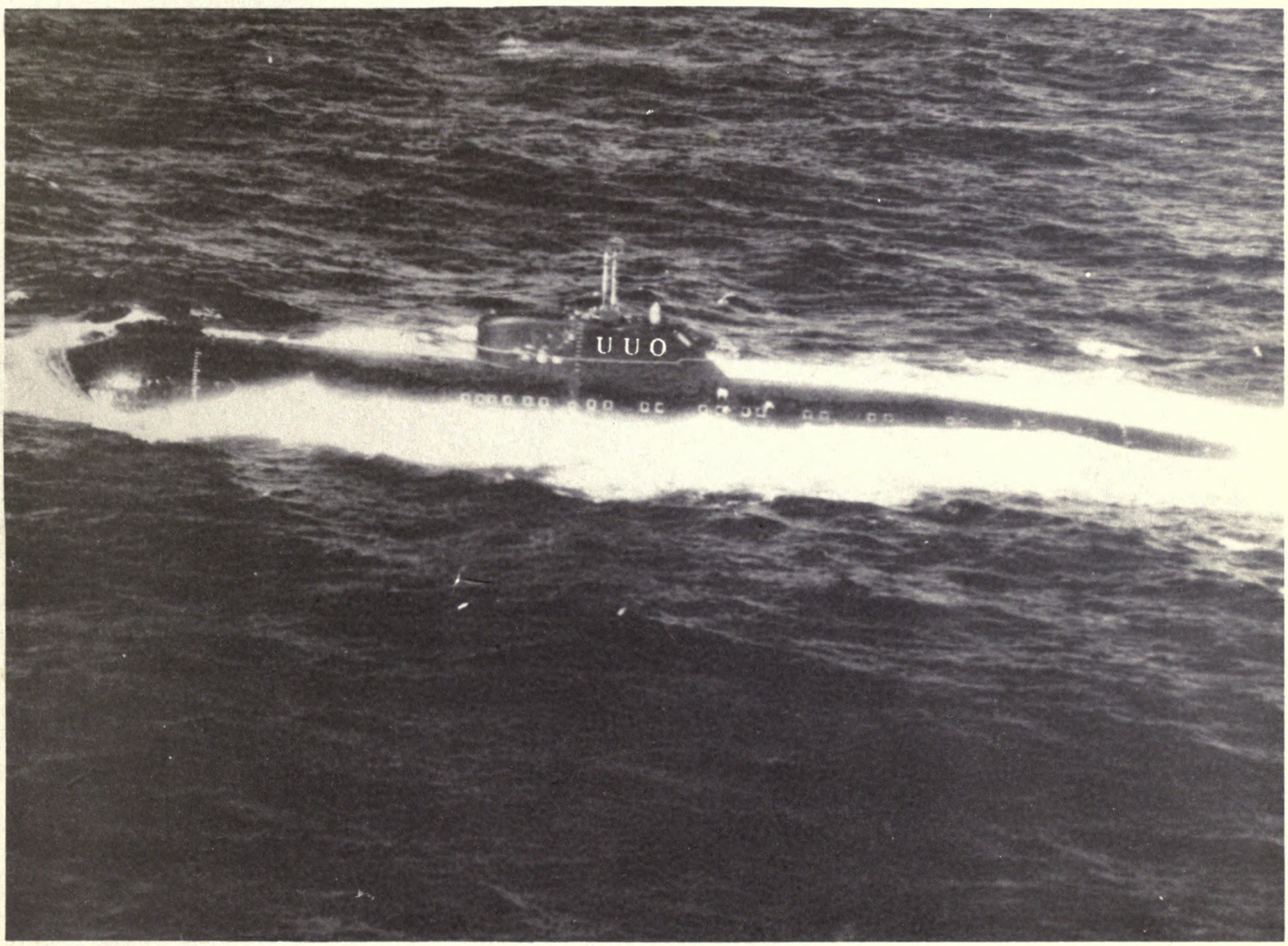

UUO

STANFORD ARTIFICIAL INTELLIGENCE LABORATORY
OPERATING NOTE 55.3

December 1973

UUO MANUAL

by

Martin Frost

ABSTRACT

This document describes the UUOs (monitor calls) available to users of the Stanford Artificial Intelligence Laboratory timesharing system. Additional general information relevant to the use of the UUOs is contained in the introductory section, and some useful tables are included in the appendices. This manual supersedes SAILON 55.2 by Andy Moorer (*Monitor Manual, Chapter II*).

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense under contract DAHC15-73-C-0435.

ACKNOWLEDGEMENTS

The author wishes to thank the following people who contributed significantly to this manual: Ralph Gorin for Section 13.4 on the XGP, Ted Panofsky for Appendix 1 and Appendix 2 on the III and Data Disc display systems (both appendices taken from Panofsky's *Facility Manual*, SAILON 56), and Andy Moorer for Section 13.12 on the IMP. Further thanks go to Brian Harvey for saving me the trouble of writing the new *Monitor Command Manual* (SAILON 54.3) and for working with me in getting our two manuals PUBed; to Dick Helliwell and Fred Wright, whose contributions to the system helped make this manual necessary; and finally to Larry Tesler for his PUB program and its many new features without which this manual could not have been printed so well on the XGP.

TABLE OF CONTENTS

APPENDICES

SECTION 1

INTRODUCTION

This document describes the UUOs (monitor calls) available to users of the Stanford Artificial Intelligence Laboratory timesharing system. Additional general information relevant to the use of UUOs is contained in this introductory section, and some useful tables are included in the appendices. This manual supersedes SAILON 55.2 by Andy Moorer *(Monitor Manual, Chapter 11)*.

The reader is assumed to know the PDP-10 instruction set and format, data types and assembly languages. However, the aspects of these subjects that are relevant to this manual are explained in Appendix 3. The user who is new to the PDP-10 should read that appendix before going any further. The experienced user may skip to Section 1.6, *Understanding this Manual.*

1.1 UUOs (Un-Used Operation codes)

UUOs are monitor calls which make use of instruction codes that would otherwise be unused or illegal. The opcodes from 000 to 077 are not used by any machine instruction, and opcodes from 700 to 777 are input/output machine instructions, which are normally illegal in user programs. All these opcodes trap to the monitor, which can then take whatever action it deems appropriate. Taking advantage of this situation, the system designates some of these opcodes to be monitor calls for certain common functions such as I/O. Thus whenever a UUO is encountered in the instruction stream, the monitor is called to execute the function corresponding to the particular UUO. When the function has been executed, control returns to the user program. Some UUOs may take skip returns; that is, control does not always return at the instruction immediately following the UUO, but sometimes at one of the next instructions after that one. The individual writeups explain when a UUO skips; unless otherwise described, a UUO's return is always at the instruction immediately following the UUO.

Some UUOs take arguments or return values in memory cells. In such cases the cells can be accumulators (ACs), but a block of such cells must not extend beyond the last accumulator (octal 17) because words 20 through 37 in a user's core image are used by the system for special temporary storage of sets of ACs. (Words 40 through 137 are used by the system to store information about the job. This part of a core image is referred to as the Job Data Area; the data stored here is described in Appendix 4.)

Note also that some UUOs have unused argument fields. Such a field should be made zero so that if at some later time it becomes used for a new feature, an old program using that UUO will still work.

Some of the opcodes not defined by the system are available to the user for defining his own special purpose UUOs. The method for defining these UUOs is explained in Section 1.4. The categories of opcodes that are used for UUOs are:

```
000     always illegal,
001:037 user-definable UUOs,
040:077 system-defined regular UUOs,
700:777 system-defined IOT UUOs.
```

The IOT UUOS are available only when the program is *not* in IOT-USER mode; in IOT-USER mode these opcodes are machine I/O instructions instead. A user program will not be in IOT-USER mode unless it has done something special to get into that mode. For a complete explanation of IOT-USER mode, see Appendix 3.

Finally, a special feature allows the user to have normal system-defined UUOs trap to a given location in the user program instead of being executed by the system. For details of this feature, see the UUOSIM UUO on page 131.

1.2 Extended UUOs

In order to define more UUOs than there are opcodes available, two primary methods are employed that allow a single opcode to represent many different UUO functions. The first of these methods is to use the value of the accumulator (AC) field in the instruction to specify one of 20 (octal) possible UUOs for a given opcode. Thus, for example, the OUTSTR UUO (which types out an ASCIZ string on the terminal) is invoked by specifying the opcode 051 and the AC field 3. There are currently six opcodes that use the value of the AC field in this manner. Each of these opcodes has a generic mnemonic which, together with a specific value for the AC field, can be used to indicate a specific UUO. In addition, each combination of generic mnemonic and specific AC field has a specific mnemonic which also can be used to indicate the UUO. Opcode 051 has the generic mnemonic TTYUUO, and TTYUUO with an AC field of 3 has the specific mnemonic OUTSTR. Thus the following three lines of code are equivalent. (ADR is an argument of this UUO; it specifies the address of the ASCIZ string to be typed out.)

```
OUTSTR ADR
TTYUUO 3,ADR
051140,,ADR
```

Note, however, that not all of the mnemonics are known by all of the assemblers or all of the debuggers. FAIL, however, gets its UUO mnemonic definitions directly from the system and thus is always up to date, even just after a new UUO has been added.

1.3 CALLs and CALLIs

The second method of defining many UUOs with the same opcode has two versions. In one of these, the address field of the UUO points to a word which contains the sixbit name of the UUO function desired. In the other version, the address field of the UUO is itself the number of the function desired. The opcode in the first case is 040 and its mnemonic is CALL; the opcode in the second case is 047 and its mnemonic is CALLI (for CALL Immediate).

CALL [OP=040]

```
        CALL AC, [SIXBIT /<name>/]
```

CALLI [OP=047]

```
        CALLI AC, <number>
```

Exactly the same UUO functions are available through these two methods. Thus, the following two lines of code are functionally equivalent; each will cause execution of the EXIT UUO.

```
        CALLI 12
        CALL  [SIXBIT /EXIT/]
```

Since there are these two versions of calling the same UUOs, the following fact should be noted. When you use a CALL instead of a CALLI, not only do you need an extra word in which to store the name of the CALL function, but also (and more importantly) you force the system to look up the function name in a table in order to find out the function number. This means that using CALLs instead of CALLIs creates a substantial amount of extra work that could be completely avoided. In addition, it is easier to use CALLIs instead of CALLs because, in FAIL and MACRO, if you use the name of a CALL as an opcode, the appropriate CALLI will be generated. (In MACRO, only the DEC CALLIs are predefined.) For example, the following two lines will produce the same machine code.

```
        CALLI 12
        EXIT
```

Thus the CALL UUO is essentially obsolete; it is mentioned here mainly for completeness sake. Please use CALLIs!

1.4 UUO Trapping and User-Defined UUOs

The method employed by the PDP-10 to trap to the monitor when an unused opcode is encountered is the following:

1. The effective address calculation for the instruction is carried out as usual with the address field, index field and indirect bit in the instruction and in any words referenced indirectly by the instruction.
2. Bits 0 to 12 (opcode and AC field) of the instruction are deposited in bits 0 to 12 of user or monitor location 40 (depending on whether the opcode represents a user or a monitor UUO). The calculated effective address from 1 above is deposited into bits 18 to 35 (address field) of the same location 40. Bits 13 to 17 (index field and indirect bit) of this location are cleared.
3. The instruction at location 41 (user or monitor as above) is then executed (as if from an XCT instruction). This location usually contains a JSR instruction to jump to a

subroutine to interpret the UUO. The JSR saves the program counter (which points to the instruction immediately following the UUO) for returning to the program containing the UUO.

Thus, for a user to define his own UUOs (selected from opcodes 001 to 037), he need only deposit a JSR or similar instruction in user location 41 to jump to the subroutine that will interpret his user UUOs. The instruction in location 41 should be one that saves the program counter for returning. For instance it could be a PUSHJ if you have a stack.

Note: Because the effective address calculation has already been completed by the time a UUO's function is executed, what the monitor (or a user's UUO code) sees in the address field of a UUO is this effective address. In the UUO writeups in this manual, the two expressions *the effective address of the UUO* and *the address field of the UUO* mean the same thing, namely, this final value of the effective address calculation.

(In the Stanford system, UUOs trapping into the monitor actually go to absolute 140 and 141 rather than 40 and 41 although user UUOs do trap to user 40 and 41.)

1.5 DEC vs. Stanford UUOs

UUOs with opcodes 040 through 077 and CALLIs with numbers 0 through 44 are essentially standard DEC UUOs modified for use at Stanford. (Some have been modified completely out of existence.) The exceptions are the TTYUUOs (opcode 051) with AC fields 14 through 17 and the SPCWAR UUO (opcode 043), which are special Stanford UUOs. All of the IOT UUOs (opcodes over 700) and all CALLIs with numbers from 400000 up are also special Stanford UUOs.

1.6 Understanding this Manual

In each of the sections that follow, a collection of related UUOs is explained along with the system concepts involved. Preceding the writeup for each UUO are 1) a line containing the UUO's mnemonics and numerical codes and 2) a sample usage (calling procedure) to which the writeup will often refer. For numerical codes, the abbreviation OP stands for the operation code field, AC for the accumulator field and ADR for the effective address of the UUO. For CALL/CALLI UUOs, the numerics will be those of the CALLI.

The phrases *AC left* and *AC right* mean, respectively, *the left half of AC* and *the right half of AC*.

Wherever there is a data block of length N used or set up by a UUO, the words of the block will be referred to as *word 0* through *word N-1* or sometimes (usually with short blocks) as *the first word* through *the Nth word*. Please note the difference between these two terminologies.

A range of bits or words will often be referenced by an expression of the form "X:Y", where X and Y are numbers. This represents all values from X through Y. For example, "bits 18:26" means bits 18 through 26.

All numbers will be in OCTAL except for the following, which will be in DECIMAL: bit numbers (e.g., "bit 35"), byte sizes (e.g., "12-bit bytes"), times (e.g., "30 seconds"), and numbers preceded by an equals sign (e.g., "=15").

References to particular bits or groups of bits will usually be made both by the bit numbers and by the octal value resulting from 1's in the specified bit positions. For example:

```
...if bit 0 (400000,,0 bit) is on...
...and bits 18:26 (0,,777000 bits)...
```

The octal value will be in half-word format, as shown above.

SECTION 2

GENERAL INPUT/OUTPUT

The purpose of input/output (I/O) is to transfer data between the computer's memory and an external device such as a tape, a disk, a printer, etc. The UUOs are set up to allow I/O with a fair amount of flexibility and device independence. I/O here is done on a very low level and involves three basic phases: initialization of the device, transfer of the data, and releasing of the device. Another phase, file selection, is necessary for the disk and other directory devices.

There are other simpler forms of I/O for certain devices (including terminals); I/O for those devices is explained in later sections.

The basic phases of I/O can be seen in the corresponding UUOs, so I will give an example sequence of the UUOs used in I/O. Since much of the I/O done by programs uses the disk, I will include the file selection phase in the example below. Note that this example is not a complete program; it contains only some excerpts involving the use of UUOs. This is intended to introduce you to various I/O concepts which will be explained in great detail in the remainder of this section.

```
INIT    1,10    ;This initializes the disk (DSK) on channel 1 in
SIXBIT  /DSK/   ; mode 10 and specifies an output buffer header at
OBUF,,0         ; location OBUF.  Upon an error in the execution
HALT    .       ; of this UUO, the program will HALT.

ENTER   1,FILE  ;This opens the file specified at location FILE for
HALT    .       ; output on channel 1; this will HALT on any error.

OUTPUT  1,      ;This is used to write out data on channel 1.

CLOSE   1,      ;This closes the file open on channel 1.

RELEAS  1,      ;This releases the device on channel 1.
```

In general I/O, the methods for doing output are very similar to those for doing input. Consequently, the following discussion will describe the basics of input; minor differences for doing output will usually be mentioned in parenthetical remarks. Any significant differences will be called to your attention.

2.1 User I/O Channels

A program is allowed to use up to 20 I/O devices at the same time. In order to keep straight which device an I/O UUO is meant for, each device in use is assigned a *channel number*. The channel number, which can be any number between 0 and 17 inclusive, is specified by the user when he initializes the device. Subsequent operations involving that device refer only to the channel number and not to the name of the device. The channel number is chosen by the user and has

significance only to the program in which it is assigned. Furthermore, when a device is released, the channel number is disassociated from it; so if the device is to be used again, it must be initialized again with a new (possibly the same) channel number.

A single I/O channel may not be used for both input and output at the same time, except on the disk in the special Read-Alter (RA) mode, which is explained on page 24.

2.2 Data Modes

When you are doing I/O, you must select the data mode to be used. The mode indicates how the data is to be transferred: primarily, whether the data transfers are to be buffered and, if so, whether the data is made up of characters or of full words.

In buffered mode, the system transfers data between the device and some buffers in your core area. A transfer of this type is initiated for input by the INPUT UUO and for output by the OUTPUT UUO. To get data from a buffer on input or to put data into a buffer for output, you simply do ILDBs (input) or IDPBs (output) with a byte pointer that is set up by the system. The data is thus handled a byte at a time, with the bytes being either characters (7 bits each) or full words (36 bits each); the mode determines the byte size. When a buffer is used up, you give an INPUT UUO to get another buffer of data (or an OUTPUT UUO to write out a buffer of data). The buffers you use are set up by the system in the form of buffer rings. Buffer rings will be described in detail in Section 2.4, but now back to data modes.

The alternative to buffered mode is dump mode. To read (write) in dump mode, you tell the system where in your core image you want the data to go (come from) and how many words are to be transferred. This is done with dump mode command lists, which will be explained in Section 2.3.

The basic data modes are listed in the following table and described below. (There are many special modes for specific devices; these modes are described in Section 13, which deals with device-dependent features.)

Mode	*Name*	*Type of transfers*
0	ASCII	Buffered characters (7-bit byte pointer)
1	ASCII LINE	Buffered characters (7-bit byte pointer)
10	IMAGE	Buffered words (36-bit byte pointer)
13	IMAGE BINARY	Buffered words (36-bit byte pointer)
14	BINARY	Buffered words (36-bit byte pointer)
16	DUMP RECORD	Unbuffered
17	DUMP	Unbuffered

ASCII mode (mode 0) is used for inputting (outputting) text. You get (put) one ascii character at a time from (into) your buffer by using the byte pointer. (Note: For TTY input in ASCII mode, an INPUT UUO will not return until your TTY buffer is full (holding =95 characters) or ↑Z (control-meta-linefeed on the displays) is typed. However, TTY I/O is usually done with the special UUOs described in Section 3.)

ASCII LINE mode (1) is the same as ASCII mode except for TTY input. There it means that an INPUT UUO will return when an activation character is typed or when the buffer is full, whichever comes first. (Normally the activation characters are carriage return, linefeed, altmode and control characters.)

IMAGE mode (10) is similar to ASCII mode except that the bytes are 36 bits instead of 7. When you do an ILDB (IDPB) you get (put) a whole word from (into) the buffer. You can read text files in this mode, in which case you will get 5 characters at a time.

IMAGE BINARY (13) and BINARY (14) modes are the same as IMAGE mode except for the paper tape reader and punch and the XGP. See Section 13.4 for the use of mode 13 with the XGP, and see Section 13.7 and Section 13.8 for the meanings of these modes with paper tape I/O.

DUMP mode (17) is used to do I/O without buffering. With each INPUT or OUTPUT UUO, you must give the address of a dump mode command list, which specifies how many words are to be transferred and where in your core image they are to come from or go. The INPUT or OUTPUT UUO does not return until the transfer is complete.. Dump mode command lists are explained below.

DUMP RECORD mode (16) is the same as DUMP mode for most devices. However, with output on magnetic tapes in DUMP RECORD mode, the data from each dump mode command is written on the tape in 200 word records, whereas in DUMP mode the data from each command is written in one large record.

2.3 Dump Mode Command Lists

The effective address of an INPUT or OUTPUT UUO in dump mode (16 or 17) must be the address of a dump mode command list. This list consists of up to 100 dump mode commands, each of which takes one word. The end of the list is indicated by a zero word after the last command.

In the left half of a dump mode command word is the negative of the number of words to be transferred, and in the right half is the address of the word *before* the first word in your core image to which (from which) the data is to go (come).

(This is the standard dump mode command format. Some devices like the TV cameras and the AD converter take non-standard dump mode commands. See the individual device descriptions in Section 13.)

In the assembly languages, there is a pseudo-op called IOWD that generates dump mode commands. The line

```
IOWD N,LOC
```

generates a word with -N in the left half and LOC-1 in the right half. As a dump mode command, this will cause N words to be transferred to (from) the block consisting of locations LOC through LOC+N-1.

Each dump mode command in a list causes a separate I/O transfer to take place. In particular, with record devices (like the disk, dectapes and magnetic tapes) each command is executed from the beginning of a record. For example the command list

```
IOWD 100,A
IOWD 100,B
0
```

will read 100 words from one record and then 100 words from the next record, *ignoring all but the first 100 words of each record.*

Consecutive commands in a list are usually fetched from consecutive words. However, if the *left* half of a dump mode command word is zero, then that word is taken *not* as a command, but as a *pointer* to the next command word, which is then fetched from the location specified by the *right* half of that word. (Of course, if the right half is zero also, then that word marks the end of the command list.) For example, the two command lists given below (beginning at LIST1 and LIST2, respectively) are equivalent.

```
LIST1:  IOWD 200,LOC1        |     LIST2:  IOWD 200,LOC1
        IOWD 200,LOC2        |             LIST2B
        0                    |             ...
                             |
                             |     LIST2B: IOWD 200,LOC2
                             |             0
```

The only difference is that in the second example, the commands are in two different places instead of being directly in line. Either of these lists would cause 200 words to be transferred to (from) LOC1 and then 200 words to (from) LOC2.

2.4 Buffer Rings

When you are doing input (output) in one of the buffered modes, a ring of buffers is set up by the system for storage of data in your core image. This allows you to empty (fill) one buffer while the device is filling (emptying) another buffer independently.

A buffer ring consists of some number of buffers with each one containing a pointer to the next. The last buffer contains a pointer to the first; hence it is a ring. The user can specify the number of buffers in a ring with the INBUF and OUTBUF UUOs (see Section 2.9), or he can accept the system default number of buffers, which is two. The current buffer in a ring is referenced through a three word block called the *buffer header*. When you initialize a device in buffered mode, you give the address of a three word block where the buffer header is. The system will initialize the header when it sets up the buffer ring.

The buffer header contains 1) a bit indicating whether the buffer has ever been used, 2) a pointer to the buffer the user is currently emptying (filling), 3) the byte pointer that is used for loading (storing) data from (into) the current buffer, and 4) a count of the number of bytes left in the current buffer. Normally only the byte pointer and the byte count are needed by the user. The

byte pointer is in the second word of the header and the byte count in the third. The first word contains the use indicator in the sign bit (400000,,0 bit) (which is set to 1 when the buffers are created and cleared to zero when the first INPUT or OUTPUT UUO is given) and the buffer pointer in the right half. For input, the byte count is the number of bytes of data left in the buffer; for output it is the number of bytes not yet filled with data by the user. The byte pointer is set up so that you can get (put) the next byte by doing an ILDB (IDPB). When you initialize a device in buffered mode, the system clears the buffer header and then sets up the size field of the byte pointer. If you so desire, you may then change the byte size to any size you want, and the system will do the right things. (The system actually uses the byte size in the byte pointer to calculate the byte count.) Finally, whenever you do an INPUT or OUTPUT UUO, the system updates the entire buffer header before returning control to you.

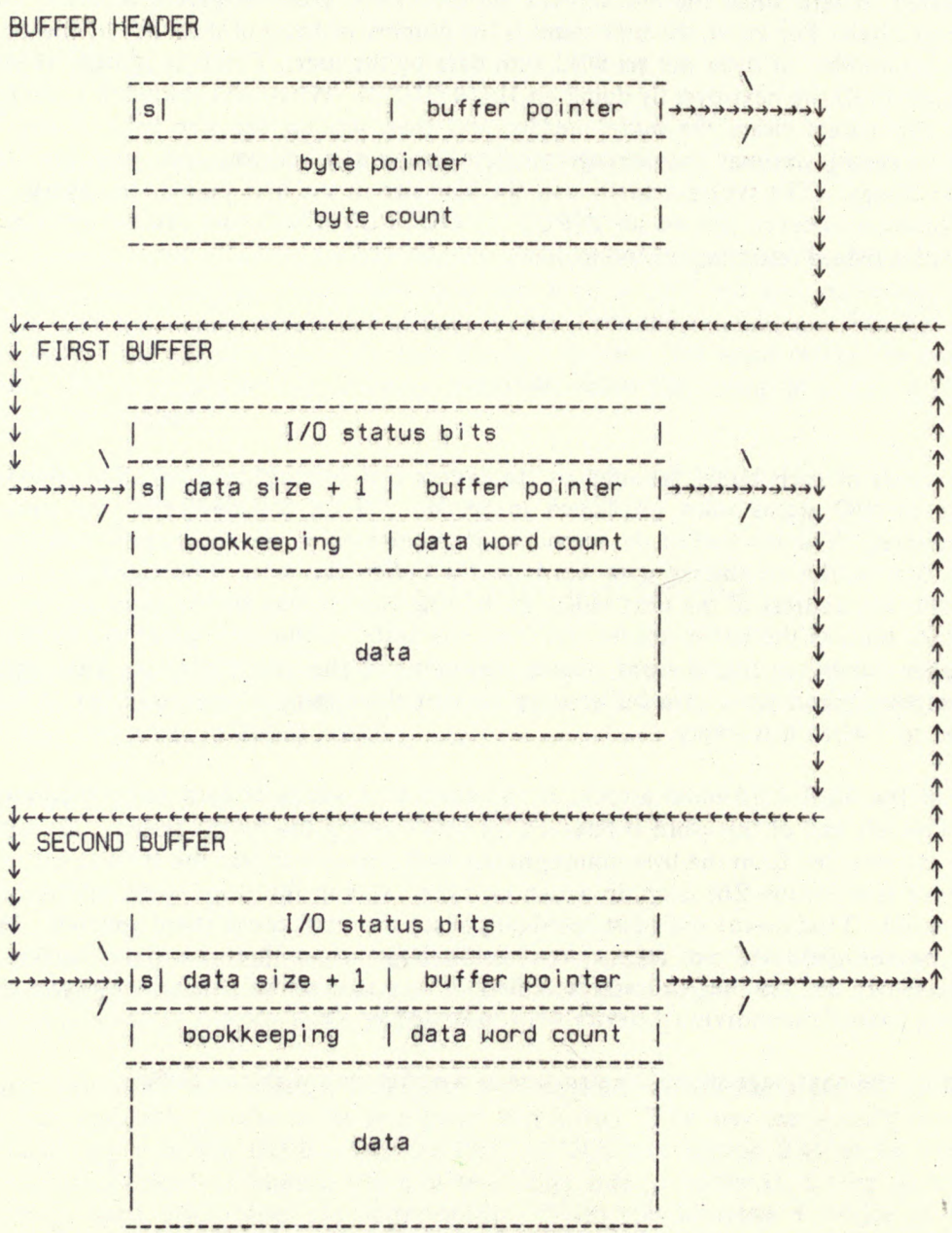
2.5 Buffers

The first three words of each buffer are used by the system and contain no data. The first word contains the device I/O status word (explained in Section 2.6) for the device at the time the buffer was transferred. The left half of the second word has the size of the buffer not counting the first two words, that is, the number of data words in the buffer plus one. The right half of the second word holds the address of the next buffer in the ring (which may be the same buffer). All pointers to buffers, both in the buffer header and from one buffer to the next, point not to the first word of the buffer but to the second word, where the pointer to the next buffer is. The sign bit (400000,,0 bit) of the second word in a buffer is set to 1 by the system when the buffer is full of data and cleared to 0 when it is empty.

The right half of the third word holds a count of the number of words of data actually contained in the buffer. The left half of this word is reserved for bookkeeping use by the system. On output the word count is computed from the byte pointer in the buffer header unless the IOWC bit in the device status word (see Section 2.6) is on, in which case the value in the third word of the buffer is taken as the count. That means you must specifically place the word count there yourself. Many devices (including the disk) will not take a word count larger than their standard buffer size. There are at least two devices that will accept buffers of any size: terminals and magnetic tapes. For other devices, consult the individual device writeup in Section 13.

The illustration on the next page shows the structure of a buffer ring with two buffers.

DIAGRAM OF A 2-BUFFER RING



2.6 Device I/O Status Word

For each device on the system there is a word called the device I/O status word. The left half of this word is used internally by the system and the right half is used to communicate with the user. The right half word is set up when the user initializes the device. Thereafter, certain bits may be set by the system to tell the user of conditions that have arisen (such as end of file). These bits can be tested or changed by the user with the STATZ, STATO, GETSTS and SETSTS UUOs (see Section 2.14). The following table gives the meanings of the bits in the right half of the device status word.

Bits	*Octal*	*Name*	*Meanings of 1's in device status word*
18	0,,400000	IOIMPM	Improper mode. This can mean many things. If you attempt to write on a write-locked dectape or UDP, you get this error bit. If you initialize a device in a mode that is not legal for that device, you will get either this error bit or a system error message.
19	0,,200000	IODERR	Device detected error.
20	0,,100000	IODTER	Device detected error. This bit and the IODERR bit generally mean that the device has detected an undesirable, if not catastrophic, condition.
21	0,,40000	IOBKTL	Dectape block number out of bounds. This bit comes on if you reference a non-existent block on a dectape.
22	0,,20000	IODEND	End of file. This bit comes on only with input; it means there is no more data to be read in because you have reached the end of the file. On teletypes and pseudo-teletypes, end of file is indicated by typing control-Z (↑Z); on Data Disc and III displays, end of file is indicated by typing control-meta-linefeed.
23	0,,10000	IOACT	Device is active.
24:29	0,,7700		Reserved for device-dependent features.
30	0,,40	IOCON	Synchronize buffered I/O. An attempt is made to make buffered I/O synchronous when this bit is on. That is, exactly one buffer is transmitted for every INPUT

			UUO and at most one buffer for each OUTPUT UUO. The best way to insure synchronous I/O, however, is to have exactly one buffer in your ring, or better yet, to use dump mode.
31	0,,20	IOWC	Inhibit system computation of output word count. The system is inhibited from computing the word count that goes in the third word of each buffer. Normally, when you do an OUTPUT UUO in buffered mode, the system uses the byte pointer in the buffer header to compute the number of words of data in the buffer. This computed word count is then deposited in the third word of the buffer. If the IOWC bit is on, the word count computation is inhibited and whatever is in the third word of the buffer is taken as gospel for the word count.
32:35	0,,17		Data mode.

2.7 Files

Data on certain devices is stored in the form of files. To access data on one of these devices, you must specify the filename in addition to the device name. Devices that are file structured are called directory devices; the disk and dectapes are such devices.

Every file has a name of one to six characters and an optional extension of one to three characters. On the disk each file is associated with some project-programmer name (PPN). A project-programmer name consists of a project code and a programmer code, each of which is one to three characters. All of the files for a particular project-programmer name are referred to collectively as that PPN's disk area.

File names, file name extensions and project-programmer names are stored in *sixbit* representation. File names are left-justified in one whole word; file name extensions are left-justified in the left half of a word. If a filename has no extension, the extension half word is zero. A zero file name is not permitted. Project-programmer names are stored in one word with the project code in the left half and the programmer code in the right half, each half being right-justified. For an example of all this, the file RAD.Y[A,BC] would be represented by the following octal values and corresponding assembly language lines of code (the date word is included to make this four-word block into a sample block for the LOOKUP UUO):

	Octal	*Assembly language*
file name:	624144000000	SIXBIT /RAD/
extension:	710000000000	SIXBIT /Y/
date word:	000000000000	0
PPN:	000041004243	SIXBIT / A BC/

where 62 is the octal value for "R" in sixbit, 41 is the octal value for "A" in sixbit, etc.

Disk File Protection System

Each file on the disk has a nine-bit protection key that indicates who may do what to the file. A one in the first bit (400 bit) position means that the file dumping program DART (that provides file backup on magnetic tape) should never dump this file. A one in the second bit (200 bit) means that COPY should not delete this file without getting special confirmation; this prevents accidental deletion of a file with the monitor DELETE command. The remaining seven bits (177 bits) are broken into three groups: the third bit (100 bit) tells what the owner of the file may do to the file, the middle three bits (070 bits) tell what a PPN consisting of the owner's programmer code and a different project code may do with the file, and the last three bits (007 bits) tell what anyone else may do. Corresponding bits in these last two groups mean the same thing but for the two different classes of users. The sole bit in the first group means the same as the third bit in each of the other groups but applies to the owner of the file.

In each group, a one in the first bit position (the 4 bit) means that the users corresponding to that group are not permitted to change the file's protection key. This is called *protection protection.* A user is always permitted to change the protection keys of his own files. A one in the second bit position (the 2 bit) means that the corresponding users may not read the file. This is called *read protection*, and, again, a user may always read his own files. A one in the third bit position (the 1 bit) means that the corresponding users may not write, alter or delete the file. This is called *write protection.*

Here is a summary of the nine protection bits.

Bits	*Octal*	*Meaning*
0	400	Dump never (DART only).
1	200	Delete protect (COPY only).
2	100	Write protection for the file's owner.
3:5	070	Protection, read, and write protection for PPNs consisting of the owner's programmer code and some other project code.
6:8	007	Protection, read, and write protection for others.

Disk Project-Programmer Names

When you reference a file on the disk, you must specify the project-programmer name of the file's owner. If the file is your own, this can be done by indicating a PPN word of zero. Sometimes, however, you would like a program to act as if it were logged in under a different PPN. This can be accomplished with respect to file references through the use of Disk PPNs and the DSKPPN UUO. Each job has associated with it a Disk Project-Programmer Name (the Disk PPN) that is used whenever the PPN word for a disk file specification contains zero. Your Disk PPN is set to your real PPN when you log in and can be changed with the monitor ALIAS command; it can also be changed or retrieved with the DSKPPN UUO. Thus if you specify a file with a zero PPN, the project-programmer name of your Disk PPN will be assumed for the file. This method does not, however, allow you violate any protection keys for the files you reference. These protection keys are applied to your real (logged in) PPN to see if you are permitted the kind of access you are requesting for the file.

DSKPPN [OP=047, ADR=400071] CALLI 400071

```
        MOVE   AC,[<code>]
        DSKPPN AC,
```

Code	Meaning
0	Return own Disk PPN in AC.
-1	Reset own Disk PPN to logged in PPN.
0,,n	Return the Disk PPN of job n.
<proj>,,0	Set own Disk PPN to: <proj>,,<logged in prog. name>.
<proj>,,<prog>	Set own Disk PPN to that in AC.

The DSKPPN UUO is used to change or retrieve your Disk Project-Programmer Name or to retrieve the Disk PPN of someone else. The action taken by this UUO is determined by the code in AC. If AC contains zero, your current Disk PPN is returned in AC. If AC contains -1, your Disk PPN is reset to your logged in PPN. If AC contains a job number, the Disk PPN for that job is returned; if the job is not logged in, zero is returned. If the right half of AC contains zero and the left half is non-zero, then the *project* part of your Disk PPN is set to the project specified by AC left and the *programmer* part is set to the programmer code under which you are logged in. If both halves of AC are non-zero and AC doesn't contain -1, then your Disk PPN is set to the PPN specified by the whole AC. No error checking is done to make sure AC holds a legal or existing PPN.

2.8 Initializing a Device

There are two UUOs available to initialize a device: the INIT UUO and the OPEN UUO. Either of these UUOs can be used; the only difference between them is the format for passing parameters to the system.

Each of these UUOs tells the system what device you want to use, what mode you want to use it in, where your buffer headers are, if any, and what channel number you want to associate with the device. If the device is a non-sharable device (such as the line printer, a terminal, a dectape or a magnetic tape) which is not available now, the system will normally type out a message asking if you will wait for it. However, the following bits in the data mode half word which you specify when you attempt to initialize the device can be used to indicate special action to be taken.

Bits	*Octal*	*Meanings of 1's in the initial data mode*
26	0„1000	Wait automatically until the device is available.
27	0„400	Take error return automatically if the device is busy. This bit takes precedence over bit 26.

If you want to have your program wait automatically without your being asked, you should have bit 26 (the 1000 bit) on in the data mode half word. If you would like to get the error return automatically when a device is busy, you should have bit 27 (the 400 bit) on in the data mode. The automatic error return bit takes precedence over the automatic wait bit. Note that these two bits (26 and 27) are among those reserved for device-dependent features. Thus, if you have either of these bits on when you initialize a device, you should use the SETSTS UUO (explained in Section 2.14) to turn them off after you get the device unless you want the particular features they represent for that device. See the device writeups in Section 13 for the meanings of these bits for the individual devices.

