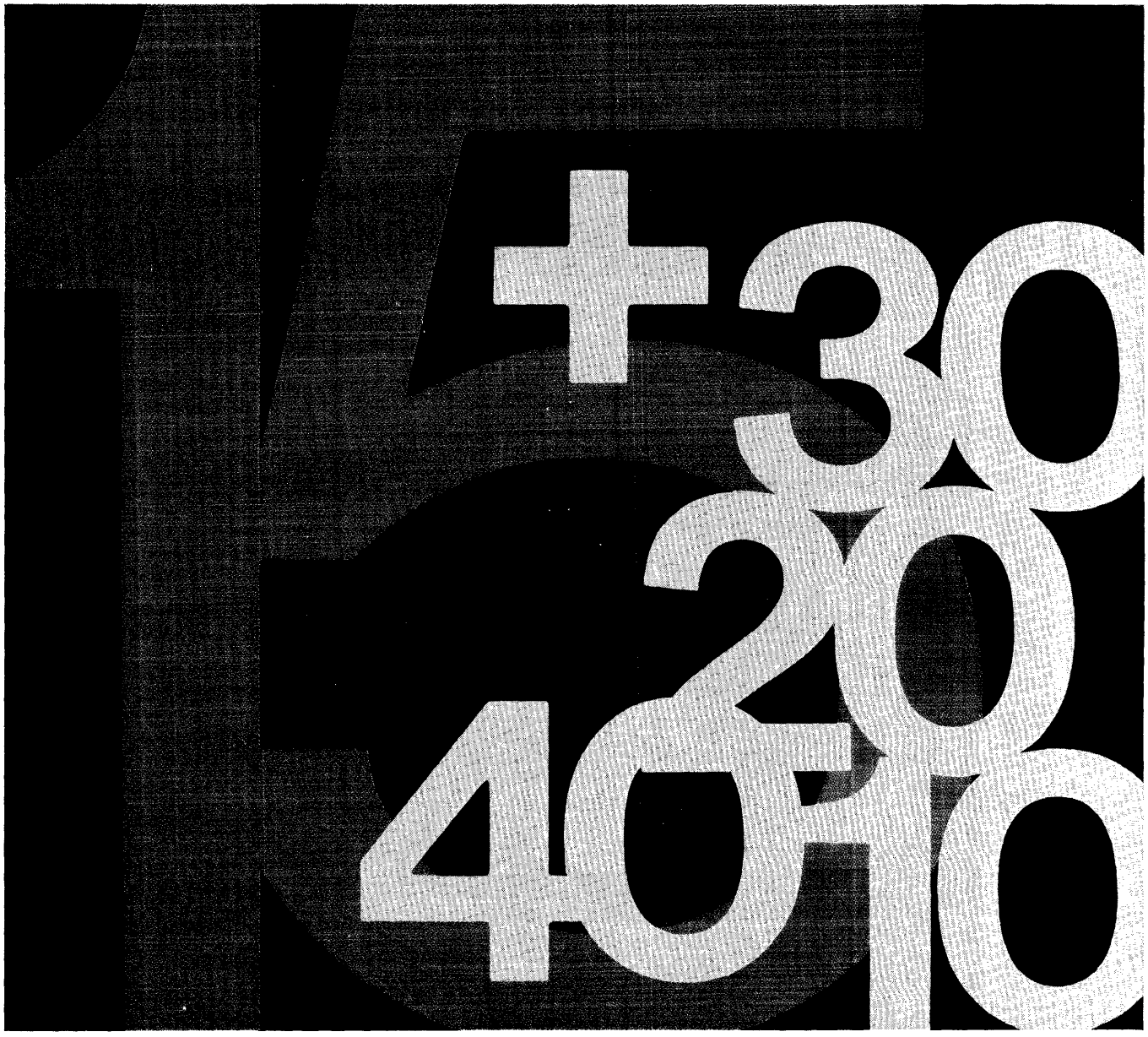
Digital Equipment Corporation
Maynard, Massachusetts

digital

PDP-15 Systems

User's Handbook Vol. 1 Processor



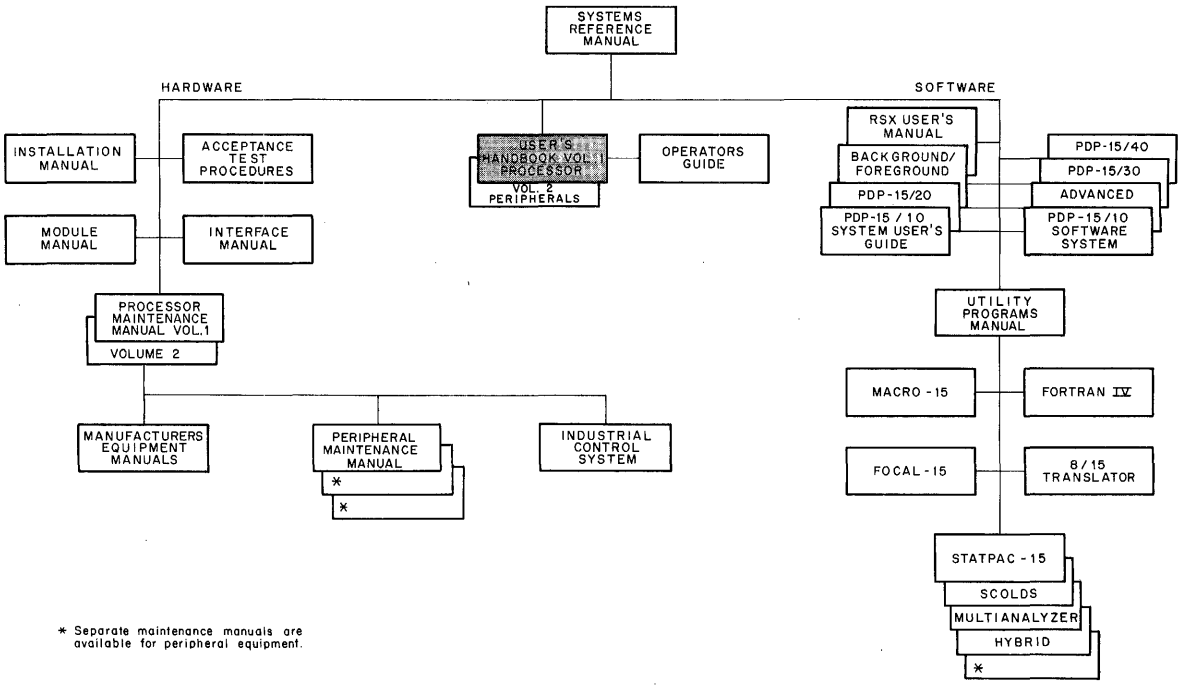
DEC-15-H2DC-B

PDP-15 SYSTEMS
USER'S HANDBOOK
VOLUME 1 PROCESSOR

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Edition, September 1970
2nd Printing (Rev) November 1970
3rd Printing (Rev) April 1971
4th Printing, June 1973

PDP-15 FAMILY OF MANUALS



* Separate maintenance manuals are available for peripheral equipment.

SYSTEMS REFERENCE MANUAL – Provides overview of PDP-15 hardware and software systems and options, instruction repertoire, expansion features, and descriptions of system peripherals. (DEC-15-BRZC-D)

USER'S HANDBOOK VOLUME 1, PROCESSOR – Principal guide to system hardware includes system and subsystem features, functional descriptions, machine-language programming considerations, instruction repertoire, and system expansion data. (DEC-15-H2DC-D)

VOLUME 2, PERIPHERALS – Features functional descriptions and programming considerations of peripheral devices. (DEC-15-H2DC-D)

OPERATOR'S GUIDE – Lists procedural data, including operator maintenance, for using the operator's console and the peripheral devices associated with PDP-15 Systems. (DEC-15-H2CB-D)

PDP-15/10 SYSTEM USER'S GUIDE – Features COMPACT and Basic I/O Monitor operating procedures. (DEC-15-GG1A-D)

PDP-15/20 SYSTEM USER'S GUIDE – Lists Advanced Monitor System operating procedures. (DEC-15-MG2B-D)

BACKGROUND/FOREGROUND MONITOR SYSTEM USER'S GUIDE – Lists operating procedures for the DECtape and disk-oriented Background/Foreground monitors. (DEC-15-MG3A-D)

PDP-15/10 SOFTWARE SYSTEM – Describes COMPACT software system and Basic I/O Monitor System. (DEC-15-GR1A-D)

PDP-15/20/30/40 ADVANCED MONITOR SOFTWARE SYSTEM – Describes Advanced Monitor System; programs include system monitor language, utility, and application types; operation, core organization, and input/output operations within the monitor environment are discussed. (DEC-15-MR2B-D)

PDP-15/30/40 BACKGROUND/FOREGROUND MONITOR SOFTWARE SYSTEM – Describes Background/Foreground Software System including the associated language, utility, and applications program. (DEC-15-MR3A-D)

RSX USER'S MANUAL – Describes the disk-oriented real time system executive language and applications.

MAINTENANCE MANUAL VOLUME 1, PROCESSOR – Provides block diagram and functional theory of operation of the processor logic; lists preventive and corrective maintenance data. (DEC-15-H2BB-D)

VOLUME 2, ENGINEERING DRAWINGS – Provides engineering drawings and signal glossary for the basic processor and options. (DEC-15-H2BB-D)

INSTALLATION MANUAL – Provides power specifications, environmental considerations, cabling, and other information pertinent to installing PDP-15 Systems. (DEC-15-H2AB-D)

ACCEPTANCE TEST PROCEDURES – Lists step-by-step procedures designed to insure optimum PDP-15 Systems operation.

PDP-15 MODULE MANUAL – Provides characteristics, specifications, timing and functional descriptions of modules used in PDP-15 Systems. (DEC-15-H2EA-D)

INTERFACE MANUAL – Provides information for interfacing devices to a PDP-15 System. (DEC-15-H0AB-D)

UTILITY PROGRAMS MANUAL – Provides utility programs common to PDP-15 Monitor systems. (DEC-15-YWZA-D)

MACRO-15 – Provides MACRO assembly language for the PDP-15. (DEC-15-AMZA-D)

FORTRAN IV – Describes PDP-15 version of the FORTRAN IV compiler language. (DEC-15-KFZB-D)

FOCAL-15 – Describes an algebraic interactive compiler level language developed by Digital Equipment Corporation. (DEC-15-KJZB-D)

CONTENTS

CONTENTS (Cont)

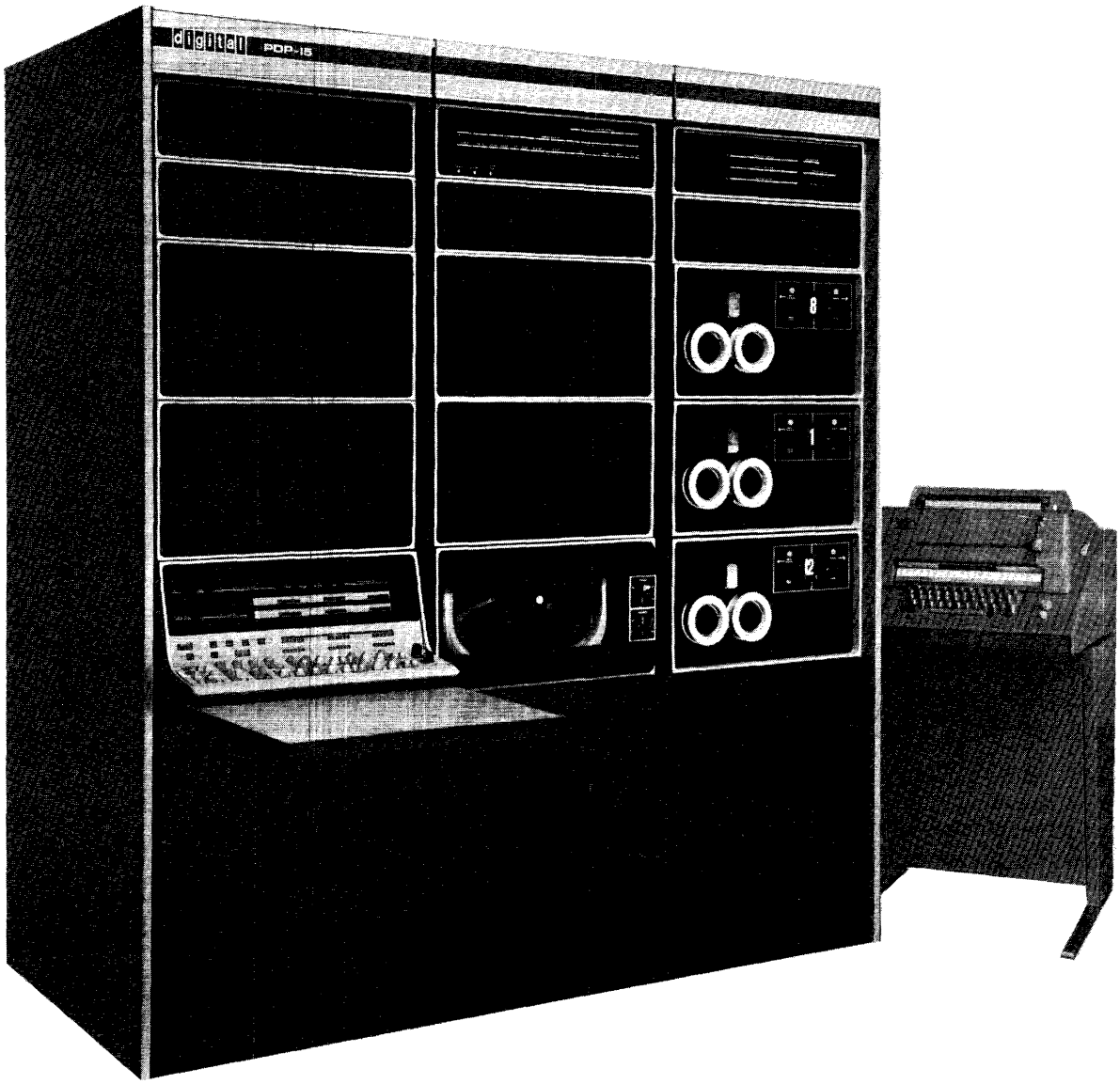
CONTENTS (Cont)

ILLUSTRATIONS

TABLES

PREFACE

The PDP-15 Users Handbook is the principal guide to the PDP-15 hardware. This manual is presented in two volumes: Volume 1 PROCESSOR and Volume 2 PERIPHERALS. The PDP-15 Users Handbook includes system features and specifications, functional descriptions, machine language programming considerations, and a detailed description of the instruction repertoire. The chart and table on pages iii and iv show the relationships of the other PDP-15 system documentation and give abstracts of their contents.


digital
PDP-15

Chapter 1
System Description

1.1 SYSTEM SOFTWARE

1.1.1 Introduction

The PDP-15 System is divided into a number of configurations; each configuration having a powerful software package available. These software packages are designed to service the needs of a particular system configuration.

The PDP-15/10 configuration software is governed by the COMPACT Software System, a complete package including Assembler, Editor, Octal Debugging Technique, and mathematical and utility routines, all designed to function in 4K or 8K systems.

COMPACT Software System

- Assembler
- Editor
- ODT (Octal Debugging Technique)
- Math Package
- Utility routines:
  - Hardware Read-in Mode (HRM) punch routine
  - Paper tape handling routines
  - Teletype I/O routines
  - Octal dump routine
  - Memory scan routine

For PDP-15/10 Systems equipped with DECtape, the FAST (Fast Acquistion of System Tape) System is provided to retrieve frequently-used programs from DECtape.

Installations with a minimum of 8K words of core memory and a high-speed paper tape reader/punch can use the Basic I/O Monitor to extend system capabilities. The PDP-15/20 Advanced Monitor

System operates from mass storage devices (DECtape or DECdisk) and is device independent; consequently programs need not be limited to the use of certain specified I/O devices.

Simple I/O statements control data handling; selection of physical devices is determined at load time on the actual machine, not when the program is written. Real-time I/O level subroutines can easily be integrated into the system as new devices are added.

1.1.2 PDP-15/20 Advanced Monitor System

The PDP-15/20 Advanced Monitor is used for batch processing. In the primary (keyboard) mode, the user has interactive access to a large set of system programs to facilitate program development and testing.

All Advanced Monitor functions, as well as the many available system software routines, are specially designed to make the system as accessible as possible to users who want "hands-on" interaction; at the same time, routine elements of programming can be handled simply and easily.

PDP-15/20 Advanced Monitor System

- Keyboard Monitor
  - Teletype handler
  - Command decoder
  - Input/Output Programming System (IOPS) data handling, device handling, and interrupt routines
  - Real-time clock handler
  - Error detector program
  - Device assignment tables
  - Batch processor (paper tape or card control)
- FORTRAN IV
- SYSTEM LOADER
- FOCAL
- MACRO-15 Macro Assembler
- DDT-15 Dynamic Debugging Technique
- Text Editor
- PIP-15 Peripheral Interchange Program
- Linking Loader
- Chain and Execute
- Patch

SGEN System Generator
Octal Dump (DUMP)
Library Update (UPDATE)
DECtape Copy (DTCOP)

1.1.3 PDP-15/30 Background/Foreground Monitor

Under control of the PDP-15/30 Background/Foreground Monitor, real-time tasks are executed in the computer foreground and have immediate call on the system's resources. Unused background time, available between service calls for the real-time tasks, is useful in program development, testing, or other lower-priority computation.

PDP-15/30 software encompasses all Advanced Monitor functions and capabilities (see list above). In addition, the PDP-15/30 Background/Foreground Monitor contains all the supervisory controls necessary for concurrent processing of background and foreground tasks.

PDP-15 System users can draw on the resources of the program library and the applications knowledge of DECUS, the Digital Equipment Computer Users Society, in addition to the Advanced Monitor programs and routines. DECUS members share in the exchange of programs and technical papers at regularly scheduled meetings throughout the year; the proceedings of all DECUS society meetings are published under DEC sponsorship.

PDP-15/30 Background/Foreground Monitor System

Background/Foreground Monitor controls the use of the PDP-15 by two co-resident programs.

System loader
Command decoder
IOPS data-handling, device-handling, and interrupt routines
Real-time clock handler
Error detector program
Device assignment tables

In addition to the above programs, the programs of the PDP-15/20 Advanced Monitor System are included in the 15/30 System.

1.1.4 PDP-15/40 Disk-Oriented Background/Foreground System

PDP-15/40 Disk-oriented Background/Foreground Systems are responsive to the high demands of industrial and engineering environments, where the need for a background/foreground mode of operation is compounded by the necessity of large random-access files. PDP-15/40 Systems with 24,576 words of core memory, high-speed paper tape facilities, and DECtape storage, also incorporate a DECdisk control and two random-access disk files. The disks, whose storage capacity is 524,288 18-bit words, can be expanded to 2,097,152 words, permit high-speed overlays, chaining, and system and user loading.

The disk-oriented background/foreground monitor system handles all the functions of the PDP-15/30 Background/Foreground Monitor in a high-speed disk environment.

PDP-15/40 Disk-Oriented Background/Foreground Monitor System

- Disk-oriented Background/Foreground Monitor
- Systems loader
- Command decoder
- IOPS data-handling, device-handling, and interrupt routines
- Real-time clock handler
- Error detector program
- Device assignment tables

The programs in the PDP-15/20 Advanced Monitor are included in this section.

1.1.5 PDP-15/10 COMPACT Software System

The PDP-15/10 COMPACT Software System is a concise programming system that includes a symbolic assembler, a text editor for creating programs on-line, debugging routines, utility routines, and mathematical routines. The COMPACT Software System is designed to operate in the 4K or 8K paper-tape input/output environment of the basic PDP-15/10. PDP-15/10 Systems with more than 8K of core are not supported by the COMPACT Software System. Installations with a minimum of 8K of core and a high-speed paper tape reader/punch can use the Basic I/O Monitor Software to extend system capabilities.

Utility routines in the COMPACT Software System include a Hardware Read-in Mode (HRM), punch routines, paper tape handling routines, Teletype I/O routines, an octal dump routine, and a memory scan routine used for scanning areas of memory for a particular bit configuration. For systems with DECtape, the FAST system can retrieve frequently used programs from DECtape.

COMPACT Assembler - The two-pass COMPACT Assembler has a useful set of selected pseudo-ops for functions such as table formations, symbol table and variable control, and text handling.

COMPACT Debugging Routines - Debugging routines are included in the COMPACT Software System. ODT (Octal Debugging Technique) is an aid to the user conducting interactive, on-line debugging sessions using octal numbers and Teletype commands.

COMPACT Editor - The COMPACT Editor takes advantage of the powerful character string, search, and modification commands developed for the larger systems. It provides for the creation and/or identification of source programs, other than ASCII text material, using keyboard commands. The Compact Editor also offers an efficient method for on-line processing of paper tapes.

1.1.6 PDP-15/10E Basic I/O Monitor

The Basic I/O Monitor, for 8K configurations, provides a link between the call for I/O, by either user or system programs, and the actual I/O execution. All I/O calls to system devices are serviced by DEC-supplied device handlers which reside in the Input/Output Programming System (IOPS). The device handlers actually move data between the program and the I/O devices. Device handlers initialize the devices and perform all other functions peculiar to a given I/O device, such as servicing interrupts in a real-time environment. User-supplied device handlers can be incorporated into the system to perform the functions described above for special I/O devices.

1.1.7 PDP-15/20 Advanced Monitor System

The PDP-15/20 Advanced Monitor combines the functions of the Basic Monitor with the executive control of bulk storage devices (to provide automatic operation), which includes batch processing, keyboard interaction, and real-time queuing. The Advanced Monitor has a large set of commands that direct the operation of the system. These commands perform three major functions:

a. Provide information about the system such as commands available and their functions; error diagnostics; the standard logical-physical I/O device associations; I/O level programs available (device handlers); special memory registers and their functions.

b. Permit the standard physical-logical device associations to be modified, thereby enabling the dynamic allocation of devices at load-time. This is a natural extension of device independence provided by the Basic I/O Monitor.

c. Supervise the loading and execution of all system and user programs, their associated I/O device handlers, and library subroutines, in addition to generating error messages and recovery procedures.

Coupled with keyboard control of system programs, the Advanced Monitor enables the user to deal with his entire problem (editing, assembling, compiling, loading, debugging and running) in a straightforward manner. The Advanced Monitor consists of command decoder, IOPS routines, real-time clock handler, error detector routine, and device assignment table (DAT).

The system loader always resides in upper memory and is responsible for loading the Monitor into lower memory. Return calls from system or user programs cause restoration of control to the Monitor.

The Monitor command decoder detects requests for system programs and loads the system loader, which brings in the requested program. In response to control cards or keyboard commands, it also manipulates the device assignment table to provide device independence. The Monitor Input/Output Programming System (IOPS) routines include data handling subroutines, device handlers, and interrupt service routines for the priority interrupt system, as well as the Teletype keyboard and printer. All other IOPS device handlers are stored on the system device until required by object programs.

The Monitor contains a device assignment for each table entry; because the contents of the table can be altered by commands to the Advanced Monitor, actual I/O devices can be changed without altering the program references to these devices. The following system software is supplied with all PDP-15/20 Advanced Monitor System.

FORTRAN IV - The PDP-15 FORTRAN IV compiler is a two-pass system which accepts statements written in the FORTRAN language and produces a relocatable object code capable of being loaded by the Linking Loader program. The PDP-15 FORTRAN IV compiler is compatible with USA FORTRAN IV, as defined in the USA Standard X3.9-1966, modified to allow the compiler to operate in 8,192 words of core storage. The FORTRAN IV compiler generates programs which operate with the program interrupt enabled and works with assembly language programs that recognize and service real-time devices. Subroutines written in either FORTRAN IV or the MACRO Assembler language can be loaded with and called by FORTRAN IV main programs. Source language diagnostics are produced during compilation, and a symbol table is generated for use in on-line debugging.

FOCAL - An on-line, interactive (conversational) algebraic language designed to aid scientists, engineers, and students in solving numerical problems. The language consists of short, easy-to-learn English imperative statements. Mathematical expressions are usually typed in standard notation. FOCAL puts the full calculating power and speed of the PDP-15 under easy conversational control. For example, FOCAL can be used to simulate mathematical models, to plot curves, to handle sets of simultaneous equations in n-dimensional arrays, and to solve many other kinds of problems. FOCAL runs in the Advanced Software Environment.

MACRO-15 Assembler - MACRO-15 Assembler enables the programmer to use mnemonic symbols to represent operation codes, locations, and numeric data. The programmer can direct the MACRO Assembler's processing through use of a full set of pseudo-operations. An output listing can be obtained to illustrate the programmer's source coding, as well as the binary object code produced by the MACRO Assembler. An optional third pass by the MACRO Assembler provides a cross reference listing. PDP-15 users can also make use of highly sophisticated macro generating and calling facilities within the context of a symbolic assembler. Some features of MACRO-15 are as follows:

a. The ability to define and call nested macros
b. Conditional assembly based on the computational results of symbols or expressions
c. Repeat functions
d. Boolean manipulation
e. Optional symbolic listing cross reference
f. Two forms of radix control (octal and decimal) and two text modes (7-bit ASCII and 6-bit trimmed ASCII)
g. Global symbols for easy linking of separately assembled programs
h. Choice of output format: relocatable, absolute binary (checksummed), or full binary (unchecksummed), capable of being loaded via the hardware READ-IN switch.
i. The ability to call I/O system macros that expand into IOPS calling sequences.

Dynamic Debugging Technique DDT-15 - A versatile tool for dynamic program checkout and modification. An operator can load a program and run all or selected portions of it in a real-time interrupt environment under interactive supervision of DDT-15. The Teletype keyboard controls DDT and program examination and modification. The operator can insert a breakpoint, specify the number of program iterations before interrupting the program, and start the program at any point using a simple set of commands. The operator can examine or alter any location symbolically and then rerun the program using other commands.

Text Editor - Using the PDP-15 Advanced Software System, an operator can create or edit symbolic text utilizing any input or output device. A "context" method is employed throughout to identify the block of data which the user wishes to modify; that is, the block is specified by its ASCII text rather than by a numbering scheme imposed externally upon the text. The Text Editor operates on these lines of ASCII text. Commands are available which facilitate insertion, deletion, and modification of data in the object file.

PIP-15 Peripheral Interchange Program - PIP-15 facilitates the manipulation and transfer of data files from any input device to any output device. It can be used to update file descriptions, verify, delete, segment, or combine files, perform code conversions, and copy tape.

Linking Loader - The Linking Loader loads any PDP-15 FORTRAN IV or MACRO-15 object program , in either relocatable or absolute format. Its tasks are loading and relocation of programs, loading of called subroutines, retrieval and loading and relocation of the necessary symbol tables.

CHAIN and EXECUTE - The programs CHAIN and EXECUTE facilitate a user-generated system of core overlays in the PDP-15 Advanced Monitor environment. This system of overlays consists of a resident main program, other indicated resident routines, a resident blank COMMON storage area, and a set of subroutines which overlay each other, as directed by the user. These subroutines are grouped into units called LINKS. Many, or all, LINKS can overlay each other, and several LINKS can overlay a larger LINK without overlaying each other. Cascading of suboverlays is not limited.

A LINK is loaded into core when a subroutine within the LINK is called and remains resident until overlayed. A LINK's core image is not recorded or "swapped out" when it is overlayed. The same image is brought into core each time a LINK is loaded.

Subroutines are called and return control to the calling routine in the normal fashion. There is no imposed order in which routines must be called, nor is there restriction of the routines callable by any routine.

The program CHAIN is used to build an XCT file, and the program EXECUTE supervises core residency during the execution of a CHAIN-built Overlay System.

PATCH - The user can conveniently examine and modify system program parameters and system programs stored on mass storage devices (DECtape or DECdisk) using the utility program PATCH.

UPDATE - The contents of binary library files on mass storage devices can be listed and updated, by insertion, deletion, or replacement operations using the library update utility program UPDATE. A binary library file is defined as any set of relocatable programs stored together as one unit in a single file. The PDP-15/20 Advanced Monitor System library file (.LIBR BIN) is a typical example.

DUMP - The user has the capability to output, on any listing device, specified core locations stored on the SAVE or QAREA of a mass storage device using the DUMP utility program. The listing output of any block of mass storage (DECtape or DECdisk) is obtained through the DUMP program.

SRCCOM - The source compare (SRCCOM) utility program compares any two symbolic programs and lists the differences between them. SRCCOM is useful in proofing an edited program and in keeping track of symbolic changes.

SGEN - The system generator (SGEN) utility program is used to build resident mass storage systems tailored to the customer's installation. Operating in conversational mode, SGEN uses the query/response technique to build the operating system to the customer's needs.

1.1.8 PDP-15/30 Background/Foreground Monitor System

The PDP-15/30 Background/Foreground Monitor system is an extension of the Advanced Monitor system which enables the concurrent, time-shared use of the PDP-15/30 through protected, foreground user programs with a background of batch processing, through program development, or through low-priority user programs. The system handles a variety of tasks, from high-speed data gathering applications such as those in physics to thousand-channel input/output applications such as warehouse inventory control. With the Background/Foreground Monitor the user can:

a. Effectively have two computers: one for on-line data acquisition and control, one for off-line program development, and data reduction at the price of one system;
b. Achieve 100% use of his system, independent of data rates.

The foreground programs are assumed to be checked out and to operate from requests to the program interrupt or priority interrupt facilities. At LOAD TIME, foreground programs have first priority over core memory and I/O devices, and at EXECUTION TIME they have priority (according to their assigned priority levels) over processing time and shared I/O devices.

The background program (or sequential series of programs) is essentially the same as the single-user program under the Advanced Monitor system; that is, it can be an assembly, a compilation, a debugging run, a production run, an editing task, or batch processing. The background program can use whatever facilities (core, I/O, processing time, etc.) are available and not required by the foreground programs.

The Background/Foreground Monitor can be used to direct the time-shared use of the PDP-15/30 by the two coresidential programs and to perform the following functions:

a. Schedules processing time
b. Protects the foreground job's core

c. Protects the foreground job's I/O devices

d. Allows the sharing of multi-unit device handlers, such as DECtape, by both foreground and background jobs

e. Directs the shared use of the system real-time clock to time specified intervals

f. Directs communication between background and foreground jobs via core-to-core transfers.

1.1.9 PDP-15/35 Real-Time System Executive

The PDP-15/35 Real-Time System Executive (RSX) is a disk-based system designed for multitask, multiprogramming environments, where real-time interrupts, time interval task activation, and a priority job queue must all be coordinated under a priority structure.

1.1.10 PDP-15/40 Disk-Oriented Background/Foreground Monitor System

The PDP-15/40 system uses a disk-oriented version of the Background/Foreground Monitor; it contains all of the features described above in the PDP-15/30 Background/Foreground Monitor section. The disk system enables high-speed overlays, chaining, and system and user program loading to occur. The number of records that can be opened on the disk is limited only by available word space. The PDP-15/40 system contains 524,288 words of disk storage, expandable to 2,097,152 words.

1.1.11 Additional Systems Software

8TRAN - The 8TRAN translator is used to translate programs written for PDP-8 in PAL III, PAL-D, or MACRO-8 assembly language to MACRO-15 assembly language. The purpose of the translator is not to produce a program which runs on the PDP-15 by simulating the PDP-8, but rather to do the straight-forward portion of the translation and clearly indicate to the programmer those parts of the code which require review in the light of the PDP-15's greater word length and more powerful instruction set.

STATPAC - STATPAC is a comprehensive and open-ended package of modular statistical programs designed to operate under the PDP-15 Advanced Monitor. The user with limited computer knowledge can use STATPAC to obtain statistically meaningful results from data. STATPAC includes modules for CONTROL, INPUT, DESCRIPTIVE STATISTICS, STEPWISE LINEAR REGRESSION, and MULTIPLE LINEAR REGRESSION functions.

1.2 PDP-15 SYSTEM CONFIGURATIONS

PDP-15 Systems offer comprehensive solutions to real-time data problems by combining new design concepts with a wide variety of traditional DEC features. Through DEC's experience in the medium-scale scientific computer field, the PDP-15 System simplifies the user's tasks in a demanding real-time environment.

Because certain data-handling tasks require specific hardware and software configurations, DEC has developed four standard PDP-15 Systems, ranging in power from the modestly priced basic PDP-15/10 to the PDP-15/40 Background/Foreground Disk Monitor System. At every level, the capabilities of the hardware are under the control of a monitor designed specifically for them.

The software systems are designed around the hardware with the user environment in mind. The principal design objectives are to provide (a) a system that is convenient for the user to implement and that affords the user access to the full power of the hardware, (b) a system that allows the user to easily integrate his applications program and special peripheral device handlers, and (c) a system that can expand naturally. PDP-15 Systems software enables the user to move from a very basic machine to a sophisticated system without the cost and complication of reprogramming at each upward step.

The hardware systems were designed with complete autonomy between central processor, input/output processor, and memory, so that processing and I/O operations can occur concurrently in overlapping cycles; TTL integrated-circuit construction for high reliability; fast internal speeds, including an 800-ns memory cycle time, to meet the demands of real-time data processing; core memory expansion to 131,072 words for future growth; and a sophisticated memory protect system for multi-user integrity. Peripheral device handling and interfacing to other instruments are easily accomplished, and system growth potential is virtually unlimited with the modular structure of PDP-15 Systems.

1.2.1 PDP-15/10 Basic System

The PDP-15/10 is the first level PDP-15 System. The system's design provides limited budget users access to the power, speed, and 18-bit word length of PDP-15 hardware, in the expectation that the system can later be expanded to take full advantage of the advanced software capabilities inherent in the system's design.

Hardware includes 4,096 18-bit words of core memory and a Model 33 ASR Teletype console teleprinter. The system has the rapid PDP-15 800-ns memory cycle time which provides 1.6-μs add capability. Facilities for later expansion are prewired into the system; additional memory and peripherals can be plugged in as required.

Software is governed by the COMPACT Programming System, a complete package including Assembler, Editor, Octal Debugging Technique, and mathematical and utility routines. All are designed to function in a 4096 word system. The software offers complete upward compatibility at the source level and field-proven reliability. Programs written for execution under COMPACT can also run, with little modification, within all PDP-15 System levels up through PDP-15/30 and PDP-15/40 Background/ Foreground Systems.

1.2.2 PDP-15/20 Advanced Monitor System

PDP-15/20 is an 8,192-word mass storage-oriented system designed for research and engineering environments where real-time data acquisition and control tasks are combined with program development and testing.

Program development, debugging, and modification are all handled under monitor control, virtually ending intermediate operations. Unique real-time input/output routines can also be integrated into the system monitor to accelerate set-up and recovery.

Users are spared the task of writing system software to handle input/outputs to all standard system peripherals, since appropriate routines are supplied with the monitor. The net result is that even inexperienced computer users can get their applications programs "on the air" in a minimum amount of time.

PDP-15/20 hardware facilities include not only, 8,192-words of core memory and high-speed paper-tape facilities but also, a DECtape control unit and two tape transports for convenient mass-memory storage. The extra-heavy duty 35 KSR Teletype unit is included in the PDP-15/20 configuration to guarantee a high degree of reliability under the strain of continued heavy use. Also included is the extended arithmetic element described in Chapter 6. This unit facilitates high-speed multiplication, division, shifting, normalization, and register manipulation.

The 15/20 Advanced Monitor System permits two types of user interaction. These are (1) batch processing for routine production jobs, and (2) keyboard interaction which enables the user to operate the system with simple commands typed at the keyboard.

Other PDP-15/20 Advanced Monitor features that make use of processor options are: a real-time clock control and a priority interrupt control.

1.2.3 PDP-15/30 Background/Foreground System

The PDP-15/30 System is designed to meet the demands of research, engineering, and industrial environments, where one or more real-time tasks typically require continuous responsiveness from the computer, but do not use 100% of its capacity.

The PDP-15/30 Background/Foreground System requires a minimum of 16,384 words of core memory and all the devices standard for the PDP-15/20. In addition, PDP-15/30 Systems are equipped with a memory protect system, a real-time clock, automatic priority interrupt, two DECtape transports, and a second on-line Teletype for background use.

1.2.4 PDP-15/35 Real-Time System Executive Disk-Oriented System

The PDP-15/35 System contains 16,384 words of core memory, high-speed paper-tape facilities, two DECtape transports, the automatic priority interrupt option, real-time clock, and a DECdisk file with 262,144 words of storage.

1.2.5 PDP-15/40 Disk-Oriented Background/Foreground System

PDP-15/40 Disk-Oriented Background/Foreground System fulfills the demands of industrial and engineering environments where the need for a Background/Foreground mode of operation is compounded by the necessity for large random-access files.

The PDP-15/40 System with 24, 576 words of core memory, high-speed paper-tape facilities, and DECtape storage, also incorporates a DECdisk control and two random-access DECdisk files. The two disks, whose storage capacity of 524,288 18-bit words can be expanded to 2,097,152 words, permit high-speed overlays chaining and system and user loading.

Other hardware features of the PDP-15/40 include a memory protect system, background Teletype, and a real-time clock.

The PDP-15/40 Disk-Oriented Background/Foreground Monitor System handles all the functions of the 15/30 Background/Foreground Monitor in a high-speed disk environment.

1.3 SYSTEM ORGANIZATION

The basic PDP-15 hardware is shown in Figure 1-1. Three autonomous subsystems, central processor, memory, and I/O processor, operating together under console control define the PDP-15 System.

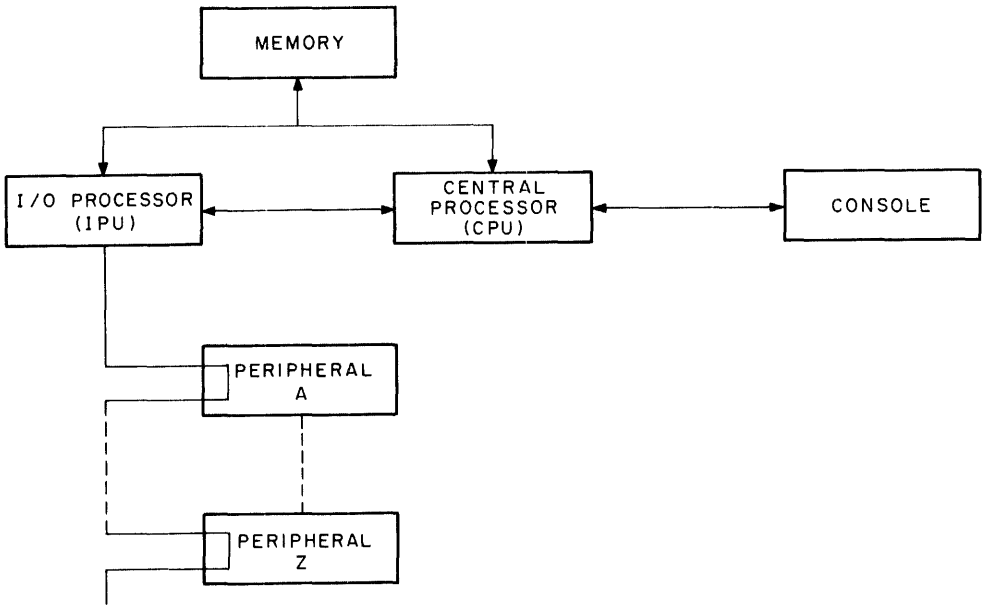


Figure 1-1 PDP-15 System Organization

An extensive line of peripherals including mass storage displays, data communication, and data acquisition equipment is coupled to the PDP-15 I/O processor and serviced under the supervision of the monitor systems.

1.3.1 Central Processor (CPU)

The central processor functions as the main component of the computer by carrying on bidirectional communication with both the memory and the I/O processor. Provided with the capability to perform all required arithmetic and logical operations, the central processor controls and executes stored programs. It accomplishes this with an extensive complement of registers, control lines and logic gates.

1.3.2 Memory

The memory, second of three autonomous subsystems, is the primary storage area for computer instructions and system data. The memory is organized into pages which are paired into memory banks. Each page has 4096, 18-bit binary words of high-speed, random-access magnetic core storage. Each bank is an asynchronous unit of 8192 words. The central processor has provisions to address up to 131,072 words of core memory. Any word in memory can be addressed by either the central processor or the I/O processor.

1.3.3 I/O Processor (IPU)

The third autonomous subsystem handles peripheral data transfer. A diverse line of system peripherals available to the PDP-15 require this processor to interface three modes of input/output:

a. Single-cycle block data transfer; blocks of data transfer at rates up to one million words per second.

b. Multicycle block data transfer; blocks of data transfer at rates up to 250,000 words per second for input and 188,000 words per second for output.

c. Program-controlled data transfers; single-word transfers to/from the accumulator in the central processor.

The I/O processor provides timing, control, and data lines for information transfers between memory or the central processor and the peripheral devices; it also includes provision for such options as the automatic priority interrupt system and the real-time clock.

1.3.4 Console

The PDP-15 control console provides facilities for operator initiation of programs, monitoring of important CPU and IPU registers during program execution, and manual examination and modification of memory contents.

1.3.5 System Peripherals

The PDP-15 System peripherals range from simple input/output Teletypes to sophisticated interactive display processors. These peripherals communicate with the PDP-15 I/O processor via one 72-wire bidirectional cable called the common I/O bus.

Figure 1-2 depicts a large system showing the CPU and IPU options and some of the PDP-15 Systems.

