forms, and 132 print hammers, which are behind the forms and ribbon, to produce the inked characters on the front of the forms and carbon transfer characters on internal pages if multicopy forms are being used. The Charaband retains the advantages of train printers and minimizes the problems of character set rigidity, friction, and wear associated with other horizontal font techniques. To add convenience and flexibility to the printing tasks, the Charaband is designed with two complete character sets – one on each side. One side of the band contains 64-character EDP character sets, and the other side of the band contains 96-character EDP character sets. Reversing the Charaband, to print with the second character set font, is a procedure which takes an operator about a minute. Additional convenience and cost reduction are attained by the individually field-replaceable character-type modules that do not require the extensive realignment of train cartridges; this is a distinct advantage over chemically etched characters on band printers where the entire band must be replaced when a character is damaged or worn.

The LP07 line printer contains a direct access vertical format unit (DAVFU). The DAVFU uses a format memory that is loaded from the data source. This relieves the operator from having to install a format tape and eliminates the risk of running a print job with the incorrect format. The DAVFU consists of a 143-word by 14-bit format memory, a line strobe assembly and check circuitry, a memory parity generator and check circuitry, and control logic. The DAVFU may be loaded at any time data is requested by the printer. Format memory data and control codes are transmitted to the printer via the normal data lines using the standard demand/strobe communications. The DAVFU can control the vertical movement of forms having up to a maximum of 143 lines. The DAVFU also permits 6 or 8 lines/inch print density under program control.

4.8 COMMUNICATIONS SUBSYSTEMS

Communications subsystems facilitate the transfer of information in serial form between the computer and one or more other points, usually some distance away. Such equipment can simply connect to teletypewriters or other terminals located in a number of offices in a single plant, allowing engineers and other personnel to communicate directly with a centrally located DECsystem-10; make a large time-sharing facility available (through private lines or the standard telephone network) to many users located over a large geographical area; or allow high-speed communication between a large computation center and other computer installations located throughout the world.

Basically there are two types of serial communication. In a synchronous system, data is transmitted as a continuous bit stream, beginning with a prearranged special sequence (protocol) through which the receiver synchronizes to the stream. In an asynchronous system, all information is transmitted in distinct characters bounded by start and stop bits. Running an asynchronous channel at its maximum rate does result in a continuous bit stream, but each data character is still separated from the next by stop and start bits, and the receiver synchronizes on each character separately. Communication at low speed (up to 300 bits per second) is invariably asynchronous, whereas high-speed transmission (generally above 2500 bits per second) is usually synchronous. Either technique is used for medium speeds (above 300 bits per second but within the capacity of a voice-band channel).

™Charaband is a trademark of the Dataproducts Corporation.

The DC75-NP communications subsystem services a number of synchronous lines and the DC76 communications subsystem services a number of asynchronous lines. Both subsystems are PDP-11 based systems and, by providing conversion of data between parallel and serial forms, act as message concentrator and distributor systems. The DN87 communication subsystem is a universal communication subsystem that handles both asynchronous and synchronous lines. These systems connect to the 1080 and 1090 I/O bus and core memory via the DL10 communications channel to facilitate high-speed transfers.

The DN87S communication subsystem is a universal communication subsystem that can handle both asynchronous and synchronous lines. This system connects to the 1090 EBus via the DTE20 Ten-Eleven Data Interface to facilitate high-speed transfers using the DTE20's byte transfer facility.

4.8.1 Communication Primer

Data Communications – the transmission of coded information between terminals and computers, or between multiple computers – is the key capability inherent in all communications and computer network systems.

Asynchronous devices transmit (or receive) a serial-bit stream consisting of a number of start bits, data bits, and stop bits. The start and stop bits perform the functions of synchronizing the transmission of each data character. Data is transmitted in the form of a string of characters, each of which has the same predefined format (i.e., start bit, data bits, stop bits).

Asynchronous communication is generally employed for low-speed communications in applications such as interactive systems used for program development, operator control of the system, data entry, program entry, interactive problem solving, student instruction terminals, and information storage and retrieval equipment. Local terminals (within 1500 feet) can usually be connected over dedicated direct lines using current loop interfaces or modems. Remote terminals are connected using modems over dedicated or dial-up telephone lines.

DECsystem-10 asynchronous communications equipment generally operates in the low-to-medium-speed ranges from 110 baud to 4800 baud. Equipment such as LA36 and VT50 user terminals operate on a character-by-character basis; each time a character is typed, it is transmitted. If characters are typed at the rate of ten 11-bit characters per second, the effective transmission rate is 110 baud, regardless of whether the line is 110 baud or 2400 baud.

Some terminals, such as certain types of CRTs, operate in a buffered mode. Here, characters are buffered in the terminal until the entry is completed, at which time they are transmitted. Unlike unbuffered terminals, which transmit at the same rate as characters are entered (as long as that rate does not exceed the baud rate of the line), buffered terminals transmit the entire buffer at the full rated speed of the line. Computers, like buffered terminals, also transmit at the rated speed of the line.

Due to the differences in effective data throughput, these differences between character-by-character transmission and buffered character transmission are very important and must be considered when designing communications and network systems using asynchronous data transmission.

Synchronous devices also transmit and receive a serial-bit stream over a communications line. However, the serial-bit stream does not include start and stop bits for each character. Instead, an entire block of data is synchronized with a unique code which, when recognized, establishes character framing. This character framing is maintained by the receiver for the duration of the message.

In contrast with the character-by-character basis of asynchronous transmission, synchronous transmission is on a message basis. Because of its higher speed and more efficient utilization of the transmission medium, large blocks of data are normally transmitted via synchronous transmission techniques.

Asynchronous transmission is employed to transmit data on a character-by-character basis; for moving large blocks of data, synchronous transmission is preferred. Asynchronous transmission, at 2400 baud, is limited to conveying no more than 240 eight-bit characters per second; synchronous transmission, at 2400 bits/second, can convey 300 eight-bit characters per second. This difference comes about because synchronous transmission uses independent, block-oriented clock synchronization, while asynchronous transmission reestablishes its character-oriented clock synchronization, with each character. Thus, each 8-bit character requires, minimally, one start bit and one stop bit.

The most important distinction between synchronous and asynchronous methods of transmitting data is the means of error detection and recovery. Error detection for asynchronous devices is normally accomplished by insertion of a parity bit into the data field of each character transmitted. Half-duplex (local character echo) receivers will typically check this parity bit and notify the sender of the existence of an error. In full-duplex systems, such as the DECsystem-10, the character echo is provided by the receiving end (the DECsystem-10) and transmission error detection and corrective action are thus the responsibility of the sender, who simply compares what was sent with what the computer says it received. This mode of error detection is highly efficient for interactive, terminal-oriented communications with buffered terminals and other computers. Synchronous communication normally uses a block-oriented error detection technique, such as the cyclic redundancy check (CRC-16) polynomial. This technique is most efficient for block-oriented data transmission and for this reason, communications with buffered terminals and remote computers is normally accomplished using synchronous data transmission.

Synchronous data transmission with intelligent terminals is generally carried on under control of a communications protocol whose function is to define the format of the message – length, destination, character size, data format of character (binary, ASCII, EBCDIC, etc.) and to detect and recover errors in transmission. With synchronous protocol, the user is assured that the message is received correctly without error. When the receiving end detects an error, it will take corrective action, normally by requesting the sender to retransmit the message. The protocol also ensures that each message transmitted is saved until the receiver acknowledges error-free reception of the message. Typical synchronous communications protocols are IBM Binary Synchronous (“BSC” or “Bisynch”), DECsynch, and DIGITAL Data Communications Message Protocol (DDCMP).

IBM Bisynch protocol is a half-duplex protocol supported by 360 and 370 systems for communications with remote batch stations and computers. The DAS78 synchronous communications system supports IBM 2780 Bisynch protocol, enabling the DECsystem-10 to emulate an IBM 2780 remote batch station.

DECsynch is the traditional full-duplex communications protocol developed for support of remote stations such as the DC72. DECsynch is implemented on the DECsystem-10 with the DS10 and the DC75 synchronous communications system.

DDCMP is a message-oriented protocol for data communications between computers. Its purpose is to assure the correct sequencing and data integrity of messages transmitted over data channels subject to noise interference. It does this through sequence checking and message retransmission techniques.

DDCMP is designed to operate over full- and half-duplex channels in point-to-point and multipoint modes, independent of the bit width and other characteristics of the data channel. It is applicable to multiple computer configurations such as computer networks, host/front-end processors, remote concentrators, and remote entry/exit systems.

The DECsystem-10 uses DDCMP for synchronous communications to DC72-NP, DN80-series, and DAS92 remote stations and general-purpose remote computer systems via the DC75-NP, DN87, DN87S, and DAS85 communications systems. Correct message sequencing is assured by numbering all messages, i.e., the transmitting station (master) increments the number sent in successive messages and the receiving stations (slave) check for the correct number on reception. The data integrity is assured by the inclusion of a cyclic redundancy checking (CRC-16) algorithm which is checked by the slave station, with notification of the check sent back to the master station.

4.8.2 DC76 Asynchronous Communication Subsystem

The DC76 asynchronous communication subsystem (Figure 4-1) provides a flexible, large-capacity asynchronous data communications capability for the DECsystem-10. The fully expanded DC76 subsystem allows users to connect up to 512 asynchronous terminals with a wide choice of transmission speeds, modem connections, and data codes.

The DC76 uses up to four PDP-11 processors for multiplexer control and front-end character handling. The PDP-11 processors are interfaced directly to the DECsystem-10 main memory via the DL10 communications channel.

Each PDP-11 processor can handle up to 128 asynchronous communications lines with a total aggregate line speed of 1500 characters/second. The maximum speed of any line is 9600 baud. Input speeds above 2400 baud are not supported by standard software. The DC76 provides for automatic baud rate recognition among 110, 134.5, 150, and 300 baud lines. Other speeds from 50 to 9600 baud and split (i.e., different) transmit and receive speeds can be obtained by monitor commands and/or by the system management presetting line speeds using monitor initialization features.

The DC76 will support asynchronous terminals which may be 8-level ASCII terminals or 7-level terminals compatible with the IBM 2741, using EBCDIC, APL or correspondence character sets. ASCII terminals compatible with DIGITAL LT33, LT35, LA30, LA36, VT05, VT06, and VT50 are fully supported. ASR units must include X-ON/X-OFF paper tape reader control for full support; the support of cursor functions on displays is a function of the user program.

The DC76 does not support the "reader-run" control feature found on some terminals (LT33-D, LT35-D). ASCII terminals should be 8-unit asynchronous code with one stop unit, except at 110 baud where two units are assumed. The eighth data bit is transmitted to the terminal as even-character parity, but is ignored on reception. Five- and six-level codes are accommodated by the hardware (except six-level code with 1-1/2 stop bits) but are not supported by standard software.

The DM11-BB 16-line interface group, in conjunction with the DC76 software, will support any modem with EIA-RS232/CCITT V.24-signal interface and operating characteristics (control level sequencing) compatible with Bell System 103A or 103E modems, providing that the modem at the DC76 end of the circuit is end-to-end compatible with the modem at the user terminal end of the circuit.

The DC76 software supports full-duplex (two-way simultaneous) line operation. Normal operating mode consists of echoing most characters (certain control characters are echoed by the monitor). Full-duplex with local copy is also supported. Two-way alternate simplex (the usual mode of Bell 202 modems) is not supported. However, the "almost two-way" alternate operation of 2741 and 2741-compatible terminals is supported over full-duplex lines. Polled operation (using Bell 103F-type modems) is not supported with either full-duplex or two-way alternate simplex modems. The "make-busy" feature provided by Bell 103E, 113B, and other similar modems is not supported.

Due to the large and ever increasing number of data sets available, it is not practical to list those which will work satisfactorily with the DC76. The following data sets are among those supported by the DM11-BB interface group:

1. Bell System 103A-type
2. Bell System 103E-type data station
3. Bell System 103F-type (but not polled or multipoint operation)
4. Bell System 113B-type data station
5. Bell System 202 full-duplex (4-wire) mode only
6. DIGITAL DF11-BB-type full-duplex, receive-only integral modem.

The basic DC76 is capable of supporting up to three additional PDP-11 multiplexer controllers which may be any combination of DC75-NP and DC76 subsystems.

4.8.3 DC75-NP Synchronous Communications Subsystem

The DC75-NP synchronous communications subsystem (Figure 4-1) provides a highly reliable, high-speed path between the central DECsystem-10 computer and DC72, DAS80-series or DAS92 remote stations, remote terminals, and/or other computer systems. Transmission over high-speed synchronous lines is on a message basis using DDCMP communications protocol.

The basic DC75-NP subsystem consists of a high-speed interface to the DECsystem-10 memory bus (DL10), a PDP-11 processor which serves as a multiplexer controller, and a multiple line synchronous multiplexer (DS11). The PDP-11 multiplexer controller packs and unpacks characters directly into the DECsystem-10 memory (via the DL10) and can execute instructions from the DECsystem-10 memory for bootstrap operations. The DS11 multiple line synchronous interface handles 8 full-duplex lines and can be expanded to handle up to 16 full-duplex lines. Standard system software supplied with the DC75-NP operates in full-duplex mode using the DDCMP communications protocol for communications with DC72, DAS80-series, or DAS92 remote stations. Up to four lines are supported by this software.

The DC75-NP is recommended for applications involving up to 64 synchronous lines to DC72-NP, DAS80-series, or DAS92 remote stations. Each multiplexer controller handles a total line speed of up to 40.8K bits/second including error checking, formatting, and line control. Individual lines may operate at speeds up to 9600 bits/second.

The basic DC75-NP subsystem is capable of supporting up to three additional PDP-11 multiplexer controllers, which may be any combination of DC75-NP, DC76, or DN87 subsystems. Compatible modems include the Bell System 201, Bell System 203, Bell System 208, Rixon PM24, ICC Modem 2200, or any synchronous modem which conforms to the Electronics Industries Association RS-232B or C Computer Interface Standards as well as CCITT V.24 (white book) standards.

4.8.4 DN87/DN87S Universal Synchronous/Asynchronous Front-End Subsystems

The latest DECsystem-10 data communications products are based on the DN87 universal synchronous/asynchronous communications front-end subsystem. The DN87 is configurable in asynchronous-only mode (up to 112 asynchronous lines), synchronous-only mode (up to 12 synchronous lines), or with a mixture of synchronous and asynchronous lines in the same DN87. These three modes allow the DN87 to be configured very cost effectively in a wide variety of customer specific configurations. This great increase in flexibility without any significant increase in price is the essence of the DN87 communications subsystem.

The DN87 is capable of communicating with the DAS80 series remote stations, the DAS92 remote station, and the DC72-NP remote station. These allow remote job entry, remote concentration of interactive Teletype® lines, or a combination of the two. The standard DN87/DAS80 series remotes support complex topologies such as multipathing, route-thru, and multiple host support. With DECnet-10, it is possible to have a DECsystem-10 communicate with any DECnet system – for example RSX-11M, RSX-11D, RSX-11S, IAS, RT-11, RSTS/E, RTS-8, DECsystem-10.

The DN87S has the same functionality as a DN87 except that it is attached to the DECsystem-10 via the DTE10 interface rather than the DL10. Up to three DN87Ss may be attached to the KL10-based DECsystem-10 via the DTE10 interfaces. The DN87 and the DN87S require TOPS-10 version 6.03.

4.8.4.1 Asynchronous Interfaces – On an 8-line group basis, the DN87/DN87S is capable of terminating 20 mA current loop, local EIA, or EIA with full modem control type of asynchronous line/terminal interfaces. On an individual line basis, the DN87/DN87S asynchronous lines can be:

1. ASCII Teletype-compatible code or 2741 EBCDIC or correspondence code
2. Full-duplex with echoplex (i.e., echo generated by computer) or full-duplex with local copy (simulated half-duplex)
3. Program-selectable line speeds of from 50 through 9600 baud
4. Split transmit/receive speeds
5. Automatically baud rate detected for 110, 134.5, 150, and 300 baud lines.

Some off-loading of the DECsystem-10 host is accomplished, in that the DN87/DN87S does the majority of the echoing for asynchronous lines. It does not echo special characters, nor does it echo when the user is in character-at-a-time mode (e.g., DDT – when an individual character can be a command). Also the DN87/DN87S does fill character generation.

® Teletype is a Registered trademark of the Teletype Corporation.

4.8.4.2 Synchronous Interfaces – The DN87/DN87S is capable of terminating EIA and/or current-loop type synchronous links. The line speeds may be 2400, 4800, 7200, 9600, 19.2K, 38.4K, or 40.8K baud on an individual line basis. These links operate only in full-duplex with simultaneous bidirectional transmission. The synchronous links use DDCMP protocol for error checking and correction and for point-to-point link control.

These synchronous links communicate only with the DC72-NP remote station (DC72-NP is a software-only upgrade of DC71 or DC72), the DAS80 series remote stations, the DAS92 remote station, or another DN87/DN87S.

4.8.5 Communication Subsystem Components

A PDP-11 processor (minicomputer) is used in the asynchronous and synchronous communication subsystems as a communications multiplexer and character handler. The basic components used in these subsystems are for the most part the same as those used in the console processor subsystem. (Refer to Subsection 4.3.1.) The basic components include:

1. KC11 Central Processor (DC75-NP)
2. KF11-A 4-Level Auto Priority Interrupt Option (DC75-NP)
3. KD11-A Central Processor (DC76, DN87, and DN87S)
4. KY11-D Programmer's Console
5. KW11-L Line Frequency Clock
6. MF11-UP Memory (DC76, DN87, and DN87S)
7. MM11-UP Memory (DN87S)
8. MM11-F Memory (DC75-NP)
9. KT11-D Segmentation Option (DN87S)
10. DL11-C/E Asynchronous Line Interface
11. BC11-A Unibus

In addition to these standard PDP-11 devices, the following devices are also used in the communication subsystems. They are:

1. KG11-A XOR and CRC unit (DC75-NP, DN87, and DN87S)
2. DS11 Multiple Line Synchronous Interface (DC75-NP)
3. DH11 Asynchronous 16-Line Multiplexer (DC76, DN87, and DN87S)
4. DM11-BB Modem Control Option (DC76, DN87, and DN87S)
5. DQ11 Double Buffered Synchronous Line Control (DN87 and DN87S)

The following paragraphs provide a brief description of these devices.

4.8.5.1 KG11-A XOR and CRC Unit (DC75-NP, DN87, and DN87S) – The KG11-A is used to compute a cyclic redundancy check (CRC) or longitudinal redundancy check (LRC) for detecting errors in serially transmitted data. It is used with a DS11 serial synchronous line interface to compute the block check character(s) (BCC) appearing at the end of a block of data transmitted over a serial synchronous line.