A device can be referred to by either its *physical* name or its *logical* name. The physical name of a device is the permanent name given that device by the system. A logical device name is a temporary name that can be specified with the monitor ASSIGN command. Device names (physical or logical) are stored left-justified in sixbit representation. For example, the device DTA1 is represented by the octal number 446441210000, which can be set up by the SIXBIT pseudo-op in the assembly languages, i.e., SIXBIT /DTA1/. If a given device name is both a physical name (of one device) and a logical name (of another device), the logical name takes precedence.

INIT [OP=041]

```
        INIT <channel number>,<data mode>
        <physical or logical device name in sixbit>
        OBUF,,IBUF
        <error return>
```

The INIT UUO initializes the named device on the channel indicated by the AC field and in the data mode specified by the address field of the instruction. OBUF should be the address for your output buffer header or zero if none. IBUF should be the the address for your input buffer header or zero if none. If you are not going to do any buffered output on this channel, OBUF can be zero. If you are not going to do any buffered input on this channel, IBUF can be zero.

The data mode half word is used to set the right half of the device I/O status word for this device. The IOACT bit, however, is masked out when this is done. (The device status word is explained in Section 2.6.)

When you initialize a device, any device open on the channel specified is released before the new device is initialized. (See the RELEAS UUO in Section 2.12.)

If there is no device with the name you give, the error return (double skip) is taken. If this UUO is successful, the triple skip return is taken.

OPEN [OP=050]

```
        OPEN <channel number>,ADR
        <error return>

ADR:    <data mode>
        <device name in sixbit>
        OBUF,,IBUF
```

The OPEN UUO does exactly the same thing as the INIT UUO above. The difference is that the information passed to the system by OPEN does not have to appear in line with the instruction stream. The location of this information is specified by the effective address of the UUO. Hence OPEN can be used by reentrant programs that change the data referenced by the UUO.

The left half of the data mode word is ignored.

2.9 Setting Up Buffer Rings

For buffered I/O, a buffer ring must be set up in your core area. Unless you issue an explicit buffer-creating UUO (described below), or make the buffer ring yourself (very carefully!), the system will set up a buffer ring for you when you first give an INPUT or OUTPUT UUO. That is, if a device open in buffered mode has no associated input (output) buffer ring when the first INPUT (OUTPUT) UUO is given, the system will set up a two-buffer ring before carrying out the normal function of the UUO. To do buffered input (output), you must have given the address of the input (output) buffer header when you initialized the device.

Whenever the system sets up a buffer ring for you, it places the ring at the address contained in JOBFF in your job data area (see Appendix 4). You may cause your buffers to be set up anywhere in your core image by temporarily changing JOBFF to point to the place where you want the buffers to be. If there is not enough room between JOBFF and JOBREL for the number of buffers you request, your core image is automatically expanded to make room. After the ring is set up, JOBFF is left pointing to the first word beyond the ring and your buffer header is made to point to the first buffer in the ring.

The following UUOs cause a buffer ring to be set up, and they permit specification of a non-standard number of buffers and non-standard sizes.

INBUF [OP=064]

```
        INBUF <channel number>,<number of buffers>
```

The INBUF UUO causes an input buffer ring to be set up in your core area and to be associated with the specified channel. The effective address of this UUO is interpreted as the number of buffers the ring is to have. If the effective address is zero, two buffers are set up (the same number of buffers you would get if you did not give this UUO at all).

OUTBUF [OP=065]

```
        OUTBUF <channel number>,<number of buffers>
```

The OUTBUF UUO causes a ring of output buffers to be set up exactly as INBUF does for input buffers.

UINBF [OP=704]

```
        UINBF <channel number>,ADR

ADR:    <number of buffers>
        <number of words of data in each buffer> + 1
```

The UINBF UUO causes an input buffer ring to be set up in your core area and to be associated with the specified channel. The effective address points to a two word block. The first word of this block contains the number of buffers to be in the ring (zero means two), and the second word contains a number which is one greater than the number of words of data in each buffer.

Some devices (including the disk) do not accept nonstandard buffer sizes. Devices that will accept nonstandard sizes include terminals and magnetic tapes. For other devices see the individual device descriptions in Section 13.

UOUTBF [OP=705]

```
        UOUTBF <channel number>,ADR

ADR:    <number of buffers>
        <number of words of data in each buffer> + 1
```

The UOUTBF UUO causes a ring of output buffers to be set up exactly as UINBF does for input buffers.

BUFLEN [OP=047, ADR=400042] CALL1 400042

```
        MOVE   AC,[<device name in sixbit, or channel number>]
        BUFLEN AC,
```

The BUFLEN UUO tells you the standard buffer size for the device specified by the contents of AC. AC should contain either the name (logical or physical) of the device or the number of the channel on which it is open. The buffer size, which is returned in AC, is one greater than the length of the data portion of a buffer that would be set up if you did an INIT and then an INBUF or OUTBUF for the particular device. This is the number you would need to use to set up a standard size buffer with a UINBF or a UOUTBF UUO. The total number of words each buffer would take up is two greater than this number (see Section 2.5). If there is no such device, zero is returned in AC.

2.10 Opening Files

After initialization of a directory device, a particular file on the device must be opened (i.e., selected) before any I/O can take place. Opening of a file for input is done with the LOOKUP UUO; opening of a file for output is done with the ENTER UUO. The RENAME UUO is available for changing a file's name or specifications (date written, protection, etc.) after the file has been opened. RENAME is also used to delete files.

If you initialize a directory device and attempt to transfer data with an INPUT (OUTPUT) UUO without having done a LOOKUP (ENTER), the system will type out a message and require you to type in a filename so that a LOOKUP (ENTER) can be done before the data is transferred.

For non-directory devices, the UUOs LOOKUP, ENTER and RENAME are no-ops; they always take the success (skip) return.

LOOKUP [OP=076]

```
        LOOKUP <channel number>,ADR
        <error return>

ADR:    <file name in sixbit>
        <file name extension in sixbit>
        <this word is ignored>
        <project-programmer name in sixbit>
```

The LOOKUP UUO opens for input the file specified by the four-word block pointed to by the effective address of the UUO. The first word of the block should contain the sixbit name of the file to be read; the second word should contain the sixbit file name extension in the left half. If the device is the disk, the fourth word of the block should contain the project-programmer name for the file or zero; zero will cause your current Disk PPN to be assumed for the file (see page 16). The right half of the file extension word is ignored as is the whole word following the extension word. For dectapes the project-programmer name is also ignored.

If the LOOKUP is successful, the skip return is taken and some information about the file is returned. If the file does not exist or if some other error condition arises, then the error return (no skip) is taken and, if the device is the disk, an error code is returned in the right half of ADR+1 (the rest of the block is unchanged). The error codes for LOOKUP, ENTER and RENAME are explained in a table on page 25. No error codes are returned for dectapes.

After a successful LOOKUP of a disk file, the following information is returned in the LOOKUP block. The word at ADR+2 contains: the file's protection in bits 0:8 (777000,,0 bits) the data mode in which the file was written in bits 9:12 (740,,0 bits), the file's time written in bits 13:23 (37,,770000 bits), and the file's date written in bits 24:35 (0,,7777 bits). These values are stored in the directory when the file is created and may be changed with the RENAME UUO (see page 23). (The date written is in system date format which is explained under the DATE UUO on page 85. The time written is in minutes past midnight on the date given. The protection bits are explained on page

15 and the data mode is explained in Section 2.2.) The word at ADR+3 contains the negative swapped word count, that is, the negative of the number of words in the file, with the left and right halves exchanged. (The word count is returned in this strange format to be compatible with DEC's format.) Finally, the right half of the word at ADR+1 contains, in system date format, the "creation date" of the file, which is a slightly less than well defined quantity. The date written in ADR+2 is a much more significant date.

Here is a summary of the information in the block after a successful LOOKUP on the disk.

```
ADR:    <file name>
ADR+1:  <file name extension>,,<"creation date">
ADR+2:  <Bits 0-8 (777000,,0 bits): protection;
         bits 9-12 (740,,0 bits): data mode;
         bits 13-23 (37,,770000 bits): time written;
         bits 24-35 (0,,7777 bits): date written>
ADR+3:  <negative swapped word count>
```

WARNING! You may not do two consecutive successful LOOKUPs for the disk with the same four-word block without restoring the project-programmer name in the fourth word of the block after the first LOOKUP! The negative swapped word count probably will not represent the PPN you want!

After a successful LOOKUP on a dectape, the following information will be found in the LOOKUP block:

```
ADR:    <file name>
ADR+1:  <extension>,,<number of first block of file>
ADR+2:  <date file written>
ADR+3:  <whatever was in 4th word of ENTER block when the file was created>
```

ENTER [OP=077]

```
        ENTER <channel number>,ADR
        <error return>

ADR:    <file name in sixbit>
        <file name extension in sixbit>,,<creation date>
        <protection key in bits 0:8 (777000,,0 bits)>
        <project-programmer name>
```

The ENTER UUO opens for output a new file with the specifications given in the four-word block pointed to by the effective address of the UUO as indicated above. For a disk file, the time and date written are set to the current time and date when the ENTER is done, and the file's protection is set from bits 0:8 of ADR+2. If the device is a dectape, the words at ADR+2 and ADR+3 are copied into the dectape's directory with the exception that if bits 24:35 (0,,7777 bits) of ADR+2 are zero, the current date is substituted for this 12-bit field before the word is written into the directory. For the disk, if the PPN is zero, your current Disk PPN is assumed. If the ENTER is successful, the skip return is taken and exactly the same information is returned in the block as after a successful LOOKUP (see above and note that for a new file the word count is zero). If there

already was a file with the given name, its creation date is returned; otherwise the specified creation date (zero means today) is returned. If the ENTER fails for any reason, then the no skip error return is taken and, if the device is the disk, a code is returned in the right half of ADR+1 (the rest of the block is unchanged). The error codes for LOOKUP, ENTER and RENAME with the disk are explained in a table on page 25. No error codes are returned for dectapes.

An ENTER can fail for a number of reasons. It will fail if the file name is zero, if the PPN (or Disk PPN) is illegal, if the file already exists and is write protected against you, if the file is already open for output (by anyone), if the device is a dectape which is already full, or if you have already done a LOOKUP on this channel and the filename for the ENTER does not agree with that given in the LOOKUP (see Read-Alter mode for the disk on page 24).

With a successful ENTER of a disk file, if the file specified already exists, then that file will be replaced with the new file *when the new file is closed.* Until that time, any attempt to read the specified file will access the old file. After the new file is closed, any attempt to read the specified file will get the new version; the old version will stay around only long enough for anyone still reading it to finish.

RENAME [OP=055]

```
        RENAME <channel number>,ADR
        <error return>

ADR:    <new file name or zero for deletion>
        <new file extension>,,<ignored>
        <new protection key, mode, time and date last written>
        <project-programmer name>
```

The RENAME UUO is used to change the name, extension, protection key, mode, or time and date written, or a combination of these, for a file, or to delete a file. This UUO *must* be given *after* a successful LOOKUP or ENTER has been done on this channel and *may* be given after a CLOSE UUO (see Section 2.12) for this channel. After an ENTER, a CLOSE is *necessary* before RENAME! As with LOOKUP and ENTER, if the project-programmer name is zero, your current Disk PPN is assumed.

If the file name specified is zero, and if the effective PPN matches the PPN of the file open on this channel, then that file is marked for deletion. This means that as soon as no one is reading the file, it will go away. (After a file has been marked for deletion, anyone attempting to start reading it will not find it.)

If the file name is not zero, then the name, extension and protection key for the file open on this channel are all changed to those specified in the four-word block. Also, if bits 9:35 at ADR+2 are not all zero, then the mode and time/date written of the file are set to those specified by these bits (9:12 are mode; 13:23 are time; 24:35 are date). Note that you cannot rename a file to a different disk area (different PPN); if the effective PPN is different from the file's original PPN, the file will be lost.

If the RENAME is successful, the skip return is taken. Otherwise, the no-skip error return is taken and (for the disk) an error code is returned in the right half of ADR+1, with the rest of the block left unchanged. The meanings of the possible disk error codes for LOOKUP, ENTER and RENAME are explained in a table on page 25.

Read-Alter Mode

There are two basic methods of updating data in a file. In the first, the file is copied, with appropriate changes, into a new file with the same name. This is accomplished by doing a LOOKUP of the old file on one channel and an independent ENTER of the same filename on a different channel. When the new version of the file is closed, the old version will be deleted (after all read references to it are finished). This method requires the whole file, however, to be read in and written out again even if only a little of the data in the file is to be changed. The second method allows you to open an already existing disk file and to change data in it *in place*, without rewriting the whole file. This method of file manipulation is known as READ-ALTER (RA) mode. When you have a file open in this mode, you may do (on the same channel) both input and output with the file. To open a file in this mode, you do a LOOKUP of the file and then an ENTER of the same file on the same channel. If both the LOOKUP and the ENTER are successful, then the file will be open in RA mode. If you give a different filename for the ENTER than you used with the LOOKUP, the ENTER will fail with an error code of 6 (see table below). In RA mode, at the moment any data is written out, that data overwrites whatever was there before. So if the file does not get closed thereafter, the new data will still have replaced the old data in the file. Data can be written into selected parts of a file by use of the random access UUOs USETI, USETO and UGETF (see Section 2.13). While a file is open in RA mode, anyone attempting to do either a LOOKUP or an ENTER of that file will get the FILE BUSY error return (code 3, see below). Also, an attempt to open a file in RA mode (by doing an ENTER after a successful LOOKUP) will also fail with the FILE BUSY error return if anyone else is reading or writing the file.

Disk Error Codes for LOOKUP, ENTER and RENAME

Code	*Meaning*
0	NO SUCH FILE. LOOKUP: File specified does not exist. ENTER: Zero file name given. RENAME: File LOOKUPed or ENTERed no longer exists.
1	ILLEGAL PPN. PPN specified has no UFD.
2	PROTECTION VIOLATION. File is protected from what you tried to do.
3	FILE BUSY. LOOKUP: File is currently open in Read-Alter mode. ENTER: File is currently being written. ENTER after LOOKUP: File is currently being read or written. RENAME: File is currently being read.
4	FILE ALREADY EXISTS. (RENAME only) RENAME: There is already a file with the new name given.
5	NO FILE OPEN. (RENAME only) RENAME: No successful LOOKUP or ENTER has been done yet.
6	DIFFERENT FILENAME SPECIFIED. (ENTER only) ENTER: The filename does not match that of a LOOKUP already done on this channel (attempt to open file in Read-Alter mode).
7	(This error code cannot occur.)
10	BAD RETRIEVAL. Some disk pointers have been garbaged somewhere. This should not happen.
11	BAD RETRIEVAL. Slightly different version of above.
12	DISK IS FULL. (ENTER only) ENTER: There is no more room on the disk.

Note: Errors 10, 11 and 12 will cause a system error message to be typed out unless bit 28 (the 0,,200 bit) is on in the device status word (see Section 2.6). When one of these error messages is typed out, the program will be stopped; if you then type CONTINUE, the error return will be taken with the appropriate error code given.

2.11 Transferring Data

The following UUOs are used to transfer data between your core image and a device, which must already have been initialized on some channel (see Section 2.8). If you give one of these UUOs for a device open in buffered mode with no buffer ring set up, a two-ring buffer will be set up for you before the normal action of the UUO is taken (see Section 2.9).

IN [OP=056]

```
        IN <channel number>,ADR
        <success return>
        <error return>
```

The IN UUO causes some data to be read in to your core image from the device open on the given channel. In buffered mode, at least one buffer will be filled with input data and the buffer header will be updated so that the byte pointer and byte count are correct for the newly filled buffer. In dump mode, ADR is taken to be the address of a dump mode command list (see Section 2.3), and the UUO will not return until all the data indicated by the command list has been transferred. In buffered mode, ADR is ignored.

If any error (including end of file) occurs, then the UUO skips. Specifically, when the UUO is to return, if any of the error bits IOBKTL, IODTER, IODERR, IOIMPM or IODEND (see Section 2.6) are on, the skip return is taken. Otherwise, the direct return (no skip) is taken.

In dump mode, if end of file occurs before the command list has been satisfied, then IODEND will come on and the UUO will skip, but there will be no way of telling how much data, if any, was read in before end of file occurred. In buffered mode, there is always a byte count that tells how much data has been read in.

On teletypes and pseudo-teletypes, end of file is indicated by typing control-Z (↑Z); on Data Disc and III displays, end of file is indicated by typing control-meta-linefeed.

INPUT [OP=066]

```
        INPUT <channel number>,ADR
```

The INPUT UUO does exactly the same thing as the IN UUO except that no error checking is done and the UUO never skips.

OUT [OP=057]

```
        OUT <channel number>,ADR
        <success return>
        <error return>
```

The OUT UUO causes some data to be written out from your core image to the device open on the given channel. In buffered mode, the buffer pointer, byte pointer and byte count in the buffer header are set up for the next buffer that you may fill and the device is started up to empty the buffer you just filled. The first OUT UUO you give in buffered mode, however, does not cause any data to be written out, only the buffer header to be set up with the buffer pointer, byte pointer and byte count for the first buffer for you to fill.

In dump mode ADR is taken as the address of a dump mode command list (see Section 2.3) that indicates what data are to be written out; the UUO does not return until the transfer is complete.

In buffered mode, if ADR is non-zero, this UUO does *not* write out your current buffer but instead switches you to the new buffer ring pointed to by ADR. (ADR should be the address of the second word of a buffer in the ring.) Your buffer header and some internal system data are adjusted so that you will next be filling the first buffer in the new ring. The buffer ring you are switching to must be completely set up with the buffer-to-buffer pointers and the buffer sizes in the second word of each buffer. The purpose of this feature is to let you switch among several output buffer rings if you so desire. Note, however, that when you switch rings there is no provision for forcing data still in buffers in the old ring to be written out.

As with the IN UUO, if any of the error bits IOBKTL, IODTER, IODERR, IOIMPM or IODEND (see Section 2.6) are on at completion of the UUO, the UUO skips. Otherwise, the direct return (no skip) is taken.

OUTPUT [OP=067]

```
        OUTPUT <channel number>,ADR
```

The OUTPUT UUO does exactly the same thing as the OUT UUO except that no error checking is done and the UUO never skips.

WAIT [OP=047, ADR=10] CALLI 10

```
        WAIT <channel number>,
```

The WAIT UUO simply waits for all I/O on the channel indicated by the AC field to finish. Normally, when a device is open in buffered mode, the system does I/O with your buffers while your program is running. This means that only the buffer pointed to by your buffer header can be

expected to be remain untouched by the system. After giving this UUO, you can expect all of your buffers to be stable and untouched by the system.

2.12 Terminating I/O

The following two UUOs are used to finish up I/O on a given channel. The CLOSE UUO essentially undoes the effect of a LOOKUP or ENTER, and the RELEAS UUO undoes the effect of an INIT or OPEN.

CLOSE [OP=070]

```
        CLOSE <channel number>,<close-inhibit flags>

(<close-inhibit flags>:
        1 (bit 35) inhibits closing output,
        2 (bit 34) inhibits closing input.)
```

The CLOSE UUO is used to terminate I/O on the channel specified by by AC field of the UUO. The effective address of the instruction determines whether input or output or both or neither is closed. If the low order bit (bit 35--the 0,,1 bit) of the effective address is on, then the closing of output is inhibited. If bit 34 (0,,2 bit) of the effective address is on, the closing of input is inhibited. The remaining bits in the effective address are ignored.

On non-directory devices like terminals and paper tape, this UUO forces out any data still in any output buffers. For magnetic tape, closing output causes two end-of-file marks to be written on the tape and causes the drive to backspace over one of them; this means that there will be one end-of-file mark between each pair of files and two end-of-file marks after the last file on the tape. The two consecutive end-of-file marks denote what is called the logical (as opposed to physical) end of tape.

On the disk and dectape, closing output forces out any data still in any output buffers and then closes the file; closing input simply closes the file. After a disk or dectape file is closed, no more data may be transferred to or from it. However, a file may be RENAMEd even after it is CLOSEd.

RELEAS [OP=071]

```
        RELEAS <channel number>,<close-inhibit flags>

(<close-inhibit flags>:
        1 (bit 35) inhibits closing output,
        2 (bit 34) inhibits closing input.)
```

The RELEAS UUO does a CLOSE of the given channel with the given <close-inhibit flags> specified by the effective address (see the CLOSE UUO above) and then frees the channel number. Thus, after giving this UUO, you must do another INIT or OPEN to do any more I/O on this channel.

The RESET UUO (see page 126) simulates a RELEAS <channel>,3 for every channel you have open. The normal EXIT UUO (that is, "EXIT 0,") simulates a RELEAS <channel>,0 for every channel you have open (see page 125).

REASSI [OP=047, ADR=21] CALL1 21

```
        MOVE   AC,[<job number>]
        MOVE   AC+1,[<device name in sixbit, or channel number>]
        REASSI AC,
```

The REASSI UUO is used to turn over a device you are using to another job. Accumulator AC should contain the number of the job to whom you wish to give the device, and accumulator AC+1 should contain the logical or physical name of the device or the channel on which it is open. This UUO gives the same results as the following sequence: 1) you release the device with *both* input and output inhibiting, 2) you deassign the device with the monitor DEASSIGN command, and 3) the job indicated assigns the device with the monitor ASSIGN command.

If the job number you give is not that of a logged in job, then this UUO will return with accumulator AC set to zero. If the device is not assigned to your job, or if the device may not be reassigned at this time, then the UUO will return with accumulator AC+1 set to zero (which might be confusing if you specified channel zero in AC+1).

2.13 Random Access to Files

A disk or dectape file consists of a series of 200 word records. Often, these records are read (or written) sequentially from the beginning of the file to the end. But sometimes one wishes to read or alter only selected parts of a file. Three random access UUOs are provided to allow the user to do exactly that. To do random access input or output, you must specify which record you want to reference next. On the disk, the records of a file are numbered consecutively from 1 to n, where

the file is n records long. (*Hidden* records can precede logical record 1 of a disk file. The hidden records have non-positive logical record numbers. See the disk file record offset feature on page 143.) On dectapes the records of a file are physical blocks and are numbered differently; for a precise explanation of dectape files, read Section 13.5.

For each disk file open, the system maintains a pointer to the record that will be referenced by the next INPUT or OUTPUT UUO. For each dectape file open, the system maintains two pointers, one for input and one for output. The following three UUOs set these pointers to specific values. If you try to set the record pointer for a disk file to a value less than that of the first physical record in the file, you get the first physical record instead. If you try to set it to a record beyond the end of a disk file, you get the first record after the last record in the file (and IODEND in the device status word is turned on; see Section 2.6). If you try to set it to a value greater than the number of the last physical block on a dectape, IOBKTL will be turned on in the device status word; thus the error return will be taken with the next IN or OUT UUO.

USETI [OP=074]

```
USETI <channel number>,<record number>
```

The USETI UUO prepares you to read from a file at a specific record. You must have a file open for input on the channel indicated by the AC field. The record pointer for the file (input block pointer for dectape files) is set to the value in the address field of the instruction and the status of any input buffers is set to unused. The IODEND bit in the device status word is cleared unless you have selected a record beyond the end of a disk file.

Note that the record number in a USETI for the disk is taken as a signed 18-bit number in twos-complement notation; see the record offset feature on page 143.

USETO [OP=075]

```
USETO <channel number>,<record number>
```

The USETO UUO prepares you for writing into a file at a specific record. You must have a file open for output on the channel indicated by the AC field. This UUO forces out the data in any output buffers that have not yet been written and then sets the record pointer for the file (output block pointer for dectape files) to the value in the address field of the instruction. The status of any output buffers is set to unused.

Note that the record number in a USETO for the disk is taken as a signed 18-bit number in twos-complement notation; see the record offset feature on page 143.

UGETF [OP=073]

```
UGETF <channel number>,ADR
```

The UGETF UUO prepares you to extend the file open on the given channel. It forces out the data in any output buffers that have not yet been written. Then, for the disk, the record pointer is set to the number of the record after the last record of the file and IODEND is turned on. For dectapes, the output block pointer is set to the number of the next free block you may write on. In either case, the number of the record (or block) so selected is returned in the word pointed to by the effective address of the UUO. The status of any input or output buffers is set to unused.

2.14 I/O Status Testing and Setting

There are various pieces of information one can find out about an I/O device or about a logical I/O channel, namely: the I/O device status bits (explained in detail in Section 2.6), the channel use bits (explained below), the device characteristics (explained below), the physical name of a device, and the number of people waiting to use a given device. This section describes the UUOs used to get and/or set these.

GETSTS [OP=062]

```
GETSTS <channel number>,ADR
```

The GETSTS UUO puts the I/O device status word for the device open on the indicated channel into the word located at ADR. See Section 2.6.

SETSTS [OP=060]

```
SETSTS <channel number>,<status bits>
```

The SETSTS UUO waits for the device open on this channel to become inactive and then sets the right half of the I/O device status word for this device from the address field of the UUO. See Section 2.6.

STATZ [OP=063]

```
        STATZ <channel number>,<status bits to be tested>
        <return if any of the indicated bits are on>
        <return if all indicated bits are off>
```

The STATZ UUO tests certain bits in the right half of the status word for the device open on the indicated channel. The address field of the UUO should contain 1's in the bit positions to be tested. If all of the tested bits are off (zero), then this UUO skips. If any of them are on, the direct return is taken.

STATO [OP=061]

```
        STATO <channel number>,<status bits to be tested>
        <return if all indicated bits are off>
        <return if any of the indicated bits are on>
```

The STATO UUO tests bits in the right half of the status word just like STATZ does but skips if any of the tested bits are on and does not skip if all of them are off.

CHNSTS [OP=716]

```
        CHNSTS <channel number>,ADR
```

The CHNSTS UUO puts the use bits for the I/O channel indicated into the word located at ADR. Here is a list of the meanings of these bits.

Bits	*Octal*	*Name*	*Meanings of 1's in channel usage word*
18	0,,400000	INITB	The channel has been initialized with an INIT or an OPEN and a RELEAS has not been given yet.
19	0,,200000	IBUFB	The INIT or OPEN which initialized this channel specified an input buffer header address.
20	0,,100000	OBUFB	The INIT or OPEN which initialized this channel specified an output buffer header address.
21	0,,40000	LOOKB	A LOOKUP (successful or not) has been done on this channel.
22	0,,20000	ENTRB	An ENTER (successful or not) has been done on this channel.
23	0,,10000	INPB	An INPUT or IN UUO has been done on this channel.

24	0,,4000	OUTPB	An OUTPUT or OUT UUO has been done on this channel.
25	0,,2000	ICLOSB	The input side of this I/O channel has been closed.
26	0,,1000	OCLOSB	The output side of this I/O channel has been closed.
27	0,,400	INBFB	An input buffer ring has been set up for this channel.
28	0,,200	OUTBFB	An output buffer ring has been set up for this channel.
29	0,,100	SYSDEV	The device open on this channel is SYS (i.e., disk area [1,3]).

DEVCHR [OP=047, ADR=4] CALLI 4

```
MOVE   AC,[<device name in sixbit, or channel number>]
DEVCHR AC,
```

The DEVCHR UUO returns the device characteristics word for the device specified by the contents of AC. AC should contain either the name (logical or physical) of the device or the number of the channel on which it is open. The characteristics are returned in the AC. If the device does not exist, a zero is returned. Here are the meanings of the bits in the returned word.

Bits	*Octal*	*Meanings of 1's in device characteristics word*
0	400000,,0	Dectape with directory in core.
1	200000,,0	Disk.
2	100000,,0	User disk pack (UDP).
3	40000,,0	Line printer (LPT) or XGP. The LPT and the XGP can be distinguished from each other by checking bit 8 (long dispatch table), which will be on for the XGP and off for the LPT.
4	20000,,0	Terminal which is attached to a job.
5	10000,,0	Terminal which is in use.
6	4000,,0	TV camera.
7	2000,,0	The IMP.
8	1000,,0	Long dispatch table. This means that the device will accept UUOs other than INPUT and OUTPUT, such as MTAPE, USETO and LOOKUP.
9	400,,0	Paper tape punch or plotter (which are logically the same device).
10	200,,0	Paper tape reader.
11	100,,0	Dectape.
12	40,,0	The device is available to the job that gave the DEVCHR UUO.
13	20,,0	Magnetic tape.
14	10,,0	Terminal.

15	4,,0	Directory device. At Stanford, this means the device is a dectape, the disk or a UDP.
16	2,,0	Input device.
17	1,,0	Output device.
18	0,,400000	ASSIGNed device.
19	0,,200000	Some job has done an INIT or OPEN on this device.
20:35	0,,177777	A one in bit 35-J means mode J is legal for this device.

DEVUSE [OP=047, ADR=400051] CALLI 400051

```
MOVE   AC,[<device name in sixbit, or channel number>]
DEVUSE AC,
```

The DEVUSE UUO can be used to find out what job, if any, is using a device and how many jobs are waiting for that device. The AC should contain the name (logical or physical) of the device or the channel number on which it is open. If there is no such device, a zero is returned in AC. Otherwise, the following information is returned in AC:

Bits	*Octal*	*Value*
0	400000,,0	One if the device has been ASSIGNed by monitor command.
1	200000,,0	One if a program has done an INIT or OPEN on this device.
2	100000,,0	One if the device is a TTY attached to a job.
12:17	77,,0	Number of the job the device is being used by, or zero if it is not in use. If the device has been detached from the system, then this field will contain zero but bit 0 (ASSIGNed device, see above) will be on.
18:35	0,,777777	The number of jobs (not including you) in the queue waiting for the device.

PNAME [OP=047, ADR=400007] CALLI 400007

```
        MOVE  AC,[<device name in sixbit, or channel number>]
        PNAME AC,
        <error return for no such device>
```

The PNAME UUO returns the physical name of the device specified by the contents of the AC. AC should contain either the device name (logical or physical) or the number of the channel on which the device is currently open. If the device exists, the skip return is taken and the device's physical name is returned in AC. If there is no such device, the direct (error) return is taken and the AC is unchanged.

DEVNUM [OP=047, ADR=4000104] CALLI 4000104

```
        MOVE   AC,[<device name in sixbit, or channel number>]
        DEVNUM AC,
        <error return for no such device>
```

The DEVNUM UUO is used to find out the unit number of a particular device. AC should contain either the logical or physical name of the device or the number of the channel on which the device is open. If there is a device with the name given or open on the channel given, then its unit number is returned in AC and the skip return is taken. If there is no such device, the direct (error) return is taken.

The unit number of a device specifies which of several logically identical devices a specific device is. For instance, the unit number of TTY41 is 41, the unit number of MTA1 is 1, etc.

SECTION 3

TTY INPUT/OUTPUT

The terminal is one of the most important I/O devices for the user. He controls his programs by typing in various commands, and the programs type back certain things to keep him informed. This section explains several UUOs that are provided to make terminal I/O control flexible but simple. The word TTY is used in this manual to mean a user terminal of any type, whether display, teletype or pseudo-teletype.

3.1 TTY Echoing and LF Insertion

The system provides two services to terminals doing input. The first is that characters typed in are normally sent back to the terminal in order for the user to see what he has typed. This is called echoing of input. The second action taken on input is that normally, when the system receives a carriage return from a TTY line, it inserts a linefeed after the carriage return. Thus the user does not have to type a linefeed (hereafter abbreviated LF) after each carriage return (hereafter abbreviated CR). The LF is put into the terminal's input buffer just as if the user had typed it; it is also usually echoed to the terminal.

These actions can be modified by the user to suit his particular purposes. Echoing can be turned off in two different manners. The first of these is intended for terminals that always print each character typed. If the system were to echo characters to this kind of terminal, each character would appear twice. This method of turning echoing off causes all echoing to be suppressed except for echoing of LFs inserted after CRs. Bit 15 (4,,0 bit) in the line characteristics word (see the GETLIN UUO on page 40) indicates the state of this type of echo suppression. This bit can be set and cleared by the monitor commands TTY NO ECHO and TTY ECHO, respectively.

The second type of echo suppression is designed to be used by programs that, for whatever reasons, do not want typed-in characters to appear on the terminal. This method turns off all echoing except when TTY input is going to the monitor rather than to the program. The state of this type of echo suppression is indicated by the NOECHO bit (bit 28--the 0,,200 bit) in the TTY I/O status word (see Section 2.6 and Section 2.14); when the NOECHO bit is on, echoing is suppressed. This bit can be turned on or off only by UUO, currently only by the CTLV UUO (see page 47), by the PTJOBX UUO (with the DOFF and DON functions--see page 56) and by the INIT and SETSTS UUOs. PTJOBX is the recommended UUO for this purpose. A RESET (see page 126) clears this bit, thus turning echoing back on. (A program can also disable just the echoing of the CONTROL and META bits; see the NOECHB bit in Section 13.2.)

Insertion of linefeeds after carriage returns is affected by three factors: 1) whether TTY input is going to the monitor or to a user program, 2) whether the terminal is a pseudo-teletype (see Section 3.5), a display or a teletype, and 3) the value of bit 16 (2,,0 bit) in the TTY line characteristics word (see the GETLIN UUO on page 40). LFs are always inserted after CRs on III and Data

Disc displays. For other TTYs in the normal case, LFs *are* inserted after CRs. They are *not* inserted if both 1) bit 16 in the line characteristics word is on and 2) either the terminal is a pseudo-teletype or input is going to a user program. Note that pseudo-teletypes (PTYs) are initialized with bit 16 on; thus LFs are normally *not* inserted after CRs on PTYs. Bit 16 in the characteristics word can be changed only by UUO, currently only by the SETLIN UUO (see page 42) and the PTSETL UUO (see page 54).

3.2 Codes Returned for Characters Typed

When a character is read from a TTY, the 7-bit Stanford ascii code for that character is returned in bits 29:35 (0,,177 bits) (see the Stanford ascii character set in Appendix 7); in addition, characters read from displays are returned with the CONTROL and META keys represented as bits 28 (0,,200--CONTROL) and 27 (0,,400--META). Furthermore, when control-Z is typed on a teletype and read by any of the UUOs described below, the value returned is 612, which is the code for control-meta-linefeed from displays. Thus the end-of-file character always appears as the same code regardless of the type of terminal on which it was typed. Note that when characters are being read from a TTY by means of the INPUT or IN UUOs described in Section 2, only 7 bits are returned for each character (the CONTROL and META keys on displays are lost), and the EOF characters (control-Z and control-meta-linefeed) do not appear as characters at all--they merely set the EOF bit in the TTY I/O status word.

3.3 TTYUUO

The most important UUO is probably TTYUUO (known some places as TTCALL). This is an extended UUO with many different functions. With a couple of exceptions, which are noted, these functions all operate on the terminal attached to the job giving this UUO.

TTYUUO [OP=051]

```
        TTYUUO <function number>,ADR
```

TTYUUO uses the accumulator field of the instruction to determine the particular function to be executed. Each of these functions is described separately below.

INCHRW [OP=051, AC=0] TTYUUO 0,

```
        INCHRW ADR
```

The INCHRW UUO waits for a character to be typed and then returns the character right-justified in the word at ADR.

OUTCHR [OP=051, AC=1] TTYUUO 1,

```
        OUTCHR ADR

ADR:    <ascii character>
```

The OUTCHR UUO types out the single ascii character represented by the right-most seven bits of the word at ADR.

INCHRS [OP=051, AC=2] TTYUUO 2,

```
        INCHRS ADR
        <return if no character has been typed>
        <success return>
```

The INCHRS UUO looks to see if a character has been typed. Then, if so, the character is returned in the word at ADR and the skip return is taken. If no character has been typed, the direct return is taken and the word at ADR is not changed.

OUTSTR [OP=051, AC=3] TTYUUO 3,

```
        OUTSTR ADR

ADR:    <asciz string>
```

The OUTSTR UUO types out the ASCIZ string that starts at location ADR. (An ASCIZ string is terminated by the first null (zero) byte.)

INCHWL [OP=051, AC=4] TTYUUO 4,

```
        INCHWL ADR
```

The INCHWL UUO waits until an entire line (ended by carriage return, linefeed, altmode or a control character) has been typed and then returns a single character right-justified in ADR. This is called *line mode* and should be used instead of *character mode* (as in INCHRW) whenever possible. In character mode you cannot always backspace over mistyped characters because your program may already have eaten them up; in line mode you can backup as far as the last activation character.

INCHSL [OP=051, AC=5] TTYUUO 5,

```
        INCHSL ADR
        <return if no entire line has been typed yet>
        <success return>
```

The INCHSL UUO looks to see if an entire line has been typed, and if so, returns one character right-justified in ADR and takes the skip return. If an entire line has not yet been typed, the direct return is taken and ADR is not changed.

GETLIN [OP=051, AC=6] TTYUUO 6,

```
        GETLIN ADR
```

The GETLIN UUO can be used to find out what terminal a job is attached to, if any, and what the characteristics are for any terminal. If the original contents of ADR are less than zero, then the characteristics for your own terminal are returned in ADR. If ADR originally contains the number of a TTY line (a number between zero and the maximum legal TTY line number), the characteristics of that terminal are returned in ADR. If ADR originally contains a number greater than the maximum legal TTY line number, then zero is returned in ADR.

