PDP-15
SYSTEM SOFTWARE
HANDOUTS
The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1975 by Digital Equipment Corporation

The following are trademarks of Digital Equipment Corporation:

<table>
<thead>
<tr>
<th>CDP</th>
<th>DIGITAL</th>
<th>INDAC</th>
<th>PS/8</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPUTER</td>
<td>DNC</td>
<td>KAT0</td>
<td>QUICKPT</td>
</tr>
<tr>
<td>LAB</td>
<td>EDGRIN</td>
<td>LAB-8</td>
<td>RAD-8</td>
</tr>
<tr>
<td>COMSYST</td>
<td>EDUSYSTEM</td>
<td>LAB-8/e</td>
<td>RSTS</td>
</tr>
<tr>
<td>COMTEX</td>
<td>FLIP</td>
<td>LAB-K</td>
<td>RSX</td>
</tr>
<tr>
<td></td>
<td>CHIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDT</td>
<td>FOCAL</td>
<td>OMNIBUS</td>
<td>RTM</td>
</tr>
<tr>
<td>DEC</td>
<td>GLC-8</td>
<td>OS/8</td>
<td>RT-11</td>
</tr>
<tr>
<td>DECCOMM</td>
<td>IDAC</td>
<td>PDP</td>
<td>SABR</td>
</tr>
<tr>
<td>DECTAPE</td>
<td>IDACS</td>
<td>PHA</td>
<td>TYPESET</td>
</tr>
<tr>
<td>DIBOL</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

UNIBUS
PDP-15 SYSTEM SOFTWARE

COURSE ABSTRACT

This course is intended for programmers who wish to acquire a working familiarity with PDP-15 Assembly Language programming and the Disk or Advanced Monitor Operating Systems and the services provided by their monitors and associated system software. A portion of course time is devoted to supervised laboratory sessions.

PREREQUISITES

A working knowledge of the material presented in the Introduction to Minicomputers course.

COURSE OBJECTIVES

Upon successful completion of this course, the student will be able to:

- Write, run and modify assembly language programs using the PDP-15 instruction set and the MACRO assembler syntax.
- Interface programs to the Advanced or DOS I/O Monitor.
- Interact with the system by means of keyboard commands, system program command strings and programmed monitor requests.

COURSE OUTLINE

I. Memory Organization; Memory and Addressing Modes

II. PDP-15 Instruction Set
   A. Memory Reference Instructions
   B. Augmented Instruction Set

III. MACRO Assembler Syntax

IV. Tape and File Formats

V. Interrupt Systems
   A. Program Interrupt Control (PI)
   B. Automatic Priority Interrupt (API)

VI. System Programs
   A. EDIT, MACRO, LINKING LOADER, DDT
   B. PIP, PATCH, SGEN, UPDATE, CHAIN and EXECUTE

VII. I/O
   A. I/O Monitor
   B. System Macros

VIII. FORTRAN and MACRO Interface

IX. Handler Format

X. Interaction with the system via keyboard commands and programmed monitor requests.

COURSE LENGTH

10 days
COURSE OUTLINE

WEEK I

MONDAY A.M.

A. Description of Course
B. Block Diagram of PDP-15
C. Software Overview
D. Memory Organization
   1. Addressing on the PDP-15
   2. Arithmetic on the PDP-15
E. Central Processor Organization
F. Introduction to the Instruction Set

MONDAY P.M.

G. Memory Reference Instructions
H. Operate Instructions
I. EAE Instructions
J. Introduction to MACRO-15

TUESDAY A.M.

A. Review
B. Subroutines
C. Sum Group of Examples
D. Indexed Instructions
E. Absolute vs. Relocatable Programs

TUESDAY P.M.

F. I/O Overview
G. IOT Instructions and Dedicated I/O
H. Paper Tape Formats

WEDNESDAY A.M.

A. Console Description and Operation
B. Operation of Disk Operating System
C. Use of the Editor

WEDNESDAY P.M.

D. LAB
THURSDAY A.M.

A. MACRO-15 (again)
B. Program Interrupt Facility (PI)
C. Automatic Priority Interrupt (API)

THURSDAY P.M.

D. LAB
E. Program and Homework Review
F. Quiz 1

FRIDAY A.M.

A. Quiz 1 Review
B. Overview of DOS Monitor
C. File Formats and Directory Structure
D. Overview of Monitor Supervised I/O

FRIDAY P.M.

E. LAB

WEEK II

MONDAY A.M.

A. Programmed I/O Commands
   1. Basic Operation
   2. System Macros
   3. DAT usage and Linking Loader
   4. User's Buffer Structure

MONDAY P.M.

B. LAB

TUESDAY A.M.

A. Linking Loader
B. Libraries and UPDATE
C. CHAIN and EXECUTE
D. PIP

TUESDAY P.M.

E. LAB
WEDNESDAY A.M.

A. Fortran and Macro Interface
B. DDT
C. ↑Q
D. DUMP

WEDNESDAY P.M.

E. LAB
F. Program and Homework Review
G. Quiz 2

THURSDAY A.M.

A. I/O Handler Format
B. Sample Handler

THURSDAY P.M.

C. LAB
D. Final Exam

FRIDAY A.M.

A. Monitor and System Modification
   1. PATCH
   2. SGEN
B. Unichannel Discussion
C. Final Exam Review

FRIDAY P.M.

D. Optional Lab
Computer: a machine which inputs data from the outside world, processes the data, perhaps puts it in temporary storage and finally outputs the data and/or results in the form of hard copy, displays or commands to other devices it may be controlling.

Program: a sequence of instructions to a computer, specifying the necessary steps to solve a problem (sometimes it includes the data it is to work on).
SOFTWARE

To utilize the powerful PDP-15 hardware a number of operating systems have been developed (special applications packages are also available). See chapter 10 of the System Reference Manual for descriptions of each of these systems.

DOS-15, DISK OPERATING SYSTEM
Disk Operating System (DOS-15) is an integrated set of software designed to meet the demands of research, engineering, and industrial environments. It includes the software necessary for simplified programming and efficient operations. DOS-15 brings to the user the advantage of disk resident storage via rapid access to the system's resources.
The DOS Monitor, the heart of the system, incorporates all the functions of the "Advanced Software System" plus the added power of fully automatic random access file operation. The user controls the operating system by instructions to the Monitor. The Monitor runs the jobs, supervises data and file manipulation, and interacts with the operator/user in a simple conversational manner.

Noteworthy features of DOS-15 are:

Disk Resident System Software
All DOS-15 System Software resides on either DECdisk, or RP15 Disk Pack, or RK15 disk cartridge.

Interactive Operation
An interactive keyboard/program Monitor permits device-independent programming, and automatic calling and loading of system and user programs.

Conversational Mode
System Utility Programs interact with the operator/user in a simple, conversational manner.

Programmed Monitor Commands
Input/Output programming is simplified by the use of a set of system commands which are standardized for system-supported I/O devices.

I/O Device Handlers
Data and file manipulating I/O device handlers are supplied for standard system peripherals, allowing device independence and overlapped computation, and I/O.

User-Created System Files
The user may easily incorporate his own software into the operating system, thereby tailoring the system to his hardware and software needs.

Programming Languages
FORTRAN IV, FOCAL, and MACRO-15 programming languages are offered.

Bank and Page Modes
Choice of 8K (Bank Mode) or 4K (Page Mode) direct addressability. Page Mode operation permits modification via the index register.

Disk File Structure
The disk file structure allows the most efficient use of disk capacity and data retrieval for processing via:

System supported DECdisk, Disk Packs, and Disk Cartridge Devices, providing both economy and storage capacity.

Virtually unlimited data capacity (Disk Pack = 83.7 million words, DECdisk = 2.09 million words, Disk Cartridge = 9.6 million words). Random/Sequential File Access furnishes file protection through unique user directories and associated user identification codes. Files can be made invisible to other users, but with privileged access via a supervisory code.

User/user file independence—identically named unformatted Input/Output (FORTRAN IV).

Random Access-formatted as well as unformatted Input/Output (FORTRAN IV).

Dynamic Storage Allocation
The available disk storage is automatically allocated for optimum storage utilization.

Dynamic Buffer Allocation
Input/Output core is automatically optimized by the Monitor. It allocates only that space which is required for the system and the user.

Batching Operation
An alternative to interactive operation is a batching mode which permits the sequencing of console commands to come from paper tape or cards.

Input/Output Spooling
DOS-15 systems using the RK15/RK05 Unichannel Disk System, provides spooling of card reader, line printer, and XY plotter data.

Spooling is a method of storing (queueing) data to and from slow speed devices on the high speed RK05 disk. This dramatically improves system performance.

Spooling is only provided for devices interfaced to the UNIBUS of the RK15 Disk System (i.e., the CR11, LP11, LS11, and XY11). Spooling requires 8K of local PDP-11 memory.
The following software is available as part of DOS-15:

**Monitors**
- Resident Monitor
- Keyboard Command Decoder
- Batch Processor
- System Loader
- PIREX (Peripheral Processor RK15 Only)

**Languages**
- FORTRAN IV (F4X, FPPF4X)
- FOCAL
- MACRO-11 (Assembler RK15 Only)
- MACRO-15 (Assembler)
- ALGOL (optional)

**Text Editors**
- EDIT
- EDITVP (Storage Scope Editor)
- EDITVT (Graphic Display Editor)

**Loaders**
- Linking Loader
- CHAIN & EXECUTE (Overlay Loaders)
- ABS 11 (RK15 Only)

**Debuggers**
- DDT (Dynamic Debugging Technique)
- DUMP (Core Dump Lister)
- QFILE (Store/Retrieve Core Dumps)

**Utilities (General)**
- DTCOPY (DECTape Copier)
- MTDUMP (Magtape Utility)
- PIP (Peripheral Interchange Program)
- SRCCOM (Source Compare)
- UPDATE (Library File Manager)
- 8TRAN (PDP-8 to PDP-15 Translator)
- 8STRAN (PDP-8 to PDP-9 Translator)
- TKB (RSX-15 Task Builder)

**Utilities (System)**
- DOSSAV (Disk Save/Restore)
- RFBBOOT (DECdisk Bootstrap)
- RPBOOT (Disk Pack Bootstrap)
- RKBOOT (Disk Cartridge Bootstrap)

**I/O Handlers**
- CD JR (Card Reader for CR03B, CR15 or CR11)
- DOSBCD (Batch Card Reader)
- DKA, DKB, DKC, DKL, (RF15/RS09 DEC disk)
- DPA, DPB, DPC, DPL (RP15/RP02 Disk Pack)
- RKA, RKB, RKC, RKL (RK15/RK05 Disk Cartidge)
- DTA, DTC, DTO, DTE, DTF, (DECTape)
- LKA (LK35 Graphics Keyboard)
- LPA (Line Printer for LP15, LS11 or LP1+)
- LVA (Line Printer/Plotter)
- MTA, MTC, MTF (Magtape)
- PPA, PPB, PPC (Paper Tape Punch)
- PRA, PRB (Paper Tape Reader)
- TTA (Teletype)
- VPA (Storage Scope)
- VTA (VT15 Graphic Display)
- VWA (VW01 Writing Tablet)
- XYA (XY11 Plotter, RK DOS only)

**Checkout-Package**
- RF.CHK (DECdisk Checkout)
- RP.CHK (Disk Pack Checkout)
- RK.CHK (Disk Cartridge Checkout)

**Minimum Hardware**
- KP15 Central Processor
- 16,384 18-bit Core Memory
- Console Terminal
- PC15 High Speed Paper Tape Reader and Punch
- KE15 Extended Arithmetic Element
- TC15 DECTape Control—or TC59 Magtape Control
- 1 TU56 Dual DECTape Transport—or 1 TU10, TU20,
- or TU30 (7 or 9 track) Magtape Transport
- RK15 DECdisk Control or RP15 Disk Pack Control or
- RK15 System
- 1 RS09 Disk Drive or 1 RP02 Disk Pack Drive or
- 1 RK05 Drive
Memory: the main storage area for computer instructions and system data.

In order for a program to be executed, it must be placed ("loaded") into memory.

Memory is storage space—a place to keep things for a while. It can hold either data or instructions.

Memory, often referred to as main storage, is much like a large chest of drawers.

a. You can store something in each drawer.

b. You must examine a basic storage unit when looking for something—in the chest this unit is a drawer; in a computer this unit is a location (word).

c. You refer to these basic units by numbers—you tell someone to look for something in the 3rd drawer from the bottom; you tell the processor to look for something in location 3.

In a computer, the numbers of the locations are called addresses. Address numbers begin with zero as shown above.
Each location or word in a PDP-15 is partitioned into smaller subdivisions called bits. Bits are subdivisions which have binary values, either 1 or 0.

Locations subdivided into bits are like drawers partitioned into small slots. To examine a bit, you must first look at the entire location; to look into a slot you must first pull out the entire drawer.

LOCATION

The bits in a location are also numbered beginning with zero, but these numbers are not generally referred to as addresses. So you may imagine memory as a chest of numbered drawers, each containing numbered slots; or you may simplify your model and imagine a matrix of m numbered locations, each containing 18 numbered bits as shown below.
WORD LENGTH

The PDP-15 has an 18 bit word.

The bits are labelled from left to right, starting with 0 and ending with 17:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

most significant bit
"msb"

least significant bit
"lsb"

Because one octal digit is the equivalent of three binary digits, the contents of an 18 bit word is often given as a string of 6 octal digits:

0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

0 1 6 2 5

Binary value: 000111001110010101
Octal value: 071625

SIGNED NUMBERS

Words may be looked at as instructions or as data (numerical values).

When viewed as numerical values, numbers are usually looked at as being SIGNED.

The sign of a number is determined by the most significant bit:

MSB=0, the number is POSITIVE.

If

MSB=1, the number is NEGATIVE.
EXAMPLES

Positive numbers --

010 111 001 011 101 000  OR  271350  
000 001 101 110 010 100  OR  015624  
000 000 000 000 000 001  OR  000001  
000 000 000 000 000 000  OR  000000  

Note that "0" (zero) is considered a positive number because bit 0, its most significant bit, is 0.

Negative numbers --

101 111 001 100 010 001  OR  571421  
111 000 000 010 110 011  OR  700263  
111 111 111 111 111 111  OR  777777  
100 000 000 000 000 000  OR  400000  

Range --

<table>
<thead>
<tr>
<th>POSITIVE</th>
<th>NEGATIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>777777</td>
</tr>
<tr>
<td>000001</td>
<td>777776</td>
</tr>
<tr>
<td>000002</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100000</td>
<td>700000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>200000</td>
<td>600000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>300000</td>
<td>500000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>377776</td>
<td>400002</td>
</tr>
<tr>
<td>377777</td>
<td>400001</td>
</tr>
<tr>
<td></td>
<td>400000</td>
</tr>
</tbody>
</table>
COMPLEMENT NUMBERS

All negative numbers on the PDP-15 are expressed in COMPLEMENT FORM rather than sign-magnitude form.

sign magnitude
+5  000 000 000 000 000 101  OR  000005
-5  100 000 000 000 000 101  OR  400005

There are two forms of complement numbers used on the PDP-15:

1's COMPLEMENT -- the 1's complement of a binary number is the result of inverting each bit position.

ex.

number  010 101 001 000 100 011  OR  251043
1's complement 101 010 110 111 011 100  OR  526734

Note that when you add a number to its 1's complement, the sum is:
111 111 111 111 111 111  OR  777777.

Hence, an easy way to compute the 1's complement of an octal number is to subtract it from 777777.

= 251043
526734

2's COMPLEMENT -- the 2's complement of a binary number is the result of adding "1" to the 1's complement.

i.e. (the 2's complement) = (the 1's complement) + 1.

ex.

number  010 101 001 000 100 011  OR  251043
1's complement 101 010 110 111 011 100  OR  526734
2's complement 101 010 110 111 011 101  OR  526735

Note that an easy way to compute the 2's complement of an octal number is to subtract it from 777778 or a similar value where the 8 is in the rightmost non-zero position of the number being complemented.

251043  777778  BUT  251040  777780
-251043  526735

Note that when you add a number to its 2's complement, the sum is:
000000.

251043  251040
+526735  +526740
000000  000000
SOME CONSEQUENCES --

1) There is no difference between the 1's and 2's complement representation of positive numbers.

2) There is a difference between the 1's and 2's complement representation of negative numbers.

<table>
<thead>
<tr>
<th>Positive Number</th>
<th>1's Complement</th>
<th>2's Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>777777</td>
<td>777777</td>
</tr>
<tr>
<td>000001</td>
<td>777776</td>
<td>(-1)</td>
</tr>
<tr>
<td>000002</td>
<td>777775</td>
<td>(-2)</td>
</tr>
<tr>
<td>077777</td>
<td>700000</td>
<td>7000001</td>
</tr>
<tr>
<td>100000</td>
<td>677777</td>
<td>700000</td>
</tr>
<tr>
<td>100001</td>
<td>677776</td>
<td>677777</td>
</tr>
<tr>
<td>377776</td>
<td>400001</td>
<td>400002</td>
</tr>
<tr>
<td>377777</td>
<td>400000</td>
<td>(-131,071)</td>
</tr>
</tbody>
</table>

Note that the 2's complement of 400000 is 400000. 400000 (-2^17) is too negative a number to have its complement (2^17) represented in 18 bits. The largest positive number which may be represented in 18 bits is 2^17-1 or 377777_8 or 131,071_10.

Zero

1) There are two forms of zero in 1's complement:

and 000000

and 777777.

2) There is only one form of zero in 2's complement:

000000 (777777 is a -1).
NOTE: There are 10000 (octal) locations in each page. Each page starts with an address that is a multiple of 10000.

There are 20000 (octal) locations in each bank. Each bank starts with an address that is an even multiple of 20000.
DOUBLE PRECISION ADDITION

Program to illustrate the use of two's complement addition in performing a double precision add. Two words are used for each DP number. The first word of the pair contains the sign and the most significant part. The second word contains the low order part. It is important to note that all bits of the low order part are numeric bits, i.e., the sign bit is considered as a numeric bit, not as a sign. For illustration purposes, assume that the word size is only six bits. Some DP numbers follow:

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>HIGH</th>
<th>LOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>0025</td>
<td>000 000</td>
<td>010 101</td>
</tr>
<tr>
<td>0063</td>
<td>000 000</td>
<td>110 011</td>
</tr>
<tr>
<td>0377</td>
<td>000 011</td>
<td>111 111</td>
</tr>
<tr>
<td>-1</td>
<td>111 111</td>
<td>111 111</td>
</tr>
<tr>
<td>-100</td>
<td>111 111</td>
<td>000 000</td>
</tr>
<tr>
<td>-40</td>
<td>111 111</td>
<td>100 000</td>
</tr>
</tbody>
</table>

The program to perform DP addition follows:

CLA!CLL  /Clear AC and link
TAD AL  /Get low half of A
TAD BL  /Add low half of B
DAC CL  /Save low half of result
CLA    /Clear AC, but link remains
GLK    /If addition of AL and BH caused a carry, link=1
        /and the one is placed in AC (17) so that it can
        /be added to the high half. If no carry, AC 17=0
TAD AH  /Add in high order parts of the two numbers
TAD BH
DAC CH

EXAMPLE 1. DP ADD 127+ 306 = 735

<table>
<thead>
<tr>
<th>HIGH</th>
<th>LOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>000 001</td>
</tr>
<tr>
<td>306</td>
<td>000 011</td>
</tr>
</tbody>
</table>

Add  low order
Add  high order
Add in LINK
Result

Add in LINK = 0
Result

EXAMPLE 2. DP ADD 1254 + 2362 = 3636

<table>
<thead>
<tr>
<th>HIGH</th>
<th>LOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>1254</td>
<td>001 010</td>
</tr>
<tr>
<td>2362</td>
<td>010 011</td>
</tr>
</tbody>
</table>

Add  low order
Add  high order
Add in LINK
Result

Add in LINK = 1
Result

RESULT

= 435

= 3636
The Central Processor Unit (CPU) functions as the main component of the computer by carrying on bidirectional communication with both the memory and 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.
INSTRUCTION SET

MEMORY REFERENCE

00 retrieve
60 modify

enter contents of memory locations

AUGMENTED

OPERATE 74
- operate on link and AC:
  - clear
  - complement
  - rotate
  - skip

IOT 70
- input/output transfers

EAE 64
- multiply
  - divide
  - shift
  - normalize

INDEX 72
- operations involving the XR the LR

OPERATION CODE
00-60
INDEX BIT *(1=INDEXED)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

INDIRECT ADDRESS
(1 INDIRE C)

OPERAND ADDRESS

*USED AS A THIRTEENTH ADDRESS BIT IN BANK MODE

Memory Reference Instruction Word

OPERATION CODE
64-6E EAE
70-71 IOT
72-73 INDEX
74-80 OPERATE

0 1 2 3 4* 5* 6 7 8 9 10 11 12 13 14 15 16 17

INSTRUCTION CODE

* THESE BITS USED AS PART OF THE INSTRUCTION CODE IN EAE AND OPERATE INSTRUCTIONS

Augmented Instruction Format
TITLE MACRO ASSEMBLY LANGUAGE EXAMPLE

ABS

LOC 100

START JMP .
START JMP START

DZM DZM DZM
LAC DZM
DZM
DZM

DZA DZM

VARIABLES

LAC CNT#
DAC CNT#
LAC CNT2#
DAC CNT2

LITERALS

A LAC (1
LAC TABLE
LAC (TABLE
LAC (A
LAC (C
LAC (B

INDIRECT

LAC 100
LAC# 100
LAC# START#2
JMP# A1
JMP 100
JMP START
JMP# (100

EJECT
LOC 1005
A1  A
B=100
C=4

01006 000100  D  B
01007 000004  C
01010 200100  LAC B
01011 200114  LAC A
01012 200220  LAC B+C+A
01013 201037  LAC (B+C+A)
01014 201040  LAC (C+N)

01015 001000  D  /MULTI-DEFINED SYMBOL
01016 001030  DD  /UNDEFINED SYMBOL

01017 000110  B=110
01020 200110  LAC B
01021 201041  LAC (B

01022 000010  A=B+C
01023 001042  A=(B+C)

01024 000000  TABLE 0
000100  .END START

01031 000001  *L
01032 001024  *L
01033 000114  *L
01034 000004  *L
01035 000200  *L
01036 000100  *L
01037 000220  *L
01040 001027  *L
01041 000110  *L
01042 777674  *L

SIZE=01044  4 ERROR LINES
.TITLE PSEUDO OPS EXAMPLES

ABS

.LOC 100

LAC 100

REPT

REPT 4

RTL

REPT 5,1

ZERO DZM BUFF

RESERVING STORAGE

BUFF BLOCK 5
BUFFE 0

TABLE 0

LOC ' +5

TABLEE 0

CONDITIONALS

IFDEF A

LAC A+1

.ENDC

B=2

IFDEF B

TAD CB

.ENDC

PAGE 1 PSEUDO SRC PSEUDO OPS EXAMPLES
/ ************************************
/ CHANGING THE LOCATION COUNTER
/ ************************************
/
01050
/
01050 405010
01051 251256
01052 455131
01053 604424
01054 292532
01055 020232
01056 426472
01057 340616
01058 425020
01059 000000

01060 406250
01061 330544
01062 314000
01063 000000

01064 010203
01065 616263

01066 406060
01067 330400

01068 010203
01069 610000

01070 406050
01071 330544

01072 010203
01073 610000

01074 406050
01075 330544

01076 010203
01077 616263

01100 010203
01101 616263

/ ************************************
/ PAGE EJECT ALSO CAUSED BY TITLE
/ ************************************

29
.TITLE PSEUDO OP EXAMPLES--THIS PAGE 1 MACROS

.DEFIN SUB A, B, C
LAC B
TCA
TAD A
DAC C
.ENDN

.CALLING A MACRO

SUB BUFFE, BUFF, TABLE
LAC BUFF
TCA
TAD BUFFE
DAC TABLE

SUB BUFFE, C5, TABLE+1
LAC C5
TCA
TAD BUFFE
DAC TABLE+1

SUB CMESS1, CMESS, TABLE+2
LAC CMESS
TCA
TAD CMESS1
DAC TABLE+2

SUB BUFFE, TABLE, TEMP
LAC TABLE
TCA
TAD BUFFE
DAC TEMP

.END

SIZE=01127
NO ERROR LINES
There are 4 programs in this series. They offer solutions (each using a different addressing mode) to the same problem:

Tables A, B and C each contain 5 one word entries. Add corresponding entries from Tables A and B and store the result in the corresponding entry in Table C.
I.e., C(i) = A(i) + B(i).
Do this without changing any of the values in Tables A and B.

These programs were written to illustrate the various types of addressing available on the PDP-15 and to demonstrate certain macro language elements and assembler directives:

(2-115) *STATEMENT FORMAT

(3-3) *TYPE OF BINARY OUTPUT TO BE GENERATED
  "ABS"
  "ABSP"

(3-10) *SETTING THE LOCATION COUNTER
  "LOC"

(3-2) *GETTING HEADINGS ON ASSEMBLY LISTINGS
  "TITLE"

(3-10) *SPECIFYING WHETHER NUMBERS ARE OCTAL OR DECIMAL
  "DEC"
  "OCT"

(3-11) *RESERVING BLOCKS OF MEMORY FOR STORAGE
  "BLOCK"

(2-6) *VARIABLES
  USING #

(2-13) *LITERALS
  USING (

(2-3) *DEFINING THE VALUE OF SYMBOLS
  USING THE SYMBOL AS A LABEL
  USING DIRECT ASSIGNMENTS

(2-15) *STORING VALUES IN SUCCESSIVE LOCATIONS

(3-11) *SPECIFYING THE PHYSICAL END OF PROGRAM
  "END"

The above is just a selected list of assembly language elements and assembler directives (also called pseudo operations). Make sure you are familiar with them. Each topic is discussed in the macro manual starting on the page given in parentheses.
.TITLE COMMENTARY ON SUM
/
/
*************************
/* POINTS TO BE NOTED */
/
*************************
/
/* 1. NOTE THE .TITLE STATEMENT AND WHAT IS PRINTED AS /
 A HEADER. */
/
/* 2. THE FIRST COLUMN GIVES LOCATIONS. NOTE THAT THE /
 LOCATION COUNTER BEGINS AT 100. (BECAUSE OF ".LOC 100) */
/
/* 3. THE "DAC COUNT" INSTRUCTION REFERENCES LOCATION 1;
 "COUNT#" IS USED IN AN ISZ INSTRUCTION IN LOCATION 110. /
 THIS INSTRUCTS THE ASSEMBLER TO SET UP A LOCATION WHICH /
 MAY BE REFERRED TO AS "COUNT". THE ASSEMBLER SET UP /
 LOCATION 133. (COUNT IS A VARIABLE) */
/
/* 4. VALUES ARE SET UP IN SEQUENTIAL LOCATIONS FOR /
 TABLES A AND B. IN EACH CASE THERE ARE ACTUALLY 5 /
 STATEMENTS ON ONE LINE (STATEMENTS ARE TERMINATED BY /
 SEMI-COLONS AND CARRIAGE RETURNS). NOTE THE SPACE /
 BEFORE EACH VALUE. */
/
/* 5. NO NEED TO ACTUALLY PLACE ANY VALUES IN /
 TABLE C==JUST TO RESERVE STORAGE SPACE. THE ".BLOCK /
 HAS THE EFFECT OF ADDING 5 TO THE LOCATION COUNTER. /
 NOTE THE ADDRESS OF LOCATION ".ENDLOC". */
/
/* 6. THE ".END" STATEMENT INDICATES THE PHYSICAL END /
 OF THE PROGRAM AND MUST BE THE VERY LAST STATEMENT /
 IN A PROGRAM. NOTE THAT IT IS BELOW COMMENTS. /
 WHAT DO YOU THINK WOULD HAPPEN ON THE LISTING IF THE /
 ".END" STATEMENT APPEARED BEFORE THOSE COMMENTS? */
/
/* 7. NOTE THE USE OF THE ".REJECT" STATEMENT (SO THAT /
 THE LISTING WILL CONTINUE ON A FRESH PAGE) IS NOT /
 NECESSARY BECAUSE A ".TITLE" STATEMENT ALSO /
 GENERATES A FORM FEED WHICH CAUSES THE LISTING /
 TO CONTINUE ON A FRESH PAGE. */
TITLE SOLUTION USES AN ADDRESS MODIFICATION TECHNIQUE

ABS / ABSOLUTE PROGRAM LOADED BY THE
/ ABSOLUTE LOADER TO RUN IN BANK MODE
LOC 100 / SET LOCATION COUNTER TO 100

START

LAW = 5
DAC COUNT
/ SET UP NEGATIVE COUNTER OF = 5
/ AS EACH TABLE HAS 5 ENTRIES.