For received data, the characters are moved to the KG11-A and a BCC is computed for the data and compared to the BCC received. If they are equal, the data is assumed to be correct and is accepted. If they do not match, the message is not accepted and the data is retransmitted.

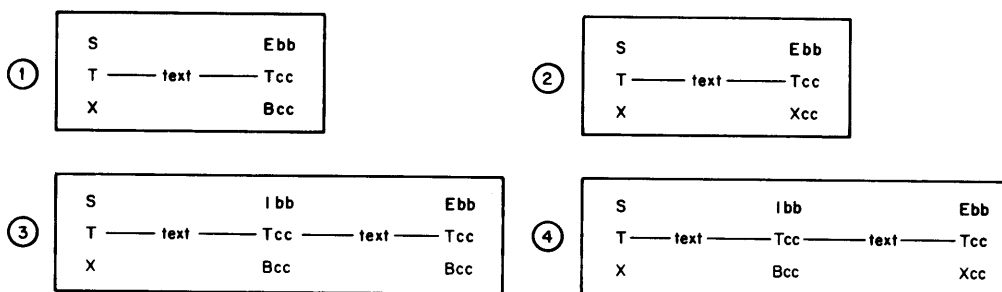
When data is being transmitted, the BCC is generated by moving all the characters to the KG11-A. The resulting BCC is transmitted at the end of the message.

Not all characters are included in the BCC. The exclusions will depend on the line protocol used.

The KG11-A, under program control, can compute the most popular CRC and LRC polynomials. They are:

1. CRC-16 $X^{16} + X^{15} + X^2 + 1$
2. CRC-12 $X^{12} + X^{11} + X^3 + X^2 + X + 1$
3. CRC-CCITT $X^{16} + X^{12} + X^5 + 1$
4. LRC-8 $X^8 + 1$
5. LRC-16 $X^{16} + 1$

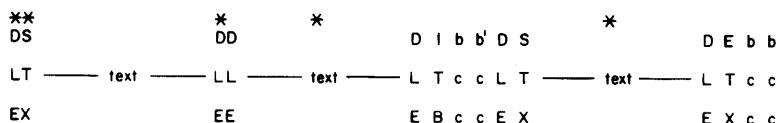
CRC-16 is used for synchronous systems that employ 8-bit characters. It is used in IBM binary synchronous systems when the transmission code is EBCDIC or 8-bit transparency. For IBM-compatible systems, the message format is shown in Figure 4-6.



10-1922

Figure 4-6 IBM Compatible Message Format

In the four examples shown in Figure 4-6, each character represents an 8-bit character. The first BCC character is the least significant 8 bits of the BCC computed in the KG11-A. The STX is not included in the BCC. The BCC includes the first text character through the ETB, ITB, or ETX. In examples 3 and 4, the second BCC begins with the character following the first BCC in the block (even if it is an STX or DLE). The examples are for normal transmission. For transparent transmission, the characters indicated by (*) in Figure 4-7 are not included in the BCC.



10-1923

Figure 4-7 BCC Computation for Transparent Transmission

The DLE DLE sequence indicates that the second DLE is really data and not the control character and is, therefore, included in the BCC. It may appear in text as often as that 8-bit representation is required. Because the DLE ITB sequence takes the system out of the transparent mode, the DLE STX sequence following the BCC is included in the next BCC and also puts the system back into the transparent mode.

CRC-12 is used for 6-bit characters. It is compatible with IBM binary synchronous communications (BSC) when the transmission code used is 6-bit transcode. The characters included in or excluded from the BCC are the same as for CRC-16. The difference is only in the length of character (6 versus 8 bits).

CRC-CCITT is the standard polynomial used to compute BCC for European systems. The characters included or excluded will depend on the line protocol used for the system in which the KG11-A is used.

Some systems use only an 8-bit LRC on the characters. LRC-8 performs an exclusive OR on an 8-bit or less character. The LRC is usually used in combination with a vertical redundancy check (VRC). VRC is possible only when the characters are 7 bits or less plus one parity bit. LRC/VRC is used for IBM BSC when the transmission code is ASCII. For IBM systems, the parity bit makes the character contain an odd number of bits.

LRC-16 performs an exclusive OR on a 16-bit or less character. It can be used to perform a word exclusive OR, or to compute LRC for 12-bit characters transmitted via the DS11.

4.8.5.2 DS11 Multiple Line Synchronous Interface (DC75-NP) – The DS11 is a multiplexer interface for up to 16 full- or half-duplex communication lines (modems) that operate in the synchronous mode. The 16 modems can be a Bell 201, 203, or equivalent; they must meet the specifications set forth in Electronic Industries Association (EIA) Standard RS-232B or C. In addition, a Bell 303 data set or equivalent may be used.

Three configurations of the DS11 are available:

1. DS11-A: Consists of the DS11 logic only; no line control modules
2. DS11-BA: Consists of one M7110 line control module, and corresponding cable for a Bell 201, 203, or equivalent
3. DS11-BB: Consists of one M7110 line control module, and corresponding cable for a Bell 303 or equivalent.

The basic function of this interface is to perform serial-to-parallel and parallel-to-serial data transfers. Parallel transfers to and from the DS11 interface transpire under program control; serial transfers between the DS11 and the modems are under DS11 control.

The maximum transfer rate between the present PDP-11 and a DS11 is 10,000 characters per second (6, 8, or 12 bits per character). Up to 16 full-duplex lines can be connected to a single DS11, but their total transfer rate cannot exceed 100 μ s per character.

4.8.5.3 DH11 Asynchronous 16-Line Multiplexer (DC76, DN87, and DN87S) – The DH11 asynchronous 16-line programmable multiplexer connects the PDP-11 with 16 asynchronous serial communications lines operating with individually programmable parameters. These parameters are:

1. Character Length: 5, 6, 7, or 8 bits.
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 DH11 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 which is addressable from the Unibus.

The transmitter in the DH11 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 DH11. This reduces the Unibus time taken for the NPR transfers to one NPR cycle per character transmitted. The NPR cycle used is lengthened slightly.

4.8.5.4 DM11-BB Modem Control Unit (DC76, DN87, and DN87S) – The DM11-BB modem control unit is used with the DH11 for modem control in dedicated and switched networks. Data is handled by the DH11 asynchronous 16-line multiplexer.

The DM11-BB is a multiplexed modem control for 16 asynchronous modem interfaces. The unit provides necessary control signals and levels to interface with Bell 103A/E/F/G/H, 202C/D, and 811B modems or their equivalent. The interface levels are EIA/CCITT-compatible for data set operation. The DM11-BB is ideally suited for applications where data is collected at remote locations and forwarded to a controlling processor. Typical applications include numerical control, data acquisition, physics, biomedical, time-sharing systems, and networks.

The modem control signals for up to 16 modems are connected to the DM11-BB through the DM11-AA distribution panel, which provides level conversion for all lines interfaced. Line adaption is achieved at the distribution panel through the DM11-DC option. The DM11-DC unit is required for modem control interfacing. Each DM11-DC implements four EIA/CCITT lines including cabling for modem interfacing. Four DM11-DCs are necessary to handle the maximum 16 modems. Each of themodem types that can be interfaced may be mixed over the 16 available lines.

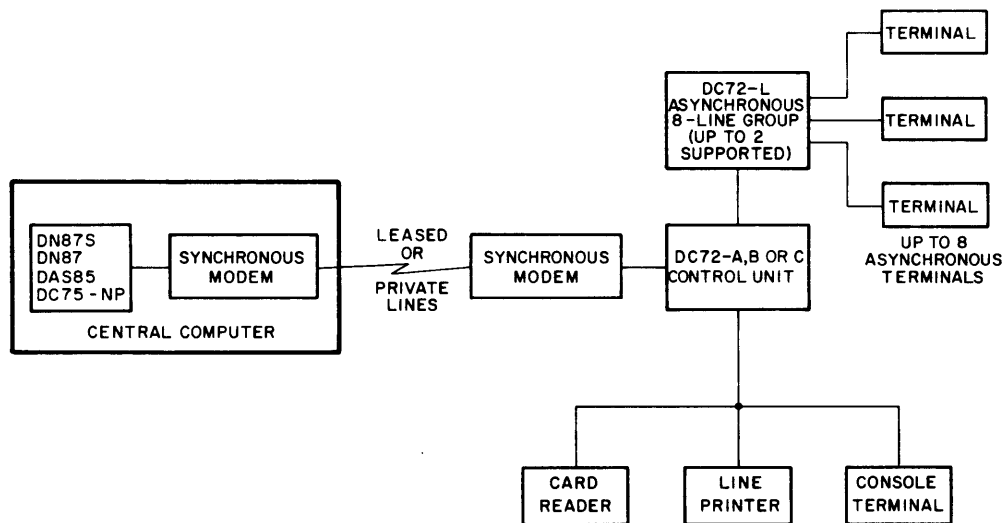
The DM11-BB scans the SEC RX, CLEAR TO SEND, CARRIER, and RING lines for each modem line sequenced by a line counter in the logic. When a transition is detected on a line, for the modem line designated by the line counter, an interrupt condition is generated. Providing interrupt enable and line enable are programmed, the interrupt requests bus control through the interrupt control logic. Likewise, through the address selector logic, the processor sends SEC TX, CLEAR TO SEND, and TERMINAL READY to the modem designated by the line counter. The line counter enables the particular transmit signal to be asserted on the line designated. The line counter is sequenced through the ring counter, which is clocked internally (scan logic) and enabled by the program-controlled scan enable and step conditions.

4.8.5.5 DQ11 Synchronous Line Interface (DN87 and DN87S) – The DQ11 is a high-speed, double-buffered communications device designed to interface the PDP-11 processor to a serial-synchronous communications channel. This interface allows the PDP-11 to be used for remote batch and remote concentrator applications. With the DQ11, the PDP-11 can also be used as a front-end, synchronous-line controller to handle remote and local synchronous terminals.

Transmit and receive data transfers between the PDP-11 Unibus and the DQ11 are handled as non-processor requests (NPR). These are direct memory or device access data transfers without processor supervision. As an NPR device, the DQ11 provides extremely fast access to the PDP-11 Unibus and can transfer data at exceptionally high rates once it gains control. The PDP-11 processor state is not affected by NPR transfers, since they occur on a cycle-steal basis.

4.8.6 DC72-NP Remote Station

The DC72-NP remote station and options (Figure 4-8) make DECsystem-10 peripherals available to any remote site that can be connected to a DECsystem-10 by a leased synchronous communication line. The remote peripherals supported include 110- to 2400-baud asynchronous ASCII terminals (e.g., the LA30 DECwriter, VT05 video display terminal, RT01 and RT02 DElink data entry terminals), line printers, and card readers. These remote peripherals are functionally equivalent to their local counterparts.



10-1924

Figure 4-8 DC72-NP Remote Station

The DC72-NP remote station which is a PDP-8-based hardware/software subsystem provides remote users of a DECsystem-10 with a full set of user-oriented input/output peripherals at prices comparable to a conventional remote job entry (RJE) terminal. In addition to the RJE capability, the DC72 provides an interactive terminal for operator control, dedicated applications, and general interactive use. The DC72-A, B, and C basic stations include a 10-character/second operator console, a 300-card/minute, reader, and a 132-column line printer.

The DC72-A features a 165-character/second, 64-character-set printer which gives high-quality dot matrix printing and a 2-channel vertical-format control.

The DC72-B has a faster drum printer with a 64-character set. The speed of this printer is 300 132-column lines per minute. The vertical format unit is a single-channel unit switch-selectable for page sizes of 3, 3-1/2, 4, 5-1/2, 6, 7, 8, 8-1/2, 11, 12, and 14 inches.

The DC72-C is similar to the DC72-B but offers a 96-character set line printer that includes lowercase characters and additional symbols; the printing speed is 230 132-column lines/minute.

The DC72-L asynchronous 8-line group is used to add eight asynchronous terminal interfaces to any of the DC72 series. The DC72-L enables line speed to be individually selected for each line. Terminal output rates of 110 to 2400 baud and transmission rates of 110 to 300 baud are supported; 134.5-baud terminals of the IBM 2741 types are not supported on the DC72-NP. User programs which use cursor movement commands must supply fill characters at 600 baud and above. Direct cursor addressing is not possible through DC72 remote stations above 300 baud.

Two DC72-Ls (16 lines maximum) can be added to each DC72-A, B, or C. Terminals may be connected to the DC72-L either locally using 20 mA current loops or through EIA RS-232C or CCITT V.24 (white book) standard asynchronous modems. Modem control is not available on the DC72-L, however.

The supporting software for the DC72-NP is an evolution of the DC71 software first delivered in 1971. It takes full advantage of full-duplex communication with the DECsystem-10 to run both the reader and line printer simultaneously, while at the same time providing good interactive response for up to 16 asynchronous terminals.

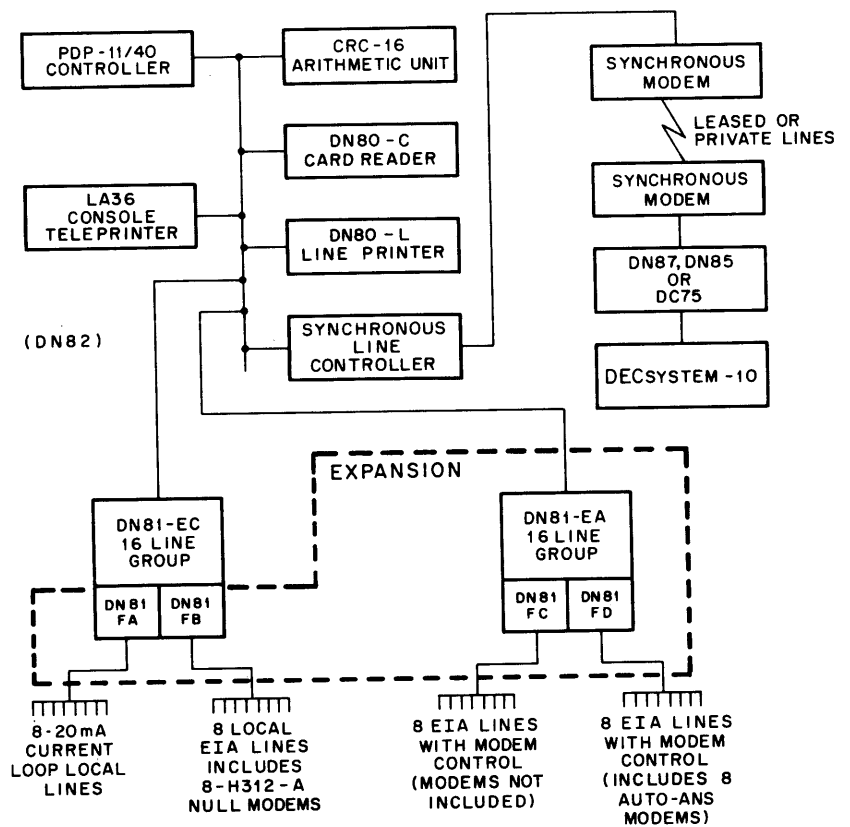
For communication with the DC75-NP, the DC72-NP uses DDCMP communication protocol. Compatible modems include the Bell System 201, Bell System 203, Bell System 208, Rixon PM24, ICC Modem 2200, or any synchronous modem which conforms to the Electronics Industries Association RS-232B or C Computer Interface Standards as well as CCITT V.24 (white book) standards.

4.8.7 DN80 (DAS80) -Series Remote Station

The DN80-series remote stations (Figure 4-9) are PDP-11-based stations designed to provide the remote user with access to the full capabilities of the DECsystem-10, as well as the capability to expand from a small remote batch station (DN80) or a small remote concentrator for interactive asynchronous terminals (DN81), to a fully expanded remote station (DN82).

The DN80-series remote stations provide a modular approach to the support of remote peripherals and asynchronous terminals for the DECsystem-10. The DN80-A remote batch station provides card reader and line printer support to any site that can be connected to the DECsystem-10 via dedicated, full-duplex synchronous communications channels. A basic batch station configuration (DN80-A) includes a 300-card/minute card reader, 300-line/minute, 64-EDP-character printer, and a single synchronous line controller. The basic remote concentrator (DN81-A) supports up to 16 asynchronous terminals such as the LA36, VT50, VT52, or similar devices. All basic DN80-series stations include a local LA36 DECwriter II that can be used as an interactive terminal for job submission and control or for diagnostic/maintenance support.

The DN81 remote concentrator provides concentration for up to 32 interactive asynchronous terminals to a DECsystem-10 host processor. It enables a significant savings in line use costs by using a single-leased, private, full-duplex synchronous communication line to a DECsystem-10 instead of one asynchronous line per terminal. User terminals can employ either local or remote asynchronous communications lines to connect to the DN81 concentrator, which in turn are connected to the remote DECsystem-10 with a single high-speed synchronous communications line. Provisions to connect the DN81 to both a primary or alternate DECsystem-10 host processor can be added as options.



10-2000

Figure 4-9 DN80-Series Remote Station

A basic DN81 includes one 16-line asynchronous control group (a maximum of two are supported). Lines are activated in groups of eight using either the DN81-FA, FB, FC, or FD line groups. The DN81 hardware and software support of asynchronous terminals is equivalent to that provided by the DC76 local asynchronous concentrator, except that total throughput is limited to 960 characters/second.

A basic DN82 remote station combines, in a single assembly, the functions of a DN80 remote batch station and a DN81 remote concentrator. It can be field installed by adding the appropriate equipment options to a DN80-A or DN81-A.

All of the DN80 series remote stations use the new DDCMP synchronous communications protocol to support full-duplex communications with the DECSYSTEM-10 via a leased voice grade channel or private 4-wire synchronous lines. DDCMP provides a highly efficient, full-duplex, simultaneous bidirectional transmission of data to make optimum use of available communications-line bandwidth. DDCMP uses a high reliability CRC-16 hardware-implemented feature for automatic error checking and an efficient method for message retransmission in case of line errors.

Connection between a DN80-series remote station and a DECsystem-10 primary or alternate host computer system requires that the host processor(s) be equipped with a DC75-NP, DN85, DN87, or DN87S synchronous communications front-end system. Older models of DECsystem-10 host processors may be equipped with DC75 synchronous communications systems that have been upgraded with a DC75-NP software/hardware option. The maximum synchronous line speed that can be supported by a DN80 series station is 9600 bits per second. Additional equipment facilities needed to complete the connection between the DECsystem-10 host processor and the DN80 series remote station(s) must include either a leased full-duplex data transmission facility (supplied by telecommunication common carriers) or customer-owned or -leased modems and compatible transmission facilities. If the DN80 series remote station is within 15 meters (50 feet) of the host DECsystem-10 communications front-end, a DN83-A synchronous null modem may be used in place of the common carrier or customer-owned data transmission facility.