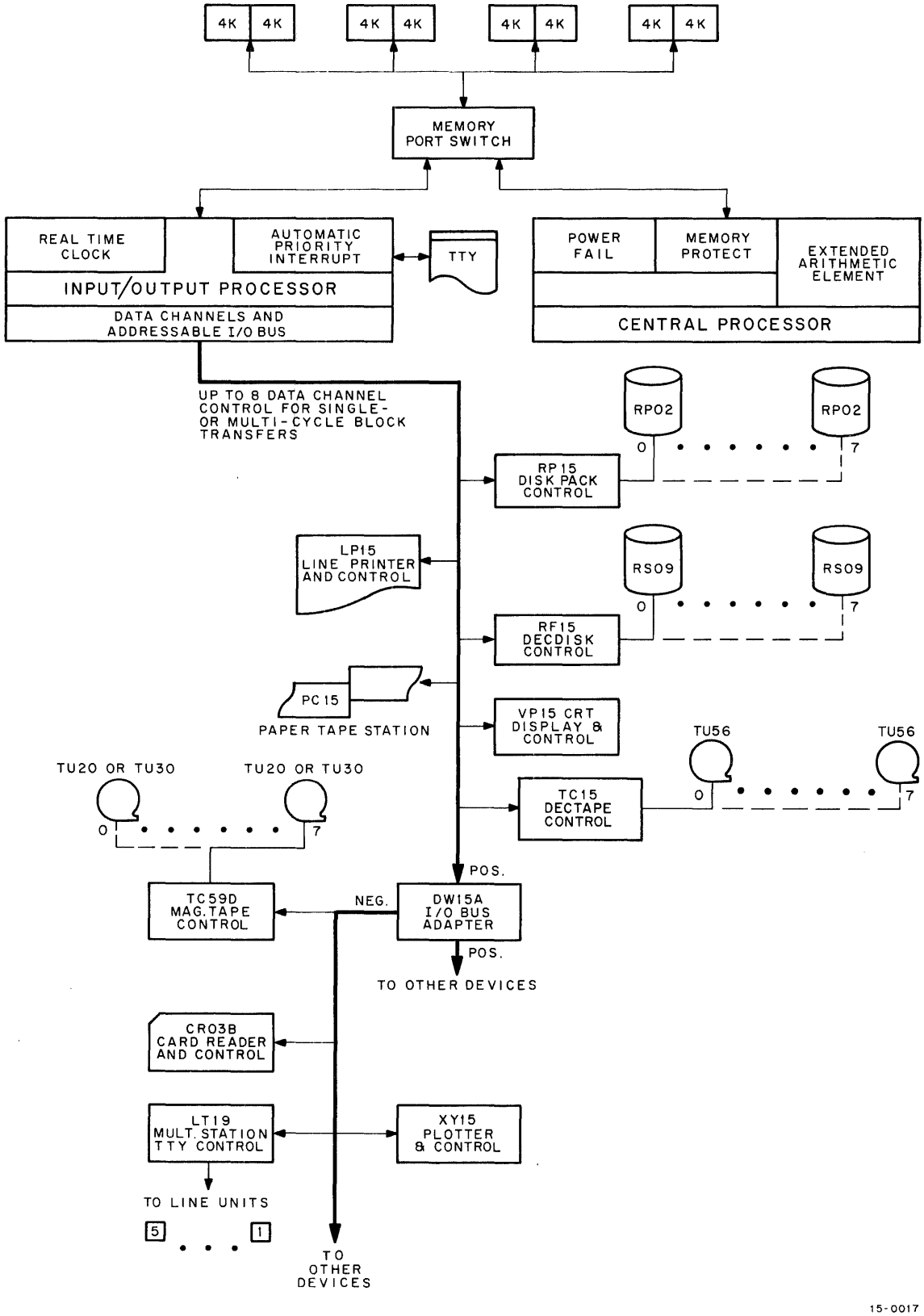


Figure 1-2 System Organization

Chapter 2
Processor Organization

2.1 CENTRAL PROCESSOR DESCRIPTION

The central processor (CPU) is the main component for control and execution of stored programs. By coordinating its operation with other subsystems, it provides supervisory control over the entire PDP-15 System.

The central processor contains arithmetic and control logic hardware for a wide range of operations. These include: high-speed, fixed-point arithmetic with a hardware multiply and divide option; extensive test and branch operations implemented with special hardware registers; high-speed input/output instructions; and other arithmetic and control operations.

The PDP-15 central processor contains several major registers for processor-memory communications, a program counter, an instruction register, an accumulator, an index register, and a limit register.

The CPU performs calculations and data processing in a parallel binary mode through step-by-step execution of individual instructions. Both the instructions and the data on which the instructions operate are stored in the core memory of the PDP-15. The arithmetic and logical operations necessary for the execution of all instructions are performed by the arithmetic unit operating in conjunction with central processor registers. Figure 2-1 shows a simplified block diagram of the central processor.

2.1.1 Internal Registers

Arithmetic Unit

The PDP-15 arithmetic unit handles all Boolean functions and contains an 18-bit, 85-ns adder. The arithmetic unit acts as the transfer path for inter-register transfers and shift operations.

Instruction Register (IR)

The instruction register accepts the six most-significant bits of each instruction word fetched from memory. Of these bits, the four most-significant constitute the operation code, the fifth signals when the instruction indicates indirect addressing, and the sixth indicates indexing.

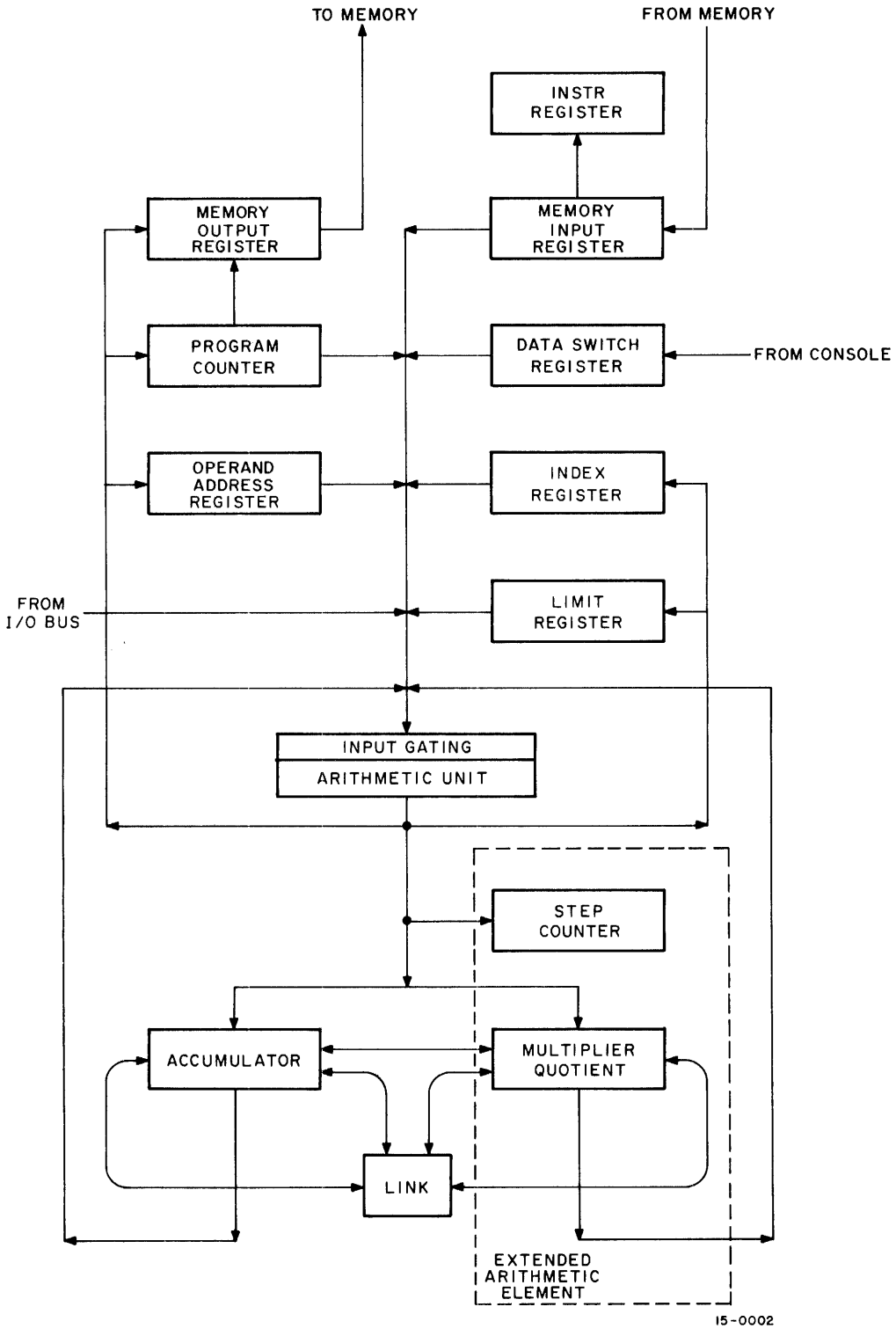


Figure 2-1 Central Processor, Simplified Block Diagram

Accumulator (AC)

This 18-bit register retains (accumulates) the result of arithmetic or logical operations for storage between instructions.

For all program-controlled input-output transfers, information is transferred between core memory and an external device through the AC. The AC can be cleared and complemented. Its contents can be rotated right or left with the Link (see below). The contents of the memory, buffered through the memory input register, can be added to the contents of the AC with the result left in the AC. The contents of both registers can be combined by the logical operations AND and exclusive OR, the result remaining in the AC. The inclusive OR can be performed between the AC and the DATA switches on the operator console (through the data switch register) and the result left in the AC.

Data Switch Register (DSW)

The data switch register receives and buffers an 18-bit word through the console.

Link (L)

This 1-bit register is used to extend the arithmetic capability of the accumulator. In 1's complement arithmetic, the Link is an overflow indicator; in 2's complement arithmetic, it logically extends the accumulator to 19 bits and functions as a carry register. The program can check carry into the Link to simplify and speed up single- and multi-precision arithmetic routines. The Link can be cleared and complemented and its state sensed independent of the accumulator. It is included with the accumulator in rotate operations and in logical shifts.

Program Counter (PC)

The program counter determines the program sequence (the order in which instructions are performed). This 18-bit register contains the address of the memory location from which the next instruction is to be taken. The least-significant 15 bits are used for addressing 32,768 words of core memory. Two remaining bits provide the capability to address memory systems greater than 32,768 words.

Operand Address Register (OA)

The operand address register is a temporary holding register (not available to the programmer) which contains the effective address of the last (or current) memory reference operand.

Memory Input and Output Buffer Registers (MI and MO)

Information is read from a memory location into the memory input register and is interpreted as either an instruction, address, or a data word. Information is read from the central processor into memory through the memory output register and is interpreted as either an address or a data word. The use

of two 18-bit registers for memory buffer functions allows the processor to overlap with memory cycle time to decrease execution time and to allow autonomous operation of the CPU and memory.

Index Register (XR)

This 18-bit register is used to perform indexing operations with no increase in instruction execution time. An indexed operation adds the contents of the index register to the address field of the instruction operand producing an effective address for the data fetch cycle. The index value is a signed 17-bit integer 131,072).

Limit Register (LR)

The limit register enables a program to detect loop completion. The base address of a data array is loaded into the index register and the ending address is loaded into the limit register. Within an indexing loop, add to index and skip (AXS) instruction, adds a signed value ($\pm$256) to the index register and compares the sum in the index to the contents of the limit register. If the contents of the index register are equal to or greater than those of the limit register, the next instruction is skipped.

2.1.2 Control Console

The PDP-15 control console contains the keys, switches, and indicators required for operator initiation, control, and monitoring of the system. Up to twenty-four 18-bit registers can be displayed to provide the user with visual indication of most registers and buses.

Some of the features of the console are:

a. A READ-IN switch to initiate the reading of binary paper tapes.

b. REGISTER indicators and REGISTER DISPLAY switches for continuous monitoring of key points in the system such as the accumulator, index register, limit register, multiplier-quotient register, program counter, memory address, interrupt status, input/output bus, input output address, and I/O status.

c. DATA switches to establish an 18-bit data or instruction word to be read into memory by the DEPOSIT switch, to be entered into the accumulator by a program instruction, or to be executed as an instruction by pressing the EXECUTE key.

d. EXAMINE switch initiates the manual examination of the contents of any memory location specified by the ADDRESS switches.

2.2 CENTRAL PROCESSOR EXPANSION OPTIONS

The following additional expansions extend the processing capabilities of PDP-15 Systems.

Extended Arithmetic Element (EAE)

The extended arithmetic element (standard on PDP-15/20/30/40 Systems) facilitates high-speed arithmetic operations and register manipulations. Installation of the EAE adds an 18-bit multiplier-quotient register (MQ) to the system as well as a 6-bit step counter register (SC). EAE instructions can be microcoded so that several operations are performed by one instruction to simplify arithmetic programming and reduce execution time. Worst case multiplication time is 7.42 μs; division time is 7.68 μs. The EAE is optionally available for the PDP-15/10.

Multiplier-Quotient Register (MQ)

The multiplier-quotient register and accumulator perform as a 36-bit register during shifting, normalizing, multiplication, and division operations. The contents of the multiplier-quotient register are displayed by the REGISTER indicators on the operator's console when the REGISTER DISPLAY control is in the MQ position.

During the multiply instruction, the MQ receives the 18 least-significant bits of the double word product formed in the AC and MQ. During the divide instruction, the MQ is the least-significant 18 bits of the double word DIVIDEND formed by the AC and MQ.

Step Counter (SC)

The step counter is used to count the number of steps in an EAE instruction. The step counter is preloaded, except during normalize operations, with the numbers of steps specified by an instruction and is counted down as the instruction is executed. When the SC reaches zero, the EAE operation is terminated.

Memory Protection

The memory protection feature, standard on PDP-15/30 and 15/40 Systems, establishes a background/foreground environment for PDP-15 processing activity by specifying the boundary between protected (lower) and unprotected (upper) regions of system core memory. Allocation of memory locations (in increments of 256 words) to the protected region is dynamic and program-controlled under the Background/Foreground Monitor. Figure 2-2 shows a PDP-15 System with the memory protect option. The protect feature increases all memory cycle times by 30 ns and write cycles in user mode by an additional 175 ns. Memory cycle times are specified in Table 3-1.

The protection option also provides a user/monitor mode of operation. When in user mode, attempted execution of any privileged instructions results in a trap to the monitor and a corresponding error message. These illegal instructions include IOT instructions, halts, chained executes, any references to the memory protect option itself, or protected memory. In monitor mode, all instructions are executable.

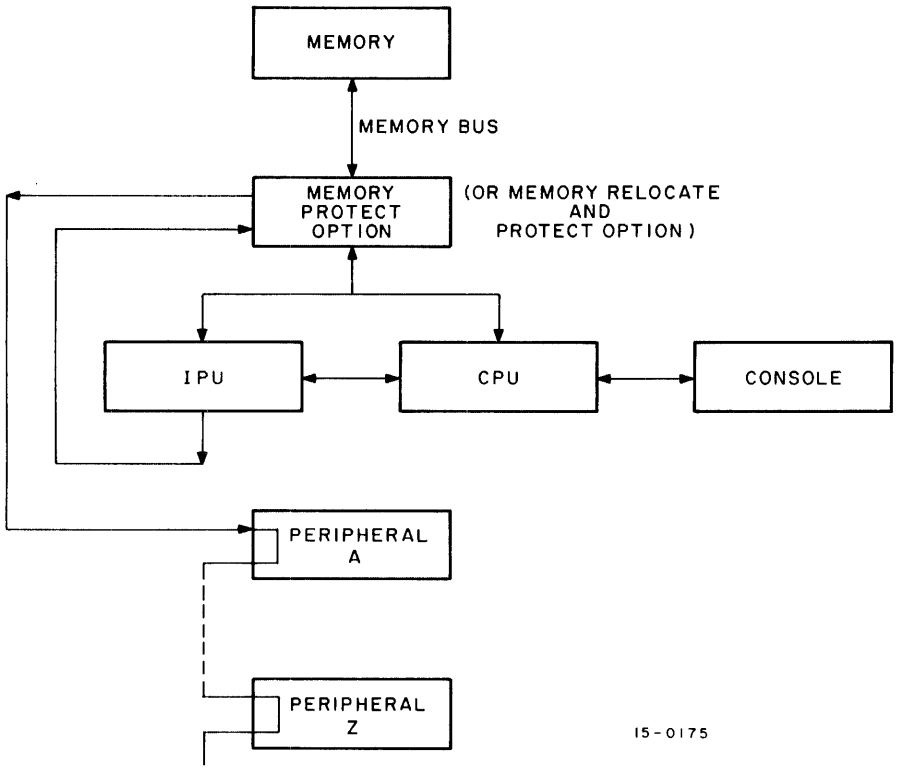


Figure 2-2 PDP-15 System With Memory Protect Option

The option is activated (set to user mode) with an I/O instruction, and when active, it monitors all CPU/memory instructions and addresses for illegal conditions and provides an interrupt if such conditions occur. Figure 2-3 gives more detail on the contents of the memory protect option.

Memory Relocate and Protect

Memory relocation is optional on all PDP-15 Systems. This feature is installed with the memory protect option on the memory bus (see Figure 2-2) and provides a relocation register and an upper boundary register to permit hardware relocation of user programs. It allows the relocated program to execute only within its specified boundaries, thereby providing protection for other programs resident in memory. Figure 2-4 shows a block diagram of the memory relocate and protect option. Note that it functions essentially the same as the basic protect hardware and gives the added capability to relocate programs in increments of 256 locations.

2.3 I/O PROCESSOR ORGANIZATION

The I/O processor is an autonomous subsystem of the PDP-15 which supervises and synchronizes all data and control transfers between the devices and the PDP-15 central processor and memory.

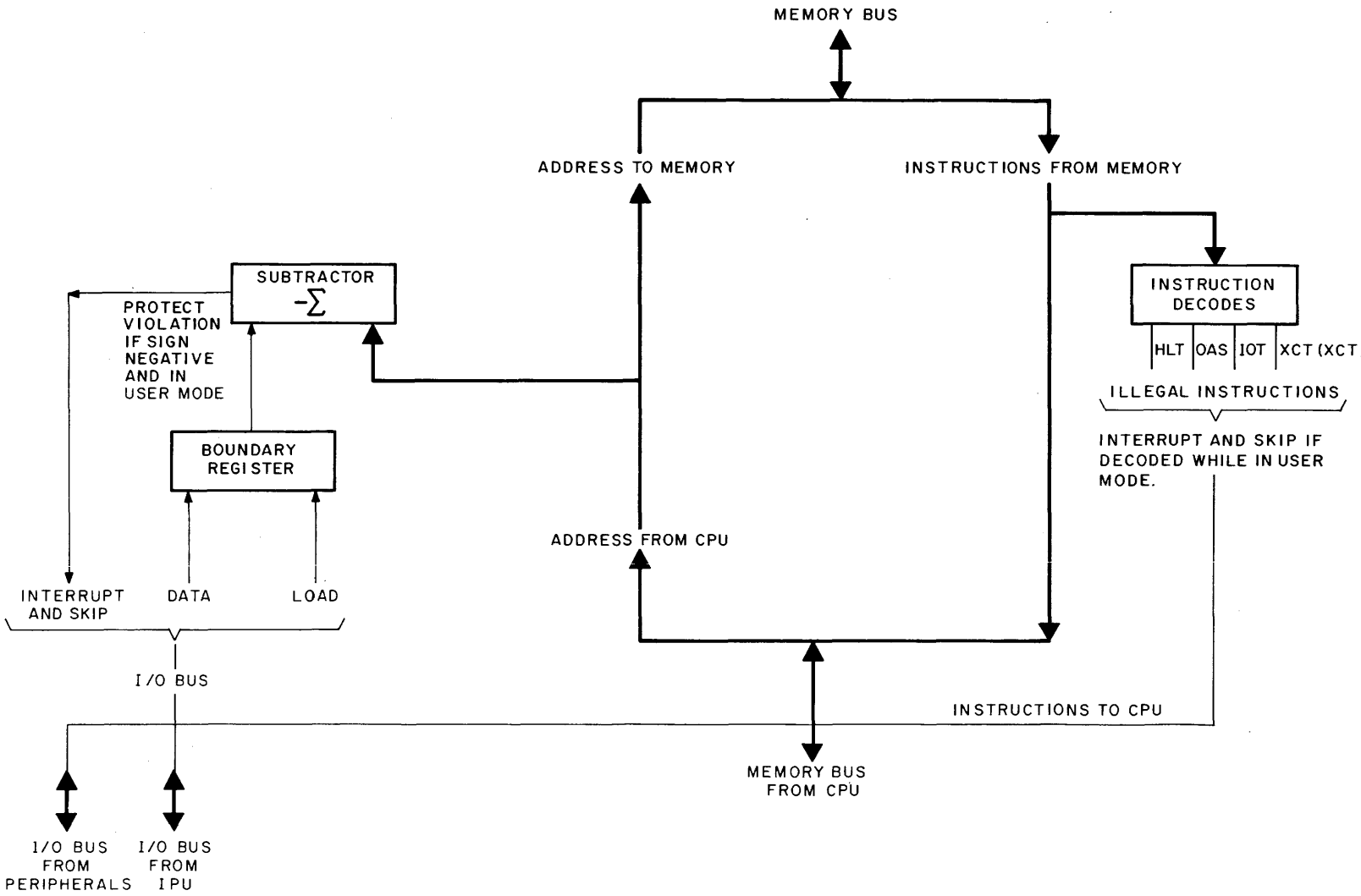


Figure 2-3 Memory Protect Block Diagram

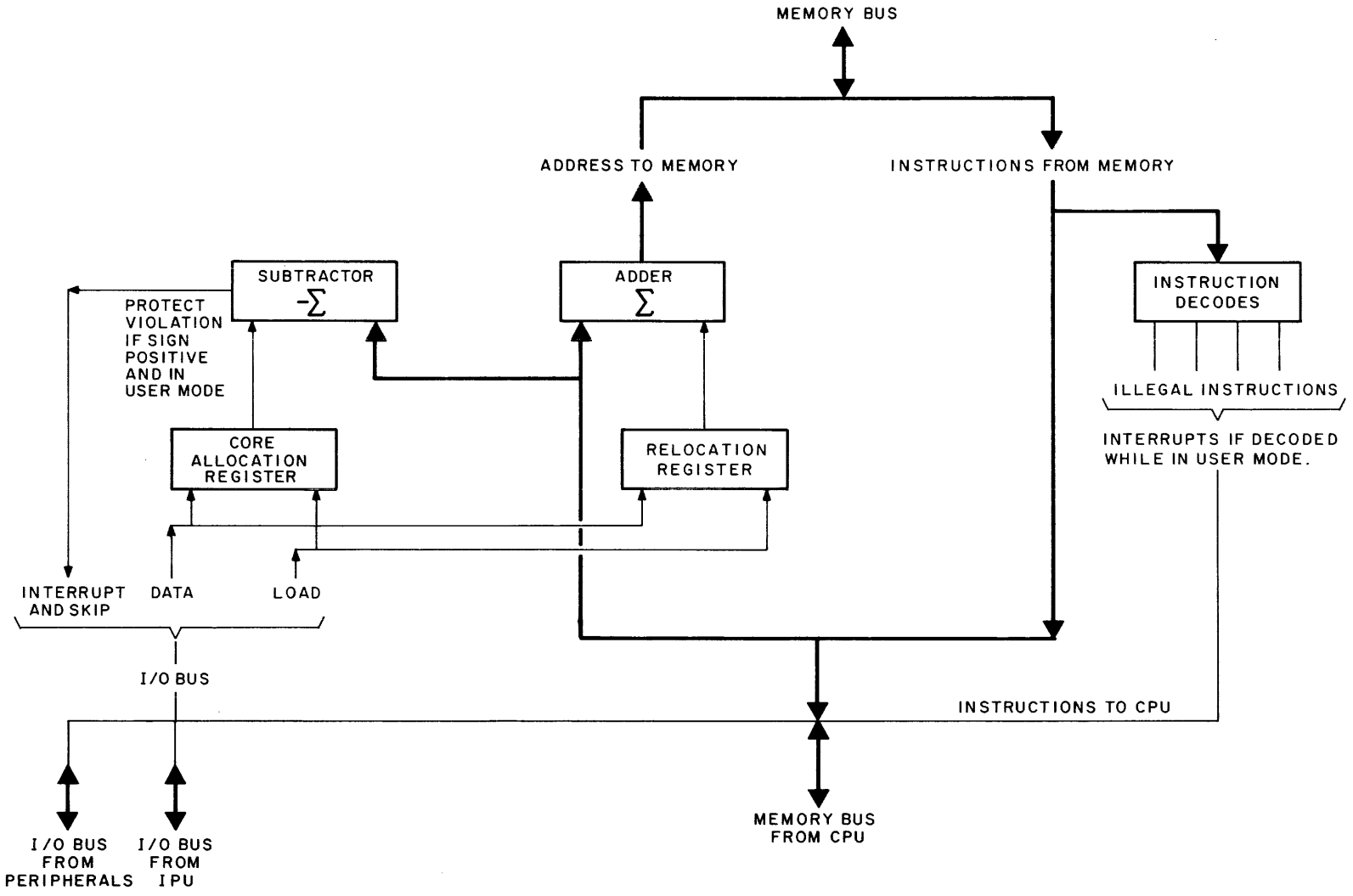


Figure 2-4 Memory Protect and Relocate Block Diagram

The I/O processor contains sufficient arithmetic and control logic hardware to supervise all I/O device activity. The IPU is, however, a passive subsystem: it responds to requests for activity from the devices or the CPU rather than initiating activity.

2.3.1 Data Transfer Facilities

The PDP-15 I/O processor contains a number of different facilities for handling I/O activity. Each facility has been designed to serve a basic requirement of the I/O devices. All I/O device transfers can be placed into one of the following catagories. (See Figure 2-5.)

Command Transfers - Command transfers from the CPU to a device initiate or stop all device activity, and establish device operating modes, transfer directions, and other control parameters.

Status Transfers - Status transfers from a device to the CPU are usually initiated by the CPU for the purpose of monitoring the progress (or status) of a previously initiated activity.

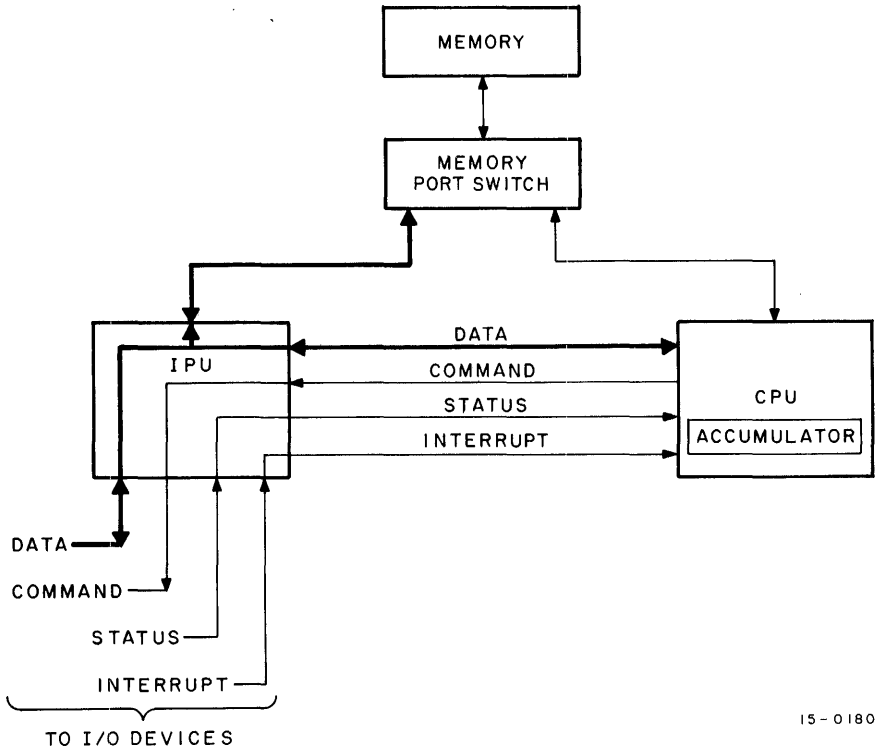


Figure 2-5 Data Transfer Facilities

Data Transfers - Data transfers take place between a device and memory or a device and the CPU under program control, and information may be transferred in either direction. Transfers of data from a device to the CPU are initiated by the CPU. Transfers of blocks of information from a device to memory or from memory to a device are initiated by the CPU. However, the transfer of individual words in a block is usually signaled by the I/O devices.

Interrupt Requests - Interrupt requests, from the I/O device to the IPU, signal the IPU that the device needs service. The interrupt system relieves the processor of the task of continuously polling each device's status to determine its need for service.

Several capabilities in each transfer category are available from the PDP-15 I/O processor: 1) maximum flexibility is afforded the user who wishes to interface special equipment to the PDP-15 and to the programmer who writes the device handler; 2) simple, inexpensive devices such as the Teletype can be easily interfaced to the PDP-15, and require total CPU supervision; 3) complex devices (such as the LP15 line printer) that need only one instruction to initiate a complete block transfer are built to minimize the amount of CPU supervision required. The trade offs between these extremes are device cost, transfer rates, and percentage of CPU time.

Table 2-1 shows the I/O capabilities of the PDP-15 under each transfer category.

Table 2-1
PDP-15 I/O Capabilities

Category	Capability
Command	IOT command instructions
	IOT AC transfer instructions
Status	IORS system read status instruction
	IOT skip instructions
	IOT AC transfer instructions
Data Transfers	IOT AC transfer instructions
	Multicycle data channel transfers
	Single-cycle data channel transfers
Special Transfers	Add to memory and increment memory

2.3.2 I/O Processor Activities

The following paragraphs describe the uses of each of the I/O processor activities. Note that some facilities have multiple uses.

IOT Commands - IOT command instructions from the CPU initiate, stop, or set the mode of the I/O device.

IOT AC Transfers - IOT AC transfer instructions from the CPU transfer up to 18 bits of data or command information from the CPU accumulator to the device's data or command registers, or command up to 18 bits of data or status information from the device's data or status registers to the CPU accumulator.

IORS Instruction - The IORS (input/output read status) instruction transfers up to 18 bits of status information (typically one bit from each device) to the CPU accumulator.

IOT Skip Instructions - IOT skip-instructions initiated by the CPU interrogate a specific flag or status bit in one of the 256 allowable devices and increments the CPU's PC (skips the next instruction) if the bit interrogated is asserted.

Multicycle Data Channel Transfer - Multicycle data channel transfers are IPU supervised transfers of data between the I/O device and sequential memory locations (in either direction). The word count and current address are kept in a pair of preassigned memory locations, and the counting and overflow detection is accomplished by the I/O processor.

Single-cycle Data Channel Transfers - Single-cycle (Direct Memory Access) transfers are device supervised transfers of information (up to 18 bits/word) between the I/O device and memory. The I/O device must contain word count and current address registers and provide overflow (job done) detection.

Program Interrupt - Program interrupt (PI) requests from the I/O devices cause the running program (at the completion of the current instruction) to transfer to a common subroutine that polls the devices to determine which device needs service. The program then transfers to the device service subroutine, and when finished handling the device, returns to the program which was interrupted by the request.

Automatic Priority Interrupt - Automatic priority interrupt (API) provide the same facility as the program interrupt except eight levels of priority are provided (4 software levels and 4 hardware levels). Instead of interrupting to a common device polling subroutine, the interrupting device provides a unique address of the subroutine call to its device handler. This eliminates the need for a device polling sequence and improves the interrupt response latency. Interrupts from different priority levels are fully nested and a debreak and restore instruction provides for orderly priority level dismissal.

Add-To-Memory - Multicycle data channel, add-to-memory facilities function in the same manner as other multicycle data channel transfers except a data word provided by the device is added to memory and the results are left in memory and transferred back to the device. Typical uses for this facility are high-speed averaging and in-core up-down counting.

Increment Memory - Data channel increment memory transfers cause the contents of a device-specified memory location to be incremented by one. A typical use for this facility is an in-core histogram updated by nuclear pulse height analyzer information.

2.3.3 I/O Processor Organization

The I/O processor has fully parallel arithmetic capabilities which provide autonomous I/O device supervision without interruption of central processor activities. In this manner, the I/O processor can perform an add-to-memory calculation initiated by an I/O device at the same time the CPU performs multiply or index instructions. To implement this capability, the I/O processor contains independent registers, adder, and control circuitry. Figure 2-6 is a block diagram of the I/O processor.

I/O Buffer

The I/O buffer is an 18-bit register which buffers input data from the I/O device.

I/O Adder

The I/O adder is an 18-bit adder which contain the basic arithmetic capabilities of the IPU.

DSR

The data storage register receives all output calculations from the I/O adder. It holds addresses or data destined for use by the memory, and it also holds data for presentation to the I/O bus lines.

The mixer logic, at the input to the I/O adder switches, appropriates data to the inputs of the adder, in order to perform the proper arithmetic operation. An example of the operation is as follows: during the data cycle of the add-to-memory data channel transfer, the contents of the memory location are

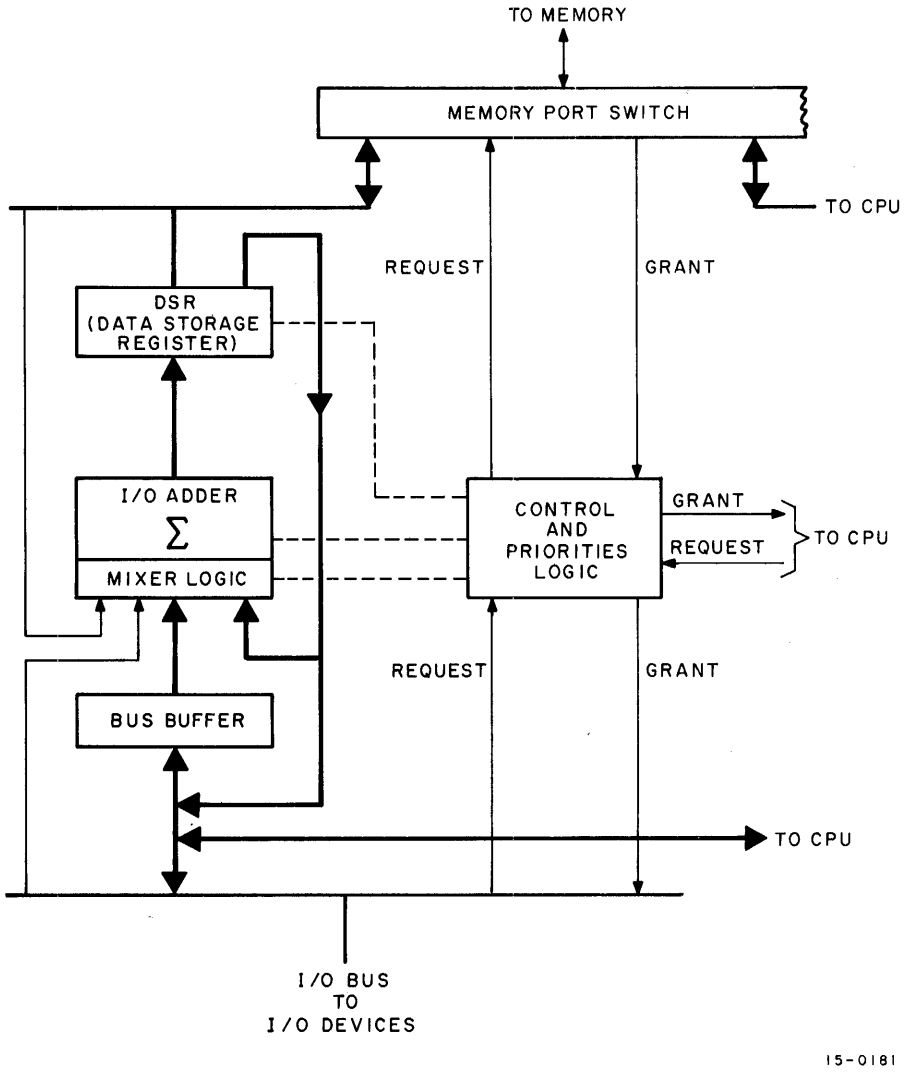


Figure 2-6 I/O Processor Block Diagram

presented to one input of the adder and the contents of the I/O buffer (which contains the device-specified word) are placed on the other input to the adder. The 18-bit sum is strobed into the DSR which presents the data to the memory and to the I/O bus.

Control and priority logic in the IPU synchronizes the requests from CPU or devices for IPU activity, grants action to the activities in appropriate order of priority, and controls the process of the transfer.

Chapter 6 contains a more detailed description of the I/O processor and its facilities.

2.4 CORE MEMORY

The magnetic core memory is the primary storage facility of the PDP-15. It provides random-access data and instruction storage for both the central processor and the I/O processor. The basic PDP-15/10 memory contains 4096 18-bit word locations. The contents of each location are available for processing in 400 ns. A parity bit can be added as an option to each word for parity checking during transfer of information into or out of core memory. If the parity option is incorporated into a PDP-15 System, all memory banks must contain that option and memory cycle time becomes 1.1 μs. The basic subsystem of memory is the memory bank; it is organized into pages, and each bank has two pages of 4096 words each for a total of 8192 words of 3D 3-wire cores. Further, every bank contains a data buffer, an address buffer, and all the necessary read/write and control circuitry to make it an autonomous unit operating on a request/grant basis with either the central processor or I/O processor. Figure 2-7 illustrates the organization of a memory bank.

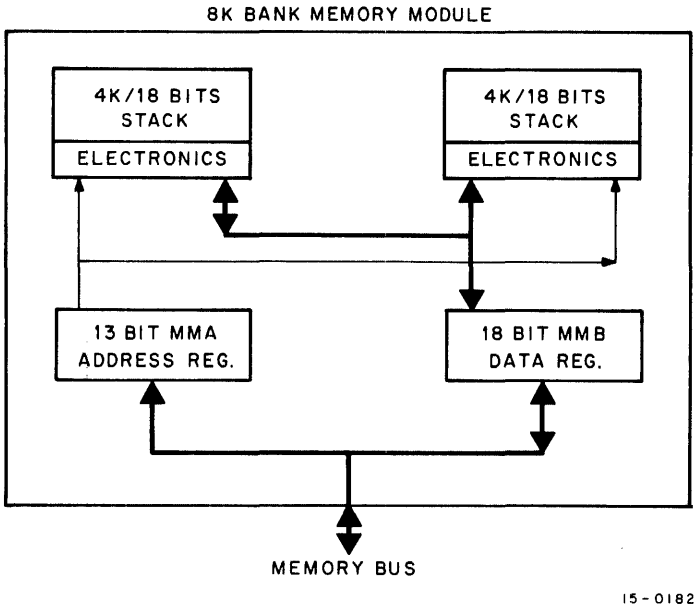


Figure 2-7 Memory Organization

2.4.1 Memory Data Transfer

The PDP-15 memory interacts directly with the central processor and the I/O processor through the memory bus. Data and instruction words of each bank are read from and written into individual memory locations through a buffered register, referred to as the memory buffer.

Words in a memory bank are selected according to the address in the memory address buffer. The capacity of the memory address buffer enables 8192 words to be referenced in each bank.

The memory address buffer receives the memory address from the central processor or I/O processor. The address provides the coordinates for locating a word in a memory bank.

Decoding of the memory address to select a particular word location containing 18 bits is performed by the memory selection logic. Bit 5 of the memory cell address selects the page of the location, and the remaining bits select the X and Y coordinates of the location.

Bits 1 to 4 of the memory bus select lines are used to select which bank of memory the word is in. Up to four banks can normally be added to the PDP-15, but a special provision to expand memory up to 16 banks can be accommodated by the 18-bit address register in the CPU.

2.4.2 Parity

The memory parity option provides core planes that have 19 bits for each word and parity checking/generating control logic. When the parity option is present, the accuracy of transfers to and from memory is verified through parity checking. A parity bit is added to each word stored in memory, so that the total number of 1 bits in the word, including the parity bit, is odd. For example, if the 18-bit word to be stored in memory contains an even number of 1s, the parity bit is automatically made a 1, and is stored with the word. When the word is later read from memory, the computed parity bit is calculated on the basis of the content of the 18-bit word. The calculated and actual parity bits are then compared, if they do not agree, the memory parity error alarm is initiated, causing a program interrupt or automatic priority interrupt request, or a halt.

All 18 bits and the accompanying parity-check bit (when present) are transferred in parallel (simultaneously) between the core array and the memory buffer. The memory buffer is connected to the memory bus, and therefore, to the rest of the PDP-15 System. This is also an 18-bit parallel transfer.

2.4.3 Memory Modularity

The PDP-15/10 System contains one page of 4096 memory words; however, additional modules (pages) can be added to the system. The basic system can accommodate up to 32,768 core memory words (eight 4K pages) in the basic 19-in. cabinet. Expansion beyond 32,768 words requires the addition of another cabinet to the system configuration. Memory communicates with the central processor and the I/O processor on the bidirectional memory bus (see Figure 2-8).

