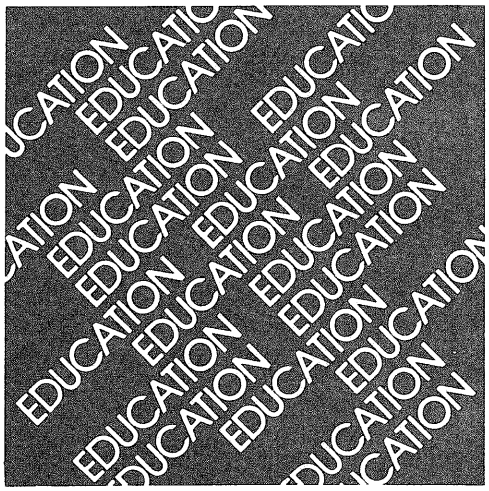
H. Baykurst

digital

PDP-15
SYSTEM SOFTWARE
HANDOUTS

PDP-15
SYSTEM SOFTWARE
HANDOUTS



EDUCATIONAL SERVICES

digital equipment corporation · maynard. massachusetts

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.

The following are trademarks of Digital Equipment Corporation:

CDP	DIGITAL	INDAC	PS/8
COMPUTER LAB	DNC	KA10	QUICKPOINT
COMSYST	EDGRIN	LAB-8	RAD-8
COMTEX	EDUSYSTEM	LAB-8/e	RSTS
DDT	FLIP CHIP	LAB-K	RSX
DEC	FOCAL	OMNIBUS	RTM
DECCOMM	GLC-8	OS/8	RT-11
DECTAPE	IDAC	PDP	SABR
DIBOL	IDACS	PHA	TYPESET 8
			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 useage 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

BLOCK DIAGRAM OF THE PDP-15 SYSTEM

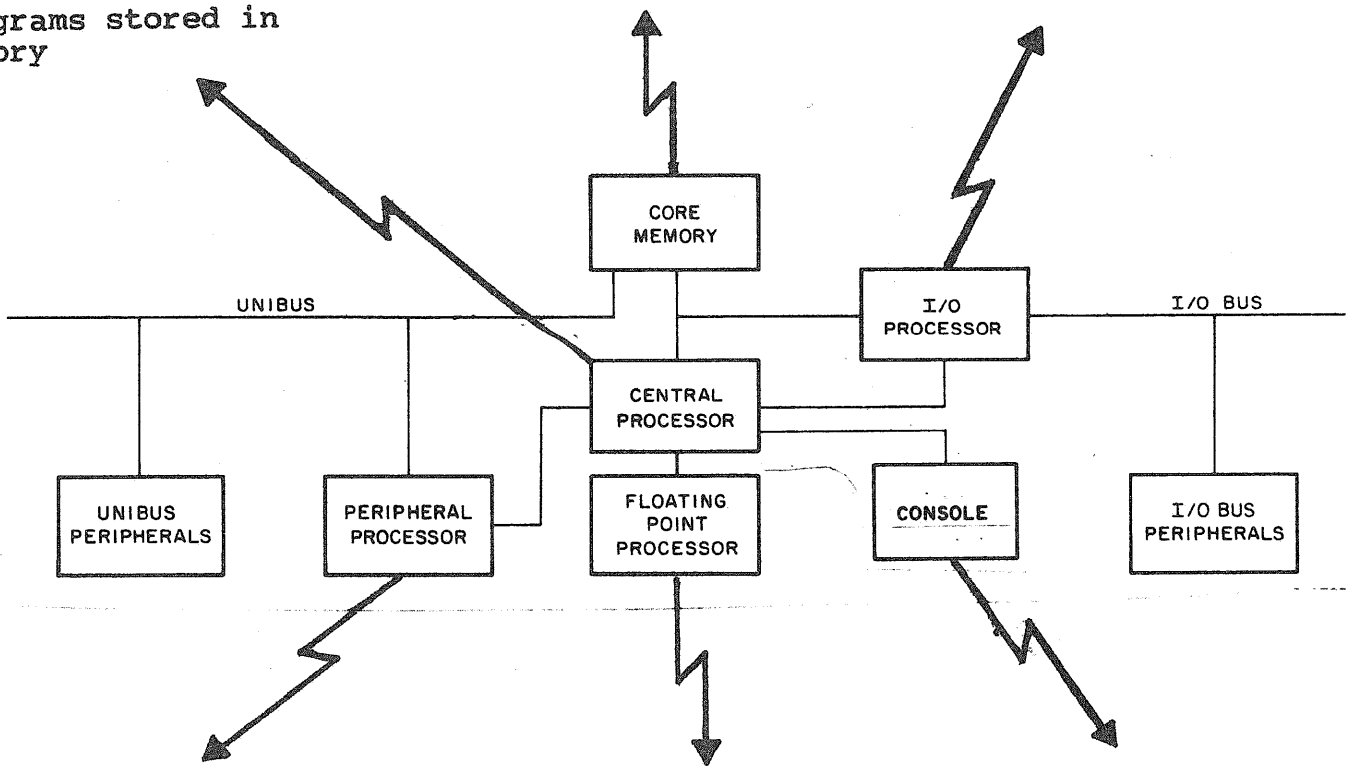
Computer: a machine which inputs data from the outside world, processes the data, perhaps puts it in temporary storage and finally outputs the data and/or results in the form of hard copy, displays or commands to other devices it may be controlling.

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).

-main component
-handles bidirectional communication with memory and I/O processor
-performs arithmetic and logical operations
-controls and executes programs stored in memory

-main storage area within computer from which instructions are fetched and executed (data may also be stored here)
-various types of memory available

-handles peripheral data transfers
-coordinates transfer between: CP & peripherals memory & peripherals



-second general purpose processor
-second I/O bus allowing use of PDP-11 peripherals

-allows use of floating point arithmetic via over 100 instructions without use of complex software routines

-allows operator communication in system:
-starting & halting of programs
-monitoring of registers
-modification of memory

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)
89TRAN (PDP-8 to PDP-9 Translator)
TKB (RSX-15 Task Builder)

Utilities (System)
DOSSAV (Disk Save/Restore)
RFBOOT (DECdisk Bootstrap)
RPBOOT (Disk Pack Bootstrap)
RKBOOT (Disk Cartridge Bootstrap)

I/O Handlers
CDB (Card Reader for CR03B, CR15 or CR11)
DOSBCD (Batch Card Reader)
DKA, DKB, DKC, DKL, (RF15/RS09 DECdisk)
DPA, DPB, DPC, DPL (RP15/RP02 Disk Pack)
RKA, RKB, RKC, RKL (RK15/RK05 Disk Cartridge)
DTA, DTC, DTD, DTE, DTF, (DECtape)
LKA (LK35 Graphics Keyboard)
LPA (Line Printer for LP15, LS11 or LP11)
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

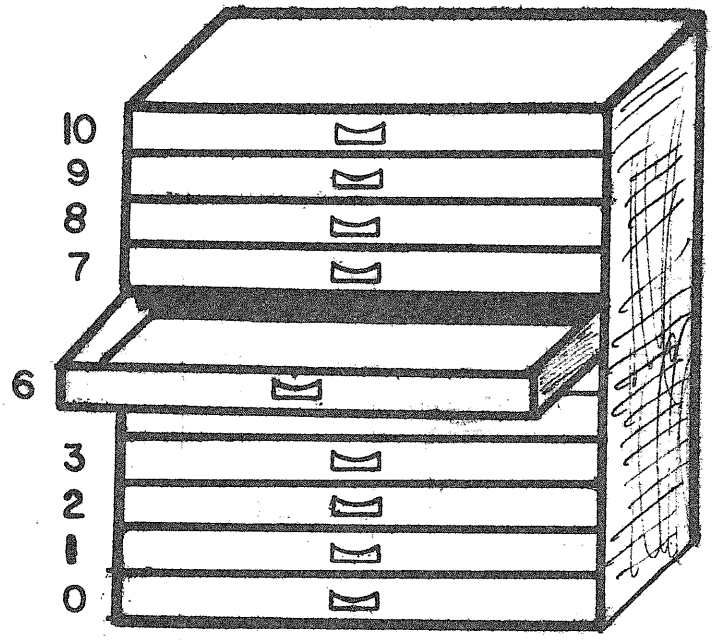
MEMORY

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.



MEMORY

a. You can store something in each drawer.

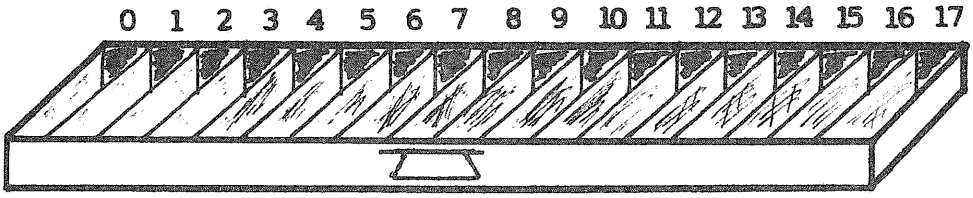
b. You must examine a basic storage unit when looking for something -- in the chest this unit is a drawer; in a computer this unit is a location (word).

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.

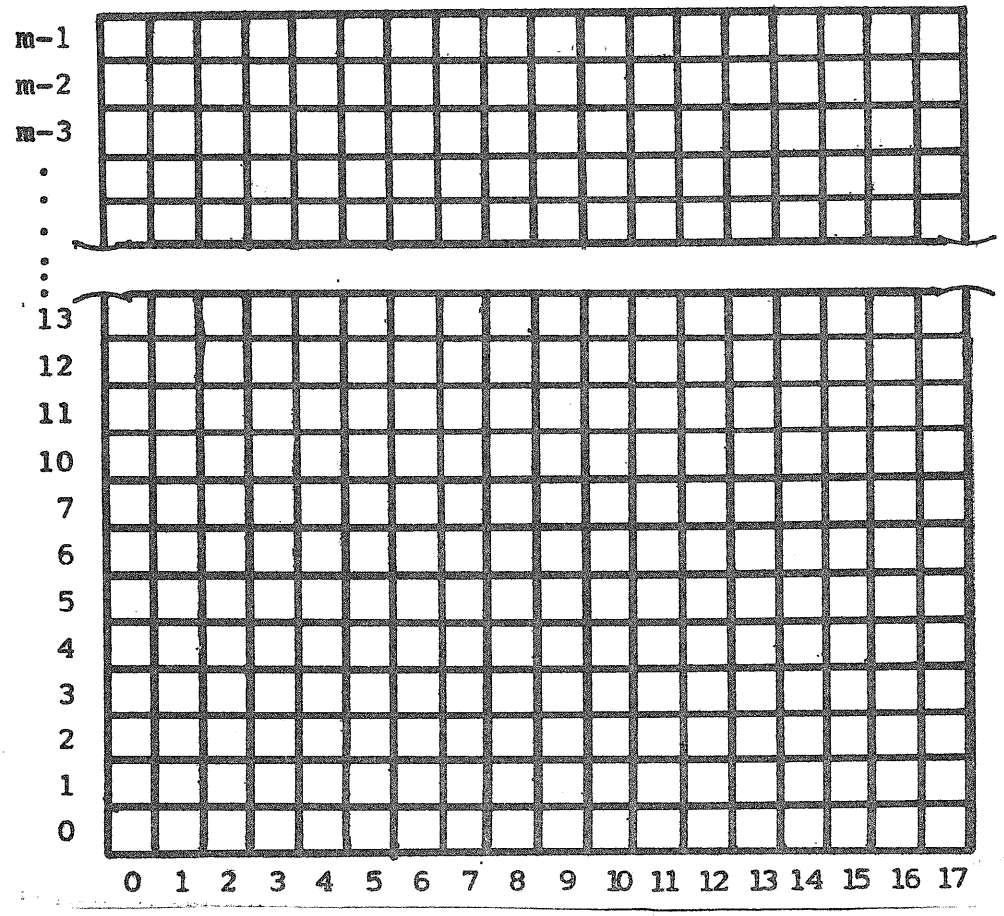
Each location or word in a PDP-15 is partitioned into smaller subdivisions called bits. Bits are subdivisions which have binary values, either 1 or 0.

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.

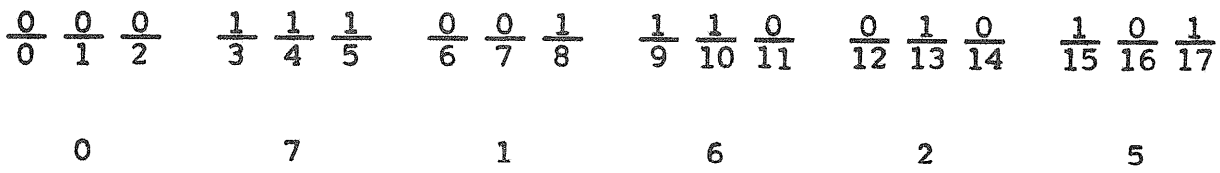


WORD LENGTH

The PDP-15 has an 18 bit word.

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

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:



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:

If 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 --

POSITIVE	NEGATIVE
000000	
000001	777777
000002	777776
⋮	⋮
100000	700000
⋮	⋮
200000	600000
⋮	⋮
300000	500000
⋮	⋮
377776	400002
377777	400001
	400000

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.

```
 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 value where the 8 is in the rightmost non-zero position of the number being complemented.

```
251043      777778          BUT      251040      777780
           -251043                              -215040
            526735                               526740
```

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 1'2 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.

Positive Number	1's Complement		2's Complement
000000	777777		------
000001	777776	(-1)	777777
000002	777775	(-2)	777776
⋮	⋮		⋮
077777	700000		700001
100000	677777		700000
100001	677776		677777
⋮	⋮		⋮
377776	400001		400002
377777	400000	(-131,071)	400001
------			400000
$2^{17}-1$			-2^{17}

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
0

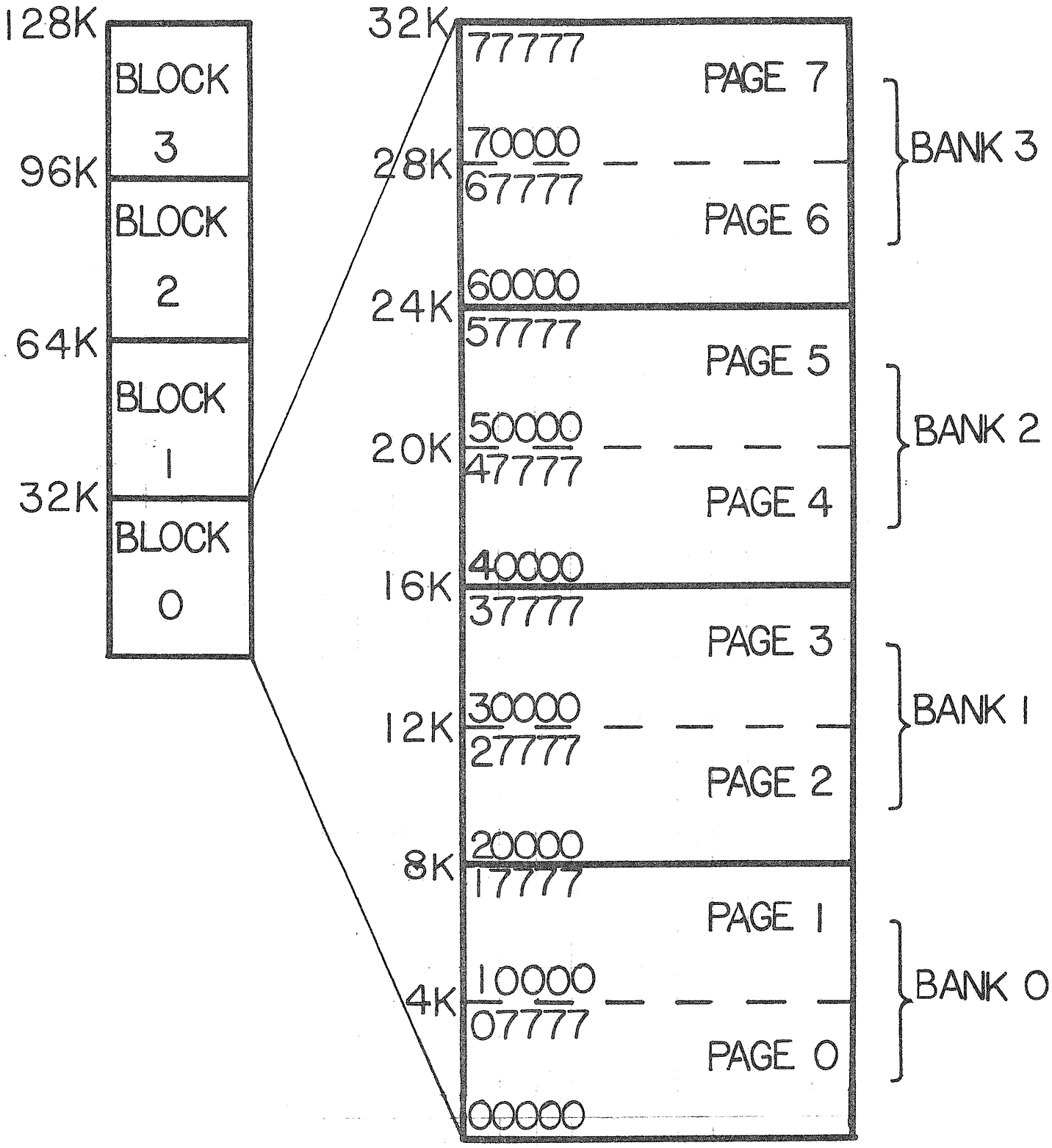
1) There are two forms of zero in 1's complement:

and 000000
777777.

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

000000 (777777 is a -1).

MEMORY ORGANIZATION



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:

NUMBER	HIGH	LOW
0025	000 000	010 101
0063	000 000	110 011
0377	000 011	111 111
-1	111 111	111 111
-100	111 111	000 000
-40	111 111	100 000

/ 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

	HIGH	LOW	
/ 127=	000 001	010 111	
/ 306=	000 011	000 110	
/ Add low order		011 101	Link = 0
/ Add high order	000 100		
/ Add in LINK	0		
/ Result	000 100	100 101	= 435

/ EXAMPLE 2. DP ADD 1254 + 2362 = 3636

	HIGH	LOW	
/ 1254=	001 010	101 100	
/ 2362=	010 011	110 010	
/ Add low order		011 110	Link = 1
/ Add high order	011 101		
/ Add in LINK	1		
RESULT	011 110	011 110	= 3636

CPU

CENTRAL PROCESSOR: 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.

IR
6 bits long
Holds Bits 0-3 OP Code
4 Indirect bit
5 Index bit

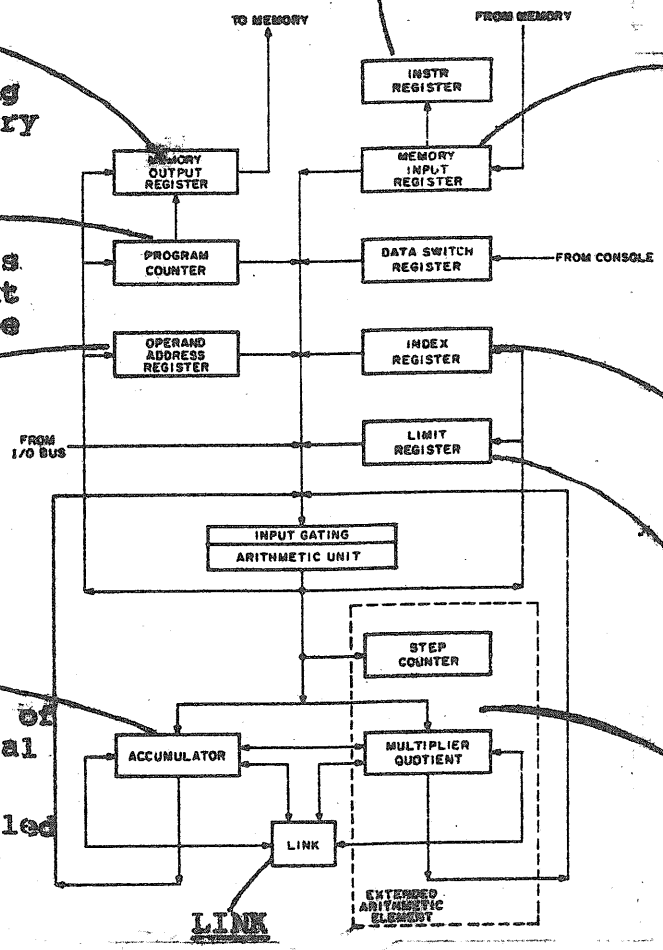
MO
Contains all information going from CPU to memory data lines

PC
Contains address in memory of next instruction to be executed

OA
Temporary storage of the operand address for all MRIs

AC
Retains results of arithmetic/logical operations

Program controlled I/O transfers go through the AC



LINK
1 bit extension of the AC

MI
All words read from memory into CPU enter MI first (data and instructions)

CONSOLE
-Data Switches
-Address Switches
-Lights
-Operate Keys

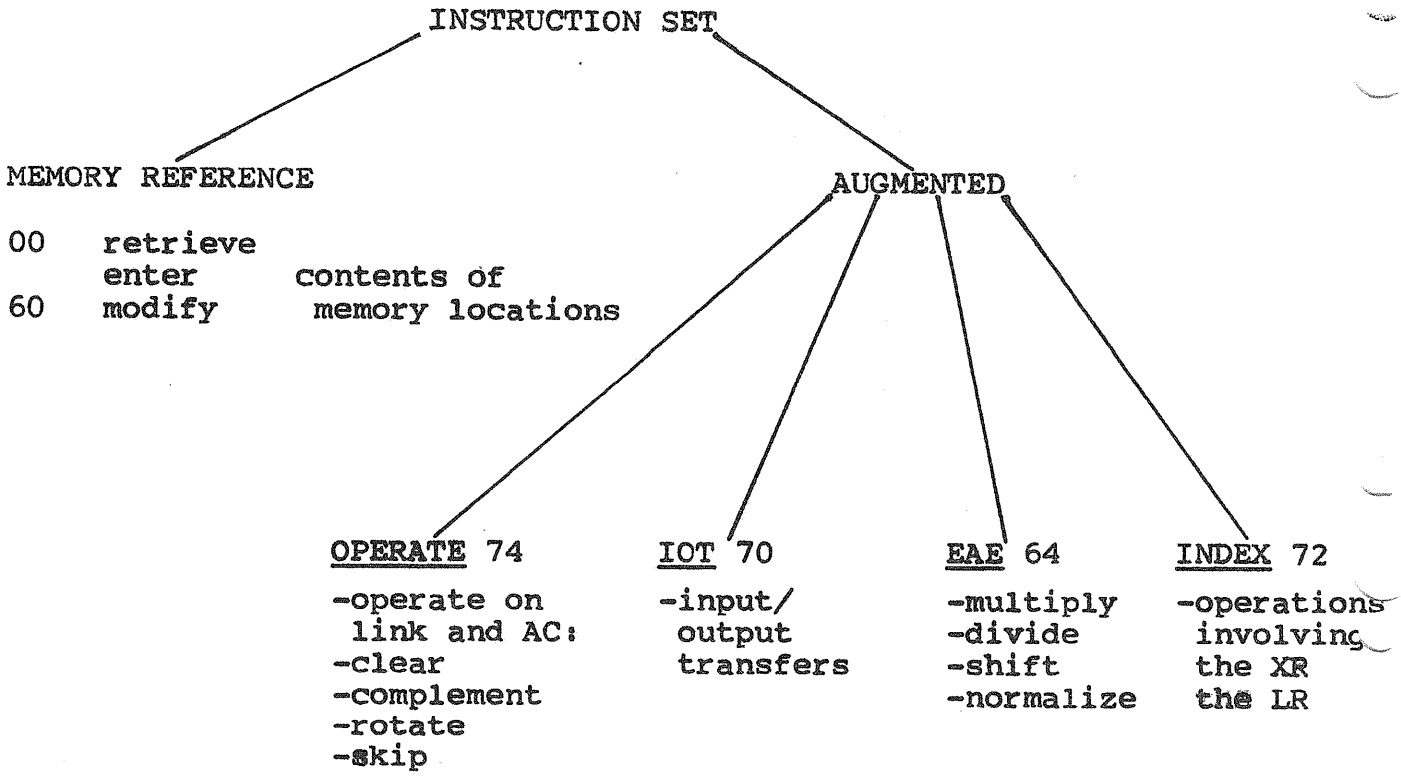
XR
Index Register used in indexed addressing operations, as a counter or for storage

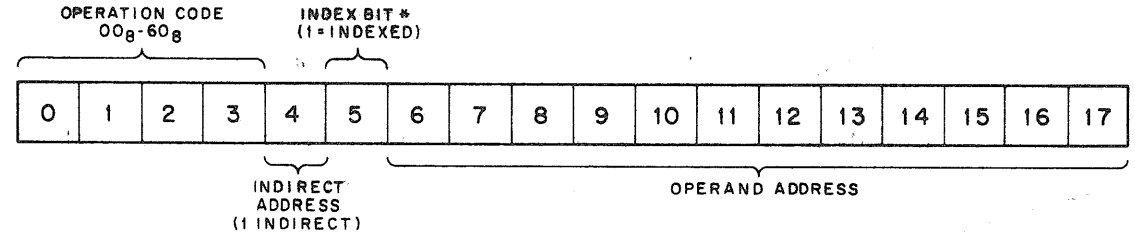
LR
Used to set a limit on operations with the Index Register or for storage

EAE
Extends capability of processor to do hardware multiply, divide, normalize and shift

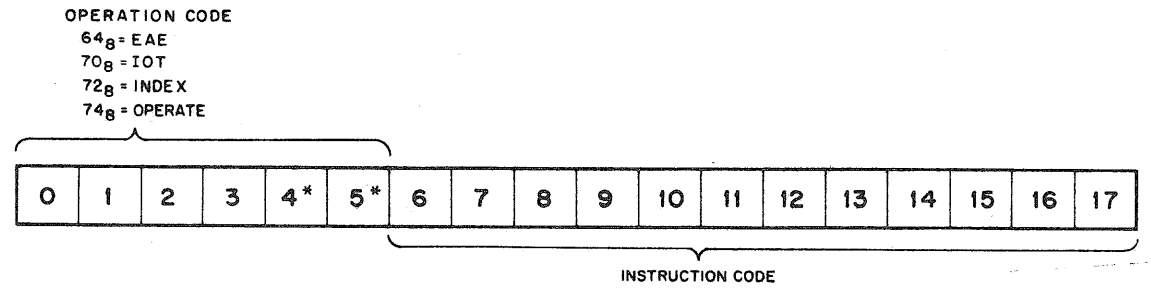
CPU Options

-Memory Protect
-Memory Protect & Relocate
-Powerfail





Memory Reference Instruction Word



Augmented Instruction Format

```
                                .TITLE MACRO ASSEMBLY LANGUAGE EXAMPLE
                        /
                        /
                        /
                                .ABS
                        /
00100                           .LOC 100
                        /
00100   600100    START   JMP .
00101   600100            JMP START
00102   000100            START
                        /
00103   140103    DZM     DZM DZM
00104   200103            LAC DZM
00105   140000            DZM
00106   000103                    DZM
00107   000103            .DSA DZM
                        /
                        /VARIABLES
00110   201025            LAC CNT#
00111   041025            DAC CNT#
00112   201026            LAC CNT2#
00113   041026            DAC CNT2
                        /
                        /LITERALS
00114   201031    A       LAC (1
00115   201024            LAC TABLE
00116   201032            LAC (TABLE
00117   201033            LAC (A
00120   201034            LAC (C
00121   001034            (C
        000200    B=200
00122   201036            LAC (B
                        /
                        /INDIRECT
00123   200100            LAC 100
00124   220100            LAC* 100
00125   220102            LAC* START+2
00126   621005            JMP* A1
00127   600100            JMP 100
00130   600100            JMP START
00131   621036            JMP* (100
                        /
                                .EJECT
```

```
                                /
                                /
     01005                              .LOC 1005
     01005    000114            A1      A
              000100            B=100
              000004            C=4
M    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 (E#
                                /
DM   01015    001006            D       D       /MULTI-DEFINED SYMBOL
U    01016    001030                    DD      /UNDEFINED SYMBOL
              000110            B=110
     01017    000110                    B
     01020    200110                    LAC B
     01021    201041                    LAC (B
                                /
     01022    000010                    A=B+C
Q    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
                        /
00100                           .LOC 100
                        /
00100   200100                  LAC 100
                        /
                        /**********************************
                        /.REPT
                        /**********************************
                        /
                                .REPT 4
00101   742010                  RTL
00102   742010   *R
00103   742010   *R
00104   742010   *R
                                .REPT 5,1
00105   140112          ZERO    DZM BUFF
00106   140113   *R
00107   140114   *R
00110   140115   *R
00111   140116   *R
                        /
                        /**********************************
                        /RESERVING STORAGE
                        /**********************************
                        /
00112                   BUFF    .BLOCK 5
00117   000000          BUFFE   0
                        /
00120   000000          TABLE   0
00126                           .LOC .+5
00126   000000          TABLEE  0
                        /
                        /**********************************
                        /CONDITIONALS
                        /**********************************
                        /
                                .IFDEF A
                                LAC A+1
                                .ENDC
        000002          B=2
                                .IFPOZ B
00127   341123                  TAD (B
                                .ENDC
                        /
                        /**********************************
                        /PAGING THE LISTING
                        /**********************************
                        /
                                .EJECT
```

```
                        /
                        /**********************************
                        /CHANGING THE LOCATION COUNTER
                        /**********************************L ****
                        /
01050                           .LOC 1050
                        /
                        /**********************************
                        /.ASCII AND .SIXBT
                        /**********************************
                        /
01050   405010          MESS    .ASCII /A BROKEN/<15><12>
01051   251236
01052   456131
01053   606424
01054   202532                  .ASCII / UP MESSAGE!/
01055   020232
01056   426472
01057   340616
01060   425020
01061   000000
                        /
01062   406050          MESS1   .ASCII 'ABC123'
01063   330544
01064   314000
01065   000000
                        /
01066   010203          NAME1   .SIXBT 'ABC123'
01067   616263
                        /
01070   406050          MESS2   .ASCII 'ABC1'
01071   330400
                        /
01072   010203          NAME2   .SIXBT 'ABC1'
01073   610000
                        /
01074   406050          MESS3   .ASCII 'AB'/C123/
01075   330544
01076   314000
01077   000000
                        /
01100   010203          NAME3   .SIXBT 'AB'/C123/
01101   616263
                        /
                        /**********************************
                        /PAGE EJECT ALSO CAUSED BY .TITLE
                        /**********************************
                        /
```

```
                                .TITLE PSEUDO OP EXAMPLES--THIS PAGE: MACROS
                        /
                        /
                        /************************************
                        /DEFINING A MACRO
                        /************************************
                        /
                                .DEFIN SUB A,B,C
                                LAC B
                                TCA
                                TAD A
                                DAC C
                                .ENDM
                        /
                        /************************************
                        /CALLING A MACRO
                        /************************************
                        /
                                SUB BUFFE,BUFF,TABLE
01102   200112  *G              LAC BUFF
01103   740031  *G              TCA
01104   340117  *G              TAD BUFFE
01105   040120  *G              DAC TABLE
                        /
                                SUB BUFFE,(5,TABLE+1
01106   201124  *G              LAC (5
01107   740031  *G              TCA
01110   340117  *G              TAD BUFFE
01111   040121  *G              DAC TABLE+1
                        /
                                SUB (MESS1,(MESS,TABLE+2
01112   201125  *G              LAC (MESS
01113   740031  *G              TCA
01114   341126  *G              TAD (MESS1
01115   040122  *G              DAC TABLE+2
                        /
                                SUB BUFFE,TABLE,TEMP*
01116   200120  *G              LAC TABLE
01117   740031  *G              TCA
01120   340117  *G              TAD BUFFE
01121   041122  *G              DAC TEMP*
                        /
        000000                  .END
01123   000002  *L
01124   000005  *L
01125   001050  *L
01126   001062  *L
            SIZE=01127      NO ERROR LINES
```

```
*****************
*               *
*  SUM  GROUP   *
*               *
*****************
```

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:

```
(2-1;2-15)      *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
(2-15)                  USING THE SYMBOL AS A LABEL
(2-6)                   USING DIRECT ASSIGNMENTS

(2-16)          *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 WHIC
/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 ".EJECT" 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 TE
                        /
                                .ABS        /ABSOLUTE PROGRAM; LOADED BY THE
                                            /ABOLUTE LOADER; TO RUN IN BANK MODE
    00100                       .LOC 100    /SET LOCATION COUNTER TO 100
                        /
    00100   777773      START  LAW -5              /SET UP NEGATIVE COUNTER OF -5
    00101   040133              DAC COUNT          /AS EACH TABLE HAS 5 ENTRIES.
                        /
    00102   200113      AA     LAC A              /PERFORM ADDITION
    00103   340120      BB     TAD B              /OF ENTRIES
                        /
    00104   040125      CC     DAC C              /STORE RESULT IN C.
                        /
    00105   440102             ISZ AA             /MODIFY LOCATIONS AA,BB AND CC
    00106   440103             ISZ BB             /SO THAT THEY REFERENCE THE
    00107   440104             ISZ CC             /NEXT LOCATION IN THE TABLE.
                        /
    00110   440133             ISZ COUNT#         /ARE WE DONE?
                        /
    00111   600102             JMP AA             /NO!--BECAUSE WE DIDN'T SKIP
                                                  /GO BACK FOR MORE.
                        /
    00112   740040             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
                        /
    00113   000001      A      1; 2; 3; 4; 5
    00114   000002
    00115   000003
    00116   000004
    00117   000005
    00120   000002      B      2; 2; 2; 2; 2
    00121   000002
    00122   000002
    00123   000002
    00124   000002
                        /JUST RESERVE SPACE FOR TABLE C
                        /
    00125               C      .BLOCK 5
                        /
    00132   000000      ENDLOC 0
                        /
                        /THIS IS THE END OF THE PROGRAM.
                        /THEREFORE, THIS IS WHERE WE WANT TO PUT
                        /THE ".END" STATEMENT.
                        /
            000100              .END START
                SIZE=00134     NO ERROR LINES
```