AA
TAD 8
/ PERFORM ADDITION
/ OF ENTRIES

BB
TAD 0

CC
DAC C
/ STORE RESULT IN C

ISZ AA
/ MODIFY LOCATIONS AA, BB AND CC
ISZ BB
/ SO THAT THEY REFERENCE THE
ISZ CC
/ NEXT LOCATION IN THE TABLE.

ISZ COUNT
/ ARE WE DONE?

JMP AA
/ NO! BECAUSE WE DIDN'T SKIP
/ GO BACK FOR MORE.

HLT
/ YES! DONE.
/ NOTE! IF THIS PROGRAM IS TO BE
/ RUN AGAIN AA, BB AND CC SHOULD
/ BE REINITIALIZED (CAN YOU THINK
/ OF WAYS THIS COULD BE DONE?)

SET UP TABLES

A 1 2 3 4 5
000001
000002
000003
000004
000005
000006
000007
000008
000009
B 2 1 2 1 2 1 2
000001
000002
000003
000004
000005

JUST RESERVE SPACE FOR TABLE C

C .BLOCK 8

ENDLOC 0

/ THIS IS THE END OF THE PROGRAM.
/ THEREFORE, THIS IS WHERE WE WANT TO PUT
/ THE "END" STATEMENT.

SIZE=00134 NO ERROR LINES

#2. NOTE THAT DATA IS SEPARATED FROM THE INSTRUCTIONS WHICH OPERATE ON THE DATA. WE DO NOT WANT TO FALL INTO THE DATA AND START EXECUTING IT. WHAT INSTRUCTION WOULD WE HAVE IF WE "FELL INTO" AND TRIED TO EXECUTE LOC 113?

#3. NOTE THE USE OF "COUNT" AS A VARIABLE, (WHAT IS IT IN THE USE OF THE SYMBOL "COUNT" THAT MAKES IT A VARIABLE?) WHAT LOCATION IS SET UP BY THE ASSEMBLER SO THAT IT MAY BE REFERENCED USING "COUNT"?

#4. NOTE THAT WE RESERVE SPACE FOR TABLE C EVEN THOUGH IT IS AT THE VERY END OF THE WRITTEN PROGRAM. WHAT PROBLEM WOULD WE RUN INTO IF WE DID NOT RESERVE THIS SPACE? (HINT: SEE NOTE #3)

SOLUTION USES INDIRECT ADDRESSING

ABS
LOC 100

START LAW =5 DAC COUNT

NEXT LAC= AA /LOC AA CONTAINS THE ADD.
/OF A. LAC= AA GETS THE
/CONTENTS OF A IN AC.

TAD= BB /ETC. FOR BB
DAC= CC /ETC. FOR CC

ISZ AA /INCREMENT THE INDIRECT
ISZ BB /ADDRESSES OF AA, BB
ISZ CC /AND CC.

ISZ COUNT /SEE IF ALL DONE.
JMP NEXT /NOT DONE. GET NEXT DATA.
HLT /ALL DONE SO HALT.

A 1 2 3 4 5

B 2 3 4 5 6

AA A
BB B
CC C

C /BLOCK 5 /RESERVE SPACE FOR TABLE C.

END START /NO ERROR LINES

SIZE=00136
1. Note that in this program, data appears before the instructions in the program. The important thing is to separate the data from the instructions, but it doesn't matter which appears first.

2. Note that the first executable location in a program is not always in the first location used by the program. That is, there can be a difference between the program start address and the program load address. In this case, the program start address is 117, whereas the program load address is 100.

3. This program makes use of auto-increment addressing (with locations 10, 11, and 12), to access sequential locations in tables A, B and C respectively (LAC=10, TAD=11, DAC=12). This means that we want to load location 10 with a value 1 less than the start address of table A, location 11 with a value 1 less than the start address of table B and location 12 with a value 1 less than the start address of table C. Note that the "LAC (A=1)" and "DAC 10" instructions are used (locations 121 and 122) to do this.

The use of "LAC (A=1)" requests the assembler to set up a location containing the value "A=1". The assembler can evaluate expressions and does so moving from left to right... A=1 = 100-1 = 77. Location 136 is set up to contain the 77. The instruction "LAC (A=1)" is assembled as 200136 (load the accumulator with the content of location 136, i.e., the 77).

Similarly "DAC (C=1)" loads the accumulator with "104" and "LAC (C=1)" loads the accumulator with "111".

4. To store the "77" in absolute location 10 a "DAC 10" instruction is used. Would we be loading the "77" into absolute location 10 if the "LOC" statement "READ "LOC 26000" rather than "LOC 100"? 
..SOLUTION USES AUTO-INCREMENT ADDRESSING

.TITLE SOLUTION USES AUTO-INCREMENT ADDRESSING

A

ABS

LOC 100

1 2 3 4 5

8

2 4 1 5 3

C

BLOCK 5

START LAW =5

DAC COUNT=

//PUT =5 INTO COUNT

//THIS METHOD MAKES USE OF
//AUTO-INCREMENT REGISTERS

LAC (A=1)

DAC 10

//10, 11 AND 12, SINCE

DAC 11

//AUTO-INCREMENT IS A PRE-

LAC (B=1)

//INCREMENT, EACH OF THE

DAC 12

//REGISTERS MUST BE LOADED WITH

LAC (C=1)

//ONE LESS THAN THE STARTING ADD

DAC 12

//TO ASSURE THAT THE LOADING OF

NXT

LAC# 10

//AN AUTO-INCREMENT REGISTER WILL

TAD# 11

//WORK IN ANY PAGE OR BANK, THE

DAC# 12

//INSTRUCTION TO LOAD LOCS 10, 11

ISZ COUNT

//AND 12 SHOULD BE OF THE FORM

JMP NXT

//LAC (A=1)

HLT

//DAC# 10

~L

//GATHER UNDER

~L

//PICKETS OR BUNTS THAN 0

SIZE=00141

NO ERROR LINES

37
.TITLE COMMENTARY ON SUM3

************************************
* POINTS TO BE NOTED *
************************************

/* 1. NOTE THE USE OF "ABSP". THIS PROGRAM MAKES USE OF
  INDEXED ADDRESSING AND, THEREFORE, I MUST BE IN PAGE MODE
*/

/* 2. NOTE THE USE OF THE "DBA" INSTRUCTION. ALTHOUGH
  A "DBA" INSTRUCTION IS CONTAINED IN THE PAGE MODE
  VERSION OF THE ABSOLUTE LOADER (WHICH IS OUTPUT ON THE
  PAPER TAPE IN FRONT OF YOUR ASSEMBLED PROGRAM), IT IS
  STILL A GOOD IDEA TO INCLUDE A "DBA" IN YOUR PROGRAM
  IN CASE THE PROGRAM IS RESTARTED FROM THE CONSOLE
  WITHOUT RELOADING IT. DON'T RELY ON THE CONSOLE
  BANK-PAGE MODE SWITCH.
*/

/* 3. NOTE THE USE OF THE "DEC" AND "OCT" ASSEMBLER
  DIRECTIVES. COMPARE THE VALUES GENERATED FOR THE
  NUMBERS STATED WHILE UNDER DECIMAL RADIX (BASE 10),
  WITH THOSE GENERATED FOR THE NUMBERS STATED WHILE
  UNDER OCTAL RADIX (BASE 8).
*/

/* 4. NOTE THE "END BEGIN=1" STATEMENT. A "END"
  STATEMENT IS USED TO SPECIFY THE PHYSICAL END OF
  A PROGRAM. A "START ADDRESS" OR "TRANSFER ADDRESS"
  (THE LOCATION AT WHICH WE WANT THE LOADER TO START THE
  PROGRAM) MAY ALSO BE SPECIFIED IN THE END STATEMENT.
  THIS IS DONE BY FOLLOWING THE "END" WITH A SPACE OR
  TAB AND THEN GIVING AN EXPRESSION WHICH MAY BE EVALU-
  ATE BY THE ASSEMBLER. IN THE PREVIOUS EXAMPLES,
  WHEN A TRANSFER ADDRESS WAS GIVEN IT WAS A VERY
  SIMPLE EXPRESSION--THE LABEL USED ON THE FIRST
  EXECUTABLE LOCATION. BUT THE ASSEMBLER CAN HANDLE
  LONGER EXPRESSIONS. IN THIS CASE, THE SYMBOL BEGIN
  THE VALUE 10001, THEREFORE, BEGIN=1#10000,
  THUS, IN THIS CASE, "END BEGIN=1" HAS THE SAME
  EFFECT AS "END 10000".
*/

/* 5. NOTE THE ERROR DIAGNOSTIC GIVEN ON THE STATEMENT:
  TO BE ASSEMBLED INTO LOCATION 10025. THE "N" INDICATES
  AN ERROR IN NUMBER USAGE. WHAT IS THE ERROR?
  HOW WAS IT HANDLED BY THE ASSEMBLER (COMPARE LOCATIONS
  10020 AND 10025)?
*/
ROUTINE USES INDEXED ADDRESSING (& LIMIT REGISTER)

.TITLE ROUTINE USES INDEXED ADDRESSING (& LIMIT

10000

10000 707762

10000 200033

10002 720000

10003 735000

10004 210014

10005 350021

10006 000026

10007 725001

10010 600004

10011 740040

10012 000000

10013 000015

10014 000002

10015 000005

10016 000014

10017 000033

10020 000072

10021 000002

10022 000005

10023 000012

10024 000027

10025 000072

10026 000000

10033 000005

*ABSP

.LOC 10000

.OBA

.BEGIN

.LAC (5

.PAL

.PUT 5 IN LIMIT REG.

.CLX

.XR=0

.AA LAC A,X

.TAD B,X

.DAC C,X

.ADDRESS OF A + C(XR).

.ETC.

.ETC.

.AXS 1

.JMP AA

.HLT

.XR=LR. WE'RE DONE!

.ETC.

.CAL

.15

-INCREMENT AND TEST XR

.ETC.

.RETURN TO MONITOR

.BY HITTING CONSOLE CONTINUE.

.DEC

.A

.2 5 12 27 58

.OCT

.B

.2 5 12 27 58

.BLOCK 5

.C

.END BEGIN=1

.SIZE=10034

1 ERROR LINES
PROGRAM TO ADD TWO NUMBERS, USING TAD, AND TEST
THE RESULT FOR ARITHMETIC OVERFLOW.

START
CLA!CLL /CLEAR THE LINK AND THE AC.
TAD A /GET THE FIRST # IN THE AC.
AND MASK /GET RID OF ALL BITS EXCEPT THE SIGN.
TAD B /ADD THE 2ND NO. TO BIT Ø OF A.
SZL /SKIP IF ZERO LINK
JMP NEGNEG /NON ZERO LINK INDICATES BOTH A AND B NEG.
/JMP POSNEG /BECUSE ONES IN BOTH SIGN POSITIONS IS
/JMP POSPOS /THE ONLY WAY THE LINK CAN BE SET SINCE ONLY
/JMP NEGNEG /THE SIGN BIT OF THE 2ND # IS ADDED
/JMP POSNEG /THE OTHER IS -. CORRECT ADDITION IS ASSURED.
/JMP POSPOS /NO TEST IS NECESSARY, SO STORE THE SUM
HLT /AND HALT.

POSNEG
LAC A /SINCE ONE VALUE IS + ADD
TAD B /A NEGATIVE RESULT, ARITHMETIC OVERFLOW HAS
SPA /OCCURED, TEST THE AC FOR POSITIVE VALUE.
JMP POSERR /IF NEGATIVE, GO TO ERROR ROUTINE.
DAC SUM /POSITIVE RESULT SO SUM OK, STORE RESULT
HLT /AND HALT.

NEGNEG
LAC A /A POSITIVE RESULT, ARITHMETIC OVERFLOW HAS
TAD BQ /OCCURED, TEST THE AC FOR NEGATIVE VALUE.
SMA /IF POSITIVE, GO TO ERROR ROUTINE.
JMP NEGERR /NEGATIVE RESULT SO SUM OK, STORE RESULT
HLT /AND HALT.

MASK 400000

/ANOTHER, SHORTER SOLUTION FOLLOWS:
LAC A /IF THE OR'ED SUM OF THE SIGN
XOR B /BIT IS A 1, THE SIGNS WERE
SMA /DIFFERENT AND THE SUM MUST BE CORRECT
JMP LIKE /SIGNS WERE THE SAME

LIKE
LAC A /PERFORM THE ADDITION, AND THEN
TAD B /CHECK THE SUM FOR A CORRECT SIGN
DAC SUM /
AND (400000 /GET SIGN OF RESULT. IF OR'ED SUM
XOR B /GIVES A SIGN OF 0, THE SIGN OF THE
/RESULT IS THE SAME AS THE SIGN OF B.
SPA /SINCE A & B WERE SAME SIGN, RESULT IS OK
JMP ERROR /SIGN CHANGED, OVERFLOW OCCURED
HLT

/A THIRD SOLUTION MAKING USE OF ADD IS MUCH SIMPLER.
CLL /CLEAR LINK
LAC A /ADD THE TWO NUMBERS
ADD B /IF EITHER NO. IS NEG., IT MUST BE 1's COMP.
SZL /IF LINK IS SET, OVERFLOW
JMP ERROR /OCCURRED, GO TO ERROR
DAC SUM /ADDITION OK IF LINK=0
HLT
ADDRESSING

References:  Volume 1 Processor Handbook  4-3
             System Reference Manual       8-1

Memory Reference Instructions specify locations to be operated on by the Central Processing Unit. The CPU computes the actual (effective) address of the location referred to by combining bits from the instruction itself and also from the PC at the time the instruction is being executed. Various addressing modes require further computations with pointer words and the index register.

In the following illustrations:

PC: refers to the contents of the Program Counter at the time the MRI is being executed.

Instruction: refers to the contents of the location being executed.

XR: refers to the contents of the Index Register.

Pointer Word: refers to the contents of the location designated as a pointer word in an indirect reference.
DIRECT ADDRESSING

**PAGE MODE**

<table>
<thead>
<tr>
<th>block</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17</td>
</tr>
</tbody>
</table>

**Effective Address**

NOTE that the effective address is in the same page as the instruction.

---

**BANK MODE**

<table>
<thead>
<tr>
<th>block</th>
<th>bank</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17</td>
</tr>
</tbody>
</table>

**Effective Address**

NOTE that the effective address is in the same bank as the instruction.
INDIRECT ADDRESSING

PAGE MODE

block page

PC + Instruction

Address of Pointer Word

block

PC + Pointer Word

Effective Address

NOTE that the pointer word is in the same page as the instruction.

NOTE that the effective address is in the same block (32K) as the instruction.
INDIRECT ADDRESSING

**BANK MODE**

<table>
<thead>
<tr>
<th></th>
<th>block bank</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PC</strong></td>
<td></td>
</tr>
<tr>
<td><strong>+</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Instruction</strong></td>
<td></td>
</tr>
</tbody>
</table>

Address of Pointer Word

<table>
<thead>
<tr>
<th></th>
<th>block</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PC</strong></td>
<td></td>
</tr>
<tr>
<td><strong>+</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Pointer Word</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Effective Address**

NOTE that the pointer word is in the same bank as the instruction.

NOTE that the effective address is in the same block (32K) as the instruction.
INDEXED ADDRESSING

PAGE MODE ONLY

NOTE that the effective address may be anywhere in 128K. Care must be taken not to address non-existent memory (this includes negative addresses).
INDIRECT INDEXED ADDRESSING

PAGE MODE ONLY

```
P C  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
+------------------------
Instruction
    block  page
                        
    Address of Pointer Word

P C  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
+------------------------
Pointer Word
    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
+------------------------
XR
```

EFFECTIVE ADDRESS

NOTE that the effective address may be anywhere in 128K. Care must be taken not to address non-existent memory (this includes negative addresses).
AUTOINCREMENT ADDRESSING

PAGE MODE and BANK MODE

Instruction

set

address

10-17

Auto-
Increment Location(10-17)

+ 000001

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

EFFECTIVE ADDRESS

NOTE that the effective address may be anywhere in 128K. Care must be taken not to address non-existent memory (this includes negative addresses).
INDEXED AUTOINCREMENT ADDRESSING

PAGE MODE ONLY

Instruction

both set

address

10-17

Auto-
Increment
Location(10-17)

+ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

XR

+ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

EFFECTIVE ADDRESS

NOTE that the effective address may be anywhere in 128K. Care must be taken not to address non-existent memory (this includes negative addresses).
SUBROUTINES

What: A section of code, usually performing one task, that may be called from various points of the main program.

How: The JMS instruction is used to enter subroutines. Recall that upon a JMS
a) The updated PC is stored at the address specified in the JMS.
b) The (specified address) +1 is now placed in the PC so that execution is picked up at the second location of the subroutine.

<table>
<thead>
<tr>
<th>ex.</th>
<th>Before</th>
<th>JMS</th>
<th>MOVE</th>
<th>After</th>
<th>JMS</th>
<th>MOVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>100</td>
<td>JMS</td>
<td></td>
<td>100</td>
<td>JMS</td>
<td>MOVE</td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>MOVE</td>
<td></td>
<td>101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

200 MOVE Ø 200 MOVE 101

PC 201

c) Return to the calling program is effected via a "JMP*" on the first location of the subroutine.

MOVE Ø

JMP* MOVE

Passing Arguments - very often a subroutine has to process data contained in the calling routine. In order to do this the calling routine must pass this data or the address(es) of the data to the subroutine. This is called "passing arguments."

There are a number of ways this can be done. Among them are:

1) AC passing arguments in CPU
2) MQ,XR,LR registers
3) trailing arguments: JMS SUB
   Arg 1
   Arg 2
   Arg 3
4) Setting up a list of arguments and passing the address via (1), (2), or (3).

\begin{tabular}{ll}
LAC & LIST ADDRESS or JMS SUB etc. \\
JMC & SUB LIST ADDRESS \\
\vdots & \\
\end{tabular}

LIST ADDRESS ARG1 ARG2 \vdots ARGn
PC15 High-Speed
Paper-Tape Reader Punch

1.1 INTRODUCTION

The PC15 High-Speed Paper Tape Reader/Punch is used to input perforated paper-tape programs into core memory, or to punch core memory programs or data on paper tape. Information is punched on 8-channel fanfolded paper tape in the form of 6- or 8-bit characters at a maximum rate of 50 characters/second. Information is read at a maximum rate of 300 characters/second. The PC15 consists of a PC05 Paper Tape Reader/Punch with interface and control logic for using the reader/punch with a PDP-15.

1.2 PAPER-TAPE READER

1.2.1 Characteristics and Capabilities

Data can be read from tape and transferred to the PDP-15, using the computer hardware readin logic or using program-controlled transfers. For hardware readin operation, the hardware readin logic supplies inputs for selecting the operating mode, starting tape motion, and implementing transfers. For program-controlled transfers, the computer issues input/output transfer (IOT) instructions that select the operating mode, advance the tape, and implement the transfer. To maintain a maximum rate of 300 characters/second, a new select IOT must be issued within 1.67 ms of the last reader flag. If not, the reader operates start-stop and reads characters at a 25 character/second rate. The requirements for maximum character rate are described in detail in Programming Considerations, Paragraph 1.4.1.

The reader interfaces with the automatic priority interrupt (API) facility, the program interrupt facility, and the input/output skip chain. For API operation, the reader is assigned API level 2; a unique entry address of 508 is assigned to its service routine.

The reader contains a no-tape sensor and flag (character ready for transfer) circuits. If a no-tape condition is detected, the reader flag is set, and a program interrupt is initiated whenever a reader select IOT is given. The states of the reader flag, the reader API 2 level, PI request and skip request
devices are displayed on an indicator panel at the top of Cabinet H963E (Bay 1R). In addition, this panel displays the reader buffer contents and the I/O address (API unique entry address). These items and the reader controls are described in Controls and Indicators, Paragraph 1.2.3.

Reader mechanical facilities include a right-hand bin for supply for tape being read, a left-hand bin for receiving the tape, and a feed-through mechanism to control passage of the tape into the receiving bin. A snap-action retainer on the feed-through mechanism facilitates simple loading of the tape.

1.2.2 Operating Modes

The PC15 reader operates in either an alphanumeric or binary mode. For program-controlled transfers, the operating mode is selected by IOT instructions. For hardware readin operation, control logic in the reader automatically selects the binary mode.

When alphanumeric mode is selected, one 8-bit character (in ASCII code) is read and transferred to the PDP-15 accumulator. In the binary mode, the reader reads three 6-bit characters (three frames with channels 7 and 8 ignored) from tape and assembles them into an 18-bit word for transfer to the accumulator.

1.2.3 Controls and Indicators

Two front panel controls are provided for the PC15 Paper-Tape Reader: ON LINE/OFF LINE and FEED. The ON LINE position places the reader under computer control. The OFF LINE position, which is used for loading paper tape, raises an out-of-tape flag and places the reader under local control. The indicators associated with reader operation are located on an indicator panel at the top of cabinet H963E (Bay 1R). Table 1-1 lists the indicators and their functions.

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>READER BUFFER 00-17</td>
<td>Indicates the contents of the paper-tape reader buffer.</td>
</tr>
<tr>
<td>API 2 RDR</td>
<td>Denotes API level 2 is active as the result of a reader interrupt.</td>
</tr>
<tr>
<td>I/O ADDRESS</td>
<td>Indicates the unique trap address associated with I/O devices; address 50g for paper-tape reader.</td>
</tr>
<tr>
<td>RDR FLG</td>
<td>Denotes information has been read from tape and is available for transfer from reader buffer.</td>
</tr>
</tbody>
</table>
Table 1-1 (Cont)
Indicators Associated with Paper-Tape Reader

<table>
<thead>
<tr>
<th>Indicator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI RQ</td>
<td>Denotes one of the I/O devices (including paper-tape reader) handled by the BA15 Peripheral Expander has generated an interrupt request.</td>
</tr>
<tr>
<td>SKIP RQ</td>
<td>Denotes one of the I/O devices (including paper-tape reader) handled by BA15 has responded to a skip IOT instruction.</td>
</tr>
</tbody>
</table>

1.2.4 Tape Formats

The format of the perforated paper tapes for the alphanumeric (ASCII usage) mode is shown in Figure 1-1. In addition, tape channels are related to the PDP-15 accumulator stages. The leader and trailer portions of the tape are used to introduce or conclude a paper-tape program. Only the feed hole is punched for the leader/trailer portions. Note that each character is read by one IOT instruction.

![Figure 1-1 Tape Format and Accumulator Bits (Alphanumeric Mode)](image-url)
The paper-tape format for binary mode using hardware readin (HRI) is shown in Figure 1-2 as well as the relationship of accumulator stages for the 18-bit word. Note that only the feed hole is perforated for the leader/trailer portion and that channel 8 is always punched in the program portion of the tape. Any character without hole 8 punched will be ignored. Channel 7 punched in the last character indicates the last 18-bit instruction is to be executed by the computer. This instruction can halt machine operation or can transfer machine control to another part of the program. When using this format, channel 7 must be punched using the alphanumeric mode.

1.2.5 Instructions

The PDP-15 IOT instructions used for program-controlled loading of paper-tape data are listed below. Refer to Volume 1 of this handbook for IOT instruction format.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Octal Code</th>
<th>Operation Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSF</td>
<td>700101</td>
<td>Skip next instruction if reader flag is a 1.</td>
</tr>
<tr>
<td>RCF</td>
<td>700102</td>
<td>Clear reader flag. Read reader buffer, inclusively OR contents of reader buffer with AC, and deposit result in AC.</td>
</tr>
<tr>
<td>RRB</td>
<td>700112</td>
<td>Read reader buffer and clear reader flag. Clear AC and transfer contents of reader buffer to AC.</td>
</tr>
<tr>
<td>RSA</td>
<td>700104</td>
<td>Select alphanumeric mode and place one 8-bit character in reader buffer. Clear flag before character is read from tape. Set reader flag to 1 when transfer to reader buffer is complete.</td>
</tr>
<tr>
<td>RSB</td>
<td>700144</td>
<td>Select binary modes. Assemble three 6-bit characters in reader buffer. Clear reader flag during assembly and set flag when assembly is complete.</td>
</tr>
</tbody>
</table>

The paper-tape reader responds to an input/output read status (IORS) instruction by supplying the status of its device flags and no-tape flags to the accumulator. The reader device flag (reader interrupt) interfaces with bit 01 of the accumulator. The reader no-tape flag interfaces with bits 08 of the accumulator.

1.2.6 Functional Description

The PC15 reader consists of an electromechanical tape feed system, a light source and photo cells for sensing tape perforations, a buffer register for storing and assembling data, and control logic for computer interface, tape advance, and transfer operations. These circuits can be used with the PDP-15 hardware readin logic, or can be used for program-controlled transfers, as described in the following paragraphs.
Figure 1-2  HRI Tape Format and Accumulator Bits (Binary Mode)
1.2.6.1 Hardware Readin Operation - The PC15 reader can be used with PDP-15 hardware readin logic to load programs from paper tape at a rate of 300 characters/second. For this operation, the desired tape is installed in the high-speed reader, and the program loading address is selected, using the console ADDRESS switches. The console RESET key is then pressed to initialize the computer and paper-tape reader. A readin operation is started by pressing the READIN key on the console.

With this key action, a Readin (RI) condition is stored in the reader, and the binary mode is selected. The reader then advances the tape, reads three characters from tape, assembles them into an 18-bit word in the reader buffer, and signals the hardware readin logic with a program interrupt. The hardware readin logic, in turn, transfers the 18-bit word to the accumulator under I/O processor and computer timing. The word is subsequently loaded into core memory by forcing a DAC instruction. The first 18-bit word is stored at the address specified by the console ADDRESS switches. Subsequent 18-bit words are stored in sequential memory locations.

The readin operation continues until a perforated hole 7 is detected. This condition is inserted in the last character of the last 18-bit instruction. When this condition is detected, the reader supplies the hardware readin logic with a skip request. As a result, the hardware readin logic causes the last instruction to be loaded into the Memory Input register for execution. This instruction can halt machine operation (HALT) or can transfer program control to another part of the program (JMP). When using the readin feature with the MP15 Memory Parity option, the last instruction on the paper tape (which will be 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. Therefore, that location should be re-stored by the program before an attempt is made to read from it. Otherwise, a parity error will occur.

1.2.6.2 Program-Controlled Operation - The PC15 reader operates in the binary or alphanumeric mode depending on the select IOT instructions issued by the computer. On decoding a reader select alphanumeric (RSA) mode IOT (700104g), the reader advances the tape one character, loads this character into the reader buffer, and sets the reader device flag. The reader then signals the computer that data are available by providing a reader interrupt to the API or PI, or by responding to an RSF IOT instruction. If the API facility is being used, program control is transferred to the reader service routine where the computer services the request, and an RCF (700102g) or RRB (700112g) instruction is issued. If the API facility is not being used, the computer issues an RSF instruction, and the reader returns a skip request whenever its flag is set. The skip request causes the next instruction (normally a JMP -1 in wait loops) to be skipped so that the character can be transferred to the accumulator by issuing an RCF or RRB instruction. The RCF or RRB instruction transfers the reader buffer character to the I/O bus and loads it into the least significant bits (10 through 17 for 8-bit alphanumeric character) of the accumulator. The character is subsequently stored in a core memory location designated by the program. The read reader buffer (RRB) instruction also clears the reader flag for the next read operation.