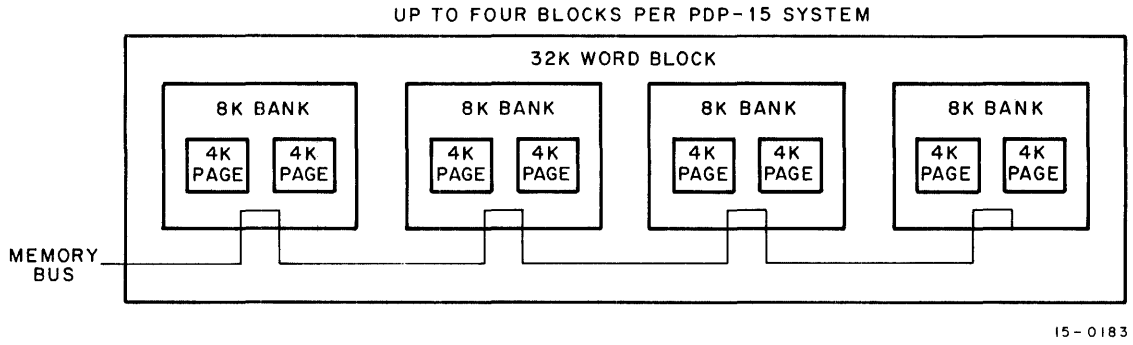


Figure 2-8 Physical Memory Organization

2.4.4 Memory Addressing

The PDP-15 memory system is broken down into four basic memory entities. The maximum configuration system contains 131,072 words of 18 or 19 bits and is subdivided into four blocks of 32K words. Each block contains up to four banks of 8K words, which contain two pages of 4K words. Figure 2-9 shows breakdown of locations, pages, banks, and blocks within the PDP-15 System.

Note that all valid addresses are positive addresses, i.e., negative addresses with bit 0 set (400000-777777) are illegal and cause the machine to wait indefinitely for memory response. Such addresses can be generated by the CPU or IPU under certain circumstances, but are trapped if the memory protect option is present.

2.4.5 Memory Port Switch

The memory port switch allows both the central processor and I/O processor to share core memory. In the event that both request a memory cycle simultaneously, the I/O processor is serviced first and the central processor must wait. However, if only one processor is using memory, both can process at the same time. For example, the central processor can be executing an EAE instruction, while the I/O processor transfers data out of memory to a DECdisk.

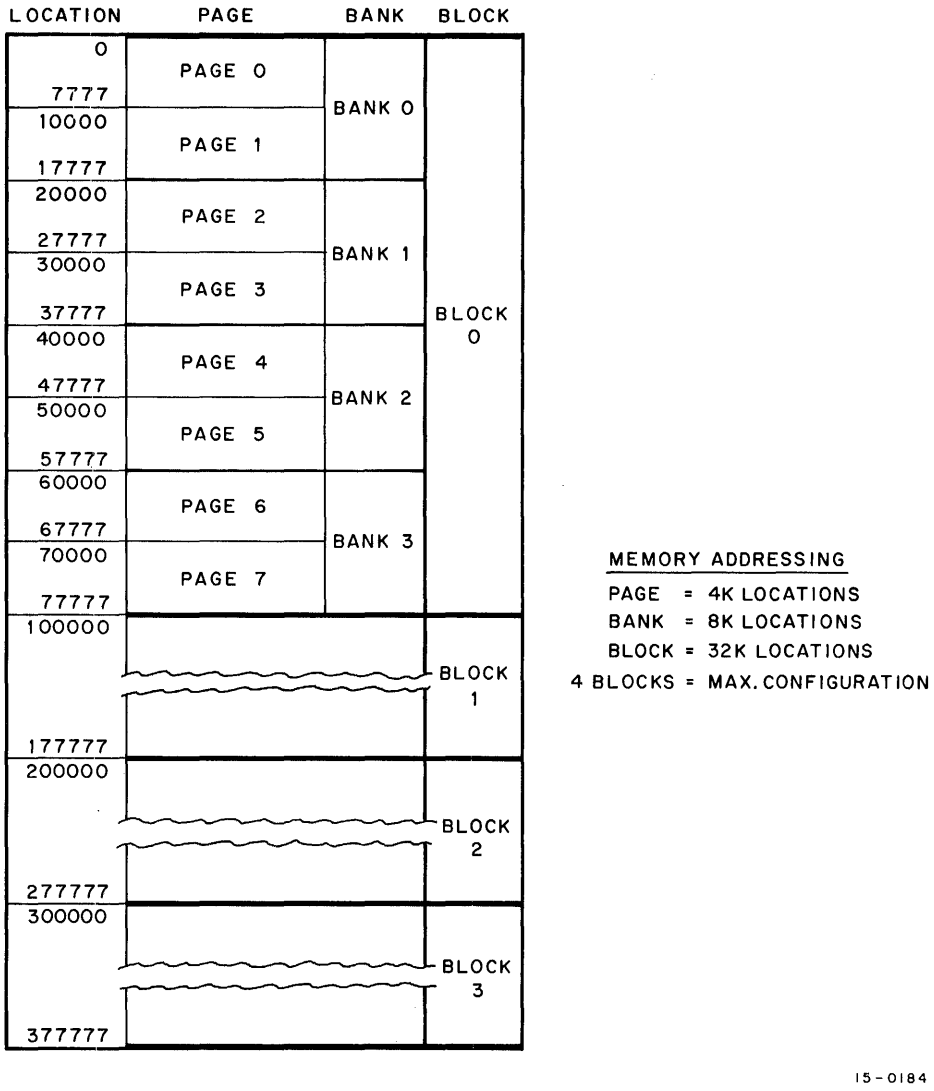


Figure 2-9 Memory Addressing

2.4.6 MX15-A Memory Bus Multiplexer

The MX15-A Memory Bus Multiplexer is a multiport memory option. It provides three ports for multiprocessor configurations, direct memory access (DMA) facilities, and the KP15-A Dual Bus Processor option. The MX15-A is a prerequisite for systems with greater than 32K of core memory. A PDP-15 System can accommodate up to four MX15-A Memory Bus Multiplexers. Each port has its own set of address switches that can be preset in any 8K increment. This feature enables one processor to address an 8K bank of core memory as its lowest bank (bank 0). A second processor can access the same bank of core memory through the MX15-A as its highest bank. The MX15-A introduces a delay for each memory cycle. Refer to the MX15-A Maintenance Manual for specific delay times introduced by the MX15-A.

Chapter 3
Instruction Formats

3.1 GENERAL

The PDP-15 instruction set is divided into "memory reference instructions," which address core memory, and "augmented instructions," which do not address core memory. Memory reference instructions address, either directly or indirectly, core memory locations for the purpose of retrieving, entering, or modifying the contents. The augmented instructions are used to execute a certain action or actions. This type of instruction is subdivided into four groups: operate instructions (link and accumulator operations including rotates, skips, clears, and complements); IOT instructions (input/output transfer of data, command and status between the central processor, and peripheral devices); EAE (extended arithmetic element, optional hardware multiply, divide, shift, and normalize); and index instructions (accumulator, limit register, and index register transfers, clears, additions, and skips).

3.2 MEMORY REFERENCE INSTRUCTION FORMAT

The memory reference instruction word consists of an operation code, an indirect address bit, an index bit, and an operand address (see Figure 3-1). The operation code, bits 0 through 3, specifies one of the 13 PDP-15 memory reference instructions. When the PDP-15 is in "page mode," the indirect bit indicates whether the 12-bit (bits 6-17) operand address is to be directly or indirectly (bit 4=1) addressed and the index bit determines whether or not the index register should be added to the operand address. In "bank mode," the indirect bit indicates whether the 13-bit (bits 5-17) operand address is

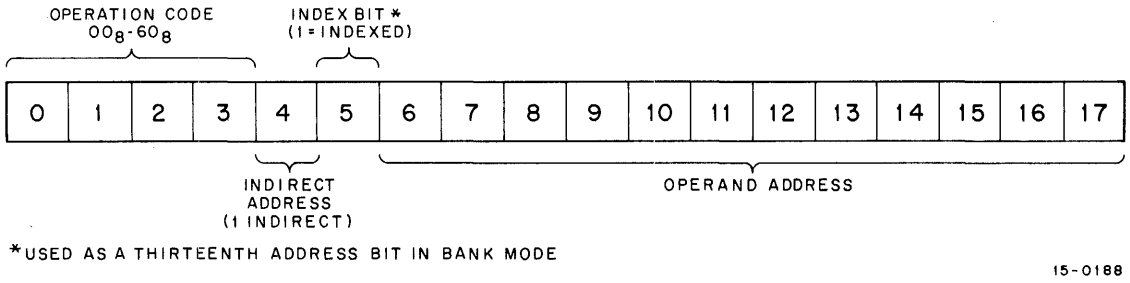


Figure 3-1 Memory Reference Instruction Word

to be used as the direct address or the indirect address (bit 4=1). The operand address is used in generating the effective address or the address in memory which will be referenced. Chapter 4 is a detailed description of addressing.

3.3 AUGMENTED INSTRUCTION FORMAT

The augmented instruction word (see Figure 3-2) consists of an operation code and an instruction code. The operation code designates whether the instruction is an extended arithmetic element instruction, 64_8 (bits 0-3), an Input/Output transfer instruction, 70_8 (bits 0-5), an Index instruction, 72_8 (bits 0-5) or an operate instruction 74_8 (bits 0-3). The instruction code designates which action is to be taken by the augmented instruction. An important and useful feature of the PDP-15 augmented instruction is its microprogramming capability. Multiple instruction codes having the same operation code can be combined to form one instruction word. Execution of all microprogrammed functions occurs during the time allocated to the type of instruction (operate instructions require one machine cycle, IOTs require two, three, or four cycles, EAE requires one or three, plus a variable time interval to complete their function, and index instructions require two cycles). Thus, microprogramming decreases program running time, lessens the number of instruction words required, and simplifies programming efforts.

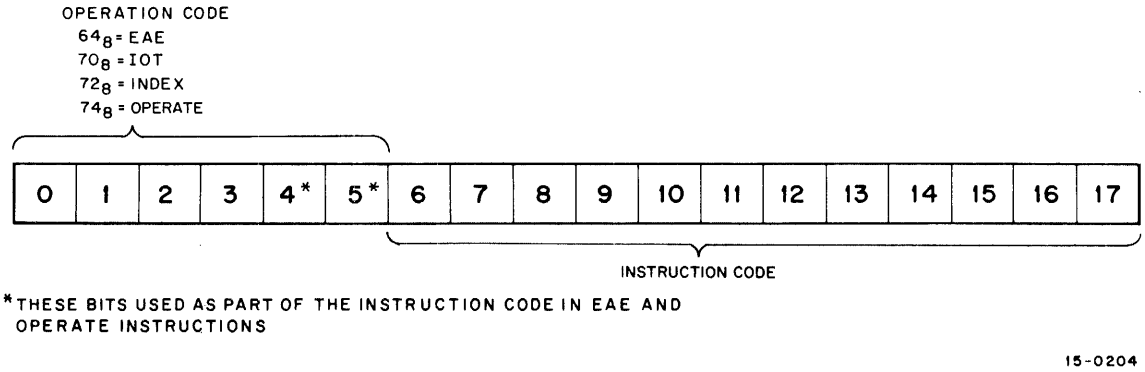


Figure 3-2 Augmented Instruction Format

3.4 TIMING

The amount of time required to perform each instruction is expressed in the number of machine cycles. The length of each machine cycle for various configurations is given in Table 3-1.

Instructions which indirectly address memory require one extra machine cycle in order to fetch and compute the indirect address. Only one level of indirect addressing is possible on the PDP-15.

Instructions which use the auto increment locations indirectly require two extra machine cycles; one for the increment of the location, and one for the indirect address.

Table 3-1
PDP-15 Central Processor Cycle Times, Basic and Expanded Configurations*

Configuration	Not In User Mode				In User Mode			
	Read		Write		Read		Write	
	Max	Typical	Max	Typical	Max	Typical	Max	Typical
Basic	800	800	800	800	800	800	800	800
KM15 Memory Protect	830	800	830	800	830	800	975	920
KM15 Memory Protect and KT15 Memory Protect/Relocate	965	880	965	880	1165	1080	1165	1080
MP15 Memory Parity	1100	1050	1100	1050	1100	1050	1100	1050
MP15 Memory Parity and KM15 Memory Protect	1130		1130		1130		1255	
MP15 Memory Parity, KM15 Memory Protect and KT15 Memory Protect/Relocate	1155		1155		1355		1355	

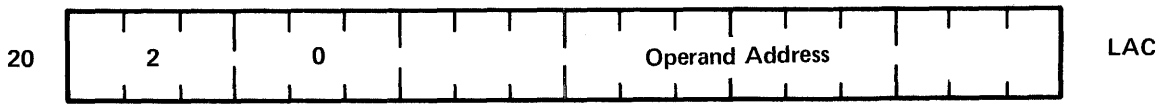
*All times indicated in nanoseconds. Refer to MX15-A Maintenance Manual or KP15-A Supplement for cycle times for MX15-A and KP15-A options.

3.5 MEMORY REFERENCE INSTRUCTIONS

In the memory reference instruction descriptions, and in succeeding paragraphs that describe other types of instructions, the following symbols are used:

Symbol	Definition
Y	The effective address of the memory location
V	Logic inclusive-OR
→	Indicates contents transferred from register or location preceding arrow to register or location following arrow.
∧	Logic AND
⊻	Logic exclusive-OR
$\overline{AC}$	Overscore indicates complemented contents of register or location
+	Addition

LOAD THE ACCUMULATOR



Mnemonic Name: LAC

Octal Code: 20

Time: 2 cycles

Operation: The contents of the effectively addressed memory location, Y, are read into the AC. The contents of Y are unchanged, the previous contents of the AC are lost.

Symbolic: Y → AC

DEPOSIT THE ACCUMULATOR

Mnemonic Name: DAC

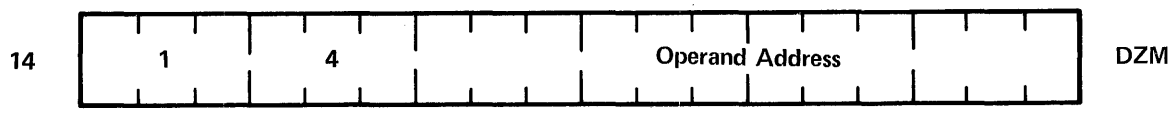
Octal Code: 04

Time: 2 cycles

Operation: The contents of the AC are deposited in the effectively addressed memory location Y. The contents of the AC are unchanged; the previous contents of Y are lost.

Symbolic: AC → Y

DEPOSIT ZERO IN MEMORY



Mnemonic Name: DZM

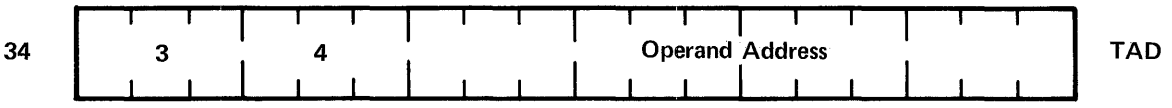
Octal Code: 14

Time: 2 cycles

Operation: An all-zeros data word is deposited in the effectively addressed memory location Y. The previous contents of Y are lost; the contents of the AC are unchanged.

Symbolic: 0 → Y

ADD (2's Complement)



Mnemonic Name: TAD

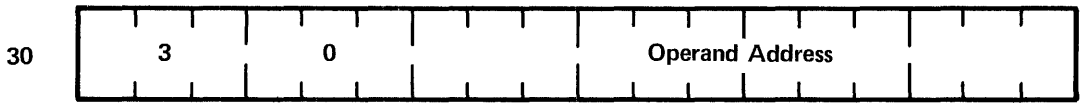
Octal Code: 34

Time: 2 cycles

Operation: The contents of the effectively addressed memory location Y, are added to the contents of the AC, following the rules of 2's complement arithmetic. The result is left in the AC. An arithmetic carry from AC_0 complements the link. The contents of Y are unchanged; the previous contents of the AC are lost.

Symbolic: Y + (L,AC) → (L,AC)

ADD (1's Complement)



Mnemonic Name: ADD

Octal Code: 30

Time: 2.3 cycles

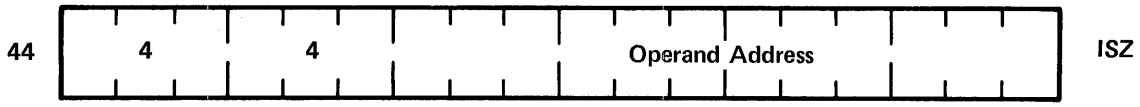
Operation: The contents of the effectively addressed location, Y, are added to the contents of the AC, following the rules of 1's complement arithmetic. The result is left in the AC. An arithmetic overflow sets the link to the binary 1 state. The contents of the AC is lost. The previous content of the link is lost. Overflow occurs if the magnitude (absolute) of the algebraic sum of the operands exceeds $2^{17}-1$; if the operands were of like sign and the result is signed differently, overflow has occurred to set the link. Overflow cannot occur if the operands are of different sign.

NOTE

The link should be cleared prior to the ADD instruction, if an arithmetic overflow check is desired.

Symbolic: Y + AC → AC
L V Overflow → L

INCREMENT AND SKIP IF ZERO



Mnemonic Name: ISZ

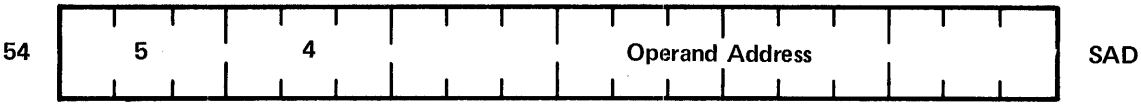
Octal Code: 44

Time: 3 cycles

Operation: The contents of the effectively addressed memory location, Y, are incremented by one (in 2's complement arithmetic) and tested. If Y now contains an all-zero word, the PC is incremented by one to skip the next instruction. If the contents of Y, after being incremented, are other than zero, the next instruction is executed. The previous contents of Y are lost; the contents of the AC are unchanged.

Symbolic: If $Y + 1 = 0$, $PC + 1 \rightarrow PC$
$Y + 1 \rightarrow Y$

SKIP IF AC DIFFERS



Mnemonic Name: SAD

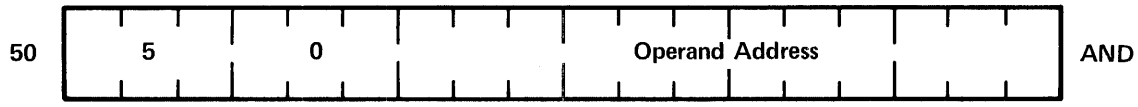
Octal Code: 54

Time: 2 cycles

Operation: The contents of the effectively addressed memory location, Y, are compared with the contents of the AC. If they differ, the PC is incremented by one to skip the next instruction. If they are the same binary quantity, the next instruction is executed. The contents of Y and the contents of the AC are unchanged.

Symbolic: If $Y \neq AC$, $PC + 1 \rightarrow PC$

BOOLEAN AND



Mnemonic Name: AND

Octal Code: 50

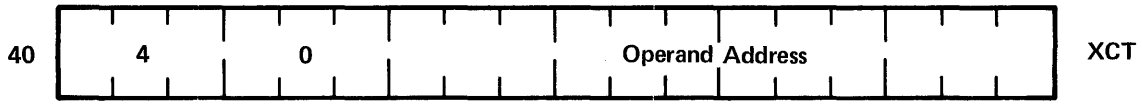
Time: 2 cycles

Operation: The contents of the effectively addressed memory location, Y, are logically ANDed with the contents of the AC on a bit-by-bit basis. The result is left in the AC. If corresponding, Y and AC bits are in the 1 state, the AC bit remains a 1; otherwise, the AC bit is cleared to the 0 state. The contents of Y are unchanged; the previous contents of the AC are lost.

Symbolic: $Y \wedge AC \rightarrow AC$

AND		AC	
		0	1
Y	0	0	0
	1	0	1

EXECUTE THE INSTRUCTION AT Y



Mnemonic Name: XCT

Octal Code: 40

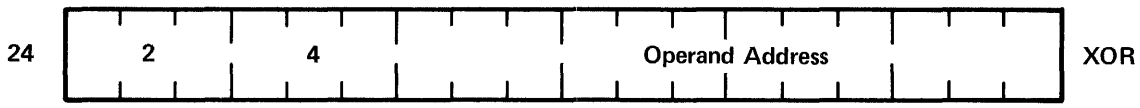
Time: 1 cycle plus time of instruction at Y

Operation: The computer executes the instruction located at the effectively addressed memory location, Y. The contents of the PC are unchanged unless Y contains a JMS, CAL, JMP, or skip instruction, each of which changes the contents of the PC to alter the program sequence. XCT could be thought of as a single-instruction subroutine causing a quasi-jump to Y, execution of the instruction specified there, and return to the program sequence (i.e., execution of the instruction following XCT) if the instruction has not changed the PC.

With the Memory Protect option installed, the XCT of an XCT instruction is not allowed when in USER mode.

Symbolic: $Y_{0-5} \rightarrow IR$

BOOLEAN EXCLUSIVE OR



Mnemonic Name: XOR

Octal Code: 24

Time: 2 cycles

Operation: The contents of the effectively addressed memory location, Y, are exclusively-ORed with the contents of the AC, on a bit-by-bit basis. The result is left in the AC. If corresponding Y and AC bits are in the same binary state (i.e., 1 or 0), the AC bit is cleared to the 0 state. If the corresponding bits differ in state, the AC bit is set to the 1 state. The contents of Y are unchanged. The previous contents of the AC are lost.

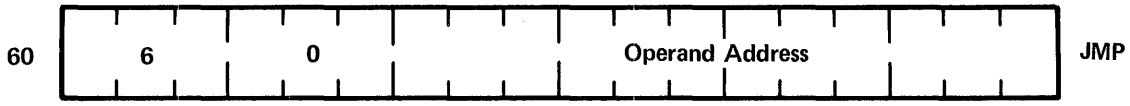
NOTE

The XOR instruction causes the operand to complement its original content only in those bits that have 1's in the accumulator mask.

Symbolic: $Y \veebar AC \rightarrow AC$

XOR		AC	
		0	1
Y	0	0	1
	1	1	0

UNCONDITIONAL JUMP



Mnemonic Name: JMP

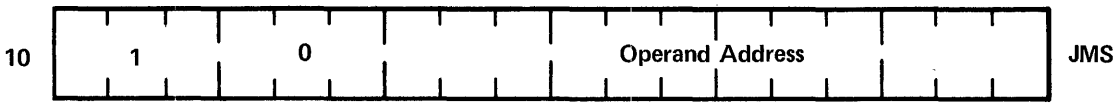
Octal Code: 60

Time: 1 cycle

Operation: A new address is computed from the operand address of the jump instruction and transferred to the PC. The next instruction fetched will be from the memory location specified by the new address. The contents of the AC are unchanged.

Symbolic: $Y_{5-17} \rightarrow PC$

JUMP TO SUBROUTINE



Mnemonic Name: JMS

Octal Code: 10

Time: 2 cycles

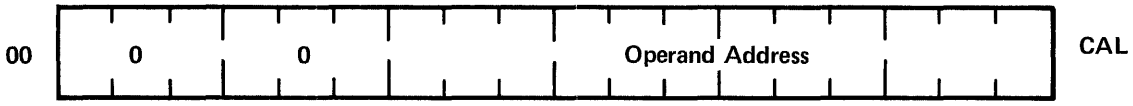
Operation: The contents of the PC and the link, and the status (on or off) of bank mode and user mode are deposited in the effectively addressed memory location, Y. The next instruction is read from the contents of memory location Y + 1, breaking the previous program sequence and starting a new sequence from Y + 1. The contents of the PC are changed, and the contents of the AC are unchanged.

When not in the user mode, or when the memory protect option is not installed, a free instruction follows the JMS. Therefore, a PI or API break cannot occur after the execution of the JMS instruction, but may occur after the execution of the next instruction.

Symbolic:

$L \rightarrow Y_0$
$BM \rightarrow Y_1$
$UM \rightarrow Y_2$
$PC \rightarrow Y_{3-17}$
$Y_{5-17} + 1 \rightarrow PC$

CALL (JUMP TO) SUBROUTINE



Mnemonic Name: CAL

Octal Code: 00

Time: 2 cycles

Operation: The CAL instruction is the equivalent of a JMS 20 instruction. The contents of the PC and the link, and the status (on or off) of bank mode and user mode are deposited in memory location 20. The next instruction is read from memory location 21, breaking the previous program sequence and starting a new sequence from 21. The contents of the AC are unchanged. If the API option is present and enabled, priority level 4 will be activated after the execution of a CAL instruction if no higher priority level is set.

When not in user mode, or when the memory protect option is not installed, a free instruction follows the CAL. Therefore, a PI or API break cannot occur after the execution of the JMS instruction, but may occur after the execution of the next instruction.

Symbolic:

$L \rightarrow 20_0$

$BM \rightarrow 20_1$

$UM \rightarrow 20_2$

$PC \rightarrow 20_{3-17}$

"21" → PC

3.6 AUGMENTED INSTRUCTIONS

3.6.1 Operate Instructions

Operate instructions (operation code of 74_8) are used to sense and/or alter the contents of the AC and link. Typical functions are: conditional or unconditional skips, complementing, setting, clearing, or rotating the contents of the two registers jointly or independently and incrementing the AC. A Halt (HLT) instruction is included. Operates are performed in one machine cycle, the actions being specified by the microprogramming of the instruction code. Each bit of the 14-bit instruction code can effect a unique response; hence, they are "microinstructions" to the computer. The important feature of the operate class is its microprogramming capability, where two or three microinstructions

can be combined to form one instruction word and, therefore, be executed in one cycle. Those microinstructions that logically conflict and occur at the same time should not be microprogrammed. Figure 3-3 illustrates the bit configuration of the instruction code. Figure 3-4 shows the allowable combinations of microinstructions.

When noninverted skip actions are microprogrammed (bit 8 is 0), the conditions to be met are inclusively ORed. For example, if SZA (740200) and SNL (740400) are combined (740600), the skip takes place if either or both conditions are present (contents of the AC are 0 or the content of the link is not 0).

When inverted skip actions are microprogrammed (bit 8 is 1), the skip occurs only if the AND of the conditions is met. For example, when SNA (741200) and SZL (741400) are specified in a microprogrammed instruction (741600), the skip occurs only if both conditions are present (the contents of the AC are other than 0 and the content of the link is 0).

Programming Note

The PDP-15 Symbolic Assembler accepts either HLT or XX (see Figure 3-4) as a valid mnemonic for the operate class instruction to stop program execution. The latter facilitates visual scanning of a program listing to determine the occurrence of program halts.

								Bit 7=0				
CLA	CLL	Additional Rotate	0=OR of	SNL	SZA	SMA	HLT	RAR	RAL	OAS	CML	CMA
			1=AND of	SZL	SNA	SPA		RTR	RTL			
								Bit 7=1				
5	6	7	8	9	10	11	12	13	14	15	16	17

NOTE: Bits 7, 13, and 14 set: SWHA
Bits 13 and 14 set: IAC

Figure 3-3 Instruction Bit Configuration

Order of Events	Column 1	Column 2	Column 3		Column 4
Level 1	SNL SZA SMA	CLA CLL	OAS CMA	CML / IAC	HLT
Level 2	SZL SNA SPA		RAR or RAL		
Level 3	SKP		RTR or RTL or SWHA		

1. Combine instructions from left to right.
2. Any instructions in a box can be combined, except the rotate instructions.
3. Instructions on different levels cannot be combined if they are in the same column. Instructions on any level can be combined if they are in different columns. (e.g., SZA!SMA!CLA!OAS!HLT! is legal - SZA!SPA is not legal.)
4. CML and IAC cannot be combined. Either one can be combined with OAS and/or CMA (e.g., OAS!CMA!CML or OAS!CMA!IAC).
5. Instructions occur in order from column 1 to column 4.

NOTE

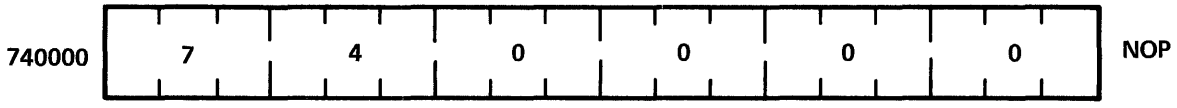
Level 1 skips (SNL, SZA, SMA) will occur if any one of the combined tests is satisfied (an OR condition).

Level 2 skips (SZL, SNA, SPA) will occur only if all the combined tests occur (an AND condtion).

Combined rotates become a SWHA or an IAC, depending on bit 7.

Figure 3-4 Allowable Microinstruction Combinations

NO OPERATION



Mnemonic Name: NOP

Octal Code: 740000

Time: 1 cycle

Operation: The program delays for one cycle before the next instruction is fetched.

Symbolic: Not applicable.

COMPLEMENT ACCUMULATOR

Mnemonic Name: CMA

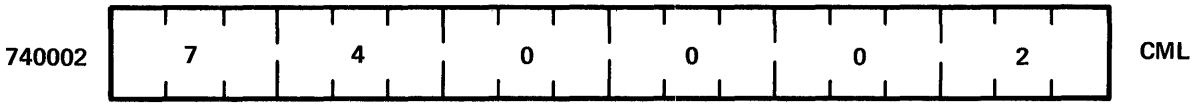
Octal Code: 740001

Time: 1 cycle

Operation: Each bit of the AC is set or cleared to the inverse of its current state. The previous contents of the AC are lost.

Symbolic: $\overline{AC} \rightarrow AC$

COMPLEMENT LINK



Mnemonic Name: CML

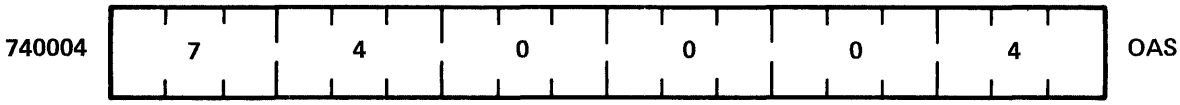
Octal Code: 740002

Time: 1 cycle

Operation: The link is set or cleared to the inverse of its current state. It previous content is lost.

Symbolic: $\overline{L} \rightarrow L$

INCLUSIVE OR ACCUMULATOR SWITCHES



Mnemonic Name: OAS

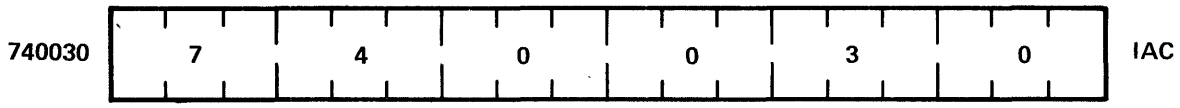
Octal Code: 740004

Time: 1 cycle

Operation: The word set up by manual positioning of the DATA switches is inclusively ORed with the contents of the AC on a bit-by-bit basis. The result is left in the AC. If corresponding, AC and DATA switch bits are in the binary 0 state, the AC bit remains 0. If either or both of the corresponding bits are in the binary 1 state, the AC bit is set to 1. The previous contents of the AC are lost. The switch settings are not affected.

Symbolic: AC V DATA switch → AC

INCREMENT THE ACCUMULATOR



Mnemonic Name: IAC

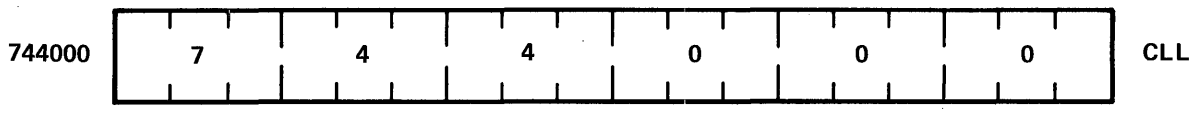
Octal Code: 740030

Time: 1 cycle

Operation: The contents of the accumulator are incremented by one and the results placed in the accumulator. The previous contents of the accumulator are lost. When overflow occurs bit zero complements the link.

Symbolic: AC + 1 → AC

CLEAR THE LINK



Mnemonic Name: CLL

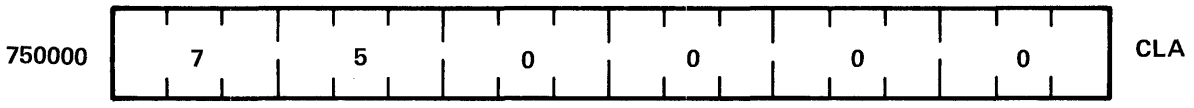
Octal Code: 744000

Time: 1 cycle

Operation: The content of the link is cleared to the binary 0 state.

Symbolic: 0 → L

CLEAR THE ACCUMULATOR



Mnemonic Name: CLA

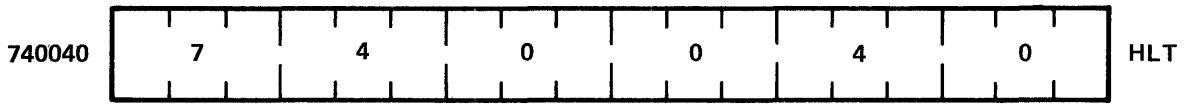
Octal Code: 750000

Time: 1 cycle

Operation: Each bit of the AC is cleared to the binary 0 state. The previous contents are lost.

Symbolic: 0 → AC

HALT PROGRAM



Mnemonic Name: HLT

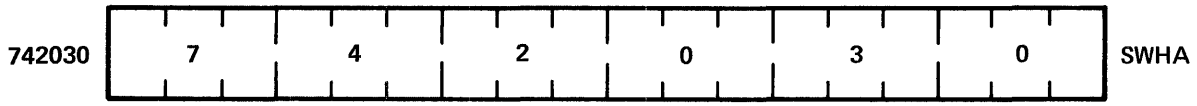
Octal Code: 740040

Time: 1 cycle

Operation: Program execution stops at completion of the current machine cycle. The run indicator is turned off.

Symbolic: 0 → RUN flip-flop

SWAP HALVES OF THE ACCUMULATOR



Mnemonic Name: SWHA

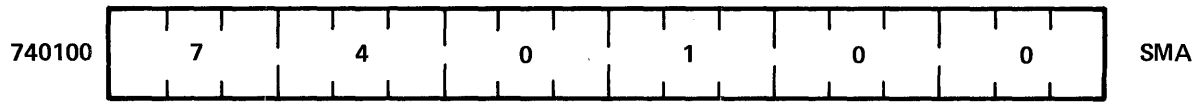
Octal Code: 742030

Time: 1 cycle

Operation: This instruction places the contents of AC bits 0-8 into AC bits 9-17 and at the same time places AC bits 9-17 into AC bits 0-8. The previous contents of the AC are lost.

Symbolic: $AC_{0-8} \rightarrow AC_{9-17}$
$AC_{9-17} \rightarrow AC_{0-8}$

SKIP ON MINUS ACCUMULATOR



Mnemonic Name: SMA

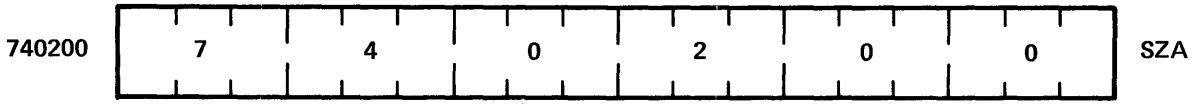
Octal Code: 740100

Time: 1 cycle

Operation: Test the contents of the sign bit, AC_0, of the data word in the AC. If the bit is in the binary 1 state, the contents of the PC are incremented by one to skip the next instruction. If AC_0 is found to be in the 0 state, the next instruction is executed. The contents of the AC are unchanged.

Symbolic: If $AC_0 = 1$, PC + 1 → PC

SKIP ON ZERO ACCUMULATOR



Mnemonic Name: SZA

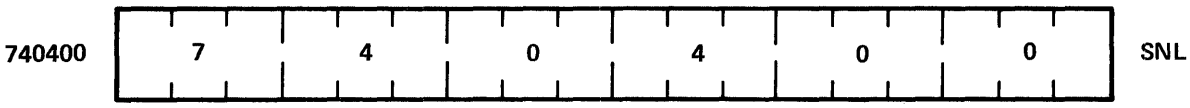
Octal Code: 740200

Time: 1 cycle

Operation: Test the contents of the word in the AC. If all bits are binary 0s, the quantity is taken to be zero (2's complement notation), and the contents of the PC are incremented by one to skip the next instruction. If any bit is in the binary 1 state, the next instruction is executed. The contents of the AC are unchanged.

Symbolic: If AC=0, PC + 1 → PC

SKIP ON NON-ZERO LINK



Mnemonic Name: SNL

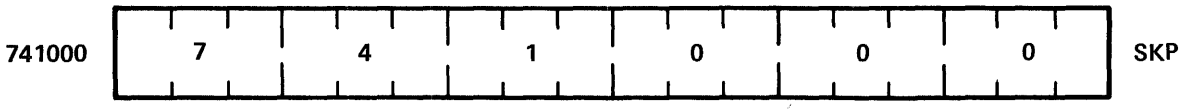
Octal Code: 740400

Time: 1 cycle

Operation: Test the content of the link. If the link is in the binary 1 state, the contents of the PC are incremented by one to skip the next instruction. If the link has a binary 0, the next instruction is executed. The content of the link is unchanged.

Symbolic: If L = 1, PC + 1 → PC

UNCONDITIONAL SKIP



Mnemonic Name: SKP

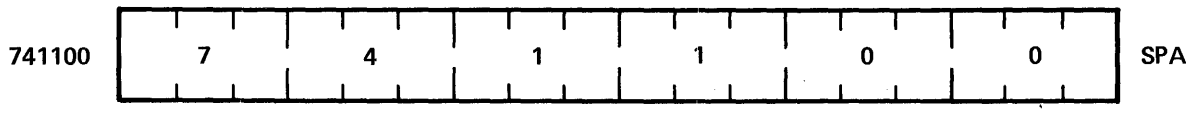
Octal Code: 741000

Time: 1 cycle

Operation: The contents of the PC are incremented by one to cause an unconditional skip of the next instruction.

Symbolic: PC + 1 → PC

SKIP ON POSITIVE ACCUMULATOR



Mnemonic Name: SPA

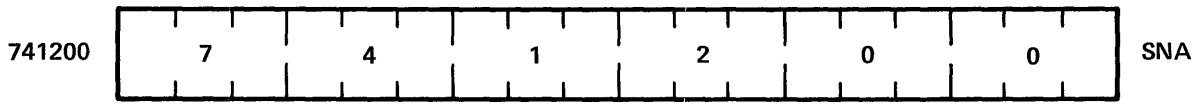
Octal Code: 741100

Time: 1 cycle

Operation: Test the contents of the sign bit, AC_0, of the data word in the AC. If the bit is in the binary 0 state, the quantity in the AC is taken to be positive. Therefore, the contents of the PC are incremented by one to skip the next instruction. If the bit is found to be in the binary 1 state, the next instruction is executed. The contents of the AC are unchanged.

Symbolic: If $AC_0 = 0$, PC + 1 → PC

SKIP ON NON-ZERO ACCUMULATOR



Mnemonic Name: SNA

Octal Code: 741200

Time: 1 cycle

Operation: Test the contents of the data word in the AC. If any bit is in the binary 1 state, the quantity is taken to be unequal to zero (2's complement notation only), and the contents of the PC are incremented by one to skip the next instruction. If all bits are found to be in the 0 state, the quantity is considered to be zero and the next instruction is executed. The contents of the AC are unchanged.

Symbolic: If AC ≠ 0, PC + 1 → PC

SKIP ON ZERO LINK

Mnemonic Name: SZL

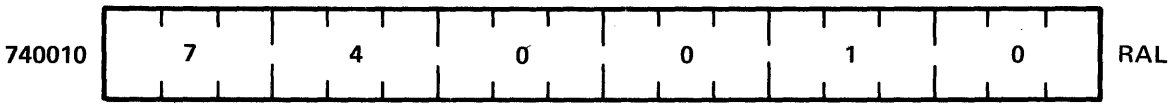
Octal Code: 741400

Time: 1 cycle

Operation: Test the contents of the link. If the link is in the binary 0 state, the contents of the PC are incremented by one to skip the next instruction. If the link has a binary 1, the next instruction is executed. The contents of the link is unchanged.

Symbolic: If L = 0, PC + 1 → PC

ROTATE AC AND LINK LEFT



Mnemonic Name: RAL

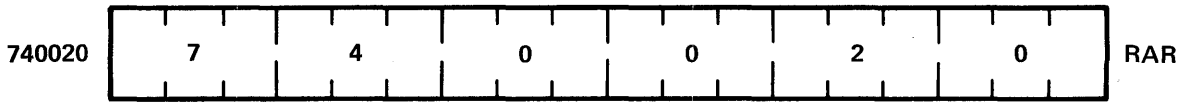
Octal Code: 740010

Time: 1 cycle

Operation: The contents of the AC and the link are rotated one bit position to the left with AC_0 entering the link and the link entering AC_{17}.

Symbolic:
$AC_i \rightarrow AC_{i-1}$; i=1,17
$AC_0 \rightarrow L$
$L \rightarrow AC_{17}$

ROTATE AC AND LINK RIGHT



Mnemonic Name: RAR

Octal Code: 740020

Time: 1 cycle

Operation: The contents of the AC and the link are rotated one bit position to the right with AC_{17} entering the link and the link entering AC_0.

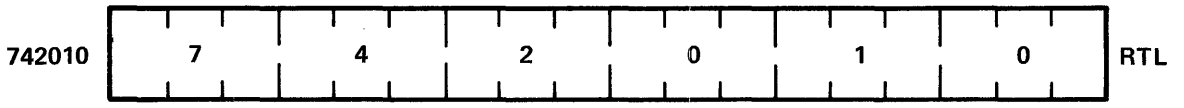
Symbolic:

$AC_i \rightarrow AC_{i+1}$; i=0,16

$AC_{17} \rightarrow L$

$L \rightarrow AC_0$

ROTATE AC AND LINK TWO LEFT



Mnemonic Name: RTL

Octal Code: 742010

Time: 1 cycle

Operation: The contents of the AC and the link are rotated two bit positions to the left with AC_0 entering AC_{17}, AC_1 entering the link, and the link entering AC_{16}.