```
                .TITLE COMMENTARY ON SUM1
/                       **************************
/                       *  POINTS TO BE NOTED    *
/                       **************************
/
/*1. THIS PROGRAM USES INDIRECT ADDRESSING THROUGH
/LOCATIONS AA,BB AND CC.  LOCATION AA CONTAINS THE
/ADDRESS OF TABLE A.  NOTE THAT THIS IS ACCOMPLISHED
/SIMPLY BY PLACING THE SYMBOL "A" IN THE OPERATOR
/FIELD.  WHEN THE ASSEMBLER EVALUATES THIS STATEMENT,
/IT LOOKS THROUGH ITS SYMBOL TABLES FOR THE VALUE OF
/SYMBOL "A". ***NOTE THAT THE VALUE OF THE SYMBOL "A"
/IS 113, THE ADDRESS OF THE LOCATION IT NAMES----
/NOT THE CONTENTS OF THE LOCATION IT NAMES.  THIS IS
/A VERY IMPORTANT DISTINCTION!!
/     SIMILARLY, LOCATION BB CONTAINS THE ADDRESS OF
/TABLE B BECAUSE THE SYMBOL "B" IS GIVEN AS THE CONTENTS
/OF LOCATION BB.  THE VALUE OF THE SYMBOL B IS THE
/ADDRESS OF THE LOCATION IT NAMES, 120, NOT THE CONTENT
/OF THAT LOCATION.  SIMILARLY FOR LOCATION CC. THE
/CONTENT OF LOCATION CC IS GIVEN AS "C".  THE VALUE OF
/THE SYMBOL "C" IS 130, THE ADDRESS OF THE LOCATION
/IT NAMES.  HENCE, THE CONTENT OF LOCATION CC IS 130.
/
/*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
/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)
/
/*5. THINK ABOUT THE "LAC* AA" INSTRUCTION IN LOC 102.
/THIS SAYS: LOAD THE ACCUMULATOR WITH THE CONTENT OF
/THE LOCATION POINTED TO BY LOCATION AA.  THE FIRST TIME
/THRU, LOCATION AA CONTAINS 113,THE ADDRESS OF THE FIRST
/LOCATION IN TABLE A.  IN LOCATION 105, THERE IS AN
/"ISZ AA",I.E. INCREMENT THE CONTENT OF LOCATION AA BY 1
/(SO THAT AA NOW POINTS TO THE NEXT ENTRY IN TABLE A).
/THUS, THE NEXT TIME "LAC*AA" IS EXECUTED, THE AC IS
/LOADED WITH THE NEXT ENTRY FROM TABLE A.  THIS METHOD
/IS VERY NICE FOR STEPPING THROUGH TABLES. NOTE,HOWEVER,
/THAT WE HAVE TO UPDATE LOCATION AA OURSELVES WITH THE
/"ISZ" INSTRUCTION.  BY USING AUTO-INCREMENT REGISTERS
/WE CAN GET AROUND THIS. *** SEE SOLUTION SUM2.
/
```

```
                                .TITLE SOLUTION USES INDIRECT ADDRESSING
                        /
                        /
                                .ABS
00100                           .LOC 100
                        /
00100   777773    START   LAW -5              /SET UP A COUNTER OF -5
00101   040135            DAC COUNT#
                        /
00102   220125    NEXT    LAC* AA             /LOC AA CONTAINS THE ADD.
                                              /OF A.  LAC* AA GETS THE
                                              /CONTENTS OF A IN AC.
00103   360126            TAD* BB             /ETC. FOR BB
00104   060127            DAC* CC             /ETC. FOR CC
                        /
00105   440125            ISZ AA              /INCREMENT THE INDIRECT
00106   440126            ISZ BB              /ADDRESSES OF AA,BB
00107   440127            ISZ CC              /AND CC.
                        /
00110   440135            ISZ COUNT           /SEE IF ALL DONE.
00111   600102            JMP NEXT            /NOT DONE. GET NEXT DATA.
00112   740040            HLT                 /ALL DONE SO HALT.
00113   000001    A       1, 2, 3, 4, 5
00114   000002
00115   000003
00116   000004
00117   000005
00120   000002    B       2, 3, 4, 5, 6
00121   000003
00122   000004
00123   000005
00124   000006
00125   000113    AA      A
00126   000120    BB      B
00127   000130    CC      C
                        /
00130               C       .BLOCK 5            /RESERVE SPACE FOR TABLE C.
                        /
        000100              .END START
            SIZE=00136      NO ERROR LINES
```

```
                .TITLE COMMENTARY ON SUM2
/
/                    **************************
/                    *                        *
/                    *  POINTS TO BE NOTED    *
/                    *                        *
/                    **************************
/
/
/*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 PRO-
/GRAM 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, WHEREA
/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*1 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
/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 "(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 "LAC (B-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 20000" RATHER THAN ".LOC 100"?
/
```

```
                                .TITLE SOLUTION USES AUTO-INCREMENT ADDRESSING
                        /
                        /
                        /
                                .ABS
00100                           .LOC  100
                        /
00100   000001         A       1; 2; 3; 4; 5
00101   000002
00102   000003
00103   000004
00104   000005
00105   000002         B       2; 4; 1; 5; 3
00106   000004
00107   000001
00110   000005
00111   000003
00112                  C       .BLOCK 5
                        /
00117   777773         START   LAW  -5          /PUT -5 INTO COUNT
00120   040135                 DAC  COUNT#
                        /
00121   200136                 LAC  (A-1        /THIS METHOD MAKES USE OF
00122   040010                 DAC  10          /AUTO-INCREMENT REGISTERS
                        /
00123   200137                 LAC  (B-1        /10,11 AND 12. SINCE
00124   040011                 DAC  11          /AUTO-INCREMENT IS A PRE-
                        /
00125   200140                 LAC  (C-1        /INCREMENT, EACH OF THE
00126   040012                 DAC  12          /REGISTERS MUST BE LOADED WITH
                        /
00127   220010         NXT     LAC* 10          /ONE LESS THAN THE STARTING ADD
00130   360011                 TAD* 11          /
00131   060012                 DAC* 12          /
00132   440135                 ISZ  COUNT       /TO ASSURE THAT THE LOADING OF
                        /
00133   600127                 JMP  NXT         /AN AUTO-INCREMENT REGISTER WIL
00134   740040                 HLT              /WORK IN ANY PAGE OR BANK. THE
                                                /INSTRUCTION TO LOAD LOCS 10,11
                                                /AND 12 SHOULD BE OF THE FORM
                                                /      LAC  (A-1
                                                /      DAC* (10 — go after anders
                                                                  PAGES or Banks than 0
        000117                  .END  START
00136   000077    *L
00137   000104    *L
00140   000111    *L
            SIZE=00141      NO ERROR LINES
```

```
                .TITLE COMMENTARY ON SUM3
/
/                  **************************
/                  *                        *
/                  *  POINTS TO BE NOTED    *
/                  *                        *
/                  **************************
/
/*1. NOTE THE USE OF ".ABSP".  THIS PROGRAM MAKES USE OF
/INDEXED ADDRESSING AND,THEREFORE,¡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 EVALUA-
/TED 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)?
/
```

```
                                .TITLE ROUTINE USES INDEXED ADDRESSING (& LIMIT
                        /
                        /
                                .ABSP
    10000                       .LOC 10000
                        /
    10000   707762              DBA                /ENTER PAGE MODE
                        /
    10001   200033      BEGIN   LAC (5
    10002   722000              PAL                /PUT 5 IN LIMIT REG.
                        /
    10003   735000              CLX                /XR=0
                        /
    10004   210014      AA      LAC A,X            /ADDRESS OF A + C(XR).
    10005   350021              TAD B,X            /ETC.
    10006   050026              DAC C,X            /ETC.
                        /
    10007   725001              AXS 1              /INCREMENT AND TEST XR
    10010   600004              JMP AA             /XR < LR
    10011   740040              HLT                /XR=LR.  WE'RE DONE!
                                                   /CHECK RESULTS VIA CONSOLE.
    10012   000000              CAL                /RETURN TO MONITOR
    10013   000015              15                 /BY HITTING CONSOLE CONTINUE.
                        /
                                .DEC
    10014   000002      A       2; 5; 12; 27; 58
    10015   000005
    10016   000014
    10017   000033
    10020   000072
                                .OCT
    10021   000002      B       2; 5; 12; 27; 58
    10022   000005
    10023   000012
    10024   000027
N   10025   000072
    10026               C       .BLOCK 5
                        /
            010000              .END BEGIN=1
    10033   000005   *L
                SIZE=10034       1 ERROR LINES
```

```
/PROGRAM TO ADD TWO NUMBERS, USING TAD, AND TEST
/THE RESULT FOR ARITHMETIC OVER FLOW.
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.
                          /BECAUSE ONES IN BOTH SIGN POSITIONS IS
                          /THE ONLY WAY THE LINK CAN BE SET SINCE ONLY
                          /THE SIGN BIT OF THE 2ND # IS ADDED
         RAL              /GET RESULT SIGN INTO LINK.
         SZL              /SKIP IF ZERO LINK
         JMP POSNEG       /LINK=1, MUST BE +-.
         JMP POSPOS       /LINK=Ø, MUST BE ++.
POSNEG   LAC A            /SINCE ONE VALUE IS + ADD
         TAD B            /THE OTHER IS -.  CORRECT ADDITION IS ASSURED.
         DAC SUM          /NO TEST IS NECESSARY, SO STORE THE SUM
         HLT              /AND HALT.
POSPOS   LAC A            /IF THE SUM OF TWO POSITIVE NUMBERS GIVES
         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            /IF THE SUM OF TWO NEGATIVE NUMBERS GIVES
         TAD BQ           /A POSITIVE RESULT, ARITHMETIC OVERFLOW HAS
         SMA              /OCCURED, TEST THE AC FOR NEGATIVE VALUE.
         JMP NEGERR       /IF POSITIVE, GO TO ERROR ROUTINE.
         DAC SUM          /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            /BITS IS A 1, THE SIGNS WERE
         SMA              /DIFFERENT AND THE SUM MUST BE CORRECT
         JMP LIKE         /SIGNS WERE THE SAME
         LAC A
         TAD B
         DAC SUM
         HLT
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

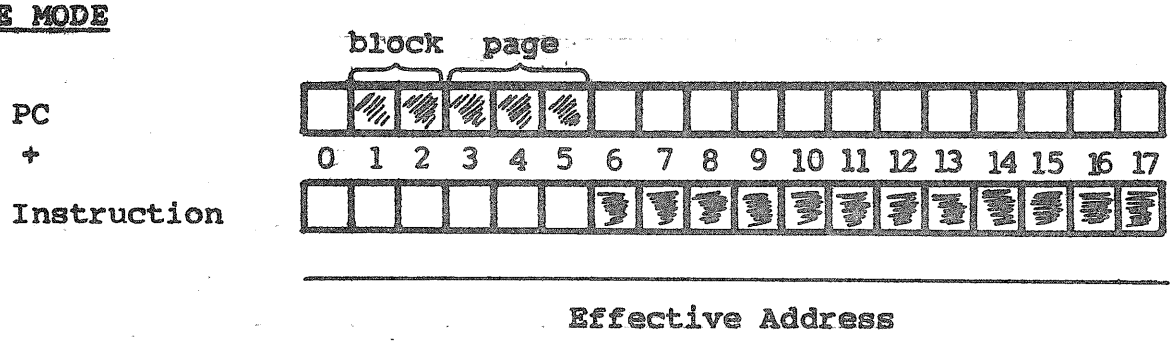
Memory Reference Instructions specify locations to be operated on by the Central Processing Unit. The CPU computes the actual (effective) address of the location referred to by combining bits from the instruction itself and also from the PC at the time the instruction is being executed. Various addressing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

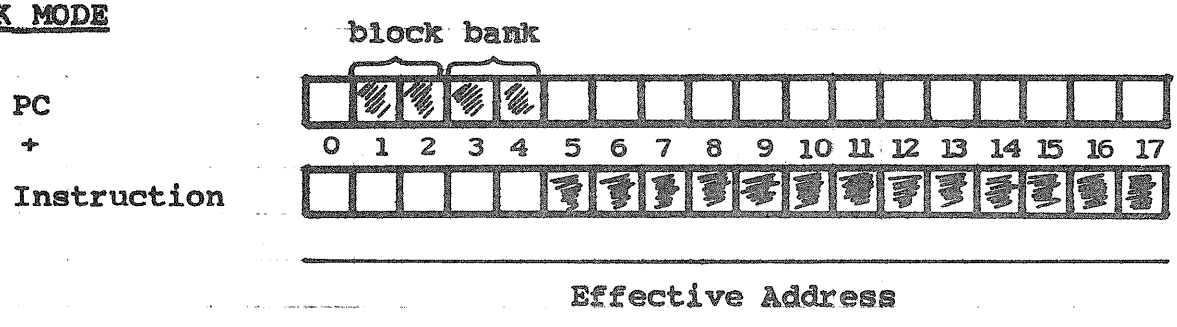
DIRECT ADDRESSING

PAGE MODE



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

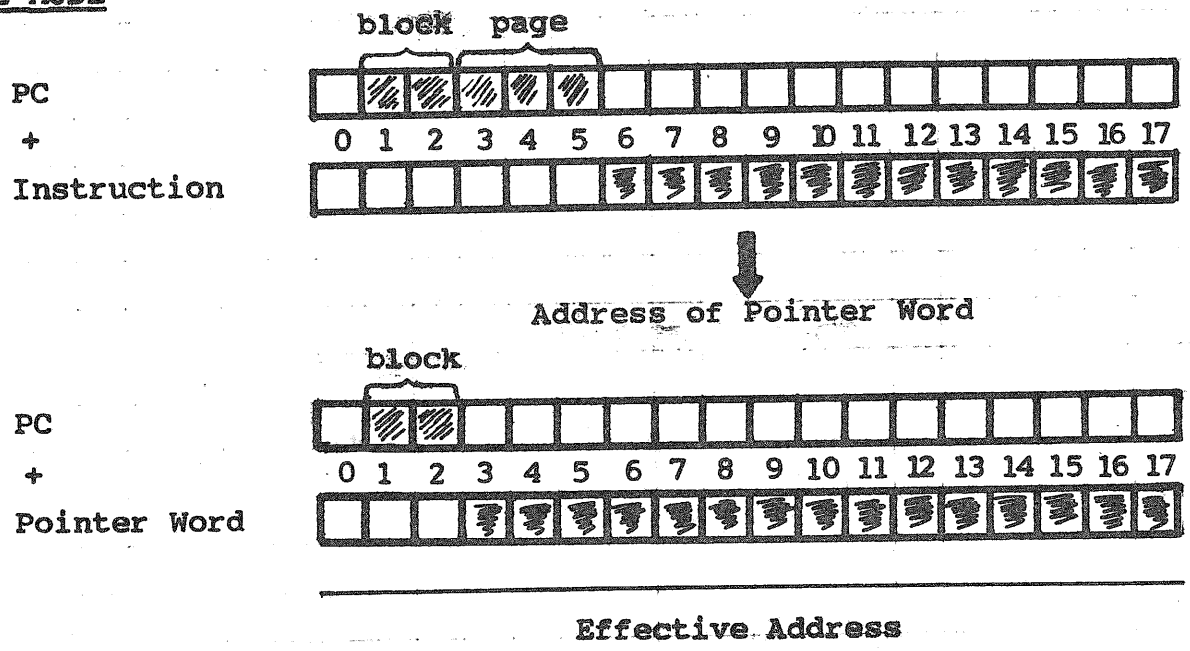
BANK MODE



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

INDIRECT ADDRESSING

PAGE MODE

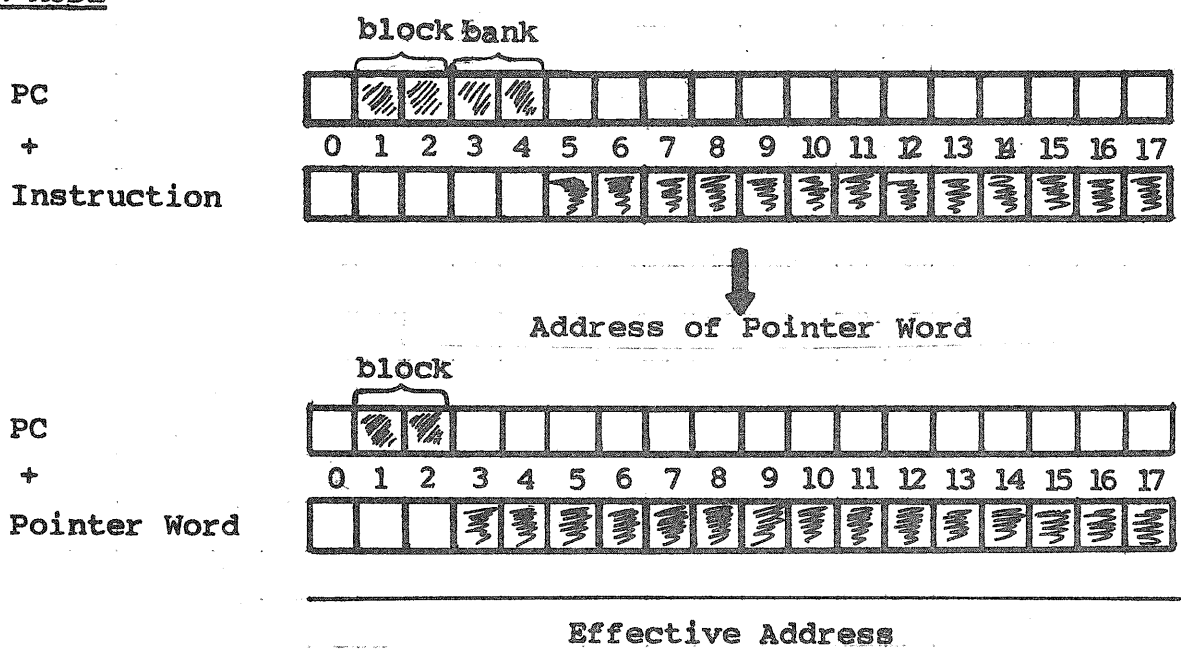


NOTE that the pointer word is in the same page as the instruction.

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

INDIRECT ADDRESSING

BANK MODE

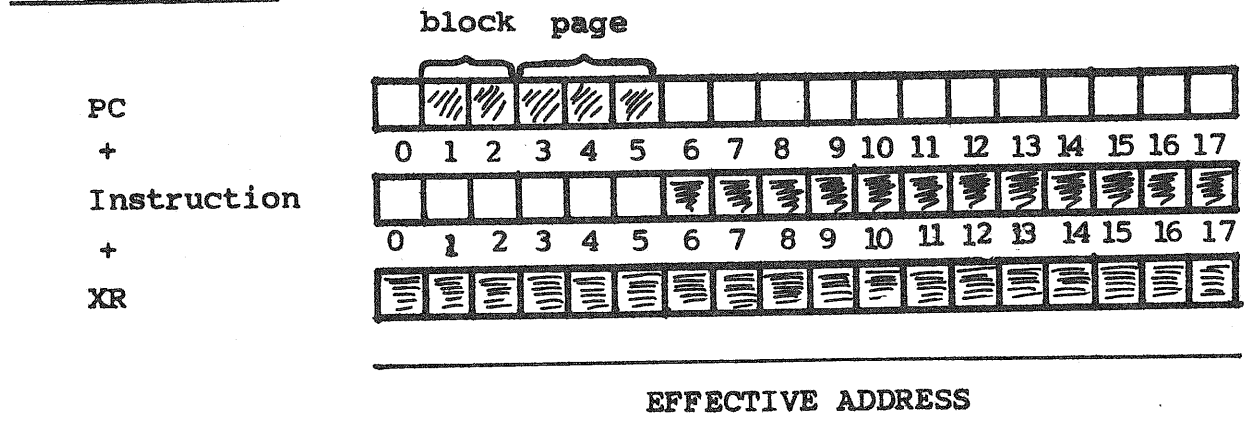


NOTE that the pointer word is in the same bank as the instruction.

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

INDEXED ADDRESSING

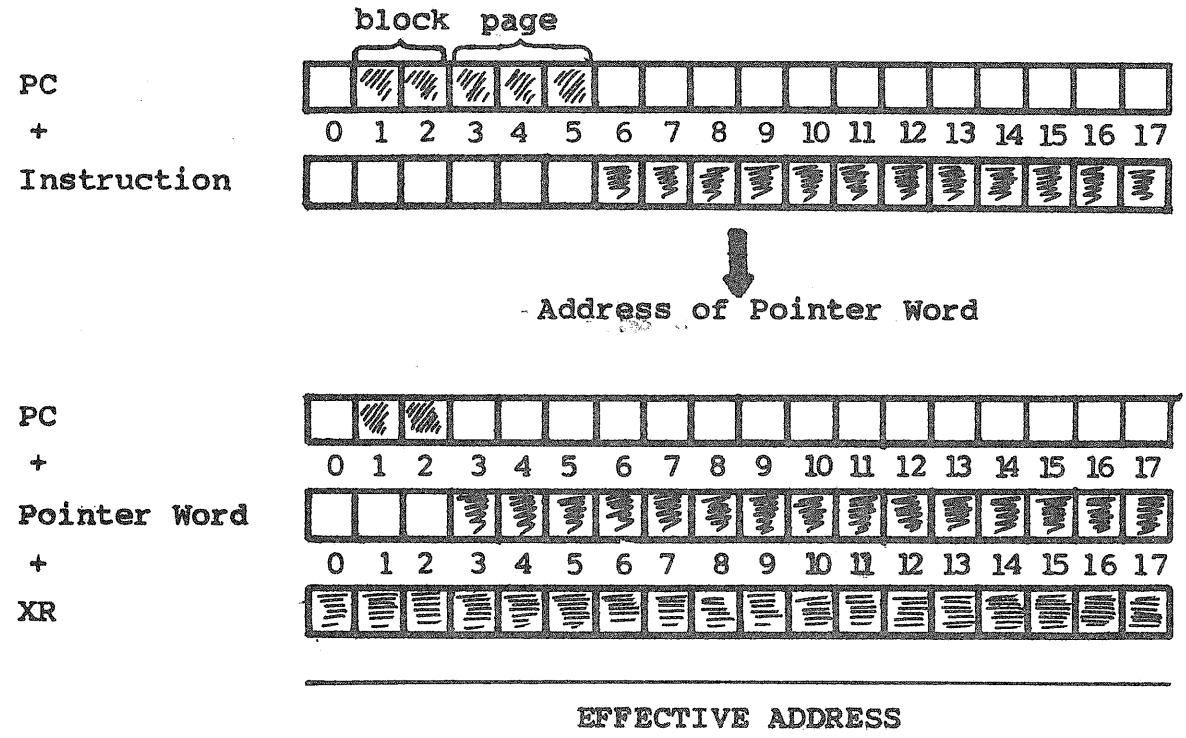
PAGE MODE ONLY



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

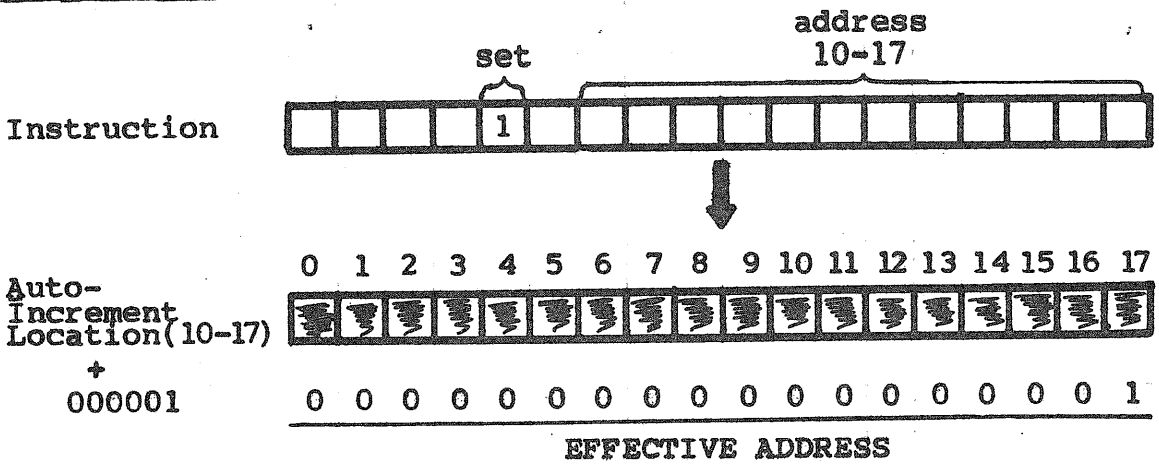
PAGE MODE ONLY



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

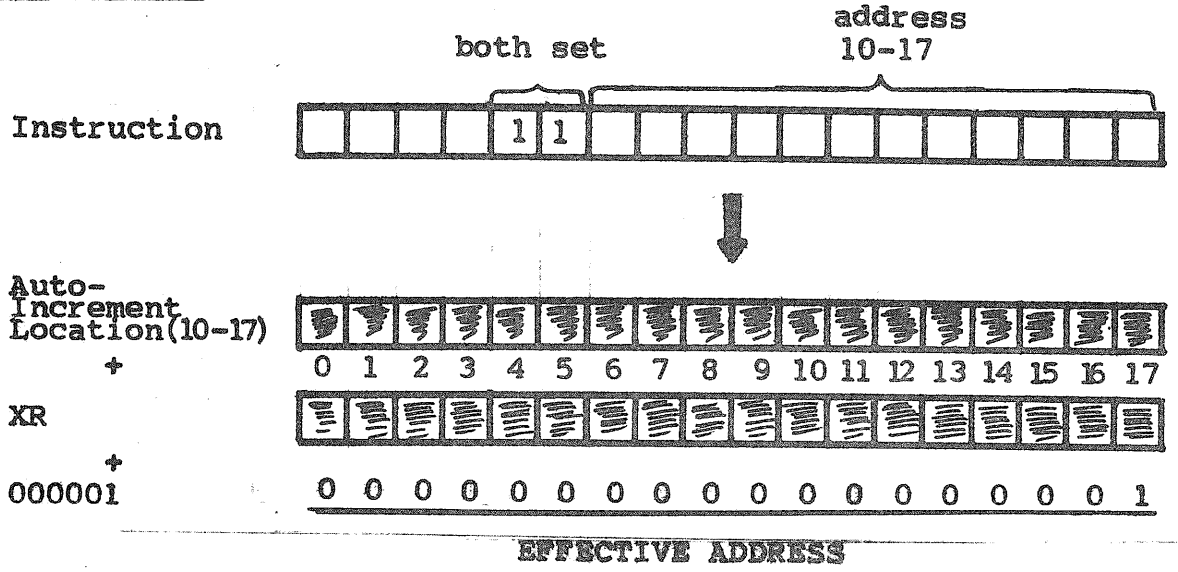
PAGE MODE and BANK MODE



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



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

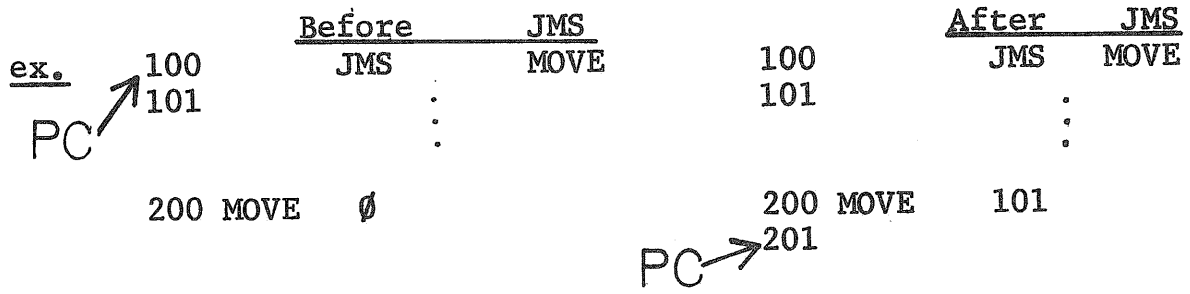
SUBROUTINES

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.



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
2) MQ,XR,LR

} passing arguments in CPU 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).

```
        LAC    (LIST ADDRESS        or        JMS    SUB          etc.
        JMC     SUB                           LIST   ADDRESS
                .
                .
                .

LIST ADDRESS  ARG1
              ARG2
              .
              .
              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 50_8 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 1-1
Indicators Associated with Paper-Tape Reader

Indicator	Function
READER BUFFER 00-17	Indicates the contents of the paper-tape reader buffer.
API 2 RDR	Denotes API level 2 is active as the result of a reader interrupt.
I/O ADDRESS	Indicates the unique trap address associated with I/O devices; address 50_8 for paper-tape reader.
RDR FLG	Denotes information has been read from tape and is available for transfer from reader buffer.

Table 1-1 (Cont)
Indicators Associated with Paper-Tape Reader

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

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.

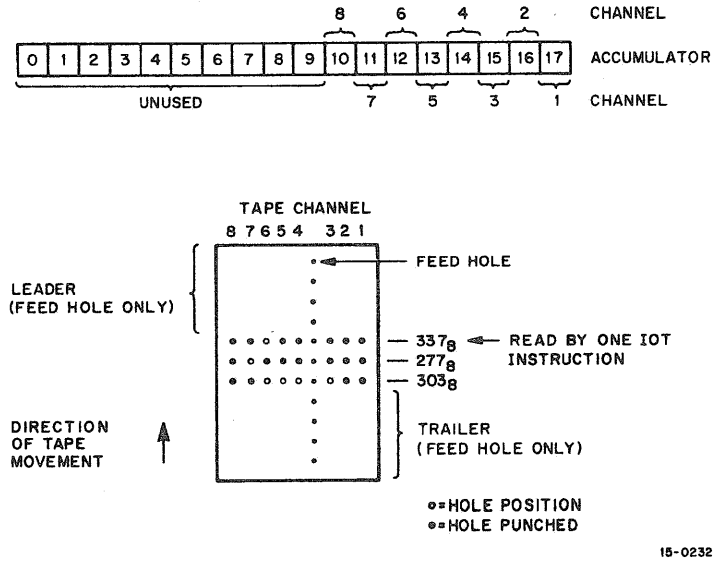


Figure 1-1 Tape Format and Accumulator Bits (Alphanumeric Mode)

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

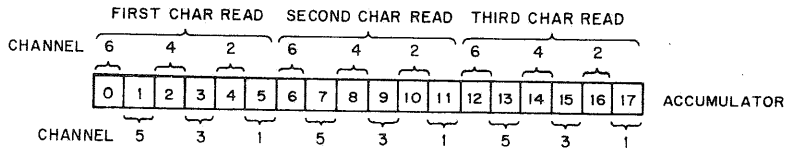
The PDP-15 IOT instructions used for program-controlled loading of paper-tape data are listed below. Refer to Volume 1 of this handbook for IOT instruction format.

Mnemonic	Octal Code	Operation Performed
RSF	700101	Skip next instruction if reader flag is a 1.
RCF	700102	Clear reader flag. Read reader buffer, inclusively OR contents of reader buffer with AC, and deposit result in AC.
RRB	700112	Read reader buffer and clear reader flag. Clear AC and transfer contents of reader buffer to AC.
RSA	700104	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.
RSB	700144	Select binary modes. Assemble three 6-bit characters in reader buffer. Clear reader flag during assembly and set flag when assembly is complete.

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.



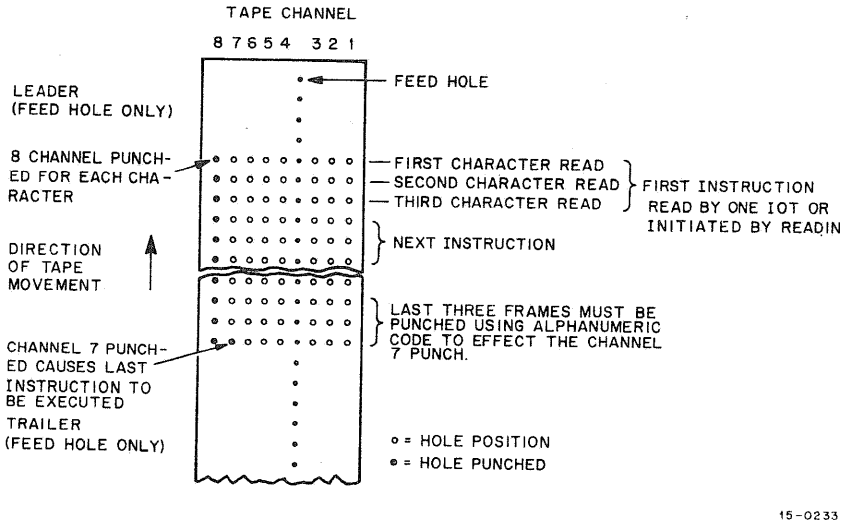


Figure 1-2 HRI Tape Format and Accumulator Bits (Binary Mode)

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 (700104_8), 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 (700102_8) or RRB (700112_8) 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 readin 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.

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

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 readin 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   (ADRESS
        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 A 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                 /ROUTINE STORES 2'S
          DAC    WORDCNT      /COMPLEMENT WORDCNT
          CLX                 /AND CLEARS XR.
NXTWORD   LAW    -3           /SET UP A COUNTER FOR
          DAC    COUNT        /3 CHARACTERS.
          LAC    ADDRESS,X    /USE XR TO GET EACH WORD.
          RAL                 /AC HOLDS 3 6-BIT ASCII
                              /CHARS. ROTATE INTO LINK.
NXTCHAR   RTL                 /ROTATE WORD 6 PLACES
          RTL                 /THRU LINK. THE NEXT
          RTL                 /6-BIT CHAR. IS IN AC12-17.
          DAC    SAVEAC       /SAVE REMAINING CHARS.
          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'
                              /TEST.
          IORS                /LOAD PUNCH STATUS INTO AC.
          AND    (400)        /TEST NO PUNCH TAPE BIT.
          SZA                 /SKIP IF TAPE OK.
          JMP    EOT          /GO TO END OF TAPE RTE.
```

```
        LAC     STORE           /LOAD AC WITH CHARACTERS
        PSA                     /SELECT ALPHA MODE & PUNCH
        PSF                     /WAIT FOR FLAG.
        JMP     .-1
        PCF
        JMP*    PPCHAR          /RETURN TO SUBPROGRAM.
```