For binary mode operation, the computer issues a reader select binary (RSB) mode instruction (octal 700144). On decoding this instruction, the reader clears its device flag, advances the tape three
characters, reads these characters from tape, and assembles them into an 18-bit word in the reader buffer. The reader also counts the number of characters with hole 8 punched read from tape and, when a count of three is reached, generates an interrupt request. The control functions for transfer of the 18-bit word to the accumulator is the same as that described for the alphanumeric mode.

1.3 PAPER-TAPE PUNCH

1.3.1 Characteristics and Capabilities

The PC15 paper-tape punch consists of a tape feed system, a mechanical punch assembly, a buffer register, and control logic for mode selection and activation of the tape feed and punch mechanism. Tape advance, mode selection, and transfer of information to the punch are controlled by IOT instructions. Tape is perforated at a rate of 50 characters/second. When the punch is selected by an IOT instruction, data from the PDP-15 accumulator (AC10-AC17) are transferred to the punch buffer. Then, without further inputs, a character is perforated on tape.

The punch contains a device flag that denotes punch status for transfers. This device flag interfaces with the PI facility and I/O skip chain. The status of the punch flag is displayed on an indicator panel at the top of Cabinet H963E (Bay 1R). An out-of-tape switch is located on the punch mechanism. This switch initiates action that stops punch operations when approximately one inch of unpunched tape remains.

Power for the punch operation is available whenever the PDP-15 power is on. The punch runs when selected by an IOT instruction or when the FEED switch is pressed.

Punch mechanical features include a magazine for unpunched tape and a container for tape chad. Both are accessible when the reader-punch drawer is extended from the cabinet.

1.3.2 Operating Modes

The PC15 Punch operates in the alphanumeric or binary mode as designated by IOT select instructions. One of these instructions is required for each character punched for mode change. In the alphanumeric mode, an 8-bit character (in ASCII or modified ASCII code) is punched for each accumulator transfer to the punch. For the binary mode, one 6-bit data character is perforated for each accumulator transfer. Hole 8 is always punched, and hole 7 is never punched. Three of these characters, however, form one computer word for reading operations.
1.3.3 Controls and Indicators

The PC15 Punch has a front panel FEED control. This control is used to advance the tape from the punch as required for leader or trailer. The punch also has one indicator (PUN FLG) directly associated with its operation. This indicator, located on an indicator panel at the top of Cabinet H963E (Bay 1R), indicates the status of the device flag and, shows that the punch is available for a punch operation when lit. The punch also shares the PI RQ and SKIP RQ indicators on this panel with other I/O devices.

1.3.4 Tape Formats

Tape formats are shown in Figures 1-1 and 1-2.

1.3.5 Instructions

The PDP-15 IOT instructions used for punching of paper tape under program control are listed below. Refer to Volume 1 of this handbook for IOT instruction format.

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Octal Code</th>
<th>Operation Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSF</td>
<td>700201</td>
<td>Skip next instruction if punch flag is a 1.</td>
</tr>
<tr>
<td>PCF</td>
<td>700202</td>
<td>Clear punch flag and punch buffer.</td>
</tr>
<tr>
<td>PSA</td>
<td>700204</td>
<td>Select alphanumeric mode and punch one character. Set punch flag when punch is complete.</td>
</tr>
<tr>
<td>PSB</td>
<td>700244</td>
<td>Select binary mode and punch one 6-bit character. Set punch flag when punch is complete.</td>
</tr>
</tbody>
</table>

The punch responds to the IORS instruction (Volume 1, Paragraph 3.7.1) by supplying the status of its device flag and no-tape flag to the accumulator. The device flag interfaces with bit 02 of the accumulator, and the no-tape flag interfaces with bit 09.

1.3.6 Functional Description

The PC15 Punch operates in the alphanumeric or binary mode, depending on whether a PSA or PSB instruction is issued. When one of these instructions is decoded, information is loaded into the punch buffer from bits 10 through 17 of the accumulator and is punched onto tape. During the interval the punch operation is in progress, the punch flag is cleared to indicate the punch is busy. When the punch operation is complete, the punch flag is set to 1 to indicate it can accept another input character.
The operating sequence for punch operations normally begins with a PSF instruction to test the device flag. If the device flag is 1, a skip request is returned to the computer, and the computer issues a PCF instruction. This instruction clears the device flag and the punch buffer. The computer then issues a PSA or PSB instruction. On decoding a PSA instruction, the reader loads the accumulator input into its buffer, advances the tape, and punches one character. For the alphanumeric mode channel 8 is punched as a function of bit AC10. For the alphanumeric mode channel 7 is perforated as a function of bit AC11. After the character is punched, the reader sets its device flag, and the process is repeated. This operation, performed by the PCF and PSA instructions, can be combined by microprogramming the two instructions to form octal 700206.

The same principles are used for punching a binary character; however, a PSB instruction is used in place of the PSA instruction. On decoding a PSB, the punch perforates channel 8 and inhibits the punching of channel 7. The remaining six channels are punched as a function of AC12 through AC17, and represent one 6-bit character of a computer word.

1.4 PROGRAMMING CONSIDERATIONS

1.4.1 High-Speed Paper-Tape Reader

To use the reader at the transfer rate of 300 cps, a select IOT (RSA or RSB) must be issued within 1.67 ms after each flag. This action is required because a 40 ms reader stop delay is present. When this delay is activated, it overrides the select IOT input and subsequently stops the tape. Thus, if a new select IOT is not received within 1.67 ms of the setting of the flag, the reader operates start-stop and reads characters at 25 cps rate. No data is lost.

The RSA (octal 700104) and RCF (octal 700102) can be microprogrammed to form an octal 700106 instruction. This instruction reads the character, transfers the character to the accumulator, and advances the tape in one operation. An RSF (octal 700101) and RRB (700112) cannot be microprogrammed.

1.4.2 High-Speed Paper-Tape Punch

Channel 7 can be punched using only the alphanumeric mode. Therefore, when punching the last character of a tape for hardware reading operation, the last character must be punched in the alphanumeric mode.

The PCF instruction can be microprogrammed with a PSA or PSB instruction to form octal 700206 or 700246. This instruction clears the punch flag and buffer, selects the applicable mode, loads the
punch buffer, advances the tape, and perforates the character on tape. After completing the punching, the punch flag is set to denote the punch can accept another character. Microprogramming the PCF and PSF instructions is not allowed.

1.5 PROGRAMMING EXAMPLES

1.5.1 Paper-Tape Reader/Punch Handlers

All PDP-15 Systems are supplied with standard I/O device handler subroutines for the paper-tape reader/punch hardware. For PDP-15/10 Systems with 4K core, the COMPACT software includes paper-tape handler routines such as PTLIST and PTDUP. The Basic I/O Monitor, supplied with PDP-15/10E Systems with 8K core or greater, include standard I/O device handlers for the high-speed paper-tape reader and punch. These standard device handlers operate in systems with or without API and are upward compatible with all other monitors on the PDP-15/20 Software System. Complete instructions on use of standard paper-tape reader and punch handlers and their modification for special applications are provided in the PDP-15/10 Software System Manual, DEC-15-GR1A-D.

1.5.2 Paper-Tape Reader Programming Example

The following subroutine illustrates the use of programmed IOT instructions to read a group of binary words from paper tape. Twenty-five 18-bit words are read and stored in a table starting at ADDRESS.

NOTE

This example is for instructional purposes only and is not to be considered a complete, fully tested software system segment.

SUBRTE 0
LAW -31  /25 DECIMAL WORDS
DAC WDCNT
LAC (ADDRESS
PAX
READLP IORS
AND (1000  /IS THE PAPER TAPE READER EMPTY?
SZA  /YES IF NON-ZERO.
JMP* SUBRTE  /EXIT....ITS EMPTY
RSB  /NO. START READING 25 WORD.
RSF
JMP .-1  /WAIT FOR IT.
RRB  /GET IT FROM HARDWARE BUFFER.
DAC 0,X
AXR 1  /POINT TO NEXT LOC AT ADDR.
ISZ WDCNT  /HAVE 25 WORDS BEEN READ?
JMP READLP  /NO...CONTINUE LOOPING.
JMP* SUBRTE  /YES. EXIT.
1.3.3 Paper-Tape Punch Programming Example

The following subroutines illustrate some paper-tape punch programming considerations. Their purpose is to unpack successive 6-bit ASCII characters from a table, convert them to 7-bit ASCII, and punch them on paper tape. The starting address of the table is placed in a location named ADDRESS. The number of words in the table is placed in WORDCNT. After these parameters have been deposited, the subroutines are entered by a JMS to PNCHOUT.

NOTE
This example is for instructional purposes only and is not to be considered a complete, fully tested software system segment.

PNCHOUT

0
LAC WORDCNT /THIS INITIALIZATION
TCA
DAC WORDCNT /ROUTINE STORES 2'S
CLX /COMPLEMENT WORDCNT

NXTWORD

LAW -3 /SET UP A COUNTER FOR
DAC COUNT /3 CHARACTERS.
LAC ADDRESS,X
RAL /USE XR TO GET EACH WORD.

NXTCHAR

RTL /AC HOLDS 3 6-BIT ASCII
RTL /CHARS. ROTATE INTO LINK.
RTL /ROTATE WORD 6 PLACES
RTL /THRU LINK. THE NEXT
RTL /6-BIT CHAR. IS IN AC12-17.

DAC SAVEAC /SAVE REMAINING CHAR.
AND (77) /THIS ROUTINE CONVERTS
TAD (40) /THE 6-BIT ASCII IN AC12-17
AND (77) /TO 7-BIT ASCII IN
TAD (40) /AC11-17.

JMS PPCHAR /READY TO PUNCH CHAR.
LAC SAVEAC /RESTORE SHIFTED AC.
ISZ COUNT /LAST CHARACTER?
JMP NXTCHAR /NO. DO NEXT CHARACTER.
AXR 1 /POINT TO NEXT WORD.
ISZ WORDCNT /LAST WORD?
JMP* PPASCII /NO. DO NEXT WORD.
JMP* PNCHOUT /YES. RETURN TO PROGRAM.

PPCHAR

0
DAC STORE /SAVE CHAR. FOR 'NO TAPE'
IORS /TEST.
AND (400) /LOAD PUNCH STATUS INTO AC.
SZA /TEST NO PUNCH TAPE BIT.
JMP EOT /SKIP IF TAPE OK.
JMP /GO TO END OF TAPE RTE.
1.5.4 Programming With API or PI

The standard device handlers for the high-speed paper-tape reader and punch include complete interrupt subroutines for both API and PI service. Details on how the Program Interrupt Control (PIC) skip chain and the Automatic Priority Interrupt (API) channels are set up and provided in Part III of the PDP-15/10 Software System Manual. The following example of a hypothetical interrupt service subroutine is provided for general understanding of interrupt servicing.

NOTE
This example is not a complete, fully-tested interrupt service handler.

1.5.4.1 Program Interrupt Example

```plaintext
LAC    STORE    /LOAD AC WITH CHARACTERS
PSA
PSF
JMP   .-1
PCF
JMP*  PPCHAR    /RETURN TO SUBPROGRAM.
```

```plaintext
1.5.4.1 Programming With API or PI

The standard device handlers for the high-speed paper-tape reader and punch include complete interrupt subroutines for both API and PI service. Details on how the Program Interrupt Control (PIC) skip chain and the Automatic Priority Interrupt (API) channels are set up and provided in Part III of the PDP-15/10 Software System Manual. The following example of a hypothetical interrupt service subroutine is provided for general understanding of interrupt servicing.

NOTE
This example is not a complete, fully-tested interrupt service handler.

1.5.4.1 Program Interrupt Example

```plaintext
RSB    /ISSUE READER SELECT
.
.
.LOC  0

PI
0    /SAVE PC, LINK, EXTEND MODE

JMP  SKPCHN    /& MEM. PROT. BITS AT LOC 0.
.
SKPCHN  SPFAL    /GO TO SKIP CHAIN.
SKP
JMP*  INT6    /GO TO NEXT TEST.
RSF
SKP
JMP*  INT2    /GO TO POWER FAIL SUBRTE.
PSF
SKP
JMP*  INT3    /GO TO PTR INTERRUPT.
.
```

/INT6, INT2, AND INT3 ARE PART OF A TABLE
/OF INTERRUPT SERVICE ROUTINE STARTING ADDRESSES.
/AN EXAMPLE OF INT2 FOLLOWS:

62
INT2  PTRPIC
       /15-BIT ADDRESS OF PAPER
       /TAPE READER SERVICE
       /ROUTINE.
       /OTHER I/O SERVICE ROUTINE
       /POINTERS

       PTRPIC  DAC   PTRAC
       LAC*   (0)
       /SAVE AC.
       /SAVE PC, LINK, BANK MODE
       /AND USER MODE IN PTROUT.
       /REST OF INTERRUPT HANDLED.

1.5.4.2 API Example

       RSB
       /SELECT READER IN
       /BINARY MODE
       /REST OF INTERRUPT
       /HANDLED.

       .LOC  50
       JMS   PTRINT
       /PAPER TAPE READER
       /API ENTRY LOCATION

       PTRINT
       0
       DAC   PTRAC
       LAC   PTRINT
       /API ENTRY. SAVE AC.
       /SAVE PC, LINK, BANK
       /MODE & USER MODE BITS.
       /
PAPER TAPE FORMATS

Assembled Programs May Appear In Two Formats

Hardware
Readin
Format

1. Loaded using Hardware control logic
2. Output from assembler when use $FULL
3. Load address supplied by DATA SWITCHES on CONSOLE
4. All words loaded sequentially - no way to change where loaded
5. Read continues until a frame with CHANNEL 7 punched is detected.

Absolute
Binary
Format

1. Loaded using Absolute Loader
2. Output from assembler when use $ABS
3. Load address for each block supplied as first word in data block
4. All words loaded sequentially until new load address supplied by data block
5. Read continues until START BLOCK.
PAPER TAPE FORMATS

Programming tapes are supplied in one of two formats:
1. HRI - hardware read-in mode (.FULL assembly parameter)
2. BINARY OR ABS - (.ABS assembly parameter)

HRI tapes consist of 18 bit data and instructions punched in binary mode (PSB), which are loaded in sequential memory locations via the Hardware Read-In feature. The last word is an instruction which is to be executed when read (i.e. HLT or JMP). The last word is indicated by channel #7 being punched in the last frame of that word.

The load address is supplied by the address switch register.

ABS or Binary paper tapes consists of 3 basic parts:
1. ABS Loader Program punched in Hardware Read-In Format
2. Data Blocks (there may be more than one)
3. Start Block (there is only one)

The ABS Loader (Absolute Loader) is a program in HRI format. When read via the Console "Read-In" Key, it is loaded and started automatically. While executing, the Absolute Loader reads and loads the remainder of the tape. The Absolute Loader expects the tape it is reading to have a particular format containing Data and Start Blocks.

DATA BLOCK - Consists of 3 control words (Data Block Header) followed by the data to be loaded:
1. Load Address
2. Word count (not exceeding 25 and stored as a 2's complement negative number)
3. Checksum

DATA

.
.
.

START BLOCK - A two word block at the end of the tape. It is distinguished from a Data Block because bit 0 of the first word is a one (i.e. channel #6 in the first of 3 frames is punched).
1. Starting address (777777 means "HLT" rather than "JMP" to some location)
2. Dummy word (not used)
PAGE 1

ABSVER SRC

00100
00100 740040 BEG
00101 750004
00102 340112
00103 040111
00104 740040
00105 750004
00106 540111
00107 750040
00110 750041
00112 000010 *L
  SIZE=00113

.END BEG.

NO ERROR LINES

END OF ABS LOADER

LA } DATA
WC } BLOCK
\ SUM HEADER

DATA

LA } DATA
WC } BLOCK
\ SUM HEADER
\ DATA

START
BLOCK
Formatting Dectapes

The format generator tape is read in under HRI mode

1. Set the address switches to 17720 (some tapes may call for 17700).

2. Depress STOP, RESET, READIN.

Conversation program begins. Place Dectape on a unit other than unit 0. Don't take up more than 2 wraps. Follow directions.

Note: Before you can use the tape, you must go to PIP to clear out the directory.

```$PI
DOSPIP VXX
>LU<DT1
NLDT2```

Loading DOS Into a Cold Machine

1. Mount DOS restore tape #1 on Dectape transport
   Set rotary switch to 1
   Set remote switch
   Set write lock

2. Load DOS save-restore paper tape thru high-speed reader
   Place tape in reader
   Set address switches to 17720
   Depress STOP, RESET and then READIN

The paper tape is read in and a conversational mode program begins.

```INPUT: DT$
UNIT: 1$
OUTPUT: DK$
```

The program will tell you when to mount the second tape.
At the completion of loading, DOS has been placed on the disk.

3. To bring the resident Monitor into core:
   Set address switches to X7637 where X = 1 for 8K
   3 for 16K
   5 for 24K
   7 for 32K

   Place the DOS bootstrap tape in the reader (usually on the same tape as the DOS SAVE & STORE paper tape)
   Depress STOP, RESET, then READIN.

   DOS announces itself.
Note: The BOOTSTRAP stays in high core unless you cleverly manage to destroy it.

To get the monitor:

(a) Control C (echoed as ^C)
(b) Set address switches to X7646 where X is same value as before: STOP, RESET, START
(c) Set address switches to X7637 and read in bootstrap again
(d) The whole DOS restore procedure again
(e) Call a maintenance person

**PIPK**

To call PIP

```
$PIPK
DOSPIPK VXX
```

To list the directory from the disk on the teletype:

```
LT TTY DT2
```

To list the directory from the Dectape on unit 2 on the line printer:

```
LT LP DT2
```

To transfer a file from the disk to dectape:

```
LT DT1 DT2 FILEA BIN DT2 FILEB BIN
```

**MACRO**

To call MACRO

```
$MACRO
MACRO VXX
BN FILENM
```

To load a program generated under .ABS or .ABSP:

1. Place tape in reader
2. Set address switches to 17720
3. Depress STOP, RESET then READIN.

This causes the absolute loader to be loaded. It starts automatically and in turn reads in the rest of the tape—that is, your program.
EXAMPLE OF WRITING, ASSEMBLING AND LOADING A RELOCATABLE PROGRAM

EDIT

OPEN SAMPLE
FILE SAMPLE SRC NOT FOUND.
INPUT
LAC (707070)
HLT
*END

EDIT
CLOSE

editor V3A000
OPEN SAMPLE
EDIT
N
C //START/
START LAC (707070)
*L *END
*END
A START
P
*END START

EXIT

DOS-15 V3A000
A LP -12
SA DK -13
SK ON

$MACRO

$MACRO-15 V3A000
SL SAMPLE
END OF PASS 1
SIZE=00003 NO ERROR LINES PAGE 1 SAMPLE SRC

DOS-15 V3A000
LOAD

BLOADER V3A000
P SAMPLE SRC 77634
S+T
The Real Time Clock option provides a user with time reference capability for accounting purposes, periodic interrupts and interval timing. The clock produces clock pulses at the rate of:

   a) 60 times a second (every 16.7 ms) for 60 Hz systems
   or  b) 50 times a second (every 20 ms) for 50 Hz systems
      (the standard clock works off the line frequency—other clocks are available to produce clock pulses at user defined rates).

When the clock is enabled (CLON), every clock pulse generates a request for a break at the completion of the current instruction. When the break is granted by the CPU, the content of memory location 000007 is incremented by 1. Location 000007 is the clock counter register. As long as the clock is enabled, the process of location 7 being incremented at each clock tick continues.

When the content of location 7 overflows (i.e. is incremented from 777777 to 000000), the clock flag is set to 1. This condition may be checked for by the use of a Skip IOT (CLSF). Note also that the clock flag is interfaced to the PI and API systems so that if interrupts are enabled when the clock flag is set, an interrupt request will be made.

Three IOT instructions are associated with the clock:

- CLON 700004
  - Clock On
  - Enable the clock...increment location 000007 every clock tick
  - Clear the clock's flag

- CLSF 700001
  - Skip on Clock Flag Set...the next instruction is skipped if the clock's flag is set

- CLOF 700044
  - Clock Off
  - Disable the clock...do not increment location 000007
  - Clear the clock's flag.

Since the clock counter register is memory location 000007, its contents may be modified by a program. A standard technique for using the clock is to preset the contents of location 7 to the complement of the desired time count (in ticks) and then to enable the clock (and the interrupt system if interrupts are to be used). The clock flag will be raised (and an interrupt occur) at the end of the specified time period. For instance, to raise the flag after:

1 second, set location 000007 to 777704 (−60₁₀ = −74₈)

5 seconds, set location 000007 to 777324 (−30₀₁₀ = −45₄₈).

Notice that it is the 2's complement that is used. This example and the following ones assume 60 ticks per second for the clock.
REAL TIME CLOCK

To check for an interval of 1 second by checking the clock's flag, the following sequence can be used:

```
LAW -74     /74 ticks = 1 second
DAC 7       /or DAC* (7 if program is not in page 0
            / or bank 0
CLON        /enable clock--start incrementing
CLSF        /check for clock overflow
JMP .-1     /not yet
next instruction /get here after 1 second
```

To check for an interval of 1 second via interrupts under the PI system, the following sequence may be used:

```
.LOC 0
0
JMP* .+1
TIMERT

/Main Routine
.LOC 10200

....
LAW -74     /set clock for
DAC* (7     /6010 ticks
CLON        /enable clock; clear flag
ION         /enable PI interrupts
....
....
continue with 1 second's worth of program
....

.LOC 20500
TIMERT      DAC ACSAVE     /save AC as it was at time of interrupt
....
.....
.....
.....
process clock interrupt--this may involve resetting location 7 or even disabling the clock
....
.....
.....
LAC* (0     /pick up the return address from location 0
DAC RETURN#
....
.....
.....
other instructions are necessary here so that when we leave TIMERT the system looks as it did before the interrupt occurred.
JMP* RETURN /go back to where we were interrupted
```
REAL TIME CLOCK

To check for an interval of 1 second via interrupts under the API system, the following sequence may be used:

`LOC 2
TIMER /address of clock routine

LOC 51
JMS* 2

LOC 10200

LAW -74 /set clock counter to -60 ticks
DAC* 7
CLON /enable clock, clear clock flag
LAC (400000) /enable interrupts
ISA /from the API system

continue with 1 second's worth of program

LOC 20500

TIMER 0

DAC ACSAVE# /save AC as it was at time of interrupt

process clock interrupt

other instructions are necessary here so that when we return to the interrupted routine, the system looks as it did before the interrupt occurred.

JMP* TIMER /go back to where we were interrupted
NOTES

1. The clock continues to count up from zero after overflow occurs. At overflow detection, however, the clock counter is usually reinitialized or the clock is disabled.

2. To enable the clock:

   Use the CLON instruction (make sure the console clock clock switch is OFF--front down; if the console is locked, then CLON will enable the clock no matter what position the console clock switch is in).

3. To disable the clock:

   a) Use the CLOF instruction
   b) Turn the console CLOCK switch ON (rear half depressed; this will have an effect only if the console is not locked).

4. Depressing the RESET switch on the console clears the clock’s flag and disables the clock.
PROGRAM TO MOVE AC LIGHTS CENTER, OUT, BACK ETC

.TITLE PROGRAM TO MOVE AC LIGHTS CENTER, OUT, BACK

LOC 10100
START IOF
     CLA IFFL
     ISA

SETUP
     LAC (000200)
     DAC LH#
     LAC (000200)
     DAC RH#

/ MANUFACTURE THE DISPLAYED ACCUMULATOR VALUE
/ BY TAKING EACH HALF AND ROTATING IT IN THE APPROPRIATE
/ DIRECTION. THEN "XOR" THE HALVES FOR THE FULL VALUE.

FORM
     LAC LH
     MOVEL RAR
     LAC LH
     LAC RH
     MOVER RAL
     DAC RH
     XOR LH
     DAC DISPLAY#

/ SET UP THE REAL TIME CLOCK COUNTER LOCATION 7

TIMER
     LAW = 74
     DAC* (7)

/ ONE SECOND INTERVAL

/ EJECT
PROGRAM TO MOVE AC LIGHTS CENTER, OUT, BACK ETC

/TURN CLOCK ON (FIRST TIME THRU) AND CLEAR CLOCK FLAG. 
/SUBSEQUENT USES OF "CLON" ARE TO CLEAR THE FLAG ONLY. 
/THE CLOCK KEEPS ON COUNTING AFTER IT OVERFLOWS TO 0.

10121 A 210140 A  LAC DISPLAY
10122 A 700044 A  CLON
10123 A 700001 A  CLSF
10124 A 610123 A  JMP .-1

/KEEP ROTATING UNTIL THE AC=400001 OR AC=001400. 
/WHEN AC EQUALS THESE VALUES CHANGE THE DIRECTION 
/OF ROTATION. THIS IS DONE BY ACTUALLY CHANGING 
/THE INSTRUCTION "RAL" TO "RAR" AND VICE versa. IT 
/IS ACCOMPLISHED USING THE "XOR" INSTRUCTION AS 
/RAL=740010 AND RAR=740020. SWITCHING BACK AND 
/FORTH IS JUST A MATTER OF XORING WITH 000030. 
/(THERE ARE OF COURSE OTHER WAYS TO SWITCH!)