If a job requests the line characteristics for its own terminal, and if that job is detached, that is, not attached to any terminal, then a -1 (all bits on) will be returned as the line characteristics. You should check for this condition before testing any of the particular bits or you will be deceived by a detached job.

If the characteristics word is not -1, then the right half contains the line number of the terminal and the left half contains the characteristics of the terminal, as explained below.

Note: If the terminal is a pseudo-teletype (see Section 3.5) controlled directly or indirectly (through a chain of pseudo-teletypes) by a Data Disc or III display, then the Data Disc bit (bit 4--the 20000,,0 bit) or the III bit (bit 0--the 400000,,0 bit) will be on in the characteristics for the pseudo-teletype.

Bits	*Octal*	*Meanings of 1's in TTY line characteristics word*
0	400000,,0	The terminal is a III display.
1	200000,,0	The terminal is the PDP-10 console teletype (CTY).
2	100000,,0	Carriage returns are made into multiple carriage returns in order to allow the teletype carriage to reach the left margin before the next character reaches the teletype. This bit can be set and cleared with the monitor commands TTY FILL and TTY NO FILL, respectively, and with the SETLIN and PTSETL UUOs (see below and page 54). Data Disc and III displays are not affected by this bit.
4	20000,,0	The terminal is a Data Disc display.
5	10000,,0	The terminal is a model 37 teletype.
6	4000,,0	The terminal is a pseudo-teletype (see Section 3.5).
7	2000,,0	The terminal is an IMLAC.
8	1000,,0	The terminal is a pseudo-teletype and is controlled by a job connected to the IMP. This means the PTY's job is being run through the ARPA network. This bit can be set and cleared with the SETLIN and PTSETL UUOs.
9	400,,0	Pseudo-teletype input wait will be terminated by TTY input also (see the PTRDIW UUO on page 52). This is the PTYWAKE bit; it can be set and cleared with the SETLIN and PTSETL UUOs.
11	100,,0	The terminal is in special activation mode. This means that line mode input will be activated by the characters whose bits are 1's in the special activation table. This bit can be set and cleared with the SETLIN and PTSETL UUOs. A RESET (see page 126) clears this bit, as well as resetting your special activation table to the standard special activation table. See the SETACT UUO on page 44.
12	40,,0	The last character typed was a rubout, and a backslash will be typed out when a character that is not a rubout is typed.
13	20,,0	The terminal is in full-character-set mode. When this bit is off, lower case letters are automatically changed

to upper case. This bit can be set and cleared with the monitor commands TTY FULL and TTY NO FULL, respectively, and with the SETLIN and PTSETL UUOs. On displays this bit can be set by [ESC] F and cleared by [BRK] F.

14	10,,0	The terminal is assumed not to have a hardware tabbing mechanism. When this bit is on, tabs get converted into the appropriate number of spaces. This bit can be cleared and set by the monitor commands TTY TAB and TTY NO TAB, respectively, and with the SETLIN and PTSETL UUOs. Data Disc and III displays are not affected by this bit.
15	4,,0	Echoing of input characters is inhibited except for linefeeds inserted by the system after carriage returns. This inhibition is provided primarily to avoid double echoing on terminals that always print each character typed. See Section 3.1. This bit can be set and cleared by the monitor commands TTY ECHO and TTY NO ECHO, respectively, and with the SETLIN and PTSETL UUOs. Data Disc and III displays are not affected by this bit.
16	2,,0	Linefeeds will not be inserted after carriage returns, except in monitor mode. This bit always determines whether linefeeds are to be inserted after carriage returns on a pseudo-teletype (PTY), regardless of whether the PTY is in monitor mode. See Section 3.1. On Data Disc and III displays, this bit does not inhibit insertion of LFs; it merely inhibits echoing of inserted LFs. This bit can be set and cleared with the SETLIN and PTSETL UUOs. A RESET (see page 126) clears this bit except on PTYs.

SETLIN [OP=051, AC=7] TTYUUO 7,

```
        SETLIN ADR

ADR:    <line characteristics bits which you want on>
```

The SETLIN UUO allows you to change certain bits in the line characteristics for your terminal. You are allowed to change only bits 2, 8, 9, 11, 13, 14, 15 and 16 (101536,,0 bits). These bits in your line characteristics word are set to their values in the word at ADR. All other bits in the word at ADR are ignored.

RESCAN [OP=051, AC=10] TTYUUO 10,

```
        RESCAN ADR              ; ADR is ignored if it is zero

ADR:    <word for returned character count>
```

The RESCAN UUO attempts to back up your TTY input buffer pointer to the beginning of the previous monitor command typed in. By using this UUO, a program started up by a monitor command can re-read the command line that started it. In fact, this UUO can be given over and over to read the command line several times. If this UUO is given with a non-zero effective address, then the number of characters over which the pointer is backed up is returned in the word pointed to by the effective address.

Warning: If more than a buffer full of characters have been typed since the beginning of the last monitor command, then the characters you get after giving this UUO will *not* be from the command. The pointer into the buffer will simply have been set to the value it had at the beginning of the command; the command itself may have been overwritten by other text typed in more recently, in which case you will be reading garbage after giving this UUO.

CLRBFI [OP=051, AC=11] TTYUUO 11,

```
        CLRBFI
```

The CLRBFI UUO clears your TTY input buffer. This is used mainly to throw away any characters the user has typed ahead when a fatal error occurs.

CLRBFO [OP=051, AC=12] TTYUUO 12,

```
        CLRBFO
```

The CLRBFO UUO clears your TTY output buffer.

INSKIP [OP=051, AC=13] TTYUUO 13,

```
        INSKIP  <flag>
        <return if no characters have been typed>
        <success return>
```

The INSKIP UUO tells you if the user has typed anything which you have not yet read. If the low order bit of the address field <flag> is on, then this UUO checks for a whole line having been typed; otherwise it checks for anything having been typed. If something has been typed, then this UUO skips; if not, the direct return is taken.

INWAIT [OP=051, AC=14] TTYUUO 14,

```
        INWAIT ADR
```

The INWAIT UUO just waits until a full line has been typed in. Then if the address ADR is non-zero, the number of characters in the last line re-edited (III and Data Disc terminals only) with a control-CR or with a PTLOAD UUO (see page 55) is returned in the word at ADR. In other words, if you give a PTLOAD UUO and then do an INWAIT ADR, you will get in location ADR the number of characters in the re-edited line. If you are not at a III or Data Disc display, or if you did not re-edit a line somehow, the number placed in ADR will be meaningless.

SETACT [OP=051, AC=15] TTYUUO 15,

```
        SETACT [OLD,,NEW]

OLD:    <4 word block to receive the current activation table>

NEW:    <4 word block to provide a new activation table>
```

The SETACT UUO is used to retrieve and/or change the activation table used in special activation mode. An activation table consists of 4 words, with one bit for each character and a couple of extra bits thrown in for some special effects. The first three words plus the high-order 20 bits of the fourth word (total of 128 bits) specify which characters are activation characters (bit 0 of first word represents ascii 0, bit 1 represents ascii 1, etc.). The activation characters are defined to be those characters whose bits are 1 in the activation table. The meanings of the low order bits of the fourth word are given below:

Bits	*Octal*	*Meaning of 1's at end of fourth word*
35	0,,1	Suppress activation on characters with control bits except for those characters that would activate without any control bits. See also bit 33 below.
34	0,,2	Disable control-carriage-return from giving back the last line typed.
33	0,,4	Always activate on characters that have both CONTROL and META on, regardless of setting of bit 35 above.

This UUO takes the current activation table and places it in the four words at OLD, then sets up the new activation table from the four words at NEW. If either address OLD or NEW is zero, the corresponding function of this UUO is omitted. Thus if OLD is zero, the old activation table is not returned, and if NEW is zero, the (old) activation table is not changed.

Your special activation table is initialized by the system so that all characters except letters and

digits cause activation (when you are in special activation mode). The precise value of the initial special activation table is shown below.

```
777777,,777777          ;Activate on octal 0:43
777700,,037600          ;Activate on octal 44:57 or 72:100
000000,,374000          ;Activate on octal 133:140
000007,,600000          ;Activate on octal 173:177
```

TTREAD [OP=051, AC=16] TTYUUO 16,

```
        TTREAD ADR

ADR:    <line number>
```

The TTREAD UUO allows you to read the microswitch keyboard bits of any display's keyboard. The effective address in this instruction specifies a location which should contain the TTY line number of the keyboard you wish to read. If the line number specified is illegal, then your line number is used. Then if the line is not on the keyboard scanner (that is, if the line is not a III or Data Disc display line), then this UUO is a no-op.

The keyboard bits are returned in the right half of ADR. The line number minus 20 is returned in the left half of ADR.

OUTFIV [OP=051, AC=17] TTYUUO 17,

```
        OUTFIV  ADR

ADR:    <five-character ASCII string>
```

The OUTFIV UUO is designed for sending special commands to IMLACs. The effective address should point to a word containing a five-character ASCII string. Characters from the string are sent to the terminal until either a null byte is encountered or the fifth character in the string has been sent. If the low order bit (bit 35--the 0,,1 bit) of the ASCII word is on, a 177 character will be appended to the front of the string sent to the terminal.

The special feature of this UUO is that it insures that the characters (including the 177) will be sent to the terminal without any intervening characters. This prevents commands sent to IMLACs from getting mixed up with normal typeout.

3.4 Miscellaneous TTY UUOs

```
TTYMES          [OP=047, ADR=400047]  CALL1 400047
-------------------------------------------------
        MOVEI  AC,ADR
        TTYMES AC,
        <error return>

ADR:    <number or physical name of destination tty>
        ppcccc,,MESS  ;where ppcccc is a 6-digit octal number

MESS:   ASCII   /...message.../
```

The TTYMES UUO can be used to type out an ASCII string on any terminal. Unlike the OUTSTR UUO, TTYMES allows the message to start in any byte of a word. The end of the message is indicated by a null (zero) byte or by exhaustion of an explicit character count.

Upon call, AC should contain the address of a two word block. The first word of this block should contain either the name (physical or logical) of the destination terminal, e.g., SIXBIT /TTY21/, or the number of the destination terminal, e.g., 21. The second word of the block should contain in its right half the address of the ASCII message. The first six bits of the left half (pp in the sample call above) indicate the position (within the first word) of the first character of the message in normal byte pointer format; that is, pp is the number of bits to the right of the first byte of the message. If the position field contains zero, then 44 is assumed; that means the first byte of the word at MESS is the first character in the message. The remaining twelve bits of the left half (cccc above) may contain either the number of characters in the message or zero. If the count field contains zero, then the count is effectively set to infinity. Characters are sent to the destination until either the character count runs out or a null (zero) byte is encountered in the message. Thus, if you don't wish to calculate the length of your message, you can use a zero count and a null byte at the end of the message (i.e., use an ASCIZ string).

If you try to send a message to a nonexistent terminal, you get the error return. You also get the error return if the message can't be sent right now (for instance, if the TTY's output is being held) unless the terminal is your own, in which case this UUO waits until it can send the message.

SNEAKW [OP=047, ADR=400063] CALLI 400063

```
        SNEAKW AC,
```

SNEAKS [OP=047, ADR=400064] CALLI 400064

```
        SNEAKS AC,
        <return if no char is waiting for you>
        <success return>
```

The SNEAKW and SNEAKS UUOs look to see if any characters have been typed that have not yet been read. If there is such a character, it is returned in the AC. SNEAKW always returns a character, after waiting until there is one if necessary. SNEAKS never waits; if there is a character present, it is returned and the skip return is taken. Otherwise, SNEAKS does not change the AC and takes the direct return.

Note that SNEAKW does not wait for a whole line to be typed, only one character. Also, neither of these UUOs actually reads in a character.

ACTCHR [OP=047, ADR=400105] CALLI 400105

```
        ACTCHR AC,
```

The ACTCHR UUO (III and Data Disc displays only) waits for a line to be typed and then returns in AC the character (including control bits) that activated the last line re-edited with a control-CR or with a PTLOAD UUO (see page 55).

If you are not on a III or Data Disc display, this UUO returns 0 without waiting for a line to be typed.

CTLV [OP=047, ADR=400001] CALLI 400001

```
        CTLV
```

The CTLV UUO inverts the state of echoing of TTY input by the system. Normally every character you type is sent back to your terminal by the system so that you can see what you have typed. If you give this UUO when echoing is turned on (normal state), echoing will cease; and if you give this UUO when echoing is turned off, echoing will resume. The system always echoes characters typed in when the terminal is in monitor mode. See Section 3.1.

TTYIOS [OP=047, ADR=400014] CALLI 400014

```
        MOVE   AC,[<job number or sixbit device name>]
        TTYIOS AC,
```

The TTYIOS UUO returns the I/O status word of the device indicated by the contents of the AC. If AC contains a logical or physical device name, that device's status word is returned. If AC contains a job number, then the status word of the terminal belonging to that job is returned; if that job has more than one TTY, then there is no telling which one's status word will be returned. The I/O status word is returned in the AC; if there is no such device, -1 is returned. The meanings of some bits in the right half of the device status word are explained in Section 2.6. Other bits for specific devices are explained in the device writeups in Section 13.

GETLN [OP=047, ADR=34] CALLI 34

```
        GETLN AC,
```

The GETLN UUO is used to find out the physical name of the terminal attached to your job. The name (in sixbit) is returned in AC. If the job is detached, zero is returned.

3.5 Pseudo-Teletypes

The pseudo-teletype (PTY) is a special system concept designed to allow users to have control of more than one job at a time. A PTY is like a physical terminal in almost all respects. However, a PTY is controlled by the job which created it, and no other job can access it. To *type characters on a PTY*, the controlling job does character output to the PTY; and to *see the characters typed out on a PTY*, a job does character input from the PTY. If you send a new PTY the character "L" followed by a carriage return and linefeed, a job will begin logging in on the PTY. You can run programs and communicate with the monitor through a PTY just as you can through a physical terminal, but PTYs are controlled by program rather than by keyboard. Thus, a single job (attached to a terminal or even detached) can control one or more PTYs and hence one or more other jobs, which themselves can control other PTYs.

Just as each physical terminal has a unique line number, so does each PTY. Currently the line numbers assigned to PTYs begin with 121 and go upward. The PTYs have physical device names, just like other terminals; for example, the physical name the PTY on line 121 is SIXBIT /TTY121/. There is a maximum number of PTYs that the system can support at any one time and this maximum is currently 24.

PTYs have line characteristics just as other terminals do (see the GETLIN UUO on page 40). When a PTY is initialized (with the PTYGET UUO), it is set up with the following bits on in the characteristics word: 6--PTY, 10--(this bit is used by the system) and 16--linefeeds are not inserted after carriage returns. You can of course change certain bits in the characteristics word to suit your

purposes. This can be done for a PTY with the PTSETL UUO (see page 54) just as SETLIN (see page 42) does it for other terminals.

When you output characters to a PTY, those characters will be echoed by the monitor as usual and will thus appear in subsequent inputs that you do from the PTY. You can turn off the automatic echoing by the usual means for doing so with terminals; namely, you can turn on bit 15 in the PTY's line characteristics, (this inhibits echoing so that only linefeeds inserted after carriage returns get echoed), or you can have the job that is running on the PTY do a CTLV UUO (see page 47), which eliminates all echoing except while the PTY is in monitor mode. You can get the same effect as doing a CTLV (to turn echoing off) but more easily by using the PTJOBX UUO with the control function DOFF. This will always turn echoing off, whereas CTLV inverts the state of echoing, turning it off when it is on and on when it is off. The control function DON for PTJOBX can be used to turn echoing back on. The PTJOBX UUO is explained on page 56.

PTYUUO

The UUO that is used to communicate with PTYs is PTYUUO, which has many different functions that it can perform, including reading from and writing on a PTY.

PTYUUO [OP=711]

```
        PTYUUO <function number>,ADR

ADR:    <PTY's line number>
        <other data depending on the function>
```

PTYUUO is an extended UUO that uses the AC field to determine which of many possible pseudo-teletype functions is to be executed. Each of these functions (which are described in detail below) expects a two word block to be pointed to by the address field of the instruction. The right half of the first word of this block should contain the line number of the pseudo-teletype for which the UUO is intended (except with PTYGET which returns this line number). The second word is a data word that is used or returned by the UUO.

Doing PTYUUOs to Physical TTYs

It is sometimes useful for a program to be able to type things into its own TTY input buffer just as if the user had typed them. This can be done with PTYUUO by specifying a PTY line number of zero in the word at ADR. When this word is zero, the PTY function is executed with your terminal instead of with a pseudo-teletype. Thus, if you do output to a PTY with a line number of zero, the characters will go to your terminal's input buffer (into your line editor if you are on a III or Data Disc display) just as if you had typed them.

If bit 18 (the 0,,400000 bit) of the word at ADR is on for a PTYUUO, the number in the remaining bits of the right half is interpreted as the line number of a terminal and the PTY function is carried out for that terminal. However, if you do not own the terminal, you must have a privilege to do anything with it.

Any attempt to do input (PTRD1S, PTRD1W or PTRDS UUOs) from a physical terminal's output buffer with either of the above two methods will not work; the PTYUUO will simply be a no-op.

Now here are the individual PTYUUO functions.

PTYGET [OP=711, AC=0] PTYUUO 0,

```
        PTYGET ADR
        <return if no PTYs available>
        <success return>

ADR:    <PTY's line number and characteristics are returned here>
```

The PTYGET UUO gets you a pseudo-teletype and places its line characteristics word (see the GETLIN UUO on page 40) in the location pointed to by the effective address of the instruction (i.e., ADR). This means that the PTY's line number will appear in the right half of ADR and its characteristics will appear in the left half.

If a PTY is available, this UUO skips and the PTY is assigned to you. If there are none available, the direct return is taken and you get no PTY.

PTYREL [OP=711, AC=1] PTYUUO 1,

```
        PTYREL ADR

ADR:    <line number of PTY to be released>
```

The PTYREL UUO releases the pseudo-teletype whose line number is in the right half of ADR. The job running on that PTY is killed, and any PTYs it may have acquired are released.

The RESET UUO (see page 126) releases all pseudo-teletypes you own.

PTIFRE [OP=711, AC=2] PTYUUO 2,

```
        PTIFRE ADR

ADR:    <PTY's line number>
        <count of free bytes is returned here>
```

The PTIFRE UUO returns to you in ADR+1 the number of free bytes left in the input buffer of the PTY whose line number is in ADR. This is the number of characters you may send to that PTY before its input buffer is full. If you use a PTY output UUO that waits, and if you send more that this many characters, you will have to wait until the program running on the PTY reads some characters and makes room for your output.

PTOCNT [OP=711, AC=3] PTYUUO 3,

```
        PTOCNT ADR

ADR:    <PTY's line number>
        <count of characters in PTY's output buffer is returned here>
```

The PTOCNT UUO returns in ADR+1 the number of characters in the output buffer of the PTY whose line number is in ADR.

PTRDIS [OP=711, AC=4] PTYUUO 4,

```
        PTRDIS ADR
        <return if no character is present>
        <success return>

ADR:    <PTY's line number>
        <one 7-bit character is returned here>
```

The PTRDIS UUO looks to see if there are any characters in the output buffer of the PTY whose line number is in ADR. If there are, one 7-bit character is read from there and returned in ADR+1 and the skip return is taken. If there are no characters in the PTY's output buffer, then a zero is returned in ADR+1 and the direct return is taken.

PTRD1W [OP=711, AC=5] PTYUUO 5,

```
        PTRD1W ADR

ADR:    <PTY's line number>
        <one 7-bit character is returned here>
```

The PTRD1W UUO reads one 7-bit character from the output buffer of the PTY whose line number is in ADR and returns the character in ADR+1. If there are no characters in the PTY's output buffer, this UUO waits for the PTY to do some output and then returns the first character.

If the PTYWAKE bit (bit 9--the 400,,0 bit) is on in the line characteristics word for your terminal (see the GETLIN UUO on page 40), then this instruction will return when a character is typed either on the PTY or on your terminal. If the first character typed is from the terminal, a zero will be returned in ADR+1 and no characters will have been read from either the terminal or the PTY. In this manner, you may wait for either PTY or TTY input.

PTWR1S [OP=711, AC=6] PTYUUO 6,

```
        PTWR1S ADR
        <return if character not sent>
        <success return>

ADR:    <PTY's line number>
        <9-bit character to be sent>
```

The PTWR1S UUO sends the 9-bit character right-justified in ADR+1 to the PTY whose line number is in ADR. If the character can be sent, the skip return is taken. If the PTY's input buffer is already full, the character is not sent and the direct return is taken. The CONTROL and META keys which display lines have are represented by bit 28 (0,,200--CONTROL) and bit 27 (0,,400--META) in ADR+1.

The [ESCAPE], [BREAK] and [CLEAR] characters (as available on Data Disc and III display keyboards) can be sent with this UUO (or with the PTWR1W UUO below) by having ADR+1 contain 0,,10042 for [ESCAPE] or 0,,10041 for [BREAK] or 0,,10044 for [CLEAR]. Also, if bit 23 (0,,10000 bit) is on in ADR+1, then the character represented by the low order 7 bits (if not 41, 42 or 44) will be sent to the PTY preceded by the [ESCAPE] character; and if both bits 23 and 24 (0,,14000 bits) are on, then the character in the low order 7 bits will be sent preceded by the [BREAK] character. In either of these two cases, if the character in the low order 7 bits is not an [ESCAPE] or [BREAK] keyboard command and is not either 41, 42 or 44, then this UUO becomes a no-op, taking the skip return. Also, if the PTY specified is not a display, then only the low order 7 bits (bits 29:35--the 0,,177 bits) of ADR+1 are used; no [ESCAPE] or [BREAK] commands can be sent for teletypes. (Imlacs can send 8 bits; the 0,,200 bit on an Imlac means that this character should not be checked against the special activation table even if the Imlac is in special activation mode.)

PTWR1W [OP=711, AC=7] PTYUUO 7,

```
        PTWR1W ADR

ADR:    <PTY's line number>
        <9-bit character to be sent>
```

The PTWR1W UUO sends the 9-bit character right-justified in ADR+1 to the PTY whose line number is in ADR. If the PTY's input buffer is already full, this UUO waits until there is room and then sends the character.

This UUO interprets the character in ADR+1 exactly as the PTWR1S UUO does. See the PTWR1S UUO above for the details of sending the [ESCAPE], [BREAK] and [CLEAR] characters to a display's input buffer.

PTRDS [OP=711, AC=10] PTYUUO 10,

```
        PTRDS ADR

ADR:    <PTY's line number>
        <address or byte pointer for returned string>
```

The PTRDS UUO reads all the characters that are in the output buffer of the PTY whose line number is in ADR and returns these characters as an ASCIZ string at the location indicated by ADR+1. If bits 6:17 (7777,,0 bits) in ADR+1 are not all zero, the word at ADR+1 is taken as a byte pointer to be used to return the ASCIZ string, with the byte pointer incremented before the first character is deposited. (Before the byte pointer is used, the size field is set up for 7-bit bytes and the index and indirect fields are cleared.) If bits 6:17 of ADR+1 are zero, the ASCIZ string is returned such that the first byte is in the high-order 7 bits of the location pointed to by the right half of ADR+1.

PTWRS7 [OP=711, AC=11] PTYUUO 11,

```
        PTWRS7 ADR

ADR:    <PTY's line number>
        <address or byte pointer for string to be sent>
```

The PTWRS7 UUO takes the ASCIZ string specified by ADR+1 and sends it to the PTY whose line number is in ADR. This UUO waits as necessary until the whole string has been sent. If bits 6:17 (7777,,0 bits) of ADR+1 are not all zero, then ADR+1 is taken as a byte pointer to be used to access the string, with the byte pointer incremented before the first character is loaded. (Before the byte pointer is used, the size field is set up for 7-bit bytes and the index and indirect fields are cleared.) If bits 6:17 are zero, the string is assumed to start in the high-order 7 bits of the word pointed to by ADR+1.

PTWRS9 [OP=711, AC=12] PTYUUO 12,

```
        PTWRS9 ADR

ADR:    <PTY's line number>
        <address or byte pointer for string to be sent>
```

The PTWRS9 UUO does the same as PTWRS7 except that the string sent is not a standard 7-bit ASCIZ string, but a string of 9-bit characters terminated by a zero (null) character. The two high order bits (400 and 200 bits) in each 9-bit character represent the CONTROL and META keys, respectively, which Data Disc and III display keyboards have. As with PTWRS7, ADR+1 can contain either a simple pointer to the string or a byte pointer to it.

This UUO is important because octal code 003 (β or control-C) does not mean control-C if you are sending this string to yourself (line number in ADR set to zero) and you are at a III or Data Disc display. The code representing control-C in that case is 600, which takes 9 bits to represent. The code 600 works as control-C for all PTYs, so it can always be sent instead of 003 to stop a job. Note that you must send *two* 600s (two control-C's) to stop a job immediately. Another method for stopping a PTY is to use the PTJOBX UUO with the HALT control function; see page 56.

PTGETL [OP=711, AC=13] PTYUUO 13,

```
        PTGETL ADR

ADR:    <PTY's line number>
        <PTY's line characteristics word is returned here>
```

The PTGETL UUO returns in ADR+1 the line characteristics for the PTY whose line number is in ADR. The meaning of the line characteristics word is explained under the GETLIN UUO on page 40.

Note: If a pseudo-teletype is controlled directly or indirectly (through a chain of pseudo-teletypes) by a Data Disc or III display, then the Data Disc bit (bit 4--the 20,,0 bit) or the III bit (bit 0--the 400000,,0 bit) will be on in the characteristics word for the pseudo-teletype.

PTSETL [OP=711, AC=14] PTYUUO 14,

```
        PTSETL ADR

ADR:    <PTY's line number>
        <new line characteristics desired>
```

The PTSETL UUO sets the line characteristics for the PTY whose line number is in ADR from the word at ADR+1. In the line characteristics word, only bits 2, 8, 9, 11, 13, 14, 15 and 16

(101536,,0 bits) can be changed by the user. Other bits in the word at ADR+1 are ignored. See the GETLIN UUO on page 40.

PTLOAD [OP=711, AC=15] PTYUUO 15,

```
        PTLOAD ADR

ADR:    <PTY's line number--must be a display>
        <address or byte pointer for string>
```

The PTLOAD UUO loads a display's line editor with the ASCII string specified by the word at ADR+1. If the left half of ADR+1 is non-zero, then ADR+1 is assumed to be a byte pointer to the string; characters are picked up from the string with ILDBs, so the byte pointer should point to the byte just before the string. The end of the string is defined by the first occurrence of a null or an activation character. The activation character, if any, is left on the end of the string in the line editor; typing carriage return will simply activate the line up to and including any activation character. Note that if the string was terminated with a null, then typing carriage return will activate the string but will not add a carriage return to it. If you activate the re-edited line with anything besides carriage return, the activation character will be inserted in the line wherever you typed it.

This UUO only works if the line number at ADR specifies a III or Data Disc display; otherwise this is a no-op. The normal use of this UUO is with a zero at ADR, which selects your own line editor to be loaded with the given string (provided you are at a display). After giving this UUO, you may read input from the terminal to get the line back with any changes that may have been made. See particularly the INWAIT UUO on page 44 and the ACTCHR UUO on page 47 for special features regarding re-edited lines.

PTJOBX [OP=711, AC=16] PTYUUO 16,

```
        PTJOBX ADR
        <error return for HALT and CONT--no job logged in>
        <error return for CONT--job cannot be continued>
        <success return for CONT>

ADR:    <pseudo-teletype line number>
        <index, or sixbit name, of control function>
```

Control functions:

Index	Name	Function	Success Return
1	HALT	Stop the pty's job.	skips
2	CONT	Continue the pty's job.	double skips
3	DOFF	Turn off echoing of input to pty.	no skip
4	DON	Turn on echoing of input to pty.	no skip
5	LOGIN	Log in a job on pty.	skips
6	IWAITS	Skip if pty waiting for input.	skips

PTJOBX is the extended PTY job control UUO. Any one of several control functions can be exercised over a PTY without sending it any character strings. The control is exercised over the job running on the PTY whose line number is in ADR; a zero line number means the control is exercised over your own job. This is a good method for turning on and off the echoing of input to your job.

The word at ADR+1 should contain either the index or the sixbit name of the control function desired. The currently available control functions are listed above with their names and indices.

The HALT function takes the skip return on success and the direct (error) return if there is no job logged in on the PTY. If you HALT your own job (PTY line 0) this way, your terminal is left in user mode and you cannot type commands to the monitor; you must type control-C to get out of this condition.

The CONT function takes the double skip return on success, the direct return if there is no job logged in on the PTY, and the single skip return if the job cannot be continued.

The DOFF and DON functions always take the direct return; DOFF sets bit 28 (0,,200 bit) in the TTY I/O status word (thus turning *off* echoing) and DON clears this bit (turning echoing back *on*). See Section 3.1 for the meaning of this bit.

The LOGIN function logs the PTY in under the PPN of the controlling job and copies the controlling job's Disk PPN and privileges; the job number of the new job is returned at ADR+1 and the skip return is taken if the job gets successfully logged in. If there is already a job logged in on the PTY, that job's number is returned in ADR+1 and the direct return is taken. (This is a good way to find out the number of the job logged in on a PTY.) If there is no job logged in on the PTY but there are no job slots available, zero is returned in ADR+1 and the direct return is taken.

The IWAITS function lets you find out if the PTY is waiting for input. That is, if the PTY has returned to monitor level and needs a monitor command typed to it, or if the job running on the PTY is waiting for TTY input, then this function skips. Otherwise, the direct return is taken.

SECTION 4

DISPLAY OUTPUT

The availability of displays for output provides great flexibility and convenience in many programs. This section explains several UUOs that allow the user to determine what will appear on a display. These UUOs include: the PPIOTs to select and position various *pieces of paper* on your screen; the PGIOTs and UPGIOT to run display programs on Data Discs and IIIs and to select from various *pieces of glass* on IIIs; DDCHAN to acquire and manipulate extra Data Disc channels; VDSMAP to select the sources for the picture on a Data Disc display; ADSMAP to select the sound source to be connected to the speaker associated with a given display; and a few other related UUOs.

This section does *not* discuss how to program the III or Data Disc display processors. These processors are explained in Appendix 1 and Appendix 2, respectively. However, since the two display processors are somewhat different in their operation, I shall attempt to explain for each how it works and how it interacts with the system so that you can understand what the various UUOs are intended to do. Some references are made to specific III or DD instructions; these are all explained in the above-mentioned appendices.

4.1 III Displays

The III display processor runs continuously, executing display instructions from main memory. The code it executes is located in system free storage. Any change to a single word of this code will cause the resultant display to change. A user-written III display program can be run by using the UPGIOT UUO. (The III instructions are explained in Appendix 1.) UPGIOT takes as arguments the location and length of the display program to be run. The system copies the display program out of the user's core image into free storage, making transformations such as address relocation.

The system uses the first word in your program to interface with other display programs; you *must* include an extra word at the beginning of each III display program you write. To exit from the middle of your display program, insert a HLT instruction. Otherwise, you should simply plan to fall through to the end of your program. You need not have a HLT at the end.

Every job running on a III is permitted to have up to 20 independent display programs. A III display program is called a piece of glass, and each piece of glass (hereafter abbreviated POG) has a number between 0 and 17 inclusive. RAID uses POG 17; please note the obvious conflict if your program uses this POG too.

You may choose which of your pieces of glass are to be visible and which are to be invisible. The display code for an invisible POG continues to reside in system free storage but is simply not executed; you may reactivate it at any time.

The only III instruction that is illegal in a user display program is the JMS instruction. To get the effect of a JMS, use the JSR and/or SAVE instructions (see Appendix 1).

You are not allowed to display anything on someone else's III unless you are privileged. You may display on a III that is not in use.

4.2 Data Disc Displays

The Data Disc (DD) display processor works by storing complete TV pictures bit by bit on a disk. The disk we have has =64 tracks; two tracks are needed to hold a complete picture. Thus =32 complete TV images can be stored on the Data Disc. Each combination of two tracks that makes up a whole picture is called a Data Disc channel. The Data Disc hardware reads the disk and puts out a TV signal for each channel. Each TV signal can then be routed by the video switch (controlled by software) to any combination of TV monitors (Data Disc displays).

When the DD processor executes a program, the resultant picture changes are recorded on the disk, and the displayed picture changes. The Data Disc display processor does not execute the same display program continuously like the III processor. The only way to make a DD picture disappear is to explicitly erase it from the disk by means of a DD program (from either the system or a user).

There is no such thing as a piece of glass on a DD display. Also, you should *not* include an extra word at the beginning of a DD display program as you do for III display programs; every word of a DD display program is executed as a display instruction. (The DD instructions are explained in Appendix 2).

More words of warning:

The DD processor executes your display program directly from your core image. No address relocation is done; thus jumps will not work correctly since their destination addresses are taken as absolute!

Your Data Disc display program should end with a HALT instruction; if it does not, the system will zero the last word of the program to make sure it halts.

Finally, you may do a channel select only to your own main channel or to an extra DD channel that you own or are permitted to write on. If you have no channel select in the first 10 words of your DD program, the system will select your main DD channel for you. (On the DD display processor, only the first channel select in a program will work; so any channel select beyond the first 10 words will be ignored.) In addition, a select of channel 0 will get your main DD channel. Thus to select the real channel 0, you select channel 40. You see, 40 is non-zero so you don't get your main channel, but only the low order 5 bits of the channel number are used in the select (since there are only =32 channels). You can always select channel 40+C and be assured of getting channel C. However, if you select a channel that you are not allowed to write on, the select is changed to your own main channel. In all of these cases where the system selects your own main DD channel for you, it does so by starting the DD processor at a special two word block that contains a channel select in the first word and a jump to your program in the second word.

4.3 Page Printer Manipulation

Your page printer is the part of the monitor that prints text on your display screen. Normally your entire screen is used by the page printer. However, you may have up to 20 logical *pieces of paper*, numbered from 0 to 17 inclusive, to which TTY output and echoing of input may be directed, and each of these pieces of paper may be placed at any part of the screen. You select the piece of paper (hereafter abbreviated PP) which is to be used currently, and all TTY printing and echoing will go to that PP until you select some other PP. Initially, PP 0 is selected.

PPIOT [OP=702]

```
PPIOT <function>,<argument>
```

PPIOT is an extended UUO that uses the AC field to determine which of several page printer functions is to be executed. The individual functions are described separately below.

PPSEL [OP=702, AC=0] PPIOT 0,

```
PPSEL <piece of paper number>
```

The PPSEL UUO selects the piece of paper whose number is the effective address of the instruction. This number should be between 0 and 17, inclusive. Piece of paper number zero is the one normally selected for you. After you give this UUO, all your TTY printing and echoing will go to the specified piece of paper. This UUO deactivates all other PPs as if you had done a PPACT UUO with only this PP specified. See the PPACT UUO below.

PPACT [OP=702, AC=1] PPIOT 1,

```
PPACT <piece of paper activation map>
```

The PPACT UUO is used to display selected pieces of paper. The effective address of the instruction is interpreted as a bit map indicating which pieces of paper are to be displayed. A one in bit 18+N will cause PP number N to be displayed. Thus bit 18 (400000 bit) of the effective address is for PP 0, bit 19 (200000 bit) for PP 1, etc.

On the IIIs any PPs turned off by this UUO will disappear. However, on the Data Discs those PPs will not disappear; you must erase them explicitly if you no longer want them displayed.

DPYPOS [OP=702, AC=2] PPIOT 2,

```
        DPYPOS <Y-position>
```

The DPYPOS UUO causes the currently selected piece of paper to be positioned on the screen so that its first line is located at the Y-position specified by the effective address of this instruction, where +1000 is the top of the screen and -1000 is the bottom.

DPYSIZ [OP=702, AC=3] PPIOT 3,

```
        DPYSIZ <G*1000 + L>
```

The DPYSIZ UUO sets two values for the currently selected piece of paper: the number of glitches (G) and the number of lines per glitch (L). Both of these numbers are set from the effective address of the instruction, G from bits 18:26 (0,,777000 bits) and L from bits 27:35 (0,,777 bits).

Now a word about glitches. When a piece of paper fills up, the text in it jumps up to provide room for more text (thus moving some text off the top of the PP). This jumping up is called a glitch. The number of lines it jumps is the number of lines per glitch (L) as set by the last DPYSIZ for this PP or by the default if no DPYSIZ has been given. The total number of lines in a PP is L*G.