Typical compatible synchronous modems that can be used include the Bell System 201, 208, 209, or equivalent ICC modems, or any synchronous modem whose terminal interface conforms to the Electronics Industries Association RS-232B or C (Computer Interface Standards) as well as CCITT (Vol. VIII, Green Book, V.24) standards.

4.8.8 DAS92 Remote Station

The DAS92 remote station (Figure 4-10) is a low-cost, PDP-8/A-based remote station designed to provide the remote user with access to the full capabilities of the DECsystem-10 while offering the flexibility of remote terminal concentration and remote batch functionality in the same basic unit.

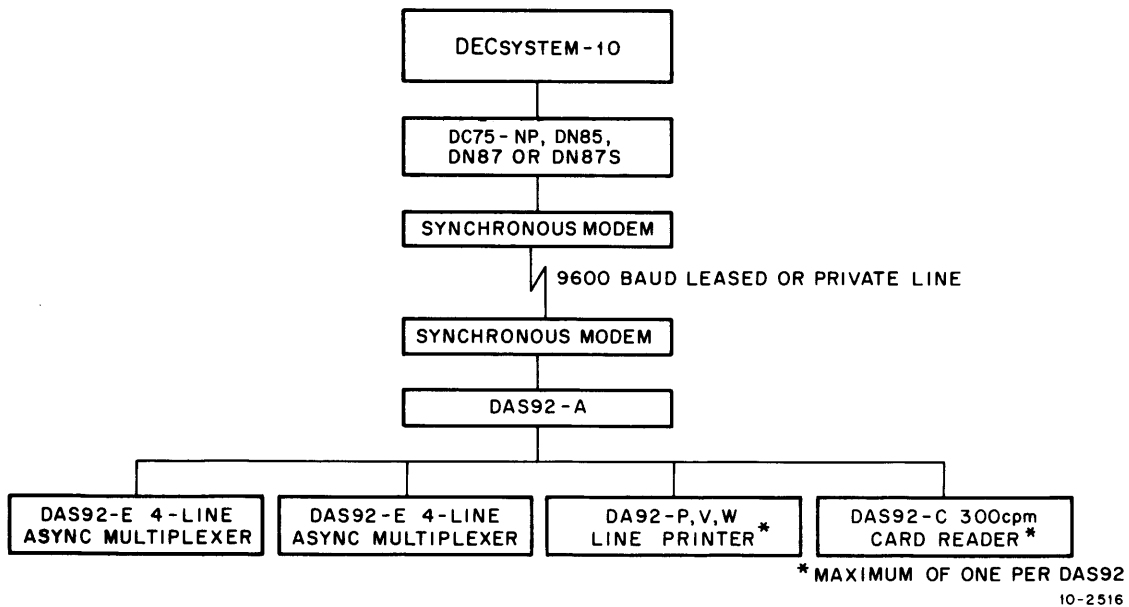


Figure 4-10 DAS92 Remote Station (Typical Configuration)

The DAS92 combines in a single assembly the basic unit and expansion capability. Typical configurations include a 4-, 8-, 12-, or 16-line concentrator; a remote batch station with a card reader and line printer; a remote station with up to eight asynchronous lines *and* a card reader and line printer; or a remote station with up to 12 lines of concentration and a card reader *or* line printer.

The DAS92 remote concentrator configuration provides concentration for up to 16 asynchronous terminals for the DECsystem-10. It enables a significant savings in line cost by using a single leased or private full-duplex synchronous communication line to the DECsystem-10 instead of one asynchronous line per terminal. A 9600-baud synchronous line is required to attain desirable throughput performance. User terminals can employ either local or remote communication lines to connect to the DAS92. One line out of each four-line group offers full modem control capability. The other three lines should be used on leased or private lines for remote terminal applications. If dial-up through a switched network is required, these lines must be connected through two Bell 103A or equivalent modems, with the "long space disconnect" option installed. On these lines it is not possible to run Bell 202 type modems in half-duplex or switched networks, but Bell type 202 modems or their equivalent may be run full-duplex on leased lines. Each DAS92 asynchronous line group has complete flexibility for interfacing to EIA or local (20 mA) lines. Both plug types are provided for each line and switch settings enable the appropriate drivers.

The DAS92 remote station is housed in an attractive walnut-topped desk and comes with a VT52 operator/diagnostic console and desk space for an optional card reader. The desk is mounted on casters for transportability and has a pull-down rear door for easy access to distribution panels. Each DAS92 has 16K of core memory, a ROM for downline loading of system software and diagnostics, power fail and auto restart features, a programmer's console, and a synchronous line controller that can interface to Bell 200 series synchronous modems or their equivalents.

The DAS92 provides an effective throughput of approximately 750 characters/second when communicating over a 9600 baud synchronous line.

SECTION 5 THE SOFTWARE

5.1 RESIDENT OPERATING SYSTEM

The resident operating system (DECsystem-10 total operating system – TOPS-10) is made up of a number of separate and somewhat independent parts, or routines (Figure 5-1). Some of these routines are cyclic in nature and are repeated at every system clock interrupt (tick) to ensure that every user of that computing system is receiving his requested services. These cyclic routines are:

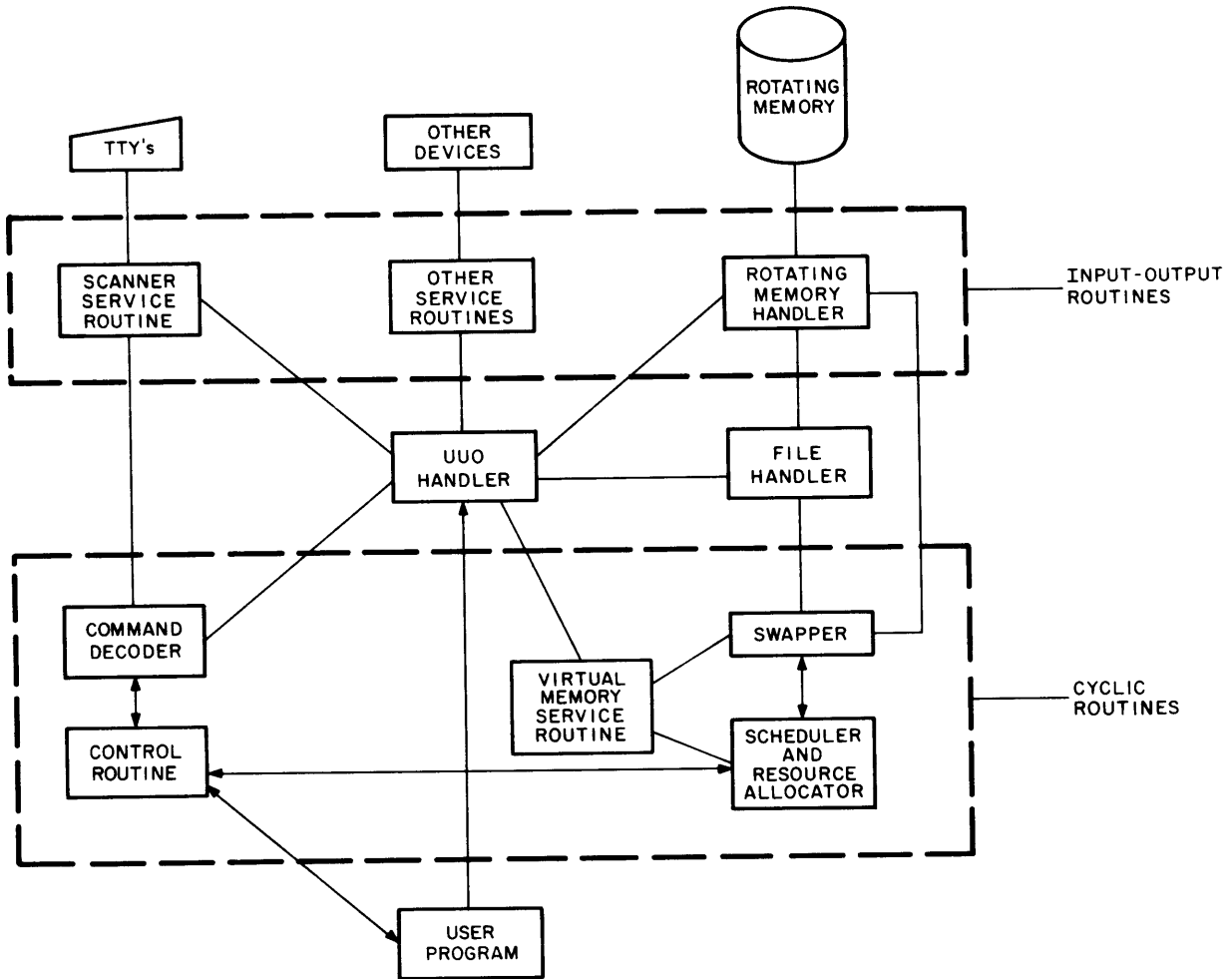
1. The command processor, or decoder
2. The scheduler
3. The swapper
4. The control routine.

The command decoder is responsible for interpreting commands typed by the user on his terminal and passing them to the appropriate system program or routine. The scheduler decides which user is to run in the interval between the clock interrupts, allocates sharable system resources, and saves and restores conditions needed to start a program interrupt by the clock. The swapper rotates user jobs between secondary memory (usually disk or drum) and core memory after deciding which jobs should be in core but are not. These routines constitute the part of the operating system that allows many jobs to be operating simultaneously.

The noncyclic routines of the operating system are evoked only by user programs and are responsible for providing these programs with the services available through the operating system. These routines are:

1. The UVO handler
2. The input/output routines
3. The file handler.

The UVO handler is the means by which the user program communicates with the operating system in order to have a service performed. Communication is by way of programmed operators (also known as UVOs) contained in the user program which, when executed, go to the operating system for processing. The input/output routines are the routines responsible for directing data transfers between peripheral devices and user programs in core memory. These routines are evoked through the UVO handler, thus saving the user the detailed programming needed to control peripheral devices. The file handler adds permanent user storage to the computing system by allowing users to store named programs and data as files.



10-0821

Figure 5-1 The Resident Operating System

5.1.1 Command Decoder

The command decoder is the communications link between the user at his terminal and the operating system. Because all the requests for system resources are initiated via the command decoder, it is the most visible part of the system to each user. When the user types commands and/or requests on his terminal, the characters are stored in an input buffer in the operating system. The command decoder examines these characters in the buffer, checks them for correct syntax, and evokes the system program as specified by the command.

On each clock interrupt, control is given to the command decoder to interpret and process one command in the input buffer. The command appearing in the input buffer is matched with the table of valid commands accepted by the operating system. A match occurs if the command typed in exactly matches a command stored in the system, or if the characters typed in match the beginning characters of only one command. When the match is successful, the legality information (or flags) associated with the command is checked to see if the command can be performed immediately. For instance, a command can be delayed if the job is swapped out to the disk and the command requires that the job be resident in core; the command is executed on a later clock interrupt when the job is back in core. If all conditions as specified by the legality flags are met, control is passed to the appropriate program.

5.1.2 Scheduler

The DECsystem-10 is a multiprogramming system, i.e., it allows several user jobs to reside in core simultaneously and to operate sequentially. It is then the job of the scheduler to decide which jobs should run at any given time. In addition to the multiprogramming feature, the DECsystem-10 employs a swapping technique whereby jobs can exist on an external storage device (e.g., disk or drum) as well as in core. Therefore, the scheduler decides not only what job is to be run next but also when a job is to be swapped out onto the disk or drum and later brought back into core.

All jobs in the system are retained in ordered groupings called queues. These queues have various priorities that reflect the status of each job at any given moment. The queue in which a job is placed depends on the system resource for which it is waiting and, because a job can wait for only one resource at a time, it can be in only one queue at a time. Several of the possible queues in the system are:

1. Run queues for jobs waiting for, or jobs in execution
2. I/O wait queues for jobs waiting for data transfers to be completed
3. I/O wait satisfied queues for jobs waiting to run after data transfers have been completed
4. Resource wait queues for jobs waiting for some system resource
5. Null queue for all numbers that are not currently being used.

The job's position within certain queues determines the priority of the job with respect to other jobs in the same queue. For example, if a job is first in the queue for a sharable device, it has the highest priority for the device when it becomes available. However, if a job is in an I/O wait queue, it remains in the queue until the I/O is completed. Therefore, in an I/O wait queue, the job's position has no significance. The status of a job is changed each time it is placed into a different queue.

The scheduling of jobs into different queues is governed by the system clock. This clock divides the time for the central processor into sixtieths of a second. When the clock ticks, the scheduler decides which job will run during the next cycle. If the currently running job is the null job and a higher priority job (any job) becomes ready to run, it will immediately run. If the currently running job is not the null job and a high priority (HPO real-time) job becomes runnable, it will run at the next clock tick. Finally if a job just becoming runnable is not an HPQ job but is of higher priority than the current job, it will only preempt the current job when the applicable quantum run time has expired.

Each job, when it is assigned to run, is given a time slice. When the time slice expires for the job, the clock notifies the central processor and scheduling is performed. The job whose time slice just expired is moved into another run queue, and the scheduler selects another job to run in the next time slice.

Scheduling may be forced before the time slice has expired if the currently running job reaches a point at which it cannot immediately continue. Whenever an operating system routine discovers that it cannot complete a function requested by the job (e.g., it is waiting for I/O to complete or the job needs a device which it currently does not have), it calls the scheduler so that another job can be selected to run. The job that was stopped is then requested and is scheduled to be run when the function it requested can be completed. For example, when the currently running job begins input from a disk, it is placed into the I/O wait queue, and the input is begun. A second job is scheduled to run while the input of the first job proceeds. If the second job then decides to access a disk, it is stopped because the disk control is busy, and it is placed in the queue for jobs waiting to access the disk control. A third job is set to run. The input operation of the first job finishes, freeing the disk control for the second job. The I/O operation of the second job is initiated, and the job is transferred from the device wait queue to the I/O wait queue. The first job is transferred from the I/O wait queue to the highest priority run queue. This permits the first job to preempt the running of the third job. When the time slice of the first job becomes zero, it is moved into the second run queue, and the third job runs again until the second job completes its I/O operation.

In addition, data transfers use the scheduler to permit the user to overlap computation with data transmission. In unbuffered data modes, the user supplies an address of a command list containing pointers to locations in his area to and from which data is to be transferred. When the transfer is initiated, the job is scheduled into an I/O wait queue where it remains until the device signals the scheduler that the entire transfer has been completed.

In buffered modes, each buffer contains information to prevent the user and the device from using the same buffer at the same time. If the user requires the buffer currently being used by the device as his next buffer, the user's job is scheduled into an I/O wait queue. When the device finishes using the buffer, the device calls the scheduler to reactivate the job.

5.1.3 Swapper

The swapper is responsible for keeping in core the jobs most likely to be runnable. It determines if a job should be in core by scanning the various queues in which a job may be. If the swapper decides that a job should be brought into core, it may have to take another job already in core and transfer it to secondary memory. Therefore, the swapper is not only responsible for bringing a job into core but is also responsible for selecting the job to be swapped out.

A job is swapped into secondary memory for one of two reasons:

1. A job that is more eligible to run needs to be swapped in and there is not enough room in core for both jobs
2. The job needs to expand its core size and there is not enough core space to do so.

If the latter case is true, the job must be swapped out and then swapped in later with the new allocation of core.

The swapper checks periodically to see if a job should be swapped in. If there is no such job, then it checks to see if a job is requesting more core. If there is no job wishing to expand its size, then the swapper does nothing further and relinquishes control of the processor until the next clock tick.

5.1.4 Control Routine

The control routine (labeled CLOCK1) calls for execution of major system modules to ensure that every user of the computing system gets needed services. CLOCK1 is run every time the real time clock interrupts the system (50 or 60 times per second which is dependent on the power line frequency). Some of the responsibilities that are typical of CLOCK1 include:

1. Saving the state of the current job
2. Time accounting
3. Processing of timing requests
4. Context switching
5. Start-up of next user
6. Reinstatement of current user
7. Call to other cyclical control routines.

5.1.5 UUO Handler

The UUO handler is responsible for accepting requests for services available through the operating system. These requests are made by the user program via software-implemented instructions known as programmed operators, or UUOs. The various services obtainable by the user program include:

1. Communicating with the I/O devices on the computing system, including connecting and responding to any special devices that may be desired on the system for real-time programming.
2. Receiving or changing information concerning either the computing system as a whole or the individual program.
3. Altering the operation of the computing system as it concerns the user job, such as controlling execution by trapping or suspending, or controlling core memory by locking.
4. Communicating and transferring control between user programs.

The UUO handler is the only means by which a user program can give control to the operating system in order to have a service performed. Contained in the user program are operation codes which, when executed, cause the hardware to transfer control to the UUO handler for processing. This routine obtains its arguments from the user program. The core location at which the UUO operation was executed is then remembered. After the UUO request has been processed, control is returned to the user program at the first or second instruction following the UUO. In this way, the software supplements the hardware by providing services that are evoked through the execution of a single core location just as the hardware services are evoked.

5.1.6 Device Service Routines

I/O programming in the DECsystem-10 is highly convenient for the user because all of the burdensome details of programming are performed by the operating system. The user informs the operating system of his requirements for I/O by means of UUOs contained in his program. The actual device service routines needed are then called by the UUO handler.

Since the operating system channels communication between the user program and the device, the user does not need to know all the peculiarities of each device on the system. In fact, the user program can be written in a similar manner for all devices. The operating system will ignore, without returning an error message, operations that are not pertinent to the device being used. Thus, a terminal file and a disk file can be processed identically by the user program. In addition, user programs can be written to be independent of any particular device. The operating system allows the user program to specify a logical device name, which can be associated with any physical device at the time the program is to be executed. Because of this feature, a program that is coded to use a specific device does not need to be rewritten if the device is unavailable. The device can be designated as a logical device name and assigned to an available physical device with one command to the operating system.

Data is transmitted between the device and the user program in one of two methods: unbuffered mode or buffered mode. With unbuffered data modes the user in his program supplies the device with an address, which is the beginning of a command list. Essentially, this command list contains pointers specifying areas to the user's allocated core to or from which data is to be transferred. The user program then waits until the operating system signals that the entire command list has been processed. Therefore, during the data transfer, the user program is idly waiting for the transfer to be completed.