10125 A 550146 A  SAD (400001
10126 A 610132 A  JMP CHANGE
10127 A 550147 A  SAD (001400
10130 A 741000 A  SKP
10131 A 610107 A  JMP FORM
10132 A 210110 A  CHANGE LAC MOVEL
10133 A 250150 A  XOR (000030
10134 A 050110 A  DAC MOVEL
10135 A 250150 A  XOR (000030
10136 A 050113 A  DAC MOVER
10137 A 610107 A  JMP FORM

700044 A  CLON=700044
700001 A  CLSF=700001

010100 A  .END START

10143 A 000200 A *L
10144 A 000200 A *L
10145 A 000007 A *L
10146 A 400001 A *L
10147 A 001400 A *L
10150 A 000030 A *L

SIZE=10151 NO ERROR LINES
.TITLE NON_GLOBAL_SUBROUTINE_CALLS

700401 A    TSF=700401
700406 A    TLS=700406

00000 R    700002 A
START       IOP
00001 R    705514 A
            ISA+10
00002 R    700416 A
            TLS+10

00003 R    200043 R
            LAC (TABLE
00004 R    000042 R
            DAC_PTR
00005 R    777772 A
            LAW =6
00006 R    000041 R
            DAC_COUNT
00007 R    100033 R
            JMS_CRLF

00010 R    220042 R
MORE        LAC* PTR
00011 R    100026 R
            JMS_PRINT
00012 R    100033 R
            JMS_CRLF

00013 R    440042 R
            ISZ_PTR
00014 R    440041 R
            ISZ_COUNT
00015 R    600010 R
            JMP_MORE
            .EXIT

00020 R    000060 A
TABLE       60
00021 R    000061 A
            61
00022 R    000062 A
            62
00023 R    000063 A
            63
00024 R    000064 A
            64
00025 R    000065 A
            65

00026 R    000000 A
            PRINT 0
00027 R    700401 A
            TSF
00030 R    000027 R
            JMP =1
00031 R    700406 A
            TLS
00032 R    620026 R
            JMP* PRINT

00033 R    000000 A
            CRLF 0
00034 R    760015 A
            LAW 15
00035 R    100026 R
            JMS_PRINT
00036 R    760012 A
            LAW 12
00037 R    100026 R
            JMS_PRINT
00040 R    620033 R
            JMP* CRLF

00043 R    000020 R *L
            .END START
SIZE=00044   NO ERROR_LINES
GLOBAL SRC
GLOBAL SUBROUTINE CALLS

.TITLE GLOBAL SUBROUTINE CALLS

700401 A TSF=700401
700406 A TLS=700406

.GLOBL CRLF,PRINT

00000 R 700002 A START -IDP
00001 R 705514 A ISA+10
00002 R 700416 A TLS+10

00003 R 200032 R LAC+TABLE
00004 R 400027 R DAC PTR#
00005 R 777772 A LAH =6
00006 R 400026 R DAC CNTN
00007 R 120030 E JMS= CRLF

00010 R 220027 R MORE LAC+ PTR
00011 R 120031 E JMS= PRINT
00012 R 120030 E JMS= CRLF

00013 R 440027 R ISZ PTR
00014 R 440026 R ISZ CNTN
00015 R 600010 R JMP MORE

.EXIT

00020 R 000000 A TABLE 00
00021 R 000001 A 11
00022 R 000002 A 62
00023 R 000003 A 63
00024 R 000004 A 64
00025 R 000005 A 65

000000 R .END START
00030 R 000030 E *E
00031 R 000031 E *E
00032 R 000020 A *L

SIZE=00033 NO ERROR LINES
SUBROUTINE TO PRINT OUT CR AND LF

.GLOBL CRLF, PRINT

CRLF
0
LAW 215
JMP# PRINT
LAW 212
JMP# PRINT
JMP* CRLF

.END

NO ERROR LINES

SUBROUTINE TO PRINT CHAR ON TT

.GLOBL PRINT

700401 A TSP=700401
700406 A TLS=700406

.CHARACTER IS EXPECTED IN THE AC

PRINT 0
TSP
JMP =1
TLS
JMP* PRINT

.END

NO ERROR LINES
DOS-15 V3A000
$A LP -12
$K ON
$MACRO

BMACRO-15 V3A000
>BLG=NOGLOB
END OF PASS 1
SIZE=00044 NO ERROR LINE
+C

DOS-15 V3A000
$LOAD

BLOADER V3A000
>P=NOGLOB
P NOGLOB SRC 77573
+S+S
0
1
2
3
4
5

---------------------------
DOS-15 V3A000
$MACRO

BMACRO-15 V3A000
>BLG=GLOBAL
END OF PASS 1
SIZE=00033 NO ERROR LINES

BMACRO-15 V3A000
>BL=CRLF
END OF PASS 1
SIZE=00007 NO ERROR LINES

BMACRO-15 V3A000
>BL=PRINT
END OF PASS 1
SIZE=00005 NO ERROR LINES

DOS-15 V3A000
$LOAD

BLOADER V3A000
>P=GLOBAL,CRLF,PRINT
P GLOBAL SRC 77604
P CRLF SRC 77575
P PRINT SRC 77570
+S+S
0
1
2
3
4
5

84
PROGRAM TO AVERAGE DECIMAL VALUES

.TITLE PROGRAM TO AVERAGE DECIMAL VALUES

.PROGRAM ACCEPTS DECIMAL VALUES FROM THE KEYBOARD,
SUMS THEM AND PRINTS OUT THE DECIMAL AVERAGE (GIVEN
TO THE TENTHS PLACE) ON THE TELEPRINTER.
USER SHOULD FOLLOW EACH VALUE WITH A COMMA; TERMINATE
THE LINE WITH CR. FOR EXAMPLE: 3,16.29,4,(CR)

BEGIN  DBA
IOF
ISA+10

TLS+10
/DINIT PRINT==GET FLAG
/CONT# # OF VALUES
/FINAL# SUM OF VALUES
JMS= CRLF
/ISSUE CR AND LF
/TEMPORARY LOCATION

KSF

JMP -=1

KRS

SAD (215

JMP ALLDUN

SAD (254

JMP DUN

AND (17

GET OCTAL NUMBER

DAC TEMP#

LAC NUMB

MUL

12

LACO

/TRESULT SMALL..IN MO

TAD TEMP

/ADD ON LAST DIGIT

DAC NUMB

/SAVE IN NUMB

JMP NEXT

GET THIS VALUE

/DADD IT TO THE SUM

GET TRACK OF HOW MANY VALUES

85
```
ALLDUN LAC COUNT
DAC +3
LAC FINAL
IDIV 0
DAC REMAIN
DAC QUOT

/ GET # OF VALUES
/ STORE FOR DIVISION
/ GET SUM
/ REMAINDER RETURNED IN AC
/ GET QUOTIENT IN MQ

CLX LAC QUOT
IDIV 12
DAC DGT.X
DAC LACO
SNA
JMP DUNN
AXR 1
JMP NEXT
JMP DUNN

/ STORE IN TABLE
/ PICK UP QUOTIENT
/ CONTINUE IF NOT 0
/ STOP WHEN QUOTIENT = 0
/ PUT IN TABLE FOR LATER OUTPUT

ISSUE CR AND LF

/ TO MAKE ASCII

/ GET NEXT CHARACTER—WORK
/ WAY BACK UP TABLE
/ ARE WE DONE—WHEN XR=0, YES

ISSUE DECIMAL POINT "."

WANT TO CONTINUE OR EXIT?
/ IF SWITCHES ARE 0, THEN EXIT
/ NON-ZERO—CONTINUE WITH ANOTHER

EXIT

/ BLOCK 5

NO ERROR LINES
```
PROGRAM INTERRUPT FACILITY (PI)

References: Volume 1 Processor Handbook 5-11
System Reference Manual 3-8

Preface: The Program Interrupt Facility increases the efficiency of input/output operations by freeing a program from the necessity of constantly monitoring device flags. When PI is enabled and a peripheral device becomes available or completes a transfer, the PI automatically interrupts the program sequence and causes a "JMS 000000" to occur. A subroutine at location 000000 may then sense the device flags to determine which of the devices caused the interrupt, service the device, and return to the main program.

The running time of programs using input and output routines is primarily made up of the time spent waiting for an I/O device to accept or transmit information. Specifically, this time is spent in loops such as:

```
TSF /SKIP ON FLAG
JMP -.1
```

Waiting loops waste a large amount of computer time. In those cases where the computer can be doing something else while waiting, these loops may be removed and useful routines included to use the waiting time. This sharing of a computer between two tasks is often accomplished through the program interrupt facility, which is standard on all PDP-15 computers. The program interrupt facility allows certain external conditions to interrupt the computer program. It is used to speed the processing of I/O devices or to allow certain alarms to halt program execution and initiate another routine.

Each of the input/output devices has associated with it a device flag which is set to 1 whenever the device has completed a transfer and is ready for another. When the Program Interrupt Facility is enabled, the setting of the device flag (connected to PI) causes a program interrupt request. When PI is disabled, program interrupts do not occur, although device flags may be set.

When the interrupt is granted, PI is disabled automatically, the main instruction sequence is suspended and the hardware executes a "JMS 000000". This causes the contents of the Program Counter (the address of the next instruction that was to be executed) to be stored in location 000000 and the instruction in location 000001 to be executed.

The routine entered due to the interrupt is responsible for finding and servicing the device that caused the interrupt. Usually, the instruction in location 000001 is a JMP to a sequence of code called a SKIP CHAIN which determines which device's flag caused the interrupt and then jumps into a service routine for that specific device.
PROGRAM INTERRUPT FACILITY

The individual service routine then handles the condition causing the interrupt, reenables the Program Interrupt system and resumes mainline program execution by JMPing to the location pointed to by location 000000.

The IOT instructions used to program the PDP-15 for program interrupts are:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ION</td>
<td>700042</td>
</tr>
<tr>
<td>IOF</td>
<td>700002</td>
</tr>
<tr>
<td>RES</td>
<td>707742</td>
</tr>
</tbody>
</table>

- ION: Interrupt ON
- IOF: Interrupt OFF
- RES: Restore...i.e. set up for the restoration of the Link, Page/Bank mode, Memory Protect bit from the pointer word given in the next indirectly referenced instruction.

Use of the interrupt system allows a mainline routine, referred to as the BACKGROUND PROGRAM, to execute without wasting a large amount of time in waiting loops while I/O devices devices are assembling and transmitting information. The interrupt service routine, called a FOREGROUND PROGRAM, is entered automatically whenever an I/O device requires servicing under program control.
REQUESTING AN INTERRUPT

Device Flag Raised

Is Flag Attached to PI?

- NO
  - no interrupt request

- YES
  - Interrupt Request

--- requests for program interrupts are made when flags are raised for devices tied to PI.

GRANTING AN INTERRUPT

Interrupt Request

Is PI On?

- NO
  - no interrupt; keep request on line

- YES
  - Done With An Instruction?
    - NO
    - Execute Another
    - YES
      - Was It Privileged?
        - YES
          - PI Interrupt Granted*
        - NO

--- when the CPU receives a request for an interrupt, it must decide if that interrupt can be granted.

PI interrupts will be granted only
1) if PI is on
2) between instructions
3) after a non-privileged instructions (privileged instructions: IOTs, JMS, CAL, XCT, NORM)

*NOTE: The following have priority over PI:
1) Data Channel Transfers
2) Clock breaks for updating location 000007
3) API hardware interrupts
CPU PROCESS OF AN INTERRUPT

--- an interrupt consists of:
1) disabling PI
2) executing a JMS 000000

Recall that information is also stored in bits 0, 1, and 2 on a JMS. Location 000000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Link at time of interrupt</td>
</tr>
<tr>
<td>1</td>
<td>Page/Bank Mode Indicator</td>
</tr>
<tr>
<td>2</td>
<td>Memory Protect Indicator</td>
</tr>
</tbody>
</table>

The word in location 000000 has the following format:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link</td>
<td>Page Bank</td>
<td>Memory Protect</td>
<td>15 bit address of the instruction that was to be executed at time of interrupt</td>
<td></td>
</tr>
</tbody>
</table>
Single Device Interrupt Programming

When programming a system with only one possible source of interrupts, say the paper tape reader, the handling of an interrupt is very straightforward and simple.

0 000000
1 JMP PTREAD

PTREAD DAC ACSAVE /save registers used by the service routine

... code to handle the reader
... -was it the completion of a read or an error condition that caused the interrupt
... -store away the character read from the tape
... -check to see if there is more to read
... if so...initiate the next read

EXIT LAC ACSAVE /restore registers
ION /reenable PI
RES /set up to restore the Link, Page/Bank
/mode, Memory Protect bit on the next
/indirectly referenced instruction from
/bits 0,1 and 2 of the pointer word

JMP* 000000 /go back to where we left off

Multiple Device Interrupt Programming

Many programming applications use the interrupt system to service several devices. For example, a PDP-15 may use the interrupt facility to control the operation of paper tape (reader and punch) through a teletype. Systems of this type require a service routine that determines the source of an interrupt request (i.e., which device flag is set). The following instruction sequence uses dummy IOT Skip instructions to determine which device requested an interrupt:

D1SF /is it Device 1?
SKP /no
JMP D1SRV /yes--go to Device 1 service routine

D2SF /is it Device 2?
SKP /no
JMP D2SRV /yes--go to Device 2 service routine

...

DnSF /is it Device n?
JMP ERR /no--not device 1-n, go to error routine
JMP DnSRV /yes--go to Device n service routine
PROGRAM INTERRUPT FACILITY

For example, suppose we have a PDP-15 system with high speed paper tape reader and punch, teletype and clock and that is all. The following gives a skip chain that could be used.

0 000000
1 JMP SKPCHN

...MAINLINE ROUTINE...

SKPCHN CLSF /did the clock cause the interrupt?
SKP /no
JMP CLOCK /yes--go to clock service routine
/
KSF /did the keyboard cause it?
SKP /no
JMP KYBD /yes--go to keyboard service routine
/
TSF /did the teleprinter?
SKP /no
JMP TPRINT /yes--go to the teleprinter service routine
/
RSF /did the paper tape reader?
SKP /no
JMP PTREAD /yes--go to reader service routine
/
PSF /did the paper tape punch?
JMP ERROR /no--illegal interrupt occurred--go to error routine
JMP PUNCH /yes--go to paper tape punch service routine

CLOCK DAC ACSAVE ...
LAC ACSAVE
ION
RES
JMP* 000000
/
KYBD DAC ACSAVE ...
JMP* 000000
/
TPRINT DAC ACSAVE ...
JMP* 000000
/
PTREAD DAC ACSAVE ...
JMP* 000000
/
PUNCH DAC ACSAVE ...
JMP* 000000
PROGRAM INTERRUPT FACILITY

An interrupt grant will cause the computer to perform the following operations automatically:

1. The PI system is disabled.
2. The contents of the PC is stored at memory location 000000.
3. The "JMP SKPCHN" in location 000001 is executed.
   (note that steps 2 and 3 are the equivalent of executing "JMS 000000")

The SKPCHN routine then determines the source of the interrupt and passes control to the appropriate device handler.

The device handler (interrupt service routine) then performs the following operations:

1. The contents of the Accumulator (and any other registers which will be used) is saved.
2. The interrupt is processed --
   -determine whether flag was raised due to completion of transfer or an error condition
   -store data transferred in and clear flag
   -determine if more is to be input (if so, initiate it)
   -determine if more data is to be output (if so, initiate it)
   -clear flag
3. Restore the Accumulator (and any other registers used and therefore saved).
4. Turn the interrupt system back on (if further interrupts are to be allowed).
5. Set up for the restoration of the Link, Page/Bank mode, Memory Protect mode. The "RES" instruction primes the system for this restoration, although it does not actually occur until the next indirectly referenced instruction is executed (and then it is done using the contents of bits 0, 1 and 2 of the pointer word).
6. Return to the mainline program via a "JMP* 000000" instruction (recall that the updated PC was stored in location 000000).
NOTES

1. Instructions like CLSF, KSF and PSF are skip-on-flag instructions. There are Skip IOTs for every device in the interrupt system. Because of the predominance of skip instructions in the instruction sequence which determines the source of an interrupt request, it is often called a SKIP CHAIN.

A skip chain may be enlarged to test for almost any number of device flags, provided that high-speed devices which retain information for a relatively short period of time are tested near the top of the skip chain, so that the chain may be traversed and the high-speed devices serviced before the information is lost. High-speed devices should never be required to wait for service while a long skip chain is traversed.

Notice that the order in which the Skip IOTs are placed in the skip chain actually determines the priority of a device. If two devices have their flags raised simultaneously, the device whose Skip IOT appears closest to the top of the skip chain will be serviced first.

2. It is possible that the SKIP CHAIN will not be in page 0 or bank 0, in which case a "JMP SKPCHN" instruction in location 1 won't allow you to get there. Instead:

```
0 000000
1 JMP* 2
2 SKPCHN
```

will allow you to get to SKPCHN because a 15 bit address is picked up from location 2.

3. Similarly, it may be that the individual device service routines will not be located in the same page or bank as the SKIP CHAIN, and therefore will have to be entered indirectly:

```
SKPCHN  KSF
        SKP
        JMP* VKB
        TSF
        SKP
        JMP* VTP
        JMP ERROR

VKB     KYBD
VTP     TPRINT
```

4. With some devices, error condition flags set to 1 will also generate interrupts. It is therefore the service routine's responsibility to determine if the interrupt was caused by the completion of a transfer or by the existence of an error condition. For example, an interrupt may be caused by the paper tape reader when:

a) it has read a character and has assembled it in its buffer
b) it has attempted to read the tape but finds a no tape condition.

Some error conditions may be checked using the IORS instruction. In other cases, devices have their own status registers indicating errors (e.g. MT, DK, DP, DT).
AUTOMATIC PRIORITY INTERRUPT (API)

References: Volume 1 Processor Handbook 6-46
System Reference Manual 3-10

Overview: The Automatic Priority Interrupt system option increases the capability of the PDP-15 to handle transfers of information to and from input-output devices. API identifies an interrupt device directly, without the need of a SKIP CHAIN routine for flag checking. Multi-level interrupts are permissible where a device of higher priority supersedes an interrupt already in progress. These functions increase the speed of the input-output system and simplify the programming. In this way devices (especially high speed devices) can be serviced efficiently.

The API option increases the I/O handling capabilities of the PDP-15 by adding eight levels of priority servicing (0 - 7) and associating 32 channels with these eight levels. The highest four levels of priority, i.e. 0, 1, 2, 3 are assigned to hardware devices. The lower four levels, i.e. 4, 5, 6, 7 are for software purposes.

Of the 32 API channels, 4 are assigned to the software levels 4 - 7. The remaining 28 channels are available for use by the hardware levels 0 - 3. Each of the four hardware levels may have eight devices (channels) tied to it, up to the total of 28 for the four levels. This is strictly a hardware limitation imposed by cable lengths and circuit delays, and attempts to circumvent this restriction will create needless problems.

Each of the 32 channels is assigned to a specific memory location called the Break Address. The break addresses are locations 40 - 77 in page 0, bank 0. Each device tied to API is associated with a specific channel (and therefore break address) and a specific priority level. The table below gives the standard assignments. The channel assignments should remain fixed for software compatibility, but the suggested priority level may be changed (re-wiring needed) at the discretion of the user.
<table>
<thead>
<tr>
<th>API Channel</th>
<th>Break Address</th>
<th>Standard Device</th>
<th>Suggested Priority Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>40</td>
<td>Software channel 0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>41</td>
<td>Software channel 1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>42</td>
<td>Software channel 2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>Software channel 3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>DECtape (TC15)</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>45</td>
<td>MagTape (TC59)</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>46</td>
<td>Paper Tape Reader (PC15)</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>47</td>
<td>Clock Overflow (KW15)</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>Power Fail (KF15)</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>51</td>
<td>Graphics (VT15/VP15)</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>52</td>
<td>Card Readers (CR15/CR03B)</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>53</td>
<td>Line Printer (LP15)</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>54</td>
<td>A/D (AD15/AF01)</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>55</td>
<td>DB99A/DB98A</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>56</td>
<td>Data Phone (DP09A)</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>57</td>
<td>DECdisk (RF15)</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>58</td>
<td>Diskpack (RP15)</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>59</td>
<td>Plotter (XY15)</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>60</td>
<td>Scanners (DC01-ED) as needed</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>use 70-77</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>61</td>
<td>UDC15</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>62</td>
<td>ADC15</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>63</td>
<td>LT19 &amp; LT15 Teleprinter</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>64</td>
<td>LT19 &amp; LT15 Keyboard</td>
<td>3</td>
</tr>
<tr>
<td>25</td>
<td>65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>67</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>69</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>77</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Each device, when granted service via the API facility, sends its specific break address to the computer. This address, which will normally contain a JMS instruction to the device service routine, will then be executed by the computer. This type of interrupt service eliminates the need for time consuming flag search routines, and extensive core use for interrupt handling routines, by automatically determining which device requested service and providing immediate entry to the proper service routine.

Higher priority devices will be able to interrupt lower priority routines upon sending and having a request granted. The priority of devices multiplexed on the same priority level is determined by the relative position of the devices on the I/O bus. The first device on the bus having highest priority at that level, the second having second highest priority, etc.

The entire API facility can be enabled or disabled by a single IOT instruction. There is no way to enable or disable specific priority levels. However, for some devices there are instructions to disconnect itself from the API facility.

In addition to the above, there are two special features in the API facility. These are:

a. The CAL Instruction - Execution of a CAL instruction with the API facility enabled automatically sets priority level 4 thereby shutting out software requests of a lower priority until this level is released.

b. Program Interrupt - A program interrupt, from any I/O device connected to the computer, sets priority level 3. This occurs whether or not the API facility is enabled. This causes all devices on priority level 3, all software requests and program interrupts to be shut out until the level is released.

Special care must be taken in the programming of the API option to take account of these two features.
REQUESTING AN API INTERRUPT - requests for API interrupts are made when device flags, which are tied to the API system, are raised.

GRANTING AN INTERRUPT - when the CPU receives a request for an interrupt, the CPU must decide if that interrupt can be granted under API.

*NOTE: The CPU decides if it can grant an API interrupt in response to the API request. The following have priority over API:
1) Data Channel Transfers
2) Clock breaks for updating location 000007.
PROCESSING AN API INTERRUPT - in processing an API interrupt, the CPU
1) sets a priority level bit to inhibit interrupts from channels of the same or lower level of priority
2) picks up the break address sent by the device
3) does a hardware forced XCT of the contents of the break address

The IOT instructions used to program the API system are:

DBK 703304 - Debreak
- Reset the highest priority level bit so that operations may be carried out on same or lower level.

RES 707742 - Restore
- Set up to restore the link, page/bank mode and memory protect mode on next indirect instruction.

DBR 703344 - Debreak and Restore

ISA 705504 - Initiate Selected Activity
- Used to initiate software level interrupts and to raise the priority level of an operating program.
SINGLE DEVICE INTERRUPT PROGRAMMING

When programming a system with only one possible source of interrupts, say the paper tape reader, the handling of an interrupt is straightforward and simple.

When an API interrupt is granted to the Paper Tape Reader, it sends to the CPU its break address (50). The contents of location 50 is "XCT"ed. Since an XCT instruction does not change the PC, the PC that gets stored in location PTREAD is the updated PC from the mainline program (location M2). The interrupt is processed by PTREAD and it does a debreak to release level 2 (its priority level) and then JMPs back to the mainline program using location PTREAD as a pointer.
MULTIPLE DEVICE PROGRAMMING

When there is more than one device attached to API, handling an interrupt simply requires setting up the associated break address with a JMS to the interrupt service routine.

50  JMS  PTREAD
51  JMS  CLOCK
52  JMS  PWRFL

PTREAD  
DAC  ACSAV1

LAC  ACSAV1
DBR
JMP*  PTREAD

CLOCK  
DAC  ACSAV2

LAC  ACSAV2
DBR
JMP*  CLOCK

PWRFL  
DAC  ACSAV3

LAC  ACSAV3
DBR
JMP*  PWRFL

With this system there is no need for polling because when a device is granted an interrupt, it sends to the CPU its associated break address, which can then contain a JMS to a routine to handle that device.

While in the service routine for one device, a higher priority interrupt may be granted. This presents no problem because of the manner in which return addresses are stored.
In this example, a level 2 device has break address N while a level 1 device has break address M. The routine to service the level 2 interrupt is LEV2RT and the routine to service the level 1 interrupt is LEV1RT.

When the mainline routine is interrupted by the level 2 device between instructions in B1 and B2, the PC is pointing to B2. The level 2 device sends its break address of N to the CPU so that the JMS LEV2RT instruction may be "XCT"ed. This causes the address B2 (the contents of the PC) to be stored in location LEV2RT and control passed to location LEV2RT + 1. If while we are at priority 2 in routine LEV2RT an interrupt is requested by a device at level 1 that interrupt can be granted because of its higher priority. Suppose it is granted between instructions at LV2A and LV2B. Then the PC contains the address LV2B. The level 1 device sends its break address of M to the CPU so that the JMS LEV1RT instruction may be executed. This causes the address LV2B to be stored at location LEV1RT and control passed to location LEV1RT + 1. The level 1 routine processes the level 1 interrupt. When it executes the JMP* LEV1RT instruction control is passed back to the level 2 routine at the point we left off, LV2B. The LEV2RT routine may now resume its operation. When the JMP* LEV2RT instruction is executed, we go back to the mainline routine at location M2 where we left off and continue from there.
RELOCATION RULES

A. IF ADDRESS IS A NUMBER (NOT A SYMBOL) THE ADDRESS IS ABSOLUTE.

B. IF THE ADDRESS IS A SYMBOL WHICH IS DEFINED BY A DIRECT ASSIGNMENT STATEMENT (I.E., =) AND THE RIGHT-HAND SIDE OF THE ASSIGNMENT IS A NUMBER, ALL REFERENCES TO THE SYMBOL WILL BE ABSOLUTE.

C. IF A USER LABEL OCCURS WITHIN A BLOCK OF CODING THAT IS ABSOLUTE, THE LABEL IS ABSOLUTE.

D. VARIABLES, UNDEFINED SYMBOLS, EXTERNAL TRANSFER VECTORS, AND LITERALS GET THE SAME RELOCATION AS WAS IN EFFECT WHEN "END" WAS ENCOUNTERED IN PASS 1.

E. IF THE LOCATION COUNTER (.LOC PSEUDO-OP) REFERENCES A SYMBOL WHICH IS NOT DEFINED IN TERMS OF AN ABSOLUTE ADDRESS, THE SYMBOL IS RELOCATABLE.