PPREL [OP=702, AC=4] PPIOT 4,

```
        PPREL <piece of paper number>
```

The PPREL UUO releases the piece of paper whose number is the effective address of the UUO. Piece of paper zero cannot be released; PPREL 0 is a no-op. The system storage associated with a PP being released is freed; you cannot possibly get back the text that was on it. If you release the currently selected PP, a PPSEL (see above) to piece of paper zero is done. On IIIs, released PPs will disappear.

PPINFO [OP=702, AC=5] PPIOT 5,

```
        PPINFO ADR

ADR:    <block 24 words long for returned information>
```

The PPINFO UUO gives you a 24-word table of information about your page printer. The effective address in the instruction should point to a 24-word block into which the table is to be placed.

The information returned in the table is indicated below, where PP means piece of paper and PG means piece of glass. To get the same information for some other job, use the PPSPY UUO (see page 64).

Words	*Value*
0	<POG activation bits>,,<PP activation bits> These are in PGACT and PPACT formats. See these two UUOs.
1	<number of the currently selected PP>
2	Bit 0 (400000,,0 bit) is 1 if the Data Disc page color is green on black. Bit 1 (200000,,0 bit) is 1 if your screen has been erased by an escape command since you last gave this UUO. Bit 2 (100000,,0 bit) is 1 if you are on a Data Disc display. Bits 18:35 (the 0,,777777 bits) hold your line editor Y-position in LEYPOS format. See the LEYPOS UUO below.
3:22	<Y-position>,,<G * 1000 + L> There is one word here for each PP; in word 3 is the status for PP 0, word 4 for PP 1, etc. The <Y-position> is in DPYPOS format, G means number of glitches, and L means lines per glitch. See the DPYPOS and DPYSIZ UUOs above.
23	Bit 0 (400000,,0 bit) is 1 if the size of the currently selected PP was last set by keyboard command rather than by UUO. Bit 1 (200000,,0 bit) is the same for the Y-position of the current PP. Bit 2 (100000,,0 bit) is the same for the line hold count. Bit 3 (40000,,0 bit) is the same for the glitch hold count. Bits 9:17 (777,,0 bits) have the actual line hold count. Bits 18:26 (0,,777000 bits) have the actual glitch hold count. Zero in either of these hold counts means that that hold count is not being used. Bits 27:35 (0,,777 bits) hold the character (including control bits) that activated the last line re-edited with a control-CR or with a PTLOAD UUO (see page 55).

LEYPOS [OP=702, AC=6] PPIOT 6,

```
LEYPOS <Y-position for line editor>
```

The LEYPOS UUO sets the Y-position of your line editor to that specified by the effective address of the instruction, which is interpreted in DPYPOS format (+1000 is top of screen, -1000 is bottom). A Y-position of zero does *not* mean the middle of the screen, but instead means return the line editor to the bottom of your page printer (its normal location).

```
PPHLD           [OP=702, AC=7]  PPIOT 7,
-------------------------------------------------
        PPHLD <LHC*1000 + GHC>
```

The PPHLD UUO sets the line hold count (LHC) and the glitch hold count (GHC) for your page printer. These two numbers indicate how many lines or glitches the system should allow to be printed before automatically holding the typeout for your display. Both the LHC and the GHC are set from the effective address of the UUO, LHC from bits 18:26 (the 0,,777000 bits) and GHC from bits 27:35 (the 0,,777 bits). If the high order bit of either of these fields is on, the corresponding hold count is *not* changed. A zero in either field disables that particular type of automatic holding.

```
PPSPY           [OP=047, ADR=400107]  CALLI 400107
-------------------------------------------------
        MOVE   AC, [<job number or -tty number>,,ADR]
        PPSPY AC,
        <error return>

ADR:    <24-word block for returned information>
```

The PPSPY UUO returns a 24-word block of information about the page printer of a specific job or display. The information returned is exactly the same as that returned by the PPINFO UUO (see page 62). The right half of AC should contain the address of the 24-word block where you want the information returned. The left half of AC should contain either the number of the job or the negative of the number of the display for which you want the page printer information. If this UUO is successful, the skip return is taken. If there is no such job or if the terminal is not a display, then the direct (error) return is taken.

4.4 Running Display Programs

This section describes the UUOs that allow the user to have his own display programs run on the III and Data Disc display processors.

UPGIOT [OP=703]

```
        UPGIOT <piece of glass number>,ADR

ADR:    <flags>,,<address of display program>
        <length of display program in words>
        <transfer-in-progress flag, if bit 0 in ADR is on>
        <address of low order line command, if bit 1 in ADR is on>
```

The UPGIOT UUO (also known as DPYOUT) causes a display program to be run. If you are on a Data Disc, the program is assumed to be a Data Disc display program and thus is run on the Data Disc display processor. If you are on a III terminal, the program is run on the III display processor. If you are on a pseudo-teletype (PTY) which is owned either directly or indirectly (that is, through a chain of PTYs) by a job running on a display, then the program is run on that display, whether it be III or Data Disc.

If the display program is to be run on a III, the AC field of the instruction indicates which piece of glass the program is to be run as. If the program is intended for a Data Disc display, the AC field is ignored.

The address field of this UUO points to a data block, of which the first word contains the address of the display program that is to be run and the second word contains the program's length in words.

For Data Disc programs there are some other optional parameters which you may specify. If bit 0 (400000,,0 bit) of the word at ADR is on, then the display program is run in overlapped mode. This means the UUO will return without waiting for the display program to finish. (However, it will wait for any previous DD program to finish.) In this mode the word at ADR+2 is set non-zero while the program is being sent to the DD processor and is set to zero when the program has finished. Thus you can test to see if the program has completed; you should not change any part of the display program until ADR+2 has been set to zero. Also, if you indicate a DD program length of zero in ADR+1, no program will be run at all, but the UUO will not return until any previous DD program has finished.

If bit 1 (200000,,0 bit) at ADR is on, the display program is run in double field mode. This is useful for writing text on a Data Disc channel. Normally you have to send text to the DD processor twice, once for each of the two tracks that make up the DD channel. In these two passes, you would indicate two line addresses that are the same except in the low order bit position. In double field mode, the system sends the program to the DD processor twice, once with the low order bit of the line address select set to zero and once with it set to one. The original value of this bit when you give the UUO is irrelevant and its final value is unspecified. The low order line address select must occur as the third command in the word pointed to by ADR+3. This feature will not work properly if you have more than one line address select in your DD program.

For details on the format of a display program, see Section 4.1 for IIIs and Section 4.2 for Data Discs. For descriptions of the instructions for the two display processors, see Appendix 1 and Appendix 2.

PGIOT [OP=715]

```
        PGIOT <function>,<argument>
```

The PGIOT UUO is an extended UUO that uses the AC field to determine which of several display functions is to be executed. The individual functions are described separately below. Of these, the PGSEL, PGACT, and PGCLR UUOs are meaningful only for IIIs and are no-ops when given on Data Discs.

PGSEL [OP=715, AC=0] PGIOT 0,

```
        PGSEL <piece of glass number>
```

The PGSEL UUO causes the piece of glass whose number is the effective address of the UUO to be selected. This means that the UPGMVM and UPGMVE UUOs (see page 67) will refer to this piece of glass until the next PGSEL is given.

PGACT [OP=715, AC=1] PGIOT 1,

```
        PGACT <piece of glass activation map>
```

The PGACT UUO is used to select which pieces of glass are to be displayed. The effective address of this UUO is interpreted as a bit map; a one in bit 18+P will cause piece of glass number P to be displayed, and a zero will cause that piece of glass to be invisible.

PGCLR [OP=715, AC=2] PGIOT 2,

```
        PGCLR
```

The PGCLR UUO causes all of your pieces of glass to be cleared. This means that the system free storage that was allocated for these PGs is freed and whatever was displayed by them disappears never to be seen again. This UUO does not affect your page printer at all.

DDUPG [OP=715, AC=3] PGIOT 3,

```
        DDUPG ADR

ADR:    <flags>,,<address of display program>
        <length of display program in words>
        <transfer-in-progress flag, if bit 0 in ADR is on>
        <address of low order line command, if bit 1 in ADR is on>
```

The DDUPG UUO causes a user's Data Disc display program to be run. This UUO works just like UPGIOT except that the program is always run on the Data Disc display processor. See the UPGIOT UUO on page 65 for an explanation of the various options.

PGINFO [OP=715, AC=4] PGIOT 4,

```
        PGIOT ADR

ADR:    <block 21 words long for returned information>
```

The PGINFO UUO returns a 21 word table of information about your pieces of glass. The effective address of the instruction should specify the location of a 21 word block where the table is to be stored. The information returned in the table is indicated below.

Words	*Value*
0	<POG activation bits>,,<PP activation bits> These are in PGACT and PPACT formats respectively; see these two UUOs.
1:20	<word count>,,<starting address> There is one word here for each piece of glass.

UPGMVM [OP=714]

```
        UPGMVM AC,ADR
```

The UPGMVM UUO is used to update a III display program. Before you give this UUO, you must have selected some piece of glass with the PGSEL UUO. This UUO is then used to update the display program of that piece of glass by replacing the word that would have been at ADR in that program with the word in the specified AC. In other words, you could update your display program by doing a MOVEM AC,ADR and then another UPGIOT, or you can give this UUO with the same AC and ADR specified. This causes a change to one word of your display program,

which is already running (unless deactivated by a PGACT UUO). The address ADR must be within the bounds of the core area that contained the display program when you created this piece of glass with the UPGIOT UUO.

UPGMVE [OP=713]

```
UPGMVE AC,ADR
```

The UPGMVE UUO is the MOVE analog of the UPGMVM UUO described above. This UUO picks up a word from the currently selected display program and returns it in the specified AC.

DPYCLR [OP=701]

```
DPYCLR
```

The DPYCLR UUO resets your display to its initial state. Any display programs running are cleared and the page printer is returned to its normal condition. On a III, this means that all pieces of glass will disappear. On a Data Disc, however, nothing is erased except that which is overwritten by the newly redrawn page printer. The RESET UUO (see page 126) simulates a DPYCLR. If you are not on a display, this UUO has no effect.

4.5 Extra Data Disc Channels

The DDCHAN UUO is provided to allow users to acquire extra Data Disc channels for use in displaying text and graphics.

DDCHAN [OP=047, ADR=400067] CALLI 400067

```
MOVE   AC,[<channel request>]
DDCHAN AC,
<error return - for get channel and set status only>
```

The DDCHAN UUO is used to get and release DD channels and to get and set the status of DD channels. This UUO should be given with a channel request in the specified AC; this request is interpreted as follows.

Bits	*Octal*	*Meanings of bits in a Data Disc channel request*
28:29	0,,300	Operation. 0 = release channel 2 = get status 1 = get channel 3 = set status A *get channel* will fail if the requested channel is unavailable. A *set status* will fail if the channel doesn't belong to you. These two commands skip on success; otherwise, the direct return is taken.
30:35	0,,77	Channel number. Values 0 through 37 specify a particular DD channel. In a *get channel* request, the value 77 specifies *any channel*. In a *get status* or *set status*, 77 means your main channel. In a *release channel*, 77 releases all channels assigned to the job. Other values for the channel number are undefined.
0	400000,,0	Privacy flag. A one in this bit means no one else can look at this channel.
1	200000,,0	Write permission. A one in this bit means that other jobs may write on this channel.

After execution of this UUO (except for a *release all channels* or a *get any channel* failure), the AC contains the channel number in the right half, the privacy and write permission status in bits 0 and 1, plus the *channel use* in bits 10:17 (377,,0 bits). A value of zero in this use field means the channel is free; 1 through 77 mean that it is an extra channel belonging to that job number; 100 through 177 mean that it is the main channel for the TTY line whose number is 52 less than this number; 200 through 377 are for special channels, such as the one used by the system to advertise free DD terminals.

The RESET UUO (see page 126) releases all of your extra DD channels.

Example: To request a private channel that only your job can write on.

```
        MOVE    AC, [400000,,177]
        DOCHAN  AC,
        JRST    LOSE
WIN:    ...
```

If you get to WIN, the channel number will be in the right half of AC and the left half will have the sign bit on with your job number in bits 10:17.

4.6 The Video Switch

The Video Switch is the device that determines which pictures will appear on which TV monitors (Data Disc displays). Available to these displays are =32 Data Disc (DD) channels, numbered 0 to 37 octal, a null channel, and 7 television channels, numbered 41 to 47.

Each TV monitor is controlled by a 36-bit map that specifies which channels are connected to that monitor. This map is explained below.

Bits	*Octal*	*Meanings of bits in a Data Disc display map*
0:31	777777,,777760	DD channels. A one in bit N means that DD channel N is connected to this monitor.
33:35	0,,7	TV channel. Zero in this field selects the null channel; 1 through 7 select the corresponding TV camera, receiver, or synthesizer. In terms of keyboard commands, these are channels 41 through 47.
32	0,,10	Asynchronous flag. A zero in this bit causes DD sync to be inserted; a one blocks DD sync and all DD channels. Whenever bits 33:35 specify an asynchronous source, the system automatically sets this bit to one.

Example: A map containing "200000,,7" selects DD channel 1 and TV channel 47.

The system keeps two maps for each TV monitor; these maps are called the permanent map and the temporary map. The video switch is set to the permanent map whenever a RESET (CALLI 0) is performed.

The following UUO has been added to permit user programs to set the maps for particular TV monitors.

```
VDSMAP          [OP=047, ADR=400070]  CALLI 400070
-------------------------------------------------
        MOVE    AC,[<switch request>,,ADR]
        VDSMAP AC,
        <error return for all operations except "get map">

ADR:    <DD channel map>
```

The VDSMAP UUO is used to select the channels that are displayed on a given TV monitor or to find out which channels are currently being displayed. The right half of the AC should point to a

channel map, as described above; the meaning of the <switch request> in the left half is explained below. If you specify a TTY line that is not a TV monitor, then -1 is returned in the AC. Otherwise, the new map for the indicated monitor is returned in AC.

Bits	*Octal*	*Meanings of bits in a video switch request*
11:17	177,,0	TTY line number of the DD display to be switched. Zero means your own TTY line. Other lines associated with TV monitors (26 through 117) may be switched only if they have no job logged in.
9	400,,0	Shadow line map. If this bit is on, then the UUO will refer to one of the six unused TV lines rather than to a normal TV monitor. The number in bits 11:17 should be in the range 0:5 and specifies which one of these six lines is to be mapped or examined.
0	400000,,0	Temporary/permanent flag. Zero means make a temporary change; one means make a permanent change.
6:8	7000,,0	Operation. 0 = Get map. The current channel map of the indicated line is returned in the AC, and the direct return (no skip) is always taken. 1 = Set channel map from word at ADR. This operation skips if it is entirely successful. 2 = Add specified channels. Bits 0:31 of the map at ADR are "or"ed into the current map. If bits 33:35 of the map at ADR are not all zero, they replace the corresponding bits in the current map. This operation skips on complete success. 3 = Delete channels (inverse of 2). The complements of bits 0:31 are "and"ed with the old map. If bits 33:35 in the new map are not all zero, this field is cleared to zero, which selects the null TV channel. This operation can fail only on a busy line number. It skips on success. 4 = Reset map. In temporary mode (per bit 0), reset the map to the permanent one. In permanent mode, reset the map to the main channel alone. This operation can fail only on a busy line number. It skips on success. 5:7 = Undefined.

An attempt to map someone else's private channel to any display will fail. However, each channel being mapped is considered separately, and a mapping operation may successfully map some channels while failing on others. If the mapping operation fails on at least one channel, then the

UUO will take the error return. Also, unless you are privileged, you cannot change the map of someone else's display. A job running on a PTY can change the map of the display of the job owning the PTY (possibly through a chain of PTYs).

Example: To temporarily connect DD channel 21 and TV channel 47 to your monitor.

```
        MOVE    AC, [1000,,[1,,7]]
        VDSMAP  AC,
        JRST    LOSE
WIN:    ...
```

Example: To get the current channel map of TTY line 37.

```
        MOVSI   AC,37
        VDSMAP  AC,
        ...
```

The channel map would be returned in AC.

4.7 The Audio Switch

Associated with each display is an audio speaker which can be connected to any one of several sound sources. The ADSMAP UUO (see below) is used to choose the sound source to be heard over the speaker at a program's attached terminal. The BEEP UUO is used to send a short beep to any display's speaker.

Each sound source that can be connected to a display's speaker is assigned an audio channel number. The current assignments of channels to sources is as follows:

Channel	*Sound source*
0	Laboratory personnel paging system.
1	Lounge TV audio.
2	Tuner in Helliwell's office.
3	Tuner in computer room.
4	AD D-to-A output converter, channel 1.
5	A continuous beeping--used by the BEEP UUO.
6:17	Unconnected.

To connect a speaker to a sound source, the user specifies the source's channel number. A speaker cannot be connected to more than one source.

Each display has both a *permanent* audio switch connection and a *temporary* connection. When a temporary connection is made, its duration (possibly infinite) must be specified; when the duration runs out, the permanent connection is reselected automatically. A RESET (see page 126) will also reselect the permanent connection.

The system allows a user to send a beep to a display terminal in order to attract the attention of that display's user. Since a beep may interrupt something a user is listening to, the user is

permitted to inhibit beeping on his display's speaker. Furthermore, the personnel paging system of the laboratory uses the audio switch to make paging announcements over users' speakers, and these pages may also interrupt a user's selected sound source. Thus the following system has been implemented to allow each user to decide what will interrupt his audio switch connections.

Four possible dispositions are allowed for handling audio interruptions; for each connection, one of these is selected for paging interruptions and one for beep interruptions:

```
INTERRUPT
DON'T INTERRUPT
INTERRUPT WITH EXTENDED DURATION
DELAY BEEP
```

Interrupt means that if an interruption comes along, the connection will be momentarily changed until the beep or page ends.

Don't interrupt means ignore all interruptions; no change will be made even momentarily to the audio switch connection.

Interrupt with extended duration means allow interruptions to take place but extend the duration of the connection. This is meaningful only for temporary connections.

Delay beep means postpone any beep interruption until the expiration of the connection. This again applies only to temporary connections, and further is not a defined disposition for paging interruptions.

```
ADSMAP          [OP=047, ADR=400110]  CALL1 400110
-------------------------------------------------
        MOVE    AC,[<audio switch connection>]
        ADSMAP AC,
```

The ADSMAP UUO is used to connect a specific sound source to a display's speaker or to find out the status of a display's audio switch connection. The job giving this UUO must be attached to a display terminal for this UUO to do anything; also it is not possible for a job to affect the audio switch connection for any display but its own.

If AC contains -1, the audio switch connection is reset to the current permanent connection. Otherwise, the value in AC specifies either the temporary or permanent connection and indicates whether that connection is to be changed or just its status returned. If a temporary connection is to be made, the duration of the new temporary connection must be given in the right half of the AC; this duration is in units of 1/4 second. The various fields of AC are interpreted as follows:

Bits	*Octal*	*Meanings of audio switch connection fields*
0	400000,,0	Temporary/permanent flag. A 0 in this bit specifies the permanent audio switch connection; a 1 means the temporary connection.
1	200000,,0	Set/get flag. If this bit is 0, the connection indicated by bit 0 will not be changed; the connection status will simply be returned in AC (with bit 0 unchanged, bits 1:4 zero, bits 5:17 containing the data indicated below, and bits 18:35 containing the time remaining in any temporary connection's duration, or 0 for infinite, even if getting permanent connection status). A 1 in this bit means the connection is to be changed.
2:3	140000,,0	Action taken if there is a current temporary connection (applies only if setting new connection): 0 Wait for any existing finite temporary connection to expire. An infinite temporary connection in progress will be flushed. 1 If making a temporary connection and there is already a finite temporary connection, don't change the connection; if there is an infinite temporary connection, it is flushed and the new connection is made. If making a permanent connection, same as 2 below. 2 Make this connection now regardless of former status. Any current temporary connection is flushed immediately. 3 Same as 2.
4	20000,,0	Return-immediately flag. If this bit is 0 and a finite temporary connection is to be made, the UUO will not return until the new connection expires. A 1 in this bit makes the UUO return immediately, possibly after waiting for an old temporary connection to expire--see bits 2:3 above.
5:6	14000,,0	Paging disposition (bit 5 is ignored for a permanent connection): 0 Interrupt. 1 Don't interrupt. 2 Interrupt and extend duration (only for temporary connection).

7:8	3000,,0	Beep disposition (bit 7 is ignored for a permanent connection): 0 Interrupt. 1 Don't interrupt. 2 Interrupt and extend duration (only for temporary connection). 3 Delay beep (only for temporary connection).
14:17	17,,0	Audio switch channel.
18:35	0,,777777	Duration in 1/4 second units, or 0 for infinite (temporary connections only).
9:13	760,,0	The original contents of this field are ignored, but if bit 1 is 0 (getting status), these bits are used to return the following status information:
9	400,,0	Temporary connection active. This bit will be a 1 if you have a current temporary connection. This bit is returned whether you are getting the status of a temporary connection or of a permanent connection. If you are getting the temporary connection status and this bit is returned as 0 (no temporary connection), then only bits 9:12 (740,,0 bits) will return significant data.
10	200,,0	Page in progress. This bit will be 1 whenever a paging announcement is being made, whether or not your display is allowing page interruptions.
11	100,,0	Paging interruption in progress. This bit will be 1 whenever a paging announcement is in progress and you are enabled for page interruptions.
12	40,,0	Beep interruption in progress. This bit will be 1 whenever a beep is happening on your display.
13	20,,0	Delayed beep pending. This bit will be 1 if you have selected the delayed beep disposition and there is a beep interruption waiting for the expiration of your temporary selection.

```
BEEP            [OP=047, ADR=400111]  CALLI 400111
-------------------------------------------------
        MOVE AC,[<TTY line number, or -1 for self>]
        BEEP AC,
```

The BEEP UUO causes a beep interruption for a specific display terminal. The AC should contain the display's line number, or -1 to beep your own display. The beep may happen at once, be delayed, or be ignored altogether, depending on the recipient's audio switch connection status. This UUO returns at once in any case, giving no indication of what happened.

If the terminal being beeped is not a display, a control-G (bell) is sent to the terminal instead of the beep.

SECTION 5

UPPER SEGMENTS

Programs may be split into two discontiguous parts. The first part goes from user address zero to an address called the job's protection constant. This address, whose low order 10 bits are always 1777, is contained in the word at JOBREL in the job data area (see Appendix 4). The second part, if it exists, starts at user address 400000 and goes up to the program's second protection constant, which is kept in the right half of JOBHRL in the job data area. This second part of a program, when it exists, is called the *upper segment, second segment* or *high segment* of that job. The first part is usually called the *lower segment* and is the controlling job. An upper segment cannot execute code except when attached to a lower segment.

An upper segment can be shared by several jobs; this saves core by eliminating all but one copy of the same piece of code. However, it uses up an extra job slot because each upper segment is given a separate job number.

Since upper segments are sometimes shared, they can be write protected to prevent any job from changing the code and/or data in a segment, which, after all, may be part of another job. Write protection is just an option, however, and shared segments are not required to be protected. The SETUWP UUO is used to change an upper segment's write protection status. The sign bit of JOBHRL will be on when your upper segment is write protected.

Another use of upper segments involves having several of them which are attached by a lower segment one at a time and detached when the next one is needed. For a job to be run, it must be entirely in core, including its attached upper segment, if any.

Upper segments have a protection scheme similar to that used on the disk. Each upper segment has a nine bit protection key which indicates who may use that segment and how they may use it. The nine bits are in three groups of three bits each. The first group tells what the creator PPN may do with the segment, the second group tells what others with the same project as the creator may do, and the third group tells what anyone else may do. Within each group, the first bit is unused, the second bit is read protection and the third bit is status change protection. Read protection prevents you from attaching to the segment; status change protection prevents you from changing the name, write protection status, core size, or protection of the segment.

5.1 Making and Killing Segments

There are three ways you can become attached to an upper segment. You can run an SSAVEd program (i.e., one that was saved with its upper segment), in which case you will get the segment that was attached to the program when it was saved; or you can attach to an already existing upper segment; or you can create a new upper segment.

Every job, including upper segments, has a list of credentials. These include the job name, the project-programmer name of the source dump file of the current program, the physical and logical names of the device the dump file was on, the creation date of the dump file and the protection. The protection for a lower segment will always be 000 unless it has been changed by the SETCRD UUO (see page 88), which can also be used to set the protection and creation date for an upper segment. When you cause a new upper segment to be created, its credentials are copied from your job. For a given job, all of the credentials except the protection are set from their values for the dump file which holds the current program. If the dump file was SSAVEd, then the upper segment will be initialized to the same protection it had when it was saved. The lower segment is set up with protection 000.

Let me explain this a bit further with some examples. If you run a system program, your job name will be the file name of the dump file on [1,3], your job PPN (not to be confused with your logged-in PPN) will be 1,3, your job physical device name will be DSK, your logical device name will probably be null, and your job creation date will be the creation date of the dump file. If you run a user program from, say, the disk area [ABC,DEF], then all this stuff will be the same except that your job PPN will be ABC,DEF.

The LINKUP UUO is used to search the system for an upper segment with credentials that match those of your job. The SETPRO UUO (see page 81) can be used to set an upper segment's protection. The SETCRD UUO (see page 88) can be used to set the creation date and protection either for a lower segment or for an upper segment.

When you are finished with an upper segment, you should kill it. This means that it will go away (giving up its job number) unless someone else is still using it.

The following UUOs are used to create, kill, attach and detach upper segments and to change the size of an upper segment.

LINKUP [OP=047, ADR=400023] CALL1 400023

```
LINKUP
<error return>
```

The LINKUP UUO attempts to find an already existing upper segment with the same job name, date of creation, and other credentials as your job has. (The list of credentials required for an upper segment to match your job is given above.) If such an upper segment is found which is not protected from you, it is attached to your job and the skip return is taken. If there is no such upper segment, you get the direct (error) return. Any segment you were attached to when you gave this UUO is killed before all this happens.

REMAP [OP=047, ADR=37] CALLI 37

```
MOVE  AC,[<write-protect flag>,,<highest address in lower>]
REMAP AC,
<error return>
```

The REMAP UUO causes your core image to be broken into two segments. The address contained in the AC right is taken as the address of the last word to be in the lower segment. The next word becomes the first word in the upper segment and its address becomes 400000. If the sign bit of the AC is on when this UUO is given, the upper segment will be write protected.

Before your core image is broken, an automatic LINKUP is attempted in order to find an already existing upper segment with your credentials; see the LINKUP UUO above. If one is found, the part of your core image that would otherwise have become your upper segment is discarded and your core size reduced appropriately before attaching to the already existing upper segment.

If this UUO is successful, the skip return is taken and the job number of your upper segment is returned in AC. If the automatic LINKUP fails and there are no more job numbers left to create a high segment under, or if there is something illegal in your specifications, the direct (error) return is taken.

Any upper segment you are attached to when you give the REMAP UUO is killed before anything else is done.

CORE2 [OP=047, ADR=400015] CALLI 400015

```
MOVEI AC,<highest upper segment address desired>
CORE2 AC,
<error return>
```

The CORE2 UUO is used to change the size of your upper segment. The address in the AC is interpreted as the highest address you want in your upper segment; this address, if non-zero, is ORed with 1777 and the 400000 bit is ignored. If the AC contains zero, any upper segment you have will be killed (unless it is protected from you, in which case it will simply be detached from your job) and the skip return will be taken.

If the AC contains a non-zero number and you do not have an upper segment, an upper segment of the specified size will be created for you. If you already have an upper segment, its size is adjusted to that specified by the number in the AC. If this UUO is successful, the skip return is taken. If there is not enough core to grant your request, or if the segment is protected from you, the direct (error) return is taken. Unless you are killing your upper segment (with a zero in AC), this UUO returns with the AC containing the total number of 1K blocks available to a single user program, counting both upper and lower segments.

ATTSEG [OP=047, ADR=400016] CALLI 400016

```
        MOVE   AC, [<job number or name>]
        ATTSEG AC,
        <error return - code in AC>
```

The ATTSEG UUO is used to attach to an upper segment that already exists. You must not already have an attached upper segment. The AC should contain either the sixbit job name or the job number of the upper segment to which you wish to attach. If this UUO is successful, the skip return is taken. Otherwise the direct (error) return is taken and a code is returned in the AC indicating the cause of failure. The error codes and their meanings are explained below.

Error Code	*Meaning*
0	A protection violation has occurred; you are not allowed to attach to this upper segment.
1	There are two or more upper segments with the job name you gave. The job number of the first one is returned in the left half of AC.
2	The job number you gave is not the job number of an upper segment.
3	There is no job with the name you gave.
4	You already have an upper segment.

DETSEG [OP=047, ADR=400017] CALLI 400017

```
        DETSEG <flag>,
```

The DETSEG UUO detaches your upper segment from your job. Normally your upper segment is placed into a list of the segments you have detached, so that when you do a RESET (see page 126), all your detached segments will go away. However, if the low order bit of the AC field is a one, then the segment will not go into the list, but will stick around like any other detached job. This is not recommended because it can result in the proliferation of unused upper segments. Only the low order bit of the AC field is looked at by this UUO.

If you have a simulated upper segment (see the SETPR2 UUO on page 92), it is killed by this UUO.

5.2 Getting/Setting Segment Status

The following UUOs are used to find out and/or change the protection, name and other information associated with an upper segment.

SETUWP [OP=047, ADR=36] CALLI 36

```
        MOVEI  AC,<zero for unprotect, non-zero for protect>
        SETUWP AC,
        <protection violation return>
```

The SETUWP UUO is used to write protect or unprotect your attached upper segment. If AC contains zero, the segment becomes unprotected; otherwise it becomes protected. If this UUO is successful, the skip return is taken. If the segment is protected from you, then you get the direct (error) return. If you have no upper segment, you always get the skip (success) return. The sign bit of JOBHRL in the job data area is a one if your upper segment is write protected.

UNPURE [OP=047, ADR=400102] CALLI 400102

```
        UNPURE
        <error return>
```

The UNPURE UUO is used to unprotect your upper segment. If you are sharing a write-protected upper segment with other users, this UUO will create an unprotected copy of that upper segment (assigning it a new job number), detach you from the old segment and attach you to this new segment. If you are the sole user of a write-protected upper segment, this UUO will simply unprotect that segment. The skip return will be taken upon success, at which time your upper segment will not be write protected. If there are no job numbers available for a copy of your upper segment, or if you cannot be granted enough core, the direct (error) return will be taken. If you have no upper segment, or if your upper segment is not write protected, you always get the skip (success) return.

SETPRO [OP=047, ADR=400020] CALLI 400020

```
        MOVE   AC,[<Bits 0:8 = new prot key; bits 30:35 = job no.>]
        SETPRO AC,
        <error return>
```

The SETPRO UUO can be used to change the protection key of any upper segment not protected from you. Bits 30:35 (0,,77 bits) in the AC should contain the job number of the upper segment whose protection you wish to change, where zero means your own attached upper segment; bits 0:8

of the AC should contain the new protection key you wish the segment to have. If this UUO is successful, the skip return is taken. If a protection violation occurs or if the job indicated is not an upper segment, the direct (error) return is taken.

SETNM2 [OP=047, ADR=400036] CALLI 400036

```
        MOVE   AC,[<sixbit name for your upper segment>]
        SETNM2 AC,
        <error return>
```

The SETNM2 UUO is used to change the job name of your upper segment. The name you wish your segment to have should be in the AC when you give this UUO. If your segment is successfully renamed, the monitor then scans the names of other upper segments in the system, and if there is one with the same name as yours, its job number is returned in the AC; if there is no other upper segment with the same name, zero is returned in the AC. The skip return is taken on success. If you are not permitted to change your upper segment's name, the direct (error) return is taken. If you have no segment attached, the skip (success) return is always taken.

POINTS [OP=712]

```
        POINTS ADR

ADR:    <block =63 words long for returned job numbers>
```

The POINTS UUO returns a list of the job numbers of all jobs, including your own, which are attached to your upper segment. This list is returned in the block pointed to by the effective address of the UUO, with one job number per word. The end of the list is indicated by a zero. This list can be up to =63 words long.

SEGNAM [OP=047, ADR=400037] CALLI 400037

```
        SEGNAM AC,
```

The SEGNAM UUO returns in AC the sixbit job name of your upper segment. If you have no upper segment attached, zero is returned.

SEGNUM [OP=047, ADR=400021] CALLI 400021

```
        MOVEI  AC,<job number>
        SEGNUM AC,
```

The SEGNUM UUO gets the job number of the upper segment attached to the job whose number is in AC. The segment number is returned in AC. A zero in AC gets your own segment number. A zero returned means the specified job has no upper segment attached.

SECTION 6

GETTING/SETTING INFORMATION

This section describes numerous UUOs that allow you to get certain types of information from the system and to change some of this information regarding your job.

6.1 Dates and Times

Here are some UUOs to get various flavors of date and time.

DATE [OP=047, ADR=14] CALLI 14

```
        DATE AC,
```

The DATE UUO returns in AC the current date in system date format. The number returned has the following value: ((year-1964)*12+month-1)*31+day-1, where all these numbers are in decimal. You can calculate the day, month and year by dividing. If you divide by =31, the remainder is then day-1. Then if you divide the quotient by =12, the new remainder is month-1. Finally, if you take the quotient again and add =1964, you get the year.

DAYCNT [OP=047, ADR=400100] CALLI 400100

```
        MOVE   AC,[<date in system date format>]
        DAYCNT AC,
```

The DAYCNT UUO converts a date from system date format (see the DATE UUO above) to the number of days from 1 January 1964 to the date indicated. AC should contain the date of interest, where zero or a negative number is taken to mean today's date. The corresponding day count is returned in AC.

TIMER [OP=047, ADR=22] CALLI 22

```
        TIMER AC,
```

The TIMER UUO returns in AC the time of day in 60ths of a second after midnight.

MSTIME [OP=047, ADR=23] CALLI 23

```
        MSTIME AC,
```

The MSTIME UUO returns in AC the time of day in milliseconds after midnight. This time is accurate only to the nearest 60th of a second.

ACCTIM [OP=047, ADR=400101] CALLI 400101

```
        ACCTIM AC,
```

The ACCTIM UUO returns the current date and time. The date is returned in the left half of AC and is in system date format (see the DATE UUO above). The time is returned in the right half of AC and is in seconds after midnight.

DSKTIM [OP=047, ADR=400072] CALLI 400072

```
        DSKTIM AC,
```

The DSKTIM UUO returns in AC the current date and time in file time/date written format: bits 13:23 (37,,770000 bits) hold the time in minutes after midnight, and bits 24:35 (0,,7777 bits) hold the date in system date format (see the DATE UUO above).

RUNTIM [OP=047, ADR=27] CALLI 27

```
        MOVE   AC, [<job number>]
        RUNTIM AC,
```

The RUNTIM UUO returns in AC the total compute time since login used by the job whose number is in the AC. A zero job number in the AC will get the compute time for your own job. The time returned is in milliseconds although it is kept by the system in 60ths of a second and is not even exact to that accuracy. An illegal job number specified will cause zero to be returned.

6.2 Job Information

Here are some UUOs to find out things about specific jobs and even to set certain values for your own job.

CORE [OP=047, ADR=11] CALLI 11

```
        MOVEI AC,<highest address you want in your lower segment>
        CORE  AC,
        <error return>
```

The CORE UUO is used to change your core size (the size of your lower segment if you have a two segment program). AC should contain the highest address (in your lower segment) that you want to be able to reference. This address, if non-zero, is ORed with 1777 (to make it a 1K boundary), and then your core size is adjusted, if necessary, to the size indicated. If you can be given the amount of core you request, the skip return is taken; if not, the direct (error) return is taken. In any case, the maximum number of 1K blocks a single program is allowed to use, counting both upper and lower segments, is returned in the AC. If the AC originally contains zero, then no change is made in your core size, but the number of 1K blocks available to a single program is returned in the AC and the direct (error) return is taken.

PJOB [OP=047, ADR=30] CALLI 30

```
        PJOB AC,
```

The PJOB UUO returns your job number in the AC.

GETPPN [OP=047, ADR=24] CALLI 24

```
        GETPPN AC,
```

The GETPPN UUO returns in AC the logged-in project-programmer name of your job.

GETNAM [OP=047, ADR=400062] CALLI 400062

```
        MOVEI  AC,<job number>
        GETNAM AC,
```

The GETNAM UUO is used to get the name of any job on the system. AC should contain the number of the job whose name you wish to know. If AC contains zero, a negative number, or an illegal job number, then your job is assumed. The sixbit name of the job specified is returned in AC.

SETNAM [OP=047, ADR=43] CALLI 43

```
        MOVE   AC,[<sixbit job name>]
        SETNAM AC,
```

The SETNAM UUO is used to change your job name to that given in the AC. Any job name is legal.