Data transfers in buffered mode utilize a ring of buffers set up in the user's core area. Buffered transfers allow the user program and the operating system's I/O routines to operate asynchronously. As the user program uses one buffer, the operating system processes another one by filling or emptying it as interrupts occur from the device. To prevent the user program and the operating system from using the same buffer at the same time, each buffer has a use bit that designates who is using the buffer. Buffered data transfers are faster than unbuffered transfers because the user program and the operating system can be working together in processing the data.

Several steps must be followed by the user program in order for the operating system to have the information it needs to control that data transfer. Each step is indicated to the operating system with one programmed operator. In the first step, the specific device to be used in the data transfer must be selected and linked to the user program with one of the software I/O channels available to the user's job (OPEN or INIT programmed operators). This device remains associated with the software I/O channel until it is disassociated from it (via a programmed operator) or a second device is associated with the same channel. In addition to specifying the I/O channel and the device name, the user program can supply an initial file status, which includes the type of data transfer to be used with the device (e.g., ASCII, binary), and the location of the headers to be used in buffered data transfers. The operating system stores information in these headers when the user program executes programmed operators, and the user program obtains from these headers all the information needed to fill or empty buffers.

Another set of programmed operators (INBUF and OUTBUF) establishes the actual buffers to be used for input and output. This procedure is not necessary if the user is satisfied to accept the two buffers automatically set up for him by the operating system.

The next step is to select the file that the user program will be using when reading or writing data. This group of operators (LOOKUP and ENTER) is not required for devices that are not file-structured (e.g., card reader, magnetic tape, paper tape punch). However, if used, they will be ignored, thus allowing file-structured devices to be substituted for non-file-structured devices without the user rewriting the program.

The third step is to perform the data transmission between the user program and the file (IN, INPUT, OUT, and OUTPUT). When the data has been transmitted to either the user program on input or the file on output, the file must be closed (CLOSE, fourth step) and the device released from the channel (RELEASE, fifth step). This same sequence of programmed operators is performed for all devices; therefore, the I/O system is truly device-independent because the user program does not have to be changed every time a different device is used.

In addition to reading or writing data to the standard I/O devices, provisions are included in the operating system for using the terminal for I/O during the execution of the user program. This capability is also obtained through programmed operators. As the user program is running, it can pause to accept input from or to type output to the terminal. The operating system does all buffering for the user, thus saving him programming time. This method of terminal I/O provides the user with a convenient way of interacting with his running program.

5.1.7 Summary

In summary, the resident operating system supervises user jobs and provides various services to these jobs. It acts as an operator by performing specific functions in response to specific events which occur within the system. Many functions are performed in accordance with a periodic event, the system clock interrupt. Other functions are responded to in accordance with the action of the user program.

5.2 COMMAND CONTROL LANGUAGE

By allowing resources to be shared among users, the time-sharing environment utilizes processor time and system resources that are wasted in single-user systems. Users are not restricted to a small set of system resources, but instead are provided with the full variety of facilities. By means of his terminal, the user has on-line access to most of the system's features. This on-line access is available through the operating system command control language, which is the means by which the time-sharing user communicates with the operating system.

Through the command language, the user controls the running of a task, or job, to achieve the desired results: create, edit, and delete files; start, suspend, and terminate a job; compile, execute, and debug a program. In addition, since GALAXY batch software accepts the same command language as the time-sharing software, any user can enter a program into the batch run queue. Thus, any time-sharing terminal can act as a remote job entry terminal.

When the user types commands and/or requests on his terminal, the characters are stored in an input buffer in the operating system. The command decoder examines these characters in the buffer, checks them for correct syntax, and invokes the system program or user program as specified by the command.

On each clock interrupt, control is given to the command decoder to interpret and process one command in the input buffer. The command appearing in the input buffer is matched with the table of valid commands accepted by the operating system. A match occurs if the command typed in exactly matches a command stored in the system, or if the characters typed in match the beginning characters of only one command (i.e., constitute a unique abbreviation). When the match is successful, the legality information (or flags) associated with the command is checked to see if the command can be performed immediately. For instance, a command must be delayed if the job is swapped out to the disk and the command requires that the job be resident in core; the command is executed on a later clock interrupt when the job is back in core. If all conditions as specified by the legality flags are met, control is passed to the appropriate program.

5.3 FILE SYSTEM


Mass storage devices, such as disks and drums, cannot be requested for a user's exclusive use, but must be shared among all users. (An exception is the assignment of a disk structure to a single user, i.e., a "private" disk structure.) Because many users share these devices, the operating system must ensure independence among the users; one user's actions must not affect the activities of another unless the users desire to work together. To guarantee such independence, the operating system provides a file system for disks, disk packs, drums, and DECTapes. Each user's data is organized into one or more 128-word blocks called files. The user gives a name to each of his files, and the list of these names is kept by the operating system for each user. The operating system is then responsible for protecting each user's file storage from intrusion by unauthorized users.

In addition to allowing independent file storage for users, the operating system permits sharing of files among individual users. For example, programmers working on the same project can share the same data in order to complete a project without duplication of effort. The operating system lets the user specify protection codes, or rights, for his files. These codes designate if other users may read the file, and after access, if the files can be modified in any way. A new facility called file DAEMON allows the user to specifically permit or deny access to any file, or set of files, for specific users. The user may also create a log of accesses to his files for later review, proprietary program billing, etc.

The user of the DECsystem-10 is not required to preallocate file storage; the operating system allocates and deallocates the file storage space dynamically on demand. Not only is this convenient for the user because he does not have to worry about allocation when creating files, but this feature also conserves storage by preventing large portions of storage from being unnecessarily tied up. However, a large batch job which needs to preallocate space may do so.

Files are assigned protection levels for each of three classes of users: self; users with a common project number; and all users. Each user class may be assigned a different access privilege; there are eight levels in each of the three user classes (Table 5-1).

Table 5-1
File Protection Scheme

Protection Level	Access Code	Access Privileges
	7	No access privileges
	6	EXECUTE ONLY
	5	READ, EXECUTE
	4	APPEND, READ, EXECUTE
	3	UPDATE, APPEND, READ, EXECUTE
	2	WRITE, UPDATE, APPEND, READ, EXECUTE
	1	RENAME, WRITE, UPDATE, APPEND, READ, EXECUTE
	0	CHANGE PROTECTION, RENAME, WRITE, UPDATE, APPEND, READ, EXECUTE
Least Protection		

In addition the file DAEMON is called on all access failures if the owner protection is 4, 5, 6, 7. . . . Note that the protection used by the system to determine access right after a file DAEMON call is provided by the file DAEMON.

5.3.1 File Handler

The disk file handler manages user and system data; thus, this data can be stored, retrieved, protected, and/or shared among other users of the computing system. All information in the system is stored as named files in a uniform and consistent fashion, thus allowing the information to be accessed by name instead of by physical disk addresses. Therefore, to reference a file, the user does not need to know where the file is physically located. A named file is uniquely identified in the system by a filename and extension, an ordered list of directory names (UFDs and SFDs) which identify the owner of the file, and a file structure name which identifies the group of disk units containing the file.

Usually a complete disk system is composed of many disk units of the same and/or different types. Therefore, the disk system consists of one or more file structures – a logical arrangement of files on one or more disk units of the same type. This method of file storage allows the user to designate which disk unit of the file structure he wishes to use when storing his files. Each file structure is logically complete and is the smallest section of file memory that can be removed from the system without disturbing other units in other file structures. All pointers to areas in a file structure are by way of logical block numbers rather than physical disk addresses; there are no pointers to areas in other file structures, thereby allowing the file structure to be removed.

5.3.2 File Structures

A file structure contains two types of files: the data files that physically contain the stored data or programs, and the directory files that contain pointers to the data files. Included in these directory files are master file directories, user file directories, and subfile directories. Each file structure has one master file directory (MFD). This directory file is the master list of all the users of the file structure. The entries contained in the MFD are the names of all the user file directories on the file structure. Each user with access to the file structure has a user file directory (UFD) that contains the names of all his files on that file structure; therefore, there are many UFDs on each file structure. As an entry in the user file directory, the user can include another type of directory file, a subfile directory (SFD). The subfile directory is similar to the other types of directory files in that it contains as entries the names of all files within that subdirectory. This third level of directory allows groups of files belonging to the same user to be separate from each other. This is useful when organizing a large number of files according to function. In addition, subfile directories allow nonconflicting simultaneous runs of the same program using the same filenames.

As long as the files are in different subfile directories, they are unique. Subfile directories exist as files pointed to by the user file directory, and can be nested to the depth specified by the installation at system generation time.

5.3.3 File Protection

All disks are composed of two parts: data and information used to retrieve data. The retrieval part of the file contains the pointers to the entire file, and is stored in two distinct locations on the device and accessed separately from the data. System reliability is increased with this method because the probability of destroying the retrieval information is reduced; system performance is improved because the number of positionings needed for random access methods is reduced. The storing of retrieval information is the same for both sequential and random access files. Thus a file can be created sequentially and later read randomly, or vice versa, without any data conversion.

One section of the retrieval information is used to specify the protection associated with the file. This protection is necessary because disk storage is shared among all users, each of whom may desire to share files with, or prevent files from being written, read, or deleted by, other users. These protection codes are assigned by the user when the file is created and designate the users who have privileges to access the file.

5.3.4 Disk Quotas

Disk quotas are associated with each user (each project-programmer number) on each file structure in order to limit the amount of information that can be stored in the UFD of a particular file structure. When the user gains access to the computing system, he automatically begins using his logged in quota. This quota is not a guaranteed amount of space, and the user must compete with other users for it. When the user leaves the computing system, he must be within his logged out quota. This quota is the amount of disk storage space that the user is allowed to maintain when he is not using the system and is enforced by the system program that is used in logging off the system. Quotas are determined by the individual installation and are, therefore, used to ration disk resources in a predetermined manner.

To the user, a file structure is like a device, i.e., a file structure name or set of file structure names can be used as the device name in command strings or UUC calls to the operating system. Although the file structures or the units composing the file structures can be specified by their actual names, most users specify a general, or generic, name (DSK) which will cause the operating system to select the appropriate file structure. The appropriate file structure is determined by a job search list. Each job has its own job search list with the file structure names in the order in which they are to be accessed when the generic name is specified as the device. This search list is established by LOGIN and thus each user has a UFD for his project-programmer number in each file structure in which LOGIN allows him to have files.

5.3.5 File Operations

File writing on the disk can be defined by one of three methods: creating, superseding, and updating. The user is creating a file if no other file of the same name exists in the user's directory on the indicated file structure. If another file with the same name already exists in the directory, the user is superseding, or replacing, the old file with the new file. Other users sharing the old file at the time it is being superseded continue using the old file and are not affected until they finish using the file and then try to access it again. At that time, they read the new file. When a user updates a file, he modifies selected parts of the file without creating an entirely new version. This method eliminates the need to recopy a file when making only a small number of changes. If other users try to access a file while it is being updated, they receive an error indication issued by the system.

5.3.6 Disk Storage Management

File storage is dynamically allocated by the file handler during program operations, so the user does not need to give initial estimates of file length or the number of files. Files can be any length, and each user may have as many files as he wishes, as long as disk space is available and the user has not exceeded his logged in quota. This feature is extremely useful during program development or debugging when the final size of the file is still unknown. However, for efficient random access, a user can reserve a contiguous area on the disk if he desires. When he has completed processing, he can keep his preallocated file space for future use or return it so that other users can have access to it.

5.4 INPUT/OUTPUT

5.4.1 Peripheral Device Assignment

With the command language, the user can also request assignment of any peripheral device (magnetic tape, DECtape, and private disk pack) for exclusive use. When the request for assignment is received, the operating system verifies that the device is available to this user, and the user is granted its private use until he relinquishes it. In this way, the user can also have complete control of devices such as card readers and punches, paper tape readers and punches, and line printers.

5.4.2 Spooling

When private assignment of a slow-speed device (card punch, line printer, paper tape punch, and plotter) is not required, the user can employ the spooling feature of the operating system. Spooling is a method by which output to a slow-speed device is placed on a high-speed disk or drum. This technique prevents the user from consuming unnecessary system resources while waiting for either a device to become available or output to be completed. In addition, the device is managed to a better degree because the users cannot tie it up indefinitely, and the demand fluctuations experienced by these devices are equalized.

5.5 MEMORY MANAGEMENT

The DECsystem-10 is a multiprogramming system, i.e., it allows multiple independent user programs to reside simultaneously in memory and to run concurrently. This technique of sharing memory and processor time enhances the efficient operation of the system by switching the processor from a program that is temporarily stopped because of I/O transmission to a program that is executable. When core and the processor are shared in this manner, each user's program has a memory area distinct from the area of other users. Any attempt to read or change information outside the area a user can access immediately stops the program and notifies the operating system.

5.5.1 Secondary Memory

Because available memory can contain only a limited number of programs at any one time, the computing system employs a secondary memory, usually disk or drum, to increase the number of users serviced. User programs exist on the secondary memory and move into main memory for execution. Programs in memory exchange places with the programs being transferred from secondary memory for maximum use of available main memory. Because the transferring, or swapping, takes place directly between main memory and the secondary memory, the central processor can be operating on a user program in one part of memory while swapping is taking place in another. This independent, overlapped operation greatly improves system utilization by increasing the number of users that can be accommodated at the same time.

5.5.2 Reentrant Software

To further increase the utilization of memory, the operating system allows users to share the same copy of a program or data segment. This prevents the excessive memory usage that results when a program is duplicated for several users. A program that can be shared is called a reentrant program and is divided into two parts or segments. One segment contains the code that is not modified during execution (e.g., compilers and assemblers) and can be used by any number of users. The other segment contains non-reentrant code and data. The operating system provides protection for shared segments to guarantee that they are not accidentally modified.

5.5.3 Virtual Memory

The virtual memory option permits a user program to execute with an address space greater than the physical memory actually allocated to that program during execution. User jobs are swapped as described above. However, the entire program may not necessarily be in core during execution. Programs are divided into pages, each of which is 512 words long. Some of these pages may remain on secondary storage while the program executes. When a virtual memory job attempts to access a page that is not in core, a page fault handler decides which page or pages to remove from core and which to bring in from secondary storage.

Unlike the virtual memory implementation on other systems, this DECsystem-10 feature is an option. Each site may determine its own need for virtual memory and install it at their convenience. The system administrator may grant the privilege for using virtual memory only to those users who truly need its capabilities. Those users who are granted the privilege of using virtual memory may elect to invoke the feature for only those programs that could not execute without the virtual memory capability.

The virtual memory users may elect to use the system page fault handler or they may use a handler that is more tailored to the particular application or program behavior. Finally, it is important to point out that only those users actually using the virtual memory feature are affected by any additional overhead associated with a demand paging system. Non-virtual users execute as they would in a non-virtual system with no discernible difference in performance.

5.6 MULTIPROCESSING SYSTEMS

DECsystem-10 dual-processor systems are composed of two CPUs, designated the primary processor (master) and the secondary processor (slave). The primary processor is connected to all of the memory in the system and has all of the system's peripheral I/O equipment connected to its I/O bus. The secondary processor also has access to all of memory; however, there are normally no I/O devices on this processor's I/O bus, although certain devices may be attached in real-time applications. The primary processor performs exactly the same operations as the processor in a single-processor system. This includes all I/O operations, swapping, core allocation, resource allocation, and command decoding. The secondary processor also performs scheduling and execution of user jobs according to the same algorithm used in a single-processor system.

The secondary processor executes user jobs and scans the same job queues as the primary processor. However, since the slave cannot do any standard I/O, it looks for any compute-bound jobs which are in core and runnable. A software interlock has been added to the scheduler to prevent the possibility of both processors trying to execute the same job at the same time. Whenever a job being executed by the secondary processor requests an I/O operation to be performed, the job is stopped and marked for execution on the master only. Thus both processors run completely asynchronously, both executing the same scheduler, doing the same job accounting, and using the same job queues.

The existence of dual processors gives DECsystem-10 users a large-scale computing capability, especially in the areas of highly compute-bound jobs and noninteractive batch jobs. The existence of a slave more or less dedicated to user computation allows these jobs to be carried on in the systems with little interference with time-sharing users. The performance goal is to provide a system in which each customer can improve the service offered to his users over a single-processor, compute-bound system. Either the customer can add up to 1/3 more users with the same response time or he can keep the same number of users and reduce turnaround time for compute-bound jobs.

5.7 INTERJOB COMMUNICATIONS

5.7.1 Shared Data Areas

The DECsystem-10 operating system enables a real-time user to communicate with other jobs through the use of sharable data areas. This also enables a data analysis program, for example, to read or write an area in the real-time job's core space. Since the real-time job associated with the data acquisition would be locked in core, the data analysis program residing on secondary memory would become core resident only when the real-time job had filled a core buffer with data. Operating system calls can be used to allow the data analysis program to remain dormant in secondary memory until a specified event occurs in the real-time job, e.g., a buffer has been filled with data for the data analysis program to read. When the specified event occurs, the dormant program is then activated to process the data. The core space for the real-time job's buffer area or the space for the dormant job does not need to be reserved at system generation time. The hardware working in conjunction with the operating systems core management facilities provides optimum core usage.

5.7.2 Interprocess Communication Facility

The interprocess communication facility (IPCF) provides the capability for independent jobs to communicate with one another. For example, if several programs are involved in processing or maintaining a data base, it is possible that one program might want to inform the others of any modifications it made to the data. A job using IPCF cannot make any changes to another job so protection is in no way sacrificed when using the IPCF feature.

In order to use IPCF, each participating job that wishes to receive communication from other jobs must request a unique process (job) identifier (PID) from the system. The transmitting job then may send a "packet" of information to another job. (In addition to the information, the system automatically provides a "return address" so that the receiving program can respond to the sender.) The monitor maintains a linear queue (the "mailbox") for each job using IPCF. The packet (or packets) will be kept in the mailbox until the receiving job retrieves it. This queue is not created until a job sends an IPCF packet and it does not occupy any space until such time. The maximum number of packets allowed in a queue at any one time is determined by a "receive" quota that may be set at each installation for each user. (If no quota is set by the installation, the standard default is five.)

On systems with the virtual memory option, the packet could be an entire page. In this case, the monitor takes advantage of the page mapping hardware of the KI10 and KL10 to transmit the page without actually copying it.

5.8 NONRESIDENT SYSTEM SOFTWARE

For the computer to execute any of the basic operations which it is capable of executing, it must be told which operation it is to perform and where to find the information on which to perform the operation. This requires that a language be established by which the user can indicate to the computer what it needs to know. This language is the machine language of the computer and is unique for each machine. This machine language is the means by which the computer's circuits are instructed to perform the desired operation and, because of this, it is the fastest and most direct method of programming. However, machine language programming is not the easiest method of programming for most users to employ. Although it is not impossible to memorize all of the operation codes recognized by the computer, it can be very difficult for the programmer to remember where each piece of information is inside memory in order to direct the computer to it. Therefore, symbolic language programming was developed to aid the programmer in manipulating the computer.