F. ALL OTHERS ARE RELOCATABLE.

EXAMPLE OF RELOCATION RULES FOR LINKING LOADER

GLOBL SUB

000005 A A=5
000000 R 000005 A =START
00000 R 200005 A START LAC A
00001 R 200010 A LAC 10
00002 R 200020 R LAC B
00003 R 200030 R LAC START
00004 R 220010 E LAC* SUB
00005 R 200011 A LAC C102
00006 R 200012 A LAC (START

00005 R .LOC START+5
00005 R 200010 A START1 LAC 10
00006 R 200015 A LAC START1
00007 R 000000 R START

00000 A .LOC 0
00000 A 200000 R LAC START
00001 A 200005 R LAC START1
00002 A 200020 A START2 LAC START2
00003 A 000000 R START

000002 A .END
00010 A 00001 E *
00011 A 00010 A *
00012 A 00000 R *
SIZE=00013 NO ERROR LINES
.C

DOS-15 V3A000
$A LP -12

$K ON

$PAGE ON

$LOAD

LOADER V3A000
>P=RELOC
P RELOC SRC 77614
+S+Q

DOS-15 V3A000
$DUMP

DUMP V3A000
>77614-77637

DUMP V3A000
> +C

DOS-15 V3A000
$BANK ON

$LOAD

BLOADER V3A000
>P=RELOC
P RELOC SRC 77614
+S+Q

DOS-15 V3A000
$DUMP

DUMP V3A000
>77614-77637
PROGRAM SHOWING RELOCATION ELEMENTS

.LAC SYMBOL
.LAC BN
.DAC SYMBOL+1
.DAC B
.LAC SYMBOL
.DZM 11
.LAC 10
.LAC 10
.LAC (END
.LAC (END-SYMBOL
.DAC C

PAGE LOAD

77614=77637

77610 000000 000000 000000 000000 000000 000000 207627 207633 047630 047633
77620 207627 160011 200010 200010 207635 207636 047634 005000
77630 000000 000004 000005 000000 000000 077632 000003 000762

BANK LOAD

77614=77637

77612 000000 000000 000000 000000 000000 000000 217627 217633 057630 057633
77622 237627 160011 220010 200010 217635 217636 057634 005000
77632 000000 000004 000005 000000 000000 077632 000003 000762

105
OS-15 V3A000
IP
DOSPIP V3A000

> L TI COPIA BIN ← DTI

06-MAR-75
DIRECTORY LISTING
346 FREE BLKS
66 USER FILES
110 SYSTEM BLKS
COPIA BIN 455 1

> L TI COPIA BIN ← DK (P)

06-MAR-75
DIRECTORY LISTING (PES)
2202 FREE BLKS
32 USER FILES
34 USER BLKS
COPIA BIN 1567(2) 1 06-MAR-75 1567 100

> tC

DOS-15 V3A000
$A DTI -14

S A TI -12

$DUMP

DUMP V3A000
455#

455#

0 015000 646531 262601 000004 000005 000070 071033 412450
10 034150 074623 230204 400000 000000 000004 040404 000010
20 000000 000000 040404 001005 000001 000000 040404 000000
30 002004 000010 015000 317540 050404 000025 777736 000004
40 040404 000012 002005 000011 050404 000025 777736 000005
50 040302 000012 600010 000067 040710 000000 407716 022600
60 230710 000025 420564 052033 050000 520247 230710 000067
70 474741 071640 232700 000010 000010 000011 001005 776773

100 TO 367 CONTAINS 000000
370 000000 000000 000000 000000 000000 000000 000000

777777

106
.IODEV 4,5

.INIT 4,0,0
CAL+0*1000 4&777
1
0+0
0

.INIT 5,1,0
CAL+1*1000 5&777
1
0+0
0

.START
.READ 4,2,BUFF,34

0010 R 002004 A *G
CAL+2*1000 4&777
10

0012 R 000025 R *G
BUFF
*G
.DEC
-34

.WAIT 4

0014 R 000004 A *G
CAL 4&777

0015 R 000012 A *G
12

.WRITE 5,2,BUFF,34

0016 R 002005 A *G
CAL+2*1000 5&777
11

0020 R 000025 R *G
BUFF
*G
.DEC
-34

.WAIT 5

0022 R 000005 A *G
CAL 5&777

0023 R 000012 A *G
12

.JMP START

0024 R 00010 R

.BUFF

0025 R A

.BLOCK 42

0067 R 00000 A ENDMRK 0

00010 R

.END START
SIZE=00070
NO ERROR LINES
1C

DOS-15 V3A000
$A DK -14

$DUMP

DUMP V3A000
>1567#

1567#

0 015500 646031 262601 000004 000005 000070 071033 412450
10 034150 074623 230204 400000 000000 000004 040404 000001
20 000000 000000 004040 001005 000001 000000 040404 000000
30 002004 000010 015500 317040 050404 000025 777736 000004
40 040404 000012 002005 000011 050404 000025 777736 000005
50 040302 000012 600010 000067 040710 000000 407716 022600
60 230710 000025 420564 052033 005500 517547 230710 000067
70 474741 071640 232700 000010 000010 000011 001005 776773

100 000001 001567 000000 000000 000000 000000 000000 000000
110 TO 367 CONTAINS 00000
370 000000 000000 000000 000000 000000 000000 000000 777777 777777
<table>
<thead>
<tr>
<th>x--</th>
<th>-x--</th>
<th>--x--</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>003100</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>006200</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>011300</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>014400</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>017500</td>
<td>E</td>
</tr>
<tr>
<td>F</td>
<td>022600</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>025700</td>
<td>G</td>
</tr>
<tr>
<td>H</td>
<td>031000</td>
<td>H</td>
</tr>
<tr>
<td>I</td>
<td>034100</td>
<td>I</td>
</tr>
<tr>
<td>J</td>
<td>037200</td>
<td>J</td>
</tr>
<tr>
<td>K</td>
<td>042300</td>
<td>K</td>
</tr>
<tr>
<td>L</td>
<td>045400</td>
<td>L</td>
</tr>
<tr>
<td>M</td>
<td>050500</td>
<td>M</td>
</tr>
<tr>
<td>N</td>
<td>053600</td>
<td>N</td>
</tr>
<tr>
<td>O</td>
<td>056700</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>062000</td>
<td>P</td>
</tr>
<tr>
<td>Q</td>
<td>065100</td>
<td>Q</td>
</tr>
<tr>
<td>R</td>
<td>070200</td>
<td>R</td>
</tr>
<tr>
<td>S</td>
<td>073300</td>
<td>S</td>
</tr>
<tr>
<td>T</td>
<td>076400</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>101500</td>
<td>U</td>
</tr>
<tr>
<td>V</td>
<td>104600</td>
<td>V</td>
</tr>
<tr>
<td>W</td>
<td>107700</td>
<td>W</td>
</tr>
<tr>
<td>X</td>
<td>113000</td>
<td>X</td>
</tr>
<tr>
<td>Y</td>
<td>116100</td>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
<td>121200</td>
<td>Z</td>
</tr>
<tr>
<td>%</td>
<td>124300</td>
<td>%</td>
</tr>
<tr>
<td>.</td>
<td>127400</td>
<td>.</td>
</tr>
<tr>
<td>0</td>
<td>132500</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>135600</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>140700</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>144000</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>147100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>152200</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>155300</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>160400</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>163500</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>166600</td>
<td>9</td>
</tr>
<tr>
<td>#</td>
<td>171700</td>
<td>#</td>
</tr>
</tbody>
</table>
PAGE 1  PROG1  SRC
00000 R  A
000000 A
SIZE=02000
.BLOCK 2000
.END
NO ERROR LINES

PAGE 1  PROG2  SRC
00000 R  A
000000 A
SIZE=07500
.BLOCK 7500
.END
NO ERROR LINES

PAGE 1  PROG3  SRC
00000 R  A
000000 A
SIZE=00500
.BLOCK 500
.END
NO ERROR LINES

PAGE 1  PROG4  SRC
00000 R  A
000000 A
SIZE=07740
.BLOCK 7740
.END
NO ERROR LINES

PAGE 1  PROG5  SRC
00000 R  A
000000 A
SIZE=05000
.BLOCK 5000
.END
NO ERROR LINES

PAGE 1  PROG6  SRC
00000 R  A
000000 A
SIZE=00010
.BLOCK 10
.END
NO ERROR LINES
$LOAD

BLOADER V3A000
>PROG1, PROG2, PROG3, PROG4, PROG5, PROG6
P PROG1 SRC 75637
P PROG2 SRC 66137
P PROG3 SRC 65437
P PROG4 SRC 50040
P PROG5 SRC 68437
P PROG6 SRC 60427
+$T+C

DOS-15 V3A000
$PAGE ON

$LOAD

LOADER V3A000
>PROG1, PROG2, PROG3, PROG4, PROG5, PROG6
P PROG1 SRC 75637
P PROG2 SRC 66300
P PROG3 SRC 75137
P PROG4 SRC 50040
P PROG5 SRC 70137
P PROG6 SRC 70127
+$T+C

DOS-15 V3A000
$
In previous chapters, MACRO calling sequences have been given for OTS and Science Library Subprograms. This general form is used in a MACRO program to call any FORTRAN external subroutine or function. A FORTRAN program may also invoke MACRO subprograms. The method for each type of linkage is given below.

### INVOKING MACRO SUBPROGRAMS FROM FORTRAN

A FORTRAN program may invoke any MACRO program whose name is declared in a MACRO GLOBL statement. The MACRO subprogram must also include the same number of open registers as there are arguments. These will serve as transfer vectors for arguments supplied in the FORTRAN CALL statement or function reference. A FORTRAN-IV program and the MACRO subprogram it invokes are shown below.

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>MACRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>C TEST MACRO SUBR</td>
<td>.TITLE MIN</td>
</tr>
<tr>
<td>C READ A NUMBER(A)</td>
<td>.GLOBL MIN, .DA</td>
</tr>
<tr>
<td>1 READ(1,100)A</td>
<td>MIN 0</td>
</tr>
<tr>
<td>100 FORMAT(E12.4)</td>
<td>JMS* .DA</td>
</tr>
<tr>
<td>C NEGATE THE NUMBER</td>
<td>MIN1 .DSA 0</td>
</tr>
<tr>
<td>C AND PUT IT IN B</td>
<td>MIN2 .DSA 0</td>
</tr>
<tr>
<td>CALL MIN(A,B)</td>
<td>LAC* MIN1</td>
</tr>
<tr>
<td>C WRITE OUT NUMBER(B)</td>
<td>DAC* MIN2</td>
</tr>
<tr>
<td>WRITE(2,100)B</td>
<td>ISZ MIN1</td>
</tr>
<tr>
<td></td>
<td>ISZ MIN2</td>
</tr>
<tr>
<td></td>
<td>LAC* MIN1</td>
</tr>
<tr>
<td>STOP</td>
<td>RAL</td>
</tr>
<tr>
<td>END</td>
<td>CML</td>
</tr>
<tr>
<td></td>
<td>CML</td>
</tr>
<tr>
<td></td>
<td>STOP</td>
</tr>
<tr>
<td></td>
<td>RAR</td>
</tr>
<tr>
<td></td>
<td>DAC* MIN2</td>
</tr>
<tr>
<td></td>
<td>JMP* MIN</td>
</tr>
<tr>
<td></td>
<td>.END</td>
</tr>
</tbody>
</table>

/ entry/exit
/ general get
/ argument
/ (OTS)
/ jump around argument registers
/ ARG1
/ ARG2
/ first word of A
/ store at B
/ point to second word of A and B
/ second word of A
/ sign bit = 1
/ store in second word of B
/ exit
The FORTRAN statement CALL MIN(A,B) is expanded by the compiler to:

00013 JMS* MIN / to MACRO subprog
00014 JMP S00014
00015 .DSA A
00016 .DSA B
S00014 = 00017

When the FORTRAN-IV program is loaded, the addresses (plus relocation factor) of A and B are stored in registers 15 and 16, respectively. When the MACRO program invokes .DA, these addresses are stored in MIN1 and MIN2 and the values themselves are accessed by indirect reference.

Arguments are, as described above, transmitted by .DA using a single word. Bits 3-17 contain the 15-bit address of the first word. Bits 0-2 serve as flag. FORTRAN uses bit 0 to indicate that the word specifying the argument contains the address of a word containing the address of the first word of the argument. The MACRO argument word always contains the address of the first word of the argument. For array name arguments (unsubscripted), the address of the fifth word of the array descriptor block is given with bit 0 on.

For external functions, the MACRO subprogram must return with a value in the AC (LOGICAL, INTEGER), AC-MQ (DOUBLE INTEGER) or in the floating accumulator (REAL or DOUBLE PRECISION).

INVOKING FORTRAN SUBPROGRAMS FROM MACRO

The MACRO calling conventions for FORTRAN subprograms are: the name of the subprogram must be declared as global; there must be a jump around the argument address; and the number and mode of arguments in the call must agree with those of the subprogram. This form is shown below.

.TITLE MACPRG
.GLOBL SUBR
JMS* SUBR
JMP .+N+1 / jump around arguments ignored by .DA
.DSA ARG1 / address of first argument - bit 0 set to 1
.DSA ARG2 / indicates indirect reference
  ...
.DSA ARGN
...

When the subprogram is compiled, a call is generated to .DA which performs the transmission of arguments from MACRO. The beginning of a subroutine might be expanded as follows.
If a value is to be returned by the subroutine, it is most convenient to have this be one of the calling arguments. An external function is called in the same manner as a subroutine but returns a value in the AC (single integers), AC-MQ (double integers), or floating accumulator (real and double-precision). To store the AC, the MACRO program uses a DAC instruction. Values from the floating accumulator may be stored via the OTS routines .AH (real) and .AP (double-precision). For FPP systems, values are returned in a hardware accumulator and stored with an FST instruction.

**COMMON BLOCKS**

FORTRAN COMMON blocks (and block-data subprograms) may be linked to MACRO programs. When the MACRO program is loaded, global symbols are first sought in the user and system libraries. Any remaining are matched, where possible, to COMMON block names. This cannot be done if programs are loaded via CHAIN and EXECUTE. For example:

<table>
<thead>
<tr>
<th>FORTRAN</th>
<th>MACRO</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER A, B, C</td>
<td>.GLOBL NAME, .XX / .XX is name given to blank COMMON</td>
</tr>
<tr>
<td>COMMON/NAME/C</td>
<td>/ by the FORTRAN Compiler</td>
</tr>
<tr>
<td>COMMON A, B</td>
<td>DZM* .XX / CLEAR A - NOTE INDIRECT REFERENCE</td>
</tr>
<tr>
<td>.</td>
<td>ISZ .XX / BUMP COUNTER</td>
</tr>
<tr>
<td></td>
<td>DZM* .XX / CLEAR B</td>
</tr>
<tr>
<td></td>
<td>DZM* NAME / CLEAR C</td>
</tr>
</tbody>
</table>

Note that if the values are REAL (two words) or DOUBLE PRECISION (three words), the MACRO program must account for the number of words when accessing specific variables.

DOS-15 and RSX-PLUS MACRO programs may also use the .CBD pseudo-op. For instance

```
BASE1 .CBD NAME 1
```

will provide the base address of the common block NAME in the word that is created and labeled BASE1; the size of the common block is 1. For blank common, use for example:

```
BASE2 .CBD .XX 2
```
MACRO ROUTINE CALLING FORTRAN PROGRAM

.TITLE MACRO ROUTINE CALLING FORTRAN PROGRAM

.GLOBL FORT

PROGRAM REQUESTS USER TO INPUT THE # OF ELEMENTS TO
BE SUMMED IN ARRAY A (1 TO 18 ELEMENTS MAY BE SUMMED).
THE USER INPUTS A VALUE THAT IS ONE LESS THAN THE
ACTUAL NUMBER DESIRED. THE INPUT RANGE IS 0-9.
MACCAL CALLS FORT TO ADD 1 TO THE NUMBER TYPED.
FORT IN TURN CALLS MACEX TO DO THE SUMMING.

IN=0
OUT=1

START .INIT =2, IN, RST
.INIT =3, OUT, RST

RST .WRITE =3, 2, MEG 1, 34
.WAIT =3

CHECK .READ =2, 3, BUFF, 3
.WAIT =2

CHECK

CK1 LAC BUFF+2
TAD (=60)
DAC BUFF+2

CK2 SPA
JMP ERRMES
TAD (=12)
SMA
JMP ERRMES

CK3 LAC BUFF+2
DAC COUNT

CK4 JMS FORT
JMP .+2
.DSA COUNT

JMP RST

EXIT

ERRMES .WRITE =3, 2, MESSERR, 34
.WAIT =3
JMP CHECK

COUNT 0
MESERR = MES1 / 2 * 1000

.ASCII / TYPE IN ONE LESS THAN THE NUMBER OF ELEM

.ASCII / YOU WOULD LIKE SUMMED (0-9) /

.ASCII <15>

MESERR = BUFF = MESERR / 2 * 1000

.ASCII / VALUE IS OUT OF RANGE. /

.ASCII / TYPE '0-9' ONLY/<15>
PAGE 3  MACCAL SRC  MACRO ROUTINE CALLING FORTRAN PROGRAM

00141 R 640262 A
00142 R 064000 A
00143 R 000000 A

00144 R 002000 A  BUFF  ENDOC=BUFF/2*1000
00145 R 000000 A  0
00146 R  A  .BLOCK 2

000150 R  ENDOC=
000000 R  .END

00150 R 000150 E  *E
00151 R 777720 A  *L
00152 R 777760 A  *L

SIZE=00153  NO ERROR LINES
FORTRAN CALLING MACRO EXAMPLE

This routine is called by the macro routine "MACCAL". "MACCAL" passes one argument, N, to FORT.

FORT adds 1 to it and then...

This FORTRAN routine calls MACRO program SUM to perform summation of array A.

Then the sum is printed out after TOT.

SUBROUTINE FORT(N)

INTEGER A(10), TOT

A(1) = 0
A(2) = 0
A(3) = 0
A(4) = 0
A(5) = 0
A(6) = 0
A(7) = 0
A(8) = 0
A(9) = 0
A(10) = 0

N = N + 1
CALL SUM(A, N, TOT)
WRITE(6, 1) TOT

FORMAT (1H , 'TOT= ', 1I10)
RETURN
END
FORTRAN CALLING MACRO EXAMPLE

THIS ROUTINE IS CALLED BY THE MACRO ROUTINE "MACCAL". "MACCAL" PASSES ONE ARGUMENT, N, TO FORT.
FORT ADDS 1 TO IT AND THEN...

THIS FORTRAN ROUTINE CALLS MACRO PROGRAM SUM
TO PERFORM SUMMATION OF ARRAY A.
THEN THE SUM IS PRINTED OUT AFTER TOT.

SUBROUTINE FORT(N)

INTEGER A(10), TOT

A(1) =
CMAICLA
TAD 000001
TAD A
DAC 000007
LAC (000005
S0007 = 00153
DAC* XIA

A(2) =
CMAICLA
TAD 000002
TAD A
DAC 000015
LAC (000006
S00015 = 00153
DAC* XIA

A(3) =
CMAICLA
TAD 000003
TAD A
DAC 000023
LAC (000007
S00023 = 00153
DAC* XIA

A(4) =
CMAICLA
TAD 000004
TAD A
DAC 000031
LAC (000010
S00031 = 00153
DAC* XIA

A(5) =
CMAICLA
TAD 000005
TAD A
DAC 000037
LAC (000011
S00037 = 00153
DAC* XIA

A(6) =
CMAICLA

125
00043 TAD (000006
00044 TAD A
00045 DAC 800045
00046 LAC (000012
00047 = 00153
00047 DAC+ XIA
020 A(7)=11
00050 CMAICLA
00051 TAD (000007
00052 TAD A
00053 DAC 800053
00054 LAC (000013
00054 = 00153
00055 DAC+ XIA
021 A(8)=12
00056 CMAICLA
00057 TAD (000010
00058 TAD A
00059 DAC 800061
00059 LAC (000014
00059 = 00153
00060 DAC+ XIA
022 A(9)=13
00061 CMAICLA
00062 TAD (000011
00063 TAD A
00064 DAC 800067
00064 LAC (000015
00064 = 00153
00065 DAC+ XIA
023 A(10)=14
00066 CMAICLA
00067 TAD (000012
00068 TAD A
00068 DAC 800075
024 C
00069 LAC (000016
00069 = 00153
00070 DAC+ XIA
025 NNN+1
00100 LAC+ N
00101 TAD (000001
00102 DAC+ N
026 CALL SUM(A,N,TOT)
00103 JMS+ SUM
00104 JMP 00110
00105 ,DSA 400000 +A
00105 ,DSA 400000 +N
00107 ,DSA TOT
027 WRITE(6,1) TOT
00110 JMS* FWIN
00111 ,DSA (000006
00112 ,DSA .1
028 C
00113 ,DSA 777777
00114 JMS* ‘FE
00115 ,DSA TOT
00115 JMS* ‘FP
029 1 FORMAT(1H,"TOT=",10)
00117 JMP 500117

126
.TITLE MACRO FROM FORTRAN EXAMPLE

PROGRAM IS CALLED BY FORTRAN ROUTINE TO
SUM ELEMENTS OF AN ARRAY—IT RETURNS THE
SUM IN A LOCATION WHOSE ADDRESS IS PASSED.

.GLOBL SUM, DA

SUM 0
JMS+ DA
JMP +4
ARG1 0 /ADDRESS OF ARRAY A
ARG2 0 /ADDRESS OF LOCATION CONTAINING VALUE N
ARG3 0 /ADDRESS OF LOCATION WHERE SUM IS RETURNED

LAC ARG2
TCA
DAC ARG2

CLA
TAD ARG1
ISZ ARG1
ISZ ARG2
JMP ADDER
DAC ARG3
JMP SUM

.END

SIZE=0020
NO ERROR LINES
**UPDATE**

DOS-15 V3A000
$R UPDATE

**CREATING A LIBRARY WITH UPDATE**

***DAT***
-15 Device   UIC   USE
-14 DKA      PES   OUTPUT
-12 ITA      PES   INPUT
-10 ITA      PES   LISTING

$A LP -12/DK -10

$K ON

$UPDATE

UPDATE V3A000
>NL<USERLB
> I CRLF
> CLOSE

UPDATE V3A000
>NL<LIBR5
> I CRLF
> I PRINT
> CLOSE

UPDATE V3A000
>NL<MAIN
> I AVGLOB
> I CRLF
> I PRINT
> CLOSE

UPDATE V3A000
>TC

DOS-15 V3A000
$

**129**
DOS-15 V3A000
$PAGE ON

LOAD

.DEVICE USE
-5 NON USER LIBR
-4 DKA USER PROG(S)
-1 DKA SYS LIBR

SA DK -5

SK ON

LOAD

LOADER V3A000
>P<AVGLOB,CRLF,PRINT
ALL THREE FILES SPECIFIED.
P AVGLOB SRC 77503
P CRLF SRC 77474
P PRINT SRC 77467
$S+5S
3,4,6,
4,3
1,2,3,
2,0

DOS-15 V3A000

LOAD

LOADER V3A000
>P<AVGLOB
ONLY ONE FILE SPECIFIED.
P AVGLOB SRC 77503
P CRLF SRC 77474
P PRINT SRC 77467
$S+5S
3,4,5,
4,0

DOS-15 V3A000

LOAD

LOADER V3A000
>P<MAIN
THE FILE SPECIFIED IS A LIBRARY.
P AVGLOB SRC 77503
P CRLF SRC 77474
P PRINT SRC 77467
$S+5S
3,4,5,
4,0

DOS-15 V3A000

LOAD

LOADER V3A000
>P<AVGLOB,PRINT
TWO FILES (PROGRAMS) ARE SPECIFIED.
P AVGLOB SRC 77503
P PRINT SRC 77476
P CRLF SRC 77467
$S+5S
3,4,5,
DOS-15 V3A000
$LOAD

LOADER V3A000
>\$=AVGLOB,USERLB,PRINT ➔THREE FILES ARE SPECIFIED.
P AVGLOB SRC 77503 ➔THE FILE AVGLOB CONTAINS ONE PROGRAM (AVGLOB)
P CRLF SRC 77474 ➔THE USER LIBRARY USERLB CONTAINS ONE
P PRINT SRC 77467 ➔PROGRAM --CRLF--.
  SIS
3,4,5,
4.0

DOS-15 V3A000
$
BATCH
(NON—BOSS)

References:  DOS USERS GUIDE  8-30
DOS KEYBOARD COMMAND GUIDE  11

DOS-15 V3A000
$L

*****************************************************************************

PREPARING A BATCH STREAM

*****************************************************************************

$EDIT

EDITOR V3A000
>OPEN BATSTR
FILE BATSTR SRC NOT FOUND.
INPUT
$JOB
PIP
L LP ← DK <SCR>
L LP FILBLK BIN ← DK (P)
$JOB
A LP -12
MACRO
BLG+ME
$JOB
GLOAD
P+ME
$EXIT
EDIT
>EXIT

DOS-15 V3A000
$L

*****************************************************************************

GETTING THE BATCH STREAM OUT ON PAPER TAPE

*****************************************************************************

$PIP

DOSPPIP V3A000
>T PP ← DK BATSTR SRC

132
RUNNING THE BATCH STREAM... PLACE THE PAPER TAPE IN THE READER

BATCH PR

DOS-15 V3A000
$JOB
PIP
DOSPIP V3A000

$L LP ← DK <SCR>

$LP FILBLK BIN ← DK (P)

$JOB
DOS-15 V3A000
$A LP -12

$MACRO
MACRO-15 V3A000
$LG=ME
END OF PASS 1
SIZE=00025 NO ERROR LINES
MACRO-15 V3A000
$JOB
DOS-15 V3A000
$GLOAD
LOADER V3A000
$P=ME
P ME SRC 77612
P LPA.15 049 77050
DOS-15 V3A000
$SEXIT

DOS-15 V3A000
$

133
DISK FILE FORMAT FOR DOS:

1ST BLK
DATA

2ND BLK
DATA

3RD BLK
DATA

1ST BLOCK

2ND BLOCK

3RD BLOCK

-1
PROGRAM TO LIST ON TT BLOCKS IN A FILE ON DAT 5

.TITLE PROGRAM TO LIST ON TT BLOCKS IN A FILE ON

PROGRAM LIST ON THE TELETYPING THE BLOCKS USED BY A FILE
STORED ON THE DEVICE ASSOCIATED WITH DAT 5.

USER MUST KNOW THE STARTING BLOCK NUMBER (GIVEN BY PIP
ON DT DIRECTORY OR OBTAINED BY USING THE (P) SWITCH
FOR DIRECTORY ON DISK., [L TT & DK (P)]

WHEN PROGRAM HALTS, PLACE THE NUMBER OF THE FIRST BLOCK
USED BY THE FILE IN THE DATA SWITCHES AND THEN HIT THE
CONTINUE SWITCH.

.IODEV 5

START OBA

.INIT =3,1,0

.INIT 5,0,0

.GET BLOCK NUMBER AND FILL IT IN APPROPRIATE WORD OF
.TRN EXPANSION.

.READ IN BLOCK NUMBER FROM DATA SWITCHES

.PUT IN .TRAN EXPANSION BLOCK # POSITION

.PRINT OUT BLOCK #

.DO .TRAN

.NAHT 5

.CHECK THE LAST WORD IN THE BLOCK JUST READ IN.
.IF THIS WORD # 777777, THEN THE BLOCK JUST READ IS THE
.IN THE FILE.
.IF THIS WORD NOT # 777777, THEN IT IS THE NUMBER OF THE
.BLOCK USED BY THIS FILE.

.PICK UP WORD 377 IN BLK

.LAST FILE BLK(777777)?

.YES

.NO=PRINT OUT BLOCK #
PROGRAM TO LIST ON TT BLOCKS IN A FILE ON DAT 5

# SET HERE WHEN THERE ARE STILL MORE BLOCKS TO THE FILE.
# PICK UP THE NEXT BLOCK NUMBER FROM WORD 377 OF FILE.
# PLACE IT IN THE APPROPRIATE WORD IN THE .TRAN EXPANSION
# THEN GO DO THE .TRAN
#
# LAC BUFF+377
# DAC TRAN+2
# JMP TRAN
#
# /***************************************************************************/
# /CONVERT 6 OCTAL DIGITS TO 6 ASCII CHARACTERS
# /***************************************************************************/
# PRINT 0
# LMB
# CLX
# LAC 16
# PAL
# NXT CLA
# LLS 3
# TAD 160
# DAC BUFF+2,X
# AXS 1
# JMP NXT
#
# /***************************************************************************/
# /OUTPUT BLOCK #
# /************************************************************/
# .WRITE -3,3,BUFF,0
# .WAIT -3
# JMP Print
#
# /***************************************************************************/
# /ON END OF FILE; CLOSE DAT -3 AND RETURN TO MONITOR
# /***************************************************************************/
# ENOFIL .CLOSE =3
# .EXIT
# BUFF 000003
# 0
# .BLOCK 6
# 0031 R 000017
# 0032 R 000015
# 0033 R 000000 A
# 0034 R 052000 A
# 0035 R 735000 A
# 0036 R 200474 R
# 0037 R 722000 A
# 0038 R 750000 A
# 0039 R 400603 A
# 0040 R 344675 R
# 0041 R 000003 R
# 0042 R 344675 R
# 0043 R 000003 R
# 0044 R 725001 A
# 0045 R 000049 R
# 0054 R 020033 R
# 0055 R
# 0061 R 005003 A
# 0062 R 000000 A
# 0063 R A
# 0071 R 000015 A
# 0072 R 000012 A
# 0073 R A
# 0074 R 000000 A
# 0075 R 000000 A
# 0043 R 777777 A
# 00474 R 000000 A
# 00475 R 000000 A
# SIZE=00476 NO ERROR LINES

136
EXAMPLE OF ONE BLOCK OF A FILE:

<table>
<thead>
<tr>
<th>HEADER WORD 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEADER WORD 1</td>
</tr>
<tr>
<td>&quot;DATA&quot;</td>
</tr>
<tr>
<td>HEADER WORD 0</td>
</tr>
<tr>
<td>HEADER WORD 1</td>
</tr>
<tr>
<td>&quot;DATA&quot;</td>
</tr>
<tr>
<td>HEADER WORD 0</td>
</tr>
<tr>
<td>HEADER WORD 1</td>
</tr>
<tr>
<td>&quot;DATA&quot;</td>
</tr>
<tr>
<td>EMPTY</td>
</tr>
<tr>
<td>NOT USED (???)</td>
</tr>
<tr>
<td>NEXT BLOCK OR (-1)</td>
</tr>
</tbody>
</table>
BLOCKS 71 THROUGH 77 MAINTAIN INDIVIDUAL BIT MAPS FOR ALL FILES ON DECTAPE.

BIT MAP FILE RELATION:

THE INCLUSIVE "OR" OF ALL INDIVIDUAL BIT MAPS IS EQUIVALENT TO THE MASTER BIT MAP.
DECTAPE FILE STRUCTURE

DIRECTORY (BLK #100):

<table>
<thead>
<tr>
<th>MASTER BIT MAP</th>
<th>32 WORD DEC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE ENTRY TABLE</td>
<td>224 WORDS DEC.</td>
</tr>
</tbody>
</table>

THE MASTER BIT MAP MAINTAINS A COMPLETE BLOCK BY BLOCK RECORD OF A DECTAPE BY RELATING AN INDIVIDUAL BIT TO AN OCTAL BLOCK NUMBER: E.G.

BIT POSITIONS 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17
WORD 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 20 | 21
WORD 1 | 22 | 23 | 24 | 25 | 26 | 27 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 40 | 41 | 42 | 43
WORD 3 | 44 | 45 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | .

THE FILE ENTRY TABLE IS DIVIDED IN 4 WORD SEGMENTS, EACH SEGMENT DESCRIBES A FILE ON THAT DECTAPE.

E.G.