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

```
                RSB                     /ISSUE READER SELECT
                 .                      /BINARY IOT WITH PI ENABLE.
                 .                      /REST OF USER PROGRAM.
                .LOC    0
PI              0                       /SAVE PC, LINK, EXTEND MODE
                                        /& MEM. PROT. BITS AT LOC0.
                JMP     SKPCHN          /GO TO SKIP CHAIN.
                 .
                 .
SKPCHN          SPFAL                   /POWER FAIL FLAG TEST.
                SKP                     /GO TO NEXT TEST.
                JMP*    INT6            /GO TO POWER FAIL SUBRTE.
                RSF                     /PAPER-TAPE READER DONE?
                SKP                     /GO TO NEXT TEST.
                JMP*    INT2            /GO TO PTR INTERRUPT.
                PSF                     /PAPER-TAPE PUNCH DONE?
                SKP                     /GO TO NEXT TEST.
                JMP*    INT3            /GO TO PTP INTERRUPT.
                 .                      /OTHER TESTS
                 .
                 .
/INT6, INT2, AND INT3 ARE PART OF A TABLE
/OF INTERRUPT SERVICE ROUTINE STARTING ADDRESSES.
/AN EXAMPLE OF INT2 FOLLOWS:
```

```
INT2        PTRPIC                      /15-BIT ADDRESS OF PAPER
              .                         /TAPE READER SERVICE
              .                         /ROUTINE.
              .
              .                         /OTHER I/O SERVICE ROUTINE
              .                         /POINTERS
              .
PTRPIC      DAC     PTRAC               /SAVE AC.
            LAC*    (0                  /SAVE PC, LINK, BANK MODE
              .                         /AND USER MODE IN PTROUT.
              .
              .                         /REST OF INTERRUPT HANDLED.
              .
```

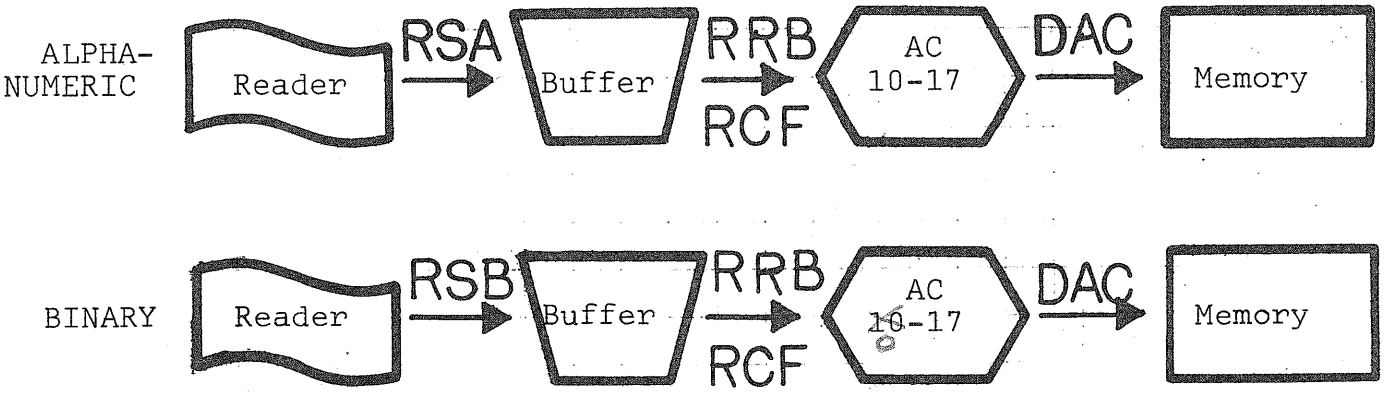
1.5.4.2 API Example

```
            RSB                         /SELECT READER IN
              .                         /BINARY MODE
              .
              .                         /REST OF INTERRUPT
                                        /HANDLED.
            .LOC    50
            JMS     PTRINT              /PAPER TAPE READER
              .                         /API ENTRY LOCATION
              .
              .
PTRINT      0
            DAC     PTRAC               /API ENTRY.  SAVE AC.
            LAC     PTRINT              /SAVE PC, LINK, BANK
                                        /MODE & USER MODE BITS.
            DAC     PTROUT              /
```

PAPER TAPE IOTS

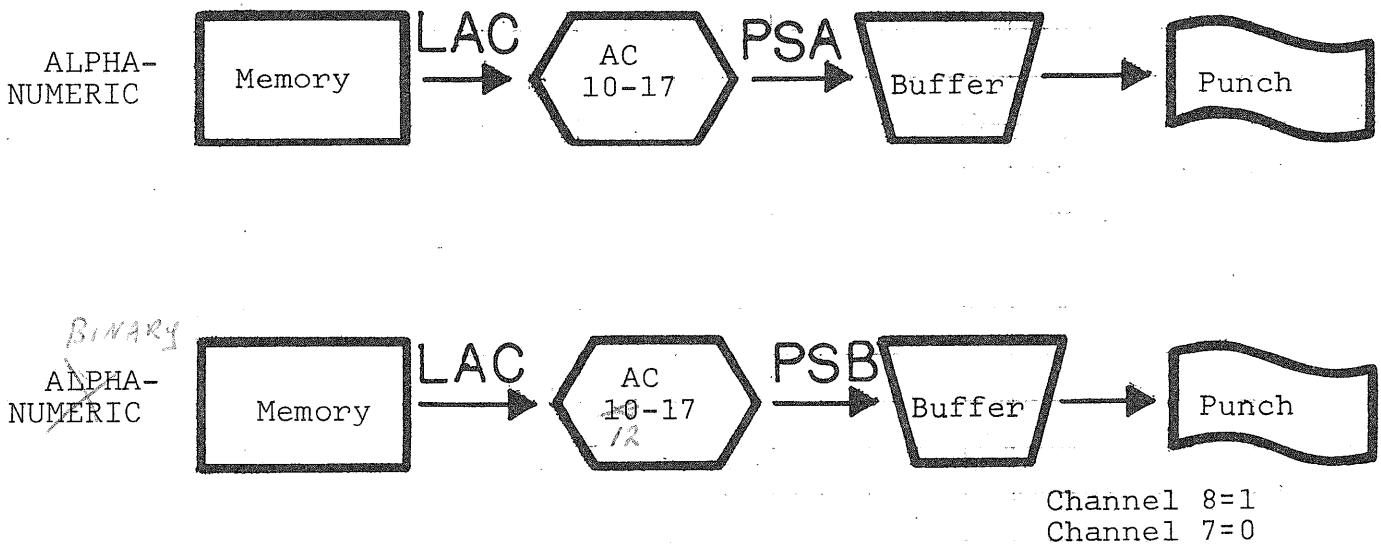
READER

INPUT



PUNCH

OUTPUT



Channel 8=1
Channel 7=0

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

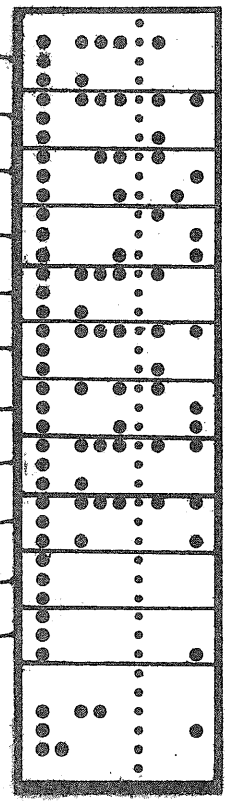
DATA

.
.
.

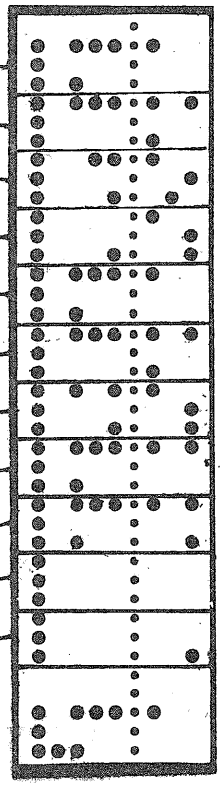
START BLOCK - A two word block at the end of the tape. It is distinguished from a Data Block because bit Ø of the first word is a one (i.e. channel #6 in the first of 3 frames is punched).

1. Starting address (777777 means "HLT" rather than "JMP" to some location)
2. Dummy word (not used)

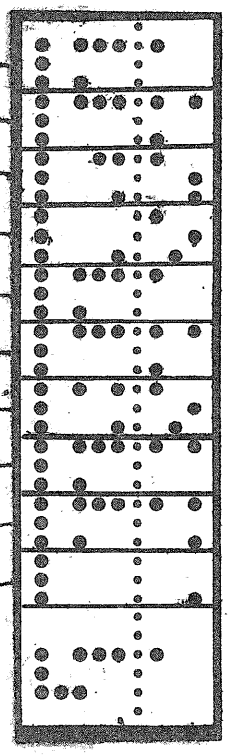
```
                                .FULL
00100                           .LOC 100
00100   740040      BEG       HLT
00101   750004                LAS
00102   340112                TAD ONE
00103   040111                DAC TEMP
00104   740040                HLT
00105   750004                LAS
00106   540111                SAD TEMP
00107   750040                CLA!HLT
00110   750041                CMA!CLA!HLT
00111   000000      TEMP      0
00112   000001      ONE       1

        000100                .END BEG
            SIZE=00113      NO ERROR LINES
```



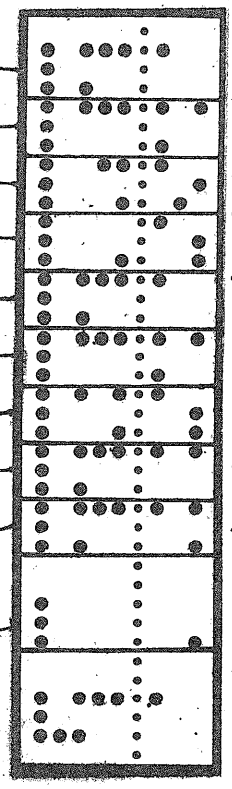
```
                                .FULL
00100                           .LOC 100
00100   740040      BEG       HLT
00101   750004                LAS
00102   340112                TAD ONE
00103   040111                DAC TEMP
00104   740040                HLT
00105   750004                LAS
00106   540111                SAD TEMP
00107   750040                CLA!HLT
00110   750041                CMA!CLA!HLT
00111   000000      TEMP      0
00112   000001      ONE       1
        000000                .END
            SIZE=00113      NO ERROR LINES
```



```
                                .FULL
00100                           .LOC 100
00100   740040      BEG       HLT
00101   750004                LAS
00102   340111                TAD ONE
00103   040112                DAC TEMP#
00104   740040                HLT
00105   750004                LAS
00106   540112                SAD TEMP
00107   750040                CLA!HLT
00110   750041                CMA!CLA!HLT

00111   000001      ONE       1

        000000                .END
            SIZE=00113        NO ERROR LINES
```



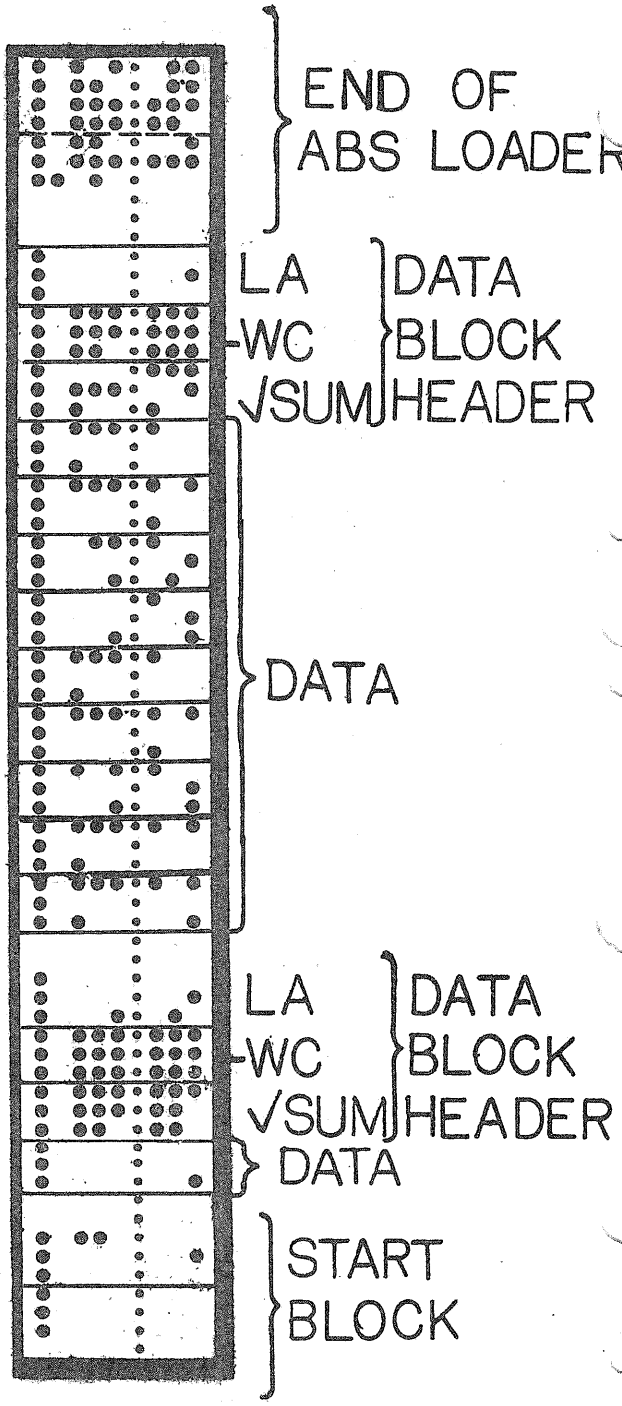
```
                                .FULL

00100                           .LOC 100
00100   740040      BEG       HLT
00101   750004                LAS
00102   340112                TAD (1
00103   040111                DAC TEMP#
00104   740040                HLT
00105   750004                LAS
00106   540111                SAD TEMP
00107   750040                CLA!HLT
00110   750041                CMA!CLA!HLT

        000000                .END
00112   000001    *L
            SIZE=00113      NO ERROR LINES
```



DO YOU SEE WHAT THE PROBLEM IS WHEN THIS IS LOADED?

CHECK LOCATIONS

```
PAGE   1     ABSVER SRC

                                .ABS
    00100                       .LOC 100
    00100   740040      BEG    HLT
    00101   750004             LAS
    00102   340112             TAD (1
    00103   040111             DAC TEMP#
    00104   740040             HLT
    00105   750004             LAS
    00106   540111             SAD TEMP
    00107   750040             CLA!HLT
    00110   750041             CMA!CLA!HLT
            000100             .END BEG
    00112   000001    *L
                SIZE=00113      NO ERROR LINES
```



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 Ø. 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.

```
$PIP
DOSPIP VXX
>L␣TT ← DT1
 N␣DT2
```

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

PIP

To call PIP

```
$PIP
DOSPIP VXX
>
```

To list the directory from the disk on the teletype:

```
>L TT ← DK↵
```

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

```
>L␣LP ← DT2↵
```

To transfer a file from the disk to dectape:

```
>T␣DT1 ←DK␣TESTFL SRC↵

>T␣DT2␣FILEA BIN ← DK␣FIL1␣BIN↵
```

MACRO

To call MACRO

```
$MACRO
MACRO VXX
>BN ← FILENM↵
                OR ALTMODE
```

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.

```
DOS-15 V3A000
$L
      ************************************************************************

        EXAMPLE OF WRITING,ASSEMBLING AND LOADING A RELOCATABLE PROGRAM

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


$EDIT



 EDITOR V3A000
>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

$A DK -13

$K ON
```

```
$MACRO

BMACRO-15 V3A000
>BL←SAMPLE
 END OF PASS 1
SIZE=00003     NO ERROR LINES  PAGE   1      SAMPLE SRC

                                00000 R 200002 R     START    LAC (707070
                                00001 R 740040 A              HLT
                                        000000 R              .END START
                                00002 R 707070 A *L
                                             SIZE=00003      NO ERROR LINES
BMACRO-15 V3A000
>↑C
```

```
DOS-15 V3A000
$LOAD

BLOADER V3A000
>P←SAMPLE
P SAMPLE SRC 77634
↑S↑S
```

REAL TIME CLOCK

References: Volume 1 Processor Handbook 6-42
System Reference Manual B-14

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 ($-60_{10}=-74_8$)

5 seconds, set location 000007 to 777324 ($-300_{10}=-454_8$).

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.

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          /60(10) 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
```

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
   or
   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.

```
                        .TITLE PROGRAM TO MOVE AC LIGHTS CENTER,OUT,BACK
                    /
                    /
                    /
                    /
                    /
10100 A                 .LOC 10100
10100 A 700002 A    START   IOF             /TURN PI OFF
10101 A 754000 A            CLA!CLL
10102 A 705504 A            ISA             /TURN API OFF (ISA WITH AC0=0)
                    /
10103 A 210143 A    SETUP   LAC (002000     /SET BIT 7 BECAUSE ROTATE RIGHT
10104 A 050141 A            DAC LH#         /BEFORE DISPLAYING
10105 A 210144 A            LAC (000200     /SET BIT 10 BECAUSE ROTATE LEFT
10106 A 050142 A            DAC RH#         /BEFORE DISPLAYING
                    /
                    /
                    /MANUFACTURE THE DISPLAYED ACCUMULATOR VALUE
                    /BY TAKING EACH HALF AND ROTATING IT IN THE APPROPRIATE
                    /DIRECTION.  THEN "XOR" THE HALVES FOR THE FULL VALUE.
                    /
10107 A 210141 A    FORM    LAC LH
10110 A 740020 A    MOVEL   RAR
10111 A 050141 A            DAC LH
10112 A 210142 A            LAC RH
10113 A 740010 A    MOVER   RAL
10114 A 050142 A            DAC RH
10115 A 250141 A            XOR LH
10116 A 050140 A            DAC DISPLY#
                    /
                    /SET UP THE REAL TIME CLOCK COUNTER--LOCATION 7
                    /
10117 A 777704 A    TIMER   LAW -74         /SET UP COUNTER FOR
10120 A 070145 A            DAC# (7         /ONE SECOND INTERVAL
                    /
                    /
                    /
                        .EJECT
```

```
                        /
                        /
                        /
                        /
                        /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 DISPLY
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 002000 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   IOF
00001 R 705514  A                   ISA+10
00002 R 700416  A                   TLS+10
                                /
00003 R 200043  R                   LAC (TABLE
00004 R 040042  R                   DAC PTR#
00005 R 777772  A                   LAW -6
00006 R 040041  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 600027  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
                                /
            000000  R               .END START
00043 R 000020  R *L
                SIZE=00044      NO ERROR LINES
```

```
                                .TITLE GLOBAL SUBROUTINE CALLS
                        /
            700401 A    TSF=700401
            700406 A    TLS=700406
                        /
                                .GLOBL CRLF,PRINT
                        /
                        /
00000 R 700002 A        START   IOF
00001 R 705514 A                ISA+10
00002 R 700416 A                TLS+10
                        /
00003 R 200032 R                LAC (TABLE
00004 R 040027 R                DAC PTR#
00005 R 777772 A                LAW -6
00006 R 040026 R                DAC COUNT#
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 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
                        /
            000000 R            .END START
00030 R 000030 E *E
00031 R 000031 E *E
00032 R 000020 R *L
            SIZE=00033      NO ERROR LINES
```

```
PAGE   1      CRLF   SRC      SUBROUTINE TO PRINT OUT CR AND LF

                                .TITLE SUBROUTINE TO PRINT OUT CR AND LF
                        /
                                .GLOBL CRLF,PRINT
                        /
00000 R 000000 A        CRLF    0
00001 R 760215 A                LAW 215
00002 R 120006 E                JMS* PRINT
00003 R 760212 A                LAW 212
00004 R 120006 E                JMS* PRINT
00005 R 620000 R                JMP* CRLF
                        /
        000000 A                .END
00006 R 000006 E *E
            SIZE=00007      NO ERROR LINES
```

```
PAGE   1      PRINT  SRC      SUBROUTINE TO PRINT CHAR ON TT

                                .TITLE SUBROUTINE TO PRINT CHAR ON TT
                        /
                                .GLOBL PRINT
                        /
        700401 A        TSF=700401
        700406 A        TLS=700406
                        /
                        /
                        /CHARACTER IS EXPECTED IN THE AC
                        /
00000 R 000000 A        PRINT   0
00001 R 700401 A                TSF
00002 R 600001 R                JMP .-1
00003 R 700406 A                TLS
00004 R 620000 R                JMP* PRINT
                        /
        000000 A                .END
            SIZE=00005      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
```

```
                                .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:  3,18,29,4,(CR)
                        /
                        /@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                        /
            700301 A    KSF=700301
            700312 A    KRB=700312
            700406 A    TLS=700406
                        /
                                .GLOBL CRLF,PRINT
                        /
    00000 R 707762 A    BEGIN   DBA                     /RUN IN PAGE MODE
    00001 R 700002 A            IOF                     /TURN OFF PI
    00002 R 705514 A            ISA+10                  /AND API INTERRUPT SYSTEMS
                        /
    00003 R 700416 A            TLS+10                  /INITIATE PRINT--GET FLAG
    00004 R 140117 R    INIT    DZM COUNT#              /COUNT= # OF VALUES
    00005 R 140120 R            DZM FINAL#              /FINAL= SUM OF VALUES
    00006 R 120125 E            JMS* CRLF               /ISSUE CR AND LF
    00007 R 140121 R    NXT     DZM NUMB#               /TEMPORARY LOCATION
                        /
    00010 R 700301 A    NEXT    KSF
    00011 R 600010 R            JMP .-1
    00012 R 700312 A            KRB
    00013 R 540127 R            SAD (215
    00014 R 600035 R            JMP ALLDUN              /CR MEANS LAST VALUE
    00015 R 540130 R            SAD (254                /"," SEPARATES VALUES
    00016 R 600030 R            JMP DUN
    00017 R 500131 R            AND (17                 /GET OCTAL NUMBER
    00020 R 040124 R            DAC TEMP#
                        /
    00021 R 200121 R            LAC NUMB
    00022 R 653122 A            MUL
    00023 R 000012 A            12
    00024 R 641002 A            LACQ                    /RESULT SMALL..IN MQ
    00025 R 340124 R            TAD TEMP                /ADD ON LAST DIGIT
    00026 R 040121 R            DAC NUMB                /SAVE IN NUMB
    00027 R 600010 R            JMP NEXT
                        /
    00030 R 200121 R    DUN     LAC NUMB                /GET THIS VALUE
    00031 R 340120 R            TAD FINAL               /ADD IT TO THE SUM
    00032 R 040120 R            DAC FINAL
                        /
    00033 R 440117 R            ISZ COUNT               /KEEP TRACK OF HOW MANY VALUES
    00034 R 600007 R            JMP NXT
```

```
00035 R 200117 R     ALLDUN  LAC COUNT          /GET # OF VALUES
00036 R 040041 R             DAC .+3           /STORE FOR DIVISION
00037 R 200120 R             LAC FINAL         /GET SUM
00040 R 653323 A             IDIV
00041 R 000000 A             0
00042 R 040123 R             DAC REMAIN#       /REMAINDER RETURNED IN AC
00043 R 641002 A             LACQ              /GET QUOTIENT IN MQ
00044 R 040122 R             DAC QUOT#
                     /
00045 R 735000 A             CLX
00046 R 200122 R             LAC QUOT
00047 R 653323 A     NEXXT   IDIV
00050 R 000012 A             12
00051 R 050112 R             DAC DIGIT,X       /STORE IN TABLE
00052 R 641002 A             LACQ              /PICK UP QUOTIENT
00053 R 741200 A             SNA               /CONTINUE IF NOT 0
00054 R 600057 R             JMP DUNN          /STOP WHEN QUOTIENT = 0
00055 R 737001 A             AXR 1             /PUT IN TABLE FOR LATER OUTPUT
00056 R 600047 R             JMP NEXXT
                     /
00057 R 120125 E     DUNN    JMS* CRLF         /ISSUE CR AND LF
                     /
00060 R 210112 R     DUNNN   LAC DIGIT,X
00061 R 340132 R             TAD (260          /TO MAKE ASCII
00062 R 120126 E             JMS* PRINT
00063 R 737777 A             AXR -1            /GET NEXT CHARACTER--WORK
                                               /WAY BACK UP TALBE
00064 R 724000 A             PXA               /ARE WE DONE--WHEN XR=0, YES
00065 R 740100 A             SMA
00066 R 600060 R             JMP DUNNN
                     /
00067 R 200133 R             LAC (256          /ISSUE DECIMAL POINT "."
00070 R 120126 E             JMS* PRINT
                     /
00071 R 200117 R             LAC COUNT
00072 R 040100 R             DAC REM
00073 R 200123 R             LAC REMAIN
00074 R 653122 A             MUL
00075 R 000012 A             12
00076 R 641002 A             LACQ
00077 R 653323 A             IDIV
00100 R 000000 A     REM     0
00101 R 641002 A             LACQ
00102 R 340132 R             TAD (260
00103 R 120126 E             JMS* PRINT
00104 R 120125 E             JMS* CRLF
00105 R 750004 A             LAS               /WANT TO CONTINUE OR EXIT?
00106 R 740200 A             SZA               /IF SWITCHES ARE 0, THEN EXIT
00107 R 600004 R             JMP INIT          /NON-ZERO--CONTINUE WITH ANOTHER
                             .EXIT
                     /
00112 R              A       DIGIT   .BLOCK 5
                     /
             000000 R                .END BEGIN
00125 R 000125 E *E
00126 R 000126 E *E
00127 R 000215 A *L
00130 R 000254 A *L
00131 R 000017 A *L
00132 R 000260 A *L
00133 R 000256 A *L
                SIZE=00134      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.

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:

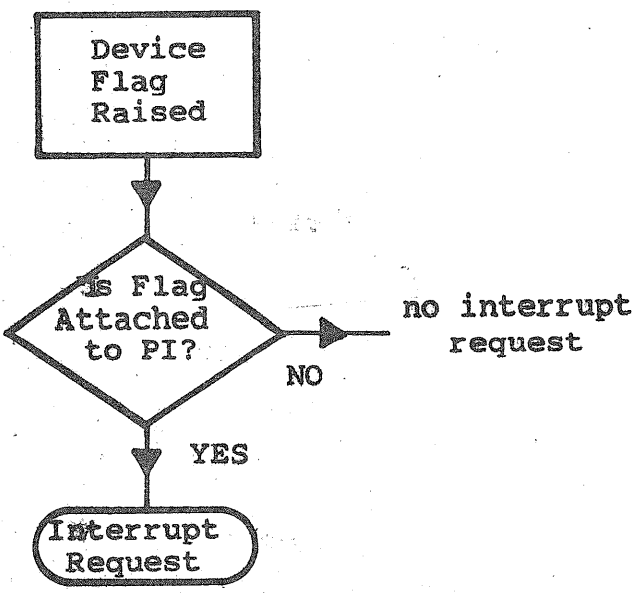
ION	700042	-Interrupt ON -Enable PI interrupts
IOF	700002	-Interrupt OFF -Disable PI interrupts
RES	707742	-Restore...i.e. set up for the restoration of the Link,Page/Bank mode, Memory Protect bit from the pointer word given in the next indirectly referenced instruction.

Vol. 9 proc 6-49

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.

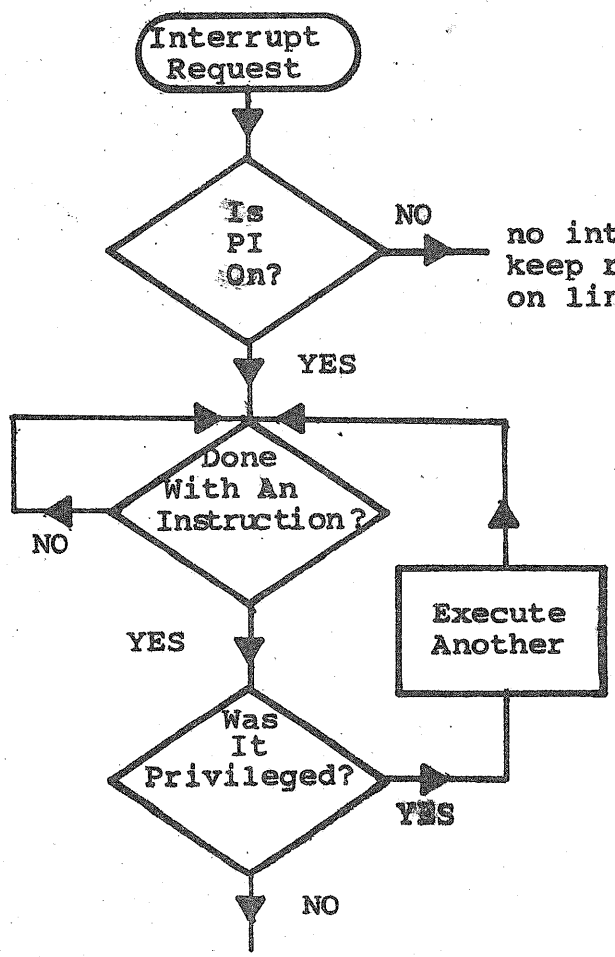
- - - - - - - - - - -

REQUESTING AN INTERRUPT



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

GRANTING AN INTERRUPT



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

PI interrupts will be granted only

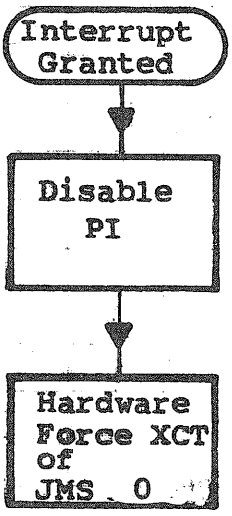
1) if PI is on
2) between instructions
3) after a non-privileged instructions (privileged instructions: IOTs, JMS, CAL,XCT,NORM)

*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

Bit 0 = Link at time of interrupt
1 = Page/Bank Mode Indicator
2 = Memory Protect Indicator

The word in location 000000 has the following format:

0	1	2	3 ... 17
Link	Page Bank	Memory Protect	15 bit address of the instruction that was to be executed at time of interrupt

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 straight forward and simple.

```
0          000000
1          JMP PTREAD


  PTREAD   DAC ACSAVE    /save registers used by the servie 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
```