With symbolic language programming, programs are written using symbols which, when translated, equal the machine language of the computer. Symbol operation codes (mnemonics that specify which operation the user wants the computer to perform) are translated to the actual, or absolute, operation codes that the computer understands. Addresses of core are designated with symbolic labels and are converted into absolute core addresses so that the computer can locate the information on which to perform the desired operation.

There are three kinds of translators used in symbolic language programming: assemblers, compilers, and interpreters. An assembler is a program that is able to take another program written in symbolic language and translate it, item by item, into machine language. Therefore, to assemble a program means to substitute one absolute value for one symbolic notation until the entire program has been translated. A compiler also translates a symbolic language program into a machine language program, but the substitution is not one-to-one. A program written in a compiler language is freer in format than an assembly language program, and the language elements usually resemble English words. The compiler is larger and more complex than most assemblers, because it translates a program that is farther away from the machine language. Generally, one statement written in a compiler language is translated into several machine language instructions. Although a compiler occupies more space in memory and is generally slower than an assembler, a program written in a compiler language is more compatible with other computer models and the language itself is easier to learn and write because of its general statements and freer format. An interpreter differs from a compiler in that a binary version of the program is not produced for storage. Each statement of the source text is translated into machine language and then executed immediately before the next statement is processed. Interpreters are generally useful when the program to be translated will probably be executed only once or twice and therefore not need some of the time-consuming features of a compiler, such as program optimization and output of the machine language instructions.

5.8.1 MACRO Assembler

MACRO is the symbolic assembly program on the DECsystem-10. 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 numeric values
3. Assigning absolute core addresses to the symbolic addresses or program instructions and data
4. Preparing an output listing of the program which includes any errors detected during the assembly process.

MACRO programs consist of a series of statements that are usually prepared on the user's terminal with a system editing program. The elements in each statement do not have to be placed in certain columns nor must they be separated in a rigid fashion. The assembler interprets and processes these statements, generates binary instructions or data words, and performs the assembly.

MACRO is a 2-pass assembler. This means that the assembler reads the source program twice. Basically, on the first pass, all symbols are defined and placed in the symbol table with their numeric values, and on the second pass, the binary (machine) code is generated. Although not as fast as a 1-pass assembler, MACRO is more efficient in that less core is used in generating the machine language code and the output to the user is not as long.

MACRO is a device-independent program; it allows the user to select at runtime 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.

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 the arguments changed. The code sequence is defined with dummy arguments as a macro instruction. Thus, a single statement in the source program referring to the macro by name, along with a list of the real arguments, generates the correct and entire sequence. This capability allows for the expansion and adaptation of the assembler in order to perform specialized functions for each programming job.

5.8.2 Compilers

5.8.2.1 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 expressed as complete and precise statements of a procedure.

The DECsystem-10 ALGOL system is based on ALGOL-60. It is composed of the ALGOL processor, or 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. Also any errors the user made in writing his program are detected by the compiler and passed on to the user.

The ALGOL object time system provides special services, including the input/output service, for the compiled ALGOL program. Part of the object time system is the ALGOL library, a set of routines that the user's program can call in order to perform calculations. These include the mathematical functions and the string and data transmission routines. These routines are loaded with the user's program when required; 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.

5.8.2.2 BASIC – The Beginner's All-purpose Symbolic Instruction Code, BASIC, is a problem-solving language that is easy to learn because of its conversational nature. It is particularly suited to a time-sharing environment because of the ease of interaction between the user and the computer. This language can be used to solve problems with varying degrees of complexity, and thus has wide application in the educational, business, and scientific markets.

BASIC is one of the simplest of the programming compiler languages available because of the small number of clearly understandable and readily learned statements that are required for solving almost any problem. The BASIC language can be thought of as divided into two sections: one section of elementary statements that the user must know in order to write simple programs and a second section of advanced techniques for more powerful programs.

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

The BASIC system has several special features built into its design:

1. BASIC contains its two editing facilities. Existing programs and data files can be modified directly with BASIC instead of with a system editor by adding or deleting lines, by resequencing the line numbers, or by combining two files into one. The user can request a listing of all or part of any of his files on either the line printer or the terminal.
2. At the editing level, BASIC allows various peripheral devices (e.g., disk, magnetic tape, DECtape, card reader and punch, high-speed paper tape reader and punch, and paper tape reader and punch attached to the user's terminal) to be used for storage or retrieval of programs and data files. Within a program, information can be read from or written to the terminal and to the disk (in the latter case, either sequentially or by random access).
3. Output to the terminal can be simply formatted by tabs, spaces, and columnar headings or more precisely formatted by using the advanced PRINT USING statement.
4. BASIC has statements designed exclusively for matrix computations.
5. An advanced string handling capability includes a concatenation operator, substring and search functions, and other string intrinsic functions. Mathematical intrinsic functions are contained in BASIC, along with methods by which the user can define his own functions.

5.8.2.3 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 easily describe the formats of his data and the actions to be performed on this data in simple English-like statements. Therefore, programming training is minimal, COBOL programs are self-documenting, and programming of desired applications is accomplished quickly and easily.

The COBOL system is composed of a number of software components. The first is the COBOL compiler which is responsible for initializing the program, scanning the command strings for correct syntax, generating the code, listing, and final assembly. The second component is the object time system, LIBOL, which consists of subroutines used by the code generated by the compiler. These subroutines include the I/O, conversion, comparison, and mathematical routines available to the COBOL user. Another component is the source library maintenance program, which builds and maintains a library of source language entries that can be included in the user's source program at compile time. A fourth component is the stand-alone report generator, COBRG, which produces COBOL source programs. When compiled and loaded, these programs generate reports. The stand-alone program, SORT, accepts commands from the user's terminal in order to perform simple sorting of files. The RERUN program is used to restart a COBOL program that was interrupted during execution because of a system failure, device error, or disk quota error. COBDDT is a utility that debugs COBOL programs. Finally, ISAM builds and maintains indexed sequential files for the user.

DECsystem-10 COBOL accepts two source program formats: conventional format and standard format. The conventional format is employed when the user desires his source programs to be compatible with other COBOL compilers. This is the format normally used when input is from the card reader. The standard format is provided for users who are familiar with the format used in DECsystem-10 operations. It differs from conventional format in that sequence numbers and identification are not used because most DECsystem-10 programs require neither. The compiler assumes that the source program is written in standard format unless the appropriate switch is included in the command string to the compiler or the special sequence numbers created by the symbolic editor LINED are detected by the compiler.

DECsystem-10 COBOL is the highest level of ANSI COBOL available and because it operates within the operating system, it offers the user the many features of the DECsystem-10 in addition to the business processing capability of the language. These features enable the COBOL user to run programs in either time-sharing or batch processing environments or both, to perform on-line editing and debugging of his programs with the system programs available, to choose various peripheral devices for input and output, and to write programs that can be shared with other users.

5.8.2.4 FORTRAN – The FORMula TRANslator language, FORTRAN, is a widely used procedure-oriented programming language. It is designed for solving scientific-type problems and is thus composed of mathematical-like statements constructed in accordance with precisely formulated rules. Therefore, programs written in the FORTRAN language consist of meaningful sequences of these statements that are intended to direct the computer to perform the specific computations.

FORTRAN has a varied use in every segment of the computer market. Universities find that FORTRAN is a good language with which to teach students how to solve problems via the computer. Scientific markets rely on FORTRAN because of the ease with which scientific problems can be expressed. In addition, FORTRAN is used as the primary data processing language by time-sharing utilities.

Because of this wide market, DECsystem-10 FORTRAN-10 is designed to meet the needs of all users. FORTRAN-10 is compatible with and encompasses the ANSI standard. FORTRAN-10 also provides many extensions and additions to this standard which greatly enhance its usefulness and increase its compatibility with other FORTRAN language sets.

FOROTS, the FORTRAN-10 object time system, implements all program data file functions and provides the user with an extensive runtime error reporting system. An additional feature is that the association between FORTRAN logical units and the file descriptions to which they refer may be made either within the source program or deferred until runtime.

DECsystem-10 FORTRAN-10 also supports FORDDT, an interactive program that is used as an aid in debugging FORTRAN programs.

The FORTRAN system is easy to use in both the time-sharing and batch processing environments. Under time-sharing, the user operates in an interactive editing and debugging environment. Under batch processing, the user submits his program through the GALAXY 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, DECtape, user terminal, paper tape reader, 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 programs LINED, PIP, or TECO. The data files contain the data needed by the user's object program during execution. The source files contain the FORTRAN compiler. Commands are entered directly to the FORTRAN compiler with a run-type command or indirectly through a system utility program that accepts and interprets the user's command string and passes it to the compiler. Output can then be received on the user's terminal, disk, DECtape, magnetic tape, card punch, or paper tape punch.

5.8.3 Interpreters

5.8.3.1 AID – The Algebraic Interpretive Dialogue, AID, in the DECsystem-10 adaptation of the language elements of JOSS, a program developed by the Rand Corporation. To write a program in the AID language requires no previous programming experience. Commands to AID are typed in via the user's terminal as imperative English sentences. Each command occupies one line and can be executed immediately or stored as part of a routine for later execution. The beginning of each command is a verb taken from the set of AID verbs. These verbs allow the user to read, store and delete items in storage; halt the current processing and either resume or cancel execution; type information on his terminal; and define arithmetic formulas and functions for repetitive use that are not provided for in the language. However, many common algebraic and geometric functions are provided for the user's convenience.

The AID program is device-independent. The user can create external files for storage of subroutines and data for subsequent recall and use. These files may be stored on any retrievable storage media, but for accessibility and speed, most files are stored on disk.

5.8.3.2 APL – APL (A Programming Language) is a concise programming language especially suitable for dealing with numeric and character array-structured data. APL is a completely conversational system which tends to increase programmer productivity and expertise by allowing the user to interact with the APL system and his running programs. APL is rich in operators that facilitate array calculations. This higher level programming is accomplished by suppressing much of the programming detail inside single APL operators. One operator may be used to sort a vector of values in ascending order, thereby making "sort" a primitive operation rather than a tedious subroutine. APL is intended for use as a general data processing language as well as a mathematician's tool.

5.8.3.3 CPL – CPL is a highly interactive language interpreter. The language itself is a subset of the PL/1 programming language. It provides a rich array of data types, function and subroutine capabilities, storage control, and input/output facilities. In addition, CPL includes its own line-oriented editor and a debugging system that allows the user to interactively and incrementally diagnose program problems. It runs under the control of current versions of the TOPS-10 operating system. CPL can handle any number of independent users, and can be used simultaneously with other languages in either batch or time-sharing modes.

5.8.4 Editors

5.8.4.1 LINED – The LINE EDitor for disk files, LINED, is used to create and edit source files written in ASCII code with line numbers appended. These line numbers allow LINED to reference a line in the file at any time without having the user close and then reopen the file. The user has the option of either specifying the beginning line number and the increment to the next line number when inserting lines or allowing LINED to assume a beginning line number and increment if the user specification is omitted.

Commands to LINED allow the user to create a new file or edit an existing file by inserting, replacing, or deleting lines within the file. Single or multiple lines of the file can be printed on the user's terminal for an aid in editing. When the user has edited the file to his satisfaction, he closes the file and can either open a new file or return to monitor level to assemble or compile his file.

5.8.4.2 TECO – The Text Editor and COrrector program, TECO, is a powerful editor used to edit any ASCII text file with a minimum of effort. TECO commands can be separated into two groups: one group of elementary commands that can be applied to most editing tasks, and the larger set of sophisticated commands for character string searching, text block movement, conditional commands, programmed editing, and command repetition.

TECO is a character-oriented editor. This means that one or more characters in a line can be changed without retyping the remainder of the line. TECO has the capability to edit any source document: programs written in MACRO, FORTRAN, COBOL, ALGOL, or any other source language; specifications; memoranda; and other types of arbitrarily formatted text. The TECO program does not require that line numbers or other special formatting be associated with the text.

Editing is performed by TECO via an editing buffer, which is a section within TECO's core area. Editing is accomplished by reading text from any device (except a user's terminal) into the editing buffer (inputting), by modifying the text in the buffer with data received from either the user's terminal or a command file (inserting), and by writing the modified text in the buffer to an output file (outputting).

A position indicator, or buffer pointer, is used to locate characters within the buffer and its position determines the effect of many of TECO's commands. It is always positioned before the first character, between two characters, or after the last character in the buffer. Various commands, such as insertion commands, always take place at the current position of the buffer pointer.

There are TECO commands to manipulate data within the editing buffer. Input and output commands read data from the input file into the buffer and output data from the buffer to the output file. There are other commands to have one or more characters inserted into the editing buffer, deleted from the buffer, searched for, and/or typed out. In addition, the user can employ iteration commands to execute a sequence of commands repeatedly and conditional execution commands to create conditional branches and skips.

5.8.4.3 SOUP – The Software Updating Package, SOUP, is a set of programs that facilitates the updating of system or user source files. Because software is constantly being updated to reflect changes and improvements made by DIGITAL, a method to make the updating process easier and faster for all concerned was developed. SOUP enables DIGITAL to distribute a file containing only the differences to the software routine instead of redistributing the entire routine. In addition, since customers frequently maintain system files that are modified to reflect their individual needs, SOUP can be used to update these modified files as well. Although SOUP was implemented to update system files, it can be employed to update any source file with more than one version.

The Software Updating Package consists of three programs. The first program, CAM, is responsible for:

1. Comparing the new version of DIGITAL's system file to the previous version to produce a correction file
2. Merging two correction files derived from the same file to produce a single correction file.

The correction file contains a series of editing changes that reflect the differences between the old and new versions of the system files. The two functions of CAM can be performed simultaneously or one at a time, depending on the user's command string to CAM.

The second program, COMP10, is used when the customer has modified DIGITAL's file to such an extent that CAM cannot compare the modified file to the original file due to buffer overflow. COMP10 has extremely large buffers and can, therefore, be used to generate the correction file.

The third program, FED, reads the correction file and edits the copy of the system file by making the changes indicated in the correction file. When FED has completed its processing, the user has an updated file. As a software manufacturer, DIGITAL sends the user a correction file, and he, in turn, need only run the FED program in order to update his system files.

5.8.4.4 RUNOFF – RUNOFF facilitates the preparation of typed or printed manuscripts by performing line justification, page numbering, titling, indexing, formatting, and case shifting as directed by the user. The user creates a file with an editor and enters his material through his terminal. In addition to entering the text, the user includes information for formatting and case shifting. 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 will remain unchanged. After the group of modifications have been added to the file, RUNOFF produces a new copy of the file which is properly paged and formatted.

5.8.5 Utilities

5.8.5.1 CREF – The Cross REFerence listing program, CREF, is an aid in program debugging and modification. It produces a sequence-numbered assembly listing followed by tables showing cross-references of all operand-type symbols, all user-defined operators, and all machine op codes and pseudo op codes.

The input to CREF is a modified assembly listing created during assembly or compilation. The command string entered by the user specifies the device on which this assembly listing is located along with the output device on which to list the cross-reference tables and assembly listings. Switches can also be included in the command string in order to control magnetic tape positioning and to select specific sections of the listing output.

5.8.5.2 DBMS-10 – The Data Base Management System (DBMS-10) is a facility of the DECsystem-10 that permits the user to consolidate his data files into one or more data bases. A data base is a collection of nonredundant data items that can be accessed by a variety of programs and/or applications that have common processing requirements and functional relationships. The data base is created and maintained through modules of DBMS-10. These modules permit the user to structure the data such that each application can access it in optimum fashion, yet no data item is actually duplicated in the data base. This arrangement is accomplished by the data base administrator who structures the data base in a manner such that each application can access it through a search pattern most suited to its needs. Once the data base has been established, users can access the data through COBOL programs containing special data base syntax.

5.8.5.3 DDT – The Dynamic Debugging Technique, DDT, is used for on-line program composition of object programs and for on-line checkout and testing of these programs. For example, the user can perform rapid checkout of a new program by making a change resulting from an error detected by DDT and then immediately executing that section of the program for testing.

After the source program has been compiled or assembled, the binary object program with its table of defined symbols is loaded with DDT. In command strings to DDT, the user can specify locations in his program, or breakpoints, where DDT is to suspend execution in order to accept further 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 begins execution or at breakpoints, the user can examine and modify the contents of any location. Insertions and deletions can be in source language code or in various numeric and text modes. DDT also performs searches, gives conditional dumps, and calls user-coded debugging subroutines at breakpoint locations.

5.8.5.4 FAILSAFE/BACKUP – The FAILSAFE/BACKUP programs are used to selectively or universally save disk files on magnetic tape. Files may be copied to magnetic tape, in a special format, selectively by filename or groups of filenames, by project/programmer number, by disk structure (logical or physical), or universally. In addition to providing backup of files, these programs allow a method of extending disk space by allowing infrequently used files to be stored on tape. Restoring of files from FAILSAFE/BACKUP format tapes can likewise be handled on an individual file, on groups of files, or on a project/programmer, logical, or physical disk basis. An added benefit is achieved when restoring files in that disk fragmentation is often minimized by the failsafe/restore operation.

5.8.5.5 FILEX – The FILE transfer program, FILEX, converts between various core image formats and reads or writes various DECTape directory formats and standard disk files. Files are transferred as 36-bit binary data with no processing performed on the data except that necessary to convert the core image representation. The core image formats that can be used in conversions are:

1. Saved-file format
2. Expanded core image file format
3. Dump format
4. Simple block format (project MAC's equivalent of DEC's .SAV format)
5. Binary file format.

The desired core image format is determined by the specific extension associated with the file but this extension may be overridden by the use of switches in command strings to FILEX.

DECtapes can be read or written in binary, PDP-6 DECtape format, MIT Project MAC PDP-6/10 DECtape format, PDP-11 DOS format, or PDP-15 format. In the latter two cases, ASCII files will be converted. The DECtape can be processed quickly via a disk scratch file, which is a much faster method for a tape with many files. This scratch file can be preserved and reused in later command strings. In addition, the DECtape directory can be listed on the user's terminal or zeroed in the appropriate format on the tape. These DECtape format and processing specifiers are indicated by command string switches.

5.8.5.6 ITPS-10 – The In-House Text Preparation System (ITPS-10) allows DECsystem-10 users to prepare office-quality or camera-ready (typeset) copy using the interactive capability of the DECsystem-10. High-quality, camera-ready copy for pamphlets, advertising brochures, product flyers, user manuals, and many other types of documentation can be produced and updated rapidly and inexpensively using ITPS-10. The system allows use of many different input and output devices, including OCRs, interactive CRT or hard copy terminals, line printers, typewriter-like devices, and photocomposition equipment. Documents are stored as standard DECsystem-10 files and can be easily updated, edited, and re-generated.