<table>
<thead>
<tr>
<th>WORD X</th>
<th>T</th>
<th>E</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORD X+1</td>
<td>T</td>
<td>@</td>
<td>@</td>
</tr>
<tr>
<td>WORD X+2</td>
<td>S</td>
<td>R</td>
<td>C</td>
</tr>
<tr>
<td>WORD X+3</td>
<td>400005</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"6" BIT FILE NAME

1ST BLOCK OF THAT FILE, MSB = ACTIVE
### Master File Directory (MFD)

<table>
<thead>
<tr>
<th>Offset (Hex)</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
<td>DUMMY WORD</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>OR POINTER TO BAD ALLOCATION TABLE</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>SYS BLK'S FIRST BLOCK NUMBER -1 IF NOT INITIAL MFD BLOCK</td>
</tr>
<tr>
<td>3</td>
<td>401776</td>
<td>ENTRY SIZE (0-2) PLUS POINTER TO STORAGE ALLOCATION TABLE</td>
</tr>
<tr>
<td>4</td>
<td>251103</td>
<td>(SIXBIT &quot;UIC&quot;)</td>
</tr>
<tr>
<td>5</td>
<td>54</td>
<td>POINTER TO USER FILE DIRECTORY</td>
</tr>
<tr>
<td>6</td>
<td>400010</td>
<td>PROTECTION CODE FOR THIS USER (Ø) AND FILE DESCRIPTION ENTRY SIZE</td>
</tr>
<tr>
<td>7</td>
<td>Ø</td>
<td>NOT USED</td>
</tr>
<tr>
<td>10</td>
<td>Ø4145</td>
<td>SIXBIT &quot;DLE&quot;</td>
</tr>
<tr>
<td>11</td>
<td>6Ø</td>
<td>POINTER TO UFD</td>
</tr>
<tr>
<td>12</td>
<td>400010</td>
<td>PROTECTION CODE AND FILE ENTRY SIZE</td>
</tr>
<tr>
<td>13</td>
<td>Ø</td>
<td>NOT USED</td>
</tr>
<tr>
<td>376</td>
<td>-1</td>
<td>POINTER TO PREVIOUS BLOCK</td>
</tr>
<tr>
<td>377</td>
<td>-1</td>
<td>POINTER TO NEXT BLOCK</td>
</tr>
</tbody>
</table>

**NOTES:**
- MFD = BLOCK #1777 if RF DISK, 47 Ø 4 Ø if RFO2
- PROTECTION CODE: 1 = Protected Directory, Ø = Unprotected
- ILLEGAL UFD'S: @@@, ???, and those that are current to the system - PAG, BNK, SYS, IOS.
<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ø3Ø114</td>
<td>SIXBT &quot;CALH@@SRC&quot;</td>
</tr>
<tr>
<td>1Ø2ØØØ</td>
<td></td>
</tr>
<tr>
<td>2322Ø3</td>
<td></td>
</tr>
<tr>
<td>ØØ1645</td>
<td>FIRST BLOCK OF THIS FILE (BIT Ø = TRUNCATION)</td>
</tr>
<tr>
<td>ØØØØ2</td>
<td>SIZE OF FILE IN BLOCKS</td>
</tr>
<tr>
<td>ØØ764</td>
<td>POINTER TO RIB BLOCK</td>
</tr>
<tr>
<td>2ØØ232</td>
<td>FILE PROTECTION CODE (Ø-2), START LOCATION OF RIB</td>
</tr>
<tr>
<td>1425Ø1</td>
<td>DATE FILE CREATED (12-21-71)</td>
</tr>
<tr>
<td>17Ø623</td>
<td>SIXBT &quot;OSCLKLST&quot;</td>
</tr>
<tr>
<td>Ø31413</td>
<td></td>
</tr>
<tr>
<td>142324</td>
<td></td>
</tr>
<tr>
<td>ØØ2045</td>
<td>FIRST BLOCK OF THIS FILE</td>
</tr>
<tr>
<td>ØØØ116</td>
<td>SIZE OF FILE IN BLOCKS</td>
</tr>
<tr>
<td>ØØ1643</td>
<td>POINTER TO RIB BLOCK</td>
</tr>
<tr>
<td>2ØØØØØ</td>
<td>FILE PROTECTION CODE (Ø-2) AND START LOCATION OF RIB</td>
</tr>
<tr>
<td>1425Ø1</td>
<td>DATE FILE CREATED (12-21-71)</td>
</tr>
</tbody>
</table>

**NOTES:**
- **PROTECTION CODE:** (Valid only if directory is protected)
  - 1 = Unprotected, 2 = Write Prot., 3 = R/W Prot.

**RIB:** The RIB may occupy its own block or, if room, occupy an area at the end of the file it is describing.

**TRUNCATION:** File was not closed.
The following patch corrects a problem in DUMP which outputs incorrect information on selective dumps.

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>OLD CONTENTS</th>
<th>NEW CONTENTS</th>
<th>NEW SYMBOLIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>16256</td>
<td>217406</td>
<td>617472</td>
<td>JMP PATCH</td>
<td>/Patch area</td>
</tr>
<tr>
<td>17472</td>
<td>-</td>
<td>116034</td>
<td>JMS DEVICE</td>
<td>/Device check</td>
</tr>
<tr>
<td>17473</td>
<td>-</td>
<td>217406</td>
<td>LAC (-1)</td>
<td>/Restore inst.</td>
</tr>
<tr>
<td>17474</td>
<td>-</td>
<td>616257</td>
<td>JMP BACK</td>
<td>/Return</td>
</tr>
<tr>
<td>17224</td>
<td>106400</td>
<td>206400</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DOS-15 VIA
$MICLOG SYS

$DUMP

DUMP V9A
↑C

$R PATCH

.DAT    DEVICE  UIC  USE
-14     DKA      SYS  I/O - SYS DEV
-10     TTA      SYS  SECONDARY INPUT

$PATCH

PATCH V19A
>DUMP
>16256
>16256/217406>617472
  16257/057172>
>L 17472
>17472/215116>116034
  17473/253512>217406
  17474/055116>616257
  17475/214753>
>L 17224
>17224/106400>206400
>EXIT

DOS-15 VIA
$DUMP

DUMP V9B
>
FIRST PRINTING, FEBRUARY 1974

THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION. DIGITAL EQUIPMENT CORPORATION ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT MAY APPEAR IN THIS DOCUMENT.

THE SOFTWARE DESCRIBED IN THIS DOCUMENT IS PURCHASED TO THE PURCHASER UNDER A LICENSE FOR USE ON A SINGLE COMPUTER SYSTEM AND CAN BE COPIED (WITH INCLUSION OF DIGITAL'S COPYRIGHT NOTICE) ONLY FOR USE IN SUCH SYSTEM, EXCEPT AS MAY OTHERWISE BE PROVIDED IN WRITING BY DIGITAL.

DIGITAL EQUIPMENT CORPORATION ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY DIGITAL.

COPYRIGHT (c) 1974, BY DIGITAL EQUIPMENT CORPORATION

>EJECT
COPYRIGHT 1970, DIGITAL EQUIPMENT CORP., MAYNARD, MASS.

/PRB.
/PRB=IDPSP PAPER TAPE READER HANDLER = IDP SP ASCII Y.
/M = SPNS/SP.MURPHY
/7=30-68
/CALLING SEQUENCE:
/INIT
/CAL*.DAT SLOT (9=17)
/1
/0
/READ
/CAL*.DM (8=8)*.DAT SLOT (9=17)
/10
/LINE BUF ADDR,
/LOC OF L.B. (2'S COMP)
/WAIT
/CAL*.DAT SLOT (9=17)
/12

00000 A MED=3
700101 A RSP=700101
700102 A RCF=700102
700112 A RR0=700112
700114 A RSR=700114
700144 A RSB=700144

00000 R 040340 R
00001 R 040341 R
00002 R 440341 R
00003 R 220341 R
00004 R 440341 R
00005 R 340422 R
00006 R 040014 R
00007 R 700314 A
00010 R 700100 A
00011 R 777749 A
00012 R 340180 R
00013 R 040052 R
00014 R 740040 A
00015 R 600031 R
00016 R 740000 A
00017 R 000056 R
00020 R 000027 R
00021 R 000027 R
00022 R 600045 R
00023 R 600052 R
00024 R 600075 R
00025 R 600027 R
00026 R 600066 R
00027 R 760306 A
00030 R 620425 R

.GLOBL PRB
.DAC PRCALP /CAL POINTER
.DAC PRCARGP /ARG POINTER
.ISZ PRCARGP /INDEX TO FUNCTION ADDR
.LAC PRCARGP /FUNCTION
.ISZ PRCARGP /INDEX TO NEXT ARGUMENT
.TAO (JMP PRTABL
.DAC PRTABL
.IORS /SET IONESOF SWiCH
.S Miks AS FUNCTION OF ITS STATE
.LAW 17748 /ON ENTRY INTO CAL LEVEL
.TAD PROCM
.DAC PROBK
.PRTABL XX

JMP PRIN
NOP
JMP PRSEEK
JMP PROG
JMP PRWAIT
JMP PROBK
JMP PREP
JMP PRG
JMP PRWATR

/INIT PTR ROUTINE
PRIN ISZ PRAKGP
PRH PRAKGP
LAC (64)
DAC PRAKGP
ISZ PRAKGP
CAL 50
PARER PTRINT
PRUNA LAC =#2
PTRNC DAC =#6
PRCHAR JMP PROSTOP
/STOP PTR ROUTINE, CLEARING I/O SWITCH
PROSTOP DZH PRUNA
/CLEAR I/O UNDERWAY INDICATOR
/PTR WAIT
PRWAIT LAC PRUNA
SNA
JMP PROBK
PRBUZY LAC PRCALP
DAC PRAKGP
PROBK XX
OBR
XCT +1
JMP* PRAKGP
/EXIT
PRNOR*PROBK
PRSEEK ISZ PRAKGP
PROBKI JMP PROBK
PRWAIT LAW 1000
AND* PRCALP
SNA
JMP PRWAIT
/WAIT
LAC PRCALP
/WAITR
AND (700000)
/LS, XM, HP
DAC PRCALP
LAC* PRAKGP
AND (77777)
DAC PRCALP
DAC PRCALP
ISZ PRAKGP
JMP PRWAIT
/TO NO BUSY
/PTR READ ROUTINE
PRRED LAC PRUNA
SZAICMA /NO START IT UP
JMP PRBUSY
/YES=WAIT IN BUSY LOOP BACK TO C
/START UP PTR
PRSTRT DAC PRUNA
/LSET I/O UNDERWAY SWITCH (777777)
DAC PROBKI
/EXIT TO USER-MAINSTREAM
DAC PRTOUT
DAC PRAKGP
DAC MPBHP
/LS, B.H. POINTER IN CALL
DAC PRAKGP
ISZ PRAKGP
DAC PRAKGP
/LS, B.H. POINTER
DAC PTRWC  /=L.B.W.C' (2'S COMP)
ISZ PRARGP  /INDEX TO POINT TO EXIT
LAW 7000  /CHECK FOR IOPS ASCII MODE
AND PRCALP  /ILLEGAL DATA MODE
SAD (2000)
JMP .+3  
LAW 7  
JMP +.MED+1  
LAC (1002)  
DAC PROTCT  
/IDPS ASCII DATA MODE AN
/NO PAIR CT. OF
/1 FOR HEADER.
JMS PRNXWD  
PR1CR LAC 17773  
/INDEX PAST L.B. HEADER
JMS PRNXWD  /FOR IOPS ASCII
PR2CR DAC PTR57  
/5/7 CHAR
DZM PRCT  
/COUNTER
DZM PRCT  
/CLEAR CHAR CT.
DZM PRCT  
/CLEAR ASCII 8TH BIT SET COUNTER
DZM PRER  
/CLEAR PARITY ERR. SWITCH
DZM 8ZA  
PROUT2 IORS  
AND (1000)  
/SEQ.
JMP PREDM  
/REV.
IOP  
/1 FOR CURSOR
R8A  
PROUT XX  
/JMP PRDK OR JMP PRDIH
/ptr interrupt service
PRINT JMP PTRPIC  /PIC ENTRY
DAC PTRAC  /API ENTRY, SAVE AC
LAC PTRINT  /PIC OR API, L, EM, MP
DAC PROTCT  
IORS  
SMACL A  
LAW 17740  
/PIC OFF
TAD PRION  
/PIC ON
JMP PRSION  
PTRPIC DAC PTRAC  /SAVE AC
LAC (0 /PIC-PC, L, EM, MP
DAC PROTCT  /SAVE FOR EXIT
DAC PRION  
PRSION DAC PRSW  
/READ PTR BUFFER
RRB  
PRION ION  
DAC PRCHAR
LAC PROSMI  
DAC PRTRUT  
LAC PRUND  
/SUCCESSFUL STOP SINCE LAST READ
SNA  
PROSMI JMP PROISM  
/I.O.K.  
/IGNORE LAST READ, STOP
IORS  
AND (1000)  
/SPI NOT READY? (IORS8+1)
SNA  
JMP PRIGA  

/END OF PAPER TAPE ROUTINE

/END LINE

/PROCESS IOPS ASCII

/SYTH BIT#1, ADD TO COUNT

/SYTH BIT#1, ADD TO COUNT

/SYTH BIT#1, ADD TO COUNT

/NOT EVEN PARITY

/CONVERT ALTMODES

/DROP ALL BUT 7 BIT8

/DELETE CODE (RUBOUT) = IGNORE

/PACK INTO L.B. IN 5/7
0022 R 740100 A
0023 R 660131 R
0024 R 200340 R
0025 R 540440 R
0026 R 600124 R
0027 R 200344 R
0028 R 540415 R
0029 R 600265 R
002A R 750000 A
002B R 100353 R
002C R 600257 R
002D R 200347 R
002E R 600338 R
002F R 200345 R
0030 R 840346 R
0031 R 600277 R
0032 R 200340 R
0033 R 740200 A
0034 R 200441 R
0035 R 260036 R
0036 R 600306 R
0037 R 100324 R
0038 R 751101 A
0039 R 600285 R
003A R 340337 R
003B R 400337 R
003C R 777408 A
003D R 520037 R
003E R 340442 R
003F R 000037 R
0040 R 440037 R
0041 R 220036 R
0042 R 500443 R
0043 R 740200 A
0044 R 600131 R
0045 R 200443 R
0046 R 600275 R
0047 R 000000 A
0048 R 440037 R
0049 R 440042 R
004A R 620317 R
004B R 600265 R
004C R 200000 A
004D R 200043 R
004E R 500034 R
004F R 540433 R
0050 R 760015 A
0051 R 540444 R
0052 R 760175 A
0053 R 540445 R

SMA
JMP PROUT2
/LAST ASCII CHAR
LAC PRCCT
SAD (1)
JMP PR1CR
/IGNORE SINGLE CR LINE
SAD PR5CNT
/WORD COUNT ALL SET.
JMP PRASE
LAC PTR57
CLAVE
/WORD PAIR
JMP PRPAD
/JMP LAST
PRPAD
PRASE
LAC PRTCT
/LAST CHAR COUNT (INCL: HDR.)
DAC+ PRLBHP
/LD = 0, L-OH,
LAC PRTCT
/DO ALL CHAR'S HAVE BIT 8
SAD PRCCT
/NO = IOPS ASCII CHECK PARITY
JMP PRASE3
/YES = ASSUME NON IOPS ASCII
PRASE2
LAC PASER
/Parity Error
SZA
/NO
LAC (20 /YES
PRASE4
XOR+ PRLBHP
/Parity
DAC+ PRLBHP
/ERROR INDICATOR,
JMP PRAE3
/Skip to End Line
SZA
LCLAICMA
PRASE3
JMP PR1BB
/C-R FOUND - EXIT,
TAD PRDBP
/POINTS TO LAST CHAR
DAC PRDBP
LAE 17400
AND+ PRDBP
XOR (33)
/put CR in LAST WORD PAIR
DAC+ PRDBP
ISZ PRDBP
/LAC PRDB+ BEFORE CR
AND (0)
SZA
JMP PROUT2
/Validity Bits already set
LAC (00
/Line buffer overflow.
JMP PRASE4
PRNW
PRASE
ISZ PRDBP
/Index to NEXT DATA WORD
ISZ PTRNC
/Index Word Count
JMP* PRNW
/EXIT FOR NEXT CHAR
JMP PRASE
/EXIT TO END OF IOPS ASCII LINE
/End Line Test - Converts ALTMODE to Standard 175
PREN1
LAC PRCHAR
AND (177
SAD (1)
/return
LAM 19
SAD (175
/LTTHMODE
SAD (176
/LTTHMODE
LAW 175
SAD (35) \ESCAPE
LAW 175
JMP* PRENDT

*/VARIABLES NOT SAVED-APPLY TO CURRENT ACTIVE REQUEST
PRCALP 0  /SCAL POINTER
PRARGN 0  /ARG, LIST AND EXIT POINTER
PRTGAC 0  /SAVED AC IN INTERRUPT
PROUT 0  /PC, L, EM, MP
PRTB7 0  /CHAR, POSITION COUNTER IN 5/7 PAIR
PRACT 0  /ASCII=WITH=8TH=BIT=SET=CHAR COUNTER
PRECT 0  /CHAR CT.
PROTCT 0  /DATA WORD PAIR IN LINE COUNTER
PRENT 0  /PARITY CHECK COUNTER
PRENT1 0  /1 BIT COUNTER FOR PARITY CHECK
PRSW 0  /SIGN OR IP

*/7 IOPS ASCII PACKING ROUTINE
PR67 IS INITIALIZED TO 777777
PRIOR TO THE 1ST CALL.

PRPK67 0  /CHAR, IN AC BITS 11-17.
RTR  /MOVE TO AC BITS 9-6
RTR
RTR
DAC PRTMP
LAW 17771
DAC PRLPCT
/P-

PRPKBK LAC PRTMP  /ROTATE CHAR LEFT
RAL 7 BITs THROUGH
DAC PRTMP  /THE DOUBLE WORD
DAC PRLHF
LAC PRLHF
LAC PRLPCT
SNA1CLL
JMP PRPDNE  /2 WORDS ALL SET.
ISZ PRLPCT  /IS, 7 TIMES COUNT EXHAUSTED?
NO.
DO WE HAVE 5 CHARS?

PRPDNE LAC PRLPCT  /NO.
JMP* PRPK57  /SHIFT LEFT ONCE MORE.
JMP PRLHF
PLACE ACCUMULATED
DAC* PRDLP  /2 WORDS INTO
JMS PRRWXD  /USER'S LINE BUFFER,
LAC PRRTHF  /UPDATING POINTERS.
DAC* PRDLP
LAC PRLPCT  /INCREMENT
TAD (1000)  /DATA WD, PAIR
DAC PROTCT
JMS PRNXWD
PRCNY LAW 17773
DAC PTR57
JMP* PRPK57
000350 R
PRTMP*PRCNY
000351 R
PRLPCT*PRCNY1
000000 R
PRLFHF 0
000000 R
PRRTMF 0
.END

COUNT
RESET 5 CHAR COUNTER
TEMP* STORAGE FOR 5/7 CHAR
ROTATE 7 BITS COUNTER
7/2 WORD ACCUMULATOR FOR
5/7 WORD PAIR.

SIZE=00446 NO ERROR LINES

152
<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>154</td>
</tr>
<tr>
<td>UNICHANNEL 15 HARDWARE ARCHITECTURE</td>
<td>155</td>
</tr>
<tr>
<td>UNICHANNEL 15 SOFTWARE ARCHITECTURE</td>
<td>158</td>
</tr>
<tr>
<td>MEMORY LAYOUT</td>
<td>162</td>
</tr>
<tr>
<td>PIREX TASKS</td>
<td>163</td>
</tr>
<tr>
<td>INTERRUPT LINK</td>
<td>167</td>
</tr>
<tr>
<td>MX15-B MEMORY MULTIPLEXERS</td>
<td>172</td>
</tr>
<tr>
<td>PDP-11/05 CONTROL REGISTERS</td>
<td>171</td>
</tr>
<tr>
<td>SYSTEM CONFIGURATION</td>
<td>174</td>
</tr>
<tr>
<td>SYSTEM RESTRICTIONS</td>
<td>175</td>
</tr>
<tr>
<td>PDP-15 UNICHANNEL OPTIONS</td>
<td>177</td>
</tr>
</tbody>
</table>
INTRODUCTION

This guide describes in more detail, the UNICHANNEL 15 operation and features presented in the RK15 Disk Cartridge System Option Bulletin.

The first section .... presents a look at the UC15 system architecture.

The second section.....describes the PI REX monitor system; how to use it and other software aids.

The final section.....provides, for those interested in creating their own programs, a complete hardware specification including IOT and register descriptions.

Supplementing this guide are two manuals:

   Unichannel 15 System Maintenance Manual:.....DEC-15-HUCMA-B-D


The maintenance manual describes the details of the MX15-B and the DR15-C logic and gives maintenance details.

The software manual describes the details of the PI REX Monitor.
The term UNICHANNEL was created because it emphasizes the union of Digital's UNIBUS with the big computer concept of the programmable I/O channel. UNICHANNEL 15 unites low cost, mass produced peripherals with big computer software and performance on the PDP-15.

UNICHANNEL 15 (UC15) is a peripheral processor for the PDP-15 utilizing the PDP-11/05 minicomputer. It provides the PDP-15 with a second general purpose processor and a second high speed I/O bus; the UNIBUS. This UNIBUS is an 18-bit pathway permitting transfer of either 18-bit words, 16-bit PDP-11 words, or two 8-bit bytes.

The UC15 allows flexible low cost configuration and expansion of PDP-15 systems.

The UC15 minimizes the peripheral processing load on the PDP-15 allowing maximum computational throughput in a low-priced, medium scale system.

UNICHANNEL 15 OPERATION

There are three major components of the UC15:

1. A PDP-11/05 computer with "local" PDP-11 memory.

2. An MX15-B memory multiplexer which allows both the PDP-15 processor and the PDP-11 processor to share common memory. The shared memory is ordinary 18-bit PDP-15 core memory.

3. An "interrupt link" to provide a real-time means of inter-processor communications.
Figure 2: Diagram of UC15 Hardware Interrelationships
SUMMARY - UNICHANNEL 15 HARDWARE ARCHITECTURE

This particular architecture was chosen because of its many advantages.......

PDP-15 Memory is addressable by the UNIBUS. Hence, DMA transfers from and to such secondary storage devices as disks are direct.

The interrupt link provides inter-processor signaling on a micro-second basis. This is ideal for efficient real-time service -- a necessity for flexible I/O control.

All PDP-15 systems may be upgraded by adding the UC15. All memory remains useable.

Cost is minimized by allowing the PDP-11 to share the PDP-15 console and paper tape loader system.

Maximum use of the PDP-15 memory is maintained through synchronization overlap with memory use by the MX15-B. This "pre set up" technique increases the number of memory cycles per second when both PDP-15 and PDP-11/05 are accessing the common PDP-15 memory.

The UNIBUS provided by the UC15 is electrically compatible with any device meeting UNIBUS interfacing specifications with the following restraints:

1. UNIBUS lengths must be kept short.

2. No provision is made for UNIBUS parity.

Data in the common PDP-15 memory may be treated as either 18 or 16 bit words or as (2) 8-bit bytes.

True simultaneous parallel processing is possible in the local and common memories.

The DMA rate is high and the worst case and average latencies are low for maximum I/O performance.

Finally, the system is highly modular allowing flexibility in configuration and excellent software utilization and control. The system permits variations in both local and common memory size. It allows almost any combination of PDP-15 and UNIBUS peripherals.
The hardware architecture is complimented by sophisticated system software. PDP-15 software systems running with a UNICHANNEL system relies on PIREX, a compact multitasking peripheral executive. In addition to PIREX, Digital supplies UNIBUS device drivers, UNICHANNEL compatible handlers, and supporting utility functions.

The software system used by UC15 consists of two parts:

1. One component is a multi-programming peripheral processor executive called PIREX and is executed by the PDP-11.

2. The other component is an operating system in a PDP-15. (e.g. DOS-15 or BOSS-15).

PIREX

PIREX is a multi-programming executive designed to accept any number of requests from a PDP-15 or PDP-11 and process them on a priority basis while processing other tasks concurrently. PIREX services all Input/Output requests from the 15 in parallel on a controlled priority basis. Requests to busy routines (called tasks) are automatically queued (entered into a waiting list) and processed whenever the task in reference is free. In a background environment, PIREX is also capable of supporting any number of priority driven software tasks initiated by the 15 or the 11 itself.

Figure 3 shows the communications flow in a UNICHANNEL system. The possible links which may exist in the system are as follows:

1. Handler to driver to allow the PDP-15 to use a UNICHANNEL device.

2. Handler to non-driver task to allow the PDP-11 to intercept output and manipulate it or store it for spooling.

3. Program to non-driver task to allow cooperative processing on the two CPU's as occurs in the use of the MAC-11 assembler.

158
Figure 3: DIAGRAM OF UNICHANNEL SOFTWARE SYSTEM
UNICHANNEL ADDRESSING

The Unichannel system makes use of a PDP-11 as an intelligent peripheral controller for the larger PDP-15 main computer. In order to effectively operate with a minimum of interference with the PDP-15, the PDP-11 uses its own LOCAL MEMORY of between 4K and 12K 16-bit words.

COMMON MEMORY is that memory directly accessible to both the PDP-15 and the PDP-11.

COMMON MEMORY occupies the upper portion of the PDP-11 address space and, at the same time, the lower portion of the PDP-15 address space.

NOTE:

PDP-11 LOCAL MEMORY +
PDP-15/PDP-11 COMMON MEMORY

must not exceed 28K.

DOS-15 requires a minimum of 16K of memory on the PDP-15. Therefore, the PDP-11 LOCAL MEMORY may be 12K or 8K or 4K.

NOTE: The PDP-11 is a byte oriented machine. The 8-bit bytes are numbered sequentially with two 8-bit bytes corresponding to one 16-bit word.

Thus--

<table>
<thead>
<tr>
<th>WORDS</th>
<th>BYTES</th>
<th>ADDRESSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K</td>
<td>8K</td>
<td>00000-17777</td>
</tr>
<tr>
<td>8K</td>
<td>16K</td>
<td>00000-17777</td>
</tr>
<tr>
<td>12K</td>
<td>24K</td>
<td>00000-57777</td>
</tr>
</tbody>
</table>

4K LOCAL MEMORY
24K COMMON MEMORY
ADDRESS CORRESPONDENCE

AD11 = (AD15 * 2) + LOCAL MEMORY size in bytes

AD15 = (AD11 - LOCAL MEMORY size in bytes) / 2

e.g. Address 7777 on the PDP-15 is address 37776 on the PDP-11 with 4K local memory
Address 60000 on the PDP-11 with 4K local memory is address 20000 on the PDP-15
MEMORY LAYOUT

Figure 4 details the memory map which exists on UNICHANNEL System. Note that both the 11 and 15 parts of the system can operate concurrently, all memory contention is resolved by the MX15-B. Note also, that if the 11 system operates with area "A" complete simultaneity is possible because no memory contention can occur.

```
UNIBUS ADDRESSES

Unavailable

124K

Available to DMA breaks

28K

"Shared" memory available to both CPU's

8 or 12K

Local 11 Memory

128K

PDP-15 ADDRESSES

D

116 or 112K

C

Available to PDP-15 and PDP-15 I/O BUS

8 or 16K

B

Shared Memory

0

A

Unavailable

"Local" PDP-11 Memory = A
"Shared" Memory = B
PDP-11 CPU Address Space = A + B
UNIBUS DMA Address Space = A + B + C
PDP-15 Address Space = B + C + D
```

Figure 4 : UNICHANNEL SYSTEM MEMORY MAP
PIREX TASKS

The PIREX software system consists of several routines to support multi-programming among tasks. These routines perform such functions as: context switching, node manipulation and scheduling. The tasks which execute in this environment are device drivers, directives to PIREX, or merely software routines which execute in a background mode.