Symbolic:

$AC_i \rightarrow AC_{i-2}$; 1=2,17

$L \rightarrow AC_{16}$

$AC_0 \rightarrow AC_{17}$

$AC_1 \rightarrow L$

ROTATE AC AND LINK TWO RIGHT

Mnemonic Name: RTR

Octal Code: 742020

Time: 1 cycle

Operation: The contents of the AC and the link are rotated two bit positions to the right with the link entering AC_1, AC_{17} entering AC_0, and AC_{16} entering the link.

Symbolic:

$AC_i \rightarrow AC_{i+2};\ i=0,15$

$L \rightarrow AC_1$

$AC_{17} \rightarrow AC_0$

$AC_{16} \rightarrow L$

2'S COMPLEMENT ACCUMULATOR

Mnemonic Name: TCA

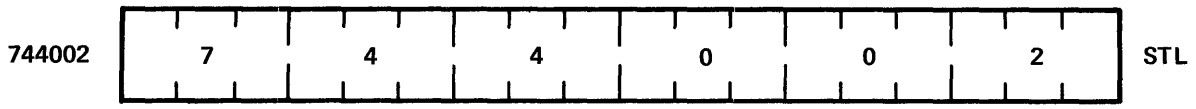
Octal Code: 740031

Time: 1 cycle

Operation: A microcoded instruction combines the complement of the AC and increments the AC, thereby, performing a 2's Complement Operation on the contents of the AC and placing the result in the AC. The previous contents of the AC are lost.

Symbolic: $\overline{AC} + 1 \rightarrow AC$

SET THE LINK



Mnemonic Name: STL

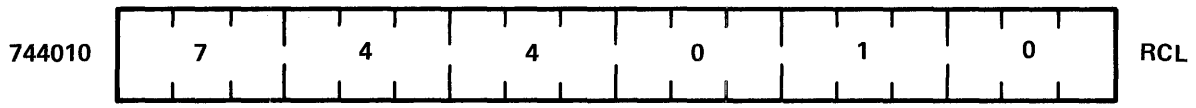
Octal Code: 744002

Time: 1 cycle

Operation: A microcoded instruction equivalent to CLL+CML. The link is first cleared to contain a binary 0; it is then complemented to contain a binary 1.

Symbolic: $1 \rightarrow L$

CLEAR LINK, THEN ROTATE AC AND L LEFT



Mnemonic Name: RCL

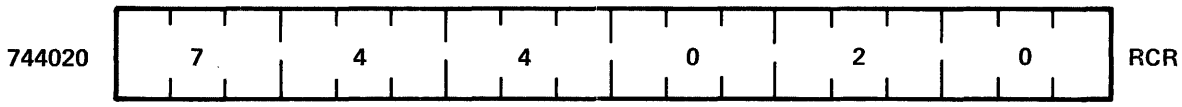
Octal Code: 744010

Time: 1 cycle

Operation: A microcoded instruction equivalent to CLL+RAL. The link is first cleared to the binary 0 state: then the contents of the AC and the link are rotated one bit position to the left.

Symbolic: $AC_i \rightarrow AC_{i-1;i=1,17}$
$AC_0 \rightarrow L$
$0 \rightarrow AC_{17}$

CLEAR LINK, THEN ROTATE AC AND L RIGHT



Mnemonic Name: RCR

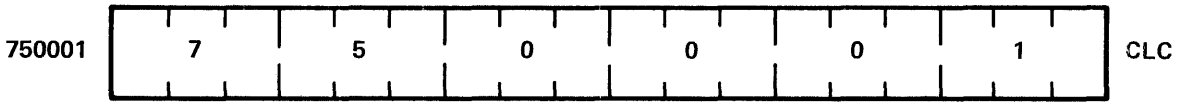
Octal Code: 744020

Time: 1 cycle

Operation: A microcoded instruction equivalent to CLL+RAR. The link is first cleared to the binary 0 state; then the contents of the AC and the link are rotated one bit position to the right.

Symbolic: $AC_i \rightarrow AC_{i+1}$; i=0,16
$AC_{17} \rightarrow L$
$0 \rightarrow AC_0$

CLEAR AND COMPLEMENT ACCUMULATOR



Mnemonic Name: CLC

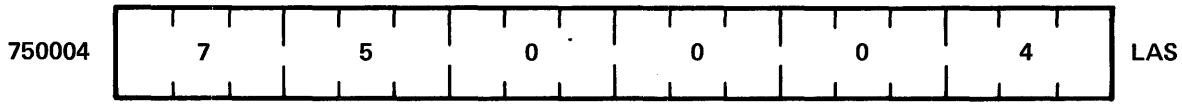
Octal Code: 750001

Time: 1 cycle

Operation: A microcoded instruction equivalent to CLA + CMA. Each bit of the AC is cleared to the binary 0 state. Then each bit is set to the binary 1 state. The previous contents of the AC are lost.

Symbolic: 777777 → AC

LOAD AC FROM ACCUMULATOR SWITCHES



Mnemonic Name: LAS

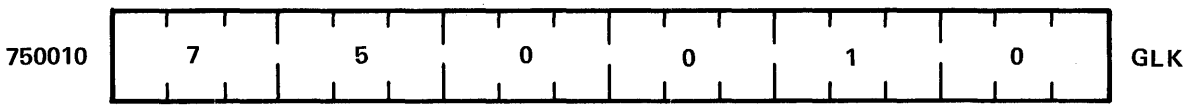
Octal Code: 750004

Time: 1 cycle

Operation: A microcoded instruction equivalent to CLA + OAS. Each bit of the AC is cleared to the binary 0 state. Then the word set up by manual positioning of the DATA switches is entered in the AC. The previous contents of the AC are lost. The switch settings are not affected.

Symbolic: DSW → AC

GET THE LINK



Mnemonic Name: GLK

Octal Code: 750010

Time: 1 cycle

Operation: A microcoded instruction equivalent to CLA + RAL. Each bit of the AC is cleared to the binary 0 state. Then the contents of the AC and the link are rotated one bit position left with the link contents entering AC_{17}. The previous contents of the AC are lost.

Symbolic:

$L \rightarrow AC_{17}$

$0 \rightarrow AC_{0-16}$

$0 \rightarrow L$

LOAD AC WITH "n"

Mnemonic Name: LAW

Octal Code: 760000 + n (n = 13-bit number)

Time: 1 cycle

Operation: A single-cycle instruction that loads itself into the AC for the purpose of generating a negative number, n, of the range of $0 \leq n \leq 17777_8$. Following the fetch, the computer enters the contents of the MI (the LAW instruction word) in the AC. The previous contents of the AC are lost. The first five AC bits will always be loaded with 1s.

Symbolic: MI → AC

3.7 INPUT/OUTPUT TRANSFER INSTRUCTIONS

Input/Output transfer (IOT) instructions initiate transmission of signals via the I/O bus to control peripheral devices, sense their status, and effect information transfers between them and the central processor. PDP-15 IOT instructions contain the following information (see Figure 3-5):

a. An operation code of 70_8.

b. An 8-bit device selection code to differentiate between up to 256 peripheral devices (selection logic in a device's I/O bus interface responds only to its preassigned code). In normal practice, bits 6 through 11 perform the primary device differentiation between up to 64 devices with bits 12 and 13 coded to select an operational mode or subdevice. A number of these device codes are hardwired into the processor and cannot be used to control peripheral devices.

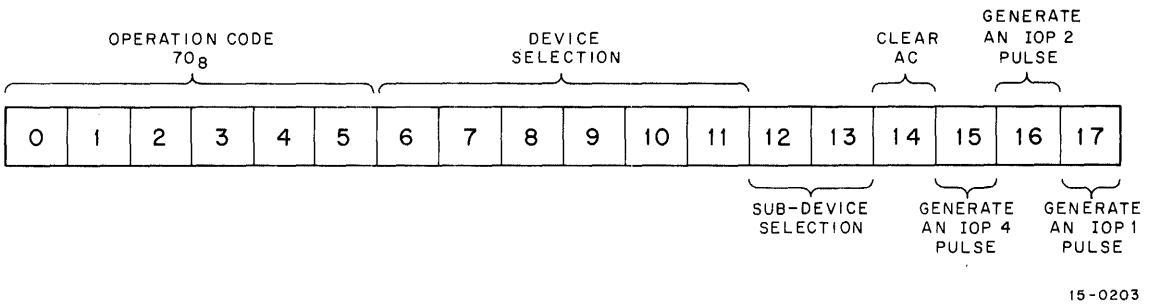


Figure 3-5 IOT Instruction Format

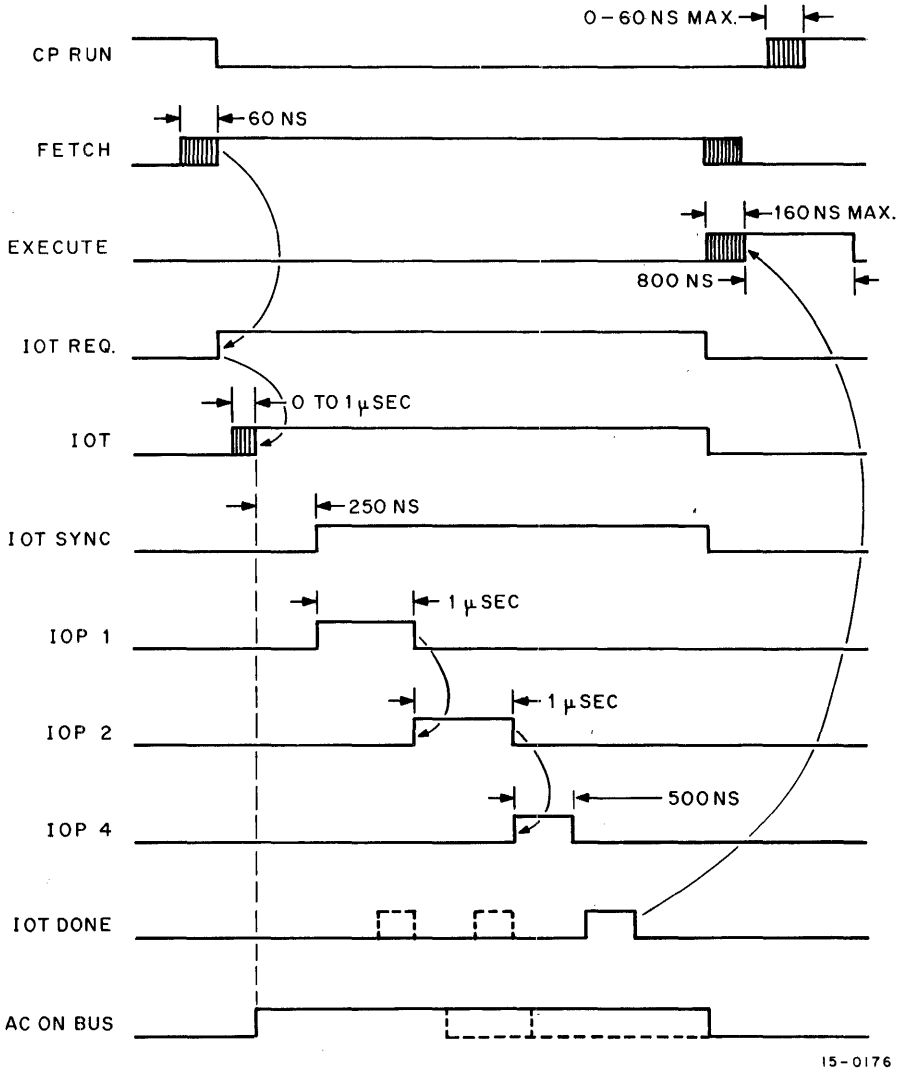


Figure 3-6 IOT Instruction Timing

c. A command code (bits 14 through 17) capable of being microprogrammed to clear the AC and issue up to three pulses via the I/O bus.

The four machine cycles required to execute an IOT instruction consists of decoding of the IOT from the central processor's memory input buffer synchronization of the central and I/O processors, issuing three sequential cycles of 1 μs each to ensure IOP pulses at event times 1, 2, and 4, and finally the fetch of the next instruction to be executed (see Figure 3-6). Bit 14 can be programmed to clear the accumulator at the start of an IOT instruction. Bits 15-17 can be microprogrammed in any manner to produce a pulse on the I/O bus for each bit set. Bit 17 causes an IOP 1 pulse, or the first pulse generated, and is normally used for testing the device status flags. Bit 16 generates an IOP 2 pulse, the second pulse, and can be used in transmitting to or from a device to the processors. On "In" transfers, data is ORed from the I/O bus into the accumulator; therefore, bit 14, clear the accumulator, is typically used when a load the accumulator from a device is needed. Bit 15 produces IOP 4 pulse, the third pulse, and is used for control and transfer of data from the accumulator to the device. A summary of IOP pulses is as follows:

a. IOP1 is normally used in an I/O skip instruction to test a device flag; however, it can be used as a command pulse or a load of a device. It cannot be used to initiate a "read from" a device.

b. IOP2 is usually used to transfer data from the device to the computer, or to clear a device information register; it cannot be used to determine a "skip" condition.

c. IOP4 is usually used to transfer data from the computer to the device; it cannot be used to determine a "skip" condition or to initiate a read from a device.

Programming Note

Execution of an IOT instruction and the next instruction in sequence cannot be interrupted; i.e., PDP-15 does not grant an interrupt request until the instruction following an IOT (and which is not an IOT itself) has completed its function.

3.7.1 PDP-15 IOTs

The following are internal PDP-15 IOTs:

IOT — Basic IOT Command — 700000

This instruction performs an IOT nop, i.e., it is an IOT which sets no device select or subdevice select bits nor produces any IOP pulses.

Execution Time: 2-3 μs

IOF — Turn Interrupt Off — 700002

Disables the Program Interrupt system of the PDP-15.

Execution Time: 3-4 μs

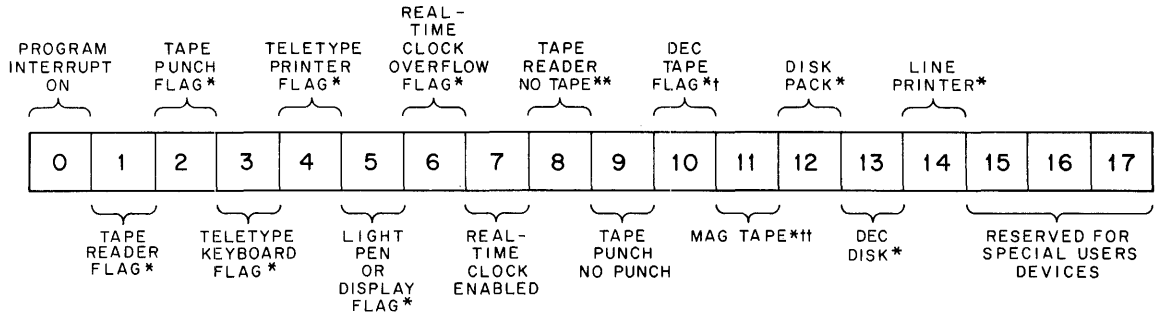
ION Turn Interrupt On 700042

Enables the Program Interrupt system of the PDP-15.

Execution Time: 3-4 μs

IORS I/O Read Status 700314

This IOT reads the status of various device flags into the AC. Figure shows the device and the bit to which each device flag is assigned.



* WILL CAUSE A PROGRAM INTERRUPT

*† INCLUSIVE OR OF TRANSFER COMPLETION AND ERROR FLAGS

*†† INCLUSIVE OR OF MTF AND EF

** CAUSES A PROGRAM INTERRUPT THROUGH THE READER FLAG

15-0202

CAF Clear All Flags 703302

CAF causes a power clear to be sent out on the I/O bus which will clear all device flags which call for I/O interrupt service, and stop I/O bus activity.

Execution Time: 3-4 μs

SPCO Skip if PC-15 703341

This instruction tests to determine whether a PC15 paper-tape reader unit is connected to the PDP-15. If it is, a skip will occur; if not, the next sequential instruction in the program will be executed.

Execution Time: 2-3 μs

SK15 Skip if PDP-15 707741

This is an unconditional skip in the PDP-15 but not in a PDP-4, 7, or 9.

Execution Time: 2-3 μs

SBA Skip if Bank Addressing 707761

This instruction will cause the next instruction to be skipped if the PDP-15 is in Bank Mode.

Execution Time: 2-3 μs

DBA Disable Bank Addressing 707762

Causes the PDP-15 to turn off Bank Mode and enter Page Mode. In Page Mode, the index register may be added to operand addresses in forming effective addresses.

Execution Time: 3-4 μs

EBA Enable Bank Addressing 707764

Causes the PDP-15 to enter Bank Mode and disables the index register from being used in the calculation of effective addresses.

Execution Time: 4-5 μs

3.7.2 Teletype Keyboard

KSF Skip on Keyboard Flag 700301

Tests the Teletype keyboard flag and causes the next instruction to be skipped if the flag is set, indicating that the keyboard control has assembled a character from the Teletype.

Execution Time: 2-3 μs

KRB Read Keyboard Buffer 700312

This IOT clears the AC and then reads the contents of the keyboard buffer into AC bits 10-17, and clears the keyboard flag.

Execution Time: 3-4 μs

KRS Keyboard Reader Select 700332

This IOT clears the AC, reads the contents of the keyboard buffer into AC bits 10-17 and enables the keyboard reader to advance another character. Reading from the keyboard reader is done in full duplex mode (no character echo). This IOT can also be used to read, full duplex, from Teletype keyboard.

Execution Time: 3-4 μs

3.7.3 Teletype Teleprinter

TSF Skip on Teleprinter Flag 700401

Tests the status of the teleprinter flag to determine if the last character has been printed. If the flag is set the next instruction will be skipped.

Execution Time: 2-3 μs

TCF Clear Teleprinter Flag 700402

Clears the teleprinter flag which had been set at the completion of the previous character.

Execution Time: 3-4 μs

TLS Load and Select Teleprinter 700406

Clears the teleprinter flag, loads the teleprinter buffer from AC bits 10-17 and initiates printing of the character. The flag is set when printing is completed.

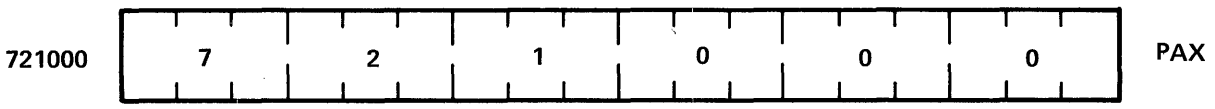
Execution Time: 4-5 μs

3.8 INDEX INSTRUCTIONS

The index instructions enable the programmer to transfer information between the accumulator, limit register, and index register, clear the limit register and index register, add a number contained in the instruction itself (±256) to the accumulator, limit register, or index register and test to determine if the index register is greater than or equal to the limit register.

All index instructions require two central processor cycles, but only one memory cycle, thus, allowing the central processor to perform operations at the same time as the I/O processor.

PLACE ACCUMULATOR IN INDEX REGISTER



Mnemonic Name: PAX

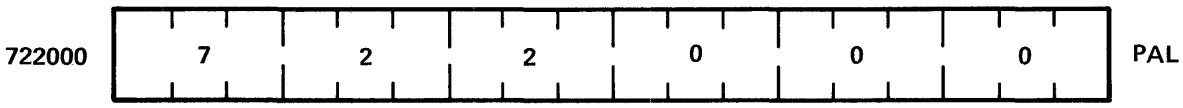
Octal Code: 721000

Time: 2 cycles (1 memory cycle)

Operation: The contents of the accumulator are transferred to the index register. The contents of the accumulator remain unchanged.

Symbolic: AC → XR

PLACE ACCUMULATOR IN LIMIT REGISTER



Mnemonic Name: PAL

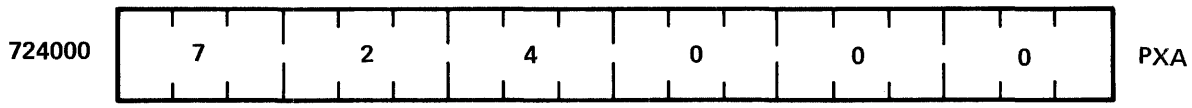
Octal Code: 722000

Time: 2 cycles (1 memory cycle)

Operation: The contents of the accumulator is transferred to the limit register. The contents of the accumulator remain unchanged.

Symbolic: AC → LR

PLACE INDEX REGISTER IN ACCUMULATOR



Mnemonic Name: PXA

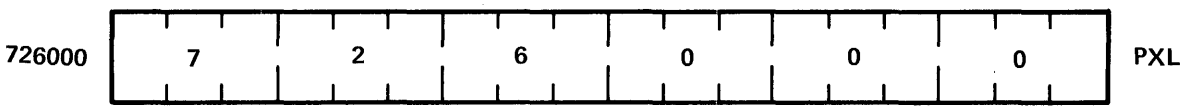
Octal Code: 724000

Time: 2 cycles (1 memory cycle)

Operation: The contents of the index register are transferred to the accumulator. The contents of the index register remain unchanged.

Symbolic: XR → AC

PLACE INDEX REGISTER IN LIMIT REGISTER



Mnemonic Name: PXL

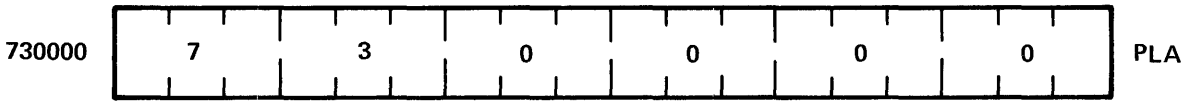
Octal Code: 726000

Time: 2 cycles (1 memory cycle)

Operation: The contents of the index register are transferred to the limit register. The contents of the index register remain unchanged.

Symbolic: XR → LR

PLACE LIMIT REGISTER IN ACCUMULATOR



Mnemonic Name: PLA

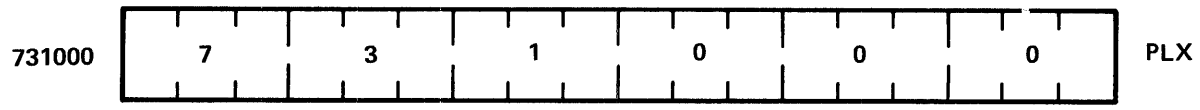
Octal Code: 730000

Time: 2 cycles (1 memory cycle)

Operation: The contents of the limit register are transferred to the accumulator. The contents of the limit register remain unchanged.

Symbolic: LR → AC

PLACE LIMIT REGISTER IN INDEX REGISTER



Mnemonic Name: PLX

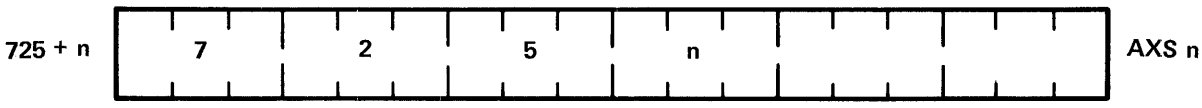
Octal Code: 731000

Time: 2 cycles (1 memory cycle)

Operation: The contents of the limit register are transferred to the index register. The contents of the limit register remain unchanged.

Symbolic: LR → XR

ADD n TO INDEX REGISTER AND SKIP IF EQUAL TO OR GREATER THAN THE LIMIT REGISTER



Mnemonic Name: AXS n

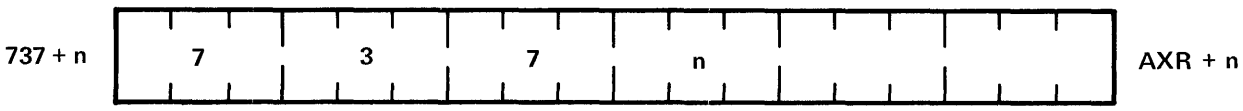
Octal Code: 725000 + n (n=9 bits)

Time: 2 cycles (1 memory cycle)

Operation: n, a signed 9-bit (8 bits plus sign) 2's complement integer is added to the contents of the index register, and the result is placed in the index register. If the sum is greater than or equal to the contents of the limit register, then the program counter is incremented by 1 and thus the next instruction is skipped.

Symbolic: XR + N → XR
If XR $\geq$ LR, PC + 1 → PC

ADD TO INDEX REGISTER



Mnemonic Name: AXR+n

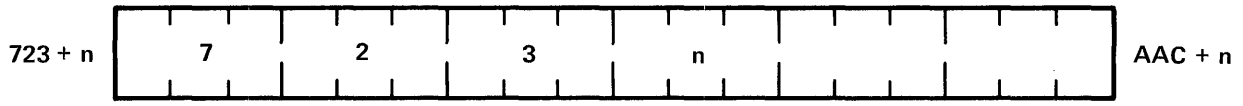
Octal Code: 737000 +n (n=9 bits)

Time: 2 cycles (1 memory cycle)

Operation: n, a signed 9-bit (8 bits plus sign) 2's complement integer is added to the content of the index register, and the result is placed in the index register.

Symbolic: XR + N →XR

ADD n TO ACCUMULATOR



Mnemonic Name: AAC+n

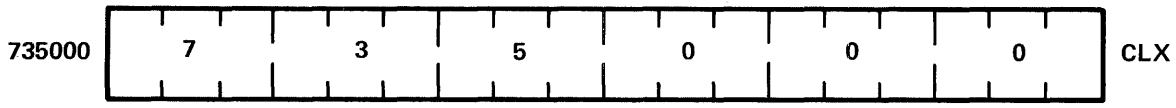
Octal Code: 723000 +n (n=9 bits)

Time: 2 cycles (1 memory cycle)

Operation: n, a signed 9-bit (8 bits plus sign) 2's complement binary number, is added to the content of the accumulator, and the result is placed into the accumulator.

Symbolic: AC + N →AC

CLEAR THE INDEX REGISTER



Mnemonic Name: CLX

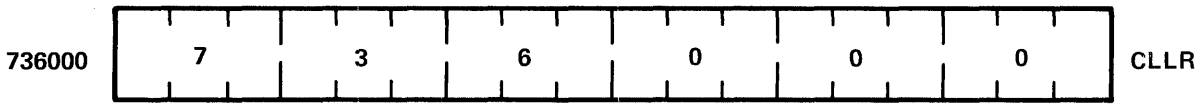
Octal Code: 735000

Time: 2 cycles (1 memory cycle)

Operation: The content of the index register is replaced with all 0s. Former content is lost.

Symbolic: 0 → XR

CLEAR THE LIMIT REGISTER



Mnemonic Name: CLLR

Octal Code: 736000

Time: 2 cycles (1 memory cycle)

Operation: The content of the limit register is replaced with all 0s. The former content is lost.

Symbolic: 0 → LR

Chapter 4
Addressing Features

4.1 INTRODUCTION TO MEMORY ADDRESSING

The PDP-15 Memory Reference instructions are used to operate on data that is stored in memory locations. A memory reference instruction consists of an operation code and an address (see Figure 4-1). The operation code determines how data is to be modified, and the address portion is used to refer to a memory location. The contents of the referenced location are operated on according to the operation code of the memory reference instruction. The address of a specific memory location always remains the same, however, the contents of the location are subject to change depending on the type of memory reference instruction executed.

4.2 TYPES OF ADDRESSING

The PDP-15 enables six types of addressing (refer to Table 4-1). Indexed addressing is allowed only when the PDP-15 is in Page mode.

Table 4-1
Types of Addressing

Address Type	Addressable Memory Locations	
	Page Mode	Bank Mode
Direct	4,096	8,192
Indirect	32,768	32,768
Auto-Increment	131,072	131,072
Indexed	131,072	Not used
Indirect - Indexed	131,072	Not used
Auto-Increment Indexed	131,072	Not used

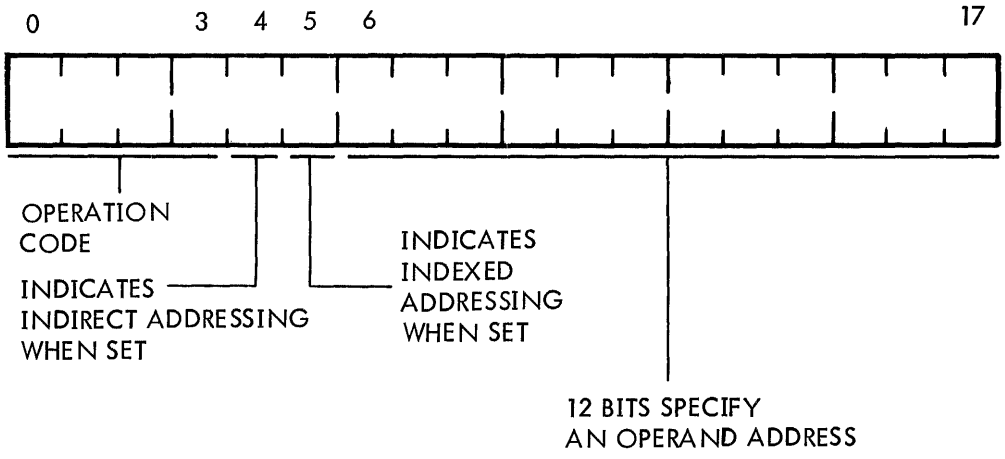
The table indicates the two modes of addressing available, i.e., Page and Bank mode. Either mode can be enabled or disabled through program control, or with a console switch at program start time.

Page mode addressing causes the PDP-15 to use 12 bits (bits 6 through 17) of the memory reference instruction as the operand address. The PDP-15 is, therefore, capable of directly addressing a maximum of 4096 memory locations. Indirect, Indexed, or Auto-increment addressing must be used to reference an address not within the addressing range of the 12 bits.

Bank mode addressing causes the PDP-15 to use 13 bits (bits 5 through 17) of the memory reference instruction as the operand address. The PDP-15 is, therefore, capable of directly addressing a maximum of 8192 memory locations in Bank mode. Indirect or Auto-increment addressing must be used to reference an address not within the addressing range of the 13 bits.

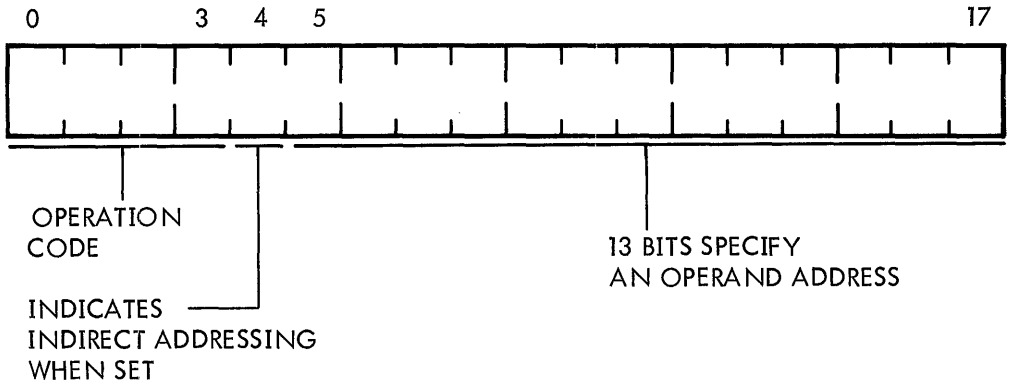
PAGE MODE ADDRESS FORMAT

The memory reference instruction word format for Page mode operation is as follows.



BANK MODE ADDRESS FORMAT

The memory reference instruction word format for Bank mode operation is as follows.



4.3 DESCRIPTION OF THE TYPES OF ADDRESSING

The following discussion is simplified by using mnemonics for the memory reference instructions. The brief programming examples use octal memory addresses. Also, the absolute address of a specific memory location is referred to as the Effective Address (EFA).

The mnemonics for the memory reference instructions are repeated below for easy reference.

CAL	Call subroutine
DAC	Deposit accumulator
JMS	Jump to subroutine
DZM	Deposit zero in memory
LAC	Load accumulator
XOR	Logical Exclusive OR
ADD	Add, 1's complement
TAD	Add, 2's complement
XCT	Execute
ISZ	Increment and skip if zero
AND	Logical AND
SAD	Skip if AC different from memory
JMP	Jump

The program counter (PC) always points to the next instruction to be executed. Thus, in Bank mode the PC is able to increment 20000 (octal) locations from memory address 00000 through 17777 of the 8K bank which contains the program. Refer to Figure 2-9 which illustrates memory organization in terms of 4K pages, 8K banks, and 32K blocks.

4.3.1 Direct Addressing - Bank or Page Mode

The PDP-15 uses bits 5 through 17 in Bank mode, or bits 6 through 17 in Page mode, as the EFA for direct addressing.

The following example illustrates direct addressing.

PC	Instruction	
000100	202222	(LAC 02222)

In the example, the instruction fetched from location 000100 (located in page 0 of bank 0) specifies "load the AC with the contents of location 2222" (located in page 0 of bank 0). Direct addressing is indicated, and the EFA is 002222.

4.3.2 Indirect Addressing - Bank or Page Mode

Indirect addressing is specified when bit 4 of a memory reference instruction is set. Some of the frequent uses of indirect addressing include building or retrieving blocks of data in core memory and referencing memory locations outside of the page or bank containing the program.

The following examples illustrate indirect addressing (* specifies indirect addressing).

PC	Instruction	
000200	221111	(LAC * 01111)

The instruction fetched from location 000200 specifies "load the AC with the contents of the memory location specified by the contents of 01111 (located in page 0 of bank 0)." If location 01111 contained 000300, the EFA would be 000300, and the contents of 000300 would be loaded into the accumulator.

Indirect addressing enables the user to reference memory locations within a 32K memory bank (000000 through 077777 octal). In the above example, memory location 300 can be referenced in any memory page by specifying the page in address bits 3, 4, and 5:

PC	Instruction	
000200	221111	(LAC * 01111)

If location 01111 contains 070300, the contents of location 300 in page 1 of bank 3 are loaded into the accumulator.

4.3.3 Auto-Increment Addressing - Bank or Page Mode

The PDP-15 has eight special registers located in page 0, bank 0, block 0. The eight registers are memory locations 000010 through 000017. Whenever these registers are indirectly referenced by a memory reference instruction, the content of the register is incremented by one before it is used as the operand. The registers are called auto-increment registers. The auto-increment feature is performed only when the register is referenced indirectly. The registers act as any other memory location when referenced directly.

An auto-increment operation can be initiated from any page, bank, or block of memory. The PDP-15 auto-increments whenever a memory reference instruction specifies indirect addressing, and the address points to any location from 000010 through 000017.

NOTE

The auto-increment register is incremented before the content is used as the operand address.

Programming examples:

Auto-increment from page 0, bank 0, block 0, and read the data word stored in location 001000 into the accumulator.

Step 1	Set the auto-increment register to the operand address value -1. In this case, 000777.
Step 2	Reference the auto-increment register with indirect addressing specified.
Step 3	The auto-increment register increments by one, and the EFA is now the new value (001000) in the register.

Steps 1, 2 and 3 are represented in assembly language form as:

```
Step 1          LAC    K777        /Auto-increment register 10
                DAC    10
Steps 2,3       LAC *  10          /C(001000) loaded into AC
                 .
                 .
                 .
          K777  000777             /Constant for initialization
```

When operating from a memory bank other than bank 0, the initial contents of the auto-increment register must be set up using indirect addressing. If direct addressing is used, locations 10 through 17 of the bank containing the program are referenced. For example, if operating in bank 2, Step 1 in the previous example must be modified as:

```
Step 1          LAC   K777
                DAC*  K10       /Deposit in location 10 of page 0,
                                  /bank 0.
Step 2,3        LAC*  10        /C (001000) loaded into AC
                 .
                 .
                 .
          K777  000777          /These constants are located
          K10   000010          /in the same memory bank
                                  /as the main program.
```

4.3.4 Indexed Addressing - Page Mode Only

The PDP-15 central processor has an 18-bit index register (17 bits + sign) and a 17-bit program counter (no sign) with appropriate data path and adder circuitry to compute 17-bit effective addresses.

Indexed addressing can only be used in Page mode operation and is indicated when bit 5 of a memory reference instruction is set. With this type of addressing, the user gains access to 131,072 memory locations (000000 through 377777 octal) without adding an additional memory cycle to compute the effective address (as does indirect addressing).

When indexed addressing is indicated, the effective address is calculated by 2's complement addition of the 18-bit index register to the current block, bank, and page address, and bits 6 through 17 (address bits) of the memory reference instruction. The block, bank, and page address is indicated by the program counter bits 1 through 5.

For example:

$$(XR) + (PC_{1-5}) + (ADDR) = \text{the effective address}$$

Where XR is the index register; PC_{1-5} is program counter bits 1 through 5; ADDR is the address portion (bits 6 through 17) of the memory reference instruction.

The following example illustrates indexed addressing:

```
                 .
                 .
                 .
PC = 003000     LAC   100,X       /(210100 octal)

XR = 000100
```

The instruction fetched from location 3000 specifies "load the AC with the content of the effective address calculated by adding PC_{1-5} (00), the XR (100) and the address field of the instruction (100)." This results in an effective address of 000200, because:

```
  000100     (XR)
+ 000100     (PC1-5) + (ADDR)
  ------
  000200     (EFA)
```

If operating in an extended memory bank or block, indexed addressing can be used to reference other pages, banks, or blocks above or below the operating area.

Programming example:

The program is operating in block 2, bank 0, page 1 and must reference location 1000 in block 3, bank 3, page 1 (location 371000):

```
PC = 213000     LAC    1000, X        /(211000 octal)

XR = 160000
```

The EFA is calculated in the following manner:

```
      160000     (XR)
    + 211000     (PC1-5) + (ADDR)
      ------
EFA=  371000     (block 3, bank3, page 1, location1000)
```

The program is operating in block 2, bank 0, page 1, and must reference location 1000 in block 0, bank 0, page 0 (location 001000).

```
PC = 213000     LAC    1000, X        /(213000 octal)

XR = 570000     (negative value)
```

The EFA is calculated by:

```
      570000     (XR)
    + 211000     (PC1-5) + (ADDR)
      ------
EFA=  001000     (block 0, bank 0, page 0, location 1000)
```

NOTE

The XR contains a negative value. In this case, it is the 2's complement value of the current block, bank, and page.

All 18 bits (17 bits + sign) of the XR are involved when using indexed addressing.

The EFA of the last example can also be calculated in the following manner.

```
PC = 213000     LAC   200, X          /(210200 octal)

XR = 570600

  570600      (XR)
+ 210200      (PC1-5) + (ADDR)
--------
  001000      EFA
```

4.3.5 Indirect Indexed Addressing - Page Mode Only

Indirect indexed addressing is implemented only in Page mode and is indicated when both the indirect addressing indicator (bit 4) and the indexed address indicator (bit 5) of the memory reference instruction are set.

When indirect indexed addressing is indicated, the PDP-15 central processor first calculates the address indirectly referenced. This calculation is done in exactly the same manner as described in the Indirect Addressing section. The PDP-15 carries the address calculation one step further, when indirect indexing is specified. The contents of the index register are added to bits 3 through 17 of the data word which was retrieved indirectly.

The addition (2's complement) of the XR is always the last step to occur (post indexing).

The following example illustrates indirect indexed addressing:

```
PC = 203000     DAC * 100, X          /(070100 octal)

XR = 000100

location 200100 = 007000
```

The EFA is calculated as:

```
  007000      (Contents of location 200100)
+ 2           (PC1-2)
--------
  207000
+ 000100      (XR)
--------
  207100      EFA
```

The instruction fetched from memory location 203000 specifies "deposit the contents of the accumulator into the memory location calculated by retrieving the contents of memory location 200100 (location 100 of the current block, bank, and page) and add the contents of the index register to the contents of memory location 200100."

The indirect address pointer is calculated by appending bits 1 through 5 of the PC to the 12-bit address (6 through 17) of the instruction word resulting in a 200100 address pointer. An additional memory cycle is required to retrieve the contents (007000) of memory location 200100. Bits 1 and 2 of the PC (current block) are then appended resulting in 207000. The contents of the XR are then added, resulting in a final EFA of 207100.

4.3.6 Auto-Increment Indexed Addressing - Page Mode Only

Auto-increment indexed addressing can be implemented only when operating in Page mode. This type of addressing is specified when the indirect address indicator (bit 4) and the indexed address indicator (bit 5) of the memory reference instruction are both set and address bits 6 through 17 equal a value of 10 through 17 octal.

When this type of addressing is specified, the PDP-15 central processor performs the following steps to calculate the EFA:

Step	Procedure
1	The contents of the auto-increment register are retrieved, incremented by one, and then restored in the register.
2	The new contents of the auto-increment register are used as an address pointer.
3	The contents of the XR bits 0 through 17 are added (2's complement) to the address pointer.
4	The sum is used as the final, or effective address (EFA).

The following example illustrates this type of addressing:

```
PC = 170245      DAC *  15, X          /(070015 octal)
000015 = 000777
XR = 001000
```

Following the procedure given above, the EFA is calculated as:

000777	contents of location 15	Step 1
+1	increment	
001000	new contents of location 15 used as address pointer	
+001000	contents of XR	Step 3
002000	EFA	Step 4

4.4 SPECIAL ADDRESSING CASES

Certain instructions in the PDP-15 cross bank, page, and block boundaries in a special case, and other instructions have limitations when crossing blocks.

JMS and JMP * Instructions

The JMS instruction saves a 15-bit address at the subroutine entry point. This means that JMS,X across block boundaries saves the return location address within a block, but not the block number. A JMP * exit from a subroutine entered from a different block is not possible. A computed JMP,X must be used.