SETCRD [OP=047, ADR=400073] CALLI 400073

```
        MOVE   AC,[<new protection and creation date>]
        SETCRD AC,
```

The SETCRD UUO is used to set the protection and creation date of either your lower segment or your upper segment. The new protection and creation date are taken from the AC specified in the UUO. If any of bits 0, 3 and 6 (444000,,0 bits) are ones, the protection and creation date of your upper segment are set; otherwise the protection and creation date of your lower are set. Bits 0:8 specify the protection, bits 13:23 the time of creation (in minutes after midnight) and bits 24:35 the date of creation (in system date format). Bits 0, 3 and 6 (444000,,0 bits) are turned off before the protection is stored. If bits 13:35 are all zero, the current time and date will be used. The protection and creation date are used mainly in conjunction with linking to or creating an upper segment; see Section 5 on upper segments.

SETPRV [OP=047, ADR=400066] CALLI 400066

```
        MOVE   AC,<privilege bits you want>
        SETPRV AC,
```

The SETPRV UUO is used to find out and/or change your privileges. AC should contain either -1 or the privilege bits you want. If AC contains -1, then your privileges will not be changed. Otherwise, an attempt will be made to set your privilege bits to those indicated in AC. New privilege bits will be granted only if either 1) you currently have the privilege privilege (bit 0--the 400000,,0 bit represents the privilege privilege) or 2) JBTSTS indicates that you are an accounting program with JACCT set. However, the system will be glad to turn off the bits for any privileges you wish to surrender. Under any circumstances, the resultant settings of your privilege bits will be returned in AC. For the meanings of the various privilege bits, or to request privileges, see any system programmer.

```
SLEVEL          [OP=047, ADR=400044]  CALLI 400044
-------------------------------------------------
        MOVEI   AC,<job number>
        SLEVEL AC,
```

The SLEVEL UUO is used to find out a job's current service level. AC should contain the number of the job whose service level you want to know; a zero job number means your own job. The service level (in percent) of the job indicated will be returned in the left half of AC; the right half will contain the job number. If you specify an illegal job number, zero will be returned in AC.

```
RLEVEL          [OP=047, ADR=400054]  CALLI 400054
-------------------------------------------------
        MOVEI   AC,<programmer name>
        RLEVEL AC,
```

The RLEVEL UUO is used to find out how much service level is reserved for the current hour by a particular programmer name. The programmer name should be in the right half of AC; the service level (in percent) is returned in the left half of AC. The right half of AC is unchanged by this UUO unless the reserved service level is zero, in which case zero is returned in the whole AC. The original value of AC left is ignored by this UUO.

6.3 Looking at the Monitor

Here are some UUOs used to examine various parts of the monitor.

```
NAMEIN          [OP=047, ADR=400043]  CALLI 400043
-------------------------------------------------
        MOVE    AC,[<sixbit job name>]
        NAMEIN AC,
        <error return - code in AC>
```

The NAMEIN UUO is used to determine if there are any jobs in the system with a particular job name. AC should contain the job name you are interested in. If there is exactly one job with the given name, the skip (success) return is taken and the job number of the job with that name is returned in the AC. Otherwise, the direct (error) return is taken and a code is returned in AC; a code of 1 means that there is no job with the given name, and a code of 3 means that there are two or more jobs with that name.

JBTSTS [OP=047, ADR=400013] CALLI 400013

```
        MOVEI  AC,<job number>
        JBTSTS AC,
```

The JBTSTS UUO is used to get from the system the job status word for a particular job. AC should contain the number of the job of interest, where zero means your own job. The table below gives the meanings of some of the bits in this word.

Bits	*Octal*	*Name*	*Meanings of 1's in the job status word*
0	400000,,0	RUN	The job is runnable, though it may be in a wait state of some kind. This bit gets turned off by typing control-C, giving the EXIT UUO, or hitting some kind of error.
1	200000,,0	CMWB	The job is waiting to be swapped in to service a monitor command.
2	100000,,0	JACCT	LOGIN or LOGOUT is running; control-C cannot be typed at this time.
3	40000,,0	JNA	A job number has been assigned to this job.
4	20000,,0	JERR	The job has hit an error and cannot be continued.
5	10000,,0	JLOG	The job is successfully logged in. System phantom jobs (see the WAKEME UUO on page 132) run with this bit off as do temporary jobs (project-programmer name of 100,100) started up by monitor commands (like WHO) that need a job but which do not require you to be logged in. A job with the JLOG bit off will go away if it hits an error (such as a parity error or illegal memory reference) or if a monitor command is typed to it.
6	4000,,0	SHF	The job is currently being shuffled in core.
7	2000,,0	SWP	The job is swapped out.
8	1000,,0	JSEG	The job is really an upper segment.

17	1,,0	JWP	This upper segment is write protected. This bit is meaningful only if bit 8 is on, that is, only if this job is an upper segment.
20	0,,100000	JLOCK	The job is locked in core by the LOCK UUO; see page 133.
23	0,,10000	FBINP	The job has a fast band transfer in progress. See Section 10.
24	0,,4000	FBERP	The job had an error on the last fast band transfer. See Section 10.
30:35	0,,77		Job number of this job's upper segment, if any; this field is zero if the job has no upper.

SWITCH [OP=047, ADR=20] CALLI 20

```
        SWITCH AC,
```

The SWITCH UUO returns in AC the current setting of the PDP-10 console data switches.

CALLIT [OP=047, ADR=400074] CALLI 400074

```
        MOVE   AC,[<opcode, CALLI number or UUO mnemonic>]
        CALLIT AC,
```

The CALLIT UUO is used to find out the opcode corresponding to a given UUO mnemonic or to find out the mnemonic for an opcode or CALLI number. AC should contain the opcode, CALLI number or sixbit mnemonic of the UUO you are interested in. The result is returned in AC: for UUO mnemonics, the opcode is returned (i.e., a full 36-bit instruction including relevant AC or address fields); for opcodes the most specific sixbit mnemonic is returned (e.g., opcode 051000,,0 returns 'INCHRW' and 051040,,0 returns 'OUTCHR'), unless bit 17 (1,,0 bit) was on originally in the AC, in which case the generic mnemonic is returned (e.g., opcode 051001,,0 returns 'TTYUUO'); for CALLI numbers, the sixbit CALL name is returned (e.g., 0,,400003 returns 'SPCWGO'). If the given mnemonic, opcode or CALLI number is undefined, zero is returned.

This UUO works by first checking bits 13:16 (36,,0 bits) in the AC. If these bits are all zero, the argument is assumed to be an opcode; if any of these bits is non-zero, the argument is assumed to be a sixbit mnemonic. Thus any one- or two-character mnemonic will be mistaken for an opcode; however all UUO mnemonics are three or more characters. Also, all irrelevant fields in the argument must be zero to avoid confusion.

```
SETPR2          [OP=047, ADR=400052]  CALLI 400052
-------------------------------------------------
        MOVE    AC,[<prot>,,<reloc>]
        SETPR2 AC,
        <error return>

(Low order bit of <prot> on means write protect "upper segment;"
low order bit of <reloc> on means <reloc> is in relative mode.)
```

The SETPR2 UUO sets your second protection/relocation register in order to simulate the possession of an upper segment. There are two purposes for doing this: the first is to allow you to look at any part of core, particularly at the monitor, efficiently; the second purpose is to enable your job to address part of your core image as if it were in an upper segment (addresses over 400000) even though it isn't. The table in Appendix 5 tells where in the monitor you can find various interesting pieces of system information which you can access by using this UUO.

Note: Any attached upper segment (real or simulated) that you have will be killed when you give this UUO. See Section 5 on upper segments. Also, both the RESET UUO (see page 126) and the DETSEG UUO (see page 80) undo the effect of SETPR2.

At the time this UUO is called, AC right should contain the relocation you wish to simulate and AC left should contain the protection you wish to simulate as an upper segment. Furthermore, if the low order bit of AC left (bit 17--the 1,,0 bit) is on, your "upper segment" will be write protected; and if the low order bit of AC right (bit 35--the 0,,1 bit) is on, the relocation specified will be assumed relative to your core image--that is, your own true relocation constant will be added in to <reloc> before setting the second prot/reloc register. Upon success, this UUO takes the skip return; if your request specifications are impossible to satisfy, then the direct (error) return is taken.

If you give an absolute <reloc> and you are not privileged, your "upper segment" will automatically be write protected. Finally, the system will adjust the values you specify in AC to 1K boundaries; for <reloc> the low order =10 bits (0,,1777 bits) will be turned off, and for <prot> the low order =10 bits (1777,,0 bits) will be turned on. Thus, a <prot> of 4321 will be made into 5777, and a <reloc> of 3210 will be made into 2000, with all this happening after the system has taken note of the low order bits of both AC left and AC right.

Now, if you still don't understand (and even if you do), let me explain further. Suppose you wish to look at certain locations in the monitor (for whatever reason). You can use this UUO once and then do simple MOVEs (or their equivalent) to get the information you want. For instance, if you would like to put into AC whatever is in the system at EXEC location 220, you can execute the following sequence of instructions.

```
        MOVSI   AC1,377777      ;<reloc> = 0, <prot> = 377777
        SETPR2  AC1,            ;make the first 128K of core
                                ;  into your "upper segment"
        HALT                    ;halt on error return
        ...
        MOVE    AC,400220       ;get whatever is in EXEC 220
        ...
```

The relative mode use of this UUO allows you to write code as if it were going to run as a second

segment, and then to execute it without making it into a second segment, provided you have used this UUO with the relative mode bit set. You could even do overlays by reading another piece of code, also written to run as a second segment, into the same place. The base address of the second segment code, however, should be on a 1K boundary or you might get confused about what is happening.

GETPR2 [OP=047, ADR=400053] CALLI 400053

```
        GETPR2 AC,
```

The GETPR2 UUO is used to fetch the protection/relocation of your simulated upper segment. The protection is returned in the left half of AC; bit 17 (1,,0 bit) is on if access to the segment is write protected. The relocation is returned in the right half of AC; bit 35 (0,,1 bit) is on if the relocation is in relative mode. See SETPR2 above for an explanation of simulated upper segments.

If you do not have a simulated second segment at the time you call this UUO (for example, if you have a real second segment), then zero is returned in AC.

PEEK [OP=047, ADR=33] CALLI 33

```
        MOVEI AC,<absolute address you want to look at>
        PEEK  AC,
```

The PEEK UUO is used to get the contents of any absolute location in memory. AC should contain the absolute address you wish to examine. The contents of that address will be returned in AC. Appendix 5 tells where you can find some interesting system information in the monitor.

This UUO has been largely replaced by the SETPR2 UUO (explained above), which makes examining memory outside your core image much more efficient. However, if you have an upper segment, you must detach it to use the SETPR2 UUO but not to use the PEEK UUO.

SECTION 7

INTER-JOB MAIL SYSTEM

The inter-job mail system provided by the monitor allows =32 word letters to be passed between jobs. Each job in the system has a mailbox which can hold exactly one =32 word letter. For a letter to be sent, the sending job identifies the destination job by either the job number or the sixbit job name. This causes the letter to be placed in the mailbox of the destination job, who can then take the letter out of his own mailbox (i.e., receive the letter) whenever he wants. While a job's mailbox is full (holding a letter he hasn't read yet), no one can send that job a letter.

Note: The RESET UUO (see page 126) will cause any letter in your mailbox to be thrown away.

MAIL [OP=710]

```
        MAIL <function>,ADR
```

The MAIL UUO is an extended UUO that uses the AC field to determine which of several mail-handling functions is to be executed. Each of these functions is described separately below. Notice that MAIL is an IOT UUO and hence cannot be given by a program that is currently in USER-IOT mode (which is explained in Appendix 3).

7.1 Sending Mail

The following two UUOs allow you to send a letter to any job.

SEND [OP=710, AC=0] MAIL 0,

```
        SEND ADR
        <error return>

ADR:    <destination job name or number>
        <address of =32 word letter to be sent>
```

The SEND UUO is used to send a letter to any job in the system. The effective address of the UUO should point to a two-word block. The first word of this block should be the job number or the sixbit job name of the job to which the letter is to be sent. The second word of the block should contain the address of the =32 word letter.

If the letter is successfully sent, the skip return is taken. If the destination job already has a letter in his mailbox (meaning the letter cannot be sent at this time), the direct (error) return is taken. If there is no job with the name or number you give, you get the system error message NON-EX JOB NAME OR NUMBER. If there are two or more jobs with the job name you give, you get the system error message AMBIGUOUS JOB NAME. With either of these last two errors, your program will be stopped and you will be permitted to type CONTINUE, which will cause this UUO to be tried again.

```
SKPSEN          [OP=710, AC=5]  MAIL 5,
-------------------------------------------------
        SKPSEN ADR
        <return for destination mailbox full>
        <return for letter successfully sent>
        <return for non ex job name or number, or ambiguous name>

ADR:    <job name or number>
        <address of =32 word letter to be sent>
```

The SKPSEN UUO is used to send a letter to another job just as the SEND UUO (see above) does except that there is an extra return, which is taken when there is no job with the given name or number or when there are two or more jobs with the given name. Thus, there are three possible returns that this UUO can take. The direct return (no skip) is taken if the letter cannot be sent because the addressee already has a letter in his mailbox. The skip return is taken if the letter is successfully sent. The double skip return is taken if there is no job with the given name or number or if there are two or more jobs with the given name.

7.2 Receiving Mail

The following two UUOs allow you to receive mail sent to you, that is, to have a letter removed from your mailbox and deposited in your core image.

```
WRCV            [OP=710, AC=1]  MAIL 1,
-------------------------------------------------
        WRCV ADR

ADR:    <block =32 words long to receive a letter>
```

The WRCV UUO takes the letter, if any, that is in your mailbox and places it in the =32 word block specified by the effective address of the UUO. If there is no letter in your mailbox, this UUO waits until someone sends you one and then gives it to you.

```
SRCV            [OP=710, AC=2]  MAIL 2,
-------------------------------------------------
        SRCV ADR
        <return if no letter is in your mailbox>
        <success return>

ADR:    <block =32 words long to receive a letter>
```

The SRCV UUO checks to see if there is a letter in your mailbox. If there is one, it is returned to you in the =32 word block pointed to by the effective address (ADR) of this UUO and the skip return is taken. If there is no letter in your mailbox, the direct return is taken and the block at ADR is untouched.

7.3 Peeking at Mailboxes

The following two UUOs allow you to find out whether a job has a letter in its mailbox.

```
SKPME           [OP=710, AC=3]  MAIL 3,
-------------------------------------------------
        SKPME
        <return for your mailbox empty>
```

The SKPME UUO tells you whether or not there is a letter in your mailbox. If there is a letter there, the skip return is taken; if not, the direct return is taken.

```
SKPHIM          [OP=710, AC=4]  MAIL 4,
-------------------------------------------------
        SKPHIM ADR
        <return for his mailbox empty>

ADR:    <name or number of job you are interested in>
```

The SKPHIM UUO is used to find out if a given job has a letter in his mailbox. The job number or sixbit job name of the job of interest should be in the word pointed to by the effective address of this UUO. If that job has a letter in his mailbox, the skip return is taken; if his mailbox is empty, the direct return is taken. If there is no job with the name or number given, you will get the system error message NON-EX JOB NAME OR NUMBER. If there are two or more jobs with the job name given, then you will get the system error message AMBIGUOUS JOB NAME. If either of these two errors occurs, your program will be stopped and you will be permitted to type CONTINUE, which will cause this UUO to be tried again.

SECTION 8

SPACEWAR MODE

In a timesharing system the available CPU time must be split up among all the programs that are trying to run. Any one program will be run only for a short period of time, then stopped for a while to let other programs run, then run a little more, etc. The intervals between, and durations of, the times when a program is allowed to run are generally irregular and depend on the system load. Certain programs require fairly regular service (in the form of CPU time allocated) in order to operate meaningfully. The system provides spacewar mode to assure regular service to such programs.

To use spacewar mode, a job tells the system the starting address of the spacewar module (process) and how often and on which processor(s) (PDP-10, PDP-6) it should be run. A spacewar module is a separate process from your job's main process (the one that initiates the spacewar module) but runs in the same core image. The spacewar module will be restarted at a fixed interval after it last stopped; you specify this interval when you initiate the module. A spacewar process cannot quite be guaranteed of running every so often because, for example, another spacewar process on the same processor could have conflicting time demands. After you have initiated a spacewar module, your job's main process can continue doing whatever it wants. You are allowed to have one spacewar module active on each processor; i.e., you can have one on the PDP-10 and another one on the PDP-6.

While you have a spacewar module active, your job usually will not be swapped out although it may be shuffled to a different place in core. Before your job is either shuffled or swapped out, your spacewar module will be warned that it is not going to be run for a while; so it can take whatever precautions are necessary to see that nothing bad happens while it is away.

Each time a spacewar process is started up, it is allowed to run until either it signals by the DISMIS UUO (see page 102) that it is done or it times out. Normally a spacewar process will time out if it runs for more that half a second during a single activation. If you set the timeout-suppression bit (see the SPCWGO UUO below) for a spacewar process, then that process will never time out. However, running for very long (like more than a few milliseconds) will cause system performance to deteriorate noticeably, especially if the process is running on the PDP-10! In fact, a spacewar module running on either processor for more than about half a second will cause the other processor to think that the first processor is dead.

If a spacewar process makes an illegal or non-existent memory reference, or if it gets a push-down overflow, then the message SPACEWAR LOSSAGE will be typed on the job's terminal and both the spacewar process and the main process will be stopped. If you try to initiate a spacewar process when one is already active, or if you indicate that a spacewar process should be run on the PDP-6 and the PDP-6 is not running, or if one of your spacewar processes times out, then you will get an error message and your spacewar processes will be killed.

Spacewar modules are started in IOT-USER mode; this means that operation codes 700:777 are machine I/O instructions rather then UUOs. Thus a spacewar process can do its own I/O directly;

however, it should make sure that its use of I/O devices will not conflict with the system. For more information on IOT-USER mode, see its description in Appendix 3.

Spacewar modules running on the PDP-6 can never do UUOs. Any attempt by such a process to do a UUO will result in termination of that run (as by DISMIS). Spacewar modules running on the PDP-10 are allowed to do certain UUOs. However, a UUO that attempts to reference any accumulators will never access the correct set of ACs (whether the AC is referenced as an AC or as a memory location); and a UUO that returns results in the ACs may in fact return the results in the ACs of the job's main process (if the main process is at UUO level), thus clobbering whatever the main process had in its AC(s)! Furthermore, any UUO that must wait for something to happen will not work from a spacewar module. Finally, some illegal UUOs will cause the SPACEWAR LOSSAGE message to be printed and the spacewar modules to be killed. With those warnings in mind, note that spacewar modules on the PDP-10 can in general do any of the IOT UUOs, that is, those with opcodes over 700 (including the display UUOs) but not until the process gets itself out of IOT-USER mode!! See the preceding paragraph.

Each time a spacewar process is started up, the system loads up several accumulators with data that might be needed by the spacewar module. The ACs set up and the data they contain are listed below.

AC	*Contents*
1	The current value of the spacewar buttons. See the SPWBUT UUO on page 134.
2	Your current protection-relocation constant. Your protection constant is in the left half and your relocation constant is in the right half of this AC.
3	A warning value. This AC usually contains zero but is set up with -1 if this is the last time your spacewar process will be run for a while (because your job is being swapped out or shuffled). The next time your spacewar module runs, this AC will contain the number of 60ths of a second for which your spacewar module was suspended.
4	The number of the processor this spacewar module is running on. This number is 1 for the PDP-10 and 2 for the PDP-6.
5	A flag indicating the status of the processor that this spacewar process is *not* running on. This flag is zero if that processor is running (normal state) and -1 if that processor is dead.
6	Your job status word. See the JBTSTS UUO on page 90. If the run bit (bit 0--the 400000,,0 bit) of this word is zero, then your main process has stopped for some reason; for instance, control-C may have been typed.

If you initiate a spacewar process with the time between runs set to zero, then the process will be run only once.

Whenever you do a RESET (see page 126), any spacewar modules active will be killed. The EXIT UUO (see page 125) will also kill any spacewar modules you have. Finally, the SPCWAR UUO can also be used to kill your spacewar modules; see below.

8.1 Spacewar UUOs

Here are the UUOs used to initiate and to kill spacewar processes and to terminate spacewar activations.

```
SPCWAR          [OP=043]
-------------------------------------------------
        SPCWAR <number of ticks between startups>,<starting address>
```

The SPCWAR UUO initiates a spacewar process on the PDP-6. If the PDP-6 is not running, the spacewar process will be run on the PDP-10 instead. The effective address of the UUO is the process' starting address. The number of 60ths of a second between startups is specified by the AC field of the instruction (possible values of 0:17). If the AC field is zero, then the spacewar process will be run only once. Timeout suppression is not possible with the SPCWAR UUO; any process started with this UUO will time out if it runs for more than half a second during a single activation.

If the effective address of this UUO is 636367 (that's 'SSW' in sixbit) and the AC field is zero, then instead of a spacewar module being initiated, all your spacewar modules will be killed. This is the normal way to kill spacewar modules. The RESET UUO (see page 126) and all flavors of the EXIT UUO (see page 125) will also kill your spacewar modules.

```
SPCWGO          [OP=047, ADR=400003]  CALLI 400003
-------------------------------------------------
        MOVE    AC,[<Bits  0:1  (600000,,0 bits) = processors;
                     bits  2:3  (140000,,0 bits) = timeout suppression;
                     bits 14:17 (000017,,0 bits) = startup interval;
                     bits 18:35 (0,,777777 bits) = starting address>]
        SPCWGO AC,
```

The SPCWGO UUO is used to initiate a spacewar process on either the PDP-6 or the PDP-10 or both. The starting address of the spacewar module should be in the right half of the AC. Bits 14:17 (17,,0 bits) of the AC should contain the time in 60ths of a second between startups of the spacewar module. A zero time means run the spacewar process only once and then kill it. Bits 0:1 (600000,,0 bits) determine which processor(s) will run this module. If bit 0 (400000,,0 bit) is a one, then the module will be run on the PDP-10; if bit 1 (200000,,0 bit) is a one, the module will be run on the PDP-6. If both bits 0 and 1 are one, then both processors will run this module, with each

processor starting at the given starting address. If both bits 0 and 1 are zero, then the module will be run on the PDP-6 unless the PDP-6 is dead, in which case the module will be run on the PDP-10. Bits 2:3 (140000,,0 bits) are used to specify timeout suppression separately for the PDP-10 process (bit 2--100000,,0 bit) and the PDP-6 process (bit 3--40000,,0 bit). If the timeout-suppression bit is off for a process, then that process will time out if it runs for more than half a second during a single activation. The remaining bits in AC (bits 4:13--the 37760,,0 bits) are reserved for future use.

Example: To initiate a spacewar process with starting address WAR to run every 17 ticks on the PDP-10 without timeout suppression, do the following:

```
MOVE   AC,[400017,,WAR]
SPCWGO AC,
```

DISMIS [OP=047, ADR=400024] CALLI 400024

```
DISMIS
```

The DISMIS UUO is used by spacewar processes to terminate individual activations. This UUO causes the process to be stopped until the next time it is supposed to be run, at which time the process will be restarted at its starting address.

This UUO has another (though similar) meaning in the user interrupt system; its meaning there is explained on page 110.

SECTION 9

USER INTERRUPTS

The user interrupt system allows a program to take action upon the occurrence of any of various special conditions, without the program having to test continuously for these conditions. There are two versions of interrupts available--the old style and the new style. The main differences between the two are: 1) while you are processing an old style interrupt you can still be interrupted, which can cause all sorts of trouble, but while you are processing a new style interrupt you cannot receive another interrupt until you dismiss the current one; 2) the only interrupts you can receive with the old system are processor interrupts such as push-down overflow, illegal memory reference and arithmetic overflow. You can also enable for clock interrupts with the old system, but only clock ticks that occur while you are actually running will be seen by your program. All interrupts are possible with the new system; and clock interrupts will happen whether or not you are actually running at the time. Before going into more differences between the old and new style interrupts, I shall explain the basics of the interrupt system and the features that are the same for both styles.

A user program indicates that it wants to use the interrupt system by enabling itself for the particular interrupts that it is interested in. Interrupt conditions that are not enabled for will be handled by the system. For instance, if you get a push-down overflow, and if you are not enabled for push-down overflow interrupts, then you will get the system error message PDL OV. If, on the other hand, you *are* enabled for this interrupt, then you will get an interrupt indicating that you had a push-down overflow. Some interrupt conditions are ignored by the system unless you are enabled for them.

When an interrupt that you are enabled for does occur, your program is stopped, the program counter (PC) and PC flags are saved in JOBTPC in your job data area (see Appendix 4), the cause of the interrupt is saved in JOBCNI and your interrupt handler is started at the address contained in JOBAPR.

The PC that you get in JOBTPC generally points to the next instruction that your main process would otherwise have executed if the interrupt had not occurred. However, there are certain conditions under which the value of this PC is not quite obvious.

First of all, if you were executing a UUO (and hence your PC was in monitor mode while executing some system code for that UUO), then the PC saved in JOBTPC will not be your real (monitor mode) PC that you had at the time of the interrupt; instead JOBTPC will contain the location of the UUO call in your core image, and the user-mode bit (bit 5--the 10000,,0 bit) in the left half of JOBTPC will be *off* to indicate this condition.

Secondly, when you receive a processor interrupt (either old or new style) such as illegal memory reference, the PC saved in JOBTPC will point to the instruction that caused the interrupt. However, if you jump to an illegal location, then the PC returned with the illegal memory reference interrupt will point to the illegal location. For instance, on an AOJA 1,777777, the AC will have been incremented and the PC changed to 777777 before the ill mem ref occurs, so the PC stored in this case would be 777777.

Finally, if an interrupt occurs in the middle of an ILDB or an IDPB instruction after the byte pointer has been incremented but before the byte has been moved, then JOBTPC will point to the byte instruction and the byte-increment suppression flag (bit 4--the 20000,,0 bit), will be on in the left half of JOBTPC. Thus the byte pointer will not be incremented again when (and if) the instruction is resumed.

Each condition for which an interrupt can occur is represented by a specific bit. You enable a given interrupt by setting to one the bit corresponding to that condition; this can be done with various UUOs that will be described in detail later. When you get an interrupt, the bit representing the cause of the interrupt is given to you in the word at JOBCNI. For new style interrupts this word will have exactly one bit on. With old style interrupts there may be some extraneous bits on that do not represent old style interrupts. The word returned in this case is the CONI word from the processor, and the extra bits currently set in this word are the 0,,6043 bits.

The interrupt conditions represented by the different bits are listed below. The bits marked with asterisks (*'s) represent the only conditions for which you can receive interrupts under the old style interrupt system. You are not allowed to enable a given interrupt condition for both old and new style interrupts at the same time.

Note: The RESET UUO (see page 126) clears all of your interrupt enablings.

Bits	*Octal*	*Name*	*Interrupt conditions*
0	400000,,0	INTSWW	Your job is about to be swapped out.
1	200000,,0	INTSWD	Your job has just been swapped back in. If you enable for both INTSWW and INTSWD, then you will receive these two interrupts as a pair in the expected order every time your job is swapped.
2	100000,,0	INTSHW	Your job is about to be shuffled.
3	40000,,0	INTSHD	Your job has just been shuffled.
4	20000,,0	INTTTY	Your user-level job would be activated by TTY input if it were waiting for it. When you are enabled for this interrupt, you will be interrupted every time either a character or a line is typed in, depending on whether you are in character mode or line mode. As long as you do not ask for more than there is in the TTY input buffer, you may read from the terminal at interrupt level.
5	10000,,0	INTPTO	A PTY job has just gone into a wait state waiting for you to send it characters.

6	4000,,0	INTMAIL	Someone has just sent you a letter (see Section 7). You may read the letter at interrupt level.
7	2000,,0	INTWAIT	A UWAIT UUO has just returned from finishing the execution of a UUO. You cannot enable for this interrupt, which is used by the monitor to make the UWAIT UUO work (see page 110).
8	1000,,0	INTPTI	A PTY you own has just sent you a character (or line).
9	400,,0	INTPAR	A parity error has occurred in your core image. The address in which bad parity was detected is given to you in AC 10.
10	200,,0	INTCLK	A clock interrupt has just happened. The default interval between clock interrupts is a 60th of a second; this interval can be changed by the CLKINT UUO (see page 110), which also enables clock interrupts. This bit is for new style clock interrupts only.
11	100,,0	INTINR	IMP interrupt from foreign receive side. See Section 13.12, specifically page 175, for explanations of this and the following three interrupt conditions.
12	40,,0	INTINS	IMP interrupt from foreign send side.
13	20,,0	INTIMS	IMP status change interrupt.
14	10,,0	INTINP	IMP input waiting.
15	4,,0	INTTTI	[ESCAPE] I has just been typed on the display terminal which is attached to this job.
16	2,,0	INTQXF	Your job is changing queues. Your new positive queue number is available from accumulator 14; see page 107. Note: This interrupt is not guaranteed to be generated every time you change queues.
19	0,,200000	* POV	A push-down stack overflow or underflow has just occurred. If the instruction causing this was a PUSHJ or a POPJ,

			then the PC stored in JOBTPC is the one specified by the instruction (PUSHJ) or by the stack (POPJ); if the instruction is a PUSH or a POP, then the PC will have been incremented and will thus point to the instruction after the PUSH or POP. In any case, the push-down pointer will have been given its new value and any value being pushed or popped will have been moved to its indicated destination. For a PUSHJ or a PUSH, the value pushed will now be occupying the last location in the stack. For a POPJ or a POP, the value popped will have come from the first location before the beginning of the stack.
22	0,,20000	* ILM	An illegal memory reference has just occurred; that is, you have just tried to reference a memory location outside your core image.
23	0,,10000	* NXM	A non-existent memory reference has just occurred; that is, you have just tried to reference a memory location that apparently does not exist. This should never really happen; if it does, it generally means a memory unit has failed.
26	0,,1000	*	The clock has just ticked while you were running. This bit is for old style clock interrupts only.
29	0,,100	* INTFOV	A floating overflow has just occurred. The PC saved will point to the word after the instruction causing the overflow.
32	0,,10	* INTOV	An integer overflow has just occurred. The PC saved will point to the word after the instruction causing the overflow.

The remaining bits are currently unused. As new interrupts are added they will use the lower-numbered available bits.

9.1 New Style Interrupts

The new interrupt system is highly recommended over the old system; thus I will explain the new system first. The bit representations of the particular interrupts are the same in both systems (except for clock interrupts), but with the new system there are more interrupts which you can enable.

When you receive a new style interrupt, all sorts of things happen. First of all, as usual your PC and flags are saved in JOBTPC and the bit representing the cause of the interrupt is stored in JOBCNI. Unless you were executing a UUO at the time of the interrupt, the PC word in JOBTPC will be perfectly accurate. If, however, you *were* executing a UUO, then the PC saved in JOBTPC is really the address in your core image where the UUO in progress is located; in this case, the user-mode bit (bit 5--the 10000,,0 bit) in JOBTPC will be off. Thus the user-mode bit in JOBTPC will tell you if a UUO was in progress when the interrupt occurred.

With a new style interrupt, your accumulators are saved. Then, before your interrupt routine is started, certain ACs are loaded up with data as listed in the table below. Your user-level ACs are saved in locations 20:37 of your core image unless you were executing a UUO at the time of the interrupt, in which case your ACs are saved somewhere in the system.

AC	*Contents*
1	The current value of the spacewar buttons. See the SPWBUT UUO on page 134.
2	Your current protection-relocation constant.
3	A warning value. This AC usually contains zero but is set up with -1 if this is the last interrupt you will get before your job is swapped out or shuffled.
4	The number of the processor this interrupt module is running on. This number is 1 for the PDP-10 and 2 for the PDP-6. Since interrupts are (currently) always run on the PDP-10, this AC will always contain 1.
5	A flag indicating the status of the processor that this interrupt-level process is *not* running on. This flag is zero if that processor is running (normal state) and -1 if that processor is dead.
6	Your job status word. See the JBTSTS UUO on page 90.
7	JOBREL for your upper segment, if any. This is the size, minus one, of your upper segment, if you have one, and zero if you do not.
10	The datum for this particular interrupt. Currently, the only interrupt with a datum is the parity error interrupt, for which you get here the address at which bad parity has been detected. This value can be

invalid if you have some pending interrupts which generate data (e.g., if you get several parity errors in a row), in which case this is the last datum seen by the system.

14 Your positive queue number. This tells you which queue your user-level process is in.

After these ACs have been set up, your interrupt routine is run at interrupt level starting at the address contained in the right half of JOBAPR. The PC flags for your interrupt-level routine are set from the bits in the left half of JOBAPR. (The PC flags are explained in Appendix 3.) For example, if bit 6 (the 4000,,0 bit) in JOBAPR is on, your interrupt process will be started up in IOT-USER mode (see Appendix 3). Your interrupt process is actually started by the system doing a JRST 13,@JOBAPR (see the JRST instruction in the PDP-10 manuals) and you are then allowed to run uninterrupted (except for I/O interrupt services) for up to 8 ticks (8/60 of a second). If you are still running at interrupt level when that time runs out, you will get the system error message I-LEVEL TIMEOUT and your program will be stopped.

When your interrupt-level routine finishes and wishes to return to the interrupted program, it should issue the DISMIS UUO (see page 110). If, however, the interrupt-level routine does not wish to return to the user-level program but wants instead to *become* the user-level program, then it should issue the UWAIT UUO and then the DEBREAK UUO. This lets your interrupt-level process keep running, but it will no longer be at interrupt level; it will be running at user level. Note that neither UWAIT (see page 110) nor DEBREAK (see page 111) causes any change in your PC flags. Thus, if you are in IOT-USER mode when you give these UUOs, then you will still be in IOT-USER when you return from them! If you UWAIT and DEBREAK and *then* want to return to the interrupted program, you can do so simply by jumping to the interrupted address (contained in JOBTPC) by doing a JRST 2,@JOBTPC which will also restore the flags of the interrupted process. However, you should probably save JOBTPC somewhere else before DEBREAKing because after you DEBREAK you can receive another interrupt, which would clobber the old JOBTPC with a new one.

To allow users to enable both old and new style interrupts at the same time, a user program can indicate a special three word block to be used with *new style* interrupts in place of the three words at JOBCNI, JOBTPC and JOBAPR, respectively. If JOBINT in your job data area contains a non-zero number, that number will be interpreted as the address of this three word block. The normal three words in the job data area will continue to be used for old style interrupts (and for new style interrupts whenever JOBINT contains zero).

Now here are the UUOs used with the new style interrupt system.

INTENB [OP=047, ADR=400025] CALLI 400025

```
        MOVE   AC,[<interrupt bits to be enabled>]
        INTENB AC,
```

The INTENB UUO enables the interrupts that correspond to the bits on (ones) in the AC and disables the interrupts corresponding to bits that are zero. This overrides all previous enablings (of the new style interrupts). If there is an interrupt pending that you are now enabling, the interrupt will be taken immediately. (See page 104 for the interrupt conditions represented by the various bits.)

INTORM [OP=047, ADR=400026] CALLI 400026

```
        MOVE   AC,[<bits to be ORed in>]
        INTORM AC,
```

The INTORM UUO enables the interrupts corresponding to the bits that are on (ones) in the AC. The enablings of other interrupts are unchanged.

INTACM [OP=047, ADR=400027] CALLI 400027

```
        MOVE   AC,[<bits to be cleared>]
        INTACM AC,
```

The INTACM UUO disables the interrupts corresponding to the bits that are on (ones) in the AC. The enablings of other interrupts are unchanged.

INTENS [OP=047, ADR=400030] CALLI 400030

```
        INTENS AC,
```

The INTENS UUO returns your new style interrupt enablings in the AC; bits which are on (ones) represent enabled interrupts.

CLKINT [OP=717]

```
        CLKINT 1,<number of ticks between interrupts>
```

The CLKINT UUO is used to set the interval between clock interrupts. If you enable for clock interrupts without using this UUO, you get the default interval of one tick (a 60th of a second); but with this UUO you can set the number of ticks per clock interrupt to any number representable in 18 bits (up to about an hour and twelve minutes).

This UUO enables for clock interrupts, masks clock interrupts on (see the INTMSK UUO on page 112), and sets the interval (in ticks) between interrupts to the number which is the effective address of the UUO. The AC field of this UUO must be a 1, as shown.

DISMIS [OP=047, ADR=400024] CALLI 400024

```
        DISMIS
```

The DISMIS UUO is used to terminate an interrupt-level process. When you give this UUO at interrupt level, the current interrupt is dismissed and your user-level program is continued at the point where it was interrupted. This UUO is illegal at user level. (The DISMIS UUO has another related meaning when given by a spacewar process; see page 102.)

UWAIT [OP=047, ADR=400034] CALLI 400034

```
        UWAIT
```

The UWAIT UUO can be used by an interrupt-level process to ensure that the interrupted program is not in the middle of executing a UUO. If a UUO was in progress when the interrupt occurred, then UWAIT will wait for that UUO to finish. UWAIT also returns with your user-level ACs set up (i.e., the ACs that were saved when the interrupt occurred), so you should not expect any ACs to be preserved through a UWAIT.