The system is easy to use and easy to learn. Current successful applications range from preparing form letters to setting daily newspapers.

5.8.5.7 LINK-10 – LINK-10, the DECsystem-10 linking loading, merges independently translated modules of the user's program into a single module and links this module with system modules into a form that can be executed by the operating system. It provides automatic relocation and loading of the binary modules producing an executable version of the user's program. When the loading process has been completed, the user can request LINK-10 either to transfer control to his program for immediate execution or to output the program to a device for storage in order to avoid the loading procedure in the future.

While the primary output of LINK-10 is the executable version of the user's program, the user can request auxiliary output in the form of map, log, save, symbol, overlay plot, and expanded core image files. This additional output is not automatically generated: the user must include appropriate switches in his command strings to LINK-10 in order to obtain this type of output. The user can also gain precise control over the loading process by setting various loading parameters and by controlling the loading of symbols and modules. Furthermore, by setting switches in his command strings to LINK-10, the user can 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 verbosity levels of the messages. Finally, he can accept the LINK-10 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.

LINK-10 has an overlay facility to be used when the total core required by a user's program is more than the core available to the user. The user organizes his program so that only some portions of the program are required in core at any one time. The remaining portions reside in a disk file and are transferred in and out of core during execution. Because the portion brought into core may overlay a portion already core-resident, the amount of core required by the entire program is reduced. Overlays can be evoked either by runtime routines called from the user's program or by automatic calls to subroutines outside the current overlay link.

5.8.5.8 PIP – The Peripheral Interchange Program, PIP, is used to transfer data files from one I/O device to another. Commands to PIP are formatted to accept any number of input (source) devices and one output (destination) device. Files can be transferred from one or more source device to the destination device as either one combined file or individual files. Switches contained in the command string to PIP provide the user with the following capabilities:

1. Naming the files to be transferred
2. Editing data in any of the input files
3. Defining the mode of transfer
4. Manipulating the directory of a device if it has a directory
5. Controlling magnetic tape and card punch functions
6. Recovering from errors during processing.

5.8.5.9 MACY11 and LNKX11 – MACY11 is a cross-assembler which operates on the DECsystem-10 and assembles MACRO-11 source code for the PDP-11 family of computers.

MACY11 will produce either an absolute binary or a relocatable object module, depending on the assembly mode. The resulting absolute binary files are transferable to the PDP-11 for execution or the relocatable object modules may be used as input to the LNKX11 linker on the DECsystem-10 and then transferred to the PDP-11 operating environment. MACY11 will append a symbol table and (optionally) a cross-reference table to the listing file. MACY11 produces a “side-by-side” assembly listing of symbolic source statements, their octal equivalents, assigned addresses, and error codes.

5.8.5.10 MCS-10 – The Message Control System (MCS-10) provides the DECsystem-10 with an efficient, transaction processing-oriented system that facilitates control of communications between a network of terminals and applications processes. The intent is to provide generalized routines that are readily used from COBOL programs and network definitions. Message control in its fullest sense includes a complex of hardware and software systems:

1. Terminals and Transmission Devices
2. Remote and Central Site Concentrators
3. Central Site Communications Multiplexers
4. DECsystem-10 Monitor and Its Message Service Module
5. Message Control System (MCS-10) Routines
6. Extended Communications Language Module and Interfaces (COBOL/LIBOL)
7. Network Definition, Activation, and Operations Control Routines
8. Message-Oriented Applications Programs
9. Data Base Management Capability via DBMS-10.

The activities normally associated with message control are cooperative processes distributed among communications concentrators, multiplexers, and a central system. Actual I/O transmission, line control, error handling, device selection, code translation, and message buffering are handled by the remote and local concentrators and front-end processors. The DECsystem-10 Central Processor, monitor, and message control modules accept, route, queue, and log terminal and application program-generated messages. Upon the occurrence of specific user-specified events, application programs are activated to perform message processing. Applications programs receive messages from queues through interface modules, perform processes related to data within messages, and send messages back through queues to terminals and/or other application processes.

5.8.6 Monitor Support Programs

5.8.6.1 MONGEN – The MONitor GENerator, MONGEN, is a dialogue program that enables the system programmer to define the hardware configuration of his individual installation and the set of software options that he wishes to select for his system. This program is a prerequisite for creating a new monitor.

The system programmer defines his configuration in one of four dialogues by answering MONGEN's questions in conversational mode. MONGEN transmits one question at a time to the user's terminal, and the user answers appropriately depending on the content of each question. After all questions have been answered, MONGEN produces MACRO source files containing the user's answers. These source files are then assembled and loaded with the symbol definition file and the monitor data base to yield a monitor tailored to the individual installation.

5.8.6.2 OPSER – The OPERator SERvice program, OPSER, facilitates multiple job control from a single terminal by allowing the operator or the user to initiate several jobs, called subjobs, from his terminal. The OPSER program acts as the supervisor of the various subjobs by allowing monitor-level and user-level commands to be passed to all of the subjobs or to individually selected subjobs. Output from the various subjobs can then be retrieved by OPSER.

The subjobs of OPSER run on pseudo-TTYs, a simulated terminals not defined by hardware. All initializations of the pseudo-TTYs are performed by OPSER; the operator need only supply a subjob name. By running system programs, which ordinarily required a dedicated terminal, as subjobs of OPSER, output from the various programs can be concentrated on one hardware terminal instead of many. In addition, OPSER is able to maintain an I/O link between the running jobs and the operator – a feature that is not available when programs run on their own dedicated terminals.

5.8.6.3 LOGIN – LOGIN is the system program used to gain access to the DECsystem-10. This program consults system administration files in order to determine whether or not a potential user currently is authorized to use the system. If he is not, LOGIN will not permit him access to the system. If the user is authorized, LOGIN informs him of messages of the day, reports any errors detected in his disk files, and allows the user to proceed with his job.

5.8.6.4 KJOB-LOGOUT – The user evokes the system programs KJOB and LOGOUT when he has finished running on the DECsystem-10. The many functions of these programs include saving the user's disk files in the state in which he desires them, enforcing logged-out quotas on all disk file structures, terminating the user's job, and returning the resources allocated to the user back to the system. These resources include the user's job number, his allocated I/O devices, and his allocated core.

5.9 CONSOLE SOFTWARE

The PDP-11 console processor is equipped with its own software package which provides the basic subroutines required for operation of KL10 and PDP-11 programs. The subroutine package also includes a separate program which is linked to/from the diagnostic segments. The software package may be described as two segments: monitor and diagnostic, where each segment is an overlay of the other. Each overlay has an associated console command module.

The monitor overlay is that code which is normally in the PDP-11 during the time that the monitor is running in the KL10. The overlay provides device support to the KL10 monitor for the console terminal (CTY), one synchronous line for remote diagnosis, the RH11/RP04 disk system and the TC11/TU56 DECTape system.

In addition, the monitor overlay can – through the console command module – communicate with one or more console terminals to perform console functions for the operator.

The diagnostic overlay consists of a subroutine package and the full console command module. The subroutine package supports diagnostics that run in the PDP-11 and diagnose the KL10. When the diagnostic overlay is running, the subroutine package is resident and allows various diagnostics to be retrieved and executed as required.

The monitor and diagnostic overlays are compatible in several ways. Communications between the KL10 and the console terminal are handled through the same communication area with the same communication protocol. When the monitor and diagnostic overlays are swapped, the KL10 and console terminal communication is maintained since the communication area retains the terminal number and its speed. Other status information is also maintained between overlays. For example, if the monitor overlay has difficulty with a particular device, a flag is passed to the diagnostic overlay when it is swapped in, indicating difficulty with that device.

An area of code not used by either the monitor or diagnostic overlays remains resident in PDP-11 memory for such functions as taking a crash dump of the KL10 in the event both the KL10 and PDP-11 crash simultaneously. In addition there will be code common to both overlays resident when either overlay is in PDP-11 core. This code would include segments of the console command facility that are common to both overlays.

Note that the diagnostic software which does stand-alone diagnosis of the KD11 and its peripheral devices will be executed independently of the console processor software package.

5.9.1 Basic Command Facility Description

The console command module in the monitor and diagnostic overlays implement the same command language. The set of console commands available while the monitor overlay is resident is a proper subset of those console commands available while the diagnostic overlay is resident.

Although the monitor console command module is limited, it is sufficient to accommodate most operator and system programmer requirements (e.g., bootstrap, run batch, trap parity errors, etc.). It should only be necessary to load the full console command module of the diagnostic overlay when:

1. The KL10 has definitely stopped
2. Field service requires access to areas not normally of interest to operators of systems programmers.

System status available through the monitor console commands will be information the KL10 monitor stores in the communication areas. This status information consists of several software and hardware status words obtained as a result of CONI and DATAI instructions to system devices.

5.9.2 General System Bootstrap

Associated with each PDP-11 are three operator-accessible buttons located on the margin check panel. The buttons are labeled TAPE, DISK, and OPERATOR. If the appropriate tape reel is mounted and the operator presses TAPE, the bootstrap operation is initiated from the DECtape. Likewise, if the appropriate disk pack is mounted and the operator presses DISK, the bootstrap operation is initiated from the RP04/06. These two options are automatic; the system is booted without any further operator intervention at least until the KL10 monitor initiates the once-only dialogue with the operator.

If the operator depresses OPERATOR, the KY11 console switch register is referenced. In this case, the content of the switch register is used for the remaining operator options. The switch register content will determine which terminal to use for further operator interaction, whether to assume a full load or initiate an operator dialogue, or whether to use the DECtape or RP04/06 for bootstrap.

5.9.3 Diagnostic Environments

Several system environments are encountered when executing KL10 or PDP-11 based diagnostics including cases where the monitor overlay is resident in the console processor. When a KL10 diagnostic is executed in the KL10 under time-sharing, the console processor is unaware that it is a diagnostic program and offers the same terminal service provided during any time-sharing program.

Available also is a facility to do limited diagnosis of PDP-11 peripherals under time-sharing. In this case a diagnostic program will run in the KL10 and after having identified itself to the monitor will be allowed to communicate with a small kernel of code in the console processor for diagnostic execution. Note that in these two cases the monitor overlay is resident.

Those diagnostics which run in the PDP-11 and diagnose the KL10 require that the KL10 be stand-alone. These diagnostics require the diagnostic overlay and use subroutines within the overlay to accomplish common functions such as Teletype input/output, error message formatting and printing, reading and writing the various RAMs, etc.

5.9.4 Diagnostic Options

The Field Service Engineer (FSE) bootstraps the system using the same procedure as the operator. When an FSE wishes to run an PDP-11 peripheral diagnostic, he mounts an appropriate DECtape and uses the bootstrap button to boot from the DECtape. He may also start the system bootstrap on the RP04/06 and indicate he wishes to execute an PDP-11 peripheral diagnostic, at which time an appropriate diagnostic monitor is loaded from an RP04/06.

The ability to bypass sections of the load or do a partial load and examine specific areas prior to executing the next step is available during bootstrap by requesting the operator dialogue. Also available is the ability to enter the console command facility at any time during the bootstrap operation.

During diagnosis of a KL10 hardware problem, the FSE would have the diagnostic overlay resident in PDP-11 memory. Using the console command language the FSE can examine and deposit locations in KL10 memory, KL10 accumulators, and EBox registers. In addition, he may place a specific bit pattern on the diagnostic bus and cause the KL10 to execute that specific diagnostic function.

5.9.5 System Failure Procedures

Depending on the type of EBox hardware crash (i.e., EBox clock stopped) the PDP-11 will obtain some pertinent error and status information and write it onto the RP04/06.

The PDP-11 will then type out an operator message stating that a Field Service stop has occurred and allow examining of system conditions using the console functions.

On a KL10 software crash, the PDP-11 will restart the KL10 at a location (similar to location 400 in TOPS-10) in the hope that enough code is left there to allow the KL10 to dump and then reload itself.

On a PDP-11 crash, the KL10 will read PDP-11 memory, using the MB873 to execute the KD11 ROM code which outputs PDP-11 memory through the DTE20. The KL10 writes the PDP-11 memory to the RHP04 through the RH10 and then reloads the PDP-11 through the DTE20.

5.10 COMMUNICATION SOFTWARE

Until the capability of remote communications was implemented, each remote user of the DECsystem-10 had been individually linked to the computer center by separate long distance telephone lines. Also, the only device that the remote user had available at his location was the terminal; he was able to utilize available devices at the central station, but he could not obtain output other than his terminal output at his remote site. Either he had to travel to the central station to obtain a listing, or he had to have the listings delivered to him. However, with remote communications hardware and software, listing files and data can be sent via a single synchronous full-duplex line to a small remote computer. That remote computer in turn services many remote peripherals, including terminals, card readers, and line printers. This eliminates the need for the user to travel to the central site to obtain his output. The remote computer and its associated peripherals constitute a remote station.

Remote station use of the central computer is essentially the same as local use. All sharable programs and peripherals available to local users at the central computer station are also available to remote users. The remote user specifies the resources he wants to use and, if available, these resources are then allocated in the same manner as to a local user. In addition to utilizing the peripherals at the central station, the remote user can access devices located at his station or at another remote station. Local users at the central station can also make use of the peripherals at remote stations. Therefore, by specifying a station number in appropriate commands to the operation system, each user of the DECsystem-10 is given considerable flexibility in allocating system facilities and in directing input and output to the station of his choice.

The DECsystem-10 allows for simultaneous operation of multiple remote stations. Software provisions are incorporated in the operating system to differentiate one remote station from another. By utilizing peripheral devices at various stations, the user is provided with increased capabilities. For example, data can be collected from various remote stations, compiled and processed at the central station, and then the results of the processing can be sent to all contributors of the data.

Operating system commands not only allow a user to access peripherals at other remote stations, but also allow him to pretend that his job is at a remote station different from the physical station at which he is actually located. In this case, the user has a logical station and can run entire jobs from this station. With this capability, a local user at the central station could become a remote user as far as the system was concerned by changing the location of his job to a remote station in contact with a central station.

DECsystem-10 users have a wide selection of communications products to enhance or facilitate their computing needs. Within the multitask environment of TOPS-10, functionality exists for:

1. Asynchronous communication – the typical vehicle for interactive time-sharing or transaction-oriented terminals
2. Synchronous communications – for connection of remote batch, remote terminal concentration and computer-to-computer links.

The important feature of DECsystem-10 communications software is its implementation as an integral part of the TOPS-10 operating system. The monitor, not the user, handles the communications house-keeping, and all communications products are fully supported within the TOPS-10 environment. Appropriate synchronous line protocols – DIGITAL's own DDCMP or BISYNC – are supported.

Users may configure networks with simple or complex topologies utilizing the DN87 family of universal front-end equipment. The DAS80 and DAS90-series remote stations/concentrators can then be used to expand the network topologies. In addition, DECnet-10 permits the DECsystem-10 to utilize task-to-task communications with other DECsystem-10s or with any other DEC-supplied computer system which supports DECnet.

5.11 DIAGNOSTIC SOFTWARE

5.11.1 Operating System Features

TOPS-10 provides a number of features which allow the system to continue operating although some devices may be inoperable.

When errors occur on disks, tapes, memory, or the slave CPU, TOPS-10 will record the error data and notify the operator. If the error can be fixed (e.g., put a disk back on-line), the user may continue running without further delay. Alternatively, the operator may instruct the system to remove (detach) the device, in which case the jobs in error will be notified and the system will continue operation without the specified device. If the error is in a bank of memory, TOPS-10 will even attempt to move its own code (if any) out of the failing memory. In the case of disks, the system will attempt to migrate any swapping space to another unit to prevent loss of jobs.

RP04 and RP06 type disks may be dual ported so that, in case of controller failure, the packs may be accessed via an alternate path. In case of a unit failure, the pack may be moved to a working unit.

With use of appropriate I/O bus, memory bus, and device control switches, DECsystem-10s may be configured to provide redundancy of hardware. Such systems may allow any piece of hardware to be taken off-line and repaired while the system continues operation. (In some cases, it may be necessary to reload the system after a significant reconfiguration.)

5.11.2 Integrated Diagnostic Logic

The diagnostic strategy has produced a high quality of maintenance aids which enforce the increased performance and availability of a DECsystem-10. The design of the KL10 processor uses integrated diagnostic logic to enable maximum visibility and reduce "hard-core" requirements. Special software techniques are employed to maximize test coverage per diagnostic load, thereby decreasing the mean time to diagnose.

The writable control store feature of the KL10 has enabled the "microdiagnostic" alternate path methodology of testing. This technology has enabled generation of diagnostics which narrow malfunctions to the board level with a degree of confidence which exceeds industry standards.

Maintainability was attacked from a total system's concept, for not only are the CPU diagnostics capable of supporting the "module swap" philosophy of repair, the new generation of peripheral diagnostics gives the maintainer the same qualities. The programs utilize special diagnostic wrap-around logic to isolate suspected malfunctions to subsystem level. Then, through comprehensive logic analysis, not just functionality, the repair person is directed to the failing module.

5.11.3 On-Line Preventive Maintenance

To further increase system availability, special features have been incorporated into the monitor to enable on-line preventive maintenance.

Emphasis has also been placed on testing the thoroughness of detectability and the correctness of isolation. Semi-automatic physical fault insertion is qualifying and providing maturity enhancements for this product.

5.11.4 Remote Diagnosis

Another feature of the maintenance and diagnostic philosophy is the support of remote diagnosis (KLINIK). This can provide distant maintenance personnel with hardware performance statistics to improve the efficiency of service on KL10-based systems.

5.11.5 Monitor Error Reporting Programs

The purpose of this subsection is to give an overview of the monitor error reporting system and to reference the appropriate detailed documentation. This subsection is intended to be short and to be read by system programmers, system managers, operators and Field Service. They may then select which of the other documents they are interested in.

5.11.5.1 Overview of the Monitor Error Reporting System – The DECsystem-10 monitor 5.06 and later collects the following information about the operation of the system:

1. The time and reason for each monitor reload
2. Error status information on disks, drums, and core memory
3. Device performance statistics.

The reload information helps the system manager and Field Service determine the mean time between crashes and schedule reloads due to hardware and software problems. The error status information aids Field Service in fixing problems before they become serious. The performance information helps the system manager understand the service demands on his system. The same information helps Field Service determine if the reliability of the system components are up to specifications.