Device drivers are tasks which typically perform rudimentary device functions (e.g.: read, write, search, process interrupts, etc.), Directives are tasks which perform some specific operation for a task under PIREX. The connecting and disconnecting tasks to/from PIREX are performed by the CONNECT and DISCONNECT directives. The third type of tasks are software routines which execute in a background mode of operation. The MACRO-ll assembler and Spooler are both run as background tasks.

To support multiprogramming among tasks, each task is required to have a format as shown in the figure below:

![Task Format Diagram]

Figure 5 TASK FORMAT
The execution of a Task by PIREX is accomplished by first scanning the Active Task List (ATL). The ATL is a priority-ordered linked list of all active Tasks in the current system currently capable of running. An Active Task is one which:

1. Is currently executing.
2. Has a new request pending in its deque (double ended queue).
3. Has been interrupted by a higher priority task.

When a runnable task is found, the stack area and general purpose registers belonging to the task are restored and program control transferred to it. Program execution begins at the first location of the task program code (See Figure 2.1) or at the point where the task was previously interrupted by a higher priority task. When a task is interrupted by other tasks, its general purpose registers and stack are saved. The ATL is rescanned when a new request is issued to a task or when a previous request is complete.

When the PIREX Software System is running, it is normally executing the NUL task (a PDP-11 WAIT Instruction); The NUL task is run whenever there are no requests pending, a task suspends itself in a wait state, or while all other tasks are waiting for I/O previously initiated. When the PDP-15 issues a request to the PDP-11 to be carried out by PIREX, it does so by interrupting the 11 at Level 7 (the highest PDP-11 Interrupt Level) and simultaneously passing it an address of a Task Control Block (TCB) through the interrupt Link.

An 11 task can issue requests via the IREQ MACRO. The contents of the TCB completely describe the request (task addressed, function, optional interrupt return address and level, status words, etc....) The TCB will usually reside in the PDP-15 memory and must be directly addressable by the 11. (i.e. It resides in shared memory).

Error conditions are passed back to the 15 in the Task Control Block (TCB) along with status information necessary for complete control and monitoring of a particular request. Usually the request is to a device on the 11 but other types are allowed.

Task Control Blocks are used for communication with PIREX and tasks running under it. The general format of a TCB consists of three words followed by optional words necessary for task communication. Optional words, generally are used to pass buffer addresses, commands and device status as may be appropriate.

TCB:    (API TRAP ADDRESS *400(8)) + API LEVEL
        (FUNCTION CODE *400(8)) + TASK CODE NUMBER

REV:    REQUEST EVENT VARIABLE
        (Optional Words)

Figure 6 STANDARD TCB FORMAT
The "TRAP ADDRESS" is a PDP-15 API trap vector and has a value between 0 and 377 (8). Location 0 here corresponds to location 0 in the PDP-15. The API Level is the priority level at which the interrupt will occur in the PDP-15 and has a value between 0 and 3. A 0 signifies API "Level" 0, a 1 for level 1 etc... The API trap address and level are used by tasks in the PDP-11 when informing the 15 that the requested operation is complete (e.g...a disk block transferred or line printed).

The Task code number is a positive number between 0 and 128 that tells PIREX which task is being referenced, (Tasks are addressed by a numeric value rather than by name).

The Function Code determining whether hardware interrupts are to be used at the completion of the request. If the code has a value of 0, an interrupt is generated at completion of the request; If a 1, an interrupt is not made.

The Request Event Variable, commonly called REV or just EV, is initially cleared by PIREX (set to zero) and then set to a value "n" (by the associated task) at the completion of the request. The values of "n" are:

\[
\begin{align*}
0 & = \text{request pending or not yet completed.} \\
1 & = \text{request successfully completed.} \\
-2 & = (\text{mod } 2\cdot16-1) \text{ non-existent task referenced.} \\
-3 & = (\text{mod } 2\cdot16-1) \text{ illegal API level given (illegal values are changed to level 3 and processed).} \\
-4 & = (\text{mod } 2\cdot16-1) \text{ illegal directive code given.} \\
-777 & = (\text{mod } 2\cdot16-1) \text{ request node was not available from the Pool, i.e. the POOL was empty, and the referenced task was currently busy or the task did not have an ATL node in the Active Task List.}
\end{align*}
\]

NOTE -- the Task Control Block specification clearly defines a modular communications structure with minimum impact on PDP-15 software.
ADDING DRIVERS TO PIREX

A powerful feature allows the PDP-15 to bring in a PDP-11 driver, (into either its own memory or the 11's local memory) connect it to PIREX via a connect directive (a disconnect directive is also provided) and then issue I/O requests through PIREX to the driver. The user can now take full advantage of the existing and future PDP-11 peripherals along with an elaborate queuing structure built into PIREX allowing complete parallel processing.

MACRO 11 ASSEMBLER (MAC11) AVAILABLE

A MACRO 11 Assembler is provided. This assembler is a Macro subset of the existing PDP-11 Macro assembler and is slightly modified to run under the control of DOS-15 and PIREX.

To accomplish this, the MACRO assembler (MAC11) is loaded by the 15 as a core image into bank 1 of the 15. MAC 11 is then connected up as a low priority driver to PIREX and requested to begin the assembly. The 11 then carries out the actual assembly while the 15 handles all of the opening and closing of files, reading and writing of test and object information until the assembly is complete. To the user at the console teletype, MAC 11 appears to be just a DOS-15 system program which is loaded in and run by the 15.

NOTE: That any customer developed software should of course, take into account PIREX context switch, the bandwidth of the UNIBUS 18 and latency consideration of the associated system.

SUMMARY

As one can easily see, the UC15 software system is a powerful tool to the user who requires the utmost in flexibility and utility. UC15 also provides an expansion capability beyond any system currently available.
INTERRUPT LINK

The following section describes the registers and control of the interrupt link. This link is used to pass Task Control Block Pointers (and through them the information in Task Control Blocks) between the PDP-15 and PDP-11 systems. The hardware which comprises this link consists of a DR15-C special purpose interface to the PDP-15, I/O BUS, and 2 DR11-C general purpose UNIBUS interfaces. The DR15-C is controlled by PDP-15 IOT's while the DR11's are accessed as registers on the UNIBUS.

Register Descriptions (PDP-11)

(CSR) 767770 Bit 6 - when bit 6 is a 1, it will enable an interrupt on BR5 to TV 300, if the API DONE flag is set in bit 7 of 767770.

Bit 7 - API DONE - set to 1 whenever none of the 4 API channels has a request pending.

 NOTE: Neither of these bits is expected to be used in normal systems programming.

(ODB) 767772 Low byte - contains the API address for an API level Ø break. Loading a new value in the byte causes the appropriate API flag to be set in the DR15-C and API break in the PDP-15 will occur, is the API is enabled and no higher activity is occurring. It also will cause a PI interrupt if API is not installed.

High byte - contains the API address for an API level 1 break. Same conditions as low byte.

(IDB) 767774 Bit Ø - contains bit "2" of the Task Control Block Pointer (TCBP). See note under bit 1.

Bit 1 - contains bit "1" of the TCBP.

 NOTE: That reading 767774 does not effect the new TCBP flag in bit 7 of 767760.

Bit 6 - API 2 DONE flag - when a 1 indicates that there is no API level 2 request pending before the PDP-15. When a 1 also indicates the 767762 low byte may be loaded with a new API level 2 address to cause a new API interrupt level 2 and set the API 2 flag in the DR15-C.

Bit 7 - API Ø DONE flag - when a 1 indicates that there is no API level Ø request pending before the PDP-15. When a 1 also indicates that 767772 low byte may be loaded with a new API level Ø and set the API Ø flag in the DR15-C.
Bit 8 - Local Memory Size bit Ø - the least significant bit of a two bit field which specifies the number of 4K word memory banks that are connected to the UNIBUS.

Bit 9 - Local Memory Size Bit 1 - the most significant bit of a two bit field which specifies that number of 4K memory banks are connected to the UNIBUS.

<table>
<thead>
<tr>
<th>LMS1</th>
<th>LMS0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 Local Memory</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4K Local Memory</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8K Local Memory</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>12K Local Memory</td>
</tr>
</tbody>
</table>

Bit 14 - API 3 DONE flag - when a 1 indicates that there is no API level 3 request pending before the PDP-15. When a 1 also indicates that 767762 high byte may be located with a new API level 1 address to cause a new API interrupt at level 3 and set the API 3 flag in the DR15-C.

Bit 15 - API 1 DONE flag - when a 1 indicates that there is no API level request pending before the PDP-15. When a 1 also indicates that 767772 high byte may be loaded with a new API level 1 address to cause a new API interrupt at level 1 and set the API in the DR15-C.

(CSR) 767760 Bit 6 - ENABLE TCBP (Task Control Block Pointer) INTERRUPT - When a 1 allows and interrupt on BR level 7 to TV 310 upon receipt of a new TCBP from the PDP-15.

Bit 7 - NEW TCBP flag - is set to 1 whenever the PDP-15 issues IOT 706006 thus placing a new TCBP in 767764 and bits 0 and 1 of 767774. It is cleared by the PDP-11 doing a DATI to location 767764.

(ODB) 767762 Low byte - contains the API address for an API level 2 break. Same conditions as 767772 low byte.

High byte - contains the API Address for an API level 3 break. Same conditions as 767772.
(IDB) 767764  
TCBP (Task Control Block Pointer) - bits 3-17.
This contains the lowest 15 bits of the address sent by the PDP-15. Note: that the address is "word" aligned. Note also that doing a DATI to this register lowers the New TCBP flag (767760 bit 7) and also sets the DONE flag cleared by IOT 706002 in the PDP-15.

**PDP-15 IOT's**

<table>
<thead>
<tr>
<th>IOT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>706001</td>
<td>SIOA - Skip I/O Accepted. Tests whether the TCBP DONE flag is set indicating the PDP-11 has read the TCBP and skips the next location if the DONE flag is a 1.</td>
</tr>
<tr>
<td>706002</td>
<td>CIOD - Clear I/O Done. Clear the TCBP DONE flag.</td>
</tr>
<tr>
<td>706006</td>
<td>LIOR - Load I/O Register and clear TCBP DONE flag. Places the contents of the PDP-15 &quot;AC&quot; into an 18-bit buffer register. The output of the buffer register is seen by the PDP-11 as TCBP at location 767764 and bits 0 and 1 767764. The IOT also causes the TCBP DONE flag to be cleared and in the PDP-11 causes bit 7 to be set in location 767760, which in turn causes the PDP-11 to do an interrupt at BR 7 to TV location 310.</td>
</tr>
<tr>
<td>706112</td>
<td>RDRS - Read Status Register - Clears the AC and loads the contents of the DR15-C status register into the AC. (This effectively moves the DR15-C enable interrupt bit into bit 17 of the AC).</td>
</tr>
<tr>
<td>706122</td>
<td>LDRS - Load Status Register. Loads the contents of the AC into the DR15-C status register. (Places value of AB bit 17 in the DR15-C &quot;enable interrupts&quot; bit).</td>
</tr>
<tr>
<td>706104</td>
<td>CAPIO - Clear APIO flag in DR15-C.</td>
</tr>
<tr>
<td>706124</td>
<td>CAPI1 - Clear AP11 flag in DR15-C.</td>
</tr>
<tr>
<td>706144</td>
<td>CAPI2 - Clear AP12 flag in DR15-C.</td>
</tr>
<tr>
<td>706164</td>
<td>CAPI3 - Clear AP13 flag in DR15-C.</td>
</tr>
<tr>
<td>706101</td>
<td>SAPIO - Tests the AP10 flag in the DR15-C and skips the next instruction if the flag is 1.</td>
</tr>
</tbody>
</table>
706121 SAPI1 - Tests the AP11 flag in the DR15-C and skips the next instruction if the flag is 1.

706141 SAPI2 - Tests the AP12 flag in the DR15-C and skips the next instruction if the flag is 1.

706161 SAPI3 - Tests the API 13 flag in the DR15-C and skips the next instruction if the flag is 1.

**PDP-15 STATUS REGISTER (DR15-C)**

**Bit 17** Enable PI/API interrupts. When a 1 enables interrupts from the PDP-11 processor. Note this bit is set to a 1 by initialize and the CAF instruction. It can only be cleared by using the LDRS (IOT 706122) instruction.

![Figure 7](image-url)
Figure: 8

PDP-11/05 CONTROL REGISTERS

Bit #

7 6

address

767770

Bit #14

8 6

0

767772

API01 Address  API 0 Address

Bit#15 14

9 8 7 6

1 0

767774

Bit#

7 6

767760

Bit# 14

8 6

0

767762

API 3 address  API 2 address

Bit# 15

PDP 15 bit number

767764

TASK CONTROL BLOCK POINTER (TCBP)

(upper 2 bits in 767774)
When the PDP-15 memory is accessed by the PDP-11/05 or any NPR UNIBUS device, the addresses are relocated by the MX15-B multiplexer.

The MX15-B multiplexer not only relocates the UNIBUS addresses but emulates byte operations in PDP-15 memory. Hence normal PDP-11 programs, with byte read and byte write operations may be executed from PDP-15 memory. Also such byte oriented NPR devices as Mag Tape may make transfers directly to PDP-15 memory.

Note: That the PDP-11 processor can access the PDP-15 memory which is between the end of local memory and the 28K of address space available to its address scheme.

A. Output - PDP-15 Memory Bus

Will connect to MM15, MX15-A, and ME15 memories.

B. Inputs

PDP-11: Modified UNIBUS with PA and PB used as D16 and D17 respectively. It meets all other UNIBUS specs. Defined as UNIBUS/18, input would have a lower address bound that could be fixed to any 4K multiple address 0-120K. This would be specified as jumpers. Note that only 8K and 12K of local memory will be supported by diagnostics and systems programs. Hence, the maximum commonly addressable memory (11 processor) will be 20K or 16K. An upper limit would be provided as 124K.

The addresses presented from the PDP-11 are relocated to prevent location 0 being the same physical address on each machine. The PDP-11 will be able to be relocated by 4K increments to 124K. Local PDP-11 memory is restricted to 4 increments.

Note that any "write" operation to a common memory location by 8 bit or 16 bit UNIBUS devices causes PDP-15 data bits 0 and 1 of the location to be forced 0.

PDP-15: Standard 15 Memory Bus Interface - no upper and lower bounds. No relocation. Emphasis is on minimum delay through multiplexer for this port.
If both processors request at the same time, PDP-15 will get use of the memory. When requests are not simultaneous, a first come, first served mode operates. Practically, all this means is that the 15 and 11 will alternate access to common memory except under the special conditions described above. NOTE: No local memory is provided on the PDP-15.

**Bus Loading:**
- MX15-B.........2 PDP-15 memory bus load
- DR15-C/DR11-C...1 Unibus Load
  - 1 PDP-15 I/O bus load

**Power: (Steady State)**
- UNICHANNEL 15 (no peripherals)...5 at 115V
  - 2.5A at 230V

**Voltage:**
- 115 Vac ‡ 10% or 230 Vac ‡ 10%

**Frequency:**
- 50 ‡ 2 Hz or 60 ‡ 2 Hz

**Environmental:**
- Temperature..............10° to 50° C
- Relative Humidity........20% to 95%

**UC15 Cabinet Dimensions:**
- Depth:....30 in (0.76 m)
- Width:.....21 in. (0.53 m)
- Height:....72 in. (1.83 m)
- Weight:....150 lbs. (70 kg)—not including peripherals.

**Unibus Compatibility:**
Can be used with any PDP-11 family processor that does not use parity. On those systems with parity, the parity must be disabled.

**Memory Cycle:**
MX15-B normally adds 200 ns to both the PDP-15 and the PDP-11 cycle times.

**DMA Facility to Common Memory:**
- Maximum transfer rate.....415K words/sec
- Worst-case latency........6 μs (no DCH transfers in PDP-15)
  - 12 μs (DCH transfers in PDP-15)
- Average latency...........2.5 μs

**DMA Facility to PDP-11/05 Local Memory:**
- Maximum Transfer rate.....1 million words/sec
- Worst-case latency........7.2 μs
- Average latency...........2.5 μs
SYSTEM CONFIGURATION

The UC15 cabinet will replace the current disk cabinet immediately to the left of the PDP-15 processor.

The increased spacing will require longer I/O or memory bus cables in some installations.

Figure 9
SYSTEM RESTRICTIONS

RK05 (RK11) Disk Pack Capability

The 18 bit RK11 disk pack will not be able to be read by RK11-C or RK11-D system (16-bit only systems).

This means that data bases and PDP-11 files created on 18-bit RK11 systems may not be taken directly to an PDP-11 only system. The transfer medium for such a transfer would have to be Mag Tape.

This situation was chosen to make RK11-C and RK11-D packs compatible (i.e. ....all PDP-11 only systems).

Memory Limits

UNIBUS NPR devices can access a maximum of 124K. The amount of shared memory available to UNIBUS NPR devices is 124K less the amount of local memory. In a "normal" configuration the PDP-11/05 would have 8K of memory, in which case the available PDP-15 memory would be limited to 116K. This limit is due to the fact that UNIBUS/18 peripherals must have access to all memory. The maximum memory of the 11 without some relocation option would be 28K.

Note: That the PDP-11 with 8K of local memory can only address the lowest 20K of common memory to access Task Control Blocks set up by the PDP-15.

I/O Latency

Multiport memories always have increased worst case latency over a single port-non-competitive situation. This system is no exception. The PDP-11 normally gives an "NPR break" a worst case latency to BSSY of 7.0 usec. On this system, we must add to that time, the time it requires the PDP-15 to do three I/O memory cycles (5.0 usec.). The worst case latency is, hence, 12.0 usec.

CAF/RESET Limitations

The following timing considerations are of interest to programmers:

A RESET instruction may cause the PDP-15 to incorrectly read the API address. The Console RESET and CAF instruction may violate UNIBUS specifications. Hence, random "initialize" pulses may cause system malfunctions. The following guidelines must always be followed:
1. CAF must not be executed while there is a Task Control Block Pointer (TCBP) waiting to be read by the PDP-11.

2. RESET must not be executed while there are API requests pending for the PDP-15.

3. RESET must not be executed if there is any NPR activity on the UNIBUS. All active NPR devices must be shut down in a power fail sequence prior to executing RESET.
### PDP-15 Unichannel Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UC15-HE</strong></td>
<td>Peripheral Processor: 11/05 or 11/10-NC or - SA, 2 DR11-C, DR15-C, MX15-B, DD11-B, KY11-JH, H950, 115V. 8K Local Memory</td>
</tr>
<tr>
<td><strong>UC15-HF</strong></td>
<td>Peripheral Processor: 11/05 or 11/10-ND or - SB, 2 DR11-C, DR15-C, MX15-B, DD11-B, KY11-JH, H950, 230V. 8K Local Memory</td>
</tr>
<tr>
<td><strong>RK15-HE</strong></td>
<td>RK05-AA, RK11-E, UC15-HE, 115V, 60Hz</td>
</tr>
<tr>
<td><strong>RK15-HH</strong></td>
<td>RK05-AB, RK11-E, UC15-HF, 230V, 60Hz.</td>
</tr>
<tr>
<td><strong>RK15-HJ</strong></td>
<td>RK05-BA, RK11-E, UC15-HE, 115V, 50Hz.</td>
</tr>
<tr>
<td><strong>RK15-HK</strong></td>
<td>RK05-AA, RK11-E, UC15-HK, 115V, 60Hz.</td>
</tr>
<tr>
<td><strong>RK15-HL</strong></td>
<td>RK05-BB, RK11-E, UC15-HL, 230V, 50Hz.</td>
</tr>
<tr>
<td><strong>RK15-HM</strong></td>
<td>RK05-AB, RK11-E, UC15-HL, 230V, 60Hz.</td>
</tr>
<tr>
<td><strong>RK15-HN</strong></td>
<td>RK05-BA, RK11-E, UC15-HK, 115V, 50Hz.</td>
</tr>
</tbody>
</table>


CHAPTER 1

INTRODUCTION

The Magnetic Tape Dump (MTDUMP) Program is a utility program of the PDP-15 ADVANCED Software System which provides users of industry-compatible magnetic tape with functions which are peculiar to this medium. In general, the program provides magnetic tape users with functions similar to those found in PATCH and DUMP. In addition, the program complements PIP with regard to magnetic tape functions; however, few functions which could be performed by PIP are duplicated.

The program MTDUMP is device dependent and accomplishes all magnetic tape I/O with .TRAN and MTAPE System Macro instructions; it cannot be used with other I/O devices.

1.1 FUNCTIONS
The following paragraphs briefly explain the basic functions of MTDUMP. A summary of commands is provided in Appendix A.

1.1.1 Dump File
One of the most common requirements of the magnetic tape user is the ability to examine portions of a tape. The Dump File facility in MTDUMP is intended to meet that need in a general and useful way.

Simply stated, the Dump File is the repository of (1) images of command lines received from the keyboard and (2) groups of ASCII lines which represent, in readable form, the contents of the tape being examined in response to typed requests. The contents and format of the file, however, are subject to considerable variation and, in fact, the destination of the file may itself be changed during the run.

1.1.2 File Modification
This feature provides a convenient means for file updating or patching. Individual records may be accessed, allowing each word in the record to become available for examination and modification. Words and entire records may be inserted or deleted from the file and new files may thus be created.

1.1.3 File Transfer
This function, consisting of one instruction, permits copying magnetic tape on a record-for-record basis.
1.1.4 Directory Listing

These commands permit rapid listing and clearing of magnetic tape directories.

1.2 I/O DEVICES

The program accesses a maximum of three devices: the teleprinter, used for command string input and error reports; the magnetic tape transports (via MTA or MTF) for all input and output to all magnetic tape units; and an optional third device which is the destination device for what is termed the "Dump Output File". This file may contain records of commands typed to the program and any hard-copy response to these commands (normally record-by-record dumps). Dump Output may be directed to any device, including a magnetic tape. If magnetic tape is used for this purpose, however, the unit assigned may not also be manipulated by commands to MTDUMP. If no Dump Output file is desired, the teleprinter should be assigned as the Dump Output device.

1.3 ADDING MTDUMP TO THE USER SYSTEM

The program MTDUMP and its associated handlers (i.e., MTA, MTC, and MTF) are supplied to the user on the ADVANCED Monitor System Peripheral DECTape (DEC-15-SZSB-UC). Users who wish more convenient access to MTDUMP should relocate the program onto the system device using the utility program PIP.

MTDUMP may also be added to the system device as a System Program, using the facilities provided by the SGEN and PATCH utility programs. Refer to PDP-15 manuals DEC-15-YWZA-DN3 and -DN5 for the procedures needed to install MTDUMP onto the system device as a System Program.

If MTDUMP is relocated to the system device by either of the above means, its associated magnetic tape handlers must also be added to the system library (.LIBR BIN). This is accomplished using the utility program UPDATE. The use of UPDATE to insert the handlers MTA, MTC, and MTF is demonstrated in the following example:
UPDATE V8A

>US< (ALT MODE)  Request Options U and S
>IMTA.,DTC.)    Insert routine MTA after routine DTE
>IMTC.)          Insert routine MTC next
>IMTF.)          Insert routine MTF next
>CLOSE)          Terminate UPDATE operations.

Refer to the Utility Programs manual DEC-15-YWZA-D for a complete description of UPDATE and its use.
CHAPTER 2
OPERATING PROCEDURE

2.1 DEVICE ASSIGNMENTS

MTDUMP is supplied as a relocatable program (MTDUMP BIN) and is loaded by the Linking Loader. Before loading, the user must make the following .DAT slot assignments:

.DAT slot -4  The device from which MTDUMP is to be loaded. If the program is on magnetic tape, MTA on .DAT slot 1 requires MTA on .DAT slot -4; MTF on .DAT slot -4 requires MTC on .DAT slot -4.

.DAT slot 1  MTAØ or MTFØ

.DAT slot 3  The Dump Output device, if required; or TTA if no Dump Output File is wanted.

2.2 PROGRAM STARTUP

After loading, the program types on the teleprinter:

MTDUMP Vnn
BUFSIZ m
>

where: "Vnn" is the current version and "m" is the total number (in decimal) of registers available for I/O buffers.

Each time the program is ready to accept a keyboard command, a right angle bracket (>) is typed.

At start (or restart) time, all magnetic tape units are automatically set to transfer in odd parity at 8ØØ BPI and at the channel count given by .SCOM+4, bit 6 (Ø means 7-channel, 1 means 9-channel). The user must issue a new FORMAT request (see paragraph 3.2.1) to effect transfer in another (non-standard) mode.

2.3 PROGRAM RESTART

To restart MTDUMP, type CTRL P, which causes the program to close the Dump Output File (if open) on .DAT slot 3. Then repeat program startup procedure.

NOTE

If the Dump Output has been directed to the teleprinter, CTRL P is acted upon only after completion of current line of output. To effect immediate termination, type CTRL P CTRL U.
CHAPTER 3

COMMANDS

3.1 COMMAND STRING

MTDUMP accepts commands from the Teletype in the general format shown below. Formats for specific commands may vary significantly from this and are shown in the descriptions of the individual commands.

MTDUMP command formats are variations of the following:

$$c_{\text{u}_1, \text{u}_2, t_j}$$

where:

- $c$ is the name of the function wanted.
- $u_1$ is a digit specifying the source unit for two-unit operations (e.g., COPY) or the one object unit for single-unit operations.
- $u_2$ is a digit specifying the destination unit for two-unit operations or is absent for single-unit operations.
- $t$ specifies a condition (either count overflow or transport status) which, when encountered, causes termination of the function whose name is "$c". "$t" may be absent and, if not given, is assigned the implicit integer value 1. Explicit values of "$t" may include:
  a. An integer in absolute value less than $262,144_{10}$ and greater than zero
  b. The character string "EOT" (END OF TAPE)
  c. The character string "BOT" (BEGINNING OF TAPE)
  d. The character string "EOF" (END OF FILE)

Parameters are separated from the command by a space ( ) and from each other by commas. The command line is terminated by a carriage return ( ).

Some commands require only a single argument, while others require all three.

Example:

```
REWIND(1)
```

Only the single object unit need be specified; further, the terminating
condition "BOT" is implicit in the command and need not be given. Copying an entire logical tape from Unit 1 to Unit 2, however, requires all three parameters.

Example:

```
COPY_1,2,EOT
```

3.1.1 Terminating Conditions

As indicated above, the "t" specification in the command line may be either an integer or a character string or absent. If "t" is an integer, the value of the numeric string represents the number of physical records to be treated during the operation requested.

Example:

```
SPACE_1,80
```

This command string means: Evaluate the string "80" according to the radix currently in effect, then space the tape on drive 1 forward until that many records have been passed over. If, in the example, tape 1 was at loadpoint and if the prevailing radix was decimal, then at the completion of the operation the read/write head would be positioned between the 80th and 81st physical records on tape.

Example:

```
COPY_1,2,80
```

The above example causes a transfer of 80 physical records from drive 1 to drive 2, leaving the read/write head on each drive positioned immediately following the last record transferred.

If "t" is a non-numeric string (EOT, BOT, EOF), then the operation requested is deemed complete when one of the following conditions is observed:

a. EOT Two consecutive EOF markers have been passed in either reverse or forward direction.

b. BOT The loadpoint marker has been reached (but not passed) in the reverse direction.

c. EOF A single EOF marker has been passed in either direction.

If "t" is the string "EOF" or "EOT", the position of the read/write head relative to the EOF marker causing termination depends upon the direction of tape motion when the condition is encountered.
Example:

```
BACKSPACE, EOF
```

The read/write head will be positioned just before the marker. The next record read in the forward direction will be the EOF marker just passed in backspacing.

If "t" is the string "BOT", the head is left positioned just after the loadpoint; the program will not backspace over BOT.

If "t" is absent from a command string in which it is required, then the value 1 is assumed. Thus the commands in the following example are equivalent.

Example:

```
SPACE, 1

```

3.1.2 Command Abbreviations

Most commands in MT DUMP may be abbreviated to a single letter (the initial character). In the command descriptions which follow, legal abbreviations are shown immediately following the command and enclosed in parentheses.