The mechanism used by UWAIT is the following. If no UUO is in progress, UWAIT sets up your user-level ACs and returns immediately. Otherwise, bit 7 (2000,,0 bit) in your interrupt enablings is turned on, the address of the instruction after the UWAIT (to which you will want to return later) is saved, and your user-level process is resumed, in the middle of some UUO. When that UUO finishes execution, the system notes that you are enabled (bit 7) for a UUO-completion interrupt; so bit 7 is cleared and your interrupt-level routine is continued at the saved address, with your user-level ACs already having been set up. At this point you are once again at interrupt level and are subject to its timeout.

After you have done a UWAIT, you can terminate your interrupt-level process with either of two

UUOs. You can DISMIS and leave interrupt level the normal way with no side effects for having done the UWAIT, or you can DEBREAK to make your interrupt-level process into your user-level process (see the DEBREAK UUO below for more details).

If you do a UWAIT and do not return immediately, then while the interrupted UUO is finishing other interrupts may occur and you may even do another UWAIT. When the UUO finally completes, the interrupt-level routine to regain control will be the last one that did a UWAIT.

If you are in the middle of a SLEEP UUO (see page 125) when you do a UWAIT, the SLEEP is terminated immediately; you do not wait until the time when it would normally have ended.

The UWAIT UUO is illegal except at interrupt level.

DEBREAK [OP=047, ADR=400035] CALLI 400035

```
        DEBREAK
```

The DEBREAK UUO is used to turn an interrupt-level process into a user-level process. In order to assure that you were not in the middle of a UUO when the interrupt occurred, you should give a UWAIT UUO before you DEBREAK. After debreaking, you will no longer be at interrupt level. However, you will have the same accumulators that you had before debreaking. DEBREAK does not alter the flow of instructions (as DISMIS does); the next instruction executed is the one following the DEBREAK, but it will be executed at user level.

After you do a UWAIT and a DEBREAK, you can return to your interrupted process by jumping to the address that was placed in JOBTPC when the interrupt started. Note, however, that as soon as the DEBREAK is done, you may receive further interrupts since you are now at user level; thus JOBTPC may get clobbered, so you might want to save it before you DEBREAK. See also the INTJEN UUO on page 113.

The DEBREAK UUO is illegal except at interrupt level.

IWAIT [OP=047, ADR=400040] CALLI 400040

```
        IWAIT
```

The IWAIT UUO causes your job to go into a wait state (INTW) that will be terminated only when an interrupt occurs.

IENBW [OP=047, ADR=400045] CALLI 400045

```
        MOVE  AC,[<interrupt bit enablings>]
        IENBW AC,
```

The IENBW UUO is used to set your interrupt enablings and to put your user-level process into the interrupt-wait state, both at the same time in order to prevent a race between going into interrupt wait and getting an interrupt. This UUO has the same effect (except for timing) that would be gotten from doing an INTENB followed by an IWAIT.

INTGEN [OP=047, ADR=400033] CALLI 400033

```
        MOVE   AC,[<bits to interrupt with>]
        INTGEN AC,
```

The INTGEN UUO generates an interrupt for you for each bit on in the AC. This will cause the interrupts to take place immediately, but you must be enabled for all of the interrupts you are generating or you will get a system error message.

INTIRQ [OP=047, ADR=400032] CALLI 400032

```
        INTIRQ AC,
```

The INTIRQ UUO returns in AC the bits representing the interrupts you currently have pending. Usually this will be zero unless you are at interrupt level. This will also be non-zero if between the time an interrupt is requested and the time it is serviced it is disabled, which can happen if your interrupt-level routine changes the interrupt enablings. Finally, this can be non-zero if you have masked off some interrupts (see the INTMSK UUO below).

INTMSK [OP=720]

```
        INTMSK 1,ADR

ADR:    <interrupt mask>
```

The INTMSK UUO is used to set your interrupt mask. In this mask, a 1 means the interrupt is masked on, a 0 means it is masked off. Your interrupt mask is initially all 1's. The only interrupts you can receive are those that are both enabled and masked on. If an enabled interrupt is masked off (with this UUO or with one of the following UUOs), then when that interrupt condition arises, the interrupt will remain pending until it is masked back on, at which time it will interrupt immediately. (The AC field of the UUO must be a 1, as shown.)

IMSKST [OP=721]

```
        IMSKST 1,ADR

ADR:    <bits to be turned on in interrupt mask>
```

The IMSKST UUO ORs the contents of location ADR into your interrupt mask (see the INTMSK UUO above). Thus this masks *on* the indicated interrupts. (The AC field of the UUO must be a 1, as shown.)

IMSKCL [OP=722]

```
        IMSKCL 1,ADR

ADR:    <bits to be turned off in interrupt mask>
```

The IMSKCL UUO turns off, in your interrupt mask, the bits that are on in location ADR. Thus this masks *off* the indicated interrupts. See the INTMSK UUO above. (The AC field of the UUO must be a 1, as shown.)

INTUUO [OP=723]

```
        INTUUO <function>,ADR
```

INTUUO is an interrupt system extended UUO that uses the AC field to determine which of several functions is to be executed. The different functions are described separately below.

INTJEN [OP=723, AC=0] INTUUO 0,

```
        INTJEN ADR

ADR:    <interrupts bits to be ORed in>
        <PC word to go to>
```

The INTJEN UUO solves an special race condition. Suppose you have done a UWAIT followed by a DEBREAK, fooled around in user level for a while, and now wish to enable your interrupts again and return to the original user-level process. You presumably have the PC word stored in a location somewhere. If you enable your interrupts and then try to jump back to the original user-level process, you may get interrupted between the enabling UUO and the jump. The second interrupt may want to do the same thing and may in the process overwrite your PC word with a new and different PC. This UUO solves this race by setting the interrupt bits at ADR and jumping to the PC contained in ADR+1 all in one indivisible operation, so that if another interrupt occurs, the PC it gets will be the one specified in ADR+1.

IMSTW [OP=723, AC=1] INTUUO 1,

```
        IMSTW ADR

ADR:    <mask bits>
        1
```

The IMSTW UUO sets your interrupt mask (see the INTMSK UUO above) and then puts you into *interrupt wait*. The effective address of the instruction indicates a two word block, the first word of which should have the mask bits you want set. The second word of the block must contain a 1, as shown.

IWKMSK [OP=723, AC=2] INTUUO 2,

```
        IWKMSK ADR

ADR:    <bits for interrupts that should awaken>
```

The IWKMSK UUO sets the mask that determines if the main job gets awakened out of IWAIT when an interrupt happens. If an interrupt occurs that should not awaken you, then it is processed but when it finishes, your main job will not be taken out of interrupt wait.

INTDMP [OP=723, AC=3] INTUUO 3,

```
        INTDMP ADR
        <error return - error code in ADR+1>

ADR:    <job name or number>
        <block of 5 words for returned information>

Error codes:

  2     ambiguous job name
  3     non existent job name
```

The INTDMP UUO returns 5 words of information about a particular job. The effective address of the UUO specifies a six word block; the first word of this block should contain the number or sixbit name of the job you want to find out about, where zero means your own job. If the UUO is successful, the skip return is taken and words 1 through 5 of the block are filled with the information indicated below. If the value in ADR specifies an ambiguous or non-existent job, then the error return (no skip) is taken and an error code (see above) is returned in ADR+1.

Word	*Value*
0	Unchanged (job name or number).
1	Interrupt enable bits of specified job.
2	Interrupt mask.
3	Zero.
4	Wakeup mask (interrupt bits which will awaken the job from IWAIT; see IWKMSK UUO above).
5	Number of the queue the job is in.

INTIPI [OP=723, AC=4] INTUUO 4,

```
        INTIPI ADR
        <error return - code in ADR+1>

ADR:    <job name or number>
        <bits for interrupts you want generated>

Error codes:

  1     non existent job number
  2     ambiguous job name
  3     non existent job name
  4     job not enabled for interrupts specified
```

The INTIPI UUO is used to send one or more interrupts to any job on the system. ADR should contain the number or sixbit name of the job you want to receive the interrupt(s). The specific interrupts you want generated should be indicated by bits on in the word at ADR+1. The job you choose must be enabled for the interrupts you send. If this UUO is successful, the skip return is taken; otherwise, the error return (no skip) is taken and an error code is returned in the word at ADR+1.

IMSKCR [OP=723, AC=5] INTUUO 5,

```
        IMSKCR ADR

ADR:    <bits to be turned off in interrupt mask>
```

The IMSKCR UUO turns off, in your interrupt mask, the bits that are on in location ADR. Thus this masks *off* the indicated interrupts. After masking off these interrupts, this UUO returns your *old* mask in the word at ADR. This avoids timing errors in saving and restoring a mask. See the INTMSK UUO above.

9.2 Old Style Interrupts

When you receive an interrupt under the old interrupt system, here is what happens: Your PC and flags are saved in JOBTPC, the CONI word from the processor is stored in JOBCNI (one old style interrupt bit will be on in this word indicating the cause of the interrupt), and your PC is changed to that in the right half of JOBAPR. Your interrupt routine may run as long as it wants; as far as the system is concerned, your interrupt routine is now your main process and no further special action will be taken by the system for the interrupt. When your interrupt routine is finished, it can do a

```
JRST 2,@JOBTPC
```

to return to the interrupted program. This will restore the PC and flags to their states when the interrupt occurred.

Note that since your interrupt routine is not treated at all specially by the system, it can be interrupted itself by an interrupt of the same or a different type. If this happens, JOBTPC and JOBCNI will be clobbered with the values for the second interrupt; then when processing resumes for the first interrupt, the address (in the main program) to which control should return will no longer be in JOBTPC. Unless this address was saved by your interrupt handler, the program will probably not be able to continue successfully. The new interrupt system avoids this problem by not allowing you to be interrupted while you are processing a new style interrupt.

The processor CONI word stored in JOBCNI with old style interrupts will have some extraneous bits on in addition to the bit representing the interrupt. The extra bits currently set in this word are the 0,,6043 bits.

Here are a couple of UUOs to enable old style interrupts.

APRENB [OP=047, ADR=16] CALLI 16

```
MOVE   AC,[<interrupt bits to be enabled>]
APRENB AC,
```

The APRENB UUO enables your job to receive old style interrupts upon any of the conditions represented by the bits in the AC. Only bits 19, 22, 23, 26, 29 and 32 (0,,231110 bits) can be enabled by the old interrupt system. See page 105 for the interrupt conditions these bits represent.

SETPOV [OP=047, ADR=32] CALLI 32

```
        MOVEI  AC,<address of interrupt routine>
        SETPOV AC,
```

The SETPOV UUO is used to enable for old style interrupts on push-down stack overflow and underflow. This UUO takes the address in AC and stores it in JOBAPR; thus this should be the address of your interrupt handling routine. Then you are enabled for old style push-down interrupts, with any previous old style enablings discarded. Thus you cannot use this UUO if you want any old style interrupts other than push-down overflow/underflow.

SECTION 10

LIBRASCOPE FAST BAND STORAGE

User programs are allowed to use bands on the Librascope disk for fast secondary storage. However, because the system uses Librascope bands for holding swapped out jobs and because there are only a limited number of bands, user programs should not depend on this resource too heavily. The system may crash if there are not enough bands for swapping.

Librascope storage is allocated in bands, with each band being =76 * =1024 (76K) words long. Each band has =2432 sectors (numbered from 0 to =2431); and each sector is =32 words long. When reading or writing a fast band, you specify the number of the sector at which the transfer is to start and the number of words to be read or written. The number of words should always be a multiple of =32; if it isn't, then it will be increased by the system to the next multiple of =32 and that many words will be transferred, whether you like it or not. If you read or write past the end of sector number =2431, you will actually be reading/writing at the beginning of the band (starting over with sector 0).

When requesting a Librascope fast band (with the UFBGET UUO, page 120) you may specify *either* a logical band number from 0 to 37 by which you can refer to the band that you are given *or* the physical number of a particular band that you want. A band number is taken as a physical band number if it has the 400000 bit on; otherwise, it will be interpreted as a logical band number. When you have been assigned a fast band, you can expect it to contain whatever garbage was last written on it.

Each fast band has associated with it a sector offset which indicates where the logical beginning of the band (sector 0) is located relative to the physical beginning of the band. The value of this offset is generally irrelevant to the user except when he may want to try to reclaim a particular band after some sort of catastrophe, such as a system restart. At such a time, the user might want to tell the system where on a particular band his data starts. This sector offset can be set to a specific value with the UFBGET UUO, and it can be determined (for instance when set by the system) for a particular band with the UFBPHY UUO (see page 122). The value of the offset is always between 0 and =1215 inclusive.

10.1 Getting and Releasing Fast Bands

Here are the UUOs used to obtain and release Librascope fast bands.

UFBGET [OP=047, ADR=400010] CALLI 400010

```
        MOVE   AC,[<enable bits and offset>,,<band number>]
        UFBGET AC,
        <return if no bands available>
```

The UFBGET UUO is used to acquire one Librascope band. The right half of AC should contain *either* the logical band number (from 0 to 37) by which you will refer to the band you get *or* the physical number of a particular band that you want; physical band numbers must have the 400000 bit on. Bits 1 (200000,,0 bit) and 2 (100000,,0 bit) of AC are write- and read-enabling bits, respectively, which, if on, allow other jobs to write and/or read this fast band of yours. Bits 7:17 (3777,,0 bits) of AC should contain the sector offset which you wish this band to have, where zero means the system should choose whatever offset it wants (see the above explanation of the offset). To get a real offset of zero, turn on bit 0 (400000,,0 bit) in the AC and make bits 7:17 zero.

If this UUO is successful, it takes the skip return and the physical number of the band assigned to you is returned in AC (but without the 400000 bit on). If there are no bands available, or if a particular band you have requested is unavailable, the direct (error) return is taken.

UFBGIV [OP=047, ADR=400011] CALLI 400011

```
        MOVEI  AC,<band number>
        UFBGIV AC,
```

The UFBGIV UUO releases the fast band whose number is in the AC. This number should have the 400000 bit on if it is a physical band number; otherwise, it will be interpreted as a logical band number.

The RESET UUO (see page 126) releases all fast bands assigned to you.

UFBCLR [OP=047, ADR=400012] CALLI 400012

```
        UFBCLR
```

The UFBCLR UUO releases all fast bands assigned to you.

10.2 Reading and Writing Fast Bands

The UUOs used to read and write Librascope fast bands are described below.

FBREAD [OP=706]

```
        MOVEI  AC,<band number>
        FBREAD AC,ADR
        <error return>

ADR:    <no-wait flag (sign bit)>,,<address where data is to go>
        <number of words to be read>
        <sector address of beginning of transfer>
```

The FBREAD UUO causes data to read into your core image from the fast band whose logical or physical band number is in the AC. The effective address of the UUO should point to a three-word block (as shown above) which contains the following values: 1) the address where the data is to be deposited in your core image, 2) the number of words to be read and 3) the number of the sector at which reading is to start. If the sign bit (400000,,0 bit) of the first word of this block is *on*, then this UUO will return immediately without waiting for the transfer (although it will wait for any previously initiated transfer to finish); if the sign bit is *off*, the FBREAD UUO will not return until the data transfer has been completed.

If there are no errors during the read (or during the initiation of the read if the no-wait bit is on), this UUO will take the skip return. Upon an error (including a request to read a band that you are not permitted to read), the direct return will be taken.

FBWRT [OP=707]

```
        MOVEI AC,<logical fast band number>
        FBWRT AC,ADR
        <error return>

ADR:    <no-wait flag (sign bit)>,,<address of data>
        <number of words>
        <beginning sector number>
```

The FBWRT UUO causes data to be written from your core image onto the fast band whose logical or physical band number is in AC. The effective address of the UUO should point to a three word block (as shown above) which indicates 1) where the data to be written out is located, 2) the number of words to be written and 3) the number of the sector at which writing is to start. If the sign bit (400000,,0 bit) of the first word of this block is *on*, then this UUO will return immediately without waiting for the transfer (although it will wait for any previously initiated transfer to finish); if the sign bit is *off*, the FBWRT UUO will not return until the data transfer has been completed.

If this UUO is successful, the skip return will be taken. Upon an error, the direct return will be taken.

10.3 Miscellaneous Fast Band UUOs

Here are some special UUOs provided for finding out various information about the status of Librascope fast bands.

UFBPHY [OP=047, ADR=400055] CALLI 400055

```
        MOVE   AC,<logical or physical band number>
        UFBPHY AC,
```

The UFBPHY UUO returns the physical band number and offset of the Librascope fast band indicated in AC. The offset is returned in AC left; the physical band number is returned in AC right (but without the 400000 bit on). Bit 1 (200000,,0 bit) in AC will be on if you have write access to the band, and bit 2 (100000,,0 bit) will be on if you have read access. Zero will be returned in AC if there is no such band.

UFBSKP [OP=047, ADR=400056] CALLI 400056

```
        UFBSKP
        <transfer in progress>
        <no transfer in progress>
```

The UFBSKP UUO simply skips unless you have a Librascope fast band transfer in progress, in which case the direct return is taken.

FBWAIT [OP=047, ADR=400057] CALLI 400057

```
        FBWAIT
```

The FBWAIT UUO simply waits until any fast band transfer in progress finishes. If you have don't have a transfer going on when you call it, it will return immediately.

UFBERR [OP=047, ADR=400060] CALLI 400060

```
        UFBERR
        <error occurred>
        <no error>
```

The UFBERR UUO simply skips unless your last fast band transfer encountered an error; if you had an error, then the direct return is taken.

SECTION II

MISCELLANEOUS UUOS

This section describes various UUOs which did not seem to fit in any other sections. These UUOs include the common and useful EXIT, SLEEP, RESET and SWAP UUOs; also in this section are the UUOSIM and TMPCOR UUOs.

EXIT [OP=047, ADR=12] CALLI 12

```
        EXIT <ac>,
```

The EXIT UUO is used to cause your program to stop and exit to the monitor. If <ac> is 0, then all I/O channels you have open will be closed (as if by the RELEAS UUO with no inhibit bits on; see page 29), then a RESET (see the RESET UUO below) will be done and you will not be permitted to CONTINUE your program after exiting. If <ac> is 1, then this UUO will not affect any of your I/O channels nor will it do a RESET; a subsequent CONTINUE monitor command will cause your program to resume execution at the instruction immediately following the "EXIT 1," instruction. Other values of <ac> are reserved for future use.

Both forms of this UUO kill any spacewar modules you have.

A phantom or detached job giving either form of this UUO will have its I/O channels closed, and then the job will be killed.

SLEEP [OP=047, ADR=31] CALLI 31

```
        MOVEI AC,<number of seconds to sleep>
        SLEEP AC,
```

The SLEEP UUO causes your job to be stopped for the number of seconds specified by the contents of the AC, which is interpreted (approximately) modulo =69. If AC contains zero, your job will sleep for 1/60 of a second. When the sleep time is up, your program is resumed at the instruction immediately after the SLEEP. If you type Control-C or receive an interrupt while you are sleeping, the SLEEP is terminated immediately.

RESET [OP=047, ADR=0] CALLI 0

RESET

The RESET UUO is used to reset various conditions pertaining to your job. All monitor commands that get a new program into your core image do a RESET first. Also, the "EXIT 0," UUO (see above) does a RESET just before exiting. Here is a list of the things that happen when a RESET is done:

All your I/O channels are released without being closed. That is, the result is the same as that from doing a RELEAS <channel>,3 for every channel you have open. See the RELEAS UUO on page 29.

The state of program-controllable echoing of typed characters is reset to echo all characters. This is done by clearing the NOECHO bit (bit 28--the 0,,200 bit) and the NOECHB bit (bit 27--the 0,,400 bit) in the TTY I/O status word (see Section 13.2). See also Section 3.1.

The special-activation-mode bit (bit 11--the 100,,0 bit) is cleared in your line characteristics word (see the GETLIN UUO on page 40), and your special activation table is reset to the standard special activation table. Also, the linefeed-insertion-inhibition bit (bit 16--the 2,,0 bit) is cleared in your line characteristics unless you are running on a PTY.

JOBFF in your job data area is reset from the value in the left half of JOBSA. See Appendix 4.

Your user interrupt enablings (both new and old style) are cleared. See Section 9.

Your core image is unlocked from core. See the LOCK and UNLOCK UUOs on page 133.

If you have a simulated upper segment (created by the SETPR2 UUO; see page 92), it goes away. Note that this does not affect any real upper segment you may have.

Any spacewar processes you have are killed. See Section 8.

Any Librascope fast bands you have are released. See Section 10.

If there is a letter in your mailbox, it is thrown away. See Section 7.

Any pseudo-teletypes (PTYs) you have are released along with any jobs logged in on those PTYs. See Section 3.5.

Any extra Data Disc channels you have are released. See Section 4.5.

If you are on a III or Data Disc display, your display is reset. This means that your page printer is normalized and any III display programs running are killed. See Section 4.

If you are on a Data Disc display, your video switch map is reset to the permanent map. See Section 4.6.

If you are on a III or Data Disc display, your audio switch connection is reset to the permanent connection. See Section 4.7.

SWAP [OP=047, ADR=400004] CALLI 400004

```
        MOVE AC,[SAVADR,,GETADR]
        SWAP AC,

SAVADR: <device name in sixbit>
        <file name in sixbit>
        <file name extension in sixbit>
        <core size in 1K blocks>,,<starting address>
        <project-programmer name>

GETADR: <device name in sixbit>
        <file name in sixbit>
        <file name extension in sixbit>,,<mode bits>
        <core size in 1K blocks>,,<starting address increment>
        <project-programmer name of file>
        <project-programmer name for new job>
```

The SWAP UUO is used to save your core image in a file and/or get or run another core image from a file. This UUO can also be used to create a job and to start up a program on that job. The format of the files used by this UUO is exactly that of the SAVE, GET, RUN and R monitor commands. Note that SWAP does not allow saving upper segments; in fact, SWAP kills your upper segment, if any, before doing anything else. Also, SWAP does a RESET (see the RESET UUO above) before saving or getting a core image.

If the left half of the AC is non-zero, then your core image will be saved in the file described by the block pointed to by the left half of the AC. After that, if the right half of AC is non-zero, then the core image contained in the file described by the block pointed to by the right-half of the AC is run or set up as either your core image or that of a new job. If both halves of AC contain zero, the SWAP UUO is a no-op.

Now for a few details.

If the left half of the AC is zero, no core image is saved. If it is non-zero, it should point to a five-word block which contains (as shown at SAVADR above) the specifications for the dump file to be saved. These specifications include the device name, file name and extension, and project-programmer name for the file, the amount of core (in 1K blocks) to be saved and the starting address for the dump file. If the core size is zero, the amount of core you currently have

will be used. If the starting address is zero, the current starting address of your job will be used. If the starting address is non-zero, it will be copied into the right half of JOBSA in your job data area before the core image is saved. If the extension is zero, 'DMP' will be used. If the project-programmer name is zero, your current Disk PPN is used (see page 16).

Next, if the right half of AC is zero, then no new core image is run. If it is non-zero, it should point to a block which contains (as shown at GETADR above) the specifications for the dump file to be run. .These include the device name, file name and extension, and project-programmer name of the dump file, the core size it is to be run in, the starting address increment, some special mode bits and, if the dump file is to be run as a new job (independent of the job giving the SWAP UUO), the project-programmer name under which that job should be logged in. The starting address increment is added to the starting address saved in the dump file to determine where the program is to be started. The mode bits are in the right half of the file extension word and have the following meanings:

Bits	*Octal*	*Meanings of 1's in mode bits of SWAP UUO*
35	0,,1	The dump file will not be started; instead, the message JOB SETUP will be typed out and the job will be put into monitor mode.
34	0,,2	The right half of GETADR+3 will be taken as an absolute starting address rather than as an increment.
33	0,,4	The program will be started on a new job which will be logged in under the project-programmer name at GETADR+5; if this PPN is zero, your logged in PPN will be used for the new job. If this bit is off, then the word at GETADR+5 is ignored.
32	0,,10	If starting up another job (bit 33 on), the job will be started up as a phantom rather than as a normal job (see the WAKEME UUO on page 132). This means that the JLOG bit in the job status word (see the JBTSTS UUO on page 90) for the new job will not be turned on; thus that job will go away if it hits any sort of error condition (such as a parity error or a push-down stack overflow).

When you start up a new job with this UUO by having bit 33 on in GETADR+2, the job number of the new job is returned in the AC specified in the UUO. A zero is returned if no new job could be started because there were no job slots left. When a new job is successfully started, its ACs are copied from your ACs; but in the AC specified in the SWAP UUO, the new job will get your job number instead of its own. Thus the old job is given the number of the new job and the new job is given the number of the old one.

If the ENTER fails on the file specified at SAVADR, or if the LOOKUP fails on the file specified at GETADR, an error message will be typed out and the program stopped.

```
RUN             [OP=047, ADR=35]  CALL1 35
-------------------------------------------------
        MOVE AC,[<starting address increment>,,GETADR]
        RUN  AC,

GETADR: <device name in sixbit>
        <file name in sixbit>
        <file name extension in sixbit>,,<mode bits>
        0
        <project-programmer name of file>
        <core size in 1K blocks>
```

The RUN UUO is DEC's version of the SWAP UUO. Except for a slightly different parameter format (see RUN UUO calling sequence above), the only differences between SWAP and RUN are that with the RUN UUO, no core image can be saved and no phantom job can be started up (mode bits 32:33 are ignored). For details, see the SWAP UUO above.

```
TMPCOR          [OP=047, ADR=44]  CALL1 44
-------------------------------------------------
        MOVE   AC,[<code>,,ADR]
        TMPCOR AC,
        <error return>

ADR:    <filename>,,0
        IOWD   BLEN,BUF

BUF:    BLOCK  BLEN


Code    Function

  0     Return in AC the number of words of free TMPCOR space.
  1     Read specified file.
  2     Read and delete specified file.
  3     Write specified file (deleting old version, if any).
  4     Read TMPCOR directory.
  5     Read and clear TMPCOR directory.
```

The TMPCOR UUO allows a job to leave several short files in core from the running of one program to the next. A RESET will not affect these files, which can be referenced only by the job that created them. All of a job's TMPCOR files are deleted when the job is killed. This system of temporary storage improves response times by reducing the number of disk operations. A TMPCOR file must be written or read all in one dump-mode operation—you cannot pick up reading or writing where you left off; however, when reading a temporary file, you do not have to read the entire file. The sum of the sizes of all the TMPCOR files for a single job is not allowed to exceed 400 words.

Each TMPCOR file has an *explicit* three-character sixbit file name and an *implicit* project-programmer name (PPN). When you create a temporary file, the file is given your current ALIAS (Disk PPN) as its project-programmer name. To reference the file later, either to read,

delete or overwrite it or to find it in a TMPCOR directory, your ALIAS must be equal to the file's PPN (i.e., your ALIAS when the file was written).

For the TMPCOR UUO, AC left should contain a code that indicates which one of several functions is to be performed. AC right should hold the address of a two-word block which contains, as indicated above, the name of the file being referenced (if any) and the length (BLEN) and location (BUF) of the user buffer area from which or into which data is to written or read. When this UUO returns, AC will contain a value that depends on the function executed. This UUO skips on successful completion of the function and takes the direct (error) return otherwise. Each function is described separately below.

Code *Function*

0 Get free TMPCOR space. The number of remaining words of TMPCOR space available to the job is returned in AC. This function always takes the skip return. (The two-word block at ADR is not referenced by this function.)

1 Read file. If the specified file exists, as much of it as possible is read into the user's buffer area (at BUF), the length of the file is returned in AC and the skip return is taken. If the file does not exist, the number of words of free TMPCOR space is returned in AC and the direct (error) return is taken.

2 Read and delete file. This function is the same as function 1 except that the file is deleted after it is read.

3 Write file. If a file already exists with the specified name, it is deleted. Next, if there is enough TMPCOR space for the new file (whose size is given by BLEN), then the file is written (with the data at BUF), the number of remaining free TMPCOR words is returned in AC and the skip return is taken. If there is not enough space to write the file completely, then the file is not written, the number of remaining free TMPCOR words is returned in AC and the direct (error) return is taken.

4 Read directory. The number of different TMPCOR files the job has is returned in AC and an entry in the user's buffer area is made for each file until either there is no more space or all the files have been listed. The entry for a file has the following format:

```
<name>,,<size>
```

where <name> is the filename and <size> is the file length in words. This function always takes the skip return.

5 Read and clear directory. This function is the same as function 4 except that after the directory is read, all of the user's TMPCOR files are deleted.

```
UUOSIM          [OP=047, ADR=400106]  CALLI 400106
-------------------------------------------------
        MOVEI  AC,ADR
        UUOSIM AC,

ADR:    <PC saved here for normal UUOs>
        <UUO saved here for normal UUOs>
        <PC to transfer to for normal UUOs>
ADR+3:  <PC saved here for 1-level UUOs>
        <UUO saved here for 1-level UUOs>
        <PC to transfer to for 1-level UUOs>
ADR+6:  <PC saved here for spacewar UUOs>
        <UUO saved here for spacewar UUOs>
        <PC to transfer to for spacewar UUOs>
```

The UUOSIM UUO allows a user to have all UUOs trap to certain locations in his core image instead of being executed by the system. At the same time the user can still have the system execute whatever UUOs the user needs. The UUOSIM UUO passes to the system the address of a 9-word block which consists of three contiguous 3-word blocks, each specifying what the system should do when a certain kind of UUO is given. The first 3-word block indicates what should be done on UUOs given by the user's main program; the second 3-word block pertains to UUOs given at interrupt level; the third block is for UUOs given at spacewar level on the PDP-10. (Note that UUOs can *never* be executed on the PDP-6.) The address of the 9-word block should be in the AC specified in the UUOSIM UUO. After this UUO is executed, subsequent UUOs will cause the following action, using the appropriate 3-word block as mentioned above.

If the second word of the block is non-zero or if the third word is zero, then the UUO is executed by the system in the usual manner. Otherwise, the PC at the time the UUO was encountered is saved in the first word of the block, the UUO itself is stored in the second word of the block and control is transferred to the PC specified in the third word of the block. Note that the PC stored in the first word has already been incremented; it points to the instruction immediately following the UUO. Note also that the UUO stored in the second word has already had the effective address calculation carried out; the effective address is in the address field and the indirect and index fields will be zero. The prior test for the second word being non-zero has the effect of disabling user handling of UUOs issued by the user's UUO handler itself.

To undo the above effect of the UUOSIM UUO, give the UUOSIM UUO with zero in the specified AC and with ADR+1 (from the original UUOSIM given) containing a non-zero value so that this UUO will not simply be handed back to you. The RESET UUO (see page 126) will also disable further special user handling of system UUOs (but again you must force the system to handle the RESET itself).

```
WAKEME          [OP=047, ADR=400061]  CALLI 400061
-------------------------------------------------
        MOVEI   AC,ADR
        WAKEME AC,
        <error return>

ADR:    <phantom's jobname in sixbit>
        <phantom's PPN in sixbit>
        <data>

data < 0  means never start this job
data = 0  means start this job now if it is not already running
data > 0  means start this job at the time specified by <data>,
          where <data> = <date>,,<time in minutes>
```

The WAKEME UUO is used to start up, or prevent from starting up, any of the system phantom jobs. A phantom is a job started by the system to do some system-related work but which runs as a user job. Phantom jobs are defined by the fact that they run with the JLOG bit off in the job status word (see the JBTSTS UUO on page 90) and are not logged in under the project-programmer name 100,100. The JLOG bit being off means that the job will go away if it hits an error (such as a parity error or illegal memory reference). Also, if you type a monitor command to a job with the JLOG bit off, the job will go away immediately.

For this UUO, AC should contain the address of a three word block. The first two words of this block should have, in sixbit, the name of the phantom to be started up and the project-programmer name where that phantom lives. The third word of the block should contain a code indicating what the system should do about the phantom. A negative code indicates that the phantom should never get started up, a zero code means that the phantom should be started now unless it is already running, and a positive code represents a date (left half) and time (right half) when the phantom should be started up. The date is in system date format (see the DATE UUO on page 85) and the time is in minutes after midnight.

If there is no phantom with the jobname and PPN given, the error return is taken. Otherwise, the skip return is taken.

N.B. You should not use this UUO unless you are the person responsible for the phantom you are referencing or unless you are absolutely sure that what you are doing is okay.

```
JOBRD          [OP=047, ADR=400050]  CALL1 400050
-------------------------------------------------
        MOVE1 AC,ADR
        JOBRD AC,
        <error return - code in ADR+1>

ADR:    <job name or number>
        -<word count>,,<address of data in his core image>
        <address in your core image where data is to go>

Error codes:    1  non-existent job number (job number = 0)
                2  ambiguous job name
                3  non-existent job name (or job number > 77)
                4  address out of bounds (either in your
                   core image or in his)
                5  job not logged in
                6  block too large (more than 1K)
```

The JOBRD UUO allows you to read a block of data out of the core image of another job. The data is BLTed from his core image to yours. AC should contain the address of a three word block, the first word of which should contain either the sixbit name or the number of the job whose data you wish to copy. The left half of the second word should contain the negated count of the number of words to be read; the right half of the second word should contain the address of the block in the other job that you want to copy. The third word should contain the address in your core image where you want the data to be put. The maximum size block that can be read using this UUO is 1K (=1024 words).

If this UUO fails for any reason, the direct (error) return is taken and a code indicating the cause of failure is placed in ADR+1. A list of the possible error conditions and their codes is given above. If this UUO succeeds in transferring the data, the skip return is taken.

```
LOCK           [OP=047, ADR=400076]  CALL1 400076
-------------------------------------------------
        LOCK AC,
```

The LOCK UUO is used to lock your job in core so that you can be sure that you will not be either swapped out or shuffled in core. Upon return from this UUO, you will have been locked in core and AC will contain your job's protection-relocation constant. Your protection constant (in the left half of AC) is the highest address in your core image and always ends in 1777. The relocation constant (in the right half of AC) is the value that is added to each memory address you reference, in order to get the real memory address of the desired word in your core image.

Jobs with upper segments are not allowed to lock themselves in core.

To undo the effect of the LOCK UUO, you can use the UNLOCK UUO (see below) or the RESET UUO (see page 126). Any system-detected error condition will also cause an UNLOCK to

be done, as will any attempt to change your core size with the CORE UUO (see page 87). The LOCK UUO itself will first do an UNLOCK before locking you in.

The LOCK UUO should be used only when really necessary. When users are locked in core, there is less core available for normal users who are being swapped in and out; and when there are or have recently been two or more users locked in core, there is the possibility of having a hole in core that cannot be used (except for other locked jobs). A job that must remain locked in core for some time should give the LOCK UUO over again whenever there is a chance to do so because this causes the job first to be unlocked, then shuffled to fill any hole and finally locked again.

UNLOCK [OP=047, ADR=400077] CALLI 400077

```
        UNLOCK
```

The UNLOCK UUO is used to unlock your job from core after you have used the LOCK UUO (see above) to lock it. Many other things also cause an UNLOCK to be done. These are listed under the LOCK UUO above.

SETDDT [OP=047, ADR=2] CALLI 2

```
        MOVEI  AC,<address of DDT or RAID>
        SETDDT AC,
```

The SETDDT UUO is used to tell the system the starting address of DDT or RAID in your core image. This address is saved in JOBDDT in your job data area (see Appendix 4); however, you are not allowed to change this value directly--you must use this UUO instead. The DDT monitor command starts your program at the address contained in JOBDDT if that address is non-zero.

SPWBUT [OP=047, ADR=400000] CALLI 400000

```
        SPWBUT AC,
```

The SPWBUT UUO returns in the AC the value of the spacewar buttons. Each spacewar button controls one bit in this value. The bit is a one if the circuit represented by that bit is open and a zero if the circuit is closed. Only bits 22:26 and 28:35 (the 0,,37377 bits) are currently wired up. If the buttons are unplugged, the wired-up bits will all be on (value of 0,,37377).

The regular spacewar buttons only utilize the low-order 8 bits (bits 28:35--the 0,,377 bits). At the time of this writing, these buttons use normally-closed switches (bit values of zero) except that the switch for bit 34 (the 0,,2 bit) is normally open (bit value of one). This means that with the buttons plugged in and not depressed, the value of the spacewar buttons is 0,,37002. No claim is

made, however, that new buttons will not replace those that are described in this paragraph. To be sure what kind of switches are in the buttons, find and examine the buttons yourself.

EIOTM [OP=047, ADR=400005] CALLI 400005

EIOTM

The EIOTM UUO puts you into IOT-USER mode, in which opcodes from 700 up are executed as machine I/O instructions rather than as UUOs. This mode is described further in Appendix 3.