5.11.5.2 Hardware Error Information – The DECsystem-10 monitor controls all of the I/O devices and operates them in an efficient manner. It also follows error recovery procedures when a hardware error is detected. Frequently the data can be recovered by retrying the operation. Such a successful recovery is termed a soft error. An error which cannot be recovered is called a hard error. The monitor keeps a record of all soft and hard errors along with all the hardware status information. This information is written into a disk file by the DAEMON program for later processing by the Field Service report program, SYSERR. The importance of keeping data about the nature of a soft error cannot be overemphasized. Such information often permits Field Service to fix a device before it progresses to the hard error stage. Most soft errors are not reproducible and so are difficult to find using stand-alone diagnostics.

5.11.5.3 Reporting Programs – There are four mechanisms for obtaining monitor error information; they are messages to the operator, SYSTAT system status program, SYSDPY system dynamic display program, and DAEMON/SYSERR error logger and error report programs.

1. *Messages to the Operator* – The operator is told about memory parity errors detected by the CPU and channel on his OPR console so that he can reconfigure the memory system if errors persist. Refer to the Operator's Procedure MEMPAR.RNO in the Software Notebooks for reconfiguration instructions. Refer to Chapter 7 on monitor algorithms in the *DECsystem-10 Monitor Calls Manual* for a description of the procedures the monitor follows to keep the system running in the face of memory parity errors. The operator is also told about system stop codes on the CTY. Refer to STOPCD.RNO in the Specifications section of the Notebooks for a listing of each of the stop codes and to CRASH.RNO in the Operator's Procedure section.
2. *SYSTAT – System Status Program* – SYSTAT, the system status program, is of particular interest to operators and Field Service to get a quick idea of how the system is running at the moment. It prints out the disk error summary and performance statistics since the system was loaded and the status information about the last error on each disk drive. It does not keep a log however. Refer to the SYSTAT.RNO Specification in the Software Notebooks. The SYSTAT P command prints disk performance information.
3. *SYSDPY – System Status Display Program* – SYSDPY, the system status display program, displays essentially the same information as SYSTAT prints. However, it is updated dynamically on a display terminal. It does not keep a log.
4. *DAEMON/SYSERR – Error Logger and Report Programs* – In order to create a permanent history log, DAEMON, an operator service program, writes out the status information about each error in a file called ERROR.SYS on device SYS:. Refer to DAEMON.RNO in the Specifications section of the Software Notebooks for a description of the DAEMON program. Refer to ERROR.RNO in the Specifications section of the Software Notebooks for a description of the format of the error file, ERROR.SYS. This information can be printed at any time by the system manager or the Field Service representative. The entire file can be printed or subsets defined by data and/or type of error can be printed using a program called SYSERR. Refer to SYSERR.RNO in the Specifications section of the Software Notebooks for a description of the meaning of the output of the report program. It is written for Field Service.

At present the monitor, DAEMON, and SYSERR cooperate to log the following information:

- a. Monitor reloads including date, time, monitor name and reason for reload. Refer to the Operator's Procedure CRASH.RNO in the Software Notebooks for a description of the reasons for reloads.
- b. Memory parity errors detected by the CPU. Refer to Chapter 7 of the *DECsystem-10 Monitor Calls Manual* for a description of the monitor's actions when a memory parity error occurs. Refer to MEMPAR.RNO for operator instructions when memory parity errors occur.
- c. Errors detected by the disk and drum controllers. Refer to Chapter 7 on Algorithms in the *DECsystem-10 Monitor Calls Manual* and SYSERR.RNO in the Specifications section of the Software Notebooks for a description of the disk error recovery algorithms.

- d. Errors detected by the channel. Refer to Chapter 7 on Algorithms in the *DECsystem-10 Monitor Calls Manual* and SYSERR.RNO in the Specifications section of the Software Notebooks for a description of the disk error recovery algorithms.
- e. Disk performance statistics (not usually gathered by DAEMON V6 unless reassembled with a conditional assembly switch since file gets too long). Refer to SYSTAT.RNO and SYSERR.RNO in the Specifications section of the Software Notebooks.

5.11.5.4 Testing Programs – Because error algorithms are not exercised very frequently, extra measures must be taken to ensure that they are correct. Two test programs are supplied which test the error recovery procedures. TSTPAR is a comprehensive test and verification program for testing the memory parity error analysis and recovery of the monitor. It uses the DF10 data channel and the special (XXX) feature to write bad parity in high and low user segments, the monitor, etc. It then checks the error data stored by the monitor for correctness. Refer to the System Programming Procedure MONTST.RNO in the Software Notebooks.

TSTDSK is a disk error exerciser. It simulates disk pack errors using the ETS function of the STRUO to test hard and soft disk error recovery for all types of disk errors. Correctness of the system is determined by hand by running SYSERR. Refer to the System Programming Procedure MONTST.RNO in the Software Notebooks.

5.11.6 Diagnostic Programs

Several classes of diagnostic programs are required to support maintenance functions for a KL10-based system. They are:

1. Those that run in the PDP-11 and diagnose the PDP-11 processor and its devices
2. Those that run in the PDP-11 and diagnose the KL10 central processor
3. Those that run in the KL10 central processor and diagnose KL10 devices
4. Those that run in the PDP-8 and diagnose PDP-8 devices.

All diagnostic programs are available on DECTape and magtape CUSPS and on the Field Service KL10 diagnostic disk pack (KLAD pack). The diagnostic programs that exercise remote stations are available on paper tape.

Refer to DECsystem-10 KL10 Diagnostic Index (MD-10-DDXXA) and DECsystem-10 KL10 Diagnostic Abstract (MAINDEC-10-MD10KL) for further information.

APPENDIX A ABBREVIATIONS AND MNEMONICS

A		B	
AC	Accumulator	BASIC	Beginner's All-Purpose
AC	Action Count		Symbolic Instruction Code
ACKN	Acknowledge	BCC	Block Check Character
ACT	Action	BG	Bus Grant
AD	Adder	BOOLE	Boolean
ADA	Adder A	BR	Buffer Register
ADB	Adder B	BR	Bus Request
ADR	Address	BRK	Break
ADX	Adder Extension	BRX	Buffer Register Extension
AF	Action Flag	BSC	Binary Synchronous
AID	Algebraic Interpretive		Communication
	Dialogue	BUF	Buffer
ALGOL	Algorithmic Language		
ALT	Alternate		
ALU	Arithmetic Logic Unit		C
API	Automatic Priority Interrupt	CAM	Cache Address Mixer
APL	A Programming Language	CBUS	Channel Bus
APR	Arithmetic Processor	CCA	Cache Clearer Address
	Register	CCL	Channel Control Logic
AR	Arithmetic Register	CCW	Channel Command Word
ARL	Arithmetic Register Left	CCWF	Channel Command
ARM	Arithmetic Register Mixer		Word Fetch
ARMM	Arithmetic Register	C DIR P	Cache Directory Parity
	Mixer Mixer	CG	Carry Generate
ARR	Arithmetic Register	CH	Channel
	Right	CHA	Channel Address
ARX	Arithmetic Register	CHAN	Channel
	Extension	CHK	Check
ARXL	Arithmetic Register	CHX	Cache Extension
	Extension Left	CLK	Clock
ARXM	Arithmetic Register	CLP	Command List Pointer
	Extension Mixer	CLR	Clear
ARXR	Arithmetic Register	COBOL	Common Business Oriented
	Extension Right		Language
ASCII	American Standard Code	COMP	Complete
	for Information	CON	Control
	Interchange	COND	Condition

CONS Constant
 CONTR Control
 CP Carry Propagate
 CP Central Processor
 CPU Central Processing Unit
 CR Control RAM
 CRA Control RAM Address
 CRAM Control RAM Address Mixer
 CRC Channel RAM Control
 CRC Cyclic Redundancy Check
 CREF Cross Reference
 CRM Control RAM
 CRT Cathode Ray Tube
 CRY Carry
 CS Controller Select
 CSH Cache
 CTL Control
 CTOM Controller-to-Memory or Cache-to-Memory
 CTR Counter
 CTY Console/Command Terminal
 CUSP Commonly Used System Program
 CWSX Called With Special Execute
 CYC Cycle

D
 D Data
 DAT Data
 DAVFU Direct Access Vertical Format Unit
 DBMS Data Base Management System
 DDCMP Digital Data Communication Message Protocol
 DDT Dynamic Debugging Technique
 DIAG Diagnostic
 DIF Difference
 DIR Directory
 DIS Disable
 DISP Dispatch
 DIV Divide
 DRAM Dispatch RAM
 DS Drive Select
 DSK Disk

E
 E to T
 EBCDIC
 EBR
 EBUS
 ECL
 EDP
 EDP
 EN
 ENA
 ERR
 ERA
 EPT
 EX
 EXP
 EXT
 EXT TRA REC

F
 FE
 FE
 FLG
 FM
 FORTRAN
 FOV
 FPD
 FPD
 FUNC
 FXU

G
 GCR
 GE
 GEN
 H
 HPQ

E
 EBox Cyc
 ECL to TTL
 Extended Binary Coded Decimal Interchange Code
 Executive Base Register
 Execution Bus
 Emitter-Coupled Logic
 EBox Data Path
 Electronic Data Processing
 Enable
 Enable
 Error
 Error Address
 Executive Process Table
 Extension
 Exponent
 External
 External Transfer Receiver

F
 Function
 Floating Exponent
 Front End
 Flag
 Fast Memory
 Formula Translator
 Floating Overflow
 First Part Done
 Floating Point Divide
 Function
 Floating Exponent Underflow

G, H
 Gated
 Group Coded Recording
 Greater or Equal
 Generate
 High
 High Priority Queue

IN	I	MR	Master
INC	Input	MRU	Most Recently Used
INH	Increment	MSI	Medium Scale Integration
INSTR	Inhibit	MTR	Meter
INT	Instruction	MTTR	Mean Time to Repair
INTR	Internal	MUOU	Monitor Unimplemented User Operation
INVAL	Interrupt		
I/O	Invalid		
IOT	Input/Output		N
	Input/Output		Next Instruction
IPCF	Transfer	NICOND	Condition
	Interprocessor		Non Processor Request
	Communications Facility	NPR	Non-Return-To-Zero Inverter
IR	Instruction Register	NRZI	Non-Existent Memory
ITPS	In-House Text Preparation System	NXM	Next
		NXT	
			O
		OCR	Optical Character Reader
		OK	O11 Korrekt
J	J, K, L	OP	Operation (code)
KLINIK	Jump	OPSER	Operator Service
	KL10 Integrated Network for Investigation and Korrection	OVN	Overrun
			P
L	Low	PA	Physical Address
LINED	Line Editor	PAG	Pager
LPC	Longitudinal Parity Check	PAR	Parity
LRC	Longitudinal Redundancy Check	PC	Program Counter
		PCF#	Previous Context Flags from Number
LRU	Least Recently Used	PCP	Previous Context Public
LSI	Large Scale Integration	PDP	Programmed Data Processor
LUUO	Local Unimplemented User Operation	PE	Phase Encoded
		PERF	Performance
		PF	Page Fault
MB	M	PGRF	Page Refill
MBC	Memory Buffer	PI	Priority Interrupt
MBC	Massbus Control	PIA	Priority Interrupt Assignment
MBX	MBox Control		Process Identifier
MBZ	MBox Control	PID	Priority Interrupt Hold
MCL	Memory Control	PIH	Peripheral Interchange Program
MCS	Message Control System		Physical Memory Address
MEM	Memory	PIP	Physical Memory Address Selector
MFD	Master File Directory		Previous
MHz	Megahertz	PMA	Page Table/Process Table
MIX	Mixer	PMA	Pointer
MONGEN	Monitor Generator		Power
MQ	Multiplier Quotient	PREV	
MQM	Multiplier Quotient Mixer	PT	
		PTR	
		PWR	

R
RAM Random Access Memory
RD Read
RE Receive ECL
REC Receive
REF Reference
REG Register
REL Release
REQ Request
RES Reset
RESP Response
RET Return
RIP Request In Progress
RJE Remote Job Entry
ROM Read Only Memory
RQ Request

S
S ADR P Storage Address
Parity
SBR Subroutine
SBUS Storage Bus
SC Shift Count
SCAD Shift Count Adder
SCADA Shift Count Adder A
SCADB Shift Count Adder B
SCD Shift Count Adder
SCM Shift Count Mixer
SEL Select
SFD Subfile Directory
SH Shifter
SHRT Shift Right
SIM Simulate
SOUP Software Updating Package
SP Special
SPEC Special
SR State Register
ST Start
SYNC Synchronize

T, U
T to E
TE Transmit ECL
T Time
TECO Text Editor and Corrector
TIM Timer
TOPS Total Operating System
TRA Transfer
TTL Transistor-Transistor
Logic
UBR User Base Register
UCODE Microcode
UFD User File Directory
UPT User Process Table
UUO Unimplemented User
Operation

V, W, X, Y, Z
VAL Valid
VMA Virtual Memory
Address
VRC Vertical Redundancy Check
XFER Transfer
XR Index Register
WARN Warning
WC Word Count
WD Word
WR Write

INDEX

- A**
- Abbreviations, A-1
 - Access Time, 1-6
 - Accounting Meters, 3-17
 - Accounting Meters, 4-27
 - Accumulators, 3-8
 - Address Calculation, 3-8
 - Adder, 4-10
 - Adder Extension, 4-10
 - Address
 - Physical Memory, 3-10
 - Virtual, 3-10
 - Address Manipulation Path, 4-10
 - Address Mapping, 3-10
 - Addressing
 - Immediate, 3-2
 - Indexing, 3-2
 - Indirect, 3-2
 - AID Language, 5-17
 - ALGOL Compiler, 5-14
 - Algorithmic Language, 5-14
 - API Function Word, 3-17
 - APL Language, 5-17
 - Application Processes, 5-22
 - APR Device, 4-40
 - Arithmetic Adder, 4-10
 - Arithmetic Logic Unit, 4-10
 - Arithmetic Register, 4-10
 - Arithmetic Register Extension, 4-10
 - Assemblers, 5-13
 - Assembler, MACRO, 5-14
 - Asynchronous Communications, 4-56
 - Asynchronous Interfaces, 4-61
 - Asynchronous Line Interface
 - DL11-C, 4-43
 - DL11-E, 4-43
 - Asynchronous 16-Line
 - Multiplexer, DH11, 4-64
 - Automatic Priority Interrupt
 - System Function Word, 3-17
 - Automatic Restart, 3-18
- B**
- BA10 Unit Record Equipment, 4-53
 - BASIC Compiler, 5-15
 - Batch, Operator Intervention, 2-8
 - Batch Controller, 2-6, 2-7
 - Batch Processing, 1-1
 - Batch Queue
 - Input, 2-7
 - Output, 2-7
 - Batch Software, 2-6
 - BC11-A Unibus, 4-43
 - Bisync Protocol, 4-58
 - BLK PI Instruction, 4-29
 - BLKI Instruction, 4-24, 4-31
 - BLKO Instruction, 4-24
 - Block, 1-6
 - Block Check Characters, 4-62
 - BM873 ROM Loader Module, 4-42
 - Bootstrap, 4-42
 - Bootstrap Facility, 4-41
 - Bootstrap Functions, 4-46
 - Bootstrap, System, 5-25
 - Buffer Register, 4-10
 - Buffer Register Extension, 4-10
 - Byte Transfer Function, 4-37
 - Byte Transfer Operations, 4-45
- C**
- Cache, 4-28
 - Cache Memory, 3-9, 4-48
 - Card Punch, CP10-D, 4-55
 - Card Reader, CR10, 4-54
 - CBus, 4-35
 - CCA Device, 4-40
 - Central Processor, 4-1
 - Central Processor Interrupt Facility, 4-39
 - Central Processor Trap Facility, 4-40
 - Central Processor, KD11-A, 4-42
 - Central Processors (KA10, KI10, KL10), 3-1
 - Channel Command List, 4-14
 - Channel Command Register, Lookahead, 3-15

- Channel Controller, 3-15
- Channel Errors, 5-30
- Channel I/O, 3-13
- Channel I/O Processor, 4-28, 4-49
- Channel Multiplexer, 3-14
- Channel Processor, 3-13
- Channel Program, 3-15
- Channel Transfers, 5-6
- Channels, 3-1
- Channels
 - External, 3-14
 - Internal, 3-14
- Clock, 1-2
- Clock
 - Master, 4-10
 - Programmable, 3-17
- Clock Control Function, Diagnostic, 4-37
- COBOL Compiler, 5-15
- Code, impure, 1-4
- Code
 - Protection, 2-5
 - Pure, 1-4
- Command Control Language, 5-7
- Command Decoder, 5-3
- Command Facility, Console, 4-41
- Command Language, 2-5
- Command List, 5-6
- Command List, Channel, 3-14
- Command Module, Console, 5-24
- Command Terminals, 2-4
- Communication, 1-5
- Communication Protocols, 4-56
- Communication Software, 5-26
- Communication Subsystem, 4-56
- Communications
 - Interjob, 5-12
 - Interprocess, 5-12
- Compilers, 5-13, 5-14
 - ALGOL, 5-14
 - BASIC, 5-15
 - COBOL, 5-15
 - FORTRAN, 5-16
- Compute Bound, 1-3
- Computing, Multimode, 2-4
- Configuration
 - DECsystem-10, 3-1
 - System, 2-2, 4-1
- CONI Instruction, 4-24
- CONO Instruction, 4-24
- CONSI Instruction, 4-24
- CONSO Instruction, 4-24
- Console Command Facility, 4-41
- Console Command Module, 5-24
- Console Computer, 3-18
- Console Functions, 4-46
- Console Processor, 4-41, 5-23
- Console Processor Functions, 4-44
- Console Software, 5-23
- Context AC Blocks, 4-22
- Context Switching, 1-2
- Control File, 2-8
- Control RAM, 4-22
- Control Routine, 5-5
- Controllers, I/O, 4-41
- Core Memory, 3-8
- Core Utilization, 2-5
- CPL Language, 5-18
- CPU
 - Model A, 4-10
 - Model B, 4-10
- CP10-D Card Punch, 4-55
- CR10 Card Reader, 4-54
- CREF Utility, 5-19
- Cross Reference Listing, 5-19
- Cyclic Redundancy Check, 4-62