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 occured-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
```

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

input
-store data transferred in and clear flag
-determine if more is to be input (if so, initiate it)

output
-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 permissable 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 Ø, bank Ø. 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 (rewiring needed) at the discretion of the user.

API ADDRESS

API Channel	Break Address	Standard Device	Suggested Priority Level
0	40	Software channel 0	4
1	41	Software channel 1	5
2	42	Software channel 2	6
3	43	Software channel 3	7
4	44	DECtape (TC15)	1
5	45	MagTape (TC59)	1
6	46		
7	47		
10	50	Paper Tape Reader (PC15)	2
11	51	Clock Overflow (KW15)	3
12	52	Power Fail (KF15)	0
13	53		0
14	54	Graphics (VT15/VP15)	2
15	55	Card Readers (CR15/CR03B)	2
16	56	Line Printer (LP15)	3
17	57	A/D (AD15/AF01)	0
20	60	DB99A/DB98A	3
21	61		
22	62	Data Phone (DP09A)	2
23	63	DECdisk (RF15)	1
24	64	Diskpack (RP15)	1
25	65	Plotter (XY15)	1
26	66		
27	67		
30	70	Scanners (DC01-ED) as needed use 70-77	3
31	71	UDC15	
32	72	ADC15	
33	73		
34	74	LT19 & LT15 Teleprinter	3
35	75	LT19 & LT15 Keyboard	3
36	76		
37	77		

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.

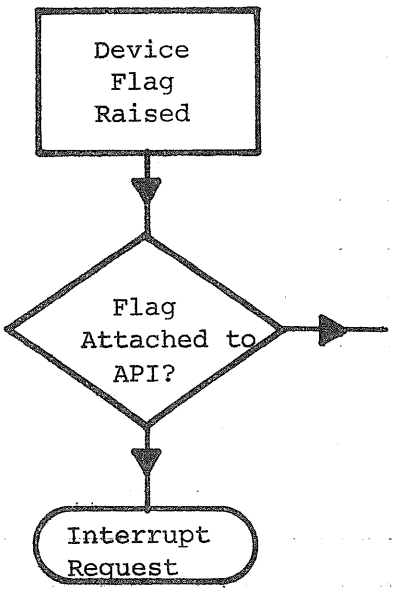
In addition to the above, there are two special features in the API facility. These are:

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.

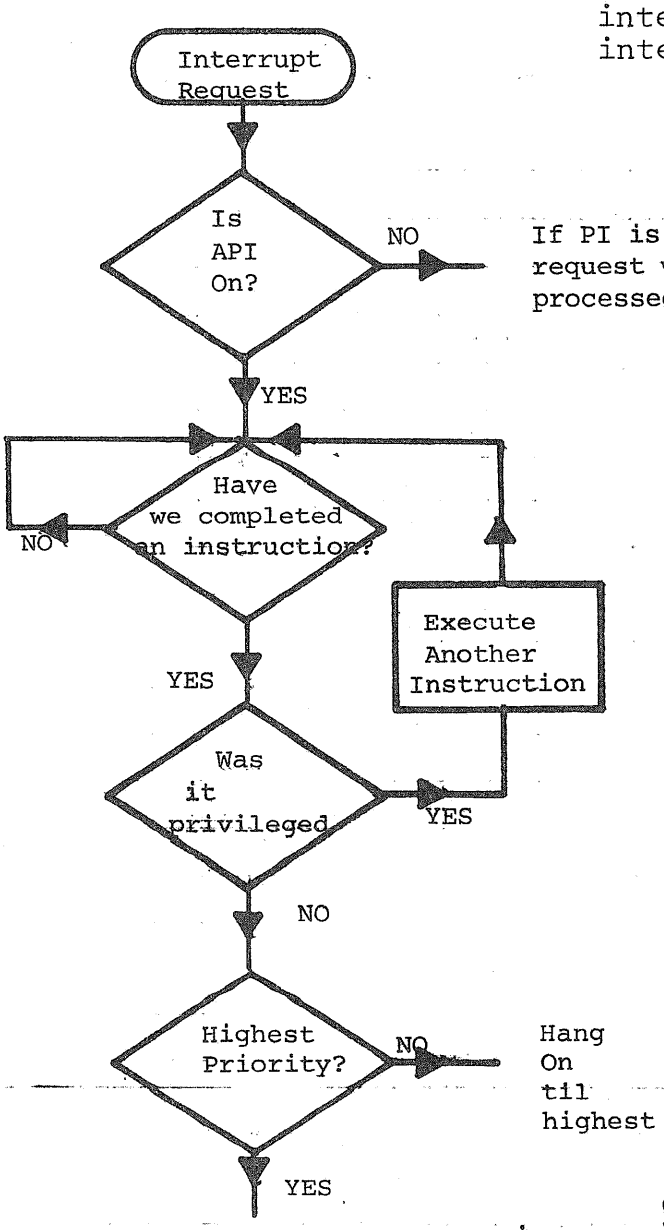
b. Program Interrupt - A program interrupt, from any I/O device connected to the computer, sets priority level 3. This occurs whether or not the API facility is enabled. This causes all devices on priority level 3, all software requests and program interrupts to be shut out until the level is released.

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 ir 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:

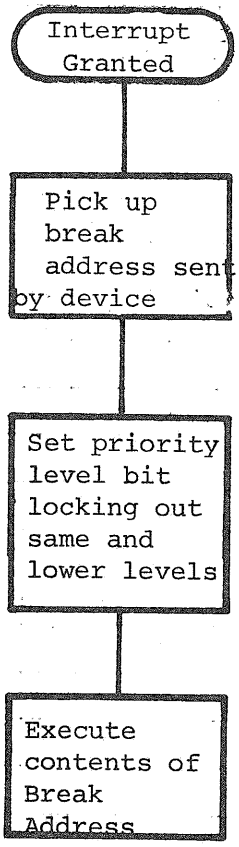
1) Data Channel Transfers
2) Clock breaks for updating location 000007.

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.

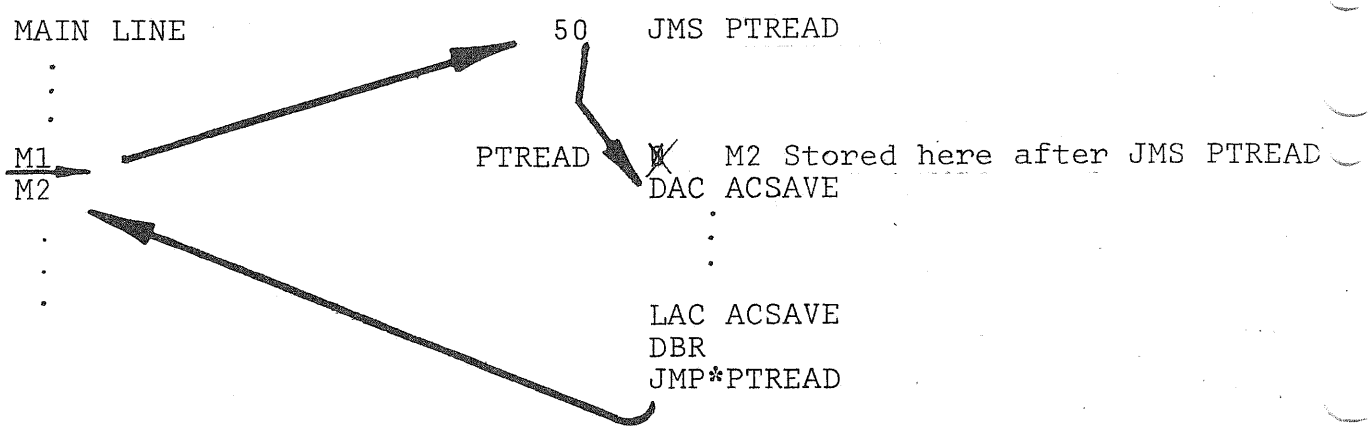
RES 707742 -Restore
-Set up to restore the link, page/bank mode and memory protect mode on next indirect instruction.

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 straight forward 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.

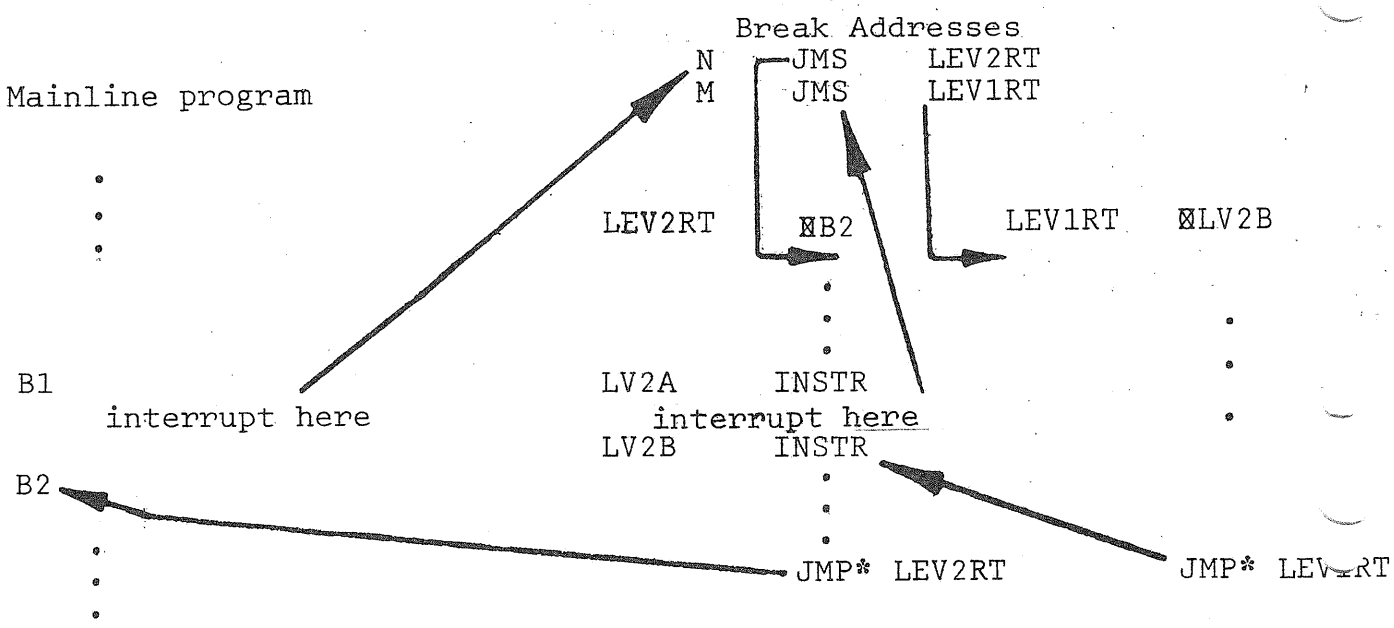
MULTIPLE DEVICE PROGRAMMING

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      Ø
            DAC     ACSAV1

            LAC     ACSAV1
            DBR
            JMP*    PTREAD
CLOCK       Ø
            DAC     ACSAV2

            LAC     ACSAV2
            DBR
            JMP*    CLOCK
PWRFL       Ø
            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.

PAGE 1 RULES EXA

```
                                    /EXAMPLE OF RELOCATION RULES FOR LINKING LOADER

                                            .GLOBL SUB
            000005 A        A=5
            000000 R        B=START
00000 R 200005 A        START   LAC A
00001 R 200010 A                LAC 10
00002 R 200000 R                LAC B
00003 R 200000 R                LAC START
00004 R 220010 E                LAC* SUB
00005 R 200011 A                LAC (100
00006 R 200012 A                LAC (START

00005 R                         .LOC START+5
00005 R 200010 A        START1  LAC 10
00006 R 200005 R                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 200002 A        START2  LAC START2
00003 A 000000 R                START

            000000 A            .END
00010 A 000010 E *E
00011 A 000100 A *L
00012 A 000000 R *L
            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


DOS-15 V3A000
$
```

```
                                .TITLE PROGRAM SHOWING RELOCATION ELEMENTS

000000 R 200013 R               LAC SYMBOL
000001 R 200017 R               LAC B#
000002 R 040014 R               DAC SYMBOL+1
000003 R 040017 R               DAC B
000004 R 220013 R               LAC* SYMBOL
000005 R 160011 A               DZM* 11
000006 R 220010 A               LAC* 10
000007 R 200010 A               LAC  10
000010 R 200021 R               LAC (END
000011 R 200022 R               LAC (END-SYMBOL
000012 R 040020 R               DAC C
000013 R 005000 A     SYMBOL    5000
000014 R 000000 A               0
000015 R 000004 A               4
000016 R 000005 A     END       5
         000000 A               .END
000021 R 000016 R *L
000022 R 000003 A *L
              SIZE=000023         1 ERROR LINES
```

PAGE LOAD

77614-77637

```
77610  000000  000000  000000  000000  207627  207633  047630  047633
77620  227627  160011  220010  200010  207635  207636  047634  005000
77630  000000  000004  000005  000000  000000  077632  000003  000762
```

BANK LOAD

77614-77637

```
77610  000000  000000  000000  000000  217627  217633  057630  057633
77620  237627  160011  220010  200010  217635  217636  057634  005000
77630  000000  000004  000005  000000  000000  077632  000003  000762
```

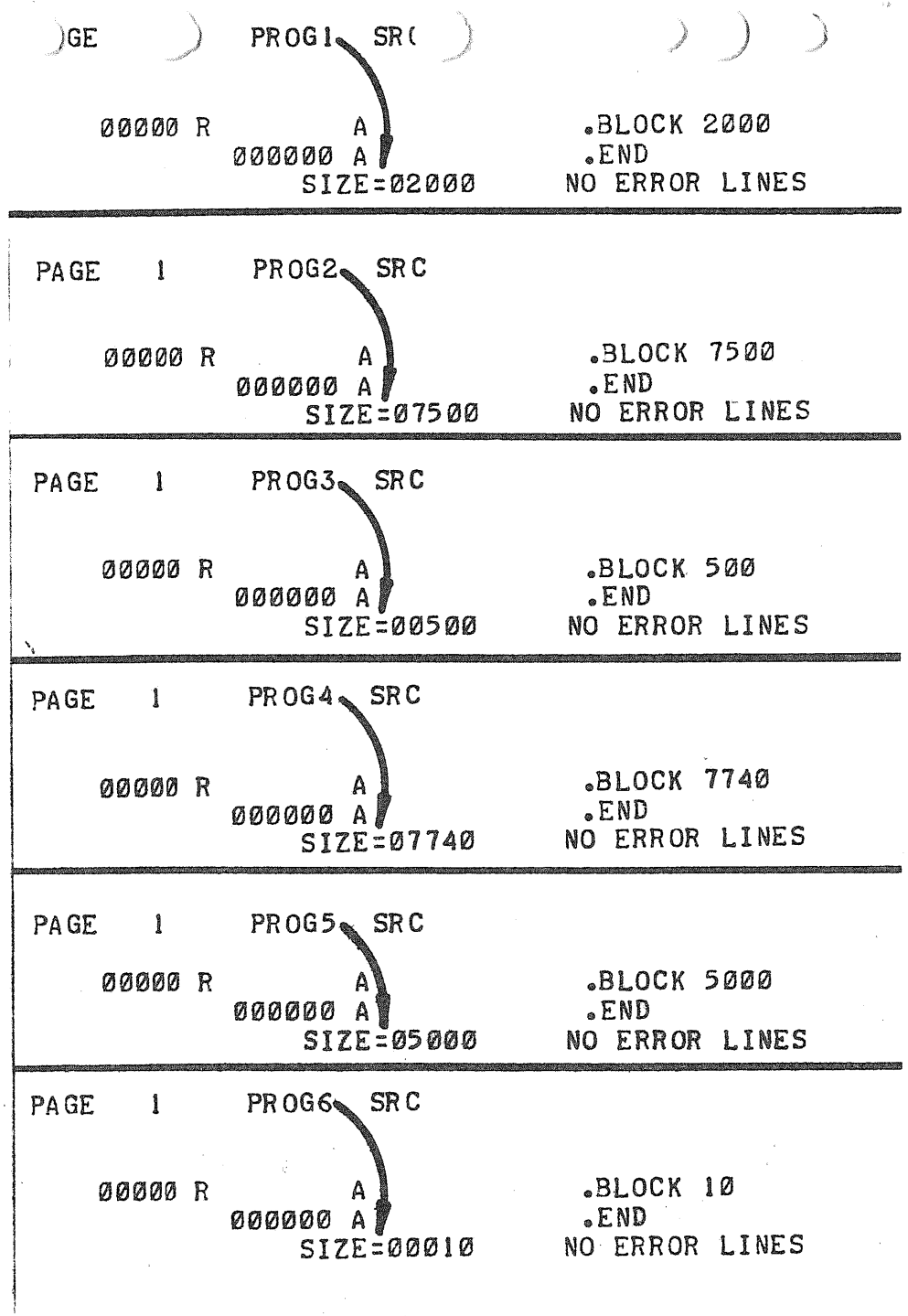
```
)OS-15 V3A000
)IP

DOSPIP V3A000

>L TT COPIA BIN ← DT1

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

>L TT 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

>↑C

)OS-15 V3A000
$A DT1 -14

$A TT -12

$DUMP

DUMP V3A000
 455#

455#

     0  015000  646531  262601  000004  000005  000070  071033  412450
    10  034150  074623  230204  400000  000000  000004  040404  000001
    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  005000  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
```

```
                                        .IODEV 4,5
                                    /
                                    /
                                    /
                                        .INIT 4,0,0
00000 R 000004 A *G                     CAL+0*1000 4&777
00001 R 000001 A *G                     1
00002 R 000000 A *G                     0+0
00003 R 000000 A *G                     0
                                    /
                                        .INIT 5,1,0
00004 R 001005 A *G                     CAL+1*1000 5&777
00005 R 000001 A *G                     1
00006 R 000000 A *G                     0+0
00007 R 000000 A *G                     0
                                    /
00010 R                         START   .READ 4,2,BUFF,34
00010 R 002004 A *G                     CAL+2*1000 4&777
00011 R 000010 A *G                     10
00012 R 000025 R *G                     BUFF
                    *G                  .DEC
00013 R 777736 A *G                     -34
                                    /
                                        .WAIT 4
00014 R 000004 A *G                     CAL 4&777
00015 R 000012 A *G                     12
                                    /
                                        .WRITE 5,2,BUFF,34
00016 R 002005 A *G                     CAL+2*1000 5&777
00017 R 000011 A *G                     11
00020 R 000025 R *G                     BUFF
                    *G                  .DEC
00021 R 777736 A *G                     -34
                                    /
                                        .WAIT 5
00022 R 000005 A *G                     CAL 5&777
00023 R 000012 A *G                     12
                                    /
00024 R 600010 R                        JMP START
                                    /
00025 R             A           BUFF    .BLOCK 42
                                    /
00067 R 000000 A                ENDMRK  0
                                    /
                                    /
            000010 R                    .END START
                SIZE=00070          NO ERROR LINES
```

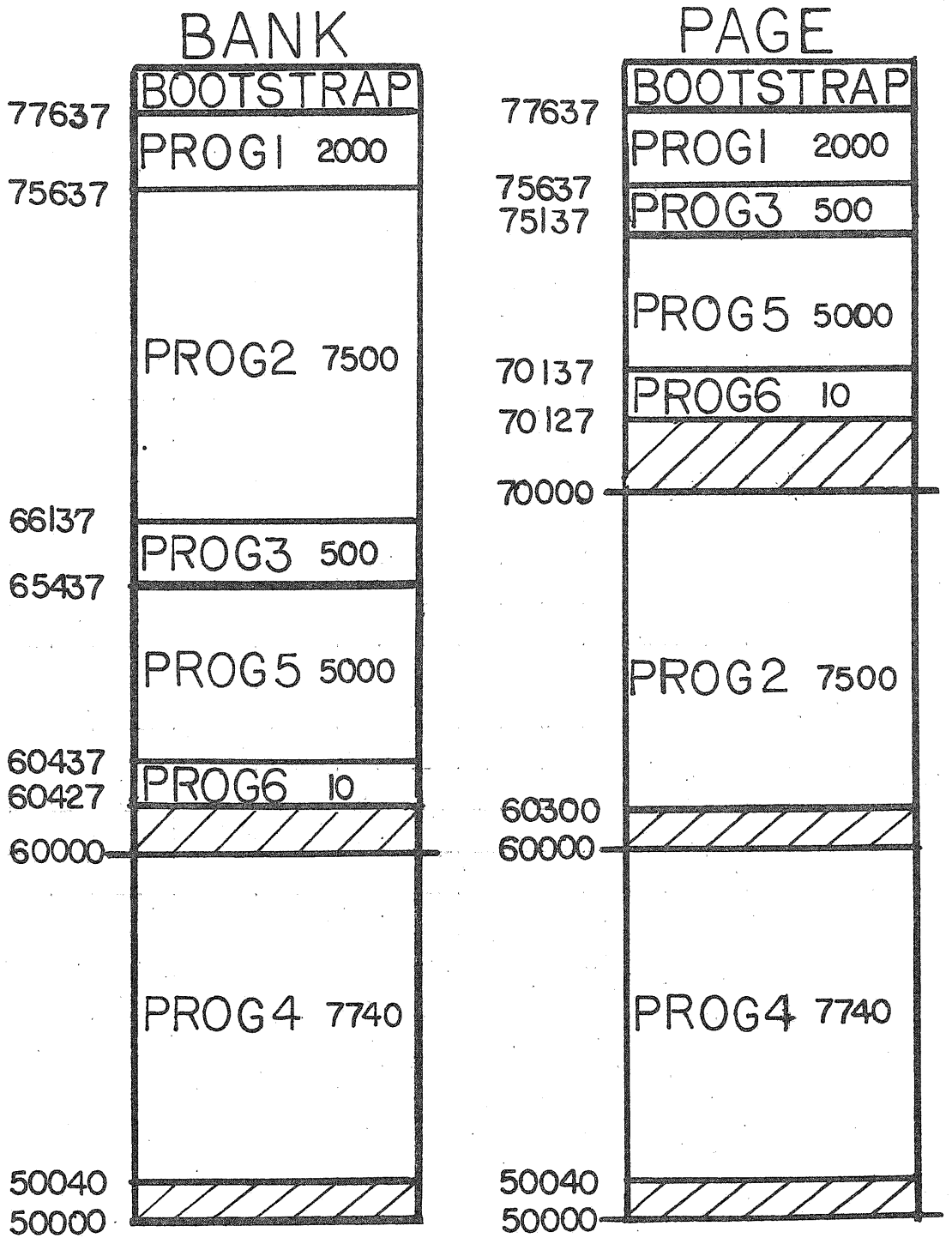
```
↑C

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  040404  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 000000
  370  000000  000000  000000  000000  000000  000000  777777  777777
```

RADIX 50_8 VALUES

X--		-X-		--X	
A	003100	A	000050	A	000001
B	006200	B	000120	B	000002
C	011300	C	000170	C	000003
D	014400	D	000240	D	000004
E	017500	E	000310	E	000005
F	022600	F	000360	F	000006
G	025700	G	000430	G	000007
H	031000	H	000500	H	000010
I	034100	I	000550	I	000011
J	037200	J	000620	J	000012
K	042300	K	000670	K	000013
L	045400	L	000740	L	000014
M	050500	M	001010	M	000015
N	053600	N	001060	N	000016
O	056700	O	001130	O	000017
P	062000	P	001200	P	000020
Q	065100	Q	001250	Q	000021
R	070200	R	001320	R	000022
S	073300	S	001370	S	000023
T	076400	T	001440	T	000024
U	101500	U	001510	U	000025
V	104600	V	001560	V	000026
W	107700	W	001630	W	000027
X	113000	X	001700	X	000030
Y	116100	Y	001750	Y	000031
Z	121200	Z	002020	Z	000032
%	124300	%	002070	%	000033
.	127400	.	002140	.	000034
0	132500	0	002210	0	000035
1	135600	1	002260	1	000036
2	140700	2	002330	2	000037
3	144000	3	002400	3	000040
4	147100	4	002450	4	000041
5	152200	5	002520	5	000042
6	155300	6	002570	6	000043
7	160400	7	002640	7	000044
8	163500	8	002710	8	000045
9	166600	9	002760	9	000046
#	171700	#	003030	#	000047


)GE PROG1 SRC
00000 R A .BLOCK 2000
000000 A .END
SIZE=02000 NO ERROR LINES
PAGE 1 PROG2 SRC
00000 R A .BLOCK 7500
000000 A .END
SIZE=07500 NO ERROR LINES
PAGE 1 PROG3 SRC
00000 R A .BLOCK 500
000000 A .END
SIZE=00500 NO ERROR LINES
PAGE 1 PROG4 SRC
00000 R A .BLOCK 7740
000000 A .END
SIZE=07740 NO ERROR LINES
PAGE 1 PROG5 SRC
00000 R A .BLOCK 5000
000000 A .END
SIZE=05000 NO ERROR LINES
PAGE 1 PROG6 SRC
00000 R A .BLOCK 10
000000 A .END
SIZE=00010 NO ERROR LINES

```
$LOAD

BLOADER V3A000
>P←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 60437
P PROG6   SRC 60427
↑S↑C

DOS-15 V3A000
$PAGE ON

$LOAD

LOADER V3A000
>P←PROG1,PROG2,PROG3,PROG4,PROG5,PROG6
P PROG1   SRC 75637
P PROG2   SRC 60300
P PROG3   SRC 75137
P PROG4   SRC 50040
P PROG5   SRC 70137
P PROG6   SRC 70127
↑S↑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.

FORTRAN	MACRO
C TEST MACRO SUBR	.TITLE MIN .GLOBL MIN, .DA
C READ A NUMBER(A)	MIN 0 / entry/exit JMS* .DA / general get argument
1 READ(1,100)A	/ (OTS)
100 FORMAT(E12.4)	JMP .+2+1 / jump around argument registers
C NEGATE THE NUMBER C AND PUT IT IN B	
CALL MIN(A,B)	MIN1 .DSA 0 / ARG1 MIN2 .DSA 0 / ARG2 LAC* MIN1 / first word of A DAC* MIN2 / store at B
C WRITE OUT NUMBER(B)	ISZ MIN1 / point to second word ISZ MIN2 / of A and B
WRITE(2,100)B	LAC* MIN1 / second word of A RAL / sign bit = 1 CML / RAR /
STOP	DAC* MIN2 / store in second word of B
END	JMP* MIN exit .END

The FORTRAN statement CALL MIN(A,B) is expanded by the compiler to:

```
 00013   JMS* MIN          / to MACRO subprog
 00014   JMP  $00014
 00015   .DSA A
 00016   .DSA B
$00014 = 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 Ø 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.

```
                C                       TITLE SUBR
                                        SUBROUTINE SUBR(A,B)
000000                                  CAL 0
000001                                  JMS* .DA
000002                                  JMP $000002
000003                                  .DSA A
000004                                  .DSA B
$000002 = 000005
```

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:

FORTRAN	MACRO
INTEGER A,B,C COMMON/NAME/C COMMON A,B . . .	.GLOBL NAME, .XX / .XX is name given to blank COMMON / by the FORTRAN Compiler DZM* .XX / CLEAR A - NOTE INDIRECT REFERENCE ISZ .XX / BUMP COUNTER DZM* .XX / CLEAR B DZM* NAME / CLEAR C

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
```

```
                                .TITLE MACRO ROUTINE CALLING FORTRAN PROGRAM
                        /
                                .GLOBL FORT
                        /
                        /000000000000000000000000000000000000000000000000000000000000
                        / PROGRAM REQUESTS USER TO INPUT THE # OF ELEMENTS TO
                        / BE SUMMED IN ARRAY A(1 TO 10 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.
                        /
                        /000000000000000000000000000000000000000000000000000000000000
                        /
          000000 A      IN=0
          000001 A      OUT=1
                        /
00000 R                 START   .INIT =2,IN,RST
                                .INIT =3,OUT,RST
                        /
00010 R                 RST     .WRITE =3,2,MES1,34
                                .WAIT =3
                        /
00016 R                 CHECK   .READ =2,3,BUFF,3       /READS 1 CHARACTER
                                .WAIT =2
                        /
00024 R 200146 R        CK1     LAC BUFF+2
00025 R 340151 R                TAD (-60
00026 R 040146 R                DAC BUFF+2
00027 R 741100 A        CK2     SPA                     /X>60, OR X=60
00030 R 600046 R                JMP ERRMES              /X<60 SO ENTRY NOT 0-9
00031 R 340152 R                TAD (-12                /X-60-12<0 MEANS X<72
00032 R 740100 A        CK3     SMA                     /X<72 MEANS ENTRY <OR= 9
00033 R 600046 R                JMP ERRMES              /X>OR=72 MEANS NOT 0-9
00034 R 200146 R        CK4     LAC BUFF+2
00035 R 040055 R                DAC COUNT
                        /
00036 R 120150 E                JMS* FORT
00037 R 600041 R                JMP .+2
00040 R 000055 R                .DSA COUNT
                        /
00041 R 750004 A                LAS
00042 R 740200 A                SZA
00043 R 600010 R                JMP RST
                        /
                                .EXIT
                        /
00046 R                 ERRMES  .WRITE =3,2,MESERR,34
                                .WAIT =3
00054 R 600016 R                JMP CHECK
                        /
00055 R 000000 A        COUNT   0
```

```
                                /
00056 R 021000 A    MES1        MESERR-MES1/2*1000
00057 R 000000 A                0
00060 R 522632 A                .ASCII /TYPE IN ONE LESS THAN THE NUMBER OF ELEM
00061 R 042500 A
00062 R 446344 A
00063 R 047634 A
00064 R 425011 A
00065 R 442646 A
00066 R 515612 A
00067 R 444202 A
00070 R 471012 A
00071 R 444212 A
00072 R 202352 A
00073 R 546604 A
00074 R 426444 A
00075 R 047614 A
00076 R 202131 A
00077 R 442632 A
00100 R 426352 A
00101 R 451500 A                .ASCII /YOU WOULD LIKE SUMMED (0-9)./
00102 R 546372 A
00103 R 520256 A
00104 R 476531 A
00105 R 442100 A
00106 R 462231 A
00107 R 342500 A
00110 R 516531 A
00111 R 546612 A
00112 R 421005 A
00113 R 030132 A
00114 R 345225 A
00115 R 000000 A
00116 R 064000 A                .ASCII <15>
00117 R 000000 A
00120 R 012000 A    MESERR      BUFF-MESERR/2*1000
00121 R 000000 A                0
00122 R 532031 A                .ASCII /VALUE IS OUT OF RANGE. /
00123 R 452612 A
00124 R 202232 A
00125 R 320236 A
00126 R 526504 A
00127 R 047614 A
00130 R 202450 A
00131 R 147216 A
00132 R 425344 A
00133 R 000000 A
00134 R 522632 A                .ASCII /TYPE '0-9' ONLY/<15>
00135 R 042500 A
00136 R 235405 A
00137 R 534516 A
00140 R 202371 A
```

```
00141 R 646262 A
00142 R 064000 A
00143 R 000000 A
                        /
00144 R 002000 A    BUFF    ENDLOC-BUFF/2*1000
00145 R 000000 A            0
00146 R         A           .BLOCK 2
                        /
          000150 R  ENDLOC=.
          000000 R            .END START
00150 R 000150 E *E
00151 R 777720 A *L
00152 R 777766 A *L
             SIZE=00153     NO ERROR LINES
```

```
001     C       FORTRAN CALLING MACRO EXAMPLE
002     C
003     C       THIS ROUTINE IS CALLED BY THE MACRO ROUTINE "MACCAL".
004     C       "MACCAL" PASSES ONE ARGUMENT, N , TO FORT
005     C       FORT ADDS 1 TO IT AND THEN...
006     C
007     C       THIS FORTRAN ROUTINE CALLS MACRO PROGRAM SUM
008     C       TO PERFORM SUMMATION OF ARRAY A.
009     C       THEN THE SUM IS PRINTED OUT AFTER TOT=.
010     C
011             SUBROUTINE FORT(N)
012     C
013             INTEGER A(10),TOT
014             A(1)=5
015             A(2)=6
016             A(3)=7
017             A(4)=8
018             A(5)=9
019             A(6)=10
020             A(7)=11
021             A(8)=12
022             A(9)=13
023             A(10)=14
024     C
025             N=N+1
026             CALL SUM(A,N,TOT)
027             WRITE(6,1) TOT
028     C
029        1    FORMAT(1H ,'TOT=',I10)
030             RETURN
031             END
```

```
0001      C         FORTRAN CALLING MACRO EXAMPLE
0002      C
0003      C         THIS ROUTINE IS CALLED BY THE MACRO ROUTINE "MACCAL".
0004      C         "MACCAL" PASSES ONE ARGUMENT, N , TO FORT
0005      C         FORT ADDS 1 TO IT AND THEN...
0006      C
0007      C         THIS FORTRAN ROUTINE CALLS MACRO PROGRAM SUM
0008      C         TO PERFORM SUMMATION OF ARRAY A.
0009      C         THEN THE SUM IS PRINTED OUT AFTER TOT=.
0010      C
0011                SUBROUTINE FORT(N)
 00000    .DSA FORT
 00001    JMS* .DA
 00002    JMP  $00002
 00003    .DSA N
 $00002   = 00004
0012      C
0013                INTEGER A(10),TOT
0014                A(1)=5
 00004    CMA!CLA
 00005    TAD  (000001
 00006    TAD  A
 00007    DAC  $00007
 00010    LAC  (000005
 $00007   = 00153
 00011    DAC* %IA
0015                A(2)=6
 00012    CMA!CLA
 00013    TAD  (000002
 00014    TAD  A
 00015    DAC  $00015
 00016    LAC  (000006
 $00015   = 00153
 00017    DAC* %IA
0016                A(3)=7
 00020    CMA!CLA
 00021    TAD  (000003
 00022    TAD  A
 00023    DAC  $00023
 00024    LAC  (000007
 $00023   = 00153
 00025    DAC* %IA
0017                A(4)=8
 00026    CMA!CLA
 00027    TAD  (000004
 00030    TAD  A
 00031    DAC  $00031
 00032    LAC  (000010
 $00031   = 00153
 00033    DAC* %IA
0018                A(5)=9
 00034    CMA!CLA
 00035    TAD  (000005
 00036    TAD  A
 00037    DAC  $00037
 00040    LAC  (000011
 $00037   = 00153
 00041    DAC* %IA
0019                A(6)=10
 00042    CMA!CLA
```

```
 00043   TAD   (000006
 00044   TAD   A
 00045   DAC   $00045
 00046   LAC   (000012
 $00045 = 00153
 00047   DAC*  %IA
020                 A(7)=11
 00050   CMA!CLA
 00051   TAD   (000007
 00052   TAD   A
 00053   DAC   $00053
 00054   LAC   (000013
 $00053 = 00153
 00055   DAC*  %IA
021                 A(8)=12
 00056   CMA!CLA
 00057   TAD   (000010
 00060   TAD   A
 00061   DAC   $00061
 00062   LAC   (000014
 $00061 = 00153
 00063   DAC*  %IA
022                 A(9)=13
 00064   CMA!CLA
 00065   TAD   (000011
 00066   TAD   A
 00067   DAC   $00067
 00070   LAC   (000015
 $00067 = 00153
 00071   DAC*  %IA
023                 A(10)=14
 00072   CMA!CLA
 00073   TAD   (000012
 00074   TAD   A
 00075   DAC   $00075
024      C
 00076   LAC   (000016
 $00075 = 00153
 00077   DAC*  %IA
025                 N=N+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
 00106   .DSA  400000 +N
 00107   .DSA  TOT
027                 WRITE(6,1) TOT
 00110   JMS*  .FW
 00111   .DSA  (000006
 00112   .DSA  .1
028      C
 00113   .DSA  777777
 00114   JMS*  .FE
 00115   .DSA  TOT
 00116   JMS*  .FF
029        1    FORMAT(1H ,'TOT=',I10)
 00117   JMP   $00117
```

```
 00120   .DSA 241431
 00121   .DSA 020130
 00122   .DSA 236511
 00123   .DSA 752172
 00124   .DSA 235311
 00125   .DSA 130540
 00126   .DSA 245004
 00127   .DSA 020100
 $00117  = 00130
@30                    RETURN
 00130   JMP   .EX
@31                      END
 00131   JMP* 00000
 00132   .DSA .DA
 00133   .BLK 000012
 00145   .DSA 000000
 00146   .DSA 000012
 00147   .DSA 000000
 00150   .DSA 000000
 00151   .DSA A
 00152   .BLK 000001
 00153   .BLK 000001
 00154   .DSA SUM
 00155   .DSA .FW
 00156   .DSA .FE
 00157   .DSA .FF
 00160   .DSA 000001
 00161   .DSA 000005
 00162   .DSA 000002
 00163   .DSA 000006
 00164   .DSA 000003
 00165   .DSA 000007
 00166   .DSA 000004
 00167   .DSA 000010
 00170   .DSA 000011
 00171   .DSA 000012
 00172   .DSA 000013
 00173   .DSA 000014
 00174   .DSA 000015
 00175   .DSA 000016
   FORT    17777
 * .DA      00132
   N        00003
   .EX      00131
   A        00133
   TOT      00152
   %IA      00153
 * SUM      00154
   .1       00117
 * .FW      00155
 * .FE      00156
 * .FF      00157
```

```
                                .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
                        /
00000 R 000000 A        SUM     0
00001 R 120020 E                JMS* .DA
00002 R 600006 R                JMP .+4
00003 R 000000 A        ARG1    0       /ADDRESS OF ARRAY A
00004 R 000000 A        ARG2    0       /ADDRESS OF LOCATION CONTAINING VALUE N
00005 R 000000 A        ARG3    0       /ADDRESS OF LOCATION WHERE SUM IS RETUR
                        /
00006 R 220004 R                LAC* ARG2
00007 R 740031 A                TCA
00010 R 040004 R                DAC ARG2
                        /
00011 R 750000 A                CLA
00012 R 360003 R        ADDER   TAD* ARG1
00013 R 440003 R                ISZ ARG1
00014 R 440004 R                ISZ ARG2
00015 R 600012 R                JMP ADDER
00016 R 060005 R                DAC* ARG3
00017 R 620000 R                JMP* SUM
                        /
      000000 A                  .END
00020 R 000020 E *E
            SIZE=00021      NO ERROR LINES
```