SECTION 12

OBSOLETE OR OTHERWISE USELESS UUOS

This section documents some UUOs that still work but which, for various reasons, are obsolete. Mentioned at the end of this section are some other UUOs that are ever more obsolete in that they do not work. In general, there are better ways to do the things all of these UUOs do, but the documentation is included for completeness and for the benefit of people trying to decode rusty old programs which use these UUOs. Users are to be discouraged from using these in any new pieces of code.

12.1 Old UUOs

Here are some UUOs that still work although they are generally unnecessary.

INTIIP [OP=047, ADR=400031] CALLI 400031

```
        INTIIP AC,
```

The INTIIP UUO is designed to be given by a process running at interrupt level. It returns in AC the bit representing the source of the current interrupt. This is a copy of the value of JOBCNI at the time your interrupt-level routine is started up. Thus it is much more efficient to do a MOVE AC,JOBCNI and the results are the same. If you give this UUO when you are not at interrupt level, then zero is returned in AC.

USKIP [OP=047, ADR=400041] CALLI 400041

```
        USKIP
        <return if no UUO in progress>
```

The USKIP UUO can be used by an interrupt-level process to determine if the interrupted program was in the middle of executing a UUO. USKIP will skip if there was a UUO in progress at the time the interrupt occurred and will take the direct return if no UUO was being executed. Thus this UUO will tell you whether a UWAIT will return immediately. The same information can be obtained by examining the user-mode bit (bit 5--the 10000,,0 bit) in JOBTPC; the user-mode bit will be on unless a UUO was in progress.

The USKIP UUO is illegal except at interrupt level.

LIOTM [OP=047, ADR=400006] CALLI 400006

```
        LIOTM
```

The LIOTM UUO gets you out of IOT-USER mode, thus making opcodes over 700 into UUOs again. However, this function can be achieved without doing a UUO at all. For instance, the instruction

```
        JRST 2,@[.+1]
```

will also get you out of IOT-USER mode. See the writeup of IOT-USER mode in Appendix 3.

RUNMSK [OP=047, ADR=400046] CALLI 400046

```
        MOVEI  AC,<processor enable bits>
        RUNMSK AC,
```

The RUNMSK UUO is intended to allow you to select which processor (the PDP-10 or the PDP-6) should run your job. However, the PDP-6 is not capable of running jobs; thus this UUO is useless.

DDTIN [OP=047, ADR=1] CALLI 1

```
        MOVEI AC,ADR
        DDTIN AC,

ADR:    <21 word block for returned characters>
```

The DDTIN UUO is used to read in all characters that have been typed on the terminal attached to your job. This UUO does not wait for any special activation character; it just reads whatever is in the TTY input buffer and returns those characters in the block pointed to by the contents of AC. This block should be at least 21 words long in order to hold all the characters being read; at most =84 characters will be read. The characters are returned as an ASCIZ string at ADR (7 bits per character, 5 characters per word, with a null (zero) byte after the last character read).

If the TTY input buffer is empty, this UUO will not return until a character is typed. Thus at least one character is returned each time this UUO is given.

TTYUUO (see Section 3.3) provides generally more convenient ways of reading characters from the terminal.

DDTOUT [OP=047, ADR=3] CALLI 3

```
        MOVEI  AC,ADR
        DDTOUT AC,

ADR:    <ASCIZ string to be typed out>
```

The DDTOUT UUO types out an ASCIZ string on the terminal. AC should contain the address of the first word of the string. The string is terminated by the first null (zero) byte. The OUTSTR UUO (see page 39) is the recommended way of typing out strings although this UUO still works.

GETCHR [OP=047, ADR=6] CALLI 6

```
        MOVE   AC,[<device name in sixbit, or channel number>]
        GETCHR AC,
```

The GETCHR UUO does exactly the same thing as the DEVCHR UUO; see page 33.

SETNAM [OP=047, ADR=400002] CALLI 400002

```
        MOVE   AC,[<sixbit job name>]
        SETNAM AC,
```

The SETNAM UUO is used to change your job name to that given in the AC. Any job name is legal. This UUO does exactly the same thing as the SETNAM UUO which is CALLI 43.

SEGSIZ [OP=047, ADR=400022] CALLI 400022

```
        SEGSIZ AC,
```

The SEGSIZ UUO returns in AC the size of your upper segment. The size returned is the protection constant of your segment, i.e., the number of words in it minus one. If you have no upper segment attached, zero is returned.

This segment size can be found out more easily by looking at JOBHRL in the job data area (see Appendix 4).

12.2 Privileged UUOs

The UUOs listed below are privileged UUOs designed for use only by the LOGIN and LOGOUT programs.

LOGIN	[OP=047, ADR=15] CALLI 15
LOGOUT	[OP=047, ADR=17] CALLI 17

12.3 Useless UUOs

Each of the following unimplemented UUOs is either illegal or a no-op.

DDTGT	[OP=047, ADR=5] CALLI 5
DDTRL	[OP=047, ADR=7] CALLI 7
TRPSET	[OP=047, ADR=25] CALLI 25
TRPJEN	[OP=047, ADR=26] CALLI 26
GETSEG	[OP=047, ADR=40] CALLI 40
GETTAB	[OP=047, ADR=41] CALLI 41
SPY	[OP=047, ADR=42] CALLI 42
GDPTIM	[OP=047, ADR=400065] CALLI 400065
XPARMS	[OP=047, ADR=400103] CALLI 400103
	[OP=042]
	[OP=044]
	[OP=045]
	[OP=046]
	[OP=052]
	[OP=053]
	[OP=054]
	[OP=700]

SECTION 13

INDIVIDUAL DEVICE DESCRIPTIONS

This section reveals the idiosyncrasies of each of the various I/O devices and of the system in handling these devices. Each device is described in a separate subsection. For features common to all (or most) of these devices, consult Section 2. For the common bits in the device I/O status word, see specifically Section 2.6.

13.1 The Disk

Disk storage is organized by files. Each file is allocated one or more disk tracks depending on the size of the file. Each track is 2.25K words long and consists of =18 records of =128 (200 octal) words each. All disk activity is in terms of whole records. In buffered mode, this means that exactly 200 words of data are transferred for each buffer, regardless of the actual buffer size; thus using a non-standard size buffer for disk I/O should not be attempted. In dump mode an output command to write a number of words that is not a multiple of 200 will cause the last record written by the command to be filled with zeroes. These zeroes will not be included in the word count for the file if and only if they are part of the last record in the file.

Warning: With a dump mode output command indicating an *odd* number of words, the low order 4 bits (the 0,,17 bits) of the last word will be written out as zero regardless of their actual values. Thus to ensure that a dump mode output does not lose any data, you should make sure either that there are an even number of words in the transfer or that the last 4 bits of the last word do not hold any significant data (for instance you can add a zero word after the last normal data word).

In buffered mode, disk I/O is optimized to account for track boundaries. On output, this means that filled buffers will not actually get written out until there is only one empty buffer left or until enough buffers have been filled to finish out a whole track. On input, as much of a whole track as will fit into your buffers is transferred all at once.

Each file belongs to some project-programmer name (PPN). The directory of all files of a particular PPN exists as a file 1) whose name is given by the sixbit PPN word (project right-justified in the left half, programmer name right-justified in the right half), 2) whose extension is .UFD and 3) whose own PPN is [1,1]. Thus the file FOOBAZ.UFD[1,1] is the directory for the disk area [FOO,BAZ]. The directory for the disk area [1,1] is the file " 1 1.UFD[1,1]", which contains the names of, and pointers to, all the directory files.

Each directory file consists of a number of 4-word entries, each of which either points to a file or is unused. These 4-word entries have the following format:

```
<file name>
<file extension>,,<creation date>
<protection, mode, and time/date written>
<pointer to file>
```

The first three words are exactly the first three words you get back from a successful LOOKUP (see page 21). The <file name> of an entry not in use is zero.

Long Block LOOKUPs, ENTERs and RENAMEs

If the 0,,400 bit (bit 27) is on in the device status word when you give a LOOKUP, ENTER or RENAME UUO for the disk, then the UUO uses a 6-word block rather than the usual 4-word block (see these UUOs starting on page 21). An ENTER using the 6-word block will set the file's date and time written directly from the third word of the block rather than from the current time and date.

The fifth word of the block is supposed to hold the date and time the file was last referenced, in the same format as the date and time written; however, after setting up this word when the file is created, the system never updates it. The sixth word of the block contains the date the file was last dumped on magnetic tape for disk backup. The format of this word is as follows:

Bits	*Octal*	*Meanings of fields in dump-date word*
4	20000,,0	This bit is a one if this dump-date word is invalid.
24:35	0,,7777	This is the date last dumped, in system date format (see the DATE UUO on page 85).
0	400000,,0	This bit is a one if the file was last dumped on a temporary class dump. This bit is zero if the dump was of system permanent class.
9:20	777,,700000	These bits hold the number of the tape on which the file was last dumped.
1:3	340000,,0	These bits hold the number of times this file has been dumped in permanent class dumps.

Record Offset

In the normal case, the data in a disk file starts in the first record of the first track allocated to the file. However, it is sometimes convenient to use the first few records of a file to store special data which is associated with the file but which is not really part of the file. To this end, the record offset feature was added to the disk system. This feature allows a user to *hide* any number of records at the beginning of a file simply by specifying the physical record number of the first record of the normal part of the file. When this is done, any program doing normal disk I/O with this file will never see the hidden records; however, any programs that need to access the hidden part can do so easily.

In a disk file, logical record number 1 is the first record of the normal part of the file. The last hidden record is logical record number 0, and preceding records are given successive negative numbers. Thus, in a file with N hidden records, the first physical record of the file would be logical record number -N+1, the remaining hidden records would be numbered, -N+2, -N+3, ..., -1, and 0. The first logical record of such a file would be physical record number N+1.

To access the hidden records (which come before logical record 1), use the USETI and USETO UUOs (see Section 2.13) with the appropriate logical record number. Note that in these UUOs the effective address is interpreted as a signed twos-complement number which specifies the logical record number of interest.

To set the record offset for a file, a special form of the MTAPE UUO for the disk is used. MTAPE for disk has several uses; you indicate the one you want by specifying a function number along with the MTAPE UUO. The function number for the set-offset UUO is 21. Another MTAPE (function number 20) can be used to retrieve the current value of the record offset. In both cases the actual offset number used is the *physical record number of the first logical record* of the file, i.e., the number of hidden records plus one. The get-offset and set-offset UUOs are described in detail below in the MTAPE writeup; see MTAPEs with function numbers 20 and 21.

Disk I/O Status Word Summary

Bits	*Octal*	*Name*	*Meaning of a 1*
27	0,,400	DMPBIT	Use 6-word blocks in LOOKUP, ENTER and RENAME.
28	0,,200	GARBIT	Suppress error message when disk is full or bad retrieval from LOOKUP, ENTER or RENAME; take error return instead.

MTAPE UUOs for the Disk

MTAPE [OP=072]

```
        MTAPE <channel number>,ADR      ;MTAPE form for the DISK

ADR:    SIXBIT /GODMOD/
        <function number>
        <other arguments depending on function>
        ...
```

This form of the MTAPE UUO is used to do special things with the disk. The exact meaning of this UUO depends on the function number in ADR+1. Some of the functions are explained below along with descriptions of what the block at ADR should contain when the UUO is called for that function. If you need to know about any of the functions not mentioned here, see a systems programmer.

To do any of the following functions, you must have INITed or OPENed the disk on the channel indicated by the AC field of the UUO; some of these functions also require that a file be open on this channel. Finally, some functions take the skip return on success and the direct return on errors; other functions always take the direct return. See the writeups below for details.

```
ADR:    SIXBIT /GODMOD/ ;GET USET POINTER
        0
```

Function 0 for a disk MTAPE gives you the current value of the USET pointer for the file open on this channel. This is the value which can be set by a USETI, USETO or UGETF UUO (see Section 2.13). The pointer is returned in ADR+1 (where the 0 was).

```
ADR:    SIXBIT /GODMOD/ ;INCLUDE RECORD IN FILE
        16              ;This function takes skip return on success.
        <record number>
```

Function 16 for a disk MTAPE allows you to include an existing record in the word count for a file. This is useful if you have managed to write out some data to extend a file but the file was never closed (system crash, etc.). The number of the last record you wish added to the file should be in ADR+2. If this function is successful, the skip return is taken and the USET pointer for the file is left pointing to the indicated record with IODEND (end of file flag) cleared. (If the record specified is not beyond the end of the file, the only result will be to have changed the USET pointer

and to have cleared IODEND, and the skip return will be taken.) If there is no file open on this channel, or if the record you specified does not exist, the direct (error) return is taken.

```
------------------------------------------------
ADR:    SIXBIT /GODMOD/ ;UPDATE RETRIEVAL
        17
```

Function 17 for a disk MTAPE forces all pointers and header information for the file being written on this channel to be updated. This is mainly useful when extending a file in Read-Alter mode. After this function has been executed, all the data written into the file is included in the retrieval. So if the system crashes, the word count for the file will be up to date.

```
------------------------------------------------
ADR:    SIXBIT /GODMOD/ ;GET RECORD OFFSET
        20
        <record number returned here>
        <physical file length returned here>
```

Function 20 for a disk MTAPE returns the physical record number of the first logical record of the file open on this channel. This record number is returned at ADR+2; the physical length of the file, including any hidden records, is returned at ADR+3. See the section above on the record offset feature.

```
------------------------------------------------
ADR:    SIXBIT /GODMOD/ ;SET RECORD OFFSET
        21
        <physical record number of first logical record>
```

Function 21 for a disk MTAPE is used to set the record offset for the file open on this channel. The physical record number of the record which is henceforth to be considered the first logical record of the file should be in the specified at ADR+2. If this function succeeds, the skip return is taken. If there is no file open on this channel, or if some other error occurs, then the direct (error) return is taken. See the section above on the disk file record offset feature.

13.2 Terminals

Here are the meanings of some of the bits in the TTY I/O status word.

Bits	*Octal*	*Name*	*Meanings of 1's in TTY I/O status word*
9	400,,0	TPMON	The terminal is in monitor mode. This bit is usually turned off when your job is running, but you can make it stay on by giving a CSTART or CCONTINUE monitor command. When this bit is on, your program will not be able to get any input from the terminal because all the characters will be going to the monitor's command decoder.
26	0,,1000	IOSUPR	Control-O ([ESCAPE] O on displays) has been typed. When this bit is on, nothing your program outputs to the terminal will be typed out. You can clear this bit by doing any TTY input operation or by re-initializing the TTY with an INIT or an OPEN. On teletypes, a second control-O will clear this bit as will [BREAK] O on displays.
27	0,,400	NOECHB	Characters typed to the program running on this terminal will not have any control bits (CONTROL or META) echoed. Control bits are always echoed when the TTY is in monitor mode (TPMON bit on--see above). The NOECHB bit can be turned on and off only by the INIT and SETSTS UUOs, except that a RESET (see page 126) will clear this bit, thus turning echoing of control bits back on.
28	0,,200	NOECHO	Characters typed to the program running on this terminal will not be echoed to the terminal by the monitor; see Section 3.1. This bit can only be turned on by UUO. A RESET (see page 126 will clear this bit, thus turning echoing back on.

13.3 The Line Printer

The line printer (LPT) has its own character set which differs slightly from the system ascii character set (see Appendix 7). Because of this, the system normally does character conversion to insure that what you get is what you want.

In addition, there are several extra characters on the line printer that have no ascii representation. To get one of these characters on the line printer, you send a 177 character followed by the appropriate code from the table below.

Code	*Character*
000	Center dot. (A period moved up to center it.)
011	Gamma.
012	Small delta.
013	Integral sign.
014	Plus-or-minus sign.
015	Circle-plus sign.
020	Skip to top of double form.
021	Space down 1 line; write over page boundary.
022	Space down 3 lines.
023	Space down to next 1/2 page boundary.
024	Space down to next 1/6 page boundary.
177	Backslash.

Line printer paper has =66 lines/page but the LPT usually skips to the top of form after =54 lines. This automatic page ejection can be overridden by use of the '177&'21 character in place of linefeeds.

If you initialize the line printer with the 100 bit (bit 32) on in the mode (I/O status word), then the conversion from system ascii to line printer codes is inhibited. In this mode, you get the following differences in characters:

Code	*Mode 0 Character*	*Mode 100 Character*
030	_ (underline)	← (left arrow)
032	~ (tilde)	↑ (up arrow)
100	@ (at sign)	' (right quote)
134	\ (backslash)	} (close brace)
136	↑ (up arrow)	circumflex
137	← (left arrow)	→ (right arrow)
140	` (left quote)	@ (at sign)
174	| (vertical bar)	\ (backslash)
176	} (close brace)	| (vertical bar)

Finally, under normal circumstances, if the line printer runs out of paper, gets jammed, or suffers some other physical ailment, you will get a system error message from which point you can CONTINUE after correcting the situation. If, however, you initialize the line printer with the 200

bit (bit 31) on in the mode (I/O status word), then when the LPT is ill an error bit will be turned on in the LPT status word and the error return will be taken by any OUTPUT UUO you try; no error message will be printed.

LPT I/O Status Word Summary

Bits	*Octal*	*Name*	*Meaning of a 1*
28	0,,200	HNGTRP	No error message on hung device (see above).
29	0,,100	LPTNCC	No character conversion (see above).

13.4 The XGP

The Xerox Graphics Printer (XGP) provides a means of making a hardcopy listing of virtually any drawing that can be expressed as a one-bit raster. The XGP accepts as data a bit array describing each scan line that is printed. Each scan line is approximately 1700 bits; scan lines are spaced at about 200 per inch along the paper. A picture is built by sending successive scan lines to the XGP. (The number of bits per scan line and the number of scan lines per inch are adjustable on the XGP and hence are not necessarily constants.)

There are presently two distinct modes of operating the XGP: video mode and character mode.

Video Mode

In video mode, 36-bit words are interpreted as video data. Words are grouped together into portions of a scan line by the use of a Group Command Word (GCW). The GCW precedes the data portion of the group and specifies how many words of video data are to be found in this group. Also the GCW allows the video data to be positioned anywhere along the scan line. The exact format of the GCW is given below.

Bits	*Octal*	*Name*	*Values of fields in GCW*
0	400000,,0	MARK	If this bit is a 1, then after the data is sent the paper will be marked for cutting. Paper cutting is not exact so a MARK should be preceded and followed by several blank lines.
1:11	377700,,0	LNSKIP	The paper will be advanced by LNSKIP blank lines before printing. LNSKIP = 1 is used for normal, single spacing. LNSKIP = 0 prevents any advance to the next scan line and prints on the same line as the last group.
12:23	77,,770000	COLSKP	The column register in the XGP interface will be set to COLSKP before the data is transmitted. This means that the first data bit in this group will appear in this column.
24:29	0,,7700	unused	This field has no meaning currently. It should be set to zero to avoid confusion in case some meaning is attached to it in the future.

30:35	0,,77	DWCNT	DWCNT words following the GCW will be transmitted to the XGP as video data (for each bit, 0 means white, 1 black). The word following those DWCNT words is then taken to be another GCW. If DWCNT = 0 then there are no data words in this group and the next word is another GCW.

Modes 17 and 117 are used for video data. These modes accept the format that is described above.

In mode 17, the effective address of the OUTPUT UUO points to a standard dump mode command list. The command list specifies the data to send to the XGP. Each OUTPUT will wait until the entire command list is processed before returning to the user. The paper will be cut at the completion of each command list.

Mode 117 is like mode 17 except that the OUTPUT UUO returns to the user while data is being sent to the XGP. In this mode the user can overlap the input of one data block with the output of another. Three data blocks are needed in this mode: one being emptied by the XGP, another pending, and another being filled by the user program. The first two OUTPUT UUOs will return immediately (having established the current and pending output blocks). After the user fills his third block and gives an OUTPUT UUO he will be forced to wait until the current block is empty (at which time the pending block becomes current and the block specified in this OUTPUT will become the new pending block). When the third OUTPUT returns, the first block will be free to use. In video mode the XGP requires up to 10,000 words of data per second. Care should be exercised in programming to always have data ready for the XGP.

Another requirement of mode 117 is that the command lists that point to the three data blocks must be disjoint. The actual requirement is that the command list for each block must be valid while the block is being output. In particular, don't use the same physical location in your program for more than one command list.

In mode 117 you must do a CLOSE UUO after the last OUTPUT to force the transmission of all buffers to the XGP. It is possible that a user program may not be able to supply data fast enough in mode 117. In this event, the paper will be cut wherever the data runs out. A status bit, (bit 25--the 0,,2000 bit, IOTEND), is provided which warns the program that this has occurred. This bit is set only in mode 117 when the data runs out and no CLOSE has been done.

Character Mode

In character mode, the XGP can be used to print text using one or more fonts and to draw vectors. Modes 0 and 13 are the character modes for the XGP. In these modes, each 36-bit word is interpreted as five 7-bit bytes. There is no fixed mapping between byte values and particular graphic symbols. The graphic symbol for any byte is defined by the current font in use. Certain byte values have special meanings consistent with ascii, and one byte value, octal 177, is used as an escape which gives the bytes that follow a special meaning.

Character mode permits vectors and multiple active text lines. The 7-bit bytes taken from the user's buffer are interpreted as follows:

Byte Value	Usual meaning	Escape significance
0	Null -- byte is ignored	Normal
1	Normal	XGP ESCAPE 1
2	Normal	XGP ESCAPE 2
3	Normal	XGP ESCAPE 3
4	Normal	XGP ESCAPE 4
5:10	Normal	Reserved
11	TAB	Normal
12	LF	Normal
13	Normal	Reserved
14	FF	Normal
15	CR	Normal
16:37	Normal	Reserved
40:176	Normal	Normal
177	ESCAPE	Normal

Normal means that the definition of this byte in the current font will be printed. If this byte is undefined in the current font, it will be ignored.

ESCAPE means that the next byte will have an alternate meaning selected from the column *Escape significance*.

TAB produces a column select to the first column which is at least the width of a blank to the right of the current column position, and some multiple of 8 blank widths to the right of the left margin.

LF activates the current text line. The current text will be queued to be printed. The default Y-position of text will be advanced by the number of scan lines it takes to draw this line, plus the number of scan lines specified by the interline space argument to the margin set function of the XGP MTAPE UUO (see page 154). This default Y-position will be used for the next text line (unless changed by a vector command or ESCAPE 3).

FF, like LF, activates the text. In addition, FF causes a page eject after the current text line is printed. FF also sets the default Y-position to the first line below the top of page margin on the new page.

CR causes a column select to the current left margin to be generated.

XGP ESCAPE 1 ('177&'001) causes the next 7-bit byte to be read as a special operation code. The following codes are implemented:

Code	*XGP ESCAPE 1 meaning*
0:17	Font select. The code, 0 to 17, is taken as the font identification number of the font to be used.
20:37	Reserved for future use.

40 Column select. The next 14 bits (2 bytes) are taken modulo =4096 as the X-position to print at next. (The intention is to allow arbitrary-width spaces for text justification.)

41 Underscore. The next 7-bit byte is taken in two's complement as the relative number of the scan line on which the underscore is to occur, where zero represents the baseline of the text, negative values represent lines above the baseline and positive values lines below it. The next 14 bits (2 bytes) are taken modulo =4096 as the length of the underscore. (If the underscore command is the first thing on a line, the baseline will be set to the baseline of the current font.)

42 Line space. This does a linefeed and then takes the next byte as the number of blank scan lines to insert.

43 Base-line adjust. The next 7 bits are taken in two's complement as the base-line adjustment to the current font. The adjustment sticks until reset by another adjust command or a font select. The intention is to allow a font to be used for subscripts and superscripts. (Increment baseline for superscript, decrement for subscript. Values 0:77 are increments; 100:177 are decrements: 100 means -100, 177 means -1.)

44 Print the paper page number. The paper page number is set to 1 by a form feed. It is incremented each time the paper is cut. The decimal value of this count is printed.

45 Accept heading text. The next byte is a count of bytes to follow. Those bytes will be read into the heading line. When that count is exhausted, the heading line will be printed. When a linefeed or line space command is given that would cause text to be printed below the current text area, a form feed is inserted by the XGP and if a heading is defined, it will be printed.

46 Start underline. Set the left end of an underline. See the stop underline command below.

47 Stop underline. The next byte is the scan line on which to write the underline (same as in XGP Underscore, 41 above). The extent of the underscore is defined by this command and the start underline command. If this command is not preceded by a start underline command, the results will be unpredictable. No underline will happen until this command is given. Beware of column selects.

XGP ESCAPE 2 ('177&'002) causes the next 7-bit byte to be taken as a twos-complement column increment. Values 0:77 are positive increments; 100:177 are negative increments: 100 means -100, 177 means -1.

XGP ESCAPE 3 ('177&'003) causes the next 2 bytes to be taken as the scan line number on which to start this text line. Scan line 0 is the first scan line on the page (immediately following the cut).

The topmost scan line of the present text line will be placed on the scan line indicated in this command. If there is no current text line, the next text line will be put there.

XGP ESCAPE 4 ('177&'004). This escape is used to specify a vector. It is followed by =11 bytes describing the vector:

```
2 bytes of Y0 Scan line number of first line of vector.
2 bytes of X0 Column position of left edge of first line of the
              vector.
3 bytes of DX Delta X. 1 bit of sign; 11 bits of integer; 9 bits
              of fraction.
2 bytes of N  The number of scan lines on which this vector is
              visible.
2 bytes of W  The column width of each scan line.
```

The XGP service must be presented with vectors sorted by ascending values of Y0. If the vectors are not sorted, the output will be wrong.

The escape significances of codes 5 through 10, 13, and 16 through 37 are not defined at the present time but are reserved for future use.

XGP I/O Status Word Summary

Bits	*Octal*	*Name*	*Meaning of a 1*
18	0,,400000	IOIMPM	Illegal mode, PDP-6 not responding, or XGP not responding.
19	0,,200000	IODERR	PDP-6 detected error: XGP reporting something wrong (out of paper, etc.), buffered mode data miss, or line too complex.
20	0,,100000	IODTER	Font Compiler lossage.
25	0,,2000	IOTEND	Data ran out before CLOSE given in mode '117.

XGP MTAPE UUO

The MTAPE UUO is used to provide extended control and status reporting of the XGP. MTAPE is not synchronized with the data stream except that certain MTAPEs imply CLOSE before their operation.

MTAPE [OP=072]

```
        MTAPE <channel number>,ADR

ADR:    <function number>
        <other data depending on function number>
        ...
```

An MTAPE which specifies a channel on which the XGP is open is interpreted as follows. The effective address of the MTAPE (ADR) points to a word containing a function number which determines the meaning of the UUO. The data at ADR+1 and following depends on the function selected.

Function	*Meaning*
0	Return error status. ADR+1/ Major error code. ADR+2,3,4/ Error data (see below).
1	Font compile and select. ADR+1/ Font file name in sixbit. ADR+2/ Font file name extension. ADR+3/ PPN of font file. ADR+4/ Font identification number: 0:17. Note: This function will skip if there is no error. The font named will be read by the font compiler. It will be assigned the font identification number that is supplied. The identification number is used only by font select commands.
2	Read margins. ADR+1/ Top of page margin. ADR+2/ Page body size. ADR+3/ Bottom of page margin. ADR+4/ Left side margin. ADR+5/ Right side margin. ADR+6/ Minimum interline space.

3 Set margins.
ADR+1/ Top of page margin; must be ≤ 37777.
ADR+2/ Page body size; must be ≤ 37777.
ADR+3/ Bottom of page margin; must be ≤ 37777.
ADR+4/ Left side margin; must be ≤ 3777.
ADR+5/ Right side margin; must be ≤ 7777 and > left margin.
ADR+6/ Minimum interline space; must be ≤ 3777.

If the bottom of page margin is zero, there will not be any paper cuts. If the page body size is zero, there will be no paper cuts except that when a FF (formfeed) is encountered, the blank space specified by the bottom of page margin will be put out and then a cut will be made.

4 Get status.
ADR+1/ The I/O device status word for the XGP.
ADR+2/ -1 if there is a data transfer in progress,
0 otherwise.

5 Pseudo close. Hardly different from CLOSE UUO.

6 Set node counts.
ADR+1/ Number of text nodes (default is =16).
ADR+2/ Number of vector nodes (default is =100).

If either node count is zero, the default value is used for that count.

The number of nodes needed for text increases with the complexity of the text (number of font switches, etc.). The number of nodes needed for vectors is related to the number and size of the vectors. Generally the default numbers of nodes is sufficient, and when it is not, there may not be enough time for the PDP-6 to do the output correctly, even if the number of nodes is increased.

Here are the meanings of the error codes returned from MTAPE function 0.

Major Error	*Meaning*
0	No error.
1	Font Compiler lossage: no job slots.
2	Font Compiler lossage: no initial response.
3	Font Compiler lossage: no intermediate response.

4 Font Compiler lossage: illegal response.
ADR+2 contains the FC response:
- 0 Ready.
- 1 Allocation made.
- 2 Compilation done.
- 3 Font compiler error.
  ADR+3 contains the error type:
  - 0 Illegal command.
    ADR+4 contains the rejected command.
  - 1 Not enough core.
  - 2 LOOKUP Failure.
    ADR+4 contains the LOOKUP error code.
  - 3 File error -- unexpected EOF.
  - 4 File error -- character redefined.
  - 5 Disk error.
  - 6 Logical font number too large.
  - 7 File error -- other illegal format.

5 Interrupt-level data missed in buffered mode.

6 XGP hung timeout.

7 Illegal mode.

10 Line too complex. The line compiler ran out of room while compiling a text line.

11 Out of order. Y0 of a vector or text line is smaller than the last item (either vector or text) that was queued. That is, the input was not properly y-sorted.

12 XGPSER missed. Somehow, the system has failed to start a vector or text node at the right place. Possibly there are too many vectors.

13 Page too long. You started a vector below the bottom of a page.

14 Illegal vector parameters. A vector you specified will go off the page.

XGPUUO UUO

XGPUUO is of interest only to the Font Compiler. This UUO is a no-op for everyone except the Font Compiler.

```
XGPUUO          [OP=047, ADR=400075]  CALL1 400075
-----------------------------------------------
        MOVE    AC, [CSB,,NSB]
        XGPUUO  AC,
```

CSB and NSB are the addresses of 20 word blocks. CSB is the Current Status Block. The Font compiler reports it's state to the system by the data it puts in the CSB before giving this UUO. NSB is the New Status Block. The system issues commands to the Font Compiler by the data it stores in the NSB when returning to the Font Compiler from this UUO. This UUO will not return until the system has something for the Font Compiler to do.

Word 0 of the status block is the opcode. All subsequent words are operands.

Opcode	*Meaning to FC from system*	*Meaning to system from FC*
0	Go away now. (System is finished with FC.)	No-op (ready).
1	Allocate. (Word 1 has size to allocate.)	Allocation made. (Word 1 has location of allocation.)
2	Compile. (File name, ext, PPN are in words 1, 2, 3; word 4 contains the logical font number, 0:17.)	Finished compiling. (Word 1 has location of base table.)

3	Lock in core. (System wants to send data to XGP.)	Compiler error. (Word 1 contains error code; see below.)

Error codes (in word 1 of block):

0	Illegal command. Word 2 has the rejected opcode.
1	Not enough core.
2	LOOKUP failure -- font file was not found. Word 2 has the LOOKUP error code.
3	File error -- unexpected EOF.
4	File error -- redundant character.
5	Disk error.
6	Logical font number too big.
7	File error -- other illegal format.

13.5 Dectapes

A dectape consists of a sequence of 1102 200-word blocks (numbered from 0 to 1101) which can be allocated to dectape files. There several formats that have been used for allocating blocks to files. The format in use at Stanford is called *old dectape format* (or sometimes *PDP-6 format*). Stanford's system is capable of reading/writing only the old dectape file format. However, the user can read and/or write almost any block on a tape (block 0 cannot be written because of the hardware) and can thus simulate any format desired (see the UDSD bit below).

Dectape I/O Status Word Summary

Bits	*Octal*	*Name*	*Meaning of a 1*
21	0„40000	IOBKTL	Dectape block number out of bounds.
29	0„100	UDSD	The system should treat the tape as if it had no directory. In this mode, the user can read and/or write blocks on the tape in any format he desires. The USETI and USETO UUOs (see Section 2.13) can be used to select which block will be read next and which block will be written next. The UGETF UUO (see page 31) will return a word whose left half contains the number of the next block to be read and whose right half contains the number of the next block to be written.

Old Dectape Format

In the old format (still standard at Stanford) block 0 is not used, block 1 is the directory which contains names of and pointers to all the files on the tape, and blocks 2 through 1101 are available for data.

Word 0 of the directory contains:

```
<LBU>,,5
```

where <LBU> is the number of the last block in use on the tape. When you do a UGETF UUO (see page 31) this number gets incremented by one and the result is returned as the number of the block you may use. The "5" points to the word within the directory block where the file entries begin.

Words 1:4 of the directory are not used at all. Words 5:174 are grouped in 4-word entries, one for each file on the tape. Thus a tape can hold a maximum of =30 files in the old format. The four words for each file contain exactly the information you get back from a LOOKUP, namely:

```
<file name>
<extension>,,<number of first block of file>
<date file written>
<whatever was in 4th word of ENTER block when file was created>
```

A zero entry marks the end of the directory.

For files written in buffered mode, each block contains (at most) 177 words of data, with the first word of each block containing

```
<BN>,,<WC>
```

where <BN> is the number of the next block in the file, or zero if none, and <WC> is the count of data words in this block.

Files written in dump mode have 200 words of data in each block and have no block-to-block pointers or word counts. Such files are always written on consecutive blocks of the tape. When a user writes a dectape file in dump mode, no information is stored with the file to indicate how long it is. Thus the user should include this information either in the data itself or possibly in the fourth word of the filename block when the ENTER is done.

UTPCLR [OP=047, ADR=13] CALLI 13

```
UTPCLR <channel number>,
```

The UTPCLR UUO causes the directory of the dectape initialized on the channel indicated to be cleared. This means that the free block pointer is set to point to block number 2 and all file entries in the directory are zeroed. The tape can then be reused as if it were a new tape. This UUO is a no-op for all devices but dectapes.

13.6 Magnetic Tapes

Data on magnetic tapes is written in files with each file being terminated with an end-of-file mark. The logical end of tape is indicated by two consecutive end-of-file marks with no intervening data.

(The mag tape is not a directory device. No filenames are associated with mag tape files, and the LOOKUP, ENTER and RENAME UUOs are no-ops with mag tapes.)

Data on mag tapes can be written in any of the standard modes. One of these is provided especially for mag tapes: mode 16--dump record mode. This mode causes data to be transferred in 200-word records, whereas normal dump mode (mode 17) transmits data in one large record.

Magnetic Tape I/O Status Word Summary

Bits	*Octal*	*Name*	*Meaning of a 1*
24	0,,4000	IOBOT	The tape is at load point.
25	0,,2000	IOTEND	The physical end of the tape has been reached. If you reach the end of the tape while reading or writing, the IOIMPM error bit will also be turned on.
26	0,,1000	IOPAR	The tape is being written or read in even parity, which is non-standard.
27:28	0,,600		These two bits indicate the tape density: 0 and 2 mean 556 bits per inch, 1 means 200 bpi, and 3 means 800 bpi. Although 800 bpi is the most efficient, 556 bpi is more reliable; 200 bpi is relatively inefficient.
29	0,,100	IONRCK	No re-reading is to be done. Re-reading is usually done on both output and input when an error is detected. After =10 tries the system gives up and sets an error bit.

MTAPE UUO for Magnetic Tapes

The following UUO is provided for doing all the special things necessary with mag tapes.

MTAPE [OP=072]

```
MTAPE <channel number>,<function number>
```

If there is a mag tape open on the channel indicated by the AC field of an MTAPE UUO, then the above form of MTAPE is assumed. The effect of the MTAPE then depends on the function number appearing in the address field. The function numbers and their meanings are given below.

Function	*Meaning*
0	Wait for any operation going on to complete.
1	Rewind the tape.
3	Write an end-of-file mark on the tape.
6	Advance the tape one record.
7	Backspace the tape one record.
10	Advance the tape to the logical end of tape. Logical end of tape is signified by two consecutive end-of-file marks. The tape is left positioned after the second mark by this operation.
11	Rewind the tape.
13	Write three inches of blank tape. The purpose of this is to cause a bad spot on the tape to be ignored. Perfectly blank tape looks like an end-of-record mark to the controller and is ignored. The monitor automatically writes blank tape over bad spots on the tape.
16	Advance the tape one file. The tape is positioned after the end-of-file mark that terminates the file.
17	Backspace the tape one file. The tape is positioned before the end-of-file mark at the end of the previous file.

13.7 Paper Tape Punch/Plotter

The paper tape punch and the plotter are considered the same device by the system; this device has the sixbit name PTP. There is a switch on the PDP-6 that determines whether data output to the PTP actually goes to the punch or to the plotter. This switch is normally kept in the plotter position.