D

- DAS80 Series Remote Station, 4-67
- DAS92 Remote Station, 4-69
- Data Base Management System, 5-20
- Data Fetch Manager, 4-25
- Data Path, 4-10
- Data Set, 1-5, 4-60
- Data Storage Manager, 4-25
- DATAI Instruction, 4-24
- DATAO Instruction, 4-24
- DATAO PAG Instruction, 4-22
- DBMS-10 Utility, 5-20
- DC72-NP Remote Station, 4-66
- DC75-NP Communication Subsystem, 4-60
- DC76 Communication Subsystem, 4-59
- DDCMP, 4-58
- DDT Utility, 5-20
- Debugging, Program, 5-20
- DECnet, 4-61
- DECsync Protocol, 4-58
- DECsystem-10 Configurations, 3-1
- DECsystem-10 Primer, 2-1
- DECsystem-1080, 4-1
- DECsystem-1080 Configuration, 2-2
- DECsystem-1090, 4-1
- DECsystem-1090 Configuration, 2-3
- Deposit Operations, 4-44
- Deposit Function, 4-36

- Device
 - APR, 4-40
 - CCA, 4-40
 - Internal, 4-40
 - I/O, 4-41
 - MTR, 4-40
 - PAG, 4-40
 - PDP-11, 4-42
 - Peripheral, 2-5
 - PI, 4-40
 - Private, 1-6
 - Public, 1-6
 - Real Time, 2-9
 - Removable Storage, 1-7
 - TIM, 4-40
 - Device Service Routines, 5-5
 - DH11 Asynchronous 16-Line Multiplexer, 4-64
 - DIA20, 4-39
 - Diagnosis, Remote, 5-28
 - Diagnostic Bus, 3-18, 4-46
 - Diagnostic Clock Control Function, 4-37
 - Diagnostic Computer, 3-18
 - Diagnostic Control Logic, 4-10
 - Diagnostic Cycle, SBus, 4-31
 - Diagnostic EBox Control Functions, 4-37
 - Diagnostic Environment, 5-25
 - Diagnostic Facilities, 4-37
 - Diagnostic Features, Operating System, 5-27
 - Diagnostic Functions, 4-46
 - Diagnostic Load Functions, 4-37
 - Diagnostic Logic, 5-27
 - Diagnostic Options, 5-25
 - Diagnostic Programs, 5-30
 - Diagnostic Read Functions, 4-37
 - Diagnostic Software, 1-8, 5-27
 - Diagnostics, Remote, 5-28
 - Direct I/O, 3-13
 - Directory
 - Master File, 1-6
 - User's File, 1-6
 - Disk Errors, 5-29
 - Disk File System, RJP04/06, 4-43
 - Disk Performance Statistics, 5-30
 - Disk Quotas, 5-9
 - Disk Storage Management, 5-10
 - Disk Subsystem, 4-50
 - Dispatch RAM, 4-22
 - DL10 Communications Channel Control, 4-57
 - DL11-C Asynchronous Line Interface, 4-43
 - DL11-E Asynchronous Line Interface, 4-43
 - DM11-BB Modem Control Unit, 4-65
 - DMA20, 4-29, 4-31
 - DN80 Series Remote Station, 4-67
 - DN87 Universal Front-End, 4-61
 - DN87S Universal Front-End, 4-61
 - Doorbell Feature, 4-45
 - Doorbell Function, 4-37
 - DQ11 Synchronous Line Interface, 4-65
 - DS11 Multiple Line Synchronous Interface, 4-64
 - DTE20, 4-36
 - DTE20 Operating Modes, 4-46
 - DTE20 Ten-Eleven Interface, 4-43, 4-57
 - Dump Operations, 4-42
 - Duplex Channels, 4-58
- E**
- EBox, 4-10
 - EBox Control Function, Diagnostic, 4-37
 - EBox Request, 4-24
 - EBus, 3-13, 4-31
 - EBus Control Logic, 4-24
 - EBus High Priority Interrupt Operation, 4-35
 - EBus Input Operation, 4-33
 - EBus Output Operation, 4-32
 - EBus Programmable Interrupt Operation, 4-34
 - Editor
 - LINED, 5-18
 - RUNOFF, 5-19
 - SOUP, 5-19
 - TECO, 5-18
 - Editors, 5-18
 - Effective Address, Calculation, 4-23
 - Effective Address Manager, 4-25
 - E/M Interface, 4-29
 - Error Information, Hardware, 5-28
 - Error Logger, 5-29
 - Error Recovery, 2-8
 - Error Report Program, 5-29
 - Error Reporting, Monitor, 5-28
 - Errors
 - Channel, 5-30
 - Disk, 5-29
 - Examine Function, 4-36
 - Examine Operations, 4-44
 - External Memory, 4-29
 - Executive Mode, 1-6, 3-9
 - Executive Process Table, 3-10
 - Executive Program, 1-2
 - Executor Routine, 4-25
- F**
- FAILSAFE/BACKUP Utility, 5-20
 - Failure Procedures, 5-26
 - Fast Memory, 4-48
 - Features, System, 3-1
 - Field Service Report Program, 5-28

File
 Control, 2-8
 Log, 2-6
 Saving, 5-20
 File Directories, 5-8
 File Handler, 5-8
 File Handling, 1-6
 File Operations, 5-10
 File Protection, 5-9
 File Protection Codes, 5-8
 File Storage, 4-49
 File Structures, 5-9
 File System, 5-7
 File System, Mass Storage, 2-5
 Filename, 5-8
 Files, 2-5
 FILEX Utility, 5-20
 Filing System, 1-6
 Firmware, 4-25
 Floating Exponent Register, 4-22
 FORTRAN Compiler, 5-16
 Front-End Privileged, 4-46
 Front-End Restricted, 4-46
 Full Duplex, 4-58

G

GALAXY-10, 2-6
 General Register Blocks, 4-22

H

Half Duplex, 4-58
 Halt Handler, 4-27
 Handler
 Page Fault, 2-6
 Service Request, 2-3
 Hardware, 2-1, 4-1

I

Index Registers, 3-8
 Indexing, 3-2
 Input/Output, 1-6, 5-10
 Input/Output Handler, 4-27
 Input/Output Processor, 1-3
 Input Spooler, 2-7
 Instruction Execution Unit, 4-10
 Instruction Format, 3-6
 Instruction Modifiers, 3-2
 Instruction Prefetch, 4-23
 Instruction Register, 4-22
 Instruction Set Constructs, 3-3
 Instructions
 DECsystem-10, 3-1
 KL10, 4-41
 PDP-11, 4-41

Interactive Languages, 1-4
 Interactive Programs, 1-2
 Interactive Terminals, 2-4
 Interleaving Modes, 4-29
 Internal Devices, 4-40
 Interpreters, 5-13, 5-17
 Interpreters
 AID, 5-17
 APL, 5-17
 CPL, 5-18
 Interprocessor Communication, 4-47
 Interprocessor Communication Areas, 4-47
 Interprocessor Interrupts, 4-45
 Interprocessor Interrupt Facility, 4-37
 Interprocessor Queue Processing, 4-47
 Interrupts, Interprocessor, 4-45
 Interrupt Facility
 Central Processor, 4-39
 Interprocessor, 4-37
 Interrupt System, 3-16
 Interval Timer, 3-17, 4-28

I/O

 Channel, 3-13
 Direct, 3-13
 I/O Bound, 1-3
 I/O Bus Adapter, 4-39
 I/O Controllers, 4-31, 4-41
 I/O Devices, 4-41
 I/O Operations
 Priority Interrupt, 4-24
 Programmed, 4-24
 I/O Processor, Channel, 4-49
 I/O Programming, 5-5
 ITPS-10 Utility, 5-21

J

Jobs
 Locking, 2-9
 User, 2-6

K

KD11-A Central Processor, 4-42
 Keyboard Terminal, LA36, 4-43
 KG11-A XOR and CRC Unit, 4-62
 KJOB-LOGOUT Program, 5-23
 KW11-L Line Clock Option, 4-42
 KY11-D Programmer's Console, 4-42

L

LA36 Keyboard Terminal, 4-43

Language

- AID, 5-17
- ALGOL, 5-14
- APL, 5-17
- BASIC, 5-15
- COBOL, 5-15
- Command, 2-5
- Command Control, 5-7
- CPL, 5-18
- FORTTRAN, 5-16

Languages, 2-1

Languages, Interactive, 1-4

Latency Optimization, 1-6

Line Clock Option, KW11-L, 4-42

Line Multiplexer, 1-5

Line Printer

- LP07, 4-55
- LP10, 4-54

Line Printer Subsystem, LP100, 4-55

Line Scanner, 1-5

LINED Editor, 5-18

LINK-10 Utility, 5-21

Loader, Linking, 5-21

Loader Module (ROM), BM873, 4-42

Log File, 2-6

Logic,

- ECL, 4-10
- TTL, 4-10

LOGIN Program, 5-23

Longitudinal Redundancy Check, 4-62

LP07 Line Printer, 4-55

LP100 Line Printer Subsystem, 4-55

LP10 Line Printer, 4-54

M

Machine Instructions, 4-41

MACRO Assembler, 5-14

Magnetic Tape Subsystem, 4-52

Magnetic Tape System, TC11-G, 4-43

Main Memory, 1-2

Main Memory Subsystem, 4-48

Maintainability, 5-28

Massbus Controller, 3-15, 4-38

Mass Storage Devices, 5-7

Mass Storage File System, 2-5

Master Clock, 4-10

Master File Directory, 1-6

MBox, 4-27

MBox Read Operation, 4-29

MBox Read-Pause-Write Operation, 4-30

MBox Write Operation, 4-30

MCS-10 Utility, 5-22

Memory Read, 4-49

Memory Address Mapping, 3-10

Memory Blocks, 1-3

Memory Bus Adapter, 4-29, 4-31

Memory

- Cache, 4-48
- External, 4-29
- Fast, 4-48
- Main, 4-48
- Management, 5-11
- MF11-UP, 4-42
- Parity Errors, 5-29
- Protection and Relocation, 3-10
- Read-Pause-Write, 4-49
- Request Logic, 4-24
- Secondary, 4-49, 5-11
- System, 3-8
- Virtual, 2-6, 5-1
- Write, 4-49

Message Control System, 5-22

Messages to Operator, 5-29

Meter Board, 4-28

MF10 Memory, 3-9, 4-29, 4-48

MF11-UP Memory, 4-42

MG10 Memory, 3-9, 4-29, 4-48

MH10 Memory, 3-9, 4-29, 4-48

Microcode, 4-22, 4-25

MM11-UP Memory, 4-42

Mnemonics, A-1

Mode,

- Executive, 1-6, 3-9
- Switch, 1-6
- User, 1-6, 3-9

Modem Control Omit, DM11-BB, 4-65

Modems, 1-5, 4-59

Modes, Processor, 3-9

MONGEN Program, 5-23

Monitor Calls, 1-6

Monitor Error Reporting Programs, 5-28

Monitor Generator Program, 5-23

Monitor Overhead, 1-4

Monitor Reloads, 5-29

Monitor Support Programs, 5-23

MTR Device, 4-40

Multiprocessing Systems, 5-12

Multiplexer, Channel, 3-14

Multimode Computing, 2-4

Multiplier Quotient Register, 4-22

Multiprogramming, 1-2

N

Network, 5-22, 5-27

Non-Resident Software, 2-4

Number System, 3-6

O

Operand, Fetch and Store, 4-23
Operating System, 1-2, 2-1, 2-3
Operating System Diagnostic Features, 5-27
Operation System, Resident, 5-1
Operator Intervention, Batch, 2-8
Operator, Messages, 5-29
Operator Service Program, 5-28
OPSER Program, 5-23
Output Spooler, 2-7
Overhead, Monitor, 1-4
Overlapping, I/O Operations, 1-6

P

PAG Device, 4-40
Page Fault Handler, 2-6, 4-27, 5-11
Pager, 3-10, 4-27
Parity Errors, Memory, 5-29
PDP-11 Devices, 4-42
Performance Analysis Counter, 3-27, 4-29
Peripheral Device Assignment, 5-10
Peripheral Devices, 2-5
Peripheral Interchange Program, 5-22
Physical Memory Address, 3-10
PI Device, 4-40
PIP Utility, 5-22
Plotter, XY10, 4-55
Portals, 3-10
Power Fail, 4-42
Power Outage, 3-18
Priority Interrupt Handler, 4-27
Priority Interrupt System, 1-6, 3-16
Privileged Front-End, 4-46
Process Tables, 3-10
Processes, Application, 5-22
Processor, Console, 4-41, 5-23
Processor Modes, 3-9
Processor Status Register, 4-10
Program Counter, 4-10
Program Debugging, 5-10
Program,
 Channel, 3-15
 Error Logger, 5-29
 Error Report, 5-29
 Field Service Report, 5-28
 Interactive, 1-2
 KSOB-LOGOUT, 5-23
 LOGIN, 5-23
 Monitor Generator, 5-23
 Monitor Error Reporting, 5-28
 Operator Service, 5-23
 Peripheral Interchange, 5-22
 Reentrant, 2-6
 System Status, 5-29

Programmable Clocks, 3-27
Programmed Operator, 1-6, 3-2, 5-6
Programmer's Console, KY11-D, 4-42
Programs,
 Diagnostic, 5-20
 Monitor Support, 5-22
 Test, 5-30
 User, 5-11
Protection Codes, 2-5
Protocol, 4-58, 4-56
Protocol, Communications, 5-24

Q

Queue,
 Job, 1-4
 Run, 2-10
Queue Manager and Scheduler, 2-6

R

Random Access Memory,
 Control, 4-22
 Dispatch, 4-22
Real Time, 2-9
Real Time Device, 2-9
Real Time User, 5-12
Record, 1-6
Reentrant Program, 2-6
Reentrant Software, 1-4, 5-11
Register Blocks, 3-13
Register Blocks, General Purpose, 3-8
Reliability, 1-7
Remote Diagnosis, 5-28
Remote Station, 4-66, 4-67, 4-69, 5-26
Remote User, 5-26
Resource Allocator, Sharable, 2-3
Restart,
 Automatic, 3-18
 Manual, 3-18
Restricted Front-End, 4-46
RH10 Massbus Controller, 4-51
RH11 Massbus Controller, 4-51
RH20, 4-38
RH20 Massbus Controller, 4-51
RHP04/06 Disk Subsystem, 4-50
RHS04 Disk Subsystem, 4-51
RP04/06 Disk Drive, 4-51
RJP04/06 Disk File System, 4-43
RTP04/06 Disk Subsystem, 4-50
Round Robin, 1-2
RUNOFF Editor, 5-19

S

Saving Files, 5-20
 SBDIAG Instruction, 4-29
 SBus, 4-29
 SBus Diagnostic Cycle, 4-31
 SBus Read Operation, 4-29
 SBus Read-Pause-Write Operation, 4-30
 SBus Write Operation, 4-30
 Scheduler, 5-3
 Scheduling Algorithm, 1-3
 Scheduling, Dynamic, 1-3
 Secondary Memory, 1-2, 4-49
 Sensors, Temperature, 3-18
 Service Request Handler, 2-3
 Shift Count Adder, 4-22
 Shift Count Register, 4-22
 Shifter, 4-11
 Software,
 Batch, 2-6
 Communication, 5-26
 Console, 5-23
 Diagnostic, 1-8, 5-27
 Nonresident, 2-4, 5-13
 Reentrant, 1-4, 5-11
 Software Sharing, 1-4
 Software Updating, 5-19
 SOUP Editor, 5-19
 Spooler,
 Input, 2-7
 Output, 2-7
 Spooling, 1-7, 2-5, 5-10
 Start-up and Stop Interface, 4-25
 Statistics, Disk Performance, 5-30
 Storage Controller, 4-27
 Storage, Mass, 2-5
 Swapper, 5-4
 Swapping, 1-2, 2-6
 Swapping Disk Subsystem, 4-51
 Swapping System, 4-49
 Synchronous Communications, 4-56
 Synchronous Interfaces, 4-26
 Synchronous Line Interface
 DQ11, 4-65
 DS11, 4-64
 System Bootstrap, 5-25
 System Configuration, 2-2
 System Features, 3-1
 System Queue Manager and Scheduler, 2-6
 System Status Program, 5-29

T

Table
 Executive Process, 3-10
 Process, 3-10
 User Process, 3-10
 TC11-G Magnetic Tape System, 4-43
 TD10 Tape Control, 4-53
 TECO Editor, 5-18
 Temperature Sensors, 3-18
 Ten-Eleven Data Interface, 4-36
 Ten-Eleven Interface, DTE20, 4-43
 Terminals, 4-59
 Terminals, Command, 2-4
 Terminals, Interactive, 2-4
 Test Programs, 5-30
 THU16 Magnetic Tape Subsystem, 4-53
 TIM Device, 4-40
 Time Base, 3-17, 4-28
 Time-sharing, 1-2, 2-4
 Time-sharing, Principles of, 1-1
 Time Slice, 1-2
 TM02 Master Tape Control, 4-53
 TM03 Master Tape Control, 4-53
 Transaction Processing, 5-22
 Translators, 5-13
 Translators, Logic Level, 4-10
 Trap Facility, 3-17
 Trap Facility, Central Processor, 4-40
 Trap Handling, 3-6
 TU16 Magnetic Tape Transport, 4-53
 TU56 DECtape Subsystem, 4-53
 TU70-72 Magnetic Tape Subsystem, 4-52
 TX01 Controller, 4-53
 TX02 Controller, 4-53

U

Unibus, BC11-A, 4-43
 Unibus Transfers, 4-44
 Unimplemented User Operations, 3-6
 Unit Record Equipment, 4-53
 Updating, Software, 5-29
 User File Directory, 1-6
 User Jobs, 2-6
 User Mode, 1-6, 3-9
 User Page Map, 3-10
 User Programs, 5-11
 User Process Table, 3-10
 User, Remote, 5-26
 Utilities, 2-1, 5-19

Utility,

CREF, 5-19

DBMS-10, 5-20

DDT, 5-20

FAILSAFE/BACKUP, 5-20

FILEX, 5-20

ITPS-10, 5-21

LINK-10, 5-21

MCS-10, 5-22

PIP, 5-22

UUO Handler, 5-5

V, W, X, Y, Z

Virtual Address, 3-10

Virtual Memory, 2-6, 5-11

Virtual Memory Address Adder, 4-10

Virtual Memory Address Register, 4-10

XOR and CRC Unit, KG11-A, 4-62

XY10 Plotter, 4-55

Reader's Comments

DECsystem-1080/1090
System Description
EK-1080U-SD-003

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 do you find with the manual? _____

Does this manual satisfy the need you think it was intended to satisfy? _____
Does it satisfy *your* needs? _____ Why? _____

Would you please indicate any factual errors you have found. _____

Please describe your position. _____

Name _____ Organization _____

Street _____ Department _____

City _____ State _____ Zip or Country _____

Fold Here

Do Not Tear - Fold Here and Staple

**FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.**

**BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES**

Postage will be paid by:

**Digital Equipment Corporation
Technical Documentation Department
Maynard, Massachusetts 01754**