UPDATE

```
DOS-15 V3A000                                  **********************************
$R UPDATE
                                                CREATING A LIBRARY WITH UPDATE
.DAT    DEVICE  UIC     USE                    **********************************
-15     DKA     PES     OUTPUT
-14     DKA     PES     INPUT
-12     TTA     PES     LISTING
-10     TTA     PES     SECONDARY INPUT

$A LP -12/DK -10

$K ON

$UPDATE

UPDATE V3A000
>NL←USERLB
>I CRLF                  } USERLB CONTAINS ONE BINARY PROGRAM
>CLOSE

UPDATE V3A000
>NL←.LIBR5
>I CRLF
>I PRINT                 } .LIBR5 CONTAINS TWO BINARY PROGRAMS
>CLOSE

UPDATE V3A000
>NL←MAIN
>I AVGLOB
>I CRLF                  } MAIN CONTAINS THREE BINARY PROGRAMS
>I PRINT
>CLOSE

UPDATE V3A000
>↑C

DOS-15 V3A000
$
```

```
↑C

DOS-15 V3A000                                  **************************
$PAGE ON
                                                MAKING USE OF LIBRARIES
$R LOAD
                                                WITH THE LINKING LOADER
.DAT    DEVICE  UIC     USE                    **************************
-5      NON     PES     USER LIBR
-4      DKA     PES     USER PROG(S)
-1      DKA     SYS     SYS LIBR

$A DK -5

$K ON

$LOAD

LOADER V3A000
>P←AVGLOB,CRLF,PRINT ➡ ALL THREE FILES SPECIFIED.
P AVGLOB SRC 77503      NONE PULLED FROM A LIBRARY.
P CRLF   SRC 77474
P PRINT  SRC 77467
↑S↑S
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      THE OTHER TWO PROGRAMS--CRLF AND PRINT--
P CRLF   SRC 77474      ARE PULLED FROM THE USER LIBRARY .LIBR5
P PRINT  SRC 77467      AUTOMATICALLY.
↑S↑S
3,4,5,
4.0

DOS-15 V3A000
$LOAD

LOADER V3A000
>P←MAIN              ➡ THE FILE SPECIFIED IS A LIBRARY.
P AVGLOB SRC 77503      EVERY PROGRAM IN THE LIBRARY IS LOADED.
P CRLF   SRC 77474
P PRINT  SRC 77467
↑S↑S
3,4,5,
4.0

DOS-15 V3A000
$LOAD

LOADER V3A000
>P←AVGLOB,PRINT      ➡ TWO FILES (PROGRAMS) ARE SPECIFIED.
P AVGLOB SRC 77503      THE THIRD PROGRAM --CRLF--
P PRINT  SRC 77476      IS AUTOMATICALLY PULLED FROM THE
P CRLF   SRC 77467      USER LIBRARY .LIBR5.
↑S↑S
3,4,5,
```

```
DOS-15 V3A000
$LOAD

LOADER V3A000
>P←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--.
↑S↑S                       THE FILE PRINT CONTAINS ONE PROGRAM (PRINT).
3,4,5,
4.0

DOS-15 V3A000
$
```

BATCH
(NON—BOSS)

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

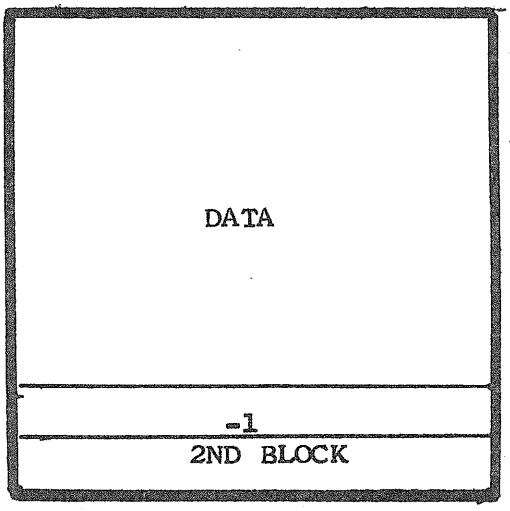
```
DOS-15 V3A000
$L
      **********************************************************************

      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

DOSPIP V3A000

>T PP ← DK BATSTR SRC
```

```
DOS-15 V3A000
$L
      **********************************************************************

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

      **********************************************************************
```

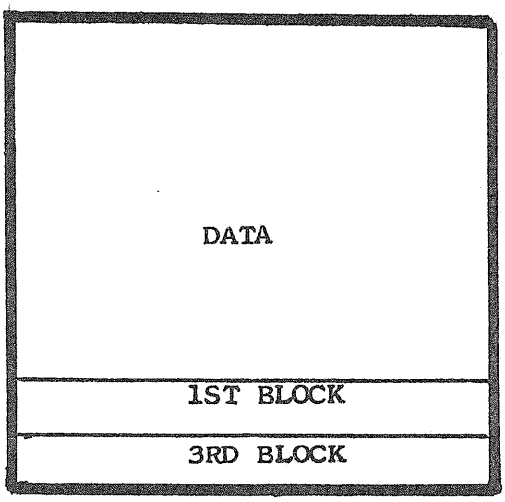
$BATCH PR

```
DOS-15 V3A000
$$JOB
PIP
DOSPIP V3A000

>L LP ← DK <SCR>

>L LP FILBLK BIN ← DK (P)

>$JOB
DOS-15 V3A000
$A LP -12

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

DOS-15 V3A000
$$EXIT

DOS-15 V3A000
$
```

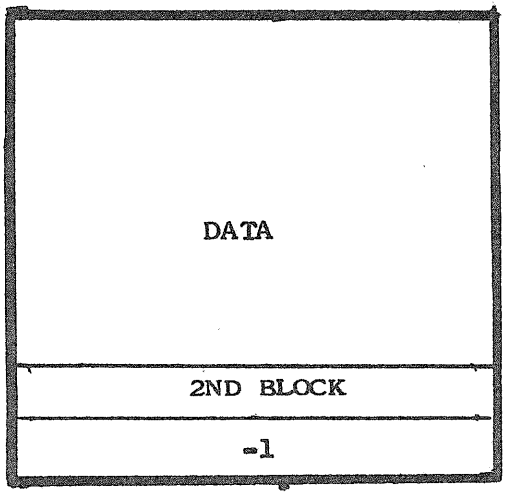
DISK FILE FORMAT FOR DOS:



1ST BLK



2ND BLK

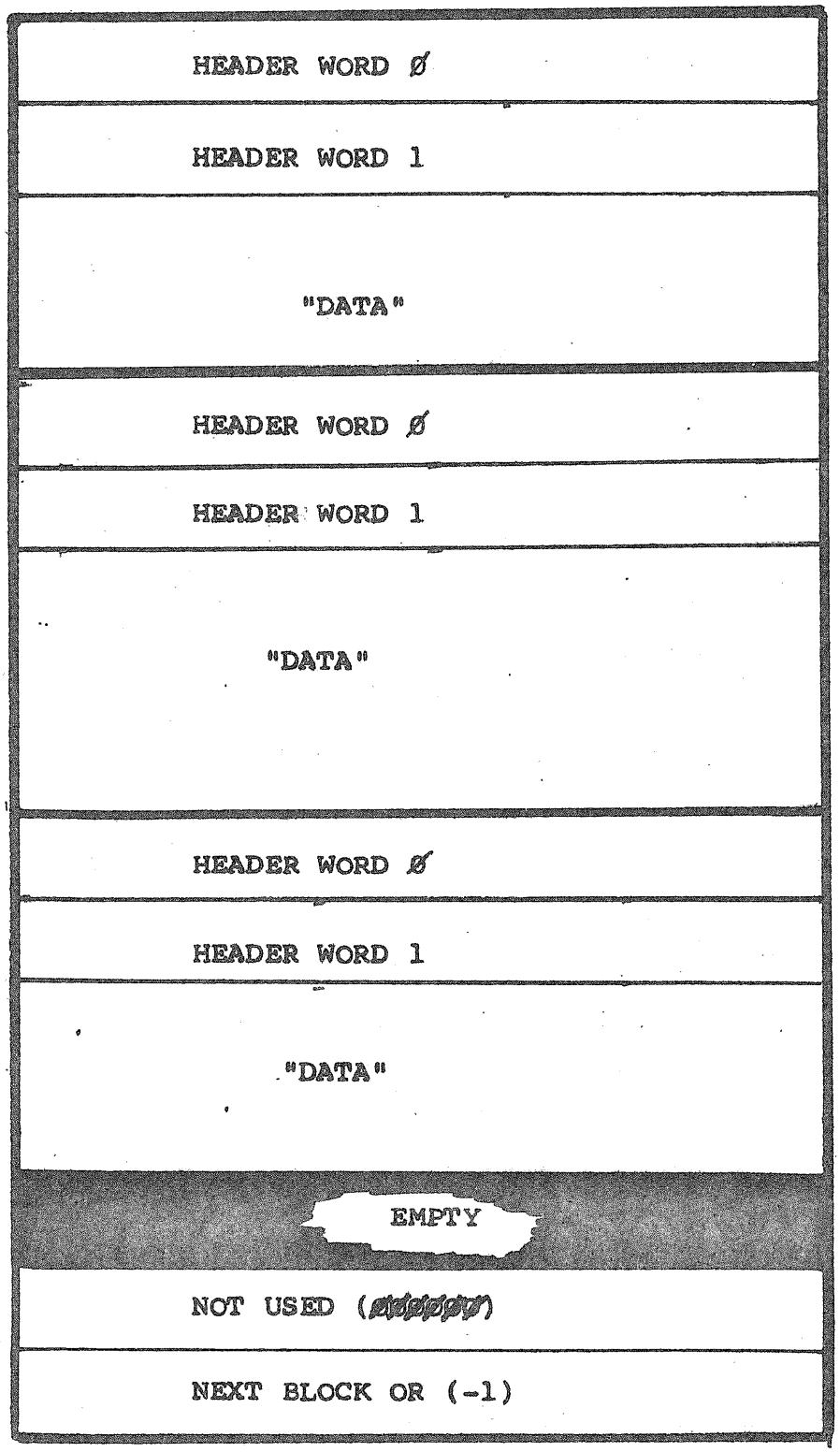


3RD BLK

```
                                .TITLE PROGRAM TO LIST ON TT BLOCKS IN A FILE ON
                        /
                        /
                        /00000000000000000000000000000000000000000000000000000
                        /
                        /PROGRAM LISTS ON THE TELETYPE 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.
                        /
                        /00000000000000000000000000000000000000000000000000000
                        /
                        /
                                .IODEV 5
                        /
00000 R 707762 A        START   DBA                     /MAKE SURE IN PAGE MODE
                                .INIT -3,1,0            /INITIALIZE DEVICES
                                .INIT 5,0,0
                        /
                        /*****************************************************
                        /GET BLOCK NUMBER AND FILL IT IN APPROPRIATE WORD OF
                        /.TRAN EXPANSION.
                        /*****************************************************
                        /
00011 R 740040 A                HLT                     /READ IN BLOCK NUMBER
00012 R 750004 A                LAS                     /FROM DATA SWITCHES
00013 R 040017 R                DAC TRAN+2              /PUT IN .TRAN EXPANSION
                                                        /BLOCK # POSITION
                        /
00014 R 100033 R                JMS PRINT               /PRINT OUT BLOCK #
                        /
                        /
                        /
00015 R                 TRAN    .TRAN 5,0,0,BUFF,256    /DO .TRAN
                                .WAIT 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 TH
                        /BLOCK USED BY THIS FILE.
                        /*****************************************************
                        /
00024 R 200472 R                LAC BUFF+377            /PICK UP WORD 377 IN BLK
00025 R 540473 R                SAD (777777             /LAST FILE BLK(777777)?
00026 R 600055 R                JMP ENDFIL              /YES
00027 R 100033 R                JMS PRINT               /NO-PRINT OUT BLOCK #
```

```
                        /*************************************************************
                        /GET 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
                        /*************************************************************
                        /
00030 R 200472 R                LAC BUFF+377            /SET UP TO TRAN IN NEXT
00031 R 040017 R                DAC TRAN+2
00032 R 600015 R                JMP TRAN                /DO NEXT TRAN
                        /
                        /*************************************************************
                        /CONVERT 6 OCTAL DIGITS TO 6 ASCII CHARACTERS
                        /*************************************************************
                        /
00033 R 000000 A        PRINT   0
00034 R 652000 A                LMQ
00035 R 735000 A                CLX
00036 R 200474 R                LAC (6
00037 R 722000 A                PAL
00040 R 750000 A        NXT     CLA
00041 R 640603 A                LLS 3
00042 R 340475 R                TAD (60
00043 R 050063 R                DAC BUFFO+2,X
00044 R 725001 A                AXS 1
00045 R 600040 R                JMP NXT
                        /
                        /*************************************************************
                        /OUTPUT BLOCK #
                        /*************************************************************
                        /
                                .WRITE -3,3,BUFFO,0
                                .WAIT -3
00054 R 020033 R                JMP* PRINT
                        /
                        /*************************************************************
                        /ON END OF FILE, CLOSE DAT -3 AND RETURN TO MONITOR
                        /*************************************************************
                        /
00055 R                 ENDFIL  .CLOSE -3
                                .EXIT
                        /
00061 R 005003 A        BUFFO   005003
00062 R 000000 A                0
00063 R         A               .BLOCK 6
00071 R 000015 A                15
00072 R 000012 A                12
                        /
00073 R         A       BUFF    .BLOCK 400
                        /
        000000 R                .END START
00473 R 777777 A *L
00474 R 000006 A *L
00475 R 000060 A *L
             SIZE=00476      NO ERROR LINES
```

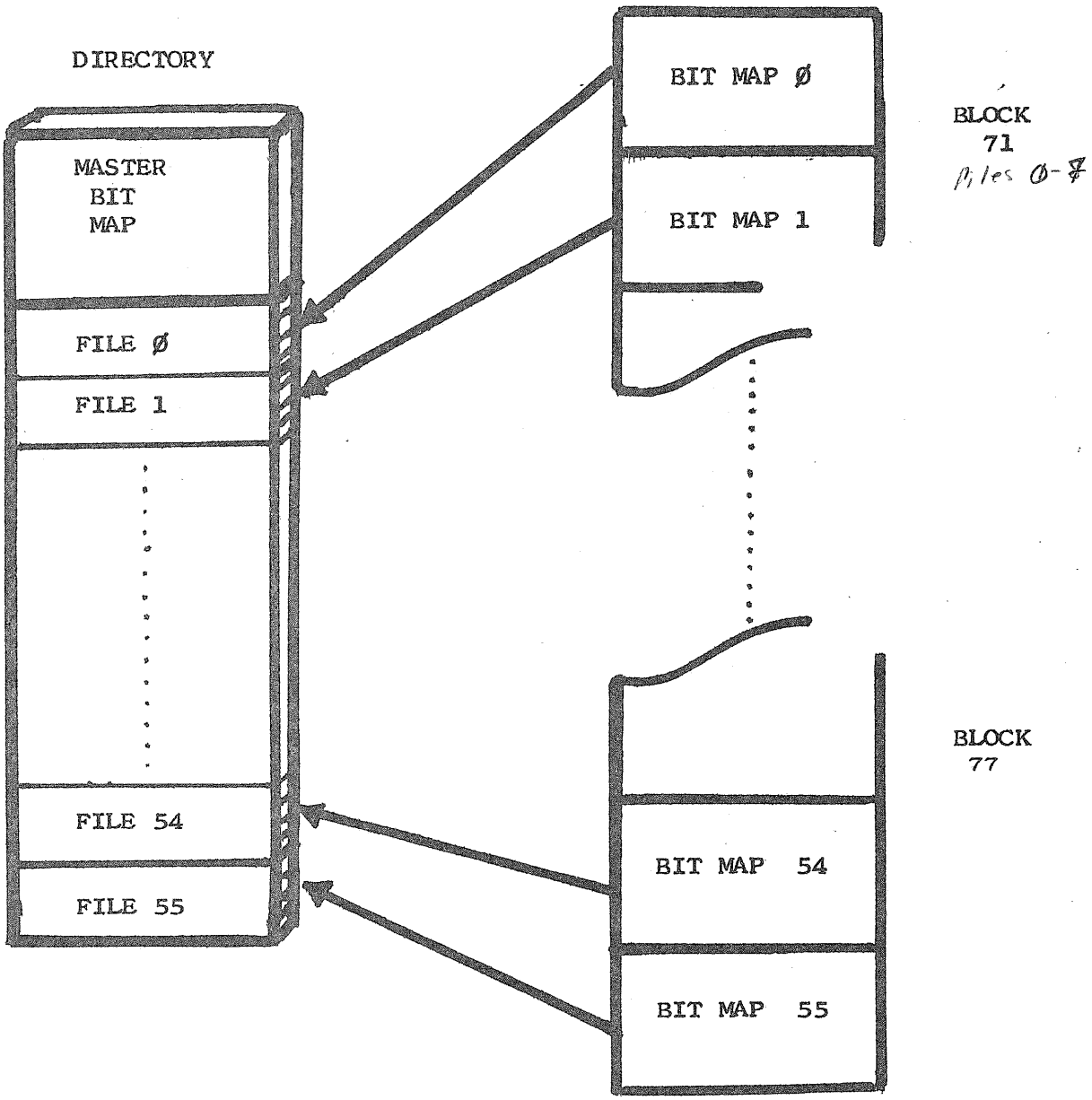
EXAMPLE OF ONE BLOCK OF A FILE:



BIT MAPS

BLOCKS 71 THROUGH 77 MAINTAIN INDIVIDUAL BIT MAPS FOR ALL FILES ON DECTAPE.

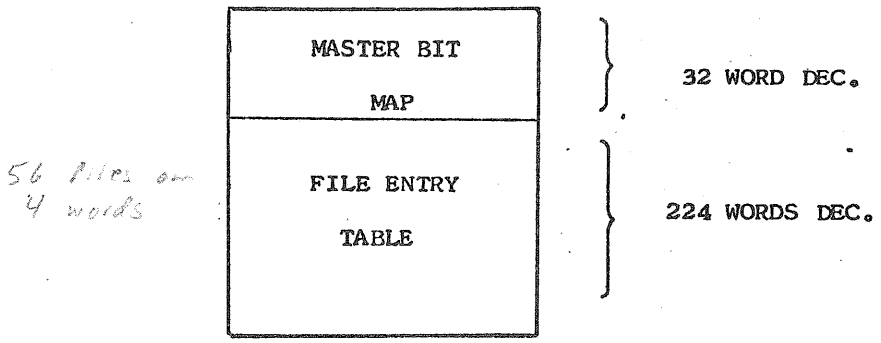
BIT MAP FILE RELATION:



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

DECTAPE FILE STRUCTURE

DIRECTORY (BLK #100):



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	Ø	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
WORD Ø	Ø	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	.	.	.	ETC.												

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

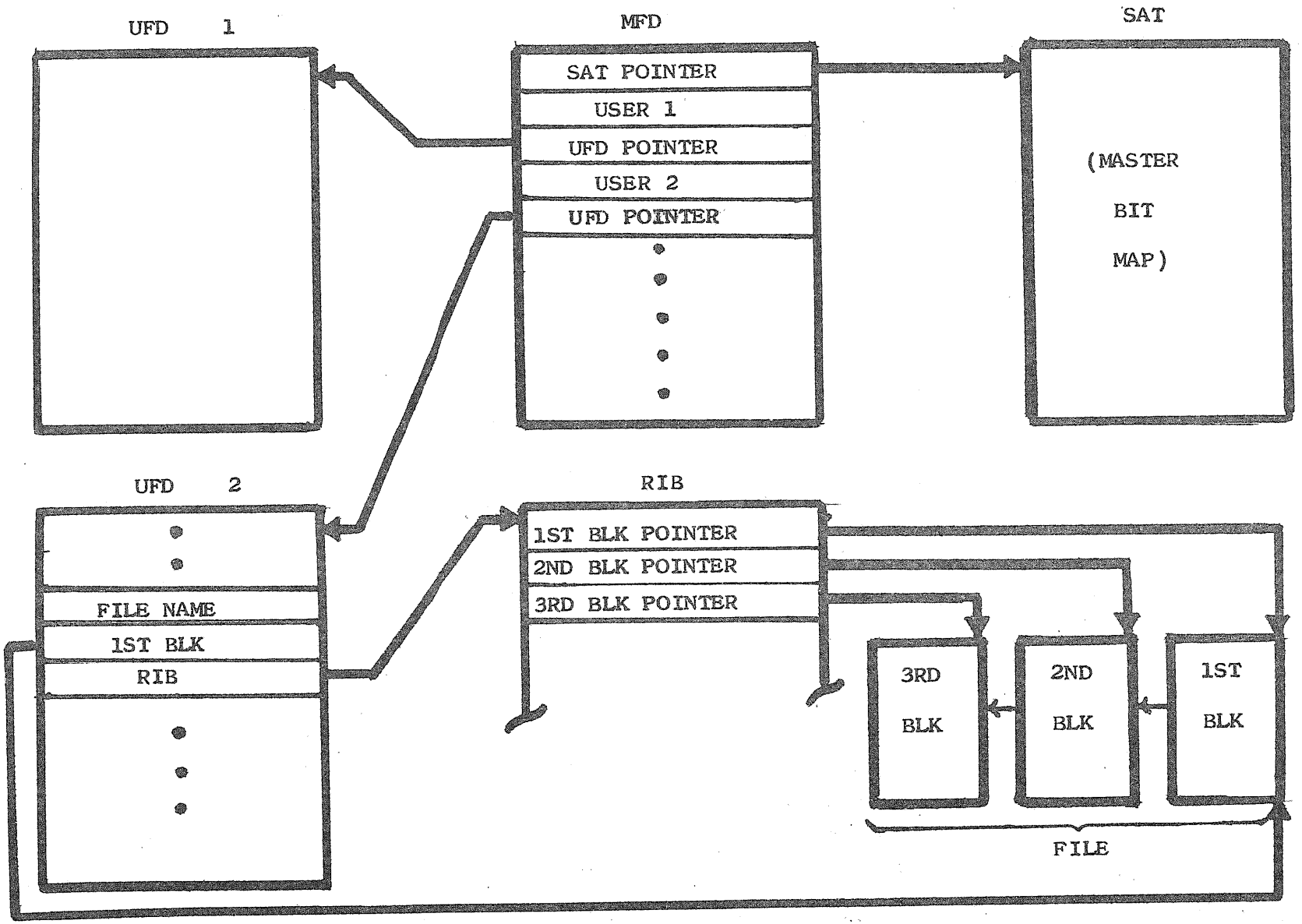
E. G.

WORD X	T	E	S
WORD X+1	T	@	@
WORD X+2	S	R	C
WORD X+3	400005		

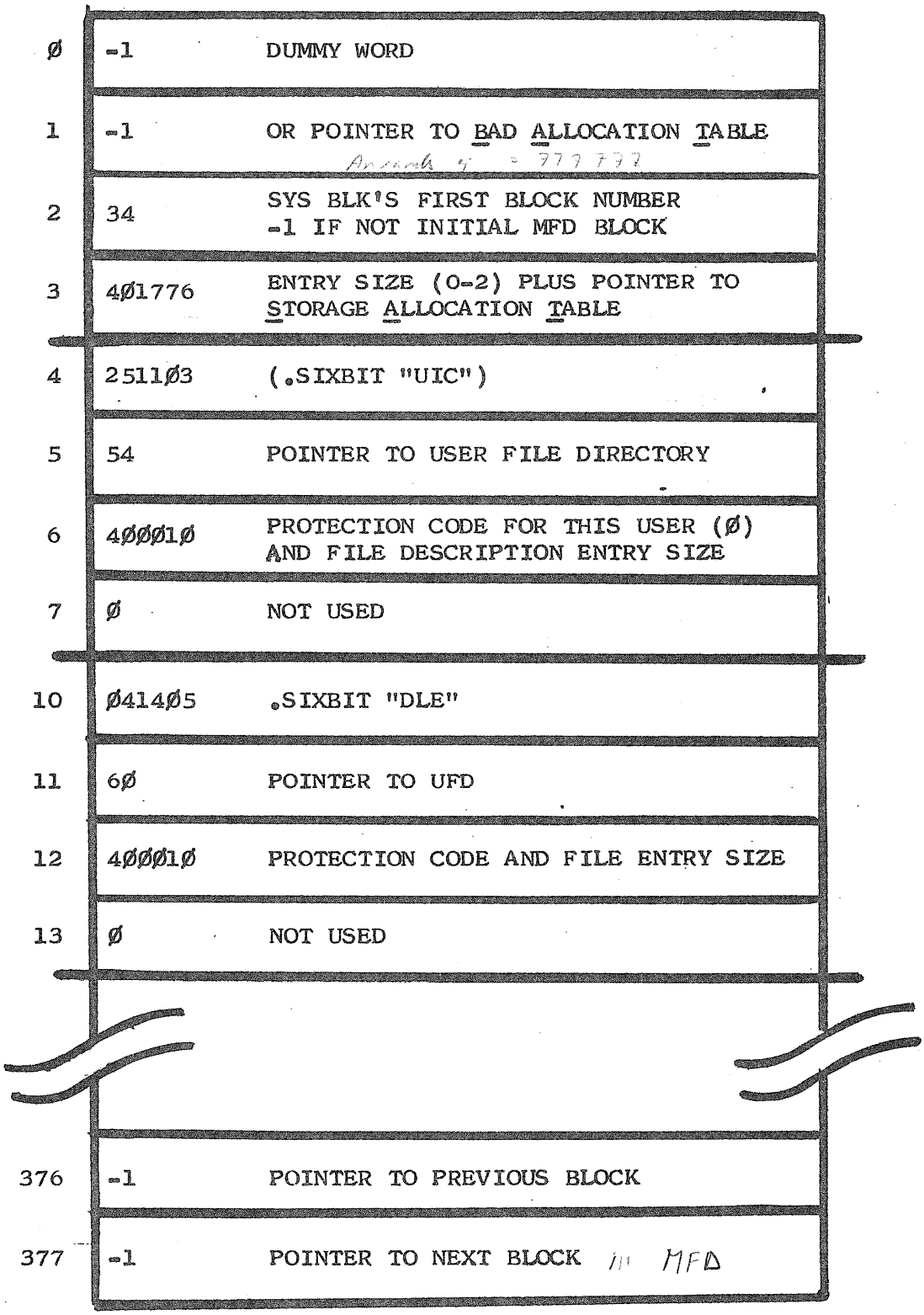
"6" BIT FILE NAME

1ST BLOCK OF THAT FILE, MSB = ACTIVE

DOS FILE STRUCTURE



Master File Directory (MFD)



Ø	-1	DUMMY WORD
1	-1	OR POINTER TO BAD ALLOCATION TABLE Records if = 777777
2	34	SYS BLK'S FIRST BLOCK NUMBER -1 IF NOT INITIAL MFD BLOCK
3	4Ø1776	ENTRY SIZE (0-2) PLUS POINTER TO STORAGE ALLOCATION TABLE
4	2511Ø3	(.SIXBIT "UIC")
5	54	POINTER TO USER FILE DIRECTORY
6	4ØØØ1Ø	PROTECTION CODE FOR THIS USER (Ø) AND FILE DESCRIPTION ENTRY SIZE
7	Ø	NOT USED
10	Ø414Ø5	.SIXBIT "DLE"
11	6Ø	POINTER TO UFD
12	4ØØØ1Ø	PROTECTION CODE AND FILE ENTRY SIZE
13	Ø	NOT USED
376	-1	POINTER TO PREVIOUS BLOCK
377	-1	POINTER TO NEXT BLOCK in MFD

NOTES: MFD = BLOCK #1777 if RF DISK, 47Ø4Ø if RPO2

PROTECTION CODE: 1 = Protected Directory, Ø = Unprotected

ILLEGAL UFD'S: @@@, ???, and those that are current to the system - PAG, BNK, SYS, IOS.

User File Directory (UFD)

	Ø3Ø114	.SIXBT "CALH@@SRC"
	1ØØØØØ	.
	2322Ø3	.
	ØØ1645	FIRST BLOCK OF THIS FILE (BIT Ø = TRUNCATION)
	ØØØØØ2	SIZE OF FILE IN BLOCKS
	ØØ1764	POINTER TO RIB BLOCK
*	2ØØ232	FILE PROTECTION CODE (Ø-2), START LOCATION OF RIB
	1425Ø1	DATE FILE CREATED (12-21-71)
	17Ø623	.SIXBT "OFSCLKLST"
	Ø31413	.
	142324	.
	ØØ2045	FIRST BLOCK OF THIS FILE
	ØØØ116	SIZE OF FILE IN BLOCKS
	ØØ1643	POINTER TO RIB BLOCK
*	2ØØØØØ	FILE PROTECTION CODE (Ø-2) AND START LOCATION OF RIB
	1425Ø1	DATE FILE CREATED (12-21-71)

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.

```
↑C

DOS-15 V3A000
$A TT -12

$A DK -14

$DUMP

DUMP V3A000
>1777#
```

A TT -12/RK -14 2P

1777# MFD

RK = 1777 # UIC-PAG UIC: 105

```
  0  777777  777777  000034  401776  111723  000053  400010  000000
 10  200107  000066  400010  000000  021613  000064  400010  000000
 20  230322  001461  000010  000000  020410  000653  000010  000000
 30  200523  001451  000010  000000  221310  001733  000010  000000
 40     TO      367     CONTAINS 000000
370  000000  000000  000000  000000  000000  000000  777777  777777
DUMP V3A000
>1776#
```

1776# SAT

STORAGE ALLOCATION TABLE RK 1776#

```
  0  004000  004000  002432  777777  777777  777777  777777  777777
 10     TO       67     CONTAINS 777777
 70  777777  775777  777777  777777  777777  777777  777777  777777
100     TO      107     CONTAINS 777777
110  777777  777777  777777  776520  000000  000000  000000  000000
120     TO      367     CONTAINS 000000
370  000000  000000  000000  000000  000000  000000  777777  777777
DUMP V3A000
>66#
```

66# PAG UFD

DOT START BLK DATE

```
  0  040424  000000  021116  000663  000013  000707  200136  073104
 10  053005  032524  021116  000666  000003  000672  200146  073104
 20  061703  011400  021116  000674  000022  000761  200062  073104
 30  561411  022200  021116  000713  000077  001327  200254  031205
 40  561417  010400  021116  001356  000011  001376  200244  101204
 50  111623  011414  232203  001205  000006  001217  200172  073104
 60  111623  052222  232203  001221  000010  001445  200026  101204
 70  111623  242203  021116  001244  000001  001244  200170  073104

100     TO      367     CONTAINS 000000
370  000000  000000  000000  000000  000000  000000  777777  777777
DUMP V3A000
>653#
```

653# BDH UFD

```
  0  062416  611516  021116  001455  000001  001455  200152  030305
 10  042515  000000  021116  001457  000001  001457  200066  030305
 20  042515  153100  300324  001465  000001  001465  200026  030305
 30  042515  153100  300325  001467  000022  001555  200002  030305
 40  561411  022265  021116  001534  000001  001534  200052  030305
 50     TO      367     CONTAINS 000000
370  000000  000000  000000  000000  000000  000000  777777  777777
DUMP V3A000
>
```

PATCH:

SUBJECT: DOS-15 PATCH TO DUMP V9A

The following patch corrects a problem in DUMP which outputs incorrect information on selective dumps.