Example:

```
REWIND (R)
```

3.2 SETUP COMMANDS

This is a group of commands which generally apply to most major functions of MT DUMP. These commands are usually given prior to the execution of a function (e.g., DUMP, COPY).

3.2.1 Set Non-Standard Tape Format

The initial setup for input and output tapes is odd parity at 800 BPI (the channel count is given by .SCOM+4, bit 6). The FORMAT command allows the user to change the parity, density, and/or channel count.

Usage:

```
FORMAT (F)
```
where: "u" is the tape whose format is being set and "pdc" is a group of three single-character parity, density, and channel-count indicators, as follows:

- p (parity) is "E" (even) or "O" (odd)
- d (density) is "2" (280 BPI), "5" (556 BPI), or "8" (880 BPI)
- c (channel) is "9" (9-channel) or "7" (7-channel)

The three descriptors may appear in any order, and any may be absent, in which case the relevant status for the tape remains unchanged.

Example:
```
FORMAT 2,E57
```
or
```
FORMAT 2,5E7
```
or
```
FORMAT 2,75E
```
All of the above examples set up tape unit 2 for even parity, 556 BPI, 7-channel operation.

Example:
```
FORMAT 2,0
```
or
```
FORMAT 2,0
```
These commands change the parity of tape unit 2 without disturbing the current density or channel count.

NOTE
The only legal density for a 9-channel tape drive is 800 BPI. Requests for other densities will not be honored.

FORMAT commands are effective until MTDUMP is restarted via the CTRL P function.

3.2.2 Set Standard Tape Format
Standard System Format may be requested for any unit. A special case of the FORMAT command is employed to unconditionally reset tape format to odd parity, 880 BPI, and 7- or 9-channel (according to .SCOM+4, bit 6.)
Usage:

\[ \text{FORMAT}(F)_{u,D} \]

where: "u" is the unit whose format is to be set and the character "D" means "default!"

3.2.3 Specify Global Radix
The program always treats certain numeric strings (e.g., unit specification) as octal. Others, however, may be specified as either octal or decimal by the NUMBER command. The following numeric groups are interpreted (on input) or printed as octal or decimal strings according to the argument given in the latest NUMBER request:

a. The "t" specification in command lines (where applicable) when "t" is an integer. If the current radix is octal, then the command:

\[ \text{SPACE}_{1,2^0} \]

causes the tape on unit 1 to be spaced forward \( 16_{10} \) records.

b. The word sequence numbers of dumped data.

c. The word sequence numbers of EXAMINE requests. (See below.)

d. The record-length argument of the SIZE request. (See below.)

The radix specified remains in effect until another NUMBER command is encountered or the program is restarted. The default radix is octal.

Usage:

\[ \text{NUMBER}(N)_{\{\text{OCTAL, DECIMAL}\}} \]

3.2.4 Specify Local Radix
The radix of a number string in a single command line may be specified by a one-character suffix, D for decimal, K for octal. Such specification overrides the current global radix, but is in effect only during the processing of the command line in which the suffix appears. Local radix control may be used following:

a. The "t" specification in command lines (where applicable) when "t" is an integer.

b. The word sequence numbers of EXAMINE requests.
c. The record-length argument of the SIZE request.

Example:

\[ \text{SPACE}_{1,2} \]

The command above causes tape unit 1 to space forward 20 records regardless of the current global radix.

Example:

\[ \text{SPACE}_{1,2} \]

Similarly, this command spaces the tape forward 20 (16) records.

3.2.5 **Command-Line Echo**

Legal keyboard requests are placed in the Dump Output File, exactly as typed, to allow the user to correlate the progress of the run, relative tape position, and the record contents during later examination of the hard-copy dump. Command-line echo can be bypassed, however, by use of the VERIFY command.

Usage:

\[ \text{VERIFY}(V) \{ \text{ON} \} \]

If ON or OFF is not specified, ON is assumed.

Example:

\[ V \]

When MTDUMP is first loaded or is restarted, VERIFY mode is set ON. If the teleprinter is the assigned dump output device (.DAT slot 3), command-line echo is not performed. Illegal commands are not echoed.

3.2.6 **Dump File Display Format**

The input tape is output to the Dump File as individual physical records. Each record is represented as a number which indicates record length in ASCII lines. Each line, in turn, contains:

1. A sequence number which reflects the position in the record of the first data word in the line displayed.
2. A string of data words or data-word pairs.

Sequence numbers are in either octal or decimal notation; the radix is chosen in response to the last previous NUMBER command.
Display format is set by the MODE request followed by the appropriate argument.

Usage:

```
MODE (OCTAL SYMBOLIC TRIMMED ASCII)
```

Where:

- **OCTAL** Displays single words as six octal digits.
- **SYMBOLIC** Displays single words as a three-character operation-code mnemonic, an "indirection" indicator (*), if present, and a 13-bit (5-digit) address.
- **TRIMMED** Displays single words as three six-bit alphanumeric characters.
- **ASCII** Displays pairs of words as five seven-bit ASCII characters. A blank is printed for each character outside the range $40_8 - 137_8$.

The default assumption is OCTAL and implicit in the request:

```
MODE
```

The table below shows examples of data-word treatment in each of the four modes.

<table>
<thead>
<tr>
<th>OCTAL</th>
<th>SYMBOLIC</th>
<th>TRIMMED</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>512132</td>
<td>AND 12132</td>
<td>)QZ</td>
<td>REWIN</td>
</tr>
<tr>
<td>744634</td>
<td>OPR @4634</td>
<td>&lt;&amp;\</td>
<td></td>
</tr>
<tr>
<td>42@32@</td>
<td>XCT*@32@</td>
<td>#CP</td>
<td>D</td>
</tr>
<tr>
<td>@3@3@</td>
<td>CAL @3@3@</td>
<td>@@@</td>
<td></td>
</tr>
<tr>
<td>777777</td>
<td>LAW 17777</td>
<td>???</td>
<td>A</td>
</tr>
<tr>
<td>@1@2@3</td>
<td>CAL 1@2@3</td>
<td>ABC</td>
<td></td>
</tr>
</tbody>
</table>

3.2.7 *Inserting Comments in the Dump File*

Explanatory notes may be placed in the output file by use of the LOG command. When the LOG request is encountered, subsequent typed input is taken as commentary and is added, exactly as it appears, to the Dump Output File. Carriage returns may be included, and multiple lines may be inserted with a single LOG request. An ALTMODE terminates each comment and causes the program to accept a new request.
Usage:

\[
\text{LOG} \leftarrow \text{comments}\nonumber \\
\text{comments} \ldots 
onumber \\
(\text{ALTMODE})
\]

3.2.8 **Return Control to Monitor**

An EXIT request causes the program to close the Dump Output File (if one is open) on .DAT slot 3, then perform an .EXIT return to the Monitor. Use this command for return to the Monitor if the program is being run in the Batch Environment.

Usage:

\[
\text{EXIT}
\]

3.3 **MANIPULATIVE FUNCTIONS**

The following commands position the tape and write EOF markers on the tape drive specified.

3.3.1 **Rewind Tape**

This command initiates a rewind on tape unit "u".

Usage:

\[
\text{REWIND(R} \leftarrow u
\]

3.3.2 **Backspace Tape**

This command backspaces the tape on unit "u" until the "t" condition is satisfied.

Usage:

\[
\text{BACKSPACE(B} \leftarrow u, t
\]

where: "t" is an integer (number of records), "EOF", "EOT", or "BOT".

3.3.3 **Space Tape**

This command spaces the tape on unit "u" forward until the "t" condition is satisfied.

Usage:

\[
\text{SPACE(S} \leftarrow u, t
\]

where: "t" is an integer (number of records), "EOF", or "EOT".
3.3.4 Write End-of-File Marker
This command writes a single "EOF" marker on tape unit "u".

Usage:

\texttt{TAPEMARK(T)_{u}}

3.4 DUMP FILE OPERATIONS

3.4.1 Dump File Management
The Dump Output File may be written on any physical device. If the device chosen is file-structured, however, the user must specify a name to be given the Dump File and must explicitly request that the file be closed (unless the EXIT command is used). Furthermore, the file name must be given before any other requests are issued.

Usage:

\texttt{OPEN_{filename}{ext}}

where: \texttt{filename} is the name of the file to be created.
\texttt{ext} is the filename extension. If omitted, "LST" is the default assumption.

If an OPEN request is not given, the program types

\texttt{NO DUMP FILE OPEN}

on the Teletype and waits for another command.

\textbf{NOTE}

The comment is actually printed when an attempt is made to write into the Dump File, i.e., at command-line echo if VERIFY is ON or at Dump-Record Output if VERIFY is OFF.

A check is made to ensure that the filename given is unique. If a file of the name specified already exists on the Dump Output device, the program types:

\texttt{FILE FOUND ON DUMP DEVICE: \textit{filename ext}}

\texttt{DO YOU WISH TO DELETE IT?}

\texttt{>}

\textbf{193}
The program then waits for the user to type a response to the query. Typing

```
;
''
or
```
''
or
```
YES;
```

indicates the affirmative, and the already-existing file is overlayed (i.e., deleted when the new file is .CLOSEd). Any other response is negative and the program returns to accept a new keyboard command.

The Dump Output File is closed upon receipt of the CLOSE command from the keyboard.

Usage:

```
CLOSE;
```

or whenever the program is restarted (CTRL P).

3.4.2 Dump Tape Records
This command dumps records from unit "u" into the named file open on .DAT slot 3. The sequencing of data words and the format in which they are written are controlled by the latest NUMBER and MODE requests.

Usage:

```
DUMP(D)u,t;
```

where:
```
"u" is the tape unit number
"t" is an integer (number of records), "EOF", or "EOT".
```

3.4.3 Dump Tape Records on the Teleprinter
This command performs the same function as the DUMP command, except that the records are unconditionally dumped on the teleprinter.

Usage:

```
LIST(L)u,t;
```

194
3.4.4 **Tape Status**

In addition to data input from magnetic tape and the Teletype, the Dump Output File contains indicators of status encountered on the tape being read. Comments are added to the file (and typed on the teleprinter) in response to the following observed conditions on the tape.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*END OF FILE ENCOUNTERED</td>
<td>An unexpected end-of-file marker was read.</td>
</tr>
<tr>
<td>*PHYSICAL BOT ENCOUNTERED</td>
<td>The end-of-tape reflective spot was reached on input or output.</td>
</tr>
<tr>
<td>*BUFFER OVERFLOW</td>
<td>The tape record read is too long to be accommodated in the available buffer space.</td>
</tr>
<tr>
<td>*BOT ENCOUNTERED</td>
<td>The loadpoint reflective spot was unexpectedly reached during a backspace operation.</td>
</tr>
<tr>
<td>*PERMANENT READ ERROR ENCOUNTERED</td>
<td>After 64,10 read attempts, the input record still has not been transferred correctly. The read/write head is positioned immediately before the record.</td>
</tr>
</tbody>
</table>

3.4.5 **Example of Dump Operation**

The following example shows the instructions required to dump the file directory of magnetic tape unit 0 in octal format (to allow the accessibility map to be examined) and then in trimmed ASCII format (to allow reading of the file name entries).
3.5 TRANSFER FUNCTION

The COPY command allows the user to perform record-for-record copying of tapes.

Usage:

COPY(C) → u₁, u₂, t

where:
"u₁" is the source drive
"u₂" is the destination drive
"t" may be an integer (number of records), "EOF", or "EOT"

Standard parity and density (odd parity, 800 BPI) prevail, unless they have been changed by a FORMAT request.

To copy an entire tape from unit 1 to unit 2, for example:

REWIND₁
REWIND₂
COPY₁,₂,EOT₁₂

To replace the last data record on unit 2 with the first data record on unit 1:

REWIND₁
SPACE₂,EO₂
BACKSPACE₂,₁
COPY₁,₂,₁
TAPEMARK₁
TAPEMARK₂

3.6 FILE MODIFICATION

The file modification feature of MTDUMP allows the user to access single records, modify or delete words in a record, delete entire records, or add new records to his file.

3.6.1 Read a Single Record

The next sequential physical record is read from tape unit "u" and is stored in core. Its length is saved in anticipation of a subsequent PUT request (see Paragraph 3.6.4).

Usage:

GET(G) → u

At the completion of input, the following message is printed indicating
the length, in words, of the record just read.

RECSIZE:nn

3.6.2 Examine and Modify Data Words
Designed for use in conjunction with the GET and PUT commands, the EXAMINE request allows the user to access and update individual data words in the program buffer. Any number of contiguous registers may be examined and modified with a single command.

Usage:

EXAMINE(E)\text{ \(n\)}

where: "\(n\)" is the relative position in the buffer (record) of the first word to be displayed. If a "D" or "K" suffix (see section 3.2.4) is present, the argument is interpreted appropriately. If no suffix is present, "\(n\)" is interpreted according to the current global radix. The argument specifies the position of a word relative to word \(0\) in the buffer.

Example:

EXAMINE\text{ \(1\)}

The above command accesses the first data word in the buffer. The program responds to the command by displaying on the teleprinter the contents of the register specified in the mode (octal, symbolic, trimmed, ASCII) currently in effect. No carriage return is executed, however, after the displayed data word typeout. The user has several options.

a. If a carriage-return is typed, the program responds by displaying the contents of the next higher register.

b. If an ALTMODE is typed, buffer examination is deemed complete and the program returns to read a new command.

c. A six-digit numeric string (octal notation) may be typed to replace the contents of the register being examined. The terminator of the line typed by the user may be either a carriage return or an ALTMODE. The terminator directs the program's activity after the desired modification has been performed. A carriage return opens the next sequential register; an ALTMODE returns control to the command processor.
3.6.3 Specify Output Record Length

The SIZE command specifies, in words, the length of the record to be written in response to a subsequent PUT request (see paragraph 3.6.4).

Usage:

```
SIZE n
```

where: the parameter "n" is the total words in the output record.
If a suffix "D" or "K" (see section 3.2.4) is present, the argument is evaluated appropriately. If no suffix is present, the numeric string is interpreted in the current global radix.

Output record size is implicitly set during input "GET" processing.
The SIZE facility offers a means of overriding the implicit setting.

3.6.4 Write Single Record

The PUT command writes data residing in the program's buffer as the next sequential record on tape unit "u". The length of the record written is either the length of the record read in response to the latest GET request or the length specified in a SIZE request which occurred after the latest GET request.

Usage:

```
PUT(P) u
```

3.7 DIRECTORY LISTING

This group of commands is available for dealing with the Magnetic Tape File Directory. The contents of the Directory on unit "u" may be printed on the teleprinter or written into the Dump Output File; and the Directory may be cleared. None of these commands may be abbreviated.

3.7.1 Write File Directory in Dump Output File

The contents of the File Directory of the tape specified are written in the Dump Output File.

Usage:

```
DDUMP u
```

3.7.2 Print File Directory on Teleprinter

The contents of the File Directory of the tape specified are printed on the Teletype.
Usage:

\[ \text{DLIST} \quad u \) \\

3.7.3 Clear Tape File Directory
Write a new (empty) File Directory on the tape specified.

Usage:

\[ \text{NEWDIR} \quad u \)
## APPENDIX A
### SUMMARY OF COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>MEANING</th>
<th>PARAGRAPH #</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT</td>
<td>Return Control to Monitor</td>
<td>3.2.8</td>
</tr>
<tr>
<td>FORMAT (P)</td>
<td>non-standard tape</td>
<td>3.2.1</td>
</tr>
<tr>
<td>FORMAT (P)</td>
<td>standard tape</td>
<td>3.2.2</td>
</tr>
<tr>
<td>LOG (comments)</td>
<td>Insert one or more lines of comments</td>
<td>3.2.7</td>
</tr>
<tr>
<td>(ALTMODE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODE {OCTAL</td>
<td>Symbolic</td>
<td>3.2.6</td>
</tr>
<tr>
<td></td>
<td>TRIMMED</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASCII</td>
<td></td>
</tr>
<tr>
<td>NUMBER {OCTAL</td>
<td>DECIMAL</td>
<td>Specify Global Radix</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOTE</td>
<td>D (decimal) or K (octal) specifies Local Radix</td>
<td>3.2.5</td>
</tr>
<tr>
<td></td>
<td>which overrides Global Radix during processing of a single command line.</td>
<td></td>
</tr>
<tr>
<td>VERIFY (V)</td>
<td>ON</td>
<td>Bypass Command-Line</td>
</tr>
<tr>
<td></td>
<td>OFF</td>
<td></td>
</tr>
</tbody>
</table>

### MANIPULATIVE COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>MEANING</th>
<th>PARAGRAPH #</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKSPACE (B)</td>
<td>Backspace Tape</td>
<td>3.3.2</td>
</tr>
<tr>
<td>REWIND (R)</td>
<td>Rewind Tape</td>
<td>3.3.1</td>
</tr>
<tr>
<td>SPACE (S)</td>
<td>Space Tape</td>
<td>3.3.3</td>
</tr>
<tr>
<td>TAPEMARK (T)</td>
<td>Write End-of-File Marker</td>
<td>3.3.4</td>
</tr>
</tbody>
</table>

### DUMP FILE COMMANDS

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>MEANING</th>
<th>PARAGRAPH #</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>Close Dump Output File</td>
<td>3.4.1</td>
</tr>
<tr>
<td>DUMP (D)</td>
<td>Dump Records into Named File</td>
<td>3.4.2</td>
</tr>
<tr>
<td>LIST (L)</td>
<td>Dump Records onto teleprinter</td>
<td>3.4.3</td>
</tr>
<tr>
<td>OPEN (filename</td>
<td>Open Named File</td>
<td>3.4.1</td>
</tr>
<tr>
<td>ext)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMAND</td>
<td>MEANING</td>
<td>PARAGRAPH #</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>COPY(C) u_1, u_2, t</td>
<td>Copy Tape Specified</td>
<td>3.5</td>
</tr>
</tbody>
</table>

**Transfer Command**

**File Modification Commands**

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>MEANING</th>
<th>PARAGRAPH #</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAMINE(E) n</td>
<td>Examine and Modify Data Words</td>
<td>3.6.2</td>
</tr>
<tr>
<td>GET(G) u</td>
<td>Read Single Record</td>
<td>3.6.1</td>
</tr>
<tr>
<td>PUT(P) u</td>
<td>Write Single Record</td>
<td>3.6.4</td>
</tr>
<tr>
<td>SIZE n</td>
<td>Specify Output Record Length</td>
<td>3.6.3</td>
</tr>
</tbody>
</table>

**Directory Commands**

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>MEANING</th>
<th>PARAGRAPH #</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDUMP u</td>
<td>Write File Directory in Dump Output File</td>
<td>3.7.1</td>
</tr>
<tr>
<td>DLIST u</td>
<td>Print File Directory on Telexprinter</td>
<td>3.7.2</td>
</tr>
<tr>
<td>NEWDIR u</td>
<td>Clear Tape File Directory</td>
<td>3.7.3</td>
</tr>
</tbody>
</table>
Marlboro Map
SOFTWARE PROBLEMS OR ENHANCEMENTS

Questions, problems, and enhancements to Digital software should be reported on a Software Performance Report (SPR) form and mailed to the SPR Center at one of the following Digital Offices: (SPR forms are available from the SPR Center.)

<table>
<thead>
<tr>
<th>Areas Covered</th>
<th>SPR Center</th>
</tr>
</thead>
<tbody>
<tr>
<td>Australia/New Zealand</td>
<td>Digital Equipment Australia Pty. Ltd. 123-125 Willoughby Road, P.O. Box 491</td>
</tr>
<tr>
<td></td>
<td>Crows Nest New South Wales, Australia 2065</td>
</tr>
<tr>
<td>Brazil</td>
<td>Digital Equipment Comercio E Industria LTDA Rua Batatais, 429 (Esq. Al. Campinas)</td>
</tr>
<tr>
<td></td>
<td>01423-Jardim Paulista São Paulo-SP-Brazil</td>
</tr>
<tr>
<td>Canada</td>
<td>Digital Equipment of Canada, Ltd. Software Services P.O. Box 11500, K2H 8K8</td>
</tr>
<tr>
<td></td>
<td>Ottawa, Ontario, Canada</td>
</tr>
<tr>
<td>Caribbean</td>
<td>Digital Equipment Latin America, Inc. 407 del Parque Street</td>
</tr>
<tr>
<td></td>
<td>Santurce, Puerto Rico 00912</td>
</tr>
<tr>
<td>United States, Far East,</td>
<td>Software Communications</td>
</tr>
<tr>
<td>Middle East, Africa,</td>
<td>P.O. Box F Maynard, MA 01754</td>
</tr>
<tr>
<td>Remainder of Latin America</td>
<td></td>
</tr>
<tr>
<td>France</td>
<td>Digital Equipment France 18, rue Saarinen Centre Silic - CIDEX L225</td>
</tr>
<tr>
<td></td>
<td>F-94533 Rungis, France</td>
</tr>
<tr>
<td>Israel</td>
<td>DEC-sys Computers Ltd. 7 Habakuk Street IL-Tel Aviv 63505, Israel</td>
</tr>
<tr>
<td>Italy</td>
<td>Digital Equipment S.P.A. Corso Garibaldi 49</td>
</tr>
<tr>
<td></td>
<td>I-20121 Milano, Italy</td>
</tr>
<tr>
<td>Japan</td>
<td>Digital Equipment Corp. Int. Kowa Building #25 (3rd Floor) B-7 Sunban-Cho</td>
</tr>
<tr>
<td></td>
<td>Chiyoda-ku, Tokyo 102, Japan</td>
</tr>
<tr>
<td>Mexico</td>
<td>Equipo Digital, S.A. de C.V. 109 Concepcion Beistequi</td>
</tr>
<tr>
<td></td>
<td>Mexico 12, D.F.</td>
</tr>
<tr>
<td>The Netherlands</td>
<td>Digital Equipment B.V.</td>
</tr>
<tr>
<td>Belgium</td>
<td>Kaap Hoornrreef 38, P.O. Box 9064 NL-Utrecht - Overvecht, The Netherlands</td>
</tr>
<tr>
<td>Scandinavia</td>
<td>Digital Equipment AR</td>
</tr>
<tr>
<td></td>
<td>Englundsvägen 7 S-17141 Solna Sweden</td>
</tr>
<tr>
<td>Switzerland</td>
<td>Digital Equipment Corp. SA 20, Quai Ernest Ansermet</td>
</tr>
<tr>
<td>Spain Portugal</td>
<td>Case Postale 23, CH-1211 Geneva 8 Switzerland</td>
</tr>
<tr>
<td>Greece Bulgaria</td>
<td>Digital Equipment Co. Ltd. Fountain House, Butts Centre</td>
</tr>
<tr>
<td>Romania Yugoslavia</td>
<td>GR-Reading RG1 70N, England</td>
</tr>
<tr>
<td>United Kingdom</td>
<td></td>
</tr>
<tr>
<td>West Germany Austria</td>
<td>Digital Equipment GmbH</td>
</tr>
<tr>
<td>East Germany Russia</td>
<td>D-8000 München 40 Wallensteinplatz 2</td>
</tr>
<tr>
<td>Hungary Poland</td>
<td>West Germany</td>
</tr>
<tr>
<td>Czechoslovakia</td>
<td></td>
</tr>
</tbody>
</table>
HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes newsletters and Software Performance Summaries (SPS) for the various Digital products. Newsletters are published monthly, and contain announcements of new and revised software, programming notes, software problems and solutions, and documentation corrections. Software Performance Summaries are a collection of existing problems and solutions for a given software system, and are published periodically. For information on the distribution of these documents and how to get on the software newsletter mailing list, write to:

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

SOFTWARE PROBLEMS

Questions or problems relating to Digital's software should be reported to a Software Support Specialist. A specialist is located in each Digital Sales Office in the United States. In Europe, software problem reporting centers are in the following cities:

Reading, England  Milan, Italy
Paris, France  Solna, Sweden
The Hague, Holland  Geneva, Switzerland
Tel Aviv, Israel  Munich, West Germany

Software Problem Report (SPR) forms are available from the specialists or from the Software Distribution Centers cited below.

PROGRAMS AND MANUALS

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center.

Digital Equipment Corporation  Digital Equipment Corporation
Software Distribution Center  Software Distribution Center
146 Main Street  1400 Terra Bella
Maynard, Massachusetts 01754  Mountain View, California 94043

Outside of the United States, orders should be directed to the nearest Digital Field Sales Office or representative.

USERS SOCIETY

DECUS, Digital Equipment Computer Users Society, maintains a user exchange center for user-written programs and technical application information. A catalog of existing programs is available. The society publishes a periodical, DECUSCOPE, and holds technical seminars in the United States, Canada, Europe, and Australia. For information on the society and membership application forms, write to:

DECUS  DECUS
Digital Equipment Corporation  Digital Equipment, S.A.
146 Main Street  P.O. Box 340
Maynard, Massachusetts 01754  1211 Geneva 26
                             Switzerland
DIGITAL'S REGIONAL EDUCATION CENTERS

FOR INFORMATION REGARDING DIGITAL'S CUSTOMER TRAINING PROGRAM AND TO ACCOMODATE COURSE ENROLLMENTS, CONTACT YOUR NEAREST EDUCATION CENTER LISTED BELOW:

PHILADELPHIA AREA:

Digital Equipment Corporation
Educational Services Department
Whitpain Office Campus
1740 Walton Road
Blue Bell, Pennsylvania 19422
Telephone: (215)825-4200 Ext.24

WASHINGTON, D.C. AREA:

Digital Equipment Corporation
Educational Services Department
Lanham 30 Office Building
5900 Princess Garden Parkway
Lanham, Maryland 20801
Telephone: (301)45-7900

PRINCETON, N.J. AREA:

Digital Equipment Corporation
Educational Services Department
U.S. Route 1
Princeton, New Jersey 08540
Telephone: (609)452-2940

CHICAGO AREA:

Digital Equipment Corporation
Educational Services Department
5600 Apollo Drive
Rolling Meadows, Illinois 60008
Telephone: (312)640-5500

BOSTON AREA:

Digital Equipment Corporation
Educational Services Department
(PDP-10 & PDP-15)
200 Forest Street
Marlboro, Massachusetts 01752
Telephone: (617)481-9511 Ext.5071

Digital Equipment Corporation
Educational Services Department
Maynard, Massachusetts 01754
Telephone: (617)897-0511 Ext. 3819 or 2564

SAN FRANCISCO AREA:

Digital Equipment Corporation
Educational Services Department
310 Soquel Way
Sunnyvale, California 94086
Telephone: (408)735-9200 Ext.221
EDUCATION CENTERS (Cont.):

UNITED KINGDOM:
Digital Equipment Company Ltd.
Fountain House, Butts Center
Reading, England RG1 70N
Telephone: 58-35-55

GERMANY:
Digital Equipment GmbH.
Educational Services Department
Wallensteinplatz 2
D-8 Munich 13, Germany
Telephone: 35-03 1

THE NETHERLANDS:
Digital Equipment N.V.
Educational Services Department
Kaap Hoornweg 38
Utrecht, Holland
Telephone: 030-63 12 22

SWEDEN:
Digital Equipment AB
Englundsvagen 7, 3TR
S-171-41 Solna, Sweden
Telephone: 08/98-13/90

FRANCE:
Digital Equipment S.A.R.L.
Educational Services Department
2 Place Gustave Eiffel
F-94533 Rungis, France
Telephone: (01) 687-2333

ITALY:
Digital Equipment SPA
Educational Services Department
Corso Garibaldi 49
1-20121 Milan, Italy
Telephone: 87-90-51

AUSTRALIA:
Digital Equipment Australia Pty. Ltd.
Educational Services Department
123-135 Willoughby Road
Crows Nest
New South Wales 2065, Australia
Telephone: 439-2566

JAPAN:
Digital Equipment Corporation Ltd.
Educational Services Department
Kowa Bldg. No. 25, Third Floor
8-7 Sanban-Cho
Chiyoda-ku, Tokyo 102, Japan
Telephone: (03) 264-7101
THE FOLLOWING PAGES ARE MISSING:

10-12
14-17
21-25
112-117
179-180
203-204