There are several different modes in which the PTP can be operated. Normally, only one of these (mode 10) is used with the plotter. Each mode and its meaning with the PTP are described below.

Modes	*Meaning*
0,1	Character modes. Seven bit characters are punched with the eighth hole as an even parity bit. A delete (ascii 177) is punched after every carriage return, vertical tab and horizontal tab, and 30 frames of blank tape are fed after every form feed.
10,110	Image binary modes. The right-most eight bits (bits 28:35--the 0,,377 bits) of each word are punched directly as the eight bits on a frame of paper tape. This is the mode used with the plotter because the bits are sent exactly as they appear in your buffer.
13	Binary mode. Every word in your buffer is punched on 6 paper tape frames with 6 bits in each frame. The seventh hole is never punched and the eighth hole always is.
14	Checksum mode. This mode is the same as mode 13 except that before each buffer is punched, the checksum of that buffer and its data word count are punched. These two numbers appear in the left and right halves respectively of the first word (6 frames) punched. After the data words have been punched, 30 frames of blank tape are fed and then a zero word is punched (a zero word is 6 frames each with only the eighth hole punched).
100,101	No-conversion modes. These are buffered character modes in which 7-bit characters are extracted from your buffer and punched in the first seven holes of each frame on the paper tape. The eighth hole is always punched and nothing special is done for any characters.
113,114	No-eighth-hole modes. These modes are the same as mode 13 except that the eighth hole is never punched.

13.8 Paper Tape Reader

The paper tape reader (PTR) has four distinct modes, which are very similar in operation to the first four modes of the paper tape punch. Each mode and its meaning is explained below. See also Section 13.7 on the paper tape punch.

Modes	*Meaning*
0,1	Character modes. From each frame on the paper tape a 7-bit byte is taken, with the eighth hole ignored. These 7-bit bytes are packed 5 to a word in your input buffer. Nulls (ascii 0) and deletes (ascii 177) on the tape are ignored.
10	Image mode. Each frame on the paper tape is returned to you exactly as it appears on the tape, with each 8-bit frame placed in the low order bits (bits 28:35--the 0,,377 bits) of a single buffer word.
13	Binary mode. Only those tape frames with the eighth hole punched are looked at in this mode. The seventh hole of each frame is ignored completely and the first 6 holes of each frame with the eighth hole punched are returned packed 6 frames to a buffer word.
14	Checksum mode. This mode is the same as mode 13 except that the tape is assumed to contain a checksum and word count at the front of each buffer of data. The checksum is checked on input to insure accuracy of the transfers.

13.9 User Disk Pack

A user with a large quantity of data that he needs to have available on high-speed storage can make use of a User Disk Pack (UDP). The UDP (physical name UDP0) can only be read/written in dump mode. A disk pack has =7600 tracks, numbered 0 to =7599; the last track (number =7599) is used to hold a password. Each track consists of =19 records, numbered 0 to =18; record 0 is 40 words long and each of records 1 to =18 is 200 words long.

Unlike most other devices that can operate in dump mode, the UDP accepts not a list of dump mode commands but only a single dump mode command for each input or output operation. A UDP dump mode command consists of two words, the first of which is the same as a normal dump mode command and the second of which contains the UDP location where the transfer is to start. Thus a UDP dump mode command looks like this:

```
-<word count>,,<in-core location of data>-1
<record number>,,<track number>
```

where <track number> should be in the range from 0 through =7598 and indicates on which track of the pack the transfer is to start and <record number> should be in the range from 0 through =18 and indicates at which record within the specified track the transfer is to start. To effect a UDP data transfer, an INPUT or OUTPUT UUO is given with the effective address pointing to the dump mode command.

Warning: When an output indicates that only part of a record is to be written, the remainder of the record is written with zeroes. Also, as with the disk, if an *odd* number of words are written out, the low order 4 bits (bits 0,,17) of the last word are lost and are written as zero.

Before any data can be written onto a UDP, the program desiring to do the output must do an ENTER UUO to check the UDP password. The RENAME UUO is used to change a UDP password. Data can be read from a UDP without first giving the password.

If a disk error occurs during a RENAME or ENTER UUO, then unless bit 28 (the 0,,200 bit) is on in the I/O status word, an error message (BAD RETRIEVAL) will be typed out and the program stopped. If bit 28 is on when bad retrieval is detected, the error return is taken and an error code of 10 is returned in the right half of the word after the password.

ENTER [OP=077]

```
        ENTER <channel number>,ADR
        <error return>

ADR:    <password>
        <returned error code>
```

An ENTER must be done to check the password before any data can be written on a UDP. The word at location ADR is compared against the password written on the UDP, if any. If there is no

password on the UDP, if the UDP password is zero, or if the word at ADR matches the UDP password, then write permission is granted to the program and the skip return is taken from the ENTER. Otherwise write permission is rejected and the direct error return is taken with an error code of 2 (protection violation) returned in the right half of ADR+1.

RENAME [OP=055]

```
        RENAME <channel number>,ADR
        <error return>

ADR:    <new password>
        <returned error code>
```

The RENAME UUO is used to change the password for a UDP. An ENTER must have been done to acquire write permission for the UDP. The password specified in location ADR replaces the old UDP password. A zero password means an ENTER (see above) will always succeed. If the change of password is successful, the skip return is taken. If no ENTER has been done, then an error message will be printed out and the program stopped.

UDP I/O Status Word Summary

Bits	*Octal*	*Name*	*Meaning of a 1*
28	0,,200	GARBIT	Suppress error message on bad retrieval from ENTER or RENAME; instead take error return with code 10.

The following I/O UUOs are illegal with the UDP: LOOKUP, USETO, USETI, UGETF and MTAPE.

13.10 AD/DA Converter

The Analog-to-Digital/Digital-to-Analog Converter, hereinafter referred to as the AD (its physical name), operates only in dump mode. The only legal modes for the AD are 16 and 17. Mode 17 is the normal mode. In mode 16 each time a transfer is requested, it will not be started until bit 28 (0,,200 bit) of the spacewar buttons is on (see the SPWBUT UUO on page 134).

Note that output from the AD operates only the 4-channel sound system. Input can come from various sources; see the explanation of the input channel number below. Data from or to the AD is in 12-bit bytes packed three to a word.

Normally, an INPUT or OUTPUT will not return until the transfer is finished; however, if bit 29 (0,,100 bit) is on in the AD status word, the UUO will return immediately so that you may overlap computation with the transfer, for instance if you are trying to keep transfers going continuously. To wait for the last transfer to finish, use the WAIT UUO (see page 27).

The AD does not take standard dump mode command lists. Each INPUT or OUTPUT UUO should point to a 5-word block which should contain the following information:

```
-<word count>,,<address of data block>-1
<136 CONO bits>
<AD CONO bits>
<CONI bits from 136 returned here after transfer>
<CONI bits from AD returned here after transfer>
```

The 136 is an interface for the AD. The CONO bits to the 136 interface should be 4250 for input and 3650 for output (unless you know about the 136 in detail).

Here is an explanation of the AD CONO bits.

Bits	*Octal*	*Meanings of AD CONO bits*
18:23	0,,770000	This is the AD multiplex input channel number. If auto-indexing is used, this is the beginning channel number. You can find a list of the channel numbers and what devices they correspond to by looking at SAILON 21, Addendum 1.1, entitled *A/D Converter Multiplexer Patch Panel and Channel Assignments*, by Edward Panofsky. The channel number is not pertinent for output.
24	0,,4000	Reset bit. This bit should be 1 for output, 0 for input.
25	0,,2000	Input: auto-indexing. If this bit is a one, then when a sample has been transmitted, the multiplex channel number (see bits 18:23 above) will be incremented by one before the next sample is transmitted.

26	0,,1000	Input: complement low-order bit of channel number after each sample. With this bit on, if you start out with either channel 16 or 17, for example, then samples will alternate between these two channels. With input, if both this bit and bit 25 are set, a logical conflict exists and nothing will happen.
25:26	0,,3000	Output: number of channels of sound. In this field, a 0 means 1 channel, 1 means 2 channels, 2 means nothing, and 3 means 4 channels. Samples are sent alternately to the number of channels indicated.
30:32	0,,70	Clock speed. The meanings of the different values possible for this field are: 0--free run, 1--20KHz, 2--25KHz, 3--10KHz, 4--50KHz, and 5--100KHz. *Free run* means that as soon as one conversion is done another one is begun; the actual speed under this condition can be somewhat irregular and is probably somewhere between 100KHz and 200KHz. The 100KHz speed (5) is not very reliable.

The system does no error checking for you so you must do your own with the bits returned in words 3 and 4 of the block. Of these bits, the only one that is really important is the 136 data-missed bit, bit 23 (0,,10000 bit) in word 3. If this bit is on, then a sample was lost in the transfer and you should do the transfer again.

If you are running in continuous mode, (bit 29 set in status word), then the CONI words for the 136 and the AD will not be returned unless the device catches up with you. This is one way of testing for errors in continuous mode transfers; if you get the CONI words back, then either there was some data error (causing the device to finish early) or you did not supply/accept data as fast as the device could go.

Here is a sample input block to transfer 100 words to an address called ADR from channel 10 with a clock rate of 20KHz.

```
IOWD 100,ADR
4250
100010
0
0
```

Here is a similar output block.

```
IOWD 100,ADR
3650
104010
0
0
```

AD I/O Status Word Summary

Bits	*Octal*	*Meaning of a 1*
29	0,,100	Continuous mode. An INPUT or OUTPUT UUO will return without waiting for its transfer to complete. However, it will wait for any previous transfer still in progress (whether input or output) to finish before the UUO returns. Thus you cannot be doing both input and output simultaneously.

13.11 TV Cameras

For reading the TV cameras, it is recommended that you use already existing routines rather than write your own. However, for those who are stubborn enough to want to do it themselves, here is an explanation of how to read the cameras.

To read a television camera, you first INIT or OPEN device TV in mode 17 and then do dump mode inputs. The TV does not take standard dump mode command lists. The effective address of an INPUT UUO for the TV should point to a 4-word block that contains the following information:

-<word count>,,<address of first word>
<TV camera CONO bits>
<TV camera DATAO word>
<CONI bits from 167 returned here upon completion>

Thus the first word of this block contains the negative of the number of words of data to be input and the address of the first word of the block where this data is to go. (Note that this is not standard IOWD format!) The second word contains (in the right half) the CONO bits for the TV. These bits have the following meanings:

Bits	*Octal*	*Name*	*Meanings of TV camera CONO bits*
18:20	0,,700000	BCLIP	The bottom converter clip level. Each sample is a 4-bit number representing the intensity at a given point. The =16 levels are taken from the voltage range specified by BCLIP and TCLIP (see below); thus you can control the camera's exposure setting from your program. A voltage of 1.0 is put out at a very bright spot and a voltage of 0 is put out at a completely dark spot. BCLIP indicates the voltage which is to represent a sample of zero. Only voltages above this value will produce non-zero samples. See the table below for the voltage levels indicated by the various possible values for BCLIP.
21:23	0,,700000	TCLIP	The top clip level. This value determines the voltage which is to represent the maximum sample value of 17. All higher voltages will also produce a sample of 17. See BCLIP above and the table below showing voltage levels corresponding to the possible values of TCLIP and BCLIP.

Value	*BCLIP*	*TCLIP*
0	.875	1.000
1	.750	.875
2	.625	.750
3	.500	.625
4	.375	.500
5	.250	.375
6	.125	.250
7	.000	.125

24:26	0,,7000	CAMERA	The camera number. There are eight possible camera numbers, but not all of them are wired in permanently. The cart camera is number 0, the Cohu camera is number 1, the Sierra camera is number 2 and the Kintel camera is number 3.
27:29	0,,700	VRESOL	The vertical resolution divided by two, minus one. A zero here gets you every other line, a one every fourth line, etc.
30:34	0,,76		This field must contain a 1.
35	0,,1		Horizontal 1/2-sample offset. If this bit is a one, the samples within each line will be taken from positions approximately 1/2 sample width to the right of the usual positions. By doing two transfers, one with this bit off and one with it on, you can essentially double the horizontal resolution.

Now here are the meanings for the bits in the DATAO word (third word of block pointed to by the INPUT UUO).

Bits	*Octal*	*Name*	*Meaning of TV camera DATAO word*
0:8	777,,0	YDISP	The Y-coordinate displacement. Here 0 is the top of the screen; the bottom is about 400. This is the Y-coordinate of the top line that you will receive. Since you can receive at most every other line (see VRESOL above), to get an entire picture you can read the picture twice, first with YDISP set to 0 for the even numbered lines and then with YDISP set to 1 for the odd numbered lines.

9:17	777,,0	XDISP	The X-coordinate displacement. Here 0 is the left side of the picture; the right side is approximately 515.
18:26	0,,777000	WWIDTH	The width of the picture in words. Each word returned will contain 9 4-bit samples. WWIDTH specifies the number of words returned for each line. Thus you will get 9*WWIDTH samples per line. The range for this field is from 0 to 44.

The data you get from an INPUT is packed 9 4-bit samples to a word. The first sample is from the upper left hand corner of the picture specified by the X and Y displacements, horizontal width and word count. Successive samples are from points just to the right of previous points until the end of a line, after which the next sample is from the leftmost point on the next line (the exact line depending on the vertical resolution).

When the transfer is finished, the CONI bits from the IOP (the 167 data channel) are returned in the fourth word of the block pointed to by the INPUT UUO. Also, if an error occurred during the transfer and bit 29 (0,,100 bit) was on in the TV's device status word, then the address within your core image where the transfer was terminated is returned to you in the left half of the CONI word (fourth word of block). The important bits in this fourth word are explained below:

Bits	*Octal*	*Meanings of 1's in 167 CONI word*
30	0,,40	DATA MISSED. A word (9 samples) was missed by the memory and the transfer was terminated. You cannot be sure that you have a complete picture; thus you should do the transfer again.
31	0,,20	NON-EX MEM. The camera referenced a non-existent memory location. This should never happen!
32	0,,10	JOB DONE. If the transfer has been completed successfully, then this bit will be on. If this bit is not on, then the camera is hung and not responding; for instance, the camera's power might be off or your specifications might have been illegal somehow.

TV I/O Status Word Summary

Bits	*Octal*	*Meaning of a 1*
29	0,,100	Return data address at time of error in left half of CONI word.

13.12 The IMP

The IMP is the interface between the system and the ARPA network. To do anything over the network, it is necessary to use the IMP. Before explaining usage of the IMP, I will define some network terms. However, these definitions do not describe all of the network protocol. Anyone writing a program that utilizes the network should first read the network documents in order to gain some understanding of the protocol involved.

SOCKET: Any connection between two hosts involves two sockets, a send socket on one end and a receive socket on the other end. In a normal interaction, there will be two connections so that data can flow both ways; thus four socket numbers are involved: a local send socket number, a local receive socket number, a foreign send socket number, and a foreign receive socket numbers. These numbers are internal connection indices and are used to keep different interactions separate. Of these four socket numbers, each receive socket number must be even and each send socket number must be the odd number which is one greater than the receive socket number at the same end. A socket number is 32 bits long.

RFC--Request For Connection: To establish a connection, the originating host sends an RFC to the destination host with a local socket number and a foreign socket number as arguments. The destination host can complete the connection by returning an RFC with the same socket numbers, or can refuse the connection by returning a close code. Normally two connections are made for full duplex.

LINK: Once a connection is established, an 8-bit link number is assigned such that the 8-bit host number concatenated with the 8-bit link number is unique at both ends of the connection. During the lifetime of the connection, the link number is used to identity the connection to the IMP network. The IMP itself is programmed in such a way that only one message may be in transit on a particular link at a time. The IMP signals the sending host that the message has arrived by returning a RFNM (request for new message) with that link number as an argument. All of this is invisible to the user.

ALLOCATION: The host to host protocol defines a kind of flow control on a higher level than the RFNM control. This is the allocation system. When a connection is first established, the receiving host sends a control message telling the sending host how many bits and messages can conveniently be buffered. The sending host must then not send more than that many bits or messages without waiting for more allocation from the receiving host. Although much of this is unseen to the user, the user has control over how much allocation the system will give the remote host. For data and file transfer purposes, it is convenient to increase the allocation over the system default amount.

BYTE SIZE: Associated with each connection is a byte size that specifies the type of data to move on that connection. The send side indicates in its RFC what the byte size of the connection is to be; the receive side of the connection has no choice in the matter. Stanford allows byte sizes of 1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 32, and 36 bits. Most other hosts allow only 8, 32, and 36 bits. The standard TELNET connection is 8 bits with the rightmost 7 bits of each byte being an ascii character unless the high-order bit (200 bit) is on, in which case the character is a command to the receiving TELNET. A byte size is established for the life of a connection and may not be changed until the connection is broken. When you do IMP input (output), the data received (sent) is packed (unpacked) into (from) your buffers effectively with IDPB (ILDB) instructions using the connection

byte size. Thus in buffered mode, after you have INITed the IMP, you should change your buffer header's byte pointer byte size to the connection byte size. In dump mode, you should unpack (pack) the data with ILDBs (IDPBs) using the connection byte size.

IMP I/O

The IMP is similar to any other sharable I/O device in that it may be initialized with an INIT and released with a RELEAS, but it is different in that one must first open a connection before one may exchange data. Connections are opened and closed separately for the receive side and the send side, even though the I/O channel number is the same. One may open and close connections liberally without having to do more than one INIT. The INPUT and OUTPUT UUOs are used to read data from, and to send data over, the connection. The CLOSE UUO may be used to close one or both of the send and receive connections (the close-inhibit bits determine which sides are closed). To open a connection or to get one of several special functions, the user does an MTAPE UUO with the effective address pointing to a variable length table whose first word is a code number. This number determines the function of the UUO. These functions will be explained below, but first here are explanations of some bits relevant to the IMP.

The status of a connection is contained in several places. The actual connection status is in a status word, one for each connection. If you have both a receive and a send connection, then you will have two status words. MTAPE 2 gives you these status words. The bits are decoded as follows:

Bits	*Octal*	*Name*	*Meaning of a 1*
1	200000,,0	RFCS	RFC sent.
2	100000,,0	RFCR	RFC received.
3	40000,,0	CLSS	Close sent.
4	20000,,0	CLSR	Close received.
11	100,,0	INTINR	Interrupt by receiver.
12	40,,0	INTINS	Interrupt by sender.

The normal state of an open connection is 300000 in the left half, possibly with other bits on that are not mentioned above (some bits are used internally by the system). When a connection is closed or closing, CLSS or CLSR will be on. No more data may be transmitted or received over the connection at this point. The foreign host may send us interrupts. He can send an interrupt from receiver or an interrupt from sender. These can be received as user interrupts or can be tested for explicitly by MTAPE 14. Interrupts can be sent using MTAPE 11; interrupts generated by this UUO will not appear in the connection status words for the user generating them.

More status bits occur in the I/O status word, which can be gotten with a GETSTS UUO, or examined with a STATZ or STATO UUO, or cleared with a SETSTS UUO. These bits are as follows:

Bits	*Octal*	*Name*	*Meanings of 1's in IMP I/O status word*
25	0,,2000	HDEAD	Host or destination IMP dead. This means that the last message sent stayed in the IMP network for a certain length of time without being eaten by the destination. In this case, the IMP flushes the message and sends us back a *host dead* message.
26	0,,1000	CTROV	Foreign host goofed and sent us more bits than we allocated him. The system closes the connection when this happens.
27	0,,400	RSET	Foreign host sent us a reset, which asks us to clear all our tables of references to this host. The connection is closed. Some sites (including CMU and Harvard) will, as standard procedure, send a reset the first time we connect to them after their system has come up. In this case, trying again will succeed. In the more normal case, RSET means the host has just crashed and been brought up again.
28	0,,200	TMO	Timeout occurred. Your job was in a wait state and timed out (see MTAPE 17).

User interrupts can be taken on a number of conditions. These are described below. See Section 9 to find out how to enable and receive interrupts.

Bits	*Octal*	*Name*	*Interrupt condition*
11	100,,0	INTINR	Interrupt from foreign receive side.
12	40,,0	INTINS	Interrupt from foreign send side.
13	20,,0	INTIMS	Status change. One or more of the bits CLSS, CLSR, RFCS, RFCR has changed.
14	10,,0	INTINP	Some new input has arrived. The next INPUT UUO will not wait. The INTINP interrupt is generated every time a regular data message is received

by the system. One can test for data received and waiting in the system with an MTAPE 10 UUO.

IMP MTAPEs

Now on to the MTAPEs themselves.

MTAPE [OP=072]

```
        MTAPE <channel number>,ADR

ADR:    <function number>
        <other data depending on function number>
        ...
```

This is the general form of the IMP MTAPE UUOs. The function number specified at ADR determines the meaning of the UUO. What is expected and/or returned in the words following ADR is also dependent on the function number. The various functions are explained below.

Some of these functions may go into a wait state. They will be awakened either because of their normal wakeup condition, or some special condition. Some of the special conditions that may occur are specified by the bits in the right half of the I/O status word. For instance, if you are waiting for input and a reset arrives, the job is awakened and RSET is set. An I/O error bit (one of IOBKTL, IODTER, IODERR, IOIMPM or IODEND) will be set if the wakeup was due to a special condition, so the normal IN and OUT UUOs will skip if the wakeup was not due to the normal wakeup condition.

In any operation which expects the status word to be returned in word 1 (CONNECT, LISTEN, etc), an error code may be returned instead in the rightmost 6-bits. These errors are as follows:

Code	*Name*	*Meaning*
1	SIU	Socket in use. There is already a connection on the specified local socket number.
2	CCS	Can't change socket numbers. This means a foreign socket number has already been set and an RFC received for this local socket number. You can't change the socket number because the foreign host already knows what it is.
3	SYS	Horrible system error. Can't happen.
4	NLA	No link available. System IMP capacity exceeded.

5	ILB	Illegal byte size.
6	IDD	IMP dead.

Here are the various IMP MTAPE functions available.

```
-------------------------------------------------
ADR:    0       ;CONNECT
ADR+1:  <status bits returned here>
ADR+2:  <local socket number, 32 bits, right-justified>
ADR+3:  <wait flag: non-zero for wait for connection>
ADR+4:  <byte size if sending, else byte size returned here>
ADR+5:  <foreign socket number>
ADR+6:  <host number, 8 bits, right-justified>
```

An RFC is sent to the host whose number is in word 6 with the local socket number taken from word 2 and the foreign socket number taken from word 5. If word 3 is non-zero, the user program goes into RFC wait until timed out (see MTAPE 17) or until an RFC with matching socket numbers and host number arrives. If a matching RFC has already arrived from the foreign host, this UUO will not wait, but will complete the connection immediately. If you wish to open a receive connection, the local socket number must be even and the foreign socket number odd. To open a send connection, the local socket number must be odd and the foreign socket number even.

If you are opening a receive connection (local socket number is even), and either the wait flag (word 3) is on or a matching RFC was already waiting, then the connection byte size is returned in word 4. If you are opening a send connection (local socket number odd), you must specify the connection byte size in word 4.

```
-------------------------------------------------
ADR:    1       ;LISTEN
ADR+1:  <status bits returned here>
ADR+2:  <local socket number>
ADR+3:  <wait flag: non-zero for wait>
ADR+4:  <byte size if sending, else byte size returned here>
ADR+5:  <foreign socket number returned here>
ADR+6:  <host number returned here>
```

This function is used to listen for an RFC from a foreign host to a specific socket number here and to connect to that foreign host automatically when the RFC is received. If word 3 is non-zero, this UUO will wait for the connection to be made.

When an RFC for the specified socket arrives from a foreign host, a connection is made. If the local socket number is even (receive side), and either the wait flag (word 3) is on or the RFC was

already waiting for you, then the connection byte size is returned in word 4. If the local socket number is odd (send side), you must specify the byte size in word 4. The foreign socket number to which you get connected is returned in word 5; a -1 is returned if no connection has been made (wait flag off). The foreign host number is returned in word 6.

```
------------------------------------------------
ADR:    2       ;GET STATUS
ADR+1:  <status bits for local send side returned here>
ADR+2:  <status bits for local receive side returned here>
```

This function returns the status bits for both the send side and the receive side of a connection. These bits are explained on page 174. If the connection is not open, zero is stored. If a reset has been received, both CLSS and CLSR will be on.

```
------------------------------------------------
ADR:    3       ;TERMINATE CONNECTION
ADR+1:  <status bits returned here>
ADR+2:  <local socket number>
ADR+3:  <Wait flag: non-zero to wait for close>
```

This sends a close to the foreign host for the socket specified in word 2. If word 3 is non-zero, the job is placed in CLS wait until a return close is received, or the CLS timeout occurs (signified by TMO being on in the I/O status word).

```
------------------------------------------------
ADR:    4       ;WAIT FOR CONNECTION COMPLETION
ADR+1:  <status bits returned here>
ADR+2:  <socket number>
```

This is meaningful only if a CONNECT or LISTEN was given without waiting (word 3 = 0). In this case, the job is put into RFC wait until an RFC arrives or a timeout occurs.

```
------------------------------------------------
ADR:    5       ;DUMP MONITOR TABLES
ADR+1:  <number of words of tables desired>
ADR+2:  <address of place to put tables>
```

This dumps the monitor tables into your core image. The format of these tables is much too lengthy to be discussed here, but is sort of described in IMPDDB[S,SYS] pages 6 and 7. The part that is dumped is after the label SYSTBS.

```
------------------------------------------------
ADR:    6       ;WAKEUP MAIN PROCESS
```

This function is valid only when given at interrupt level. This will wake up the main process if it is in any wait state due to the IMP. It also sets TMO in the I/O status bits.

```
------------------------------------------------
ADR:    7       ;GET SOCKET NUMBERS AND HOST-LINK NUMBERS
ADR+1:  <host-link number for send side>
ADR+2:  <local send socket>
ADR+3:  <foreign receive socket>
ADR+4:  <host-link number for receive side>
ADR+5:  <local receive socket>
ADR+6:  <foreign send socket>
```

This function returns the above socket numbers and host-link numbers. A host-link number is a 16 bit right-justified quantity. The first 8 bits are the host number and the last 8 bits are the link number.

```
------------------------------------------------
ADR:    10      ;SKIP IF ANY INPUT PRESENT

This function skips if there is input waiting that has not yet been
transferred to the user's buffers.  Otherwise the direct return is taken.
```

```
------------------------------------------------
ADR:    11      ;SEND INTERRUPT
ADR+1:  <status bits returned here>
ADR+2:  <local socket number>
```

This function sends an interrupt by receiver (INR) if the socket number is even (receive side) and an interrupt by sender (INS) if the socket number is odd (send side).

```
------------------------------------------------
ADR:    12      ;RESURRECT IMP IF DOWN
```

This is a privileged UUO and is applicable only if the IMP system is down, in which case it attempts to bring it up.

```
------------------------------------------------
ADR:    13      ;BLESS HOST
ADR+1:  <host number>
```

If a destination IMP returns a host dead message, a bit is set in the system and no further transfers are permitted to that host. This UUO clears that bit and sends a null message to the host to see if he is still dead.

```
------------------------------------------------
ADR:    14      ;TEST AND CLEAR INTERRUPTS
ADR+1:  <-1 returned here if any send side interrupts>
ADR+2:  <-1 returned here if any receive side interrupts>
```

This UUO tells you if you have received either an interrupt by foreign sender or an interrupt by foreign receiver. It also clears the interrupt condition. For each of the two types of interrupts, a minus one is returned if the corresponding interrupt has been received, and a zero is returned if no interrupt has been received. When one of these interrupts arrives, the corresponding bit (INTINS or INTINR) is set in the status word for that connection, and if you are enabled for that interrupt condition, a user interrupt is generated.

```
------------------------------------------------
ADR:    15      ;ALLOCATE
ADR+1:  <type code: 0 for as specified, 1 for system max,
          2 for min, and 3 for system default values>
ADR+2:  <number of bits of allocation (for type = 0)>
ADR+3:  <number of messages of allocation (for type = 0)>
```

This changes the allocation we have given the foreign host. If this UUO is not given, the system default value is used, which is fairly small. The system maximum is =1024 words, and the system minimum is 2 words. The default value is about =50 words. We allocate in two dimensions. We allocate both the number of bits and the number of messages he may send. Since free storage blocks are about =50 words long, the system maximum number of messages is 1024/50. If the function code (word 1) is 0, the new allocation is taken from words 2 and 3, clipped to fit within the system maximum and minimum, and set accordingly. Notice that if the connection is not yet open, this does not actually cause the allocation to happen, just sets the values that will be given. Allocation does not actually occur until the first IN (INPUT) UUO is given, or an MTAPE 10 is given. Do not expect any input interrupts until one of these UUOs has been given! If we do not allocate him anything, nothing will happen!

```
-------------------------------------------------
ADR:    16      ;GET ALLOCATIONS
ADR+1:  <number of bits we have allocated him>
ADR+2:  <number of messages we have allocated him>
ADR+3:  <number of bits he has left>
ADR+4:  <number of messages he has left>
ADR+5:  <number of bits in free storage>
ADR+6:  <number of messages in free storage>
ADR+7:  <number of bits he has allocated us>
ADR+10: <number of messages he has allocated us>
```

Function 16 returns the above allocation values.

```
-------------------------------------------------
ADR:    17      ;SET TIMEOUTS
ADR+1:  <word of 6-bit bytes: number of 2-second units
          for timeouts on CLS, RFNM, ALLOC, RFC, INP;
          0 means never timeout.>
```

A job can enter a wait state waiting for any one of five different kinds of messages to arrive over the network: CLS (a return close of a connection), RFNM (a request for new message, indicating that the last message sent reached its destination), ALLOC (an allocation from a foreign receive socket, needed so that we can send some data), RFC (a request for connection, necessary to complete a connection), or INP (input data from a foreign host). In each of these cases, a timeout may be set. IMP MTAPE function 17 takes 6-bit fields from word 1 for five timeout values: CLS timeout is in bits 0:5 (770000,,0 bits), RFNM in bits 6:11 (7700,,0 bits), ALLOC in bits 12:17 (77,,0 bits), RFC in bits 18:23 (0,,770000 bits), and INP in bits 24:29 (0,,7700 bits). The value in each field is the number of 2-second units of timeout duration. The maximum timeout duration is thus =126 (2*63) seconds. Zero in any field means never time out waiting for that condition (i.e., wait forever for that condition). Bits 30:35 (0,,77 bits) are not currently used.

When a RFNM, ALLOC or INP timeout occurs, one or more I/O status error bits are set (which will cause an IN or OUT UUO to take the error return). When an RFC or CLS timeout occurs, the UUO in progress just goes on as if it had never tried to wait. When any of the five timeouts occurs, the timeout bit (TMO) is set in the connection status word.

Note that we can go into CLS wait inside a CONNECT or LISTEN if the local socket has a half-closed connection associated with it. For this reason, it is always good to wait at least a few seconds for the return CLS.

The system default timeout values are 6 seconds for CLS, 40 seconds for RFNM, and no timeout for ALLOC, RFC, and INP.

```
-------------------------------------------------
ADR:    20      ;GET TIMEOUTS
ADR+1:  <current timeout word returned here>
```

Function 20 returns the current timeout word as defined above in function 17.

SECTION 14

EXAMPLES

This section presents some examples of usage of UUOs. There are example programs that do file I/O, run display programs and use interrupts. All of the examples are written in FAIL; note the comments appearing alongside the code.

14.1 Example of General I/O

Here is a program that copies a text file called FOO into a new file called GARPLY.TMP (copying is done character by character).

```
        TITLE   EXAMPLE OF GENERAL I/O

CHR←1   ;AC to hold character being copied into new file
P←17    ;AC for pushdown pointer

IC←←0   ;channel number used for input
OC←←1   ;channel number used for output

IOTEST: RESET                           ;Good thing to start with.
        MOVE    P,[IOWD LPDL,PDL]       ;Set up pushdown pointer.

        INIT    IC,0            ;Initialize device DSK on channel IC
        SIXBIT  /DSK/           ; in mode 0 with input buffer
        IBUF                    ; header at IBUF.
        HALT    .               ;Impossible error: can't INIT the DSK.

        SETZM   INAME+3         ;Clear the PPN word of LOOKUP block--use
                                ; current DISK PPN (ALIAS) for old file.
                                ; This word is clobbered by LOOKUPs.
        LOOKUP  IC,INAME        ;Open file FOO for reading.
        JRST    NOLOOK          ;LOOKUP failed--find out why.

        INIT    OC,0            ;Initialize device DSK on channel OC
        SIXBIT  /DSK/           ; in mode 0 with output buffer
        OBUF,,0                 ; header at OBUF.
        HALT    .               ;HALT on failure.

        SETZM   ONAME+3         ;Clear the PPN word of ENTER block--use
                                ; current DISK PPN (ALIAS) for new file.
        SETZM   ONAME+2         ;Set protection to 000, use current
                                ; date/time for date/time file written.
        ENTER   OC,ONAME        ;Open file GARPLY.TMP for output.
        JRST    NOENTR          ;ENTER failed--find out why.
```

```
LOOP:   PUSHJ   P,GETCH         ;Get a character from input file.
        JRST    DONE            ;End-of-file return from GETCH.
        PUSHJ   P,PUTCH         ;Put character into output file.
        JRST    LOOP            ;Loop until EOF.

DONE:   RELEAS  OC,             ;Close output file and release DSK.
        RELEAS  IC,             ;Close input file and release DSK.
        EXIT                    ;Return to monitor level.

NOLOOK: OUTSTR  [ASCIZ /
LOOKUP FAILED -- /]
        HRRZ    CHR,INAME+1     ;Get LOOKUP error code.
        CAILE   CHR,ERRMAX      ;Make sure error code is reasonable.
        MOVEI   CHR,ERRMAX      ;Unreasonable code.
        OUTSTR  @ERROR(CHR)     ;Type appropriate error message.
        HALT    .

NOENTR: OUTSTR  [ASCIZ /
ENTER FAILED -- /]
        HRRZ    CHR,ONAME+1     ;Get ENTER error code.
        CAILE   CHR,ERRMAX      ;Make sure error code is reasonable.
        MOVEI   CHR,ERRMAX      ;Unreasonable code.
        OUTSTR  @ERROR(CHR)     ;Type appropriate error message.
        HALT    .

GETCH:  SOSG    IBUF+2  ;Decrement character count for input buffer.
        IN      IC,     ;No more in current buffer.  Read some more.
        JRST    GETCH1  ;Go get a character out of input buffer.

        STATZ   IC,20000        ;IN failed.  Have we hit end of file?
        POPJ    P,              ;Yes.  Take direct return.
        OUTSTR  [ASCIZ /
INPUT FILE ERROR.
/]                              ;No.  Type error message.
        HALT    .               ;HALT on some input error.

GETCH1: ILDB    CHR,IBUF+1      ;Get a character out of input buffer.
        AOS     (P)             ;Take skip return from GETCH with
        POPJ    P,              ; new character.

PUTCH:  SOSG    OBUF+2  ;Decrement character count for output buffer.
        OUT     OC,     ;No room.  Do output and get fresh buffer.
        JRST    PUTCH1  ;Go put character into output buffer.
        OUTSTR  [ASCIZ /
OUTPUT FILE ERROR.
/]                      ;OUT UUO failed--type error message.
        HALT    .       ;HALT on any output error.

PUTCH1: IDPB    CHR,OBUF+1      ;Put character into output buffer.
        POPJ    P,              ;Return from PUTCH.

OBUF:   BLOCK   3       ;Output buffer header
IBUF:   BLOCK   3       ;Input buffer header
INAME:  SIXBIT  /FOO/   ;Block used for LOOKUP: file name of FOO.
        BLOCK   3       ; No extension, space for date and PPN words.
ONAME:  SIXBIT  /GARPLY/;Block used for ENTER: file name of GARPLY,
        SIXBIT  /TMP/   ; extension of TMP.
        BLOCK   2       ; Space for date word and PPN word.
```

```
LPDL←←10                        ;Length of pushdown list
PDL:    BLOCK   LPDL            ;Block for pushdown list

ERROR:  [ASCIZ /NO SUCH FILE.
/]
        [ASCIZ /ILLEGAL PPN.
/]
        [ASCIZ /PROTECTION FAILURE.
/]
        [ASCIZ /FILE BUSY.
/]
ERRMAX←←.-ERROR
        [ASCIZ /HOORRIIBBLEL ERRO RNBR 87.
/]                              ;impossible error

        END     IOTEST          ;End of program, starting address.
```

14.2 Example of Display Programming

Here is a simple program to display some (fixed) text on either a III or a Data Disc. This program also positions the page printer at the bottom of the screen and draws a vector across the screen just above the page printer.

```
        TITLE   DISPLAY PROGRAMMING EXAMPLE

A←1
L←2     ;AC used to hold TTY line characteristics

DDDLIN←←20000           ;Data Disc bit in line characteristics.
IIILIN