LOCATION	OLD CONTENTS	NEW CONTENTS	NEW SYMBOLIC	COMMENTS
16256	217406	617472	JMP PATCH	/Patch area
17472	-	116Ø34	JMS DEVICE	/Device check
17473	-	2174Ø6	LAC (-1)	/Restore inst.
17474	-	616257	JMP BACK	/Return
17224	1Ø64ØØ	2Ø64ØØ		

```
DOS-15 VIA
$MICLOG SYS

$DUMP

DUMP V9A
↑C

$R PATCH

.DAT      DEVICE   UIC   USE
-14       DKA      SYS   I/O - SYS DEV
-1Ø       TTA      SYS   SECONDARY INPUT

$PATCH

PATCH V1ØA
>DUMP
>L 16256
>16256/2174Ø6>617472
 16257/Ø57172>
>L 17472
>17472/215116>116Ø34
 17473/253512>2174Ø6
 17474/Ø55116>616257
 17475/214753>
>L 17224
>17224/1Ø64ØØ>2Ø64ØØ
>EXIT


DOS-15 VIA
$DUMP

DUMP V9B
>
```

```
                .TITLE PRB.
/
/
/                       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 RESPON-
/ SIBILITY FOR ANY ERRORS THAT MAY APPEAR IN THIS
/ DOCUMENT.
/
/ THE SOFTWARE DESCRIBED IN THIS DOCUMENT IS FUR-
/ NISHED 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 PRO-
/ VIDED IN WRITING BY DIGITAL.
/
/ DIGITAL EQUIPMENT CORPORATION ASSUMES NO RESPONSIBILIT
/ FOR THE USE OR RELIABILITY OF ITS SOFTWARE ON EQUIP-
/ MENT THAT IS NOT SUPPLIED BY DIGITAL.
/
/ COPYRIGHT (C) 1974, BY DIGITAL EQUIPMENT CORPORATION
/
/
                .EJECT
```

```
                        /COPYRIGHT 1970, DIGITAL EQUIPMENT CORP., MAYNARD, MASS.
                        /PRB.
                        /PRB.=IOPS PAPER TAPE READER HANDLER == IOPS ASCII  .Y.
                        /M. SIFNAS/J. MURPHY
                        /7-30-68
                        /CALLING SEQUENCE:
                        /.INIT
                        /CAL+.DAT SLOT(9-17)
                        /1
                        /0
                        /0
                        /.READ
                        /CAL+D.M.(6-8)+.DAT SLOT (9-17)
                        /10
                        /LINE BUF. ADDR.
                        /-WC OF L.B. (2'S COMP)
                        /.WAIT
                        /CAL+.DAT SLOT (9-17)
                        /12
          000003  A     .MED=3
          700101  A     RSF=700101
          700102  A     RCF=700102
          700112  A     RRB=700112
          700104  A     RSA=700104
          700144  A     RSS=700144
                                .GLOBL PRB.
00000  R  040340  R     PRB.    DAC PRCALP              /CAL POINTER
00001  R  040341  R             DAC PRARGP      /ARG POINTER
00002  R  440341  R             ISZ PRARGP      /INDEX TO FUNCTION ADDR.
00003  R  220341  R             LAC* PRARGP     /FUNCTION
00004  R  440341  R             ISZ PRARGP      /INDEX TO NEXT ARGUMENT.
00005  R  340422  R             TAD (JMP PRTABL
00006  R  040014  R             DAC PRTABL
00007  R  700314  A             IORS            /SET ION=IOF SWITCH
00010  R  750100  A             SMAICLA /AS FUNCTION OF ITS STATE
00011  R  777740  A             LAW 17740       /ON ENTRY INTO CAL LEVEL
00012  R  340160  R             TAD PRION
00013  R  040052  R             DAC PRDBK
00014  R  740040  A     PRTABL  XX
00015  R  600031  R             JMP PRIN                /1=.INIT
00016  R  740000  A             NOP             /2=.DELET, .RENAM, .FSTAT = IGN
00017  R  600056  R             JMP PRSEEK      /3=.SEEK = IGNORED.
00020  R  600027  R             JMP PRERR               /4=.ENTER
00021  R  600027  R             JMP PRERR               /5=.CLEAR
00022  R  600045  R             JMP PRWAIT              /6=.CLOSE
00023  R  600052  R             JMP PRDBK               /7=.MTAPE = IGNORED.
00024  R  600075  R             JMP PRRED               /10=.READ
00025  R  600027  R             JMP PRERR               /11=.WRITE
00026  R  600060  R             JMP PRWATR              /12=.WAIT OR .WAITR
00027  R  760006  A     PRERR   LAW 6           /ILL. FUNCTION=CAL ADDR IN .MED
00030  R  620423  R             JMP* (.MED+1
                        /INIT PTR ROUTINE
```

```
00031 R 440341 R    PRIN    ISZ PRARGP      /INDEX TO BUFFER SIZE RETURN REG
00032 R 200424 R            LAC (64
00033 R 060341 R            DAC* PRARGP             /STANDARD BUFFER SIZE=52
00034 R 440341 R            ISZ PRARGP              /INDEX TO RETURN ADDRESS
00035 R 000050 A            CAL 50          /API ADDR. = ONCE ONLY CODE, THE
00036 R 000016 A    PRLBHP  16              /.SETUP = L.B.H. POINTER,
00037 R 700101 A    PRDBP   RSF             /       = DATA WORD POINTER
00040 R 000140 R    PARER   PTRINT          /PARITY ERROR SWITCH
00041 R 200043 R    PRUND   LAC .+2         / = I/O UNDERWAY
00042 R 040035 R    PTRWC   DAC .-5         / = 2'S COMP WORD COUNT
00043 R 600044 R    PRCHAR  JMP PRSTOP              / = CHAR PROCESSED
                    /STOP PTR ROUTINE, CLEARING I/O SWITCH
00044 R 140041 R    PRSTOP  DZM PRUND       /CLEAR I/O UNDERWAY INDICATOR
                    /PTR WAIT
00045 R 200041 R    PRWAIT  LAC PRUND       /STILL READING
00046 R 741200 A            SNA
00047 R 600052 R            JMP PRDBK
00050 R 200340 R    PRBUSY  LAC PRCALP      /BUSY. RETURN
00051 R 040341 R            DAC PRARGP      /TO USER CAL.
00052 R 740040 A    PRDBK   XX              /ION OR IOF
00053 R 703344 A            DBR             /DEBREAK FROM CAL LEVEL AND REST
00054 R 400055 R            XCT .+1
00055 R 620341 R            JMP* PRARGP     /EXIT
      000052 R      PRNOR=PRDBK
00056 R 440341 R    PRSEEK  ISZ PRARGP
00057 R 600052 R    PRDBKI  JMP PRDBK
00060 R 761000 A    PRWATR  LAW 1000
00061 R 520340 R            AND* PRCALP
00062 R 741200 A            SNA
00063 R 600045 R            JMP PRWAIT      /.WAIT
00064 R 200340 R            LAC PRCALP      /.WAITR
00065 R 500425 R            AND (700000     /L.XM.MP
00066 R 040340 R            DAC PRCALP
00067 R 220341 R            LAC* PRARGP     /BUSY ADDRESS
00070 R 500426 R            AND (77777
00071 R 240340 R            XOR PRCALP
00072 R 040340 R            DAC PRCALP
00073 R 440341 R            ISZ PRARGP      /TO NON-BUSY
00074 R 600045 R            JMP PRWAIT
                    /PTR READ ROUTINE
00075 R 200041 R    PRRED   LAC PRUND       /I/O UNDERWAY?
00076 R 740201 A            SZA!CMA /NO START IT UP
00077 R 600050 R            JMP PRBUSY      /YES-WAIT IN BUSY LOOP BACK TO C
                    /START UP PTR
00100 R 040041 R    PRSTRT  DAC PRUND       /SET I/O UNDERWAY SWITCH (777777
00101 R 200057 R            LAC PRDBKI      /EXIT TO USER-MAINSTREAM
00102 R 040137 R            DAC PTROUT
00103 R 220341 R    PRNEXR  LAC* PRARGP     /L.B.H POINTER IN CALL
00104 R 040037 R            DAC PRDBP
00105 R 040036 R            DAC PRLBHP      /L.B.H. POINTER
00106 R 440341 R            ISZ PRARGP
00107 R 220341 R            LAC* PRARGP
```

```
00110 R 040042 R            DAC PTRWC       /-L.B.W.C. (2'S COMP)
00111 R 440341 R            ISZ PRARGP      /INDEX TO POINT TO EXIT
00112 R 767000 A            LAW 7000
00113 R 520340 R            AND* PRCALP             /CHECK FOR IOPS ASCI' MO
00114 R 540427 R            SAD (2000
00115 R 600120 R            JMP .+3
00116 R 760007 A            LAW 7           /ILLEGAL DATA MODE
00117 R 620423 R            JMP* (.MED+1
00120 R 200430 R            LAC (1002               /IOPS ASCII DATA MODE AN
00121 R 040347 R            DAC PRDTCT      /WD. PAIR CT. OF
                                            /1 FOR HEADER.
00122 R 100317 R            JMS PRNXWD              /INDEX PAST L.B. HEADER
00123 R 100317 R            JMS PRNXWD              /FOR IOPS ASCII
00124 R 777773 A    PR1CR   LAW 17773               /5/7 CHAR
00125 R 040344 R            DAC PTR57               /COUNTER
00126 R 140346 R            DZM PRCCT       /CLEAR CHAR CT.
00127 R 140345 R            DZM PR8CT       /CLEAR ASCII 8TH BIT SET COUNTER
00130 R 140040 R            DZM PARER       /CLEAR PARITY ERR. SWITCH
00131 R 700314 A    PROUT2  IORS
00132 R 500431 R            AND (1000
00133 R 740200 A            SZA
00134 R 600172 R            JMP PREOM
00135 R 700002 A            IOF
00136 R 700104 A            RSA
00137 R 740040 A    PTROUT  XX      /JMP PRDBK OR JMP PRDISM
                    /PTR INTERRUPT SERVICE
00140 R 600151 R    PTRINT  JMP PTRPIC      /PIC ENTRY
00141 R 040342 R            DAC PTRAC       /API ENTRY, SAVE AC
00142 R 200140 R            LAC PTRINT      /PIC OR API, L, EM, MP
00143 R 040343 R            DAC PROUT
00144 R 700314 A            IORS
00145 R 750100 A            SMA!CLA
00146 R 777740 A            LAW 17740               /PIC OFF
00147 R 340160 R            TAD PRION               /PIC ON
00150 R 600155 R            JMP PRSION
00151 R 040342 R    PTRPIC  DAC PTRAC       /SAVE AC
00152 R 220432 R            LAC* (0 /PIC=PC, L, EM, MP
00153 R 040343 R            DAC PROUT       /SAVE FOR EXIT
00154 R 200160 R            LAC PRION
00155 R 040352 R    PRSION  DAC PRSW
00156 R 700112 A            RRB     /READ PTR BUFFER
00157 R 040043 R            DAC PRCHAR
00160 R 700042 A    PRION   ION
00161 R 200165 R            LAC PRDSMI
00162 R 040137 R            DAC PTROUT
00163 R 200041 R            LAC PRUND               /CHECK FOR STOP SINCE LA
00164 R 741200 A            SNA             /O.K.
00165 R 600200 R    PRDSMI  JMP PRDISM              /IGNORE LAST READ, STOP
00166 R 700314 A            IORS
00167 R 500431 R            AND (1000       /PTR NOT READY? (IORS8=1)
00170 R 741200 A            SNA
00171 R 600207 R            JMP PRIOA
```

```
                        /END OF PAPER TAPE ROUTINE
00172 R 200433 R    PREOM   LAC (15          /CLEAR I/O UNDERWAY
00173 R 040043 R            DAC PRCHAR      /FAKE OUT END OF LINE TEST
00174 R 200423 R            LAC (4          /CHANGE MODE
00175 R 340347 R            TAD PRDTCT      /TO EOM
00176 R 040347 R            DAC PRDTCT
00177 R 600257 R            JMP PRPAD
00200 R 200342 R    PRDISM  LAC PTRAC
00201 R 400352 R            XCT PRSW
00202 R 703344 A            DBR             /DEBREAK FROM HANDLER LEVEL.
00203 R 400204 R            XCT .+1
00204 R 620343 R            JMP* PROUT
                        /END LINE
00205 R 140041 R    PRIOBB  DZM PRUND       /CLEAR INPUT UNDERWAY INDICATOR
00206 R 400137 R            XCT PTROUT      /EXIT
                        /PROCESS IOPS ASCII
00207 R 200042 R    PRIOA   LAC PTRWC               /SEE IF EXCESS DATA
00210 R 740100 A            SMA
00211 R 600277 R            JMP PRASE3              /YES---CONTINUE UNTIL C.
                        /COMPUTE PARITY AND EXIT IF NULL
00212 R 777770 A            LAW 17770       /PARITY COUNTER (-8)
00213 R 040350 R            DAC PRCNT
00214 R 140351 R            DZM PRCNT1
00215 R 200043 R            LAC PRCHAR
00216 R 500434 R            AND (177
00217 R 540435 R            SAD (12
00220 R 600131 R            JMP PROUT2      /IGNORE LF
00221 R 540436 R            SAD (13
00222 R 600131 R            JMP PROUT2      /IGNORE VT
00223 R 540437 R            SAD (14
00224 R 600131 R            JMP PROUT2      /IGNORE FF
00225 R 200043 R            LAC PRCHAR
00226 R 741200 A            SNA
00227 R 600131 R            JMP PROUT2              /NULL
00230 R 740020 A            RAR
00231 R 741400 A            SZL
00232 R 440351 R            ISZ PRCNT1              /1 BIT COUNTER
00233 R 440350 R            ISZ PRCNT
00234 R 600230 R            JMP .-4
00235 R 741400 A            SZL
00236 R 440345 R            ISZ PRBCT       /8TH BIT=1, ADD TO COUNT
00237 R 440346 R            ISZ PRCCT
00240 R 200351 R    PRIOA1  LAC PRCNT1      /PARITY COUNT=SHOULD BE EVEN
00241 R 740020 A            RAR
00242 R 741400 A            SZL
00243 R 440040 R            ISZ PARER       /NOT EVEN PARITY
00244 R 100324 R            JMS PRENDT      /CONVERT ALTMODES
00245 R 500434 R            AND (177        /DROP ALL BUT 7 BITS
00246 R 540434 R            SAD (177        /DELETE CODE (RUBOUT)=IGNORE
00247 R 600131 R            JMP PROUT2
00250 R 100353 R            JMS PRPK57      /PACK INTO L.B. IN 5/7
00251 R 100324 R            JMS PRENDT
```

```
00252 R 740100 A            SMA
00253 R 600131 R            JMP PROUT2              /NEXT ASCII CHAR
00254 R 200346 R            LAC PRCCT
00255 R 540440 R            SAD (1
00256 R 600124 R            JMP PR1CR           /IGNORE SINGLE CR LINE
00257 R 200344 R    PRPAD   LAC PTR57                   /WORD COUNT ALL SET.
00260 R 540415 R            SAD PR5CNT
00261 R 600265 R            JMP PRASE
00262 R 750000 A            CLA                 /PAD LAST
00263 R 100353 R            JMS PRPK57          /WORD PAIR
00264 R 600257 R            JMP PRPAD
                    /END OF IOPS ASCII LINE
00265 R 200347 R    PRASE   LAC PRDTCT          /WD. PAIR COUNT (INCL. HDR.)
00266 R 060036 R            DAC* PRLBHP         /WD.0 L.B.H.
00267 R 200345 R            LAC PR8CT           /DID ALL CHAR'S HAVE BIT 8
00270 R 540346 R            SAD PRCCT           /NO = IOPS ASCII CHECK PARITY
00271 R 600277 R            JMP PRASE3          /YES= ASSUME NON IOPS ASCII
00272 R 200040 R    PRASE2  LAC PARER           /PARITY ERROR
00273 R 740200 A            SZA     /NO
00274 R 200441 R            LAC (20 /YES
00275 R 260036 R    PRASE4  XOR* PRLBHP         /PARITY
00276 R 060036 R            DAC* PRLBHP         /ERROR INDICATOR.
00277 R 100324 R    PRASE3  JMS PRENDT          /SKIP TO END LINE
00300 R 751101 A            SPA!CLA!CMA
00301 R 600205 R            JMP PRIOBB          /C.R. FOUND = EXIT.
00302 R 340037 R            TAD PRDBP
00303 R 040037 R            DAC PRDBP           /POINTS TO LAST CHAR
00304 R 777400 A            LAW 17400
00305 R 520037 R            AND* PRDBP
00306 R 240442 R            XOR (33             /PUT CR IN LAST WORD PAIR
00307 R 060037 R            DAC* PRDBP
00310 R 440037 R            ISZ PRDBP                   /INCASE MORE BEFORE CR
00311 R 220036 R            LAC* PRLBHP
00312 R 500443 R            AND (60
00313 R 740200 A            SZA
00314 R 600131 R            JMP PROUT2                  /VALIDITY BITS ALREADY S
00315 R 200443 R            LAC (60             /LINE BUFFER OVERFLOW.
00316 R 600275 R            JMP PRASE4
00317 R 000000 A    PRNXWD  0
00320 R 440037 R            ISZ PRDBP           /INDEX TO NEXT DATA WORD
00321 R 440042 R            ISZ PTRWC           /INDEX WORD COUNT
00322 R 620317 R            JMP* PRNXWD         /EXIT FOR NEXT CHAR
00323 R 600265 R            JMP PRASE           /EXIT TO END OF IOPS ASCII LINE
                    /END LINE TEST = CONVERTS ALTMODE TO STANDARD 175
00324 R 000000 A    PRENDT  0
00325 R 200043 R            LAC PRCHAR
00326 R 500434 R            AND (177
00327 R 540433 R            SAD (15             /RETURN
00330 R 760015 A            LAW 15
00331 R 540444 R            SAD (175                    /ALTMODE
00332 R 760175 A            LAW 175
00333 R 540445 R            SAD (176                    /ALTMODE
```

```
00334 R 760175 A            LAW 175
00335 R 540442 R            SAD (33          /ESCAPE
00336 R 760175 A            LAW 175
00337 R 620324 R            JMP* PRENDT
                      /VARIABLES-NOT SAVED-APPLY TO CURRENT ACTIVE REQUEST
00340 R 000000 A      PRCALP  0          /CAL POINTER
00341 R 000000 A      PRARGP  0          /ARG. LIST AND EXIT POINTER
00342 R 000000 A      PTRAC   0          /SAVED AC(INTERRUPT)
00343 R 000000 A      PROUT   0          /PC,L,EM,MP
00344 R 000000 A      PTR57   0          /CHAR. POSITION COUNTER IN 5/7 PAIR
00345 R 000000 A      PR8CT   0          /ASCII-WITH-8TH-BIT-SET-CHAR COUNTER
00346 R 000000 A      PRCCT   0          /CHAR CT.
00347 R 000000 A      PRDTCT  0          /DATA WORD PAIR IN LINE COUNTER
00350 R 000000 A      PRCNT   0          /PARITY CHECK COUNTER
00351 R 000000 A      PRCNT1  0          /1 BIT COUNTER FOR PARITY CHECK
00352 R 000000 A      PRSW    0                    /ION OR IOF
                      /8/7 IOPS ASCII PACKING ROUTINE.
                      /       PTR57 IS INITIALIZED TO 777773
                      /       PRIOR TO THE 1ST CALL.
                      /
00353 R 000000 A      PRPK57  0                    /CHAR. IN AC BITS 11-17.
00354 R 742020 A              RTR                  /MOVE TO AC BITS 0-6
00355 R 742020 A              RTR
00356 R 742020 A              RTR
00357 R 742020 A              RTR
00360 R 040350 R              DAC PRTMP
00361 R 777771 A              LAW 17771                  /-7
00362 R 040351 R              DAC PRLPCT
00363 R 200350 R      PRPKBK  LAC PRTMP                  /ROTATE CHAR LEFT
00364 R 740010 A              RAL                  /7 BITS THROUGH
00365 R 040350 R              DAC PRTMP                  /THE DOUBLE WORD
00366 R 200421 R      PR8CK2  LAC PRRTHF           /ACCUMULATOR
00367 R 740010 A              RAL                  /PRLFHF/PRRTHF.
00370 R 040421 R              DAC PRRTHF
00371 R 200420 R              LAC PRLFHF
00372 R 740010 A              RAL
00373 R 040420 R              DAC PRLFHF
00374 R 200351 R              LAC PRLPCT
00375 R 745200 A              SNA!CLL
00376 R 600404 R              JMP PRPDNE           /2 WORDS ALL SET.
00377 R 440351 R              ISZ PRLPCT           /IS 7 TIMES COUNT EXHAUSTED?
00400 R 600363 R              JMP PRPKBK           /NO.
00401 R 440344 R              ISZ PTR57                  /DO WE HAVE 5 CHARS.
00402 R 620353 R              JMP* PRPK57          /NO. EXIT
00403 R 600366 R              JMP PR8CK2           /SHIFT LEFT ONCE MORE.
00404 R 200420 R      PRPDNE  LAC PRLFHF           /PLACE ACCUMULATED
00405 R 060037 R              DAC* PRDBP           /2 WORDS INTO
00406 R 100317 R              JMS PRNXWD           /USERS LINE BUFFER,
00407 R 200421 R              LAC PRRTHF           /UPDATING POINTERS.
00410 R 060037 R              DAC* PRDBP
00411 R 200347 R              LAC PRDTCT           /INCREMENT
00412 R 340431 R              TAD (1000                  /DATA WD. PAIR
```

```
00413 R 040347 R                DAC PRDTCT      /COUNT
00414 R 100317 R                JMS PRNXWD
00415 R 777773 A     PRSCNT     LAW 17773               /RESET 5 CHAR COUNTER
00416 R 040344 R                DAC PTR57
00417 R 620353 R                JMP* PRPK57
        000350 R     PRTMP=PRCNT                /TEMP. STORAGE FOR 5/7 CHAR.
        000351 R     PRLPCT=PRCNT1              /ROTATE 7 BITS COUNTER.
00420 R 000000 A     PRLFMF     0               /2 WORD ACCUMULATOR FOR
00421 R 000000 A     PRRTMF     0               /5/7 WORD PAIR.
        000000 A                .END
00422 R 600014 R *L
00423 R 000004 A *L
00424 R 000064 A *L
00425 R 700000 A *L
00426 R 077777 A *L
00427 R 002000 A *L
00430 R 001002 A *L
00431 R 001000 A *L
00432 R 000000 A *L
00433 R 000015 A *L
00434 R 000177 A *L
00435 R 000012 A *L
00436 R 000013 A *L
00437 R 000014 A *L
00440 R 000001 A *L
00441 R 000020 A *L
00442 R 000033 A *L
00443 R 000060 A *L
00444 R 000175 A *L
00445 R 000176 A *L
            SIZE=00446     NO ERROR LINES
```

TABLE OF CONTENTS

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 PIREX montior 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

UC15 Software Manual:.....DEC-15-XUCMA-A-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 PIREX Monitor.

UNICHANNEL - 15 HARDWARE ARCHITECTURE

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.

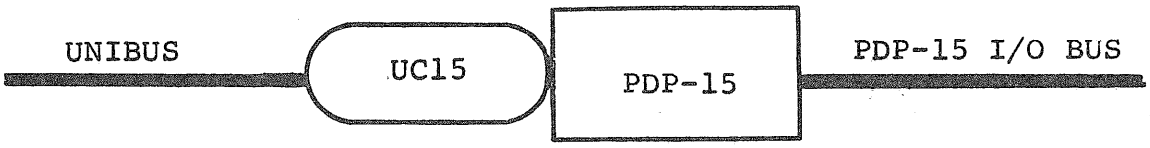


FIGURE 1: Simplified UC15 Diagram

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 interprocessor communications.

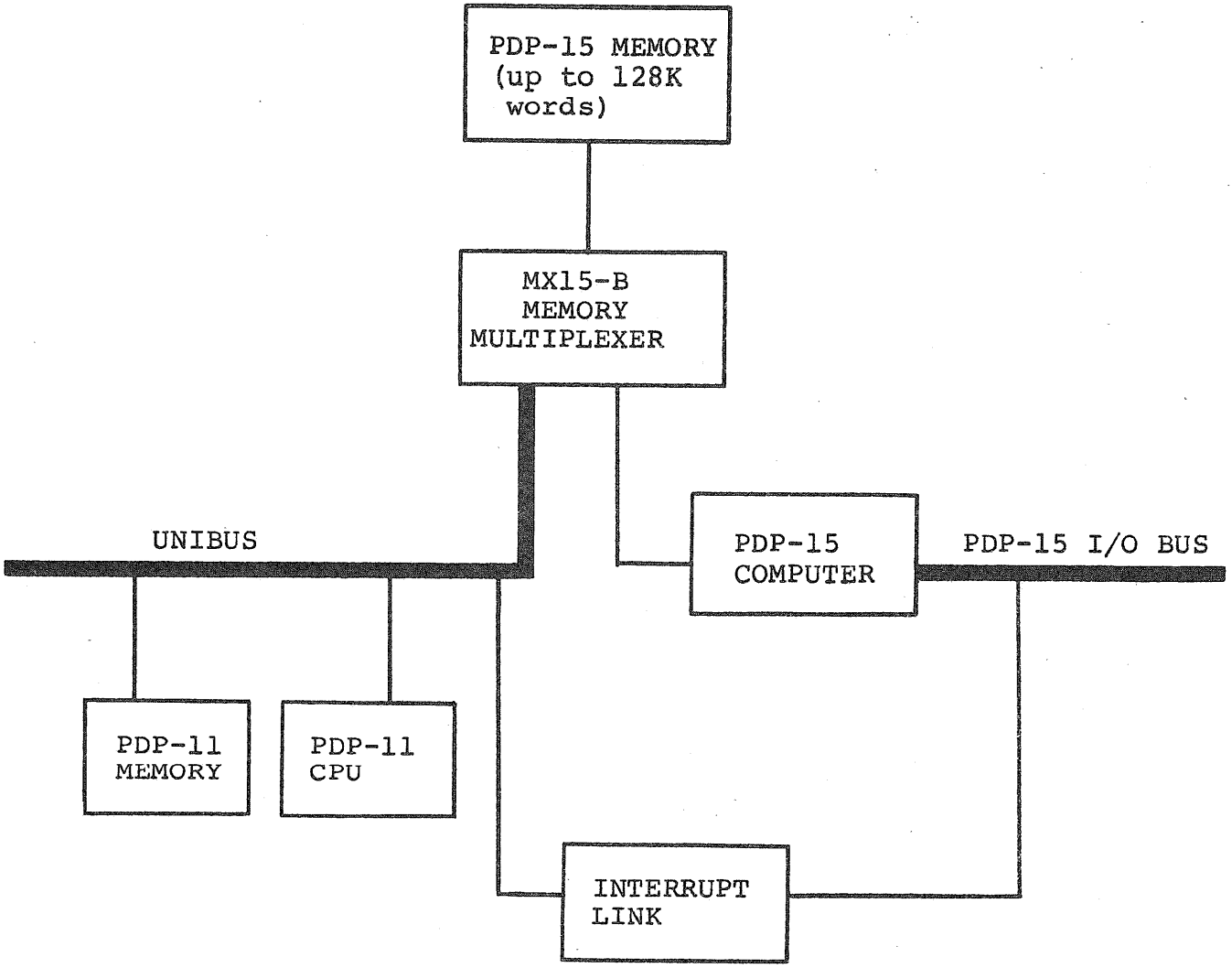


Figure 2: Diagram of UC15 Hardware Interrelationships

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 microsecond 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.

UNICHANNEL - 15 SOFTWARE ARCHITECTURE

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 compalible handlers, and supporting utility functions.

The software system used by UC15 consists of two parts:

1. One component is a mutli-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.

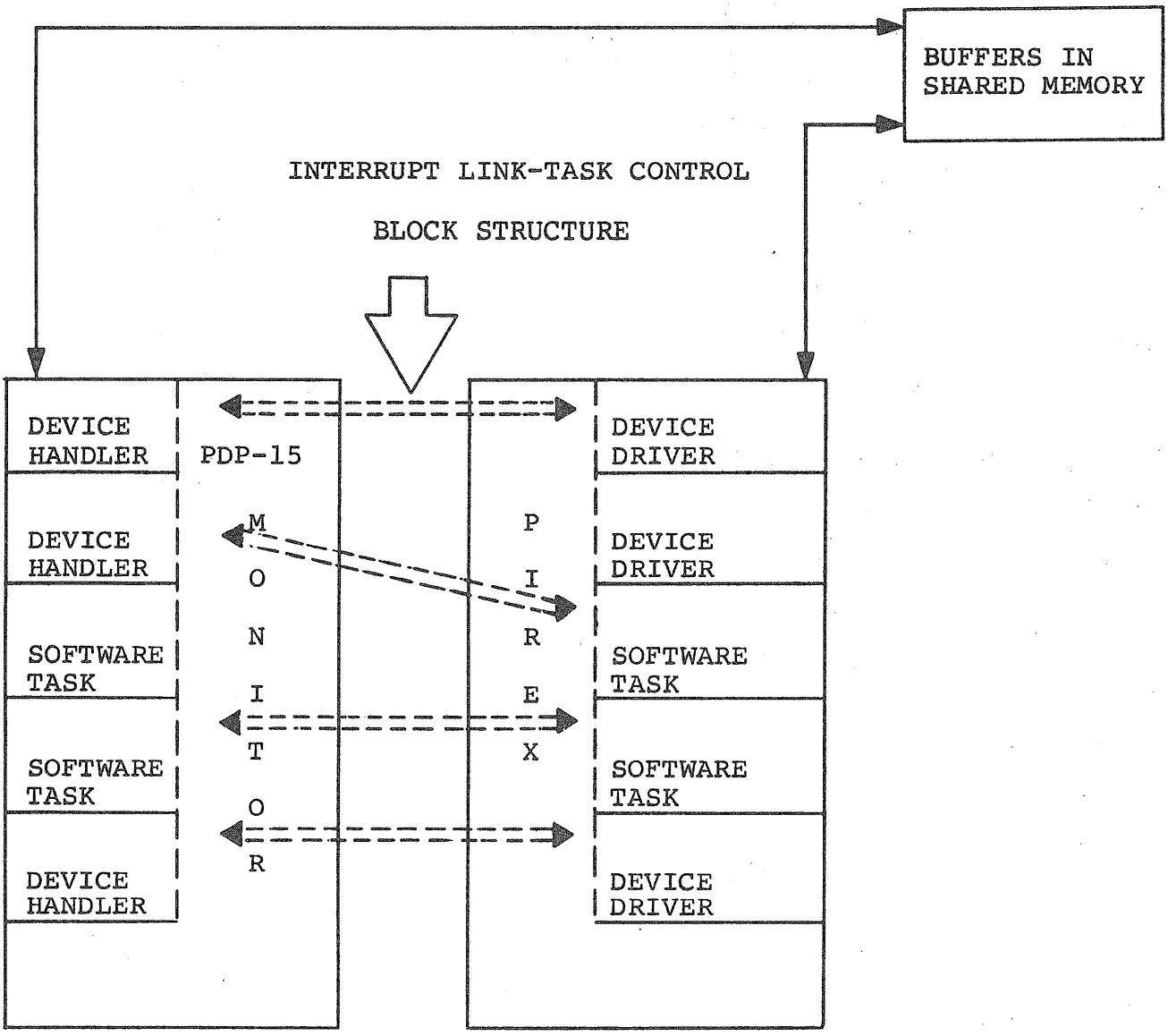


Figure 3 : DIAGRAM OF UNICHANNEL SOFTWARE SYSTEM

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 accessable 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.

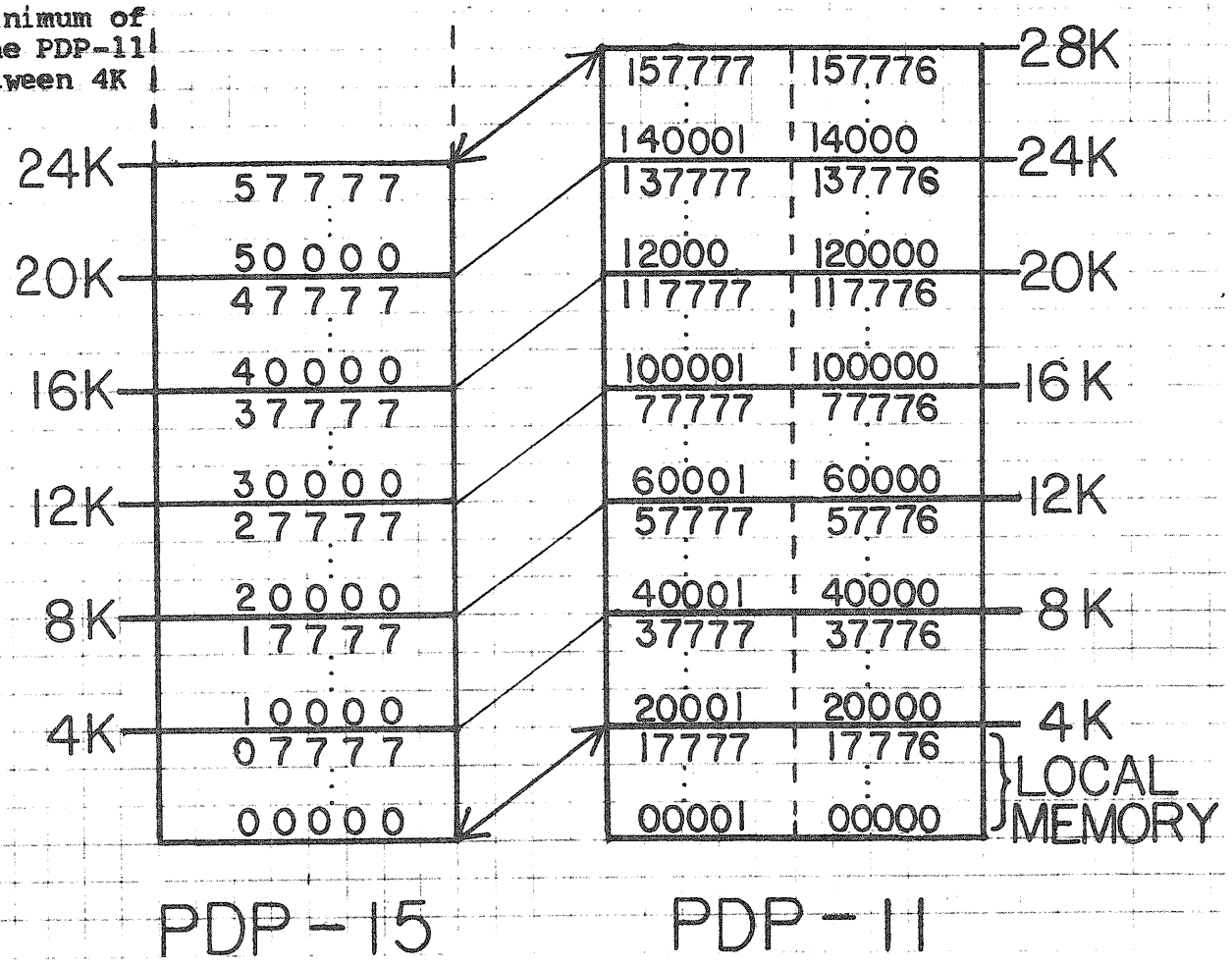
NOTE:

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

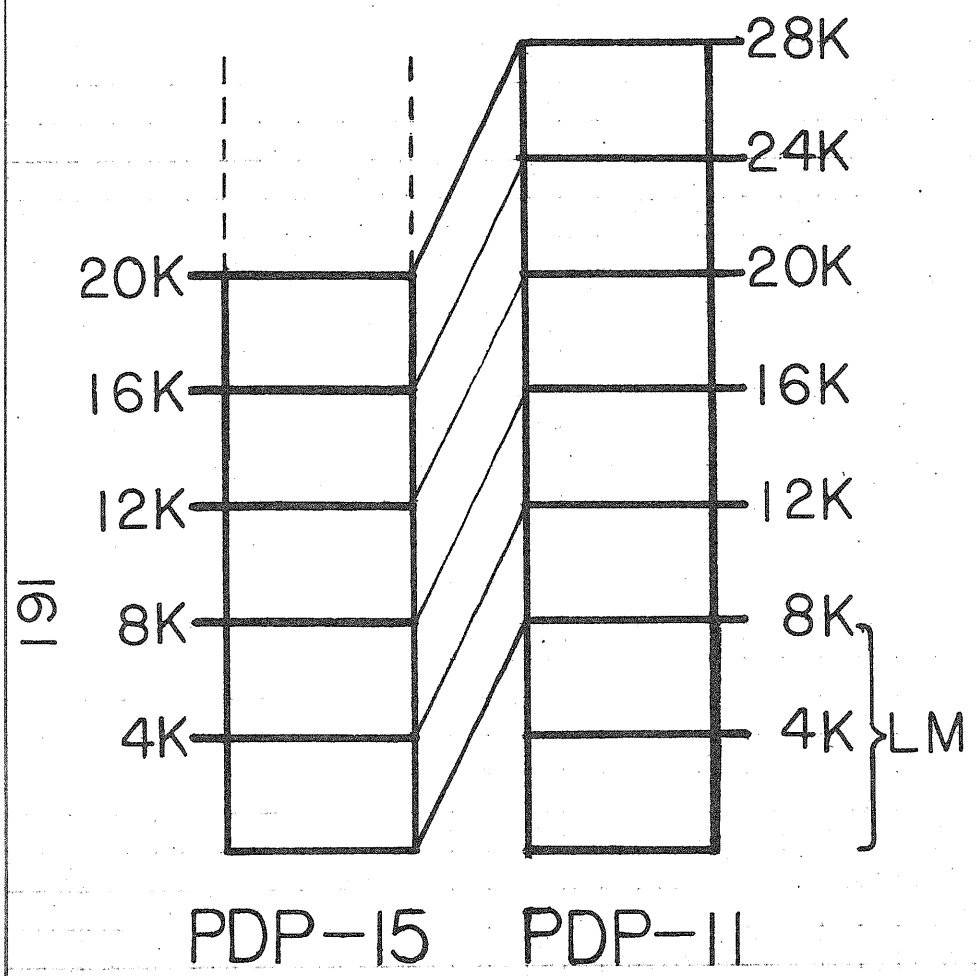
must not exceed 28K.
DOS-15 requires a minimum of 16K of memory on the PDP-15. Therefore, the PDP-11 LOCAL MEMORY may be 12K or 8K or 4K.