CAL Instruction

The CAL instruction always falls to location 20 of block 0, bank 0, page 0, regardless of which block, bank, or page it is issued from. This instruction has the same limitations as the JMS instruction.

Auto-Increment Instructions

All auto-increment instructions (indirect memory references to locations 10-17 of any page, bank, or block) increment the auto-increment registers in locations 10-17 of page 0, bank 0, block 0. The content of location 10-17 can point to any page, bank, or block.

All 18 bits are used as address bits. If the 18-bit address in the auto-increment register points to a nonexistent memory location and the memory protect option is installed, a nonexistent memory (NEXM) error occurs. If the memory protect option is not available, the machine will hang, waiting for memory to respond to the request.

Skips

If a skip instruction (OPERATE, ISZ, SAD, or IOT) is located in the next to last memory location of a bank, in Bank mode, or a page, in Page mode, and a skip is affected, the program will not wrap around within the bank or page. Instead, the PC will be incremented over the bank or page boundary to the first location of the next bank or page of memory.

4.5 PROCESSOR ADDRESSING

Program Interrupts

Program interrupts always go to location 0, page 0, of block 0 and save a 15-bit address (most-significant address bits truncated) in the same fashion as CAL.

Automatic Priority Interrupts

Automatic priority interrupts can be directed through the I/O device to any location within block 0 using 15 bits of I/O address lines. The JMS or JMS * in the vector location is always to a location within block 0 and the JMS,X is relative to block 0.

Three-Cycle Data Transfers

Three-cycle data transfers can transfer data anywhere within memory because the word count and current address registers are 18 bits in length, however, the WC,CA register pair must be located within block 0 and are specified by a 15-bit I/O address from the I/O device.

Single-Cycle Data Transfers

Single-cycle data transfers can be made to all of memory. The I/O device must specify a 15-bit address on the I/O address lines and a 2-bit block number on the Program Interrupt Request and Skip Request lines. These lines are dual purpose lines which act as the most significant address bits during single-cycle transfers, only the increment Skip Request acts as address bit 1 and the Program Interrupt Request as bit 2, bits 3-17 are represented by the I/O address lines. All single-cycle devices should have provisions for generating 17-bit addresses.

Chapter 5
I/O Processor System

5.1 GENERAL DESCRIPTION

The I/O processor is the communication link between a diverse line of peripherals and the PDP-15 main memory and CPU. The I/O processor system provides a number of facilities for data transmission. These I/O facilities enable the I/O processor system to provide data transfers to/from memory, data transfers to/from the CPU, command status transfers, and interrupts. The architecture design of the I/O processor provides a number of benefits for the user:

a. Special purpose equipment can be easily and inexpensively interfaced to the system.

b. Synchronous and asynchronous devices can be handled with equal ease.

c. Real-time applications are easily implemented because of the speed and efficiency of the I/O processor.

The capabilities of the I/O facilities are described in Table 5-1.

Table 5-1
I/O Capabilities

Facility	Remarks
Data Transfers To/From Memory	
Multicycle Data Channel Input	Used to transfer data to core memory in up to 18-bit bytes at high speed (250 kHz).
Multicycle Data Channel Output	Used to transfer data directly from memory in up to 18-bit bytes. Maximum speed is 188 kHz.
Add-to-Memory	Used to add the contents of a device register to the contents of a specified core location in 18-bit bytes. Good for signal averaging. Maximum speed is 188 kHz.
Increment Memory	This facility allows an external device to increment the content of a core location by 1. Useful for generating histograms. Maximum speed is 333 kHz.

Table 5-1 (Cont)
I/O Capabilities

Facility	Remarks
	Data Transfers To/From Memory (Cont)
Single-Cycle Data Channel Output	With this DMA facility a device can transfer a burst of data from core memory at 1 mHz in 18-bit bytes.
Single-Cycle Data Channel Input	Used to transfer a burst of data from a device to core memory at 1 mHz per 18-bit word.
	Data Transfers To/From CPU
Addressable I/O Bus	With this facility, up to 40 devices can transfer data in 18-bit bytes to or from the central processor. Cost of interfacing is minimal. A typical transfer is one transfer every 200 μs.
	Command and Status Transfers
Addressable I/O Bus	Command and status information can be transferred to or from the CPU in the same manner as ordinary data.
Read Status	This is a special facility designed to aid the user in monitoring vital flags in the system. Each device is assigned a bit for its flag(s), which is read onto the addressable I/O bus into the CPU when the Read Status command is given. No two devices should use the same bit.
Skip	The addressable I/O bus allows the computer to test the status of a flag (typically) by issuing a pulse which will echo if the addressed flag is up. Every flag that posts a program interrupt should be identifiable by the skip facility.
	Interrupts
Program Interrupt	All devices share a common program interrupt line. When a device posts an interrupt the computer is forced to JMS to location 0, bank 0, and then on to a service routine designed to identify the requesting device using the skip facility. The process requires CPU and memory overhead and takes time.
Automatic Priority Interrupt	The automatic priority interrupt (API) facility provides priority servicing of many I/O devices with minimum programming and maximum efficiency. Its priority structure permits high data rate devices to interrupt the service routines of slower devices, with a minimum of system overhead.

5.2 I/O PROCESSOR PRIORITY STRUCTURE

All I/O related transfers function within the precedence of the following priority structure:

a. Data Channel (DCH) requests (highest priority)
b. Real-Time Clock (RTC) (optional)
c. Automatic Priority Interrupts (API), 8 Levels (optional)
d. Program Interrupts (PI)
e. Main Program in Progress (lowest priority)

The data channel requests are the highest priority and will be serviced first, even if all other requests are raised simultaneously. If a lower priority request is being serviced and a DCH request is generated, the DCH request must wait until the end of the current I/O activity execution to be serviced.

5.3 THE DATA CHANNEL CONTROLLER

The data channel controller provides the system user with one channel for high-speed data transmission. This channel has the capacity for eight I/O devices, which are chained linked. Devices using either single-cycle block transfer or multicycle block transfer can easily be intermixed on this data channel in any configuration desired.

5.4 MULTICYCLE CHANNEL BLOCK TRANSFER

The data channel controller supervises the multicycle channel block transfer function. When the multicycle block transfer has been initiated, the data transfer becomes completely automatic and requires no access to the CPU. The CPU is free to do computation while the data channel is active. The only limitation on simultaneity lies in sharing the main memory. Because the I/O processor has first priority on memory requests, the CPU is effectively locked out for three cycles. As data channel block transfers approach the maximum rate, (back-to-back breaks), the CPU can be completely locked out.

To transfer data using the multicycle block transfer mode, the user must initialize two sequential core locations. The contents of these core locations contain the word count and current address.

The word count represents the number of words to be transferred in the block. The current address represents the location to which the data is to be transferred. The I/O processor contains the control logic and I/O adder to automatically fetch the contents of these locations and increment the contents of each.

The multicycle channel block transfer is a three-cycle sequence. Data is written into and read from memory in three I/O processor cycles. The output cycle occurs during the third I/O processor cycle. The I/O processor is stopped to allow settling of the I/O bus and control gates, prior to strobing the data word into the device buffer register.

The multicycle block transfer is flowcharted in Figures 5-1 and 5-2. The data transfer is initiated by an input/output instruction to the device after the two core locations have been initialized to minus the word count and current address minus one. During the first cycle, the contents of the word count location are incremented by one and restored. During the second cycle, the current address is incremented by one and restored. The I/O processor continues to transfer data sequentially until the word count register reaches zero, at which time an interrupt is generated to notify the monitor that the block transfer is complete.

Assuming initialization of the two core locations has taken place, the data transfer from device to memory (see Figure 5-2) occurs as follows:

a. An instruction from the service routine enables the device controller. This allows the device controller to request a data transfer from the I/O processor.

b. When the device controller's data buffer registers are full, the device issues a "data channel request."

c. The I/O processor acknowledges the request by returning a "data channel grant."

d. The device controller then generates a fixed code pointing to the initialized word count core memory location. This fixed address is transmitted over the common I/O bus address lines and is stored in the data storage register of the data channel controller.

The I/O processor then generates a "memory cycle request." The address data, in the data storage register, is then stored in the memory address register of the memory bank; the data (word count), from the first word of the two locations that the MA is now pointing to, is transmitted out of memory and into the data channel controller's adder. The word count data is incremented by one and stored back in memory.

When the word count data register overflows, a signal (Word Count Overflow) is sent to the device to terminate all future block transfers. The monitor is also notified that this condition exists.

During the second I/O processor cycle, the fixed code from the device controller is gated through the I/O adder and is incremented by one. The memory address register then receives this address. The contents of this address are read out of memory, incremented by one, then restored in, memory. The new contents, known as the current address, will be the location where the next data transfer occurs.

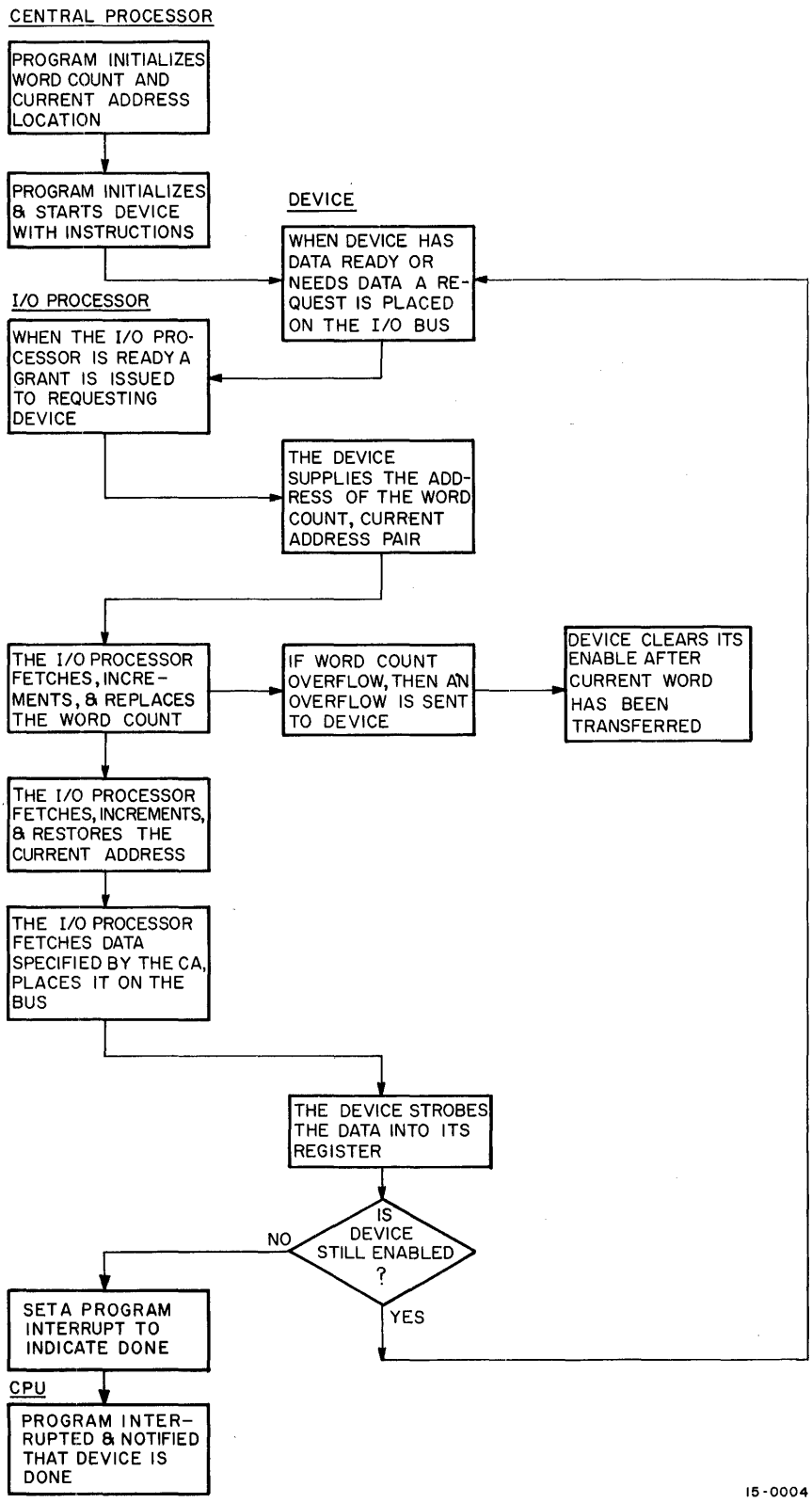


Figure 5-1 Multicycle Out Block Transfer, Flowchart

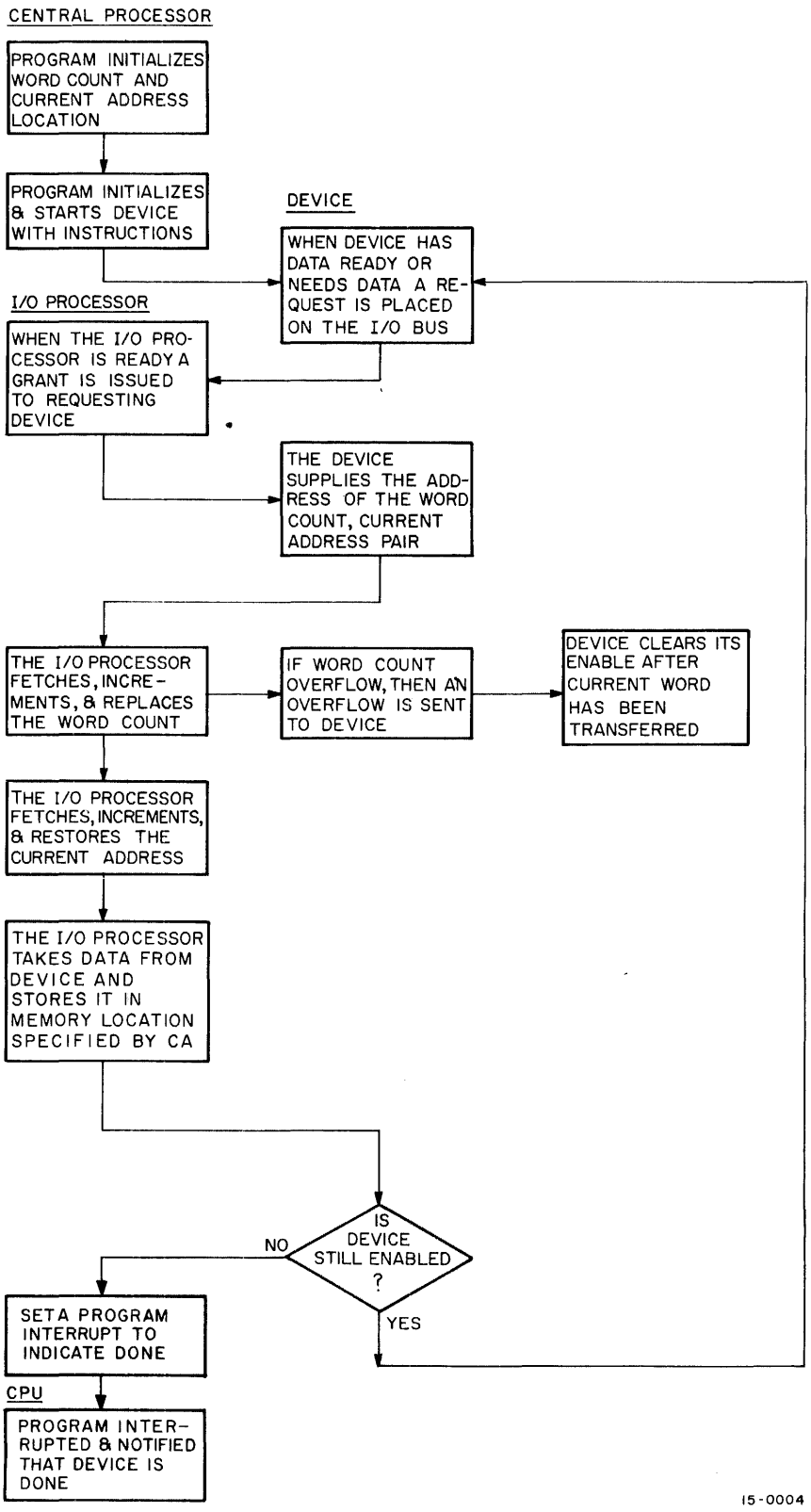


Figure 5-2 Multicycle In Block Transfer, Flowchart

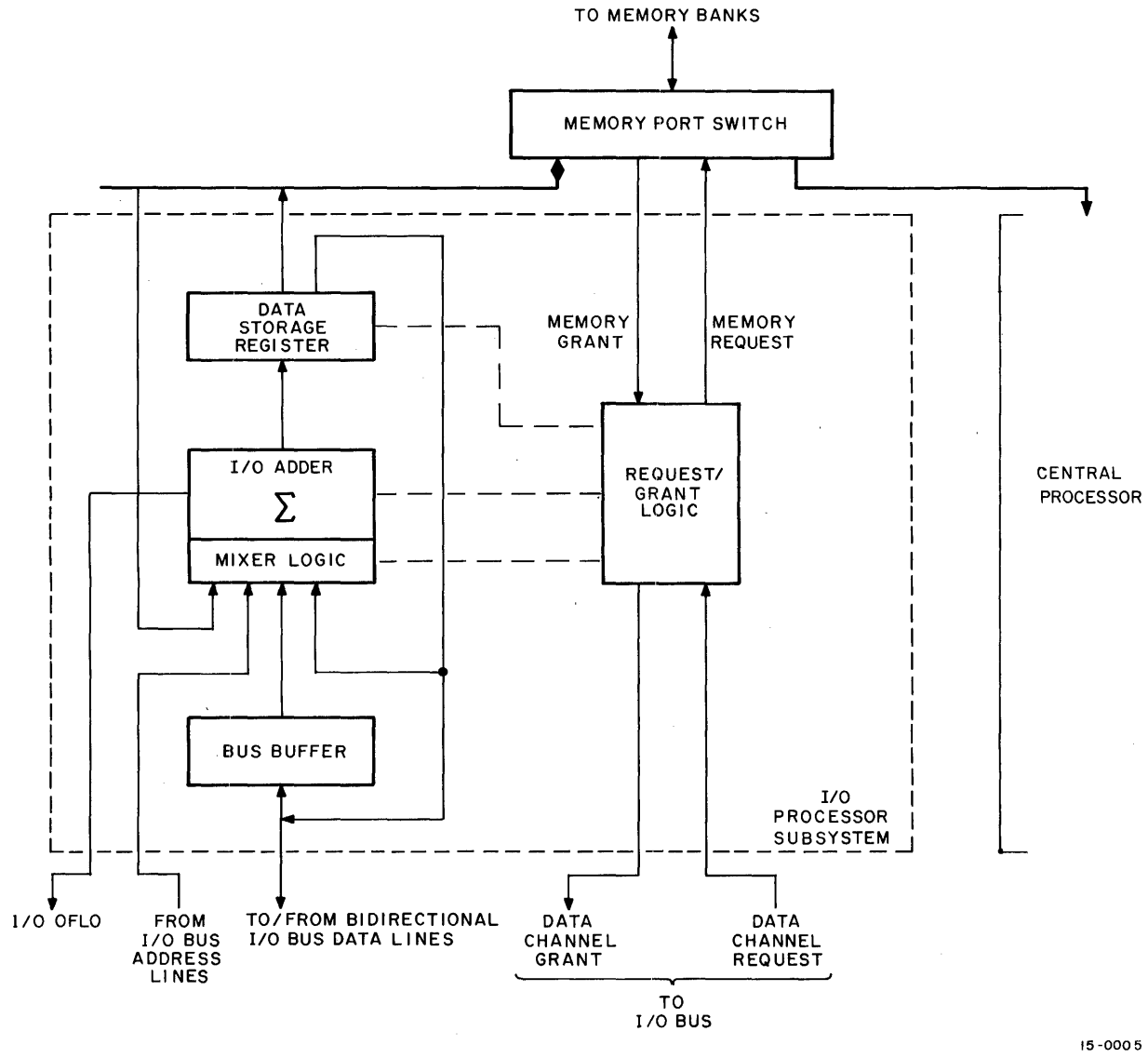


Figure 5-3 Multicycle Transfer Implementation

During the third I/O processor cycle, the current address is read into the memory address register. This address then points to the location where the I/O data word will be transferred.

A memory request/grant synchronization again occurs, and the data in the storage register is strobed into the memory location ending the cycle. Data output follows the same sequence, with the exception that one additional I/O processor cycle is required in order for the I/O bus to have sufficient time to settle down before data from the bus is strobed into the device register.

5.5 SINGLE-CYCLE BLOCK TRANSFERS

Single-cycle block transfers (see Figure 5-4), are used by high-speed peripherals that normally transfer complete records (blocks) of information, such as disks. A single cycle of the I/O processor takes 1 μs, allowing a maximum transfer rate.

Hardware registers, designed into the device controllers of the high-speed peripherals, store the "current address" (the memory location where data is currently being transferred), and the "word count" (the number of words remaining to be transferred in a block). These registers are loaded by input/output transfer (IOT instructions issued by the CPU).

Device testing and initialization are handled by the CPU via IOTs to provide supervisory control. A subsequent IOT initiates the data transfer. The I/O processor uses the current address information to address core memory, then strobes the data between memory and the device controller buffer register. Logic within the device controller then increments the current address register and the word count register to provide sequential block transfer.

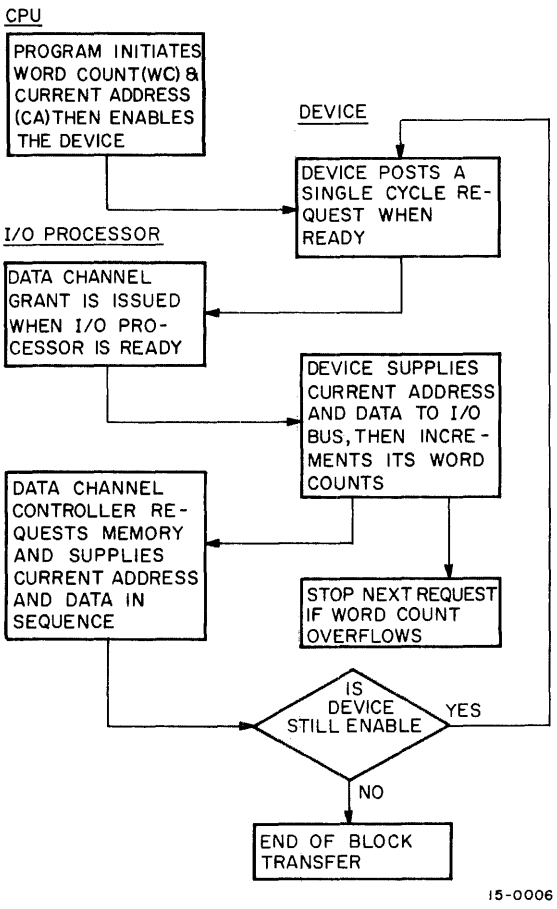


Figure 5-4 Single-Cycle Block Transfer Flowchart

When the word count register overflows at the end of a block transfer, an interrupt is generated to allow the monitor system to take further action. This action includes disconnecting the device from the I/O bus, or reloading the device controller registers for another block transfer. The maximum number of transferrable words in a single block is 131,072.

Figure 5-4 illustrates the method the data controller uses to handle a single-cycle transfer. Assuming that the program has initiated the word count and address of the device controller and has then enabled it, the following occurs:

a. The device controller sets a single-cycle data channel request to the I/O processor.

b. The I/O processor, as soon as it becomes available, acknowledges the request by returning a "data channel grant." The device then

b. (Cont) strobes both its current address and its data onto the I/O bus and to the processor. The data channel controller feeds the current address through its adder to the data storage register. A memory cycle is requested, and this address is strobed into the memory address buffers. The data is then strobed off the 18 I/O data lines and into the memory location specified by the current address. During this operation, the device increments its own word count and disables itself on overflow. It then sends an interrupt to the monitor to indicate that its operation has been completed.

5.6 INCREMENT MEMORY

The increment memory mode enables an external device to add to the contents of any memory location in a single-cycle operation. The device controller supplies the core address and the I/O processor simply goes through the word-count cycle of a multicycle channel transfer. This effectively adds 1 to the specified location. This feature is particularly useful for in-core scaling and counting in pulse-height analysis.

5.7 ADD-TO-MEMORY

Add-to-memory is a standard feature of the PDP-15 that adds unique capabilities to the already powerful I/O facilities. In add-to-memory mode, the contents of an external register can be added to the contents of a memory location in four cycles. This feature is extremely valuable in signal averaging and other processes requiring successive sweeps for signal enhancement.

The add-to-memory operation is a combination of multicycle data channel input and output operations. The data transmitted by the device is added to a word read out of memory as specified by the current address, and the result is rewritten into the same location. It is simultaneously transmitted to the device via the I/O bus.

5.8 PROGRAM-CONTROLLED TRANSFER

Program-controlled transfers, implemented by input/output transfer (IOT) instructions, can move up to 18 bits of data between a selected device and the accumulator (AC) in the CPU. The devices involved are connected to the addressable I/O bus portion of the I/O processor. A total of up to 42 device controllers can be attached to this bus. IOT instructions are microcoded to effect response only for a particular device. The microcoding includes the issuing of both a unique device selection code and the appropriate processor-generated input/output pulses to initiate a specific operation. For an "out" transfer, the program reads a data word from memory into the AC. A subsequent IOT instruction places the data on the bus, selects the device, and transfers the data to the device. For an "in" transfer, the process is reversed: an IOT instruction selects the device and transfers data into the AC. A subsequent instruction in the program transfers the word from the AC to memory.

As previously mentioned, IOT instructions are also used to initialize the single- and multicycle channels and the transfer word count and current address information to the single-cycle controllers. In addition, these instructions are used to test or clear device flags, select modes of device operation, and control a number of processor operations.

The IOT instruction consists of the IOT fetch from core memory and three sequential cycles. The IOP1 and IOP2 are 1 μs intervals; the IOP4 is 500 μs interval. The positive assertion of the IOT signal is variable from 0 to 1 μs. This variation is caused by I/O processor synchronizing. Refer to Chapter 3 for IOT instruction formats.

The total time required to fetch and execute an IOT instruction is a maximum of 5.02 μs and a minimum of 3.96 μs. The I/O processor allows the generation of an IOP1 only or an IOP1 and IOP2 only as required. When an out transfer is used generating the IOP4, the full sequence of IOP1 and IOP2 are also generated. Refer to Table 5-2 for total execution times of IOPs.

Table 5-2
Total Execution Times for IOPs

Issue	Instruction	Total Time for IOT Fetch and Execute	
		Min	Max
IOP1 (only)	SKIP	2.21 μs	3.27 μs
IOP2 (only)	In Transfer	3.21 μs	4.27 μs
IOP1 and IOP2	In Transfer	3.21 μs	4.27 μs
IOP4	Out Transfer	3.96 μs	5.02 μs

The IOP1 is normally used in an I/O SKIP instruction to test a device flag. The IOP1 can be used as a command pulse, but cannot be used to initiate a "read from" a device. Because the CPU accumulator register is used for both data "in" and "out" transfers, a "clear AC" microinstruction (Bit 14) can be used during the 1 μs interval of the IOP1.

The IOP2 is usually used to transfer data to and from the device to the computer, or to clear the device's information register. It cannot be used to determine a "skip" condition.

The IOP4 is normally used to effect programmed transfers of information from the AC to a selected device.

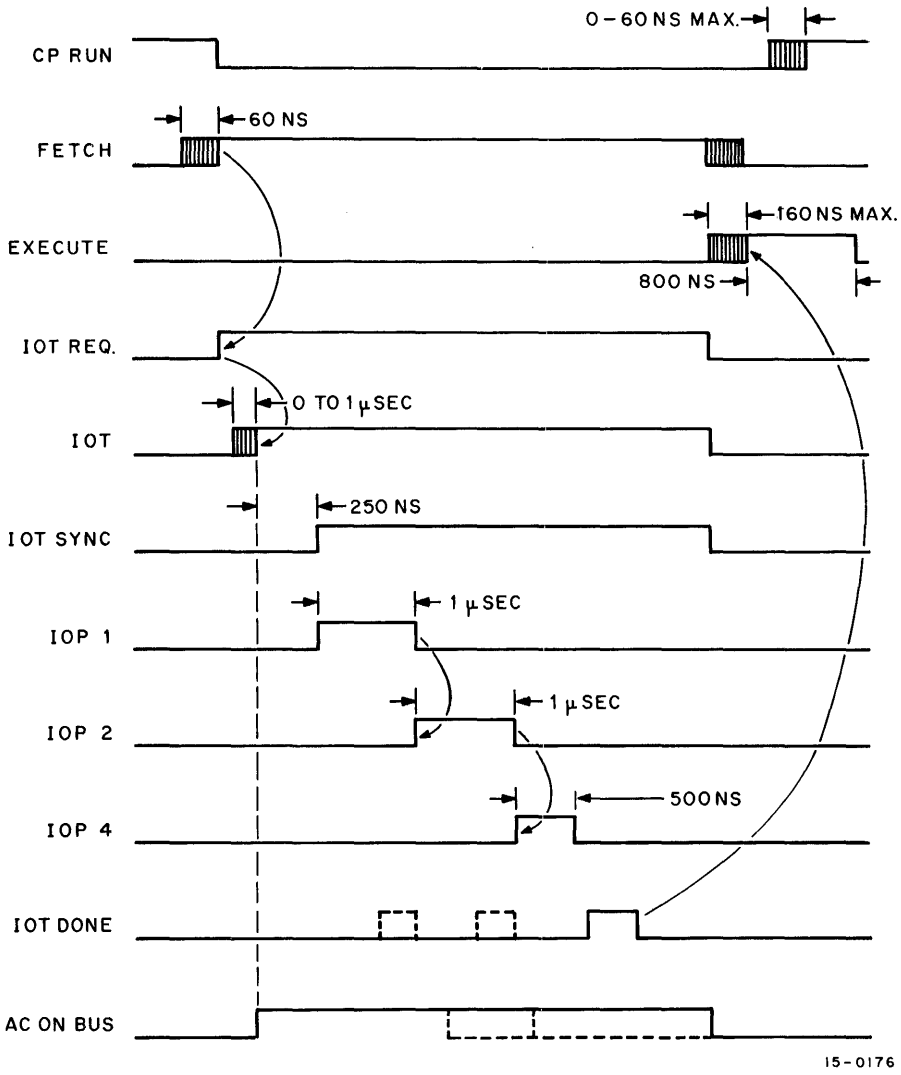


Figure 5-5 IOT Instruction Timing

5.9 PROGRAM INTERRUPT FACILITY

The program interrupt (PI) system is standard on all PDP-15 Systems. The program interrupt (PI) facility, when enabled, relieves the main program of the need for repeated flag searching by allowing the ready status of I/O device flags, to automatically cause a program interrupt. The CPU can continue with execution of a program until a previously selected device signals that it is ready to transfer data. At that time, the program in process is interrupted and the contents of the program counter (15 bits), user mode (1 bit), link (1 bit), and bank mode are stored in location zero. The instruction, in location 000001, is then executed, transferring control to an I/O service routine. When completed, the routine restores the system to the status prior to the interrupt, enabling the

interrupted program segment to continue. Where multiple peripherals are connected to the PI line, a search routine containing device - status testing (skipping) instructions must be added to determine which device initiated the interrupt request. The program interrupt (PI) control is enabled or disabled by IOT instructions. When disabled, the PI ignores all service requests, but each request remains on line and is answered when the PI is enabled. The program interrupt is automatically disabled when an interrupt is granted or when the I/O Reset Key (on the console) is depressed. The program interrupt is temporarily inhibited while the automatic priority interrupt system is processing a priority interrupt request. The PIE indicator (on the console) is lighted while the PI is enabled.

A free instruction follows the program interrupt and therefore, the instruction in location 1 will always be executed immediately after the program interrupt.

Chapter 6
Options

6.1 KE15 EXTENDED ARITHMETIC ELEMENT

The extended arithmetic element (EAE) option adds the hardware necessary to implement the EAE instructions. This class of instructions, identified by an operation code of 64_8, performs high-speed data manipulation and multiply-divide operations as specified by microprogramming of individual instructions. Figures 6-1 through 6-5 illustrate the microinstruction capabilities for register setup, data shift, normalize, multiply, and divide.

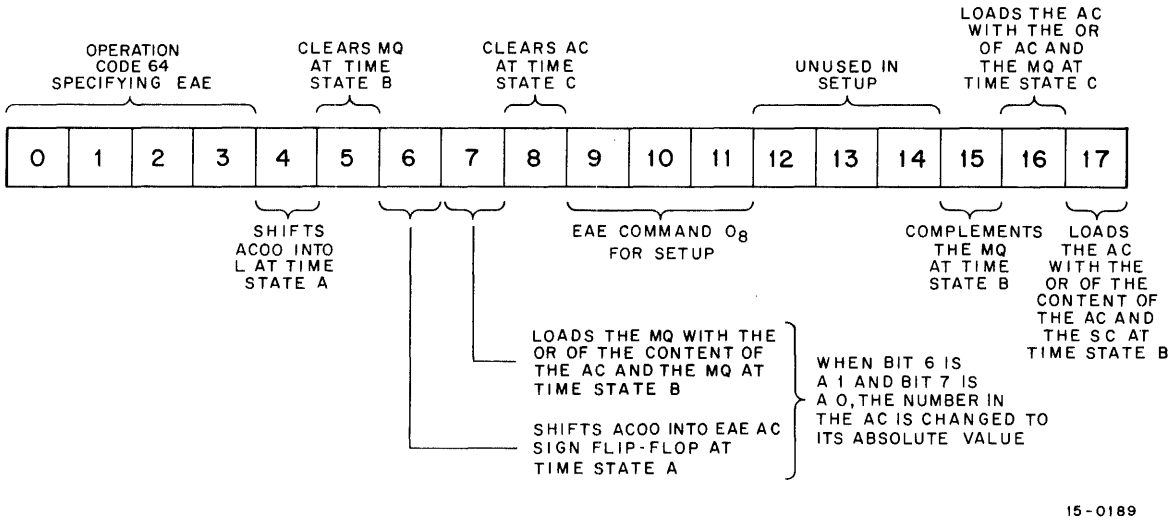
The time required to execute an EAE instruction is a function of the operation and/or the shift, or step count specified by programming. In general, the following considerations apply to the different types of EAE operations.

1. All set-up instructions require 1.324 μs.
2. Long register shift instructions require a time equal to 2.915 μs plus 0.133 μs per "n-1" bit-position shifts. This count is specified by the addition of n(octal) to the instruction code. For example, the input of the symbolic instruction LLS+14 to the PDP-15 assembler would result in an instruction code that specified a long left shift of the AC and MQ (taken as a 36-bit register) 12_{10} bit positions to the left. This instruction would require 4.378 μs.
3. The ASL and ALSS instructions, respectively, AC left shift and AC left shift signed, also require the specification of "n."
4. The normalizing instructions, NORM and NORMS, require an execution time equal to 2.9 μs plus 0.133 μs per number of bit positions shifted to normalize ($AC_0 \neq AC_1$) quantity. These instructions are microprogrammed to set the 6-bit step count to 44_8 (36_{10}). Hence, $-44+n_8$ (the step count is entered in 2's complement notation at execution) equals the biased scale factor of a normalized quantity.
5. Multiply instructions require a time equal to 2.915 μs plus 0.265 μs per -1"n" bit position shifts. Multiply instructions are microprogrammed to set the step count to 22_8 (18_{10}), representing the multiplication of one 18-bit quantity (sign bit and 17 magnitude bits for signed quantities) by another to produce a 36-bit product. The execution time is 7.420 μs. Where such precision is not required, the microprogrammed step count can be decreased by subtracting the appropriate number "n" (octal) from the instruction code. The product is always left justified in the AC, MQ. If "-n" is appended to a multiply instruction, the "n" low-order bits in the long register are meaningless.

6. Divide instructions require a time equal to 2.915 μs plus 0.265 μs per "n" bit position shifts. Divide instructions are microprogrammed to set count to 23_8 (19_{10}), representing division of a 36-bit dividend (actual or implied) by an 18-bit divisor. The execution time is 7.685 μs. Where such precision is not required, the microprogrammed step count can be decreased by subtracting the appropriate number "n" (octal from the instruction code). For example, the symbolic instruction DIV-12 would result in a right-justified quotient with the most significant bit in MQ_9. The execution time is decreased in correspondence to the decrease in the step count.

6.1.1 EAE Microinstructions

Figure 6-6 and Tables 6-1 and 6-2 describe the EAE instructions and illustrate the microinstructions of the EAE instructions. If an existing instruction is not satisfactory, the programmer can combine the appropriate microinstructions to achieve the required result.



Note: Setup Instructions cannot be microprogrammed with Normalize Multiplication or Division Instructions.

Figure 6-1 EAE Setup Microinstructions

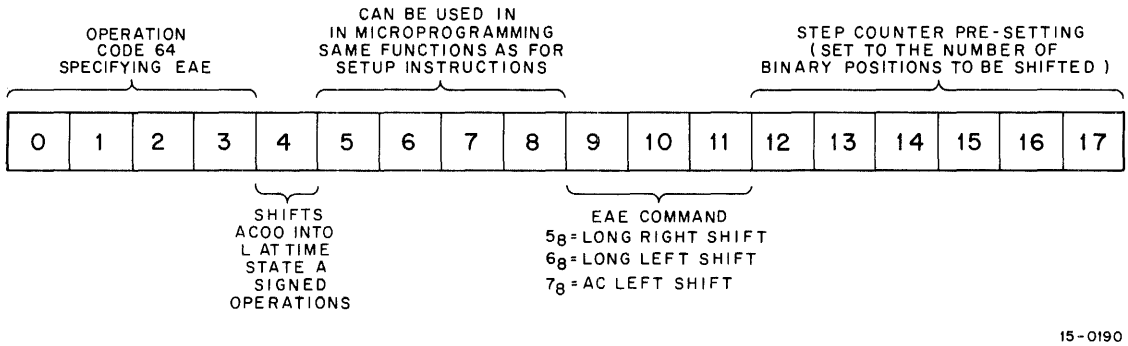


Figure 6-2 EAE Shift Microinstructions

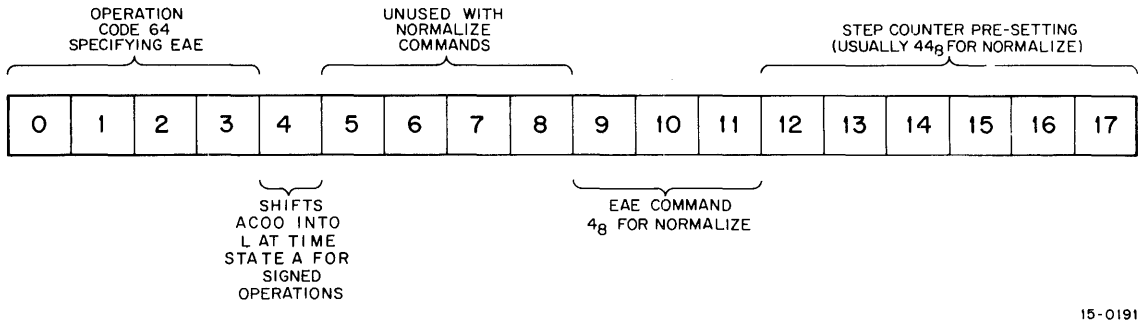


Figure 6-3 EAE Normalize Microinstructions

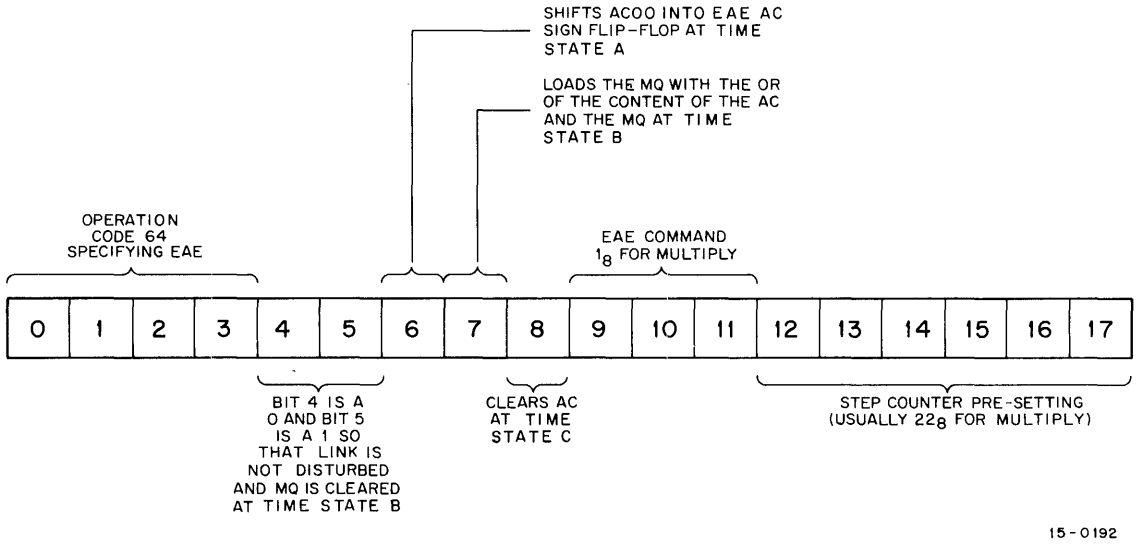


Figure 6-4 EAE Multiplication Microinstructions

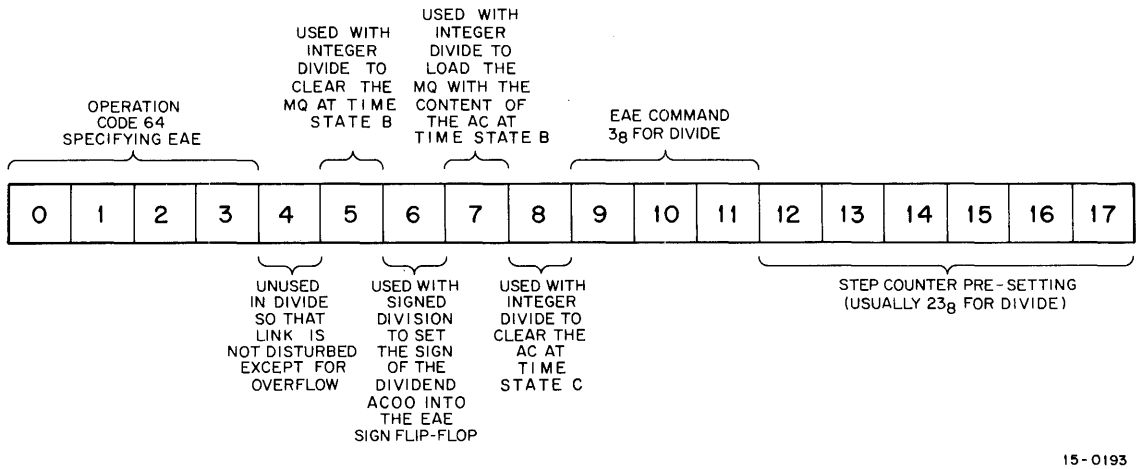


Figure 6-5 EAE Division Microinstructions

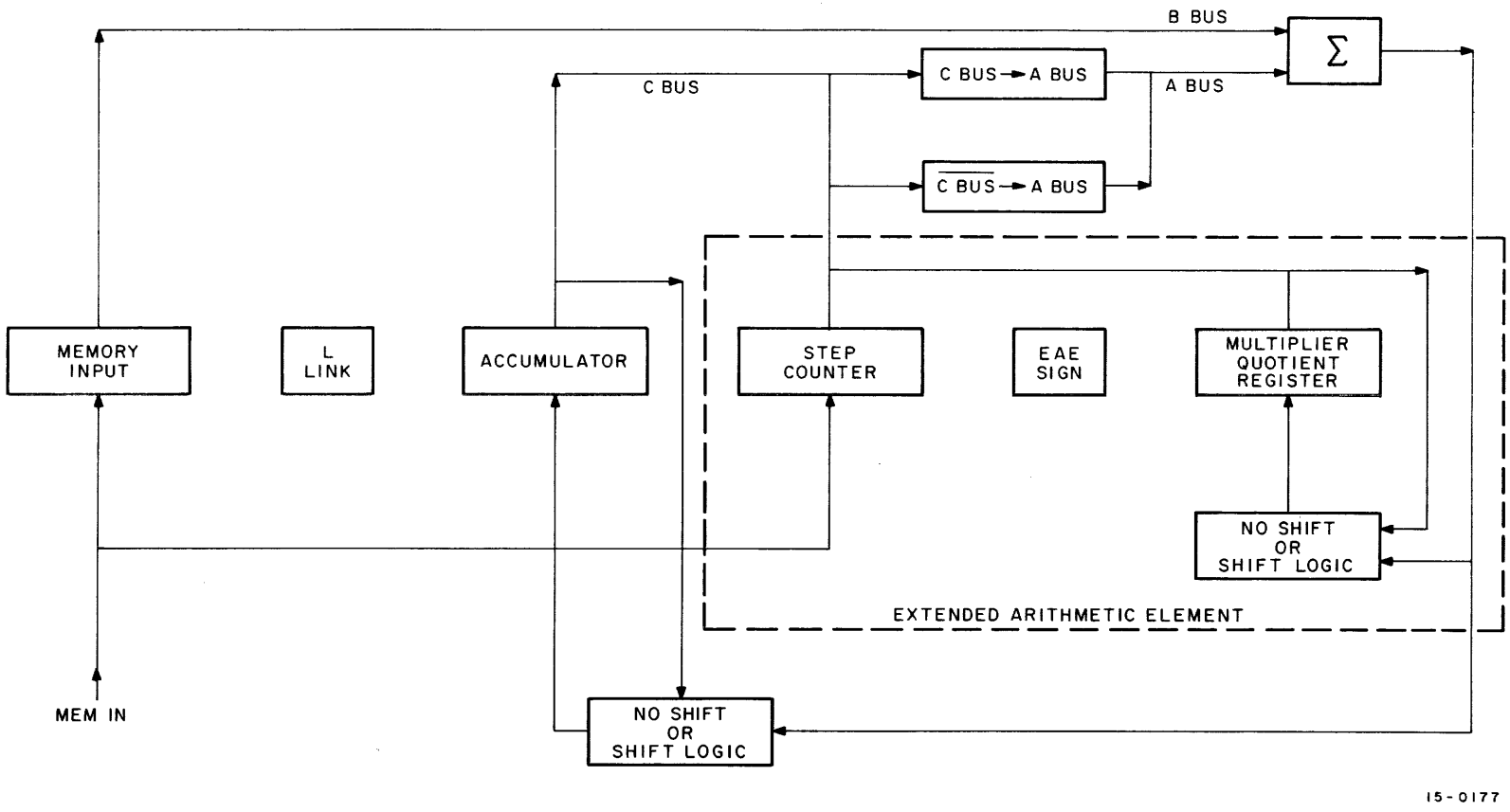
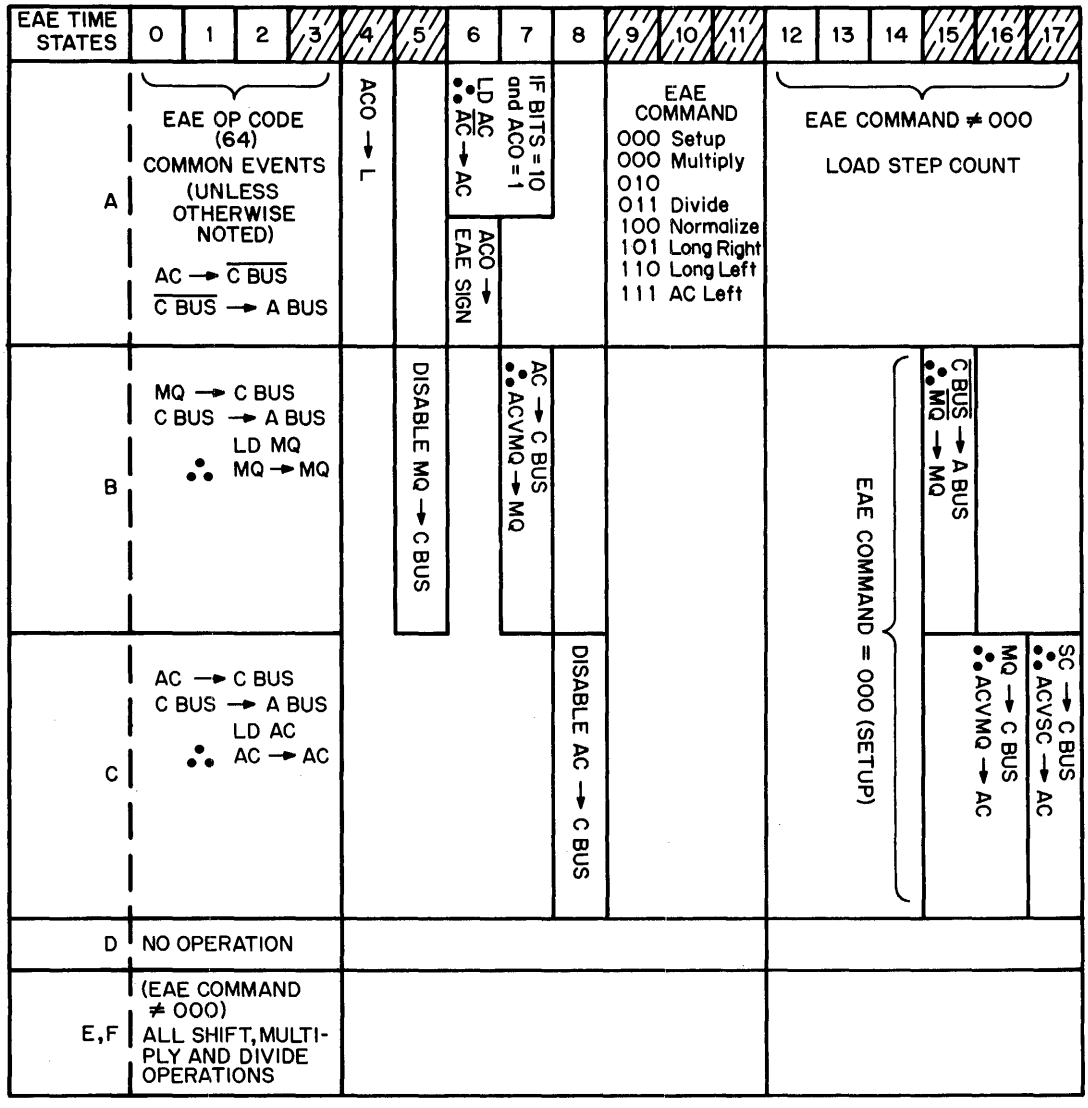


Figure 6-6 EAE Simplified Block Diagram

Table 6-1
EAE Microinstructions



15-0422

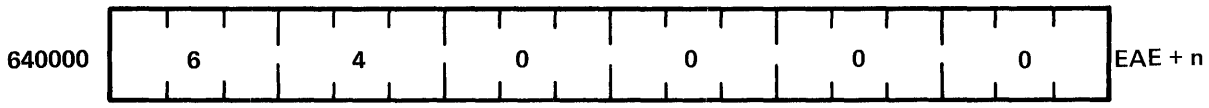
Table 6-2
EAE Microinstructions

Bit Positions	Binary Code	Function
4	1	Enter the content of AC_0 in the link for signed operations.
5	1	Clear the MQ.
6	1	Read the content of AC_0 into the EAE AC sign register prior to-carrying out a signed multiply and divide operation.
6,7	10	Take the absolute value of the AC. Takes place after the content of AC_0 is read into the EAE AC sign register.
7	1	Inclusive OR the AC with the MQ and read into MQ.
8	1	Clear the AC.
9,10,11	000	Setup. Accompanies code in bits 15, 16, and 17.
9,10,11	001	Multiply. Causes the number in the MQ to be multiplied by the number in the memory location following this instruction. If the EAE AC sign register is 1, the MQ is complemented prior to multiplication. The exclusive OR of the EAE AC sign and the link is entered in the EAE sign register. The product is in the AC and MQ, with the lowest order bit in MQ bit 17. At completion, the link is cleared and if the EAE sign is a 1, the AC and MQ are complemented.
9,10,11	010	Unused operation code.
9,10,11	011	Divide. Causes the 36-bit number in the AC and MQ to be divided by the 18-bit number in the memory register following the instruction. If the EAE AC sign is 1, the MQ is complemented prior to starting the division. The exclusive OR of AC_0 and the link is placed in the EAE sign register. The AC portion of the dividend must be less than the divisor or divide overflow occurs. In such cases, the link is set and divide does not occur. Otherwise, the link is cleared. At completion of this instruction, if the EAE sign was a 1, the MQ is complemented. Thus, the remainder has the sign of the dividend.
9,10,11	101	Long right shift. Causes the AC and MQ to be shifted right together as a 36-bit register the number of times specified in the instruction. On each step, the link fills AC bit 0, AC bit 17 fills MQ bit 0, and MQ bit 17 is lost. The link remains unchanged.
9,10,11	110	Long left shift. Causes the AC and MQ to be shifted left together the number of times specified in the instruction. On each step, MQ bit 17 is filled by the link; the link remains unchanged. MQ bit 0 fills AC bit 17, and AC bit 0 is lost.

Table 6-2 (Cont)
EAE Microinstructions

Bit Positions	Binary Code	Function
9,10,11	100	Normalize. Causes the AC and MQ to be shifted left together until the step count is equaled or AC bit 0 ≠ AC bit 1. MQ bit 17 is filled by the link; the link is not changed. The step count of this instruction is normally 44 (octal). When the step counter is read into the AC, it contains the number of shifts minus the initial shift count as a 2's complement 6-bit number.
9,10,11	111	Accumulator left shift. Causes the AC to be shifted left the number of times specified in the shift count. AC bit 17 is filled by the link, but the link is unchanged.
12-17		Specify the step count for all EAE commands (9-11) except the setup command.
15	1	The setup command only, causes the MQ to be complemented.
16	1	The setup command only, causes the MQ to be inclusively ORed with the AC and the result placed in AC.
17	1	The setup command only, causes the AC to be inclusively ORed with the SC and the results placed in AC bits 12-17.

BASIC EAE INSTRUCTION



Mnemonic Name: EAE+n

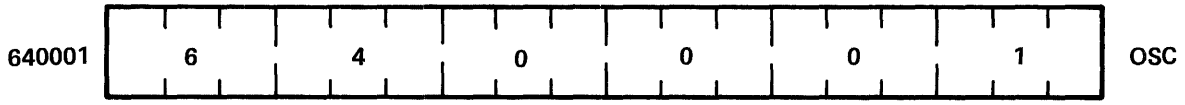
Octal Code: 640000

Time: Depends on instruction

Operation: The addition of "n" (octal) to the mnemonic converts the basic instruction into a microcoded instruction to accomplish a setup, shift, or arithmetic operation not already in the instruction repertoire. Refer to Table 6-1 for descriptions of the functional use of the individual bits of an EAE instruction. The sole restriction for the development of "n" is that the microcoded operations must not occur during the same time state, if they logically conflict.

Symbolic: No operation.

INCLUSIVE OR SC WITH AC



Mnemonic Name: OSC

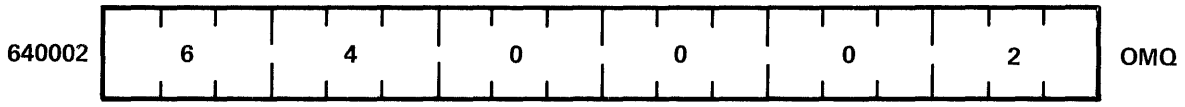
Octal Code: 640001

Time: 1.325 μs

Operation: The contents of the AC are inclusively ORed with the 6-bit contents of the step counter (SC) on a bit-by-bit basis. The result is left in AC_{12-17}. If corresponding SC and AC bits are in the binary 1 state, the AC bit is set to 1. The previous contents of the AC are lost. The contents of the SC are unchanged.

Symbolic: SC V AC → AC

INCLUSIVE OR MQ WITH AC



Mnemonic Name: OMQ

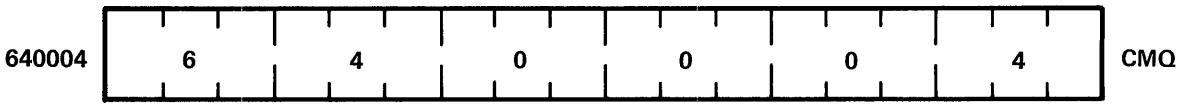
Octal Code: 640002

Time: 1.325 μs

Operation: The contents of the MQ are inclusively ORed with the contents of the AC on a bit-by-bit basis. The result is left in the AC. If corresponding MQ and AC bits are in the binary 0 state, the AC bit is cleared to 0. If either of the corresponding bits is in the binary 1 state, the AC bit is set to 1. The previous contents of the AC are lost. The contents of the MQ are unchanged.

Symbolic: MQ V AC → AC

COMPLEMENT MQ



Mnemonic Name: CMQ

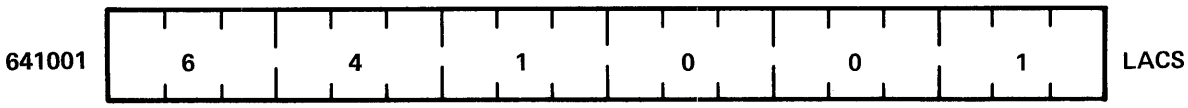
Octal Code: 640004

Time: 1.325 μs

Operation: Each bit of the MQ is set or cleared to the inverse of its current state. The previous contents of the MQ are lost.

Symbolic: MQ → MQ

LOAD AC FROM SC



Mnemonic Name: LACS

Octal Code: 641001

Time: 1.325 μs

Operation: This microcoded instruction clears each bit of the AC to 0 and then enters the contents of the SC in AC_{12-17}. The previous contents of the AC are lost. The contents of the SC are unchanged.

Symbolic: SC → AC

LOAD AC FROM MQ

Mnemonic Name: LACQ

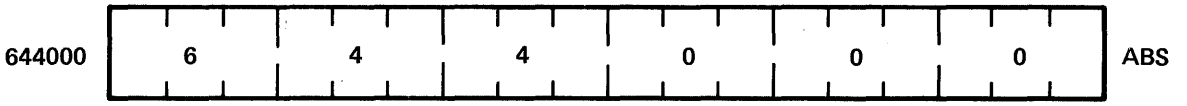
Octal Code: 641002

Time: 1.325 μs

Operation: This microcoded instruction clears each bit of the AC to 0 and then enters the contents of the MQ in the AC. The previous contents of the AC are lost. The contents of the MQ are unchanged.

Symbolic: MQ → AC

LOAD AC WITH ABSOLUTE VALUE TO AC



Mnemonic Name: ABS

Octal Code: 644000

Time: 1.325 μs

Operation: A microcoded instruction which complements the contents of the AC (1's complement notation), if the content of AC_0 is 1.

Symbolic: If $AC_0 = 1$, $\overline{AC} \rightarrow AC$

CLEAR MQ

Mnemonic Name: CLQ

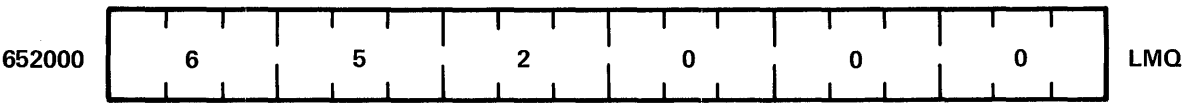
Octal Code: 650000

Time: 1.325 μs

Operation: Each bit of the MQ is cleared to 0. The previous contents of the MQ are lost.

Symbolic: 0 → MQ

LOAD MQ



Mnemonic Name: LMQ

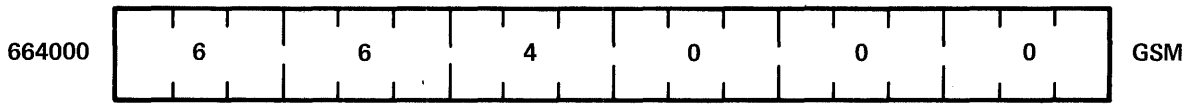
Octal Code: 652000

Time: 1.325 μs

Operation: A microcoded instruction which clears each bit of the MQ to 0 and then enters the contents of the AC in the MQ. The previous contents of the MQ are lost. The contents of the AC are unchanged.

Symbolic: AC → MQ

GET SIGN AND MAGNITUDE OF AC



Mnemonic Name: GSM

Octal Code: 664000

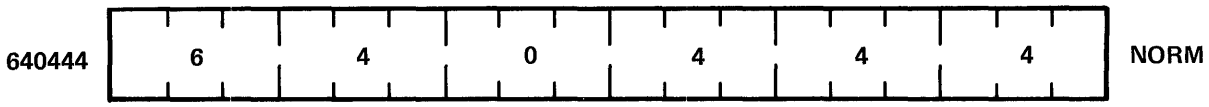
Time: 1.325 μs

Operation: A micrododed instruction which enters the contents of the AC_0 in the link and then complements the contents of the AC (1's complement notation), if AC_0 is a 1. The previous content of the link is lost.

Symbolic: $AC_0 \rightarrow L$
If $AC_0 = 1$, $\overline{AC} \rightarrow AC$

6.1.2 EAE Shifting Instructions

NORMALIZE



Mnemonic Name: NORM

Octal Code: 640444

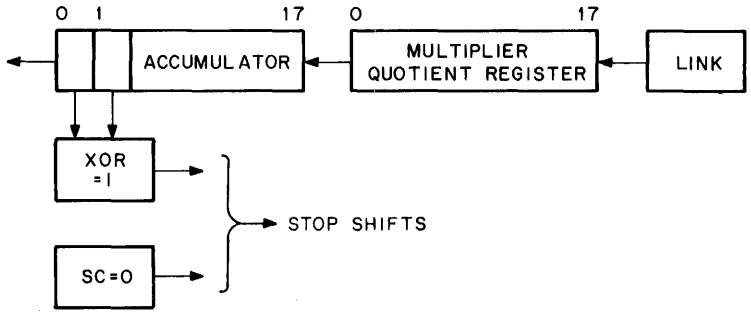
Time: 2.915 + 0.133 (n-1)* μs

Operation: The contents of the AC and the MQ are shifted left (i.e., leading zeros are shifted out) with the AC and MQ functioning as a serial 36-bit register until the content of the AC_0 does not agree with the content of AC_1, i.e., the bits differ in their binary states, or the contents of the step counter reaches zero.

This 6-bit counter is initialized to the 2's complement of 44_8 (36_{10} steps). The contents of the six low order bits of the NORM instruction word specify the step count. For each shift step, the contents of MQ_0 enter AC_{17} and the contents shifted out of AC_0 are lost. The content of the link, usually initialized to zero, enters MQ_{17} to replace the contents of vacated bits. If shifting halts because AC_0 does not equal AC_1, the contents of the step counter reflect the number of steps executed to reach the condition. The counter's contents (2's complement of the step count plus the steps executed) are accessible through use of the OSC or LACS instruction.

When not in user mode, or when the memory protect option is not installed, two free instructions follow the execution of the NORM instruction. A PI or API break cannot occur until the second instruction following the NORM instruction is completed.

Graphic:



*This quantity is also 0 for n=0.

NORMALIZE, SIGNED

Mnemonic Name: NORMS

Octal Code: 660444

Time: 2.915 + 0.133 (n-1)* μs

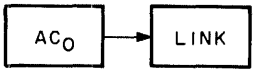
Operation: The contents of AC_0 enter the link. Then, the contents of the AC and the MQ are shifted left (i.e., leading zeros are shifted out) with the AC and MQ functioning as a serial 36-bit register until the contents of the AC_0 do not agree with the contents of AC_1, i.e., the bits differ in their binary states, or the contents of the step counter reaches zero.

This counter is initialized to the 2's complement of 44_8 (36_{10} steps). The contents of the six low order bits of the NORMS instruction word specify the step count. For each shift step, the content of MQ_0 enters AC_{17} and the contents shifted out of AC_0 are lost. The content of the link enters MQ_{17} to replace the contents of vacated bits. If shifting halts because AC_0 does not equal AC_1, the contents of the step counter reflect the number of steps executed to reach the condition. The counter's contents (2's complement of the step count plus the steps executed) are accessible through use of the OSC or LACS instruction.

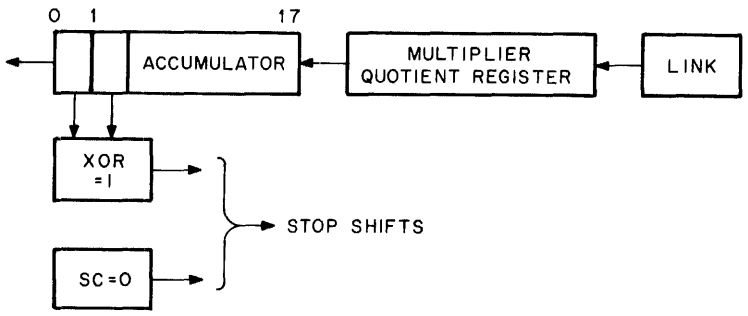
When not in user mode, or when the memory protect option is not installed, two free instructions follow the execution of the NORMS instruction. A PI or API break cannot occur until the second instruction following the NORMS instruction is completed.

Graphic:

Setup



Execution



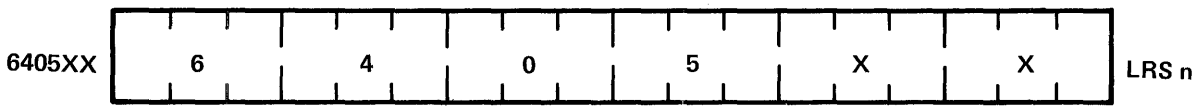
15-0195

*This quantity is 0 for n=0.

Programming Note

The EAE instruction set does not provide a convenient way to restore the contents of the step counter. To obviate the need to do so, the PDP-15 is designed to inhibit program or automatic priority interrupts occurring for two instructions following the NORM or NORMS (normalize, signed) instruction. These two instructions are normally a DAC followed by a LACS which saves the contents of the AC, then puts the contents of the step counter in the AC. Thus, if interrupt-accessed subroutines make use of the EAE, the AC and MQ are the only registers which must be preserved during the interrupt, then restored in the EAE at the completion of the interrupt service.

LONG RIGHT SHIFT



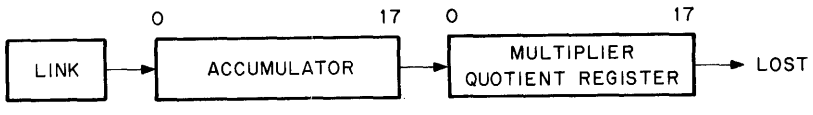
Mnemonic Name: LRS n

Octal Code: 6405XX +n

Time: $2.915 + 0.133\ (n-1)^{\dagger}\ \mu s$

Operation: The AC and MQ function as a 36-bit register to permit serial shifting of their contents "n" bit positions to the right, "n" being specified by the contents of the six low order bits of the instruction word. Shifting halts when the contents of the step counter, initialized to the 2's complement of "n", reach zero. For each shift step, the contents of AC_{17} enter MQ_0 and the contents shifted out of MQ_{17} are lost. The contents of the link, usually initialized to zero, remains unchanged and enters AC_0 at each step to replace the contents of vacated bits.

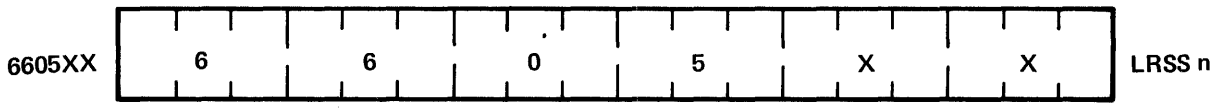
Graphic:



15-0196

† This quantity is also 0 for n = 0.

LONG RIGHT SHIFT, SIGNED



Mnemonic Name: LRSS n

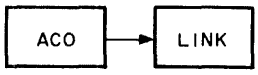
Octal Code: 6605XX +n

Time: 2.915 + 0.133 (n-1) † μs

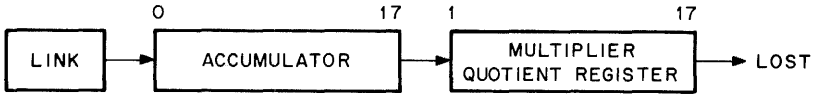
Operation: The content of AC_0 is entered in the link. Then, the AC and the MQ function as a 36-bit register to permit serial shifting of their contents "n" bit positions to the right, "n" being specified by the contents of the six low order bits of the instruction. Shifting halts when the contents of the step counter, initialized to the 2's complement of "n", reach zero. For each shift step, the contents of AC_{17} enter MQ_0 and the contents shifted out of MQ_{17} are lost. The content of the link remain unchanged and enters AC_0 at each step to replace the contents of vacated bits.

Graphic:

Setup



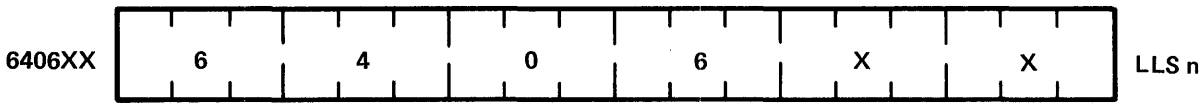
Execution



15-0197

†This quantity is also 0 for n = 0.

LONG LEFT SHIFT



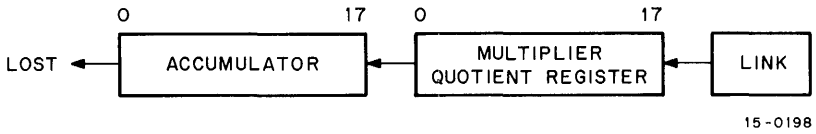
Mnemonic Name: LLS n

Octal Code: 6406XX +n

Time: 2.915 + 0.133 (n-1) † μs

Operation: The AC and the MQ function as a 36-bit register to permit serial shifting of their contents "n" bit positions to the left, "n" being specified by the contents of the six low order bits of the instruction word. Shifting halts when the contents of the step counter initialized to the 2's complement of "n", reach zero. For each shift step, the contents of MQ_0 enter AC_{17} and the contents shifted out of AC_0 are lost. The content of the link, usually initialized to zero, remains unchanged and enters MQ_{17} at each step to replace the contents of vacated bits.

Graphic:



† This quantity is also 0 for n = 0.

LONG LEFT SHIFT, SIGNED

Mnemonic Name: LLSS n

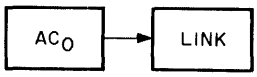
Octal Code: 6606XX +n

Time: 2.915 + 0.133 (n-1)† μs

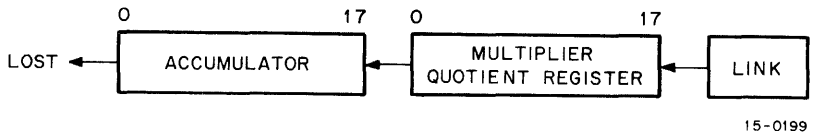
Operation: The content of AC is entered in the link. The AC and MQ function as a serial 36-bit register to permit serial shifting to their contents "n" bit positions to the left, "n" being specified by the contents of the six low order bits of the instruction word. Shifting halts when the contents of the step counter, initialized to the 2's complement of "n", reach zero. For each shift step, the contents of MQ_0 enter AC_{17} and the contents shifted out of AC_0 are lost. The content of the link remains unchanged and enters MQ_{17} at each step to replace the contents of vacated bits.

Graphic:

Setup

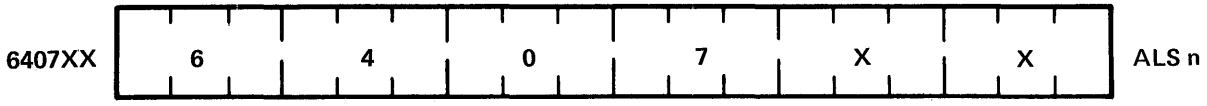


Execution



† This quantity is also 0 for n = 0.

ACCUMULATOR LEFT SHIFT



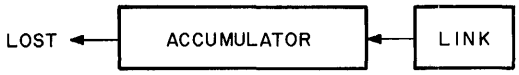
Mnemonic Name: ALS n

Octal Code: 6407XX +n

Time: 2.915 + 0.133 (n-1)† μs

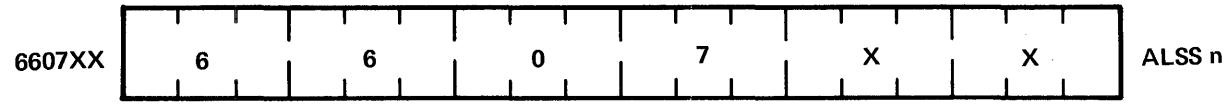
Operation: The contents of the AC are shifted "n" bit positions to the left, "n" being specified by the contents of the six low order bits of the instruction word. Shifting halts when the contents of the step counter, initialized to the 2's complement of "n", reach zero. For each shift step, the content of the link, usually initialized to zero, enters AC_{17} to replace the contents of vacated bits. The contents shifted out of AC_0 are lost.

Graphic:



15-0200

ACCUMULATOR LEFT SHIFT, SIGNED



Mnemonic Name: ALSS n

Octal Code: 6607XX +n

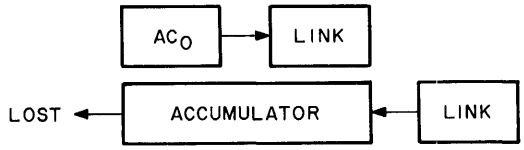
Time: 2.915 + 0.133 (n-1)† μs

Operation: The content of AC_0 enters the link. Then, the contents of the AC are shifted "n" bit positions to the left, "n" being specified by the contents of the six low order bits of the instruction word. Shifting halts when the contents of the step counter, initialized to the 2's complement of "n", reach zero. For each shift step, the content of the link remains unchanged and enters AC_{17} to replace the contents of vacated bits. The contents shifted out of AC_0 are lost.

Graphic

Setup

Execution

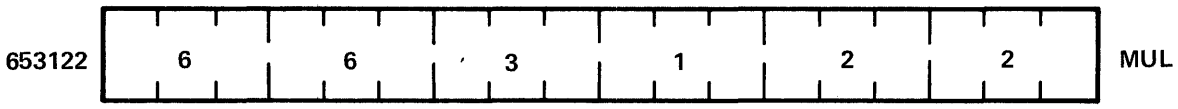


15-0201

†This quantity is also 0 for n = 0.

6.1.3 EAE Arithmetic Instructions

MULTIPLY, UNSIGNED



Mnemonic Name: MUL

Octal Code: 653122

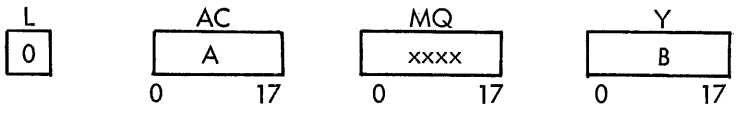
Time: 7.420 μs

Operation: Multiply the contents of memory register Y (the multiplicand) by the contents of the MQ (the multiplier), and place the resulting 36-bit product in the AC and the MQ with the more significant half appearing in the AC. The address of Y is taken to be sequential to the address of the MUL instruction word. Prior to this instruction, the contents of the link must be zero and the multiplier must be entered in the AC. During the set-up phase of MUL, the multiplier is transferred to the MQ, the AC is cleared to zero, and the step counter is initialized to the 2's complement of 22_8 (18_{10} steps); the six low order bits of the instruction word specify the step count. The arithmetic phase, executed as multiplication of one unsigned quantity by another (18 bits, binary point of no consequence), halts when the step counter counts up to zero. The content of the link remains zero. The contents of Y are unchanged. The program resumes as the next instruction (memory register Y + 1).

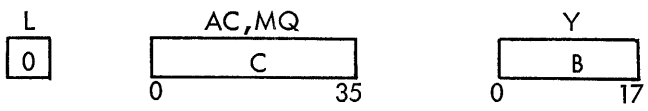
Symbolic:
0 → SC
Y · MQ → (AC, MQ)
0 → L
PC+2 → PC

Data Structure: C = A · B

Pre-execution:



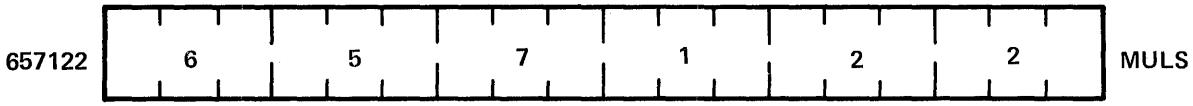
Post execution:



INSTRUCTION SEQUENCE:

Register	Contents
Y-2	LAC Multiplier
Y-1	MUL
Y	Multiplicand
Y+1	Next Instruction

MULTIPLY, SIGNED



Mnemonic Name: MULS

Octal Code: 657122

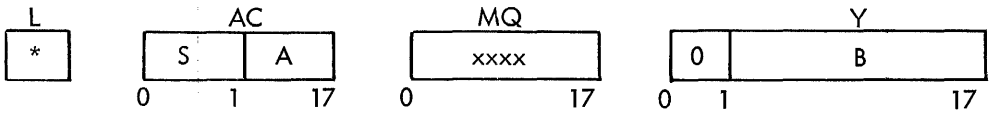
Time: 7.420 μs

Operation: Multiply the contents of memory register Y (the multiplicand) by the contents of the MQ (the multiplier), and place the signed product in the AC and MQ with the sign notation and more significant portion in the AC. Bits AC_0 and AC_1 each receive the sign of the product; the remaining AC and MQ bits represent the magnitude of the product in 1's complement form. The address of Y is taken to be sequential to the address of the MULS instruction word. The contents of the Y are taken to be the absolute value of the multiplicand; the contents of the link are taken to be the original sign of the multiplicand (MULS assume previous execution of an EAE GSM instruction, q.v.). Just prior to this MULS instruction, the multiplier must be entered in the AC. During the setup phase of the MULS instruction, the multiplier is transferred to the MQ and 1's complemented if negative, the AC is cleared to zero, and the step counter is initialized to the 2's complement of 228 (18_{10} steps); the six low order bits of the MULS instruction word specify the step count. The arithmetic phase, executed as multiplication of one signed quantity by another (sign bit plus 17 magnitude bits, binary point position of no consequence), halts when the step counter counts up to zero. The link is cleared to zero. The contents of Y are unchanged. The program resumes at the next instruction (memory register Y + 1).

Symbolic:
0 → SC
Y · MQ → (AC, MQ)
0 → L
PC+2 → PC

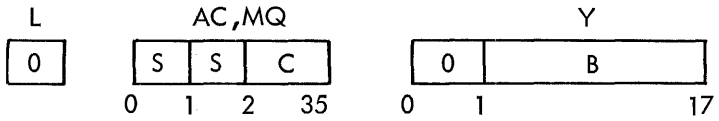
Data Structure: C = A · B

Pre-execution:



*Original sign of B.

Post execution:



S = L ⊻ Sign A

INSTRUCTION SEQUENCE:

Register	Contents
Y-5	LAC Multiplicand
Y-4	GSM (take absolute value and save sign in link)
Y-3	DAC Y
Y-2	LAC Multiplier
Y-1	MULS
Y	Multiplicand (absolute value)
Y+1	Next Instruction

DIVIDE, UNSIGNED

Mnemonic Name: DIV

Octal Code: 640323

Time: 7.685 μs

Operation: Divide the contents of the AC and the MQ (an unsigned 36-bit dividend) by the contents of memory register Y (the divisor). The resulting quotient appears in the MQ; the remainder is in the AC. The address of Y is taken to be sequential to the address of the DIV instruction word. Prior to this, the contents of the link must be zero, and the dividend must be entered in the AC and MQ (LAC least significant half). If the divisor is not greater than the AC portion of the dividend, divide overflow occurs (magnitude of quotient exceeds the 18-bit capacity of the MQ), and the link is set to one to signal the overflow condition; data in the AC and the MQ are of no value. A valid division halts when the step counter, initialized to the 2's complement of 23_8 (19_{10} steps), counts up to zero (the six low order bits of the DIV instruction word specify the step count). The contents of the Y are unchanged. The program resumes at the next instruction (memory register Y + 1).

Symbolic:

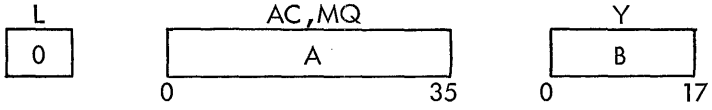
If Y $\leq$ AC, 1 → L (divide overflow)

If Y > AC,
0 → SC
(AC,MQ)/Y → MQ (quotient), AC (remainder)

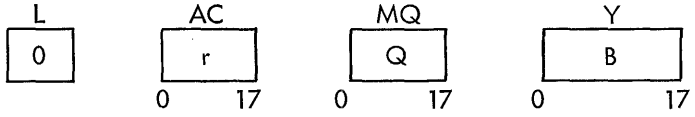
0 → L
PC+2 → PC

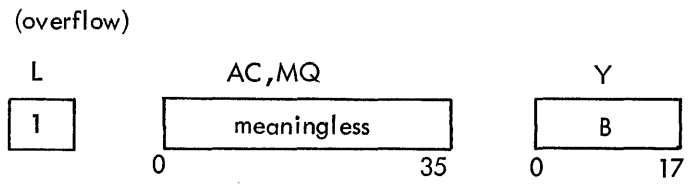
Data Structure: A = BQ + r

Pre-execution:



Post execution: (no overflow)

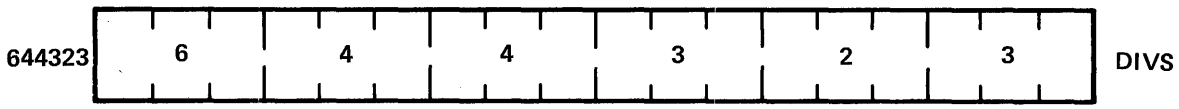




INSTRUCTION SEQUENCE:

Register	Contents
Y-4	LAC Dividend (least significant half)
Y-3	LMQ
Y-2	LAC Dividend (most significant half)
Y-1	DIV
Y	Divisor
Y+1	Next instruction

DIVIDE, SIGNED



Mnemonic Name: DIVS

Octal Code: 644323

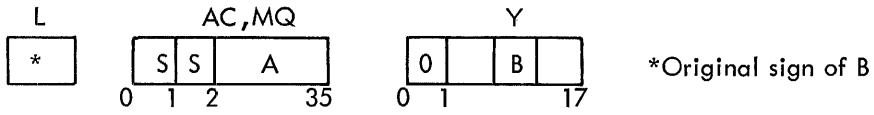
Time: 7.685 μs

Operation: Divide the contents of the AC and MQ (a 36-bit signed dividend with the sign in bits AC_0 and AC_1 and the remaining 34 bits devoted to magnitude) by the contents of memory register Y (the divisor). The resulting quotient appears in the MQ with the algebraically determined sign in bit MQ_{1-17}. The remainder is in the AC with bit AC_0 containing the sign of the dividend and bits AC_{1-17} containing the magnitude (1's complement). The address of Y is taken to be sequential to the address of the DIVS instruction word. The contents of Y are taken to be the absolute value of the divisor; the contents of the link are taken to be the original sign of the divisor (DIVS assumes previous execution of an EAE GSM instruction, q.v.). Prior to this DIVS instruction, the dividend must be entered in the AC and MQ (LAC of lease significant half, LMQ, and LAC of most significant half). The MQ portion of a negative dividend is 1s complement prior to the division. If the divisor is not greater than the AC portion of the dividend, divide overflow occurs (magnitude of the quotient exceeds the 17-bit plus sign capacity of the MQ), and the link is set to one to signal the overflow condition; data in the AC and the MQ are of no value. A valid division halts when the step counter, initialized to the 2's complement of 23_8 (19_{10} steps), counts up to zero (the six low order bits of the DIVS instruction word specify the step count). The content of the link is cleared to zero. The contents of Y are unchanged. The program resumes at the next instruction (memory register Y + 1).

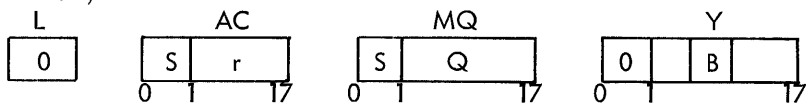
Symbolic:
If $Y \leq |AC|$, 1 → L (divide overflow)
If $Y > |AC|$,
0 → SC
(AC,MQ)/Y → MA (quotient), AC (remainder)
0 → L
PC+2 → PC

Data Structure: |A| = B Q+r

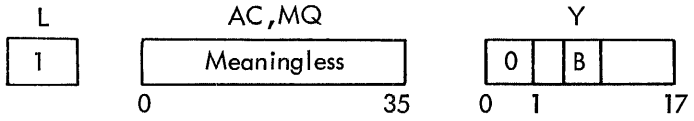
Pre-execution:



Post execution:
(no overflow)



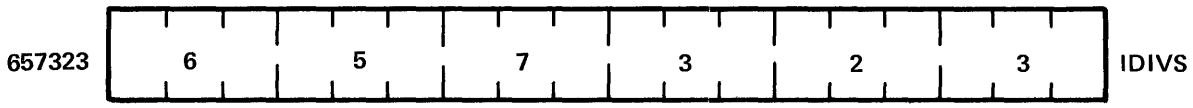
(overflow) (S=Sign A) (S=Sign A ⊻ L)



INSTRUCTION SEQUENCE:

Register	Contents
Y-7	LAC Divisor
Y-6	GSM
Y-5	DAC Divisor in Y
Y-4	LAC Dividend (least significant half)
Y-3	LMQ
Y-2	LAC Dividend (most significant half)
Y-1	DIVS
Y	Divisor (absolute value)
Y+1	Next Instruction

INTEGER DIVIDE, SIGNED



Mnemonic Name: IDIVS

Octal Code: 657323

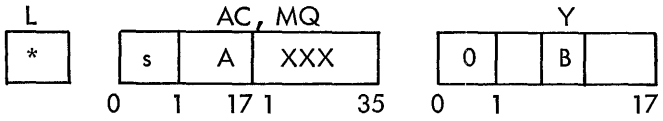
Time: 7.685 μs

Operation: Divide the contents of the AC and the MQ (AC is zero, MQ contains a signed integer dividend) by the contents of memory register Y (the divisor). The resulting quotient appears in the MQ with the algebraically determined sign in bit MQ_0 and the magnitude (1's complement) in bits MQ_{1-17}. The remainder is in the AC with bit AC_0 containing the sign of the dividend and bits AC_{1-17} containing the magnitude (1's complement). The address of Y is taken to be sequential to the address of the IDIVS instruction word. The contents of Y are taken to be the absolute value of the divisor; the contents of the link are taken to be the original sign of the divisor (IDIVS assumes previous execution of an EAE GSM instruction, q.v.). Prior to this IDIVS instruction, the dividend must be entered in the AC (the setup phase of IDIVS transfers the dividend to the MQ, clears the AC, and 1's complements the MQ if the dividend is negative). Divide overflow occurs only if division by zero is attempted; i.e., the quotient's magnitude will not exceed the 17-bit plus sign capacity of the MQ. The division halts when the step counter, initialized to the 2's complement of 23_8 (19_{10} steps), counts up to zero (the six low order bits of the IDIVS instruction word specify the step count). The contents of the link are cleared to zero. The contents of Y are unchanged. The program resumes at the next instruction (memory register Y + 1).

Symbolic:
0 → SC
MQ/Y → MQ (quotient), AC (remainder)
0 → L
PC+2 → PC

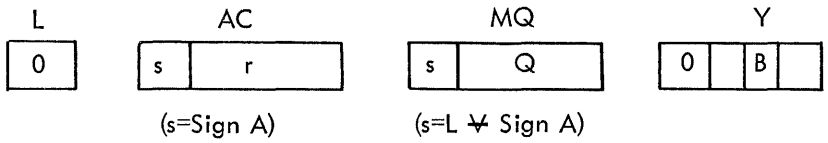
Data Structure: A = B Q+r

Pre-execution:

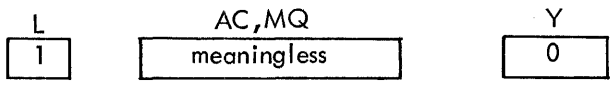


*Original sign of B

Post execution:



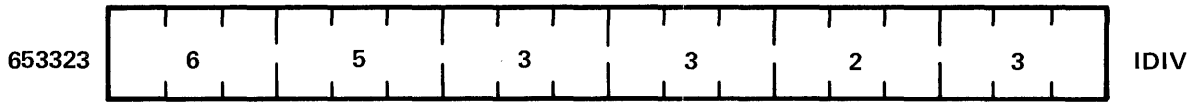
If Y=0 (overflow)



INSTRUCTION SEQUENCE:

Register	Contents
Y-5	LAC Divisor
Y-4	GSM
Y-3	DAC Divisor (absolute value) in Y
Y-2	LAC Dividend
Y-1	IDIVS
Y	Divisor (absolute value)
Y-1	Next Instruction

INTEGER DIVIDE, UNSIGNED



Mnemonic Name: IDIV

Octal Code: 653323

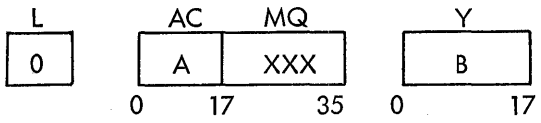
Time: 7.685 μs

Operation: Divide the contents of the AC and the MQ (AC is zero, MQ contains a 18-bit integer dividend) by the contents of memory register Y (divisor). The resulting quotient appears in the MQ; the remainder is in the AC. The address of Y is taken to be sequential to the address of the IDIV instruction word. Prior to this instruction, the contents of the link must be zero, and the dividend must be entered in the AC (the setup phase of IDIV transfers the dividend to the MQ and clears the AC). Division overflow occurs only if division by zero is attempted, i.e., the quotient's magnitude will not exceed the 17-bit plus sign capacity of the MQ. The division halts when the step counter, initialized to the 2's complement of 23_0 (19_{10} steps), counts up to zero (the six low order bits of the IDIV instruction word specify the step count). The content of the link is cleared to zero. The contents of Y are unchanged. The program resumes at the next instruction (memory register Y+1).

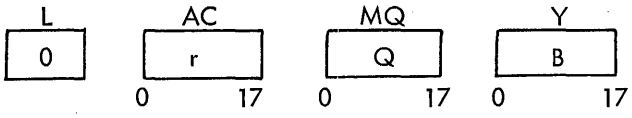
Symbolic:
0 → SC
MQ/Y → MQ (quotient), AC (remainder)
0 → L
PC+2 → PC

Data Structure: A = BQ+r

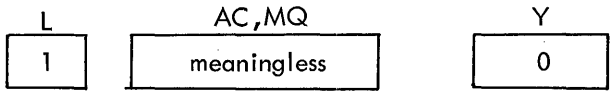
Pre-execution:



Post execution:



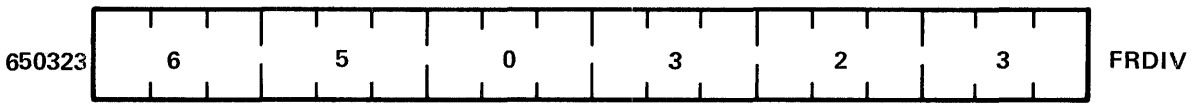
If Y=0 (overflow)



INSTRUCTION SEQUENCE:

Register	Contents
Y-2	LAC Dividend
Y-1	IDIV
Y	Divisor
Y+1	Next Instruction

FRACTION DIVIDE, UNSIGNED



Mnemonic Name: FRDIV

Octal Code: 650323

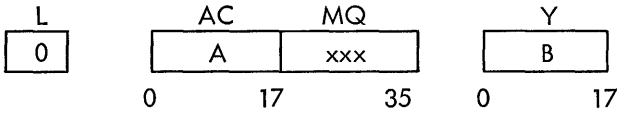
Time: 7.685 μs

Operation: Divide the contents of the AC and the MQ (AC contains an 18-bit fractional dividend, MQ is zeroed at stepup) by the contents of memory register Y (the divisor). The binary point is assumed at the left of AC_0. The quotient appears in the MQ; the remainder is in the AC. The address of Y is taken to be sequential to the address of the FRDIV instruction word. Prior to this instruction, the contents of the link must be zero, and the dividend must be entered in the AC (the set-up phase of FRDIV clears the MQ). If the divisor is not greater than the dividend, divide overflow occurs (magnitude of quotient exceeds the 18-bit capacity of the MQ), and the link is set to one to signal the overflow condition; data in the AC and the MQ are of no value. A valid division halts when the step counter, initialized to 238 (19_{10} steps), counts up to zero (the six low order bits of the FRDIV instruction word specify the step count). The contents of the link remain zero. The contents of Y are unchanged. The program assumes at the next instruction (memory register Y + 1).

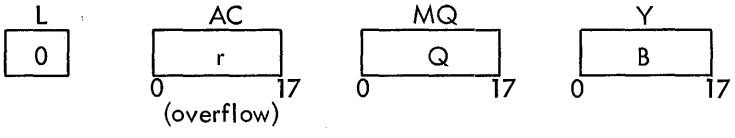
Symbolic:
If $Y \leq |AC|$, 1 → L (divide overflow)
If $Y > AC$,
0 → SC
AC/Y → MQ (quotient), AC (remainder)
0 → L
PC+2 → PC

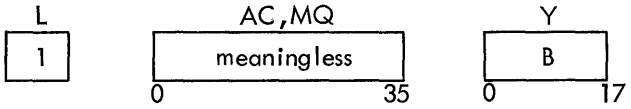
Data Structure: A = BQ+r

Pre-execution:



Post execution: (no overflow)

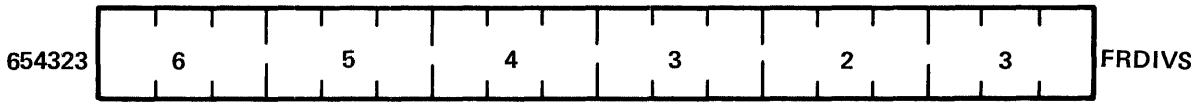




INSTRUCTION SEQUENCE:

Register	Contents
Y-2	LAC Dividend
Y-1	FRDIV
Y	Divisor
Y+1	Next Instruction

FRACTION DIVIDE, SIGNED



Mnemonic Name: FRDIVS

Octal Code: 654323

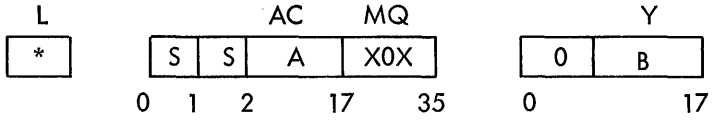
Time: 7.685 μs

Operation: Divide the contents of the AC and the MQ (AC contains a 18-bit signed dividend with the sign in bits AC_0 and AC_1 and the remaining 16 bits devoted to magnitude, MQ is zeroed at setup) by the contents of memory register Y (the divisor). The binary point is assumed between AC_0 and AC_1. The resulting quotient appears in the MQ with the algebraically determined sign in bit MQ_0 and the magnitude (1's complement) in bits MQ_{1-17}. The remainder is in the AC with bit AC_0 containing the original sign of the dividend and bits AC_{1-17} containing the magnitude (1's complement). The address of Y is taken to be sequential to the address of the FRDIVS instruction word. The contents of Y are taken to be the absolute value of the divisor; the contents of the link are taken to be the original sign of the divisor (FRDIVS assumes previous execution of an EAE GSM instruction, q.v.). Prior to this FRDIVS instruction, the dividend must be entered in the AC (the setup phase of FRDIVS clears the MQ and 1's complements the dividend, if negative, prior to the division). If the divisor is not greater than the dividend, divide overflow occurs (magnitude of the quotient exceeds the 18-bit capacity of the MQ) and the link is set to one to signal the overflow condition. Data in the AC and the MQ are of no value. A valid division halts when the step counter, initialized to the 2's complement of 23_8 (19_{10} steps), counts up to zero (the six low order bits of the FRDIVS instruction word specify the step count). The contents of the link are cleared to zero. The contents of Y are unchanged. The program resumes at the next instruction (memory register Y + 1).

Symbolic:
If $Y \leq AC$, 1 → L (divide overflow)
If $Y > AC$,
0 → SC
AC/Y → MQ (quotient), AC (remainder)
0 → L
PC+2 → PC

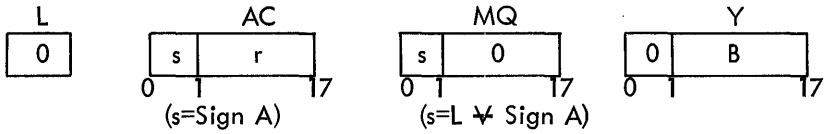
Data Structure: A = BQ+r

Pre-execution:

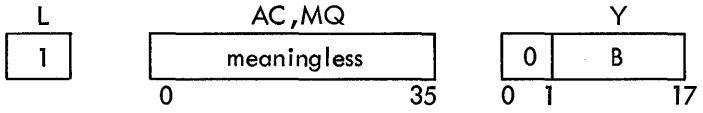


*original sign of B

Post Execution: (no overflow)



(overflow)



INSTRUCTION SEQUENCE:

Register	Contents
Y-5	LAC Divisor
Y-4	GSM
Y-3	DAC Divisor (absolute value) in Y
Y-2	LAC Dividend
Y-1	FRDIVS
Y	Divisor (absolute value)
Y+1	Next Instruction

6.2 KM15 MEMORY PROTECT

The KM15 provides the PDP-15 with the capability of running in a background/foreground environment. This capability is accomplished through a programmable boundary register, which establishes the boundary between protected and unprotected areas of core memory. There is also a specific set of instructions termed illegal, because they would interfere with background/foreground operation.

A PDP-15, equipped with a KM15 Memory Protect option, has two modes of operation: User Mode (KM15 enabled) and Monitor Mode (KM15 disabled).

When operating in User Mode, the KM15 monitors addresses and instructions to determine their legality and prevent execution of anything illegal. When operating in Monitor Mode, operation is identical to normal PDP-15 operation. When in User Mode, upon detection of a violation, a flag is raised, User Mode is turned off, and a pseudo program interrupt is caused.

When enabled, the option traps the following:

OAS
IOT
HLT
XCT of XCT
References below the boundary
References to nonexistent memory

User Mode is enabled by either placing the Protect switch on the console in the "1" position and depressing the Start key, or by issuing the IOT, MPEU.

User Mode is disabled in the following ways:

I/O Reset Key
Detection of a violation
CAL Instruction
Program Interrupt
API Interrupt

The state of the Protect Mode (a "1" for User Mode) is stored in bit 2 of the storage word by those operations that save the state of the machine (CAL, JMS, PI, and TRAP). For a violation, the Stored Address is one more than the location containing the violation instruction, except for a JMP to a protected area. In this case, the stored PC equals the Protected Address.

Not all memory reference instructions are prevented from addressing below the boundary; any "read" instruction with the exception of JMP and ISZ is permitted to address below the boundary, because a read instruction cannot modify core. All write instructions and JMP and ISZ are prevented from addressing below the boundary.

A nonexistent memory violation is one in which a reference is made to a memory location which does not exist in that particular system. When running in User Mode, such a violation causes the Protect violation flag and the nonexistent memory flag to get set and the computer proceeds through the trap. When running in Monitor Mode, a reference to a nonexistent memory causes the nonexistent memory flag to get set and the computer hangs up waiting for memory to respond. Because a memory with that address is not present, the computer remains hung until manually reset by depressing Stop and Reset at the same time.

All violations trap to address 20 where the state of the machine is saved. The instruction in 21 is then executed. However, if the program interrupt facility is enabled prior to the trap, the trap goes to address 0 where the state of the machine is saved and location "1" is executed. The program interrupt facility is disabled as when servicing any program interrupt. The API, if present, is raised to Level 3.

Some special cases of importance are explained as follows.

CAL Instruction

When in User Mode and the CAL is executed, User Mode is disabled (Monitor Mode envoked). The CAL goes to location 20 as usual and saves the state of the machine. No violation occurs.

Program Interrupt

When a program interrupt occurs, User Mode is disabled. The interrupt goes to location 0 and saves the state of the machine. No violation occurs.

Automatic Priority Interrupt

An API break causes Monitor Mode to be entered. The instruction in the address specified by the I/O device is executed. No violation occurs. The instruction in this address is usually a JMS, JMS I, or a CAL to the device handler. The device handler entry receives the state of the machine.

Data Channel

Data channel operations do not cause violations, even though they can reference addresses below the boundary. They do, however, produce a nonexistent memory violation, if nonexistent memory is referenced.

Real-Time Clock

The real-time clock increments address 7, which is below the boundary. No violation occurs. However, attempts to reinitialize address 7 must be done in Monitor Mode.

Auto Increment Register

The auto increment register can be used while in User Mode. No violation occurs, unless the address to which the register points is below the boundary. Reinitialization of these registers must be done in Monitor Mode.

DBR and RES Instructions

The DBR and RES instructions can be used to return to User Mode from Monitor Mode. This is accomplished by issuing DBR or RES and then a JMP I to the subroutine entry. If bit 2 is a "1" in the indirect location, User Mode is restored. The instruction set for the KM15 option is listed in Table 6-3.

NORM Instruction

In User Mode, the execution of a NORM instruction will not cause two free instructions following the NORM.

Table 6-3
KM15 Instruction Set

Mnemonic	Octal Code	Operation Executed
MPSNE	701741	Skip on Nonexistent Memory flag
MPSK	701701	Skip on Protect Violation flag
MPEU	701742	Enter User Mode. Actual entry occurs after the start of the next instruction.
MPCV	701702	Clear Protect Violation flag
MPCNE	701744	Clear Nonexistent Memory flag
MPLD	701704	Load the boundary register with the contents of AC 01-09.

The following are sample programs which demonstrate how to program the KM15 option:

1. How to enter User Mode.

```
.LOC    100
        LAC X           /x=value to be set into boundary reg.
        MPLD            /load boundary reg.
        MPEU            /enter User Mode
        JMP * Y         /Y contains starting address of User's
                        /program. Should be greater than
                        /value in X.
```

2. What happens when violating instruction occurs in program.

```
.LOC    500
        LAC Z           /Z is above the boundary
        IAC
        DAC 200         /200 is below the boundary
                        /DAC does not get executed

.LOC    20
        100503          /User Mode and address + 1 saved

.LOC    21
        MPSNE           /Check NEXM flag
        SKP             /No
        JMP NEXMER      /Yes, Handle it
        MPSK            /Check PV flag
        SKP             /No
        JMP PVER        /Yes, Handle it
```

3. How to get back into program after interrupt.

```
.LOC    0
        100510          /Bit 2="1" User Mode was on.
        JMP PISERV

.LOC    150
        DBR or RES      /End of PI service routine
        JMP * 0         /User Mode will be restored.
```

6.3 KT15 MEMORY PROTECT AND RELOCATE

With the KT15, the PDP-15 has the capabilities of a programmable core allocation register or upper boundary register for specifying protected areas of core, a programmable relocation register for specifying a base address to which all addresses from the CPU are added producing a relocated address and an illegal instruction set. The KT15 also provides a detection network for detecting addressing of nonexistent memory.

A PDP-15, equipped with a KT15 Memory Protect and Relocate option, has two modes of operation: User Mode (KT15 enabled) and Monitor Mode (KT15 disabled). In Monitor Mode, the relocation and protect hardware are disabled and the machine functions as it would without a KT15. A program running in Monitor Mode addresses real locations within the system. In User Mode, the relocation and protect hardware is enabled. The machine is programmed as though the user had the machine all to himself. His memory begins from location 0 and goes up to and includes the last 256_{10} word page specified by the core allocation register (upper boundary register). In the real machine, the program is located from the contents of the relocation register up. In User Mode, addresses and instructions are checked for their legality. Anything illegal is trapped (hardware activated JMS).

Trap is a condition where the execution of an illegal instruction is inhibited and the systems monitor takes control.

The definition of a real machine in reference to the memory and relocate option is a machine having an absolute addressable bank 0 memory. While a virtual machine is defined as having a relative bank 0 addressing capacity.

The following instructions are trapped in User Mode.

OAS
IOT
HLT
XCT of XCT
References above the boundary
References to nonexistent memory

User Mode is enabled by either placing the Protect switch on the console in the "1" position and depressing the Start key, or by issuing the IOT MPEU.

User Mode is disabled in the following ways:

I/O Reset Key
Detection of a violation
CAL instruction
Program Interrupt
API Interrupt

The state of User Mode is stored in bit 2 of the storage word (a "1" for User Mode, a "0" for Monitor Mode) by those operations that save the state of the machine (CAL, JMS, PI, and TRAP).

A nonexistent memory violation is one in which a reference is made to a memory location which does not exist in that particular system. When running in User Mode, such a violation causes the protect violation flag and the nonexistent memory flag to get set; the computer proceeds through the trap. When running in Monitor Mode, a reference to a nonexistent memory causes a nonexistent memory flag to set, and the computer hangs up waiting for memory to respond. Because a memory with that address is not present, the computer remains hung until manually reset by depressing Stop and Reset at the same time.

All violations trap to address 20 in the real machine where the state of the machine is saved. The instruction in real 21 is then executed. However, if the program interrupt facility is enabled prior to the trap, the trap goes to real address 0 where the state of the machine is saved and real location "1" is executed. The program interrupt facility is disabled, as it is when servicing any program interrupt. The API, if present, is raised to Level 3.

Some special cases are defined below.

CAL Instruction

When in User Mode with the CAL instruction given, the User Mode is disabled (Monitor Mode envoked). The CAL goes to location 20 in the real machine (not the relocated machine) and saves the state of the machine. The PC saved is the virtual PC. The virtual PC is equivalent to the PC, if the program were operating from location 0 up.

Program Interrupt

When a program interrupt occurs, the User Mode is disabled and the interrupt goes to real location 0. The state of the machine is saved in real 0 and real 1 is executed.

Automatic Priority Interrupt

When in User Mode, an API break causes Monitor Mode to be entered, and an instruction in the real machine address, specified by the I/O device, is executed. This instruction is usually a JMS, JMS I, or a CAL to the device handler. The device handler runs in Monitor Mode. The device handler entry receives the state of the machine.

Data Channel

Data channel operations are never relocated.

Real-Time Clock

The real-time clock always increments real location 7. Attempts to reference the contents of location 7 in the real machine must be done in Monitor Mode.

Auto Increment Register

Each user has a complete set of auto increment registers located in locations 10 to 17 of the user's virtual machine. In addition, 10 to 17 in the real machine may be used in Monitor Mode.

DBR and RES Instructions

The DBR and RES instructions may be used when returning from Monitor Mode. The protect bit in the indirect location causes relocation if it is a "1". Since CAL, PI, API, JMS, and TRAP save the virtual PC, returns are sent to the correct location in memory. The virtual PC is restored and User Mode is envoked.

The instruction set for the KT15 is listed in Table 6-4.

Table 6-4
KT15 Instruction Set

Mnemonic	Octal Code	Operation Executed
MPSK	701701	Skip on Protect Violation flag
MPCV	701702	Clear Protect Violation flag
MPLD	701704	Load core allocation register (boundary register) with contents of AC or 01-09.
MPSNE	701741	Skip on Nonexistent Memory flag
MPEU	701742	Enter User Mode

Table 6-4 (Cont)
KT15 Instruction Set

Mnemonic	Octal Code	Operation Executed
MPCNE	701744	Clear Nonexistent Memory flag
MPLR	701724	Load relocation register with contents of AC 01-09.

The core allocation register of boundary register specifies the amount of core available to the user. An example of how to set the BR (boundary register) follows.

If the User is allotted 4K of core, the AC is loaded with bits 6, 7, 8, and 9. The IOT MPLD then loads the BR with those bits. Because bits 10-17 are not included in the BR and therefore not checked, addresses up to and including 7777 are legal. Address 10000, however causes a violation. Thus, the user was allotted 4K of core. The user can be allotted as little as 256_{10} locations by setting the BR to all zeros. Then, addresses 0 to 377 are legal; address 400 is illegal.

A few simple programming examples are given below.

1. Sequence for entering User Mode.

```
.LOC    100
        LAC X           /X= 017400
        MPLD            /Set BR for 8K
        LAC Y           /Y = 010000
        MPLR            /Set relocation register for base
                        /of 10000.
        MPEU            /Enter User Mode
        JMP 0           /0 gets relocated to 10000
```

Example 1 The user was alloted 8K of core and his storage area started in real address 10000. If the actual starting address of the user's program was an address other than 0, the JMP instruction should be modified to reflect this.

2. What happens when violating instruction occurs in a program? (Operating under Example 1 conditions.)

```
Relocated 200/  LAC * Z         /Z=020000
Real 20/ 100201                 /User Mode and virtual
                                /address +1.
Real 21/  MPSNE                 /Check NEXM flag
          SKP                   /No
          JMP NEXMER            /Yes, Handle it
```

```
MPSK            /Check PV flag
SKP             /No
JMP PVER        /Yes, Handle it
```

3. How to return after interrupt. (Operating under Example 1 conditions.)

```
Real 0/  100300          /User Mode was on
     1/  JMP PI SERV     /Go service interrupt

Real 150/ End of PI service
          DBR or RES
          JMP * 0        /User Mode will be restored.
                         /Return will go to 300 relocated.
```

6.4 MP15 MEMORY PARITY

The MP15 enables the PDP-15 to continuously check information being read from core memory, and to determine whether information has been erroneously picked up or dropped. It does this by first monitoring all information as it is being sent to the memory for storage. If there are an even number of bits in the data word, the MP15 control causes memory to write the parity bit, thus making the total number of bits odd. If there are an odd number of bits in the data word, the MP15 control inhibits the memory from writing the parity bit. Again, the word is stored with an odd number of bits. When information is read from core, the parity control checks to see that an odd number of bits are read. If it finds an even number, then a parity error has occurred. The parity error flag can be used to cause an API interrupt, a program interrupt, a skip request, or an immediate stop of the processor.

Table 6-5 contains the instruction set for the MP15.

The MP15 uses API channel address 53 and is on priority level 0.

When using the read-in feature, the last instruction on the paper tape (which is executed by the processor) will not be written into the next sequential memory location. That location, however, will be loaded with data that may contain wrong parity. This location should be restored by the program before an attempt is made to read from it. Otherwise, a parity error will occur.

Table 6-5
MP15 Instruction Set

Mnemonic	Octal Code	Operation Executed
SPE	702701	Skip on Parity Error flag
CPE	702702	Clear Parity Error flag
FWP	702704	Force wrong parity (maintenance only)

Mounted in the BB15 Peripheral Expander (slot A19) is a W714 switch card containing two microswitches. The top switch is used in conjunction with the MP15. When the switch is placed in the "up" position, a parity error causes an API interrupt, a program interrupt, or a skip request. With the switch in the "down" position, a parity error causes the processor to halt.

When using the program interrupt facility, the SPE instruction must be included in the skip chain used to determine the flag that causes the interrupt. When using the API facility, address 53 must contain a JMS to a parity error service routine. In all cases, before returning to the main program, the parity error flag must be cleared through use of CPE or CAF.

6.5 KF15 POWER FAIL OPTION

The Power Fail Option is designed to offer maximum protection to programs during power failure turn off, and recovery of power after failure. The option enables the PDP-15 System to store active registers in memory before power diminishes to a point beyond which data will be lost. The computer can be in one of three conditions when a power failure occurs: (1) the console lock has not been turned on (manual function), (2) the console lock is on and the program interrupt facility (PI) is also enabled, or (3) the console lock is on and the automatic priority interrupt facility is enabled.

Figures 6-7, 6-8, and 6-9 illustrate the power fail sequence.

6.6 KW15 REAL-TIME CLOCK OPTION

The real-time clock option gives the user a time reference capability to use in control processing. The real-time clock produces clock pulses at the rate of one every 16.7 ms for 60 Hz systems. When the real-time clock is enabled by an IOT instruction (CLON); the occurrence of each pulse initiates a request for a break at the completion of the current instruction.

At the grant of the break, the contents of the clock counter register (memory location 00007) are incremented by one. This memory location is program initialized to contain the 2's complement of the desired number of clock pulses. Clock breaks continue to be requested until the memory location 00007 overflows (reaches the all zeros condition).

At this time, the CLOCK flag is set to initiate interruption of the program in progress. The CLOCK flag is interfaced to the program interrupt control and to the API system. The real-time clock has priority over the API and PI requests.

The following IOT instructions are provided for use with the real-time clock.

Table 6-6
IOT Instructions for Real-Time Clock

Mnemonic	Octal Code	Function
CLON	700044	Clock on. This IOT instruction enables the real-time clock; the clock increments location 00007, and the CLOCK flag is cleared.
CLOF	700004	Clock off. This IOT instruction disables the real-time clock; the clock does not increment location 00007, and the CLOCK flag is cleared.
CLSF	700001	Skip on CLOCK flag. The next instruction is skipped if the CLOCK flag is set.

A number of sources can be used as the clock time source.

a. PDP-15 real-time clock (standard with option)
b. DEC positive logic clocks, both RC and crystal type.

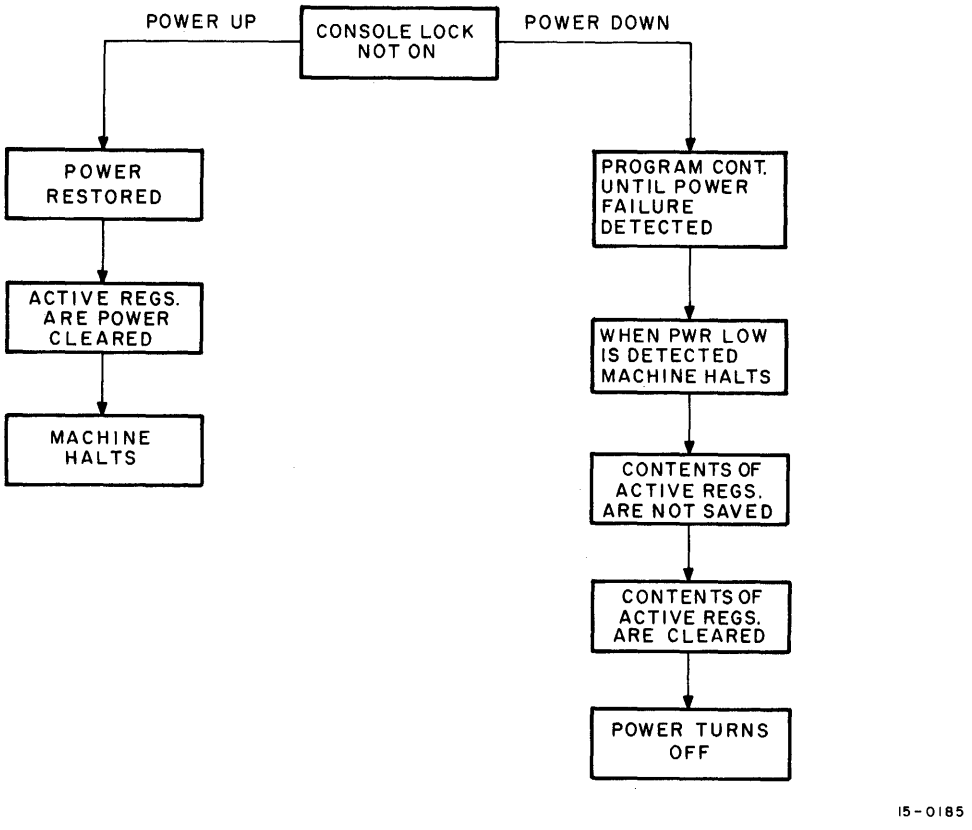


Figure 6-7 Power Fail Up/Down Sequence

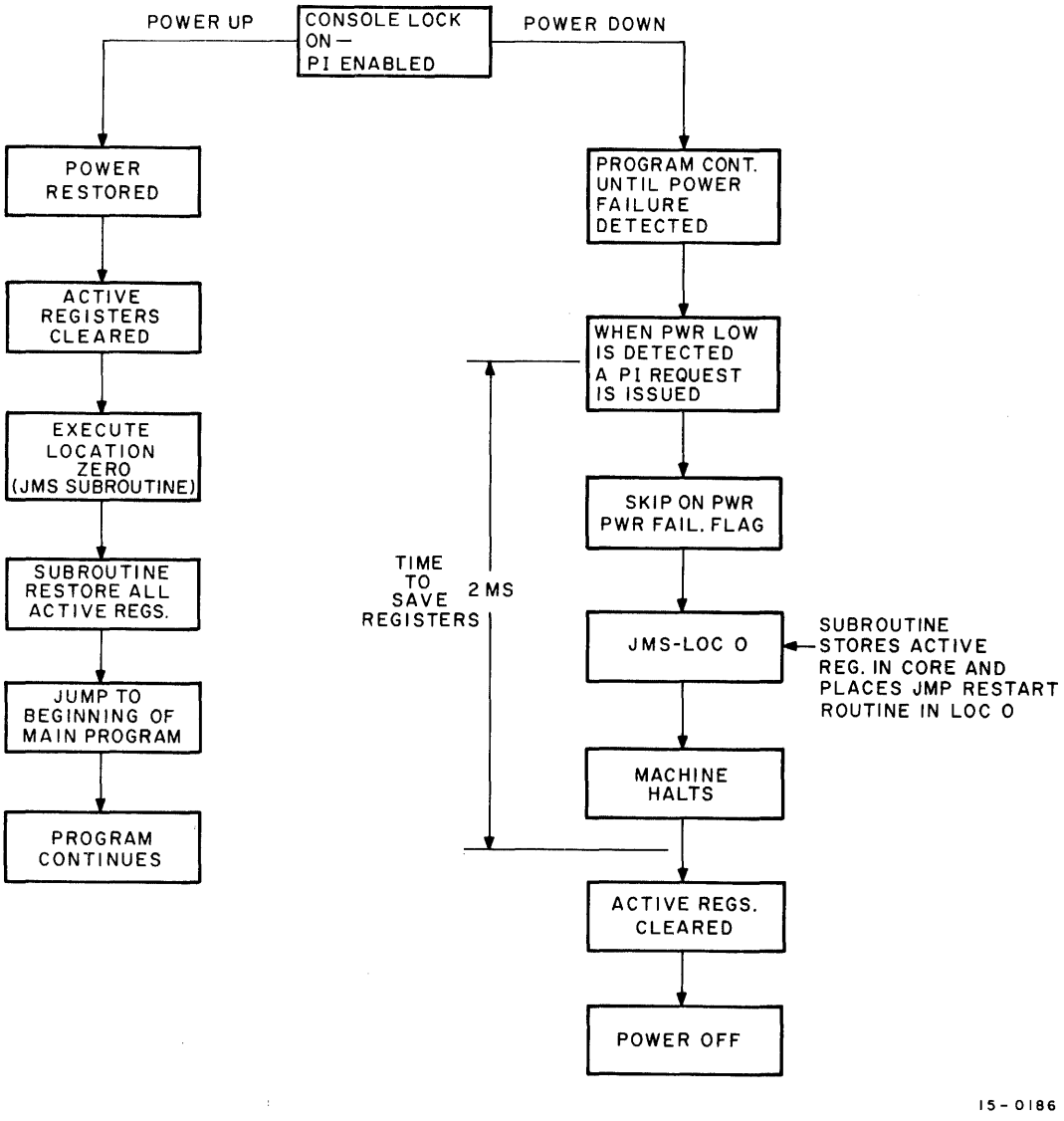


Figure 6-8 Power Fail Up/Down Sequence

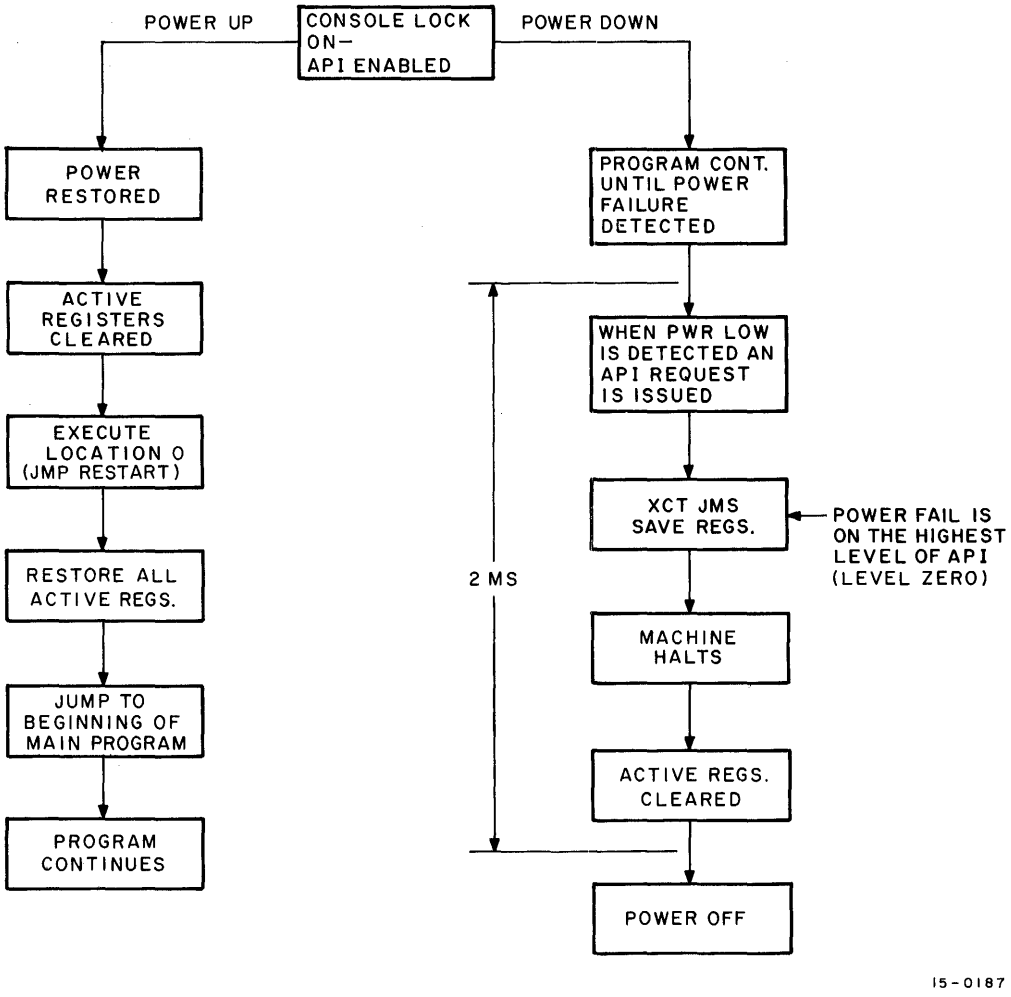


Figure 6-9 Power Fail Up/Down Sequence

6.7 KA15 AUTOMATIC PRIORITY INTERRUPT (API)

The API option extends the PDP-15 capabilities by providing priority servicing for as many as 28 I/O devices, with minimum programming and maximum efficiency. The API priority structure enables high data rate devices to interrupt the service routines of slower devices with a minimum of system "overhead." With the API option, the device service routines can enter directly from hardware-generated entry points, eliminating the need for time-consuming flag searches to identify the device that is causing the interrupt.

The API option gives the PDP-15 System 32 unique channels, or entry points, for the device service routines, and 8 levels of priority. The four higher levels are for fast access to service routines in response to device-initiated service requests. Each of these levels can be multiplexed to handle up to eight devices, assigned an equal priority level. The four lower levels are assigned to program-initiated software routines for transferring control to programs or subroutines on a priority basis. Four of the 32 channels are reserved for these software levels.

Each device interfaced to the API option specifies (sends) its unique service routine entry point to the processor when granted an API break by the processor. Core memory locations 40_8 through 77_8 are assigned as these entry points, in PDP-15 System Software. JMS or JMS * instructions contained in these locations provide linkage to the actual service routines.

Of the 28 hardware channels, three are assigned internally to the optional real-time clock, optional power failure detection system, and optional memory parity. The API interface logic for these devices is contained in the BB15 option panel.

Each software level services one interrupt and uses a single address (locations 40_8 to 43_8). The software requests are initiated by a program issuing an ISA instruction with the appropriate AC bits set (refer to Table 6-9).

The I/O interrupts permit the asynchronous operation of many devices, each at its proper priority level. The software priority levels are used to establish a priority queue for the processing of real-time data without inhibiting the hardware interrupts to service devices.

Each hardware API priority takes precedence over lower API priorities, program interrupts, and the main program. The program segment of highest priority interrupts lower priority program segments when activated. The DCH and RTC are above all these in priority.

6.7.1 API Hardware

Figure 6-10 relates the activity of the automatic priority interrupt system from the initiation and acceptance of the request, to the servicing of the accepted request, and the debreak from the serviced priority level.

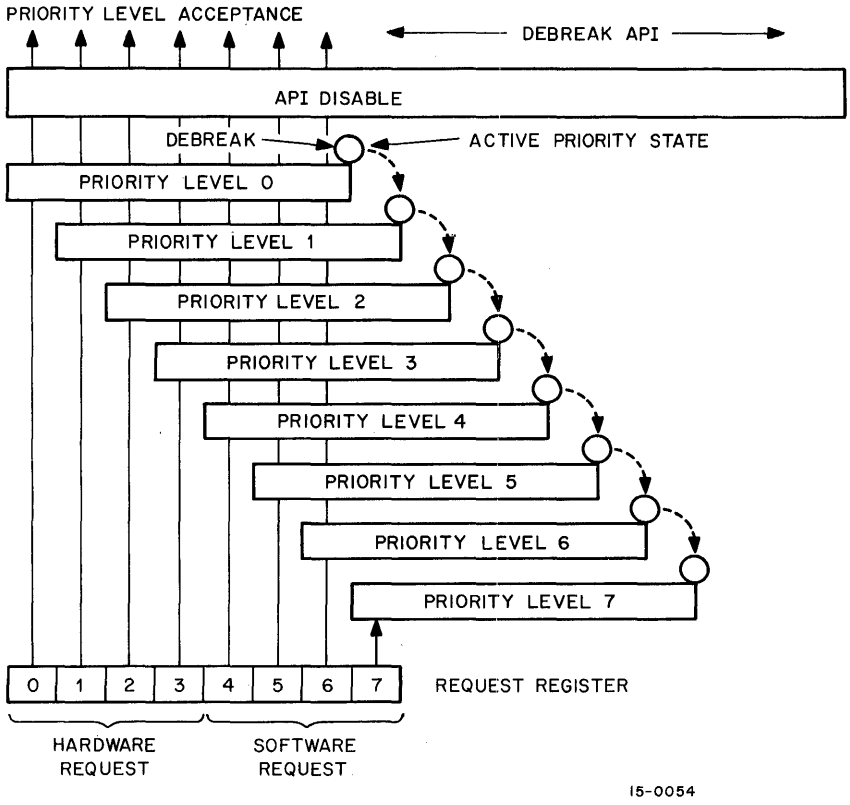


Figure 6-10 API System Simplified Block Diagram

The request register contains eight levels; four levels are activated by the devices (hardware) on the I/O bus, and four are activated under software supervision. The hardware requests are assigned the highest priority and are demonstrated as requests 0, 1, 2, and 3. The software requests are initialed by requests 4, 5, 6, and 7.

The priority level (PL) bars depict the priority level selected by the ISA instruction, or raised by the API control when it has granted a request on that specific level. The PL bars indicate that any request equal to, or less than (in priority) the priority level selected, will not be accepted. At the end of the subroutine currently being performed by an active request, a debreak and restore instruction is

issued to lower the priority to the next selected priority level. The ball, representing the priority debreaking, will fall as long as there is no bar present (i.e., no priority level set). If a lower priority level is set, the debreaking ceases at that level.

The API request register (RR) buffers inputs from the hardware interrupt on levels 0 through 3 and the inputs from the monitors on levels 4 through 7. Up to eight interrupts can be attached to a single level. If two or more of these make simultaneous interrupt requests, the interrupt closest to the processor on the I/O bus is given priority. An interrupt request sets a bit in the RR according to its preassigned priority level. When the scanner detects that bit, the API system signals the CPU to stop execution at the completion of its current instruction. The API system then gates the I/O processor's 15 address lines, which contain the address of the interrupt's unique core location, into the CPU memory output register. The CPU then requests a memory cycle and executes the instruction it fetched from that location. During this operation, the program counter remains unchanged. The API system also sets a bit in the PL corresponding to the level of the interrupt. This prevents interrupts on the same level or lower levels from interrupting the current interrupt. The scanner continues to sample the higher levels so that higher priority devices can interrupt lower priority devices. At the completion of the interrupt subroutine, a debreak and restore (DBR) instruction must be issued to reset the bit in the PL and in the RR.

The API hardware ensures that simultaneous requests by multiple devices are handled in the proper priority sequence. If interrupt requests occur at different priority levels, the highest priority requests are serviced first. Higher priority devices can interrupt lower priority devices. The entire API system can be enabled or disabled with a single instruction; however, most devices provide facilities to connect and disconnect their flags from the interrupt separately. If the API system is disabled, the device automatically signals the program interrupt to obtain a response at that priority level.

6.7.2 API Instructions

The API logic adds six IOT instructions to the basic PDP-15 repertoire. Table 6-7 briefly describes these instructions, and programming considerations for their use follow.

Table 6-7
API IOT Instructions

Mnemonic	Octal Code	Description
DBK	703304	Debreak. Releases the highest currently active priority level.
DBR	703344	Debreak and restore. Releases the highest currently active priority level and provides for restoration of the LINK, Bank Mode, and User Mode status to the interrupt program at the next indirect reference (typically JMP *).
RES	707742	Restore. Provides for restoration of the LINK, Bank Mode, and User Mode status to the interrupted program.
SPI	705501	Skip on priorities inactive. Tests for the successful raising of a ISA-initiated priority level.
RPL	705512	Reads API status bits from API logic into the AC.
ISA	705504	Initiate selected activity. Requests service at a software priority level or raises the currently active priority to a higher level. Also, enables or disables the API system.

6.7.3 Programming Considerations

DBK Instruction (703304)

The DBK instruction is used in a currently active API service routine to return the routine to its normally assigned priority level, after the need for its temporary raising (by ISA or CAL) has been satisfied. DBK is not normally used to terminate an API or Program Interrupt service routine because it does not enable the PDP-15 to restore LINK, Bank Mode, and User Mode status to the interrupted program.

DBR Instruction (703344)

The DBR instruction also returns the currently active API routine to its normally assigned priority level. Additionally, it enables the PDP-15 System to restore the LINK, Bank Mode, and User Mode to the status they occupied at the time of interrupt. The status of these modes is stored in core memory by JMS; the interrupt program count is also stored in core memory when the API service routine is entered. Normally the next to the last instruction in the service routine, DBR is followed by a JMP * to the interrupted program, which performs the actual restoration of the program count and the status

information. As for all IOT instructions, another interrupt cannot occur until execution of the subsequent instruction, i.e., JMP *, is completed. Restoration actually occurs at the first indirect instruction after the DBR.

RES Instruction (707742)

The RES instruction restores the status of the LINK, Bank Mode, and User Mode, at the first indirect instruction after it is executed. It does not, however, affect the API priority levels.

SPI Instruction (705501)

The SPI instruction tests for the successful ISA-initiated raising of a priority and uses a control word previously placed in the AC (by LAC) to test the priority level of the currently active API service routine. In the API logic, the control bits are compared with corresponding API status conditions. The program skips the next instruction if any corresponding API conditions for the set control bit are true (refer to Table 6-8).

Table 6-8
SPI Control Word Format

AC Bit	API Condition Tested
00	API ENABLE (1)
01-09	Not Used
10	Priority level 0 inactive (highest)
11	Priority level 1 and higher inactive
12	Priority level 2 and higher inactive
13	Priority level 3 and higher inactive
14	Priority level 4 and higher inactive (software)
15	Priority level 5 and higher inactive (software)
16	Priority level 6 and higher inactive (software)
17	Priority level 7 and higher inactive (software)

ISA Instruction (705504)

The ISA instruction controls the status of API priorities. It initiates the activity specified by a control word placed in the AC by a previous LAC instruction. Table 6-9 shows the control word format.

Within lower priority service routines, it may be necessary to raise the service routine's priority level in order for it to continue without interruption by any higher priority API request; for example, this

may be necessary because of some calculation within the service routine. By issuing the ISA instruction (with the proper bit set in the AC), the priority of the service routine is raised, and no instruction in a channel address is executed. The service routine continues at the higher priority level. Thus, the two priority levels are currently active to restore the routine to its original priority level; a DBK releases the highest currently active priority level. ISA instructions cannot be used to lower the priority of a currently active service routine because the logic does not recognize the request.

Table 6-9
ISA Control Word Format

AC Bit	Activity Specified
00	Enable API (disable if 0)
01	Not Used
02	Test Request level 0 (Maintenance Only)
03	Test Request level 1 (Maintenance Only)
04	Test Request level 2 (Maintenance Only)
05	Test Request level 3 (Maintenance Only)
06	Request service at priority level 4 (software)
07	Request service at priority level 5 (software)
08	Request service at priority level 6 (software)
09	Request service at priority level 7 (software)
10	Raise priority to level 0
11	Raise priority to level 1
12	Raise priority to level 2
13	Raise priority to level 3
14	Raise priority to level 4
15	Raise priority to level 5
16	Raise priority to level 6
17	Raise priority to level 7

In addition to its normal function, the ISA instruction is also used to test API hardware levels in the API test program.

Because a PDP-15 and API option can be obtained with no hardware devices on levels 0 through 3, bits 2 through 5 of the AC, when executing a ISA, are used to set test requests which check out the

operation of the API. Therefore, under normal program operations, if any of these bits are set, a break occurs from one of the test requests and results in an error.

RPL Instruction (705512)

The RPL instruction is used to read API status bits (refer to Table 6-10) from the API logic into the AC through the Input Mixer.

Table 6-10
Maintenance Instruction Status Word

Status Bit	Status Of	Status Bit	Status Of
00	API ENABLE	09	API 7 RQ
01	Not used	10	PL0
02	API 0 RQ	11	PL1
03	API 1 RQ	12	PL2
04	API 2 RQ	13	PL3
05	API 3 RQ	14	PL4
06	API 4 RQ	15	PL5
07	API 5 RQ	16	PL6
08	API 6 RQ	17	PL7

CAL Instruction with API

The CAL instruction can be used in conjunction with the software API levels and, when executed, raises priority level 4, provided no hardware levels are set. This function can be used to prevent other software levels from interrupting the level currently active. No break, other than the actual CAL instruction, occurs at this time. CAL must not be used when servicing hardware level interrupts.

Program Interrupt with API

Whenever a program interrupt occurs in the PDP-15, priority level 3 is raised, giving the program interrupt a priority between the hardware levels and the software levels. Program interrupts can occur while software priority levels are set, but do not occur when hardware levels or requests are enabled. A DBR instruction in the interrupt service routine is used to release the system from priority level 3.

Dynamic Priority Re-allocation

Three distinct methods for dynamic priority re-allocation are described below.

a. Device-Dependent - Because channel number and priority level are independent, a device can be designed to interrupt at any one of several priority levels without grossly affecting programming. In a control application, the device raises its priority under program control when the data rate increases.

b. Program-Generated Service Requests - The program can generate interrupt requests on any of four software priority levels. If the level is below the currently active priority, the request is honored when the higher priority levels are released. If the level is higher than the currently active level, the request is honored immediately. The JMS instruction in the software priority channel is executed, storing the current program count and entering the new program segment.

c. Programmed Priority Changes - For an interruptable program to change parameters in an interrupt service subroutine, the priority interrupt system is turned off while the changes are effected. Unfortunately, all interrupts are shut out during this time, including those that indicate machine errors or are vital in controlling real-time processes. Thus, the API has been designed to enable a program segment to raise its priority only high enough to shut out those devices whose service routines require changes. This method of raising and lowering priority requires the least amount of time. By issuing the ISA instruction with the proper bits set in the accumulator, the priority of the currently active program segment is raised. No instruction in a channel is executed, and the program continues on at its higher priority level. To restore the program segment to its original priority level, a DBK instruction is issued.

   For example, a priority 2 routine is entering data in memory locations A through A + 10; however, based on a calculation made by a priority 6 routine, it becomes necessary to move the data to memory locations B through B + 20. The changes in the routine at level 2 must be completed, without interruption, once begun. It is possible to complete the changes having the level 6 program raise itself to level 2 (devices on the same or lower priority may not interrupt), complete the changes, and debreak back to level 6.

6.7.4 Programming Examples

Input Ten Words from A/D Converter - A service routine INAD inputs 10 words to a FORTRAN array for later processing. The core location of the A/D channel contains a JMS INAD. The basic components of INAD are:

```
INAD    0              /ENTRY POINT
        DAC SAVAC      /SAVE AC
        IOT            /READ A/D BUFFER
          .            /STORE IN ARRAY
          .            /TEST FOR LAST WORD-IF YES, INITIATE
          .            /SOFTWARE INTERRUPT TO ACCESS DATA
          .            /FORMATTING ROUTINE
        IOT            /ELSE, START NEXT CONVERSION
        LAC SAVAC      /RESTORE AC
```

```
IOT             /CLEAR DEVICE FLAG
DBR             /DEBREAK AND RESTORE
JMP * INAD      /RETURN
```

The program segment to start the conversion is as follows:

```
 .              /INITIALIZE INAD
IOT             /SELECT CONVERTER FOR FIRST CONVERSION
 .              /CONTINUE WITH PROGRAM
```

If INAD were active, it could be instructed to input an additional 10 words with the following segment:

```
LAC ( )         /CONTROL WORD
ISA             /RAISE PRIORITY TO
 .              /LOCK OUT INAD
 .              /CHANGE INAD PARAMETERS
DBK             /RESTORE PRIORITY TO ORIGINAL LEVEL
```

Simulation of Hardware Interrupt - A hardware interrupt can be simulated by:

```
LAC ( )         /CONTROL WORD
ISA             /RAISE TO HARDWARE PRIORITY
JMS INAD        /ENTER INAD
```

Use of Software Levels - An organizational example of a program using five levels is as follows:

Interrupt level 0	Highest priority alarm conditions, computer or processor malfunctions.
Interrupt level 1	Control process A/D-D/A, sense and control input/output routines.
Interrupt level 2	Teletype I/O routines for operator interface, operator can query or demand changes as required. Program interrupt.
Interrupt level 3	FORTRAN subroutines to calculate process control input/output data. Direct digital control routines.
Main Program	Lowest Priority, operator interface programming, requested readout, etc.

Queueing - High priority/high data rate/short access routines cannot perform complex calculations based on unusual conditions without holding off further data inputs. To perform the calculations, the high priority program segment must initiate a lower priority (interruptable) segment to perform the calculation. Because in general, many data handling routines are requesting calculations, there is a queue of calculation jobs waiting to be performed at the software level. Each data handling routine must add its job to the appropriate queue and issue an interrupt request (ISA instruction) at the corresponding software priority level.

6.8 FP15 FLOATING-POINT PROCESSOR

The FP15 Floating-Point Processor performs single- and double-precision floating-point arithmetic, and integer arithmetic operations. The FP15 is a hardware option for PDP-15/20, -15/30, -15/35, and -15/40 systems that can perform arithmetic operations ten times faster than existing software routines. It features 9-digit precision arithmetic on numbers within the $10^{-131,072}$ to $10^{131,071}$ range.

The FP15 is a complete processor, with its unique instruction set, that interfaces directly with up to 128K of core memory. It monitors every instruction fetched by the KP15 central processor. When it recognizes a floating-point instruction, the FP15 inhibits the KP15 and begins the specified function.

Basically, FP15 operations consist of memory transfers to obtain and store data and arithmetic operations within the FP15. Memory cycle time and I/O processor operations, as well as data channel latency, are not affected by the FP15 option. Thus, block transfers to and from memory via the I/O processor may occur simultaneously with FP15 operations. However, program and priority interrupts are inhibited.

Because the FP15 Floating-Point Processor executes a set of more than 100 instructions, the complete description of the purpose and use of this option is provided in a separate manual--FP15 Floating-Point Processor Programmers Reference Manual, DEC-15-HQEB-D.

Appendix A
Instruction Summary

Table A-1
Memory Reference Instructions

Mnemonic Symbol	Octal Code	Machine Cycles	Operation Executed
CAL Y	00	2	Call subroutine. The address portion of this instruction is ignored. The action is identical to JMS 20.
DAC Y	04	2	Deposit AC. The content of the AC is deposited in the memory cell of location Y.
JMS Y	10	2	Jump to subroutine. The content of the PC and the content of the L are deposited in memory cell Y. The next instruction is taken from cell Y + 1.
DZM Y	14	2	Deposit zero in memory. Zero is deposited in memory cell Y.
LAC Y	20	2	Load AC. The content of Y is loaded into the AC.
XOR Y	24	2	Exclusive OR. The exclusive OR is performed between the content of Y and the content of the AC, with the result left in the AC.
ADD Y	30	2.3	Add (1's complement). The content of Y is added to the content of the AC in 1's complement arithmetic and the result is left in the AC.
TAD Y	34	2	2's complement add. The content of Y is added to the content of the AC in 2's complement arithmetic and the result is left in the AC.
XCT Y	40	1+	Execute. The instruction in memory cell Y is executed.

Table A-1 (Cont)
Memory Reference Instructions

Mnemonic Symbol	Octal Code	Machine Cycles	Operation Executed
ISZ Y	44	3	Increment and skip if zero. The content of Y is incremented by one in 2's complement arithmetic. If the result is zero, the next instruction is skipped.
AND Y	50	2	AND. The logical operation AND is performed between the content of Y and the content of the AC with the result left in the AC.
SAD Y	54	2	Skip if AC is different from Y. The content of Y is compared with the content of the AC. If the numbers are different, the next instruction is skipped.
JMP Y	60	1	Jump to Y. The next instruction to be executed is taken from memory cell Y.

Table A-2
Operate Instructions

Mnemonic Symbol	Octal Code	Operation Executed
OPR or NOP	740000	Operate group or no operation. Causes a single-cycle program delay.
CMA	740001	Complement accumulator. Each bit of the AC is complemented.
CML	740002	Complement link.
OAS	740004	Inclusive OR ACCUMULATOR switches. The word set into the ACCUMULATOR switches is OR combined with the content of the AC, the result remains in the AC.
RAL	740010	Rotate accumulator left. The content of the AC and L are rotated one position to the left.
RAR	740020	Rotate accumulator right. The content of the AC and L are rotated one position to the right.
IAC	740030	Increment the accumulator.
TCA	740031	2's complement AC.

Table A-2 (Cont)
Operate Instructions

Mnemonic Symbol	Octal Code	Operation Executed
HLT	740040	Halt. The program is stopped at the conclusion of the cycle.
SMA	740100	Skip on minus accumulator. If the content of the AC is negative (2's complement) number the next instruction is skipped.
SZA	740200	Skip on zero accumulator. If the content of the AC equals zero (2's complement), the next instruction is skipped.
SNL	740400	Skip on nonzero link. If the L contains a 1, the next instruction is skipped.
SKP	741000	Skip. The next instruction is unconditionally skipped.
SPA	741100	Skip on positive accumulator. If the content of the AC is zero (2's complement) or a positive number, the next instruction is skipped.
SNA	741200	Skip on nonzero accumulator. If the content of the AC is not zero (2's complement), the next instruction is skipped.
SZL	741400	Skip on zero link. If the L contains a 0, the next instruction is skipped.
RTL	742010	Rotate two left. The content of the AC and L are rotated two positions to the left.
RTR	742020	Rotate two right. The content of the AC and L are rotated two positions to the right.
SWHA	742030	Swap halves of the AC.
CLL	744000	Clear link. The L is cleared.
STL	744002	Set link. The L is set to 1.
RCL	744010	Clear link, then rotate left. The L is cleared, then the L and AC are rotated one position left.
RCR	744020	Clear link, then rotate right. The L is cleared, then the L and AC are rotated one position right.
CLA	750000	Clear accumulator. Each bit of the AC is cleared.
CLC	750001	Clear and complement accumulator. Each bit of the AC is set to contain a 1.
LAS	750004	Load accumulator from switches. The word set into the ACCUMULATOR switches is loaded in to the AC.

Table A-2 (Cont)
Operate Instructions

Mnemonic Symbol	Octal Code	Operation Executed
GLK	750010	Get link. The content of L is set into AC17.
LAW N	76XXXX	Load the AC with 76XXXX.

Table A-3
Index Register Transfer Instructions

Mnemonic Symbol	Octal Code	Memory Cycle	Operation Executed
PAX	721000	1	Place accumulator in index register.
PAL	722000	1	Place accumulator in limit register.
PXA	724000	1 (1)	Place index register in accumulator.
PXL	726000	1	Place index register in limit register.
PLA	730000	1	Place limit register in accumulator.
PLX	731000	1	Place limit register in index register.

Table A-4
Register Control Instructions

Mnemonic Symbol	Octal Code	Memory Cycle	Operation Executed
AXS n	725 + n	1*	Add n to index register and skip if $\geq$ limit register.
AXR n	737 + n	1	Add n to index register.
AAC n	723 + n	1	Add n to accumulator.
CLX	735000	1	Clear index register.
CLLR	736000	1	Clear limit register.

*For these twelve instructions, although only one memory cycle is required, the CPU requires another cycle to complete the operation.

Table A-5
EAE Instructions

Mnemonic Symbol	Octal Code	Execute Time (μs)	Operation Executed
EAE	640000	1.325	Basic EAE Command
LRS	640500	2.915 + .13h*	Long right shift
LRSS	660500	2.915 + .13h*	Long right shift, signed
LLS	640600	2.915 + .13h*	Long left shift
LLSS	660600	2.915 + .13h*	Long left shift, signed
ALS	640700	2.915 + .13h*	Accumulator left shift
ALSS	660700	2.915 + .13h*	Accumulator left shift, signed
NORM	640444	2.915 + .13h*	Normalize, unsigned
NORMS	660444	2.915 + .13h*	Normalize, signed
MUL	653122	2.915 + .26L***	Multiply, unsigned
MULS	657122	2.915 + .26L***	Multiply, signed
DIV	640323	2.915 + .26m**	Divide, unsigned
DIVS	644323	2.915 + .26m**	Divide, signed
IDIV	653323	2.915 + .26m**	Integer divide, unsigned
IDIVS	657323	2.915 + .26m**	Integer divide, signed
FRDIV	650323	2.915 + .26m**	Fraction divide, unsigned
FRDIVS	654323	2.915 + .26m**	Fraction divide, signed
LACQ	641002	1.325	Load AC with MQ
LACS	641001	1.325	Load AC with SC
CLQ	650000	1.325	Clear MQ
ABS	644000	1.325	Load AC with AC
GSM	664000	1.325	Get sign and magnitude
OSC	640001	1.325	OR SC to AC
OMQ	640002	1.325	OR MQ to AC
CMQ	640004	1.325	Complement MQ
LMQ	652000	1.325	Load MQ from AC

*Where "h" is the number of steps, the instruction must carry out $0 \leq n \leq 36$

**Where "m" is the number of steps, a divide instruction carries out $0 \leq m \leq 19$

***Where "L" is the number of steps, a multiply instruction carries out $0 \leq L \leq 18$

Table A-6
Standard API Channel/Priority Assignments

Channel	Device	Option Number	Priority	Address
0	Software Priority	------	4	40
1	Software Priority	------	5	41
2	Software Priority	------	6	42
3	Software Priority	------	7	43
4	DECtape	TC02 or TC15	1	44
5	Magtape	TC59	1	45
6	Not assigned		1	46
7	Not assigned		1	47
8	Paper Tape Reader	PC15	2	50
9	Clock Overflow	KW15	3	51
10	Power Fail	KF15	0	52
11	Parity	MP15	0	53
12	Display (Lightpen Flag)	VP15	2	54
13	Card Readers	CR03B	2	55
14	Line Printer	LP15 C/F	2	56
15	A/D	AD15	0	57
16	DB99A/DB98A	DB09A	3	60
17	Not assigned		3	61
18	Dataphone	DP09A	2	62
19	Disk	RF15	1	63
20	Disk	RP15	1	64
21	Plotter	XY15	2	65
24	Not assigned			70
25	Not assigned			71
26	Not assigned			72
27	Not assigned		3	73
28	Teletype Keyboard	LT19/LT15A	3	74
29	Teletype Printer	LT19/LT15A	3	75
30	DECtape (DCH Channel 36)	TC02 or TC15*	1	76
31	Dataphone	DP09*	2	77

*Channel allocated for systems with more than one of the above options.

Table A-7
PDP-15 IOT Device Selection Codes

00 1 RT Clock 2 Prog Interrupt 4 RT Clock	10 AFC-15 UDC-15	20 AFC-15 UDC-15	30 VT15 Graphic Processor	40 LT19 Line 1, 2, 3, 4 Teleprinter or LT15A	50	60	70 DECdisk RF15
01 PC15 High Speed Paper Tape Reader	11 Analog-to-Digital or Digital-to-Analog Converter	21 Relay Buffer DR09A	31 VT15 Graphic Processor	41 Line 1,2,3,4 Keyboard or LT15A	51 AA01	61	71
02 PC15 High Speed Paper Tape Punch	12 A/D or D/A Converter	22 IPB DB09A	32 SD0 - KF15 SD1-3 - VT09 Display Option	42 Line 5,6,7,8 Teleprinter	52	62	72 DECdisk RF15
03 1 Keyboard 2 Keyboard 4 IORS	13 A/D Converter	23	33 1 33 KSR Skip 2 Clear All Flags 4 DBR, DBK	43 Line 5,6,7,8 Keyboard	53	63 Disk Pack RP09/RP15	73 Magnetic Tape Control TC59
04 Teleprinter	14 AM03 & AM09 SD0,1 SYS I SD2,3 SYS II	24 Incremental Plotter Control XY15	34	44 Line 9,10,11,12 Teleprinter DC01EB #102	54	64 Disk RP09/RP15	74 Magnetic Tape Control TC59
05 VP15A, B, BL C, CL	15	25 DP09A Data Communication	35	45 Line 9,10,11,12 Keyboard DC01EB #304	55 Automatic Priority Interrupt KA15	65 Line Printer LP15 C/F	75 DECtape Control TC02/TC15
06 VP15A B BL CL	16	26 DP09A Data Communication	36	46 Line 13,14,15,16 Teleprinter	56	66 Line Printer LP15 C/F	76 DECtape Control TC02/TC15
07 VP15A, B, BL, CL	17 Memory Protect and Relocate KM15 KT15	27 Memory Parity MP15	37	47 Line 13,14,15,16 Keyboard	57	67 Card Reader Type CR03B	77 61 Skip on Bank Mode 62 Disable Bank Mode 64 Enable Bank Mode

Table A-8
Input/Output Transfer Instructions

Mnemonic Symbol	Octal Code	Operation Executed
		Program Interrupt
IOF	700002	Interrupt off. Disable the PIC.
ION	700042	Interrupt on. Enable the PIC.
		KW15 Real-Time Clock
CLSF	700001	Skip the next instruction, if the CLOCK flag is set to 1.
CLOF	700004	Clear the CLOCK flag and disable the clock.
CLON	700044	Clear the CLOCK flag and enable the clock.
		PC15 High Speed Paper Tape Reader
RSF	700101	Skip, if READER flag is a 1.
RCF	700102	Clear READER flag, then inclusively OR the contents of the reader buffer into the AC.
RRB	700112	Read reader buffer. Clear READER flag and AC, and then transfer content of reader buffer into AC.
RSA	700104	Select reader in alphanumeric mode. One 8-bit character is read into the reader buffer.
RSB	700144	Select reader in binary mode. Three 6-bit characters are read into the reader buffer.
		PC15 High Speed Paper Tape Punch
PSF	700201	Skip, if the PUNCH flag is set to 1.
PCF	700202	Clear the PUNCH flag.
PSA or PLS	700204 700206	Punch a line of tape in alphanumeric mode.
PSB	700244	Punch a line of tape in binary mode.
		I/O Equipment
IORS	700314	Input/output read status. The content of given flags replaces the content of the assigned AC bits.
TTS	703301	Test Teletype, and skip if 33 KSR Teletype is connected to computer.
CAF	703302	Clear all flags.
SPCO	703341	Skip, if a PC15 is connected to the system.
SK15	707741	Skip, if processor is a PDP-15.

Table A-8 (Cont)
Input/Output Transfer Instructions

Mnemonic Symbol	Octal Code	Operation Executed
		I/O Equipment (Cont)
SBA	707761	Skip, if processor is in Bank Mode.
DBA	707762	Disable Bank Addressing (enter Page Mode).
EBA	707764	Enable Bank Addressing
		Teletype Keyboard
KSF	700301	Skip, if the KEYBOARD flag is set to 1.
KRB	700312	Read the keyboard buffer. The content of the buffer is placed in AC10-17 and the KEYBOARD flag is cleared (half-duplex operation).
KRS	700332	Read keyboard buffer and select keyboard reader (full-duplex operation).
		Teletype Teleprinter
TSF	700401	Skip, if the TELEPRINTER flag is set.
TCF	700402	Clear the TELEPRINTER flag.
TLS	700406	Load teleprinter buffer. The content of AC10-17 is placed in the buffer and printed. The flag is cleared before transmission takes place and is set when the character has been printed.
		VP15A Storage Tube Display
CXB	700502	Clear X-coordinate buffer
CYB	700602	Clear Y-coordinate buffer
LXB	700504	Load X-coordinate buffer from AC8-17
LYB	700604	Load Y-coordinate buffer from AC8-17
EST	700724	Erase storage tube
SDDF	700521	Skip on DISPLAY DONE flag.
CDDF	700722	Clear DISPLAY DONE flag.
LXBD	700564	Load X-coordinate buffer and display the point specified by XB and YB (store mode).
LYBD	700664	Load Y-coordinate buffer and display the point specified by XB and YB (store mode).
LXDNS	700544	Load the X-coordinate buffer and display the point specified by XB and YB (nonstore mode).

Table A-8 (Cont)
Input/Output Transfer Instructions

Mnemonic Symbol	Octal Code	Operation Executed
		VP15A Storage Tube Display (Cont)
LYDNS	700644	Load the Y-coordinate buffer and display the point specified by XB and YB (nonstore mode).
		VP15B (RM503), BL (RM503 and Light Pen), C (VR12) and CL (VR12 and Light Pen)
DXL	700504	Load the X-coordinate buffer from AC8-17
DXS	700544	Load the X-coordinate buffer and display the point specified by the XB and YB.
DYL	700604	Load the Y-coordinate buffer from AC8-17.
DYS	700644	Load the Y-coordinate buffer and display the point specified by the XB and YB.
DXC	700502	Clear the X-coordinate buffer.
DYC	700602	Clear the Y-coordinate buffer.
DLB	700704	Load the brightness register from bits 16-17 of the AC. This instruction clears the display flag associated with the light pen.
DSF	700501	Skip, if DISPLAY (light pen) flag is a 1.
DCF	700702	Clear DISPLAY (light pen) flag.
		VP15M Storage Tube Display Multiplexer
LUDU	700764	Load unit designation register from AC 10-17.
		KM15 Memory Protect
MPSK	701701	Skip on PROTECT VIOLATION flag.
MPCV	701702	Clear PROTECT VIOLATION flag.
MPLD	701704	Load core allocation register.
MPLR	701724	Load relocation register.
MPSNE	701741	Skip on nonexistent MEMORY flag.
MPEU	701742	Enter User Mode.
MPCNE	701744	Clear nonexistent MEMORY flag.
		KT15 Memory Relocate
MPSK	701701	Skip on PROTECT VIOLATION flag.
MPCV	701702	Clear PROTECT VIOLATION flag.

Table A-8 (Cont)
Input/Output Transfer Instructions

Mnemonic Symbol	Octal Code	Operation Executed
		KT15 Memory Relocate (Cont)
MPLD	701704	Load the boundary register with the contents of AC 01-09.
MPSNE	701741	Skip on nonexistent MEMORY flag.
MPEU	701742	Enter user mode.
MPCNE	701744	Clear nonexistent MEMORY flag.
		MP15 Memory Parity
SPE	702701	Skip on PARITY ERROR flag.
CPE	702702	Clear parity error.
FWP	702704	Force wrong parity.
		VT15 Graphic Processor
RS1	703002	Read status 1
RS2	703022	Read status 2
RS3	703142	Read status 3
RYP	703042	Read Y register
RPC	703062	Read program counter
RXP	703102	Read X register
SSA	703122	Single-step advance (Debugging)
SPSF	703001	Skip on STOP flag.
SPLP	703021	Skip on LIGHT PEN flag.
SPPB	703041	Skip on PUSHBUTTON flag.
SPEF	703061	Skip on EDGE flag.
SPDF	703101	Skip on any flag.
SPDI	703121	Skip on any interrupting flag.
SSLP	703141	Skip on SLAVE LIGHT PEN flag (Multiplexer with more than one VT04-374).
SPES	703161	Skip on external stop (Check STPD accomplished).
LSD	703004	Load and start display (Initializes VT15)
SIC	703024	Set initial conditions.
STPD	703044	External stop display (PDP-15 stops display).
RES	703064	Resume display after flag.

Table A-8 (Cont)
Input/Output Transfer Instructions

Mnemonic Symbol	Octal Code	Operation Executed
		KF15 Power Fail Option
SPFAL	703201	Skip, if POWER-LOW flag is set.
		KA15 Automatic Priority Interrupt
DBK	703304	Debreak
DBR	703344	Debreak and restore.
SPI	705501	Skip on priorities inactive.
RPL	705512	Read API status
ISA	705504	Initiate selected activity
ENB	705521	Enable breaks
INH	705522	Disable breaks
RES	707742	Restore
		RP15 Disk Pack Control
DPSF	706301	Skip on DISK flag.
DPOSA	706302	OR the status register A into AC.
DPRSA	706312	Read the status register A into AC.
DPOU	706402	OR the unit cylinder address register into the AC.
DPRU	706412	Read the unit cylinder address register into the AC.
DPSA	706321	Skip on Attention flag.
DPOSB	706322	OR status register B into the AC.
DPRSB	706332	Read status register B into the AC.
DPLZ	706424	Load the accumulator zeros into status register A bits 0 through 7 and execute.
DPLO	706444	Load the accumulator ones into status register A bits 0 through 7 and execute.
DPCN	706454	Execute the function register.
DPLF	706464	Load the status register A and execute.
DPLA	706304	Load the cylinder, head, and sector address registers from the accumulator.
DPCA	706344	Load the current address register.
DPWC	706364	Load the word count register.

Table A-8 (Cont)
Input/Output Transfer Instructions

Mnemonic Symbol	Octal Code	Operation Executed
		RP15 Disk Pack Control (Cont)
DPOA	706422	OR the cylinder, head, and sector address register into the AC. AC bits 13 through 17 are ORed with the sector.
DPRA	706432	Read the cylinder, head, and sector address into the AC.
DPOC	706442	OR the current address register into the AC.
DPRC	706452	Read the current address register into the AC.
DPOW	706462	OR the word count register into the AC.
DPRW	706472	Read the word count register into the AC.
DPCS	706324	Clear status.
DPCF	706404	Clear function register.
		RP15 Maintenance IOTs
DPSJ	706341	Skip, if the JOB DONE flag is set.
DPSE	706361	Skip, if an error condition is present.
DPOM	706342	OR the six-bit maintenance register into AC.
DPRM	706352	Read the six-bit maintenance register into AC.
DPEM	706401	Execute maintenance instruction.
DPLM	706411	Leave maintenance mode. The AC is left cleared.
		LP15C/F Line Printer Controls
LPSF	706501	Causes a skip request, if done or error is set.
LPPM	706521	Initializes the control, sets header; sets multiline.
LPP1	706541	Initializes the control, sets header; does not set multiline.
LPRS	706542	Read status.
LPEI	706544	Sets the ENABLE INTERRUPT flop.
LPDI	706561	Clears the ENABLE INTERRUPT flop.
LPCD	706621	Clears DONE flag.
LPCF	706641	Clears STATUS and ERROR flag.
		Line Printer Maintenance IOTs
MRVFU	706502	Read VFU register.
MCVFU	706504	Clear VFU register.

Table A-8 (Cont)
Input/Output Transfer Instructions

Mnemonic Symbol	Octal Code	Operation Executed
		Line Printer Maintenance IOTs (Cont)
MSM	706524	Set maintenance control.
MRDB1	706562	Read data buffer 00-17.
MCDB	706564	Clear data buffer.
MCM	706601	Clear maintenance control.
MRDB2	706602	Read data buffer 18-35.
MLDB1	706604	Load data buffer 0-17 from AC.
MRM1	706622	Read maintenance word 1.
MLDB2	706624	Load data buffer 18-35 from AC.
MRM2	706642	Read maintenance word 2.
MLS	706644	Load status.
		CR03B Card Reader
CRCS	706704	Clear status register and data buffer.
CRSI	706721	Skip on CARD READER flag.
CROR	706712	Load data buffer and machine status into AC.
CRSC	706722	Clear the status register and data buffer; select a card.
CRLA	706724	Load status and data register from AC.
		RF15 DECdisk Control
DSSF	707001	Skip on DISK flag.
DRBR	707002	OR the contents of the buffer register with the AC.
DLBR	707004	Load the contents of the AC into the buffer register.
DSCC	707021	Clear the disk control and disable the "freeze" status of the control.
DRAL	707022	OR the contents of the address pointer 0 (AP0) into the AC. Bits 0 through 6 contain the track address, and bits 7 through 17 contain the word address of the next word to be transferred.
DRAH	707062	OR the contents of the disk number (AP1) into the AC. Bits 15, 16, and 17 contain the disk number. Bit 14 is read back if a data transfer exceeded the capacity of the disk control. (Causes a NED error status.)

Table A-8 (Cont)
Input/Output Transfer Instructions

Mnemonic Symbol	Octal Code	Operation Executed
		RF15 DECdisk Control (Cont)
DLAL	707024	Load the contents of the AC into the APO.
DLAH	707064	Load the contents of the AC (15, 16, 17) into the disk number (AP1).
DSCF	707041	Clear the function register, interrupt mode.
DSFX	707042	XOR the contents of AC bits 15-17 into the function register (FR).
DSCN	707044	Execute the condition held in the FR.
DLOK	707202	OR the contents of the 11-bit disk segment address (ADS) into the AC.
DGHS	707204	Generate simulated head signals.
DGSS	707224	Generate simulated disk signals.
DSCD	707242	Clear the status register and DISK flag.
DSRS	707262	OR the contents of the disk status register with the AC.
		Type TC59 Magnetic Tape Control IOT Instructions
MTTR	707301	Skip on tape transport ready (TTR).
MTCR	707321	Skip on tape control ready (TCR).
MTSF	707341	Skip on ERROR flag or MAGNETIC TAPE flag (EF and MTF).
MTAF	707322	Clear status and command registers and EF and MTF.
LCM	707324	Inclusively OR content of AC_{0-11} into command register.
MTLC	707326	Load content of AC_{0-11} into command register.
MTCC	707356	Terminate write continuous mode.
	707342	Inclusively OR content of status register into AC_{0-11}.
MTRS	707352	Read content of status register into AC_{0-11}.
MTRC	707312	Read command register into AC_{0-11}.
MTGO	707304	Set "go" bit to execute command in command register.
		TC15 DECtape Control
DTCA	707541	Clear status register A.
DTRA	707552	Read status register A.
DTXA	707544	XOR status register A.

Table A-8 (Cont)
Input/Output Transfer Instructions

Mnemonic Symbol	Octal Code	Operation Executed
		TC15 DECtape Control (Cont)
DTLA	707545	Load status register A.
DTEF	707561	Skip on ERROR flag.
DTRB	707572	Read status B.
DTDF	707601	Skip on DECtape flag.
		AD15 Analog Subsystem
ADCV	701304	Load status register from accumulator, clear A/D done flag, and initiate conversion.
ADRB	701302	Read data buffer into accumulator and clear A/D done flag.
ADRS	701342	Read status register into accumulator.
ADCF	701362	Clear all AD15 flags.
ADSF	701301	Skip on A/D flag.
WCSF	701341	Skip on word count overflow flag.
MSSF	701321	Skip on memory overflow flag.
		UDC-15 Universal Digital Control
UMOD	701001	Set UDC mode
USINT*	702002	Interrupt Select
ULA*	702024	Load address.
URA*	702012	Read deferred address.
URD*	702032	Read data in.
USCAN*	702021	Start interrupt scan.
USNB*	702041	Skip if not busy.
URCG	701072	Clear AC, read COS gates.
ULD	701064	Load data out.
ULPS	701044	Load previous status.
USI	701041	Skip on immediate flag.
USD	701061	Skip on deferred flag.
URAA	701052	Read immediate address.

*The UMOD IOT must be issued before these IOTs will be decoded as UDC-15 IOT instructions.

Table A-8 (Cont)
Input/Output Transfer Instructions

Mnemonic Symbol	Octal Code	Operation Executed
		AFC-15 Automatic Flying Capacitor
FCMOD	701021	Set AFC mode.
FCEI*	702004	Enable AFC interrupt.
FCDI*	702001	Disable AFC interrupt.
FCLAG*	702024	Load address.
FCRB*	702032	Read A/D buffer.
FCSD*	702041	Skip on A/D done flag.
FCRA*	702012	Read AFC address register.
		BD-15 Maintenance
MCLK	702044	Maintenance clock.
MSM	701004	Set maintenance mode.
MCM	701022	Clear maintenance mode.
MLS	701024	Load status register.
MRS	701012	Read status register.
FCCV	702021	A/D convert.

*The FCMOD IOT must be issued before these IOTs will be decoded as AFC-15 IOT instructions.