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--

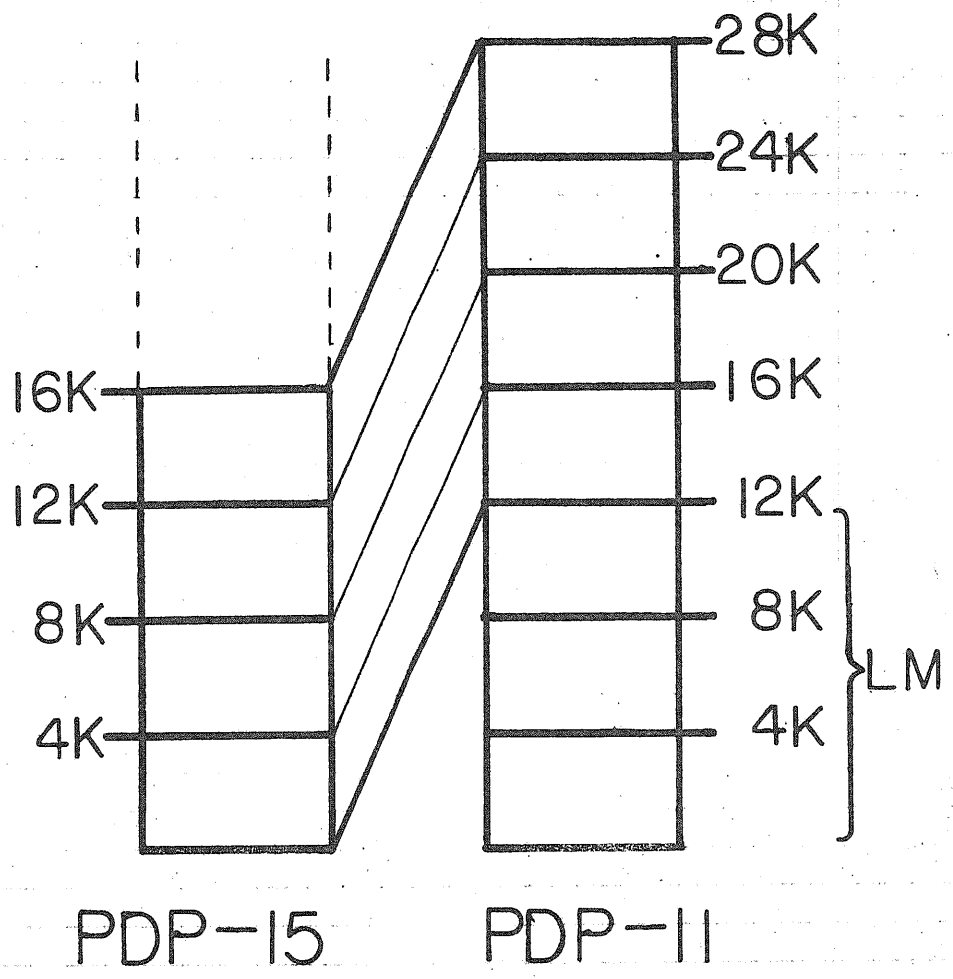
WORDS	BYTES	OCTAL ADDRESSES
4K =	8K	00000-17777
8K =	16K	00000-37777
12K =	24K	00000-57777



4K LOCAL MEMORY
24K COMMON MEMORY



8K LOCAL MEMORY
20K COMMON MEMORY



12K LOCAL MEMORY
16K COMMON MEMORY

ADDRESS CORRESPONDENCE

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

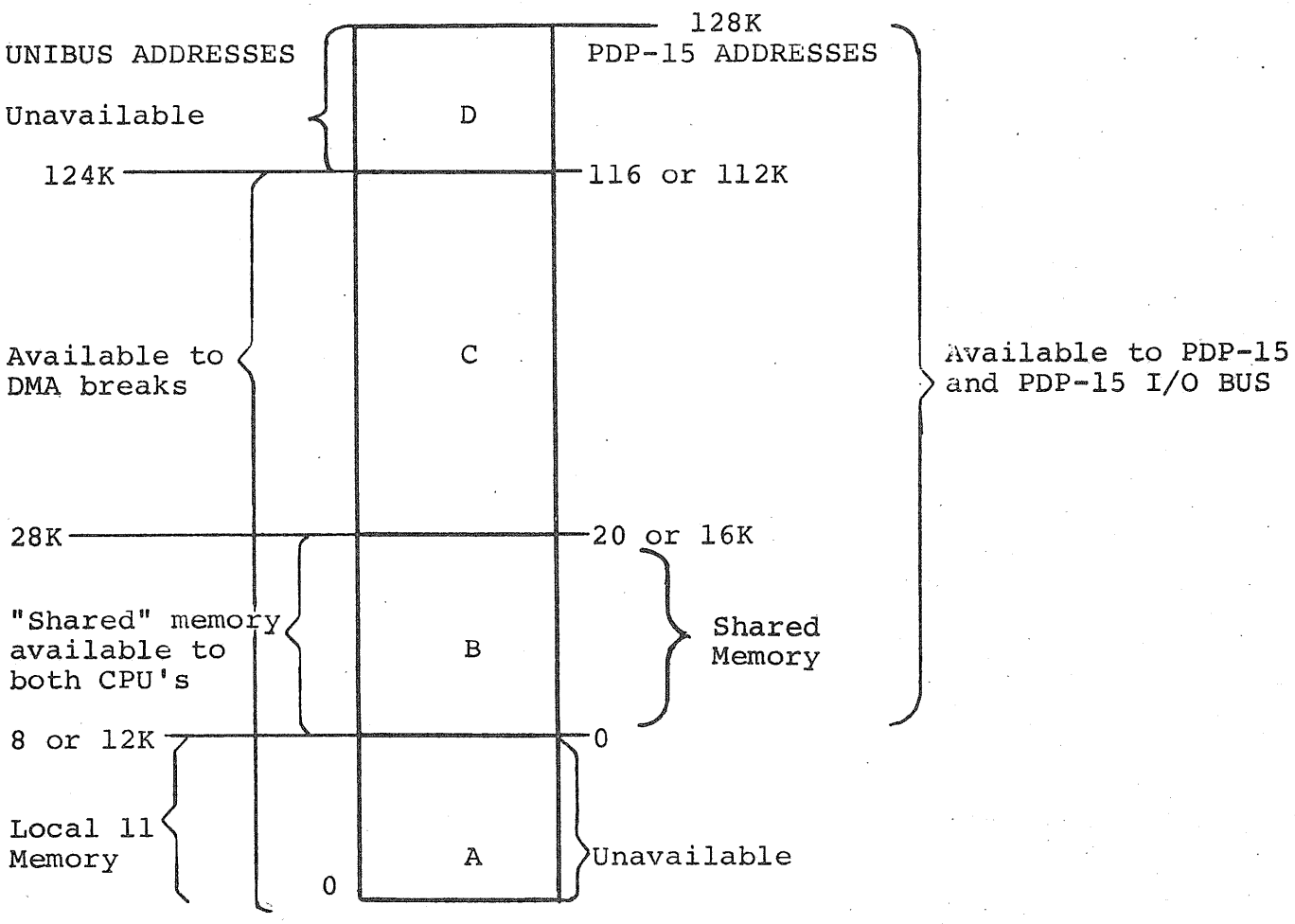
AD15 = (AD11 - LOCAL MEMORY size in bytes)/2

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

LM: LOCAL MEMORY

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 simultinaiety is possible because no memory contention can occur.



"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-11 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:

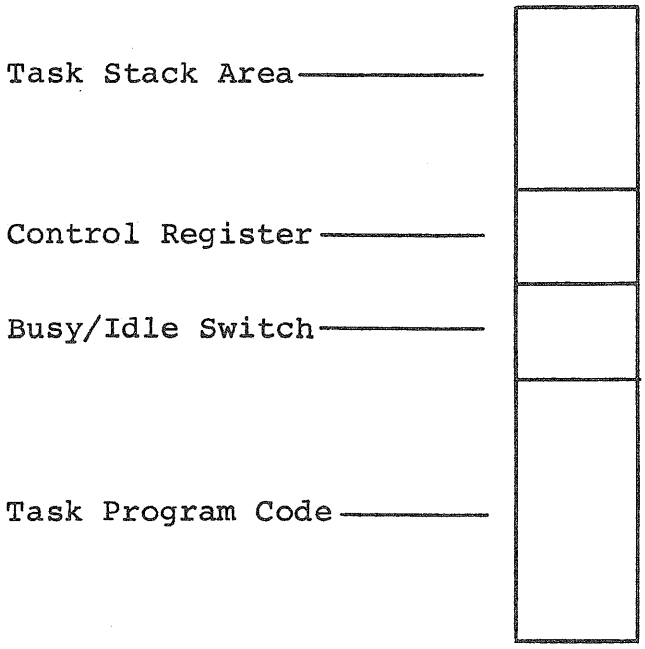


Figure 5 TASK FORMAT

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 Ø and $377_{(8)}$. Location Ø here corresponds to location Ø 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 Ø and 3. A Ø signifies API "Level" Ø , 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 Ø 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 Ø, 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:

Ø = request pending or not yet completed.

1 = request successfully completed.

-2 = (mod 2·16-1) non-existent task referenced.

-3 = (mod 2·16-1) illegal API level given (illegal values are changed to level 3 and processed).

-4 = (mod 2·16-1) illegal directive code given.

-777 = (mod 2·16-1) 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.

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.

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 and API break in the PDP-15 will occur, is the API is enabled and no higher activity is occuring. 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.

LMS1	LMS0	
0	0	0 Local Memory
0	1	4K Local Memory
1	0	8K Local Memory
1	1	12K Local Memory

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

706001 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.

706002 CIOD - Clear I/O Done. Clear the TCBP DONE flag.

706006 LIOR - Load I/O Register and clear TCBP DONE flag. Places the contents of the PDP-15 "AC" 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.

706112 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).

706122 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 "enable interrupts" bit).

706104 CAPIO - Clear APIO flag in DR15-C.

706124 CAPI1 - Clear AP11 flag in DR15-C.

706144 CAPI2 - Clear AP12 flag in DR15-C.

706164 CAPI3 - Clear AP13 flag in DR15-C.

706101 SAPIO - Tests the AP10 flag in the DR15-C and skips the next instruction if the flag is 1.

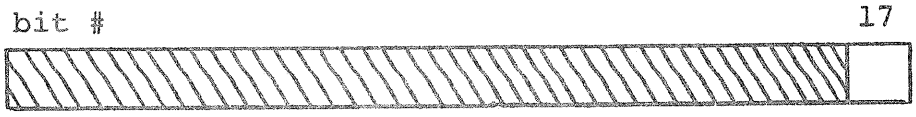
706121 SAPI1 - Tests the API1 flag in the DR15-C and skips the next instruction if the flag is 1.

706141 SAPI2 - Tests the API2 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.

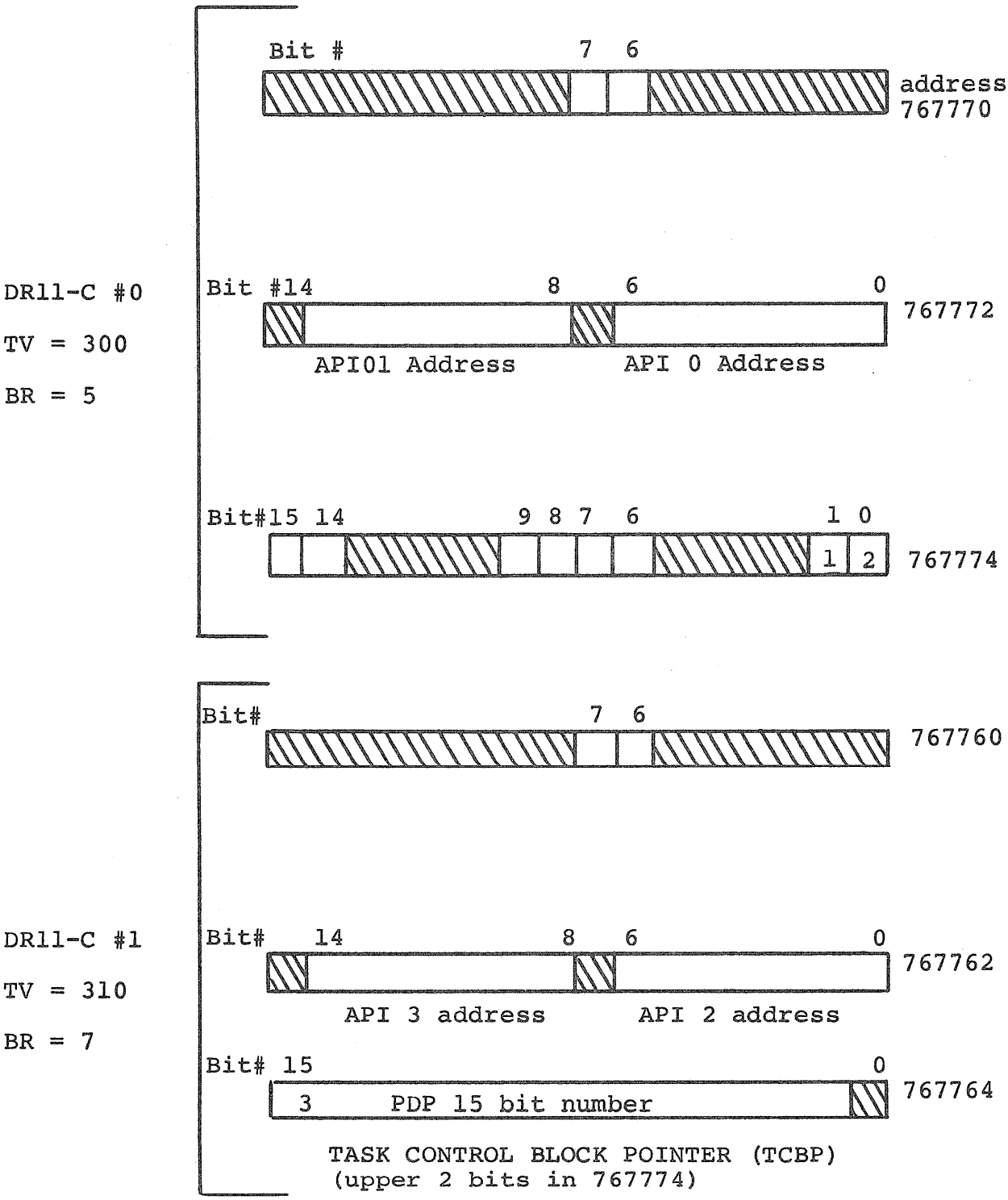


enable PI/API

Figure 7

Figure: 8

PDP-11/05 CONTROL REGISTERS



MX15-B MEMORY MULTIPLEXERS

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
Drives 4 PDP-15 memory bus loads

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 $\pm$ 10% or 230 Vac $\pm$ 10%

Frequency: 50 $\pm$ 2 Hz or 60 $\pm$ 2 Hz

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

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

Unibus Compatability: 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 curent 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.

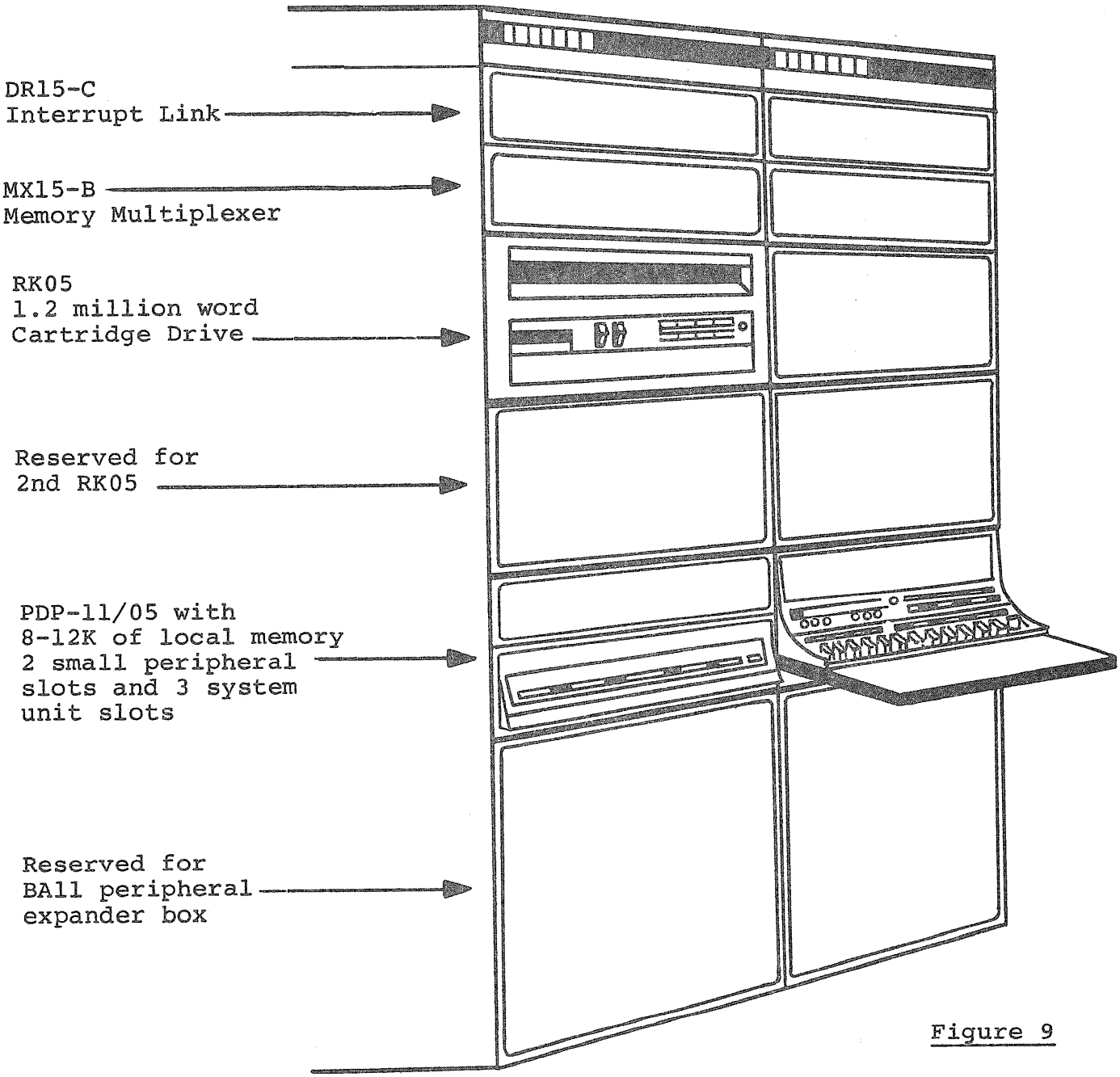


Figure 9

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

UC15-HE	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
UC15-HF	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
UC15-HK	Peripheral Processor: 11/05 or 11/10 - NC or - SA, 2 DR11-C, DR15-C, MX15-B, DD11-B, KY11-JH, H950, MM11-K, 115V.	12K Local Memory
UC15-HL	Peripheral Processor: 11/05 or 11/10-ND or - SA , 2 DR11-C, DR15-C, MX15-B, DD11-B, KY11-JH, H950, MM11-K, 230V.	12K Local Memory
RK15-HE	RK05-AA, RK11-E, UC15-HE, 115V, 60Hz	
RK15-HF	RK05-BB, RK11-E, UC15-HF, 230V, 50Hz.	
RK15-HH	RK05-AB, RK11-E, UC15-HF, 230V, 60Hz.	
RK15-HJ	RK05-BA, RK11-E, UC15-HE, 115V, 50Hz.	
RK15-HK	RK05-AA, RK11-E, UC15-HK, 115V, 60Hz.	
RK15-HL	RK05-BB, RK11-E, UC15-HL, 230V, 50Hz.	
RK15-HM	RK05-AB, RK11-E, UC15-HL, 230V, 60Hz.	
RK15-HN	RK05-BA, RK11-E, UC15-HK, 115V, 50Hz.	
15/76-DE	KP15, ME15-EA, LA30-CA, PC15, KE15, KW15, TC15, TU56, RK15-HE, 115V, 60Hz.	
15/76-DF	KP15, ME15-EB, LA30-CD, PC15-A, KE15, KW15, TC15, TU56, RK15-HF, 230V, 50Hz.	
15/76-DK	KP15, ME15-EA, LA30-CA, PC15, KE15, KW15, TC15, TU56, RK15-HK, 115V, 60Hz.	
15/76-DL	KP15, ME15-EB, LA30-CD, PC15-A, KE15, KW15, TC15, TU56, RK15-HL, 230V, 50Hz.	
15/76-ME	KP15, ME15-EA, LA30-CA, PC15, KE15, KW15, TC59-D, TU10, RK15-HE, 115V, 60Hz.	

15/76-MF	KP15, ME15-EB, LA30-CD, PC15-A, KE15, KW15, TC59-D, TU10, RK15-HF, 230V, 50Hz.
15/76-MK	KP15, ME15-EA, LA30-CA, PC15, KE15, KW15, TC59-D, TU10, RK15-HK, 115V, 60Hz.
15/76-ML	KP15, ME15-EB, LA30-CD, PC15-A, KE15, KW15, TC59-D, TU10, RK15-HL, 230V, 50Hz.

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-SZZB-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
>I MTA.,DTC.↵	Insert routine MTA after routine DTE
>I MTC.↵	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 1 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:

$c\,u_1,u_2,t$)

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␣1,8Ø)
```

This command string means: Evaluate the string "8Ø" 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 8Øth and 81st physical records on tape.

Example:

```
COPY␣1,2,8Ø)
```

The above example causes a transfer of 8Ø 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␣␣1,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␣␣1,1↵
SPACE␣␣1↵

3.1.2 Command Abbreviations

Most commands in MTDUMP 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)␣␣u,t↵

3.2 SETUP COMMANDS

This is a group of commands which generally apply to most major functions of MTDUMP. 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)␣␣u,pdc↵

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" (2ØØ BPI), "5" (556 BPI), or "8" (8ØØ 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,O↵
      or
F␣2,O↵
```

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, 8ØØ BPI, and 7- or 9-channel (according to .SCOM+4, bit 6.)

Usage:

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:

   SPACE␣1,2Ø)

   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:

NUMBER(N) ␣ {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:

SPACE␣1,2ØD↵

The command above causes tape unit 1 to space forward 20_{10} records regardless of the current global radix.

Example:

SPACE␣1,2ØK↵

Similarly, this command spaces the tape forward 20_8 (16_{10}) 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:

VERIFY(V)␣{ON / OFF}↵

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

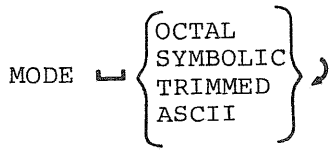
The input tape is output to the Dump File as individual physical records. Each record is represented as a number which indicates record length in ASCII lines. Each line, in turn, contains:

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:



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.

OCTAL	SYMBOLIC	TRIMMED	ASCII
512132	AND 12132	)QZ	REWIN
744634	OPR Ø4634	<&\	
42Ø32Ø	XCT*ØØ32Ø	#CP	D
ØØØØØØ	CAL ØØØØØ	@@@	
777777	LAW 17777	???	A
Ø1Ø2Ø3	CAL 1Ø2Ø3	ABC	

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:

```
LOG␣comments)
comments.....)
(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:

```
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:

```
REWIND(R)␣u)
```

3.3.2 Backspace Tape

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

Usage:

```
BACKSPACE(B)␣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:

```
SPACE(S)␣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:

```
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:

```
OPEN␣filename␣ext↵
```

where: filename is the name of the file to be created.

ext is the filename extension. If omitted, "LST" is the default assumption.

If an OPEN request is not given, the program types

```
NO DUMP FILE OPEN
>
```

on the Teletype and waits for another command.

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:

```
FILE FOUND ON DUMP DEVICE:      filenam ext
DO YOU WISH TO DELETE IT?
>
```

The program then waits for the user to type a response to the query. Typing

)

or

Y)

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)

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.

Message	Meaning
*END OF FILE ENCOUNTERED	An unexpected end-of-file marker was read.
*PHYSICAL EOT ENCOUNTERED	The end-of-tape reflective spot was reached on input or output.
*BUFFER OVERFLOW	The tape record read is too long to be accommodated in the available buffer space.
*BOT ENCOUNTERED	The loadpoint reflective spot was unexpectedly reached during a backspace operation.
*PERMANENT READ ERROR ENCOUNTERED	After 64_{10} read attempts, the input record still has not been transferred correctly. The read/write head is positioned immediately before the record.

3.4.5 Example of Dump Operation

The following example shows the instructions required to dump the file directory of magnetic tape unit Ø 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).

Examples:

```
REWIND 0
SPACE 0,1
MODE OCTAL
DUMP 0,1
     1  747377 000000 747750 777777 750000 000000 000000 000000 000000
    10  000000 000000 000000 000000 000000 000000 000000 000000 131571
    19  000000 233123 231320 021413 233123 111702 141300 233123 562331
    28  231404 233123 561411 022200 021116 561417 010400 021116 040424
    37  710000 021116 031001 111600 021116 302205 060000 021116 050105
    46  061114 021116 161716 061114 021116 252004 012405 233123 053005
    55  032524 233123 050411 240000 233123 201120 000000 233123 066400
    64  000000 233123 150103 221700 233123 066401 000000 233123 150103
    73  221701 233123 152423 070516 233123 031716 260000 233123 152402
    82  171724 232203 152404 251520 021116 000000 000000 000000 000000
    91  000000 000000 000000 000000 000000 000000 000000 000000 000000
   100  000000 000000 000000 000000 000000 000000 000000 000000 000000
   109  000000 000000 000000 000000 000000 000000 000000 000000 000000
   118  000000 000000 000000 000000 000000 000000 000000 000000 000000
   127  000000 000000 000000 000000 000000 000000 000000 000000 000000
   136  000000 000000 000000 000000 000000 000000 000000 000000 000000
   145  000000 000000 000000 000000 000000 000000 000000 000000 000000
   154  000000 000000 000000 000000 000000 000000 000000 000000 000000
   163  000000 000000 000000 000000 000000 000000 000000 000000 000000
   172  000000 000000 000000 000000 000000 000000 000000 000000 000000
   181  000000 000000 000000 000000 000000 000000 000000 000000 000000
   190  000000 000000 000000 000000 000000 000000 000000 000000 000000
   199  000000 000000 000000 000000 000000 000000 000000 000000 000000
   208  000000 000000 000000 000000 000000 000000 000000 000000 000000
   217  000000 000000 000000 000000 000000 000000 000000 000000 000000
   226  000000 000000 000000 000000 000000 000000 000000 000000 000000
   235  000000 000000 000000 000000 000000 000000 000000 000000 000000
   244  000000 000000 000000 000000 000000 000000 000000 000000 000000
   253  000000 000000 000000 000000 000000
```

```
BACKSPACE 0,1
MODE TRIMMED
DUMP 0,1
     1  <;? @@@ <?( ??? =@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
    17  @@@ KM9 @@@ SYS SKP BLK SYS IOB LK@ SYS .SY SLD SYS .LI BR@ BIN
    33  .LO AD@ BIN DDT 9@@ BIN CHA IN@ BIN XRE F@@ BIN EAE FIL BIN NON
    49  FIL BIN UPD ATE SYS EXE CUT SYS EDI T@@ SYS PIP @@@ SYS F4@ @@@
    65  SYS MAC RO@ SYS F4A @@@ SYS MAC ROA SYS MTS GEN SYS CON V@@ SYS
    81  MTB OOT SRC MTD UMP BIN @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
    97  @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
   113  @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
   129  @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
   145  @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
   161  @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
   177  @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
   193  @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
   209  @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
   225  @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
   241  @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@ @@@
   257  @@@
```

3.5 TRANSFER FUNCTION

The COPY command allows the user to perform record-for-record copying of tapes.

Usage:

COPY(C)␣u_1,u_2,t)

where: "u_1" is the source drive

"u_2" 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:

```
REWIND␣1)
REWIND␣2)
COPY␣1,2,EOT)
```

To replace the last data record on unit 2 with the first data record on unit 1:

```
REWIND␣1)           /find first record on 1.
SPACE␣2,EOT)        /find last record on 2.
BACKSPACE␣2,3)      /backspace over two EOF's plus one data record.
COPY␣1,2,1)         /copy 1 record from 1 to 2.
TAPEMARK␣2)         /make a new EOT
TAPEMARK␣2)         /   indicator on 2.
```

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)␣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 ∅ in the buffer.

Example:

EXAMINE␣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:

DLIST ␣ u)

3.7.3 Clear Tape File Directory

Write a new (empty) File Directory on the tape specified.

Usage:

NEWDIR ␣ u)

APPENDIX A

SUMMARY OF COMMANDS

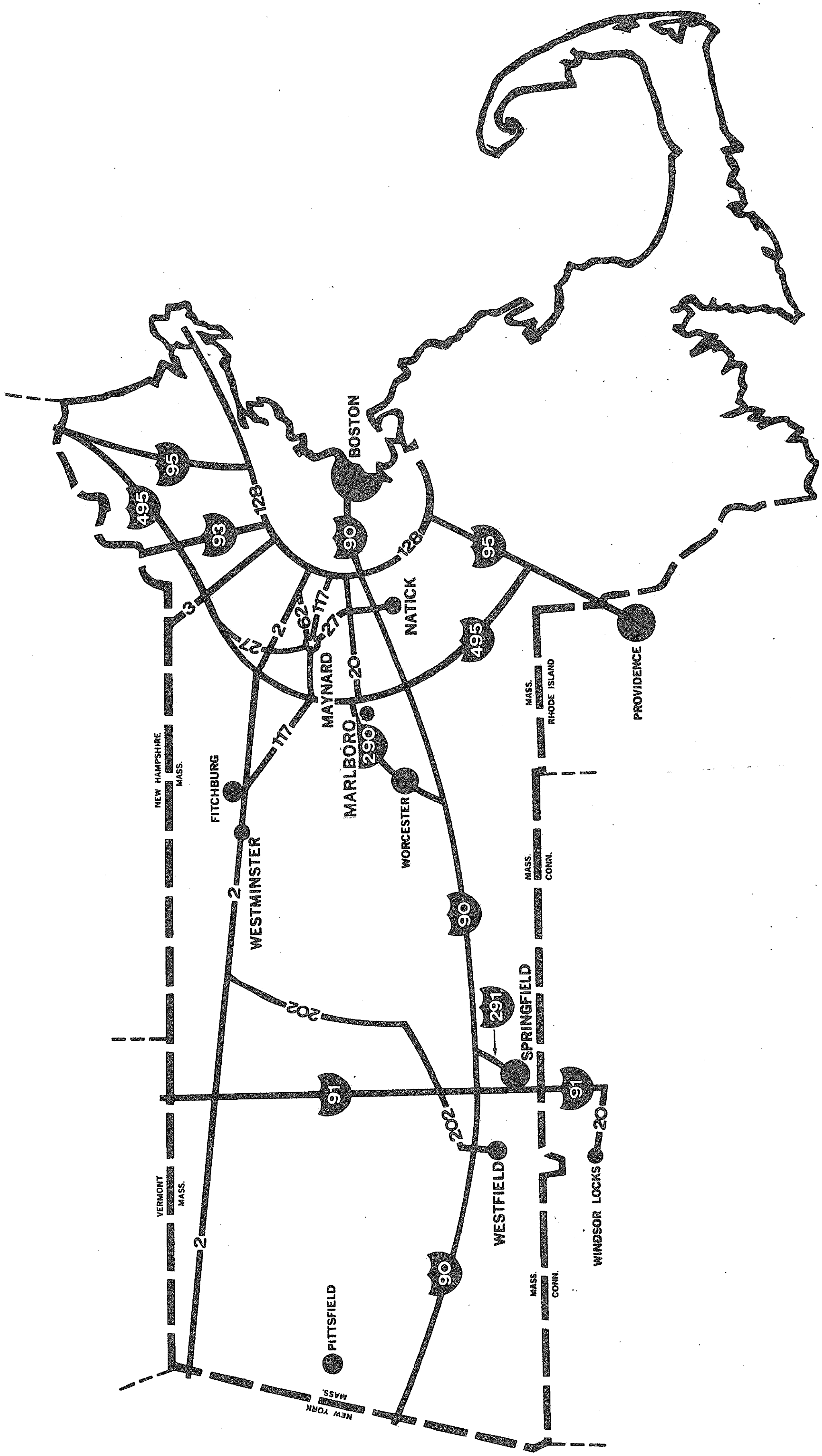
COMMAND	MEANING	PARAGRAPH #
SETUP COMMANDS		
EXIT)	Return Control to Monitor	3.2.8
FORMAT(F)␣u,pdc)	Set Non-Standard Tape	3.2.1
FORMAT(F)␣u,d	Set Standard Tape	3.2.2
LOG␣comments) comments.....) (ALTMODE)	Insert one or more lines of comments	3.2.7
MODE␣{OCTAL / SYMBOLIC / TRIMMED / ASCII})	Dump File Display	3.2.6
NUMBER␣{OCTAL / DECIMAL})	Specify Global Radix	3.2.3

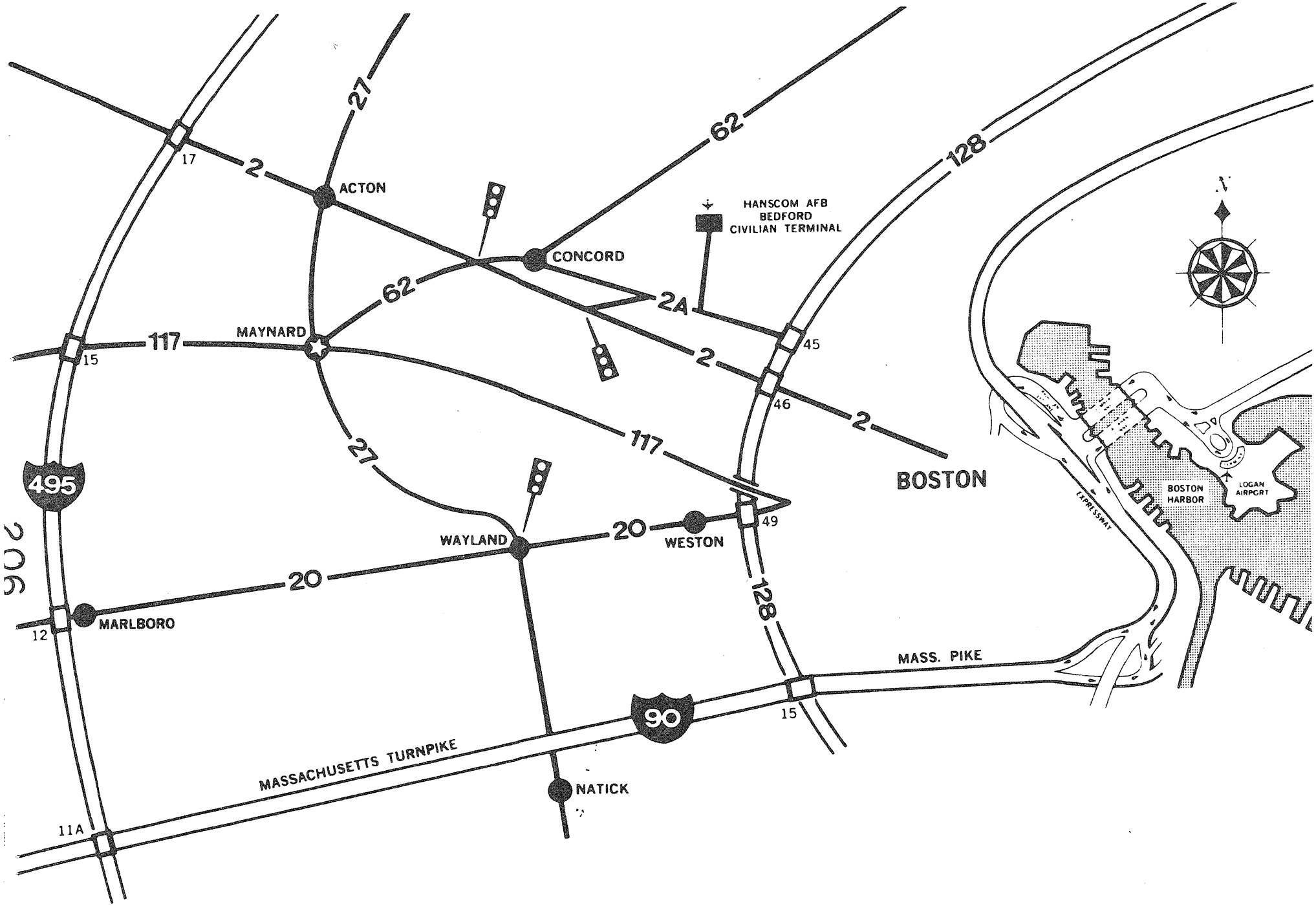
NOTE

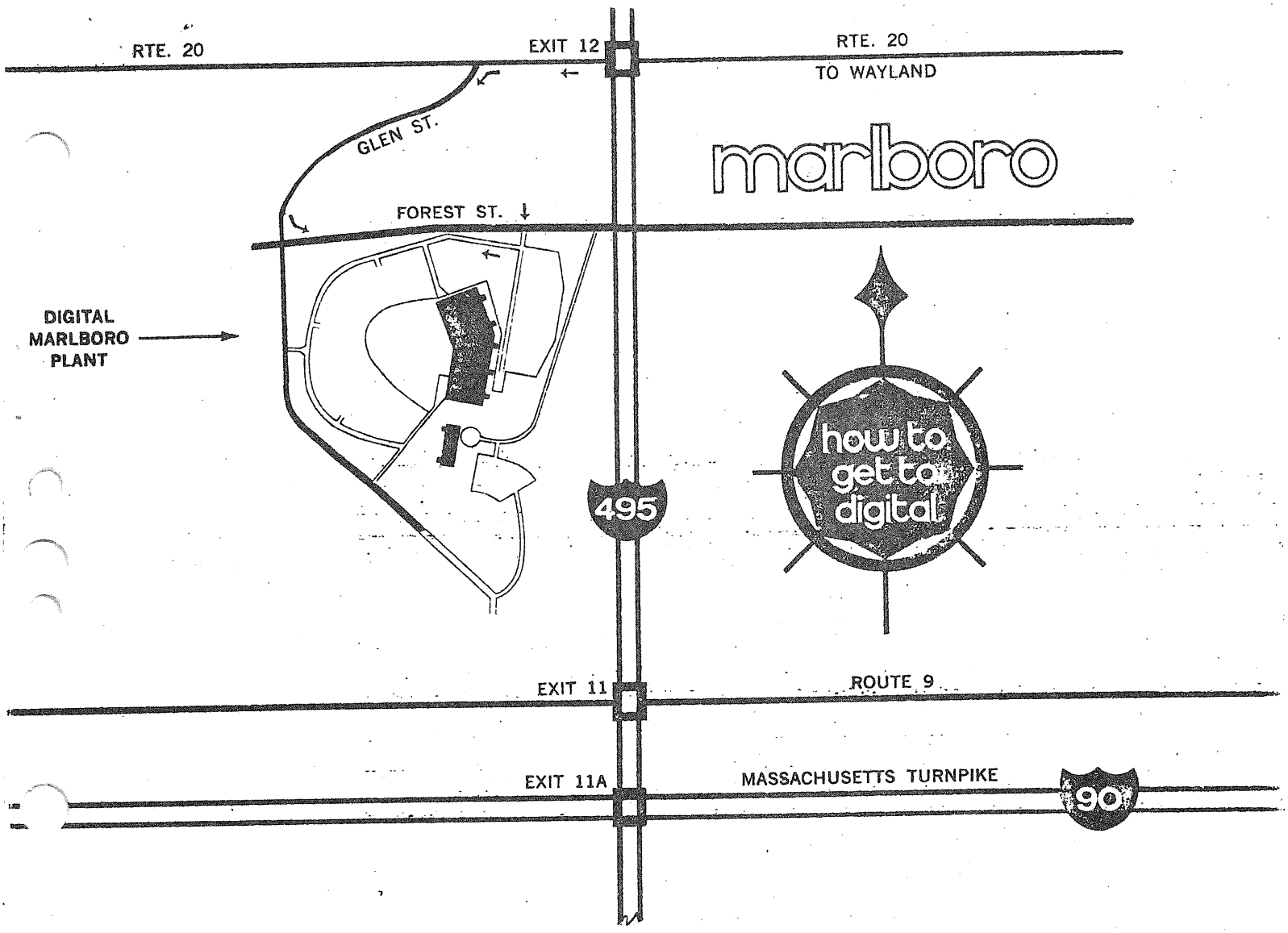
D (decimal) or K (octal) specifies Local Radix which overrides Global Radix during processing of a single command line.

VERIFY(V)␣{ON / OFF})	Bypass Command-Line	3.2.5
MANIPULATIVE COMMANDS		
BACKSPACE(B)␣u,t)	Backspace Tape	3.3.2
REWIND(R)␣u)	Rewind Tape	3.3.1
SPACE(S)␣u,t)	Space Tape	3.3.3
TAPEMARK(T)␣u)	Write End-of-File Marker	3.3.4
DUMP FILE COMMANDS		
CLOSE)	Close Dump Output File	3.4.1
DUMP(D)␣u,t)	Dump Records into Named File	3.4.2
LIST(L)␣u,t)	Dump Records onto teleprinter	3.4.3
OPEN␣filename␣ext)	Open Named File	3.4.1

COMMAND	MEANING	PARAGRAPH #
TRANSFER COMMAND		
COPY(C) u_1,u_2,t	Copy Tape Specified	3.5
FILE MODIFICATION COMMANDS		
EXAMINE(E) n	Examine and Modify Data Words	3.6.2
GET(G) u	Read Single Record	3.6.1
PUT(P) u	Write Single Record	3.6.4
SIZE n	Specify Output Record Length	3.6.3
DIRECTORY COMMANDS		
DDUMP u	Write File Directory in Dump Output File	3.7.1
DLIST u	Print File Directory on Teleprinter	3.7.2
NEWDIR u	Clear Tape File Directory	3.7.3


BOSTON
PROVIDENCE
NATICK
MAYNARD
MARLBORO
WORCESTER
FITCHBURG
WESTMINSTER
SPRINGFIELD
WESTFIELD
WINDSOR LOCKS
PITTSFIELD
NEW HAMPSHIRE
MASS.
VERMONT
MASS.
NEW YORK
MASS.
MASS.
RHODE ISLAND
MASS.
CONN.
MASS.
CONN.
95
495
93
128
90
3
2
62
117
27
20
290
291
91
202


ACTON
CONCORD
MAYNARD
HANSCOM AFB
BEDFORD
CIVILIAN TERMINAL
BOSTON
WAYLAND
WESTON
MARLBORO
NATICK
MASS. PIKE
MASSACHUSETTS TURNPIKE
BOSTON HARBOR
LOGAN AIRPORT
EXPRESSWAY
2
2A
20
27
62
117
128
495
90
17
15
46
45
49
12
11A



marlboro map

DIGITAL EQUIPMENT CORPORATION digital WORLDWIDE SALES AND SERVICE

MAIN OFFICE AND PLANT

Maynard, Massachusetts, U.S.A. 01754 • Telephone: From Metropolitan Boston 646-8600 • Elsewhere: (617)-897-5111
TWX: 710-347-0212 Cable: DIGITAL MAYN Telex: 94-8457

DOMESTIC

NORTHEAST

REGIONAL OFFICE:
235 Wyman Street, Waltham, Mass. 02154
Telephone: (617)-890-0330/0310 Dataphone: 617-890-3012 or 3013

CONNECTICUT
Meriden
240 Pomeroy Ave., Meriden, Conn. 06450
Telephone: (203)-237-8441/7466 Dataphone: 203-237-8205

Fairfield
1275 Post Road, Fairfield, Conn 06430
Telephone (203)-255-5991

NEW YORK
Rochester
130 Allens Creek Road, Rochester, New York
Telephone (716)-461-1700 Dataphone: 716-244-1680

Syracuse
6700 Thompson Road, Syracuse, New York 13211
Telephone: (315)-437-1593/7085 Dataphone: 315-454-4152

MASSACHUSETTS
Marlborough
One Iron Way
Marlborough, Mass. 01752
Telephone (617)-481-7400 Telex 710-347-0348

MID-ATLANTIC

REGIONAL OFFICE:
U.S. Route 1, Princeton, New Jersey 08540
Telephone: (609)-452-2940

FLORIDA
Orlando
Suite 130, 7001 Lake Ellenor Drive, Orlando, Florida 32809
Telephone (305)-851-4450 Dataphone: 305-859-2360

GEORGIA
Atlanta
2815 Clearview Place, Suite 100
Atlanta, Georgia 03040
Telephone: (404)-451-7411 Dataphone: 305-859-2360

NORTH CAROLINA
Durham/Chapel Hill
Executive Park
3700 Chapel Hill Blvd.
Durham, North Carolina 27707
Telephone: (919)-489-3347 Dataphone: 919-489-7832

NEW JERSEY
Fairfield
253 Passaic Ave., Fairfield, New Jersey 07006
Telephone (201)-227-9260 Dataphone 201-227-9260

Metuchen
95 Main Street, Metuchen, New Jersey 08840
Telephone (201)-549-4100/2000 Dataphone: 201-548-0144

MID-ATLANTIC (cont.)

Princeton
U.S. Route 1, Princeton, New Jersey 08540
Telephone: (609)-452-2940 Dataphone: 609-452-2940

NEW YORK
Long Island
1 Huntington Quadrangle
Suite 1S07 Huntington Station, New York 11746
Telephone: (516)-694-4131, (212)-895-8095
Dataphone: 516-293-5693

Manhatten
810 7th Ave., 22nd Floor
New York, N.Y. 10019
Telephone (212)-582-1300

PENNSYLVANIA
Philadelphia
Digital Hall
1740 Walton Road, Blue Bell, Pennsylvania 19422
Telephone: (215)-825-4200

TENNESSEE
Knoxville
6311 Kingston Pike, Suite 21E
Knoxville, Tennessee 37919
Telephone: (615)-588-6571 Dataphone: 615-584-0571

WASHINGTON D.C.
Lanham 30 Office Building
4900 Princess Garden Parkway, Lanham, Maryland
Telephone (301)-459-7900 Dataphone: 301-459-7900 X53

CENTRAL

REGIONAL OFFICE:
1850 Frontage Road, Northbrook, Illinois 60062
Telephone (312)-498-2500 Dataphone 312-498-2500 Ex. 78

INDIANA
Indianapolis
21 Beachway Drive, Suite G
Indianapolis, Indiana 46224
Telephone: (317)-243-8341 Dataphone: 317-247-1212

ILLINOIS
Chicago
1850 Frontage Road
Northbrook, Illinois 60062 Dataphone: 312-498-2500

LOUISIANA
New Orleans
3100 Ridgelake Drive, Suite 108
Metairie, Louisiana 70002
Telephone: (504)-837-0257 Dataphone: 504-833-2800

CENTRAL (cont.)

MICHIGAN
Ann Arbor
230 Huron View Boulevard, Ann Arbor, Michigan 48103
Telephone: (313)-761-1150 Dataphone: 313-769-9883

Detroit
23777 Greenfield Road
Suite 189
Southfield, Michigan 48075 Dataphone: 313-557-3083

MINNESOTA
Minneapolis
8030 Cedar Ave. South, Minneapolis, Minnesota 55420
Telephone (612)-854-6562-3-4-5 Dataphone: 612-854-1410

MISSOURI
Kansas City
12401 East 43rd Street, Independence, Missouri 64055
Telephone: (816)-252-2300 Dataphone: 816-461-3100

St. Louis
Suite 110, 115 Progress Parkway
Maryland Heights, Missouri 63043
Telephone (314)-878-4310 Dataphone: 816-461-3100

OHIO
Cleveland
2500 Euclid Avenue, Euclid, Ohio 44117
Telephone: (216)-948-8484 Dataphone: 216-948-8477

Dayton
3101 Kettering Boulevard
Dayton, Ohio 45439
Telephone: (513)-294-3323 Dataphone: 513-298-4724

OKLAHOMA
Tulsa
3140 S. Winston
Winston Sq. Bldg., Suite 4, Tulsa, Oklahoma 74135
Telephone (918)-749-6476 Dataphone: 918-749-2714

PENNSYLVANIA
Pittsburgh
400 Penn Center Boulevard, Pittsburgh, Pennsylvania 15235
Telephone: (412)-243-9404 Dataphone: 412-824-9730

TEXAS
Dallas
Plaza North, Suite 513
2680 LBJ Freeway, Dallas, Texas 75234
Telephone: (214)-620-2051 Dataphone: 214-620-2081

HOUSTON
6565 Hornwood Drive
Monterey Park, Houston, Texas 77036
Telephone: (713)-777-3471 Dataphone: 713-777-1071

WISCONSIN
Milwaukee
8531 West Capitol Drive, Milwaukee, Wisconsin 53222
Telephone (414)-463-9110 Dataphone: 414-463-9115

WEST

REGIONAL OFFICE:
310 Soquel Way, Sunnyvale, California 94086
Telephone: (408)-735-9200 Dataphone: 408-735-1620

ARIZONA
Phoenix
4358 East Broadway Road, Phoenix, Arizona 85040
Telephone: (602)-268-3468 Dataphone: 602-268-7371

CALIFORNIA
Santa Ana
2110 S. Anne Street, Santa Ana, California 92704
Telephone (714)-979-2460 Dataphone: 714-979-7850

San Diego
6154 Mission Gorge Road
Suite 110, San Diego, California
Telephone: (714)-280-7880/7970 Dataphone: 714-280-7825

San Francisco
1400 Terra Bella, Mountain View, California 94040
Telephone (415)-964-6200 Dataphone: 415-964-1436

Oakland
7850 Edgewater Drive, Oakland, California 94621
Telephone: (415)-635-5453/7830 Dataphone: 415-562-2180

West Los Angeles
1510 Cotner Avenue, Los Angeles, California 90025
Telephone: (213)-479-3791/4318 Dataphone: 213-478-5626

COLORADO
7901 E. Bellevue Avenue
Suite 5, Englewood, Colorado 80110
Telephone: (303)-770-8150 Dataphone: 303-770-6828

NEW MEXICO
Albuquerque
10200 Menual N.E., Albuquerque, New Mexico 87112
Telephone: (505)-296-5411/5428 Dataphone: 505-294-2330

OREGON
Portland
Suite 168
5319 S.W. Westgate Drive, Portland, Oregon 97221
Telephone: (503)-297-3761/3765

UTAH
Salt Lake City
429 Lawn Dale Drive, Salt Lake City, Utah 84115
Telephone (801)-487-4669 Dataphone 801-467-0535

WASHINGTON
Bellevue
13401 N.E. Bellevue, Redmond Road, Suite 111
Bellevue, Washington 98005
Telephone: (208)-545-4058/455-5404 Dataphone 208-747-3754

INTERNATIONAL

EUROPEAN HEADQUARTERS
Digital Equipment Corporation International Europe
81 route de l'Aire
1211 Geneva 26, Switzerland
Telephone: 42 79 50 Telex: 22 683

FRANCE
Digital Equipment France
Centre Silic — Cidex L 225
94533 Rungis, France
Telephone 687-23-33 Telex 26840

GRENOBLE
Digital Equipment France
Tour Mangin
16 Rue Du Gal Mangin
38100 Grenoble, France
Telephone (76)-87-56-01 Telex: 212-32682

GERMAN FEDERAL REPUBLIC
Digital Equipment GmbH
MUNICH
8 Muenchen 13, Wallensteinplatz 2
Telephone 0811-35031 Telex: 524-228

COLOGNE
5 Koeln 41, Aachener Strasse 311
Telephone: 0221-44-40-95 Telex: 888-2269
Telegram: Flip Chip Koeln

FRANKFURT
6078 Neu-Isenburg 2
Am Forstaus Gravebruch 5-7
Telephone 06102-5526 Telex: 41-76-82

HANNOVER
3 Hannover, Podbielskistrasse 102
Telephone: 0511-69-70-95 Telex: 922-852

STUTTGART
D-7301 Kemnat, Stuttgart
Marco-Polo-Strasse 1
Telephone (0771)-45-50-65 Telex: 841-722-393

AUSTRIA
Digital Equipment Corporation Ges.m.b.H
VIENNA
Mariahilferstrasse 136, 1150 Vienna 15, Austria
Telephone 85 51 88

UNITED KINGDOM
Digital Equipment Co. Ltd
U.K. HEADQUARTERS
Fountain House, Butts Centre
Reading RG1 7QN, England
Telephone (0734)-583555 Telex: 8483278

BIRMINGHAM
Maney Buildings
29/31 Birmingham Rd., Sutton Coldfield
Warwickshire, England
Telephone 021-355-5501 Telex: 337-080

BRISTOL
Fish Ponds Road, Fish Ponds
Bristol, England BS163HQ
Telephone: Bristol 651-431

EALING
Bilton House, Uxbridge Road, Ealing, London W 5
Telephone 01-579-2334 Telex: 22371

EDINBURGH
Shiel House, Craigshill, Livingston,
West Lothian, Scotland
Telephone 32705 Telex 727113

LONDON
Management House
43 Parker St., Holborn, London
WC 2B 5PT, England
Telephone: 01-405-2614/4087 Telex: 27560

MANCHESTER
Arndale House
Chester Road, Stretford, Manchester M32 9BH
Telephone (061)-865-7011 Telex 668999

UNITED KINGDOM (cont.)
READING
Fountain House, Butts Centre
Reading RG1 7QN, England
Telephone (0734)-583555 Telex 8483278

NETHERLANDS
Digital Equipment N.V.
THE HAGUE
Sir Winston Churchillian 370
Rijswijk/The Hague, Netherlands
Telephone: 94 9220 Telex: 32533

BELGIUM
Digital Equipment N.V./S.A.
BRUSSELS
108 Rue D'Arlon
1040 Brussels, Belgium
Telephone 02-139258 Telex: 25297

SWEDEN
Digital Equipment AB
STOCKHOLM
Englundavegen 7, 171 41 Solna, Sweden
Telephone 98 13 90 Telex: 170 50
Cable: Digital Stockholm

NORWAY
Digital Equipment Corp. A/S
OSLO
Trondheimsveien 47
Oslo 5, Norway
Telephone 02/68 34 40 Telex: 19079 DEC N

DENMARK
Digital Equipment Aktiebolag
COPENHAGEN
Hellerupveg 68
2900 Hellerup, Denmark

FINLAND
Digital Equipment AB
HELSINKI
Titiamaentie 6
SF-00710 Helsinki 71
Telephone: (090) 370133
Cable: Digital Helsinki

SWITZERLAND
Digital Equipment Corporation S.A.
GENEVA
20, Quai Ernest Ansermet
Boite Postale 23, 1211 Geneva 8, Switzerland
Telephone No. 022 20 40 20 and 20 58 93 and 20 68 93
Telex 26 92 01

ZURICH
Digital Equipment Corp. AG
Schaffhauserstr. 315
CH-8050 Zurich, Switzerland
Telephone 01-46-41-91 Telex 56059

ITALY
Digital Equipment S.p.A.
MILAN
Corso Garibaldi 49, 20121 Milano, Italy
Telephone: (02)-879-031/2/3/4/5 Telex: 843-33815

SPAIN
Digital Equipment Corporation Ltd.
MADRID
Ataio Ingenieros S.A., Enrique Larreta 12, Madrid 16
Telephone: 215 35 43 Telex: 27249

BARCELONA
Ataio Ingenieros S.A., Granduxer 78, Barcelona 6
Telephone: 221 44 66

ISRAEL
DEC Systems Computers Ltd.
TEL AVIV
Suite 103, Southern Habakuk Street
Tel Aviv, Israel
Telephone (03) 443114/440763 Telex: 922-33-3163

CANADA
Digital Equipment of Canada, Ltd
CANADIAN HEADQUARTERS
P.O. Box 11500
Ottawa, Ontario, Canada
K2H 8K8
Telephone: (613)-592-5111 TWX: 610-562-8732

TORONTO
2550 Goldenridge Road, Mississauga, Ontario
Telephone: (416)-270-9400 TWX: 610-492-7118

MONTREAL
9045 Cote De Liesse
Dorval, Quebec, Canada H9P 2M9
Telephone (514)-636-9393 Telex: 610-422-4124

CALGARY/Edmonton
Suite 140, 6840 Fisher Road S.E.
Calgary, Alberta, Canada
Telephone (403)-435-4681 TWX: 403-255-7408

VANCOUVER
Suite 202
644 S.W. Marine Dr., Vancouver
British Columbia, Canada V6P 5Y1
Telephone (604)-325-3231 Telex 610-929-2006

GENERAL INTERNATIONAL SALES
REGIONAL OFFICE
148 Main Street, Maynard, Massachusetts 01754
Telephone: (617) 897-5111
From Metropolitan Boston, 646-8600
TWX: 710-347-0217/0212
Cable: DIGITAL MAYN
Telex: 94-8457

AUSTRALIA
Digital Equipment Australia Pty. Ltd.
ADELAIDE
6 Montrose Avenue
Norwood, South Australia 5067
Telephone (08)-42-1339 Telex: 790-82825

BRISBANE
133 Leichhardt Street
Spring Hill
Brisbane, Queensland, Australia 4000
Telephone (072)-293088 Telex: 790-40316

CANBERRA
27 Collie St.
Fyshwick, A.C.T. 2609 Australia
Telephone (062)-958073

MELBOURNE
60 Park Street, South Melbourne, Victoria 3205
Australia
Telephone (03)-699-2688 Telex: 790-30700

PERTH
643 Murray Street
West Perth, Western Australia 6005
Telephone (092)-21-4993 Telex: 790-92140

SYDNEY
P.O. Box 491, Crows Nest
N.S.W. Australia 2065
Telephone (02)-439-2566 Telex: 790-20740

NEW ZEALAND
Digital Equipment Corporation Ltd.
AUCKLAND
Hilton House, 439 Queen Street, Box 2471
Auckland, New Zealand
Telephone: 75533

JAPAN
Digital Equipment Corporation International
Kowa Building No. 16 — Annex, First Floor
9-20 Akasaka 1-Chome
Minato-Ku, Tokyo 107, Japan
Telephone: 586-2771 Telex: J-26426
Riker Trading Co., Ltd. (sales only)
Kozato-Kaikan Bldg.
No. 18-14 Nishishimbashi 1-Chome
Minato-Ku, Tokyo, Japan
Telephone: 5915246 Telex: 781-4208

PUERTO RICO
Digital Equipment Corporation De Puerto Rico
407 del Parque Street
Santurce, Puerto Rico 00912
Telephone (809)-723-8068/67 Telex: 385-9056

ARGENTINA
BUENOS AIRES
Cossin S.A.
Virrey del Pino, 4071, Buenos Aires
Telephone: 52-3185 Telex: 012-2284

BRAZIL
RIO DE JANEIRO — GB
Ambriex S.A.
Rua Ceará, 104, 2 e 3 andares ZC - 29
Rio De Janeiro — GB
Telephone: 284-7408/0481/7625

SAO PAULO
Ambriex S.A.
Rua Tupi, 535
Sao Paulo — SP
Telephone: 52-7808/1870, 51-0912

PORTO ALEGRE — RS
Rua Coronel Vicente 421/101
Porto Alegre — RS
Telephone: 24-7411

CHILE
SANTIAGO
Cossin Chile Ltda. (sales only)
Casilla 14569, Correo 15,
Telephone: 386713 Cable: COACHIL

INDIA
BOMBAY
Hinditron Computers Pvt. Ltd.
69/A, L. Jagmohandas Marg.
Bombay-6 (WB) India
Telephone: 38-1615, 36-5344 Telex: 011-2584 Plenty
Cable: TEKHIND

MEXICO
MEXICO CITY
Mexitek, S.A.
Eugenia 408 Depto. 1
Apdo. Postal 12-1012
Mexico 12, D.F.
Telephone: (905) 538-09-10

PHILIPPINES
MANILA
Stanford Computer Corporation
P.O. Box 1608
416 Desmarinas St., Manila
Telephone 49-68-96 Telex: 742-0352

VENEZUELA
CARACAS
Cossin, C.A.
Apartado 50939
Sabana Grande No. 1, Caracas 105
Telephone: 72-8682, 72-9837
Cable: INSTRUVEN

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.)

Areas Covered	SPR Center
Australia/New Zealand	Digital Equipment Australia Pty. Ltd. 123-125 Willoughby Road, P.O. Box 491 Crows Nest New South Wales, Australia 2065
Brazil	Digital Equipment Comercio E Industria LTDA Rua Batatais, 429 (Esq. Al. Campinas) 01423-Jardim Paulista São Paulo-SP-Brazil
Canada	Digital Equipment of Canada, Ltd. Software Services P.O. Box 11500, K2H 8K8 Ottawa, Ontario, Canada
Caribbean	Digital Equipment Latin America, Inc. 407 del Parque Street Santurce, Puerto Rico 00912
United States, Far East, Middle East, Africa, Remainder of Latin America	Software Communications P.O. Box F Maynard, MA 01754
France	Digital Equipment France 18, rue Saarinen Centre Silic - CIDEX L225 F-94533 Rungis, France
Israel	DEC-sys Computers Ltd. 7 Habakuk Street IL-Tel Aviv 63505, Israel
Italy	Digital Equipment S.P.A. Corso Garibaldi 49 I-20121 Milano, Italy
Japan	Digital Equipment Corp. Int. Kowa Building #25 (3rd Floor) 8-7 Sunban-Cho Chiyoda-ku, Tokyo 102, Japan
Mexico	Equipo Digital, S.A. de C.V. 109 Concepcion Beistegui Mexico 12, D.F.
The Netherlands Belgium	Digital Equipment B.V. Kaap Hoorndreef 38, P.O. Box 9064 NL-Utrecht - Overvecht, The Netherlands
Scandinavia	Digital Equipment AB Englundavägen 7 S-17141 Solna Sweden
Switzerland Spain Portugal Greece Bulgaria Romania Yugoslavia	Digital Equipment Corp. SA 20, Quai Ernest Ansermet Case Postale 23, CH-1211 Geneva 8 Switzerland
United Kingdom	Digital Equipment Co. Ltd. Fountain House, Butts Centre GB-Reading RG1 7QN, England
West Germany Austria East Germany Russia Hungary Poland Czechoslovakia	Digital Equipment GmbH D-8000 München 40 Wallensteinplatz 2 West Germany

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
Software Distribution Center
146 Main Street
Maynard, Massachusetts 01754

Digital Equipment Corporation
Software Distribution Center
1400 Terra Bella
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
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

DECUS
Digital Equipment, S.A.
P.O. Box 340
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

BOSTON AREA:

Digital Equipment Corporation
Educational Services Department
(PDP-1Ø & 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)89705111 Ext. 3819
or 2564

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

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 Hoorndrief 38
Utrecht, Holland
Telephone: 030-63 12 22

SWEDEN:

Digital Equipment AB
Englundavaegen 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 Departme
Corso Garibaldi 49
1-20121 Milan, Italy
Telephone: 87-90-51

AUSTRALIA:

Digital Equipment Australia
Pty. Ltd.
Educational Services Departm
123-135 Willoughby Road
Crows Nest
New South Wales 2065, Australia
Telephone: 439-2566

JAPAN:

Digital Equipment Corporatio
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

digital

DIGITAL EQUIPMENT CORPORATION, Maynard, Massachusetts, Telephone: (617) 897-5111 • ARIZONA, Phoenix • CALIFORNIA, Sunnyvale, Santa Ana, Los Angeles, San Diego and San Francisco (Mountain View) • COLORADO, Engelwood • CONNECTICUT, Meriden • DISTRICT OF COLUMBIA, Washington (Riverdale, Md.) • FLORIDA, Orlando • GEORGIA, Atlanta • ILLINOIS, Northbrook • INDIANA, Indianapolis • LOUISIANA, Metairie • MARYLAND, Riverdale • MASSACHUSETTS, Cambridge and Waltham • MICHIGAN, Ann Arbor and Detroit (Southfield) • MINNESOTA, Minneapolis • MISSOURI, Kansas City and Maryland Heights • NEW JERSEY, Fairfield, Metuchen and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Huntington Station, Manhattan, New York, Syracuse and Rochester • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland, Dayton and Euclid • OKLAHOMA, Tulsa • OREGON, Portland • PENNSYLVANIA, Bluebell, Paoli and Pittsburgh • TENNESSEE, Knoxville • TEXAS, Dallas and Houston • UTAH, Salt Lake City • WASHINGTON, Bellevue • WISCONSIN, Milwaukee • ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Crows Nest, Melbourne, Norwood, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BRAZIL, Rio de Janeiro, Sao Paulo and Porto Alegre • CANADA, Alberta, Vancouver, British Columbia; Hamilton, Mississauga and Ottawa, Ontario; and Quebec • CHILE, Santiago • DENMARK, Copenhagen and Hellerup • FINLAND, Helsinki • FRANCE, Grenoble and Rungis • GERMANY, Cologne, Hannover, Frankfurt, Munich and Stuttgart • INDIA, Bombay • ISRAEL, Tel Aviv • ITALY, Milano • JAPAN, Osaka and Tokyo • MEXICO, Mexico City • NETHERLANDS, The Hague • NEW ZEALAND, Auckland • NORWAY, Oslo • PHILIPPINES, Manila • PUERTO RICO, Miramar and Santurce • REPUBLIC OF CHINA, Taiwan • SCOTLAND, West Lothian • SPAIN, Barcelona and Madrid • SWEDEN, Solna and Stockholm • SWITZERLAND, Geneva and Zurich • UNITED KINGDOM, Birmingham, Bristol, Edinburgh, London, Manchester, Reading and Warwickshire • VENEZUELA, Caracas

PRINTED IN USA 0302 00174 3534/E 14 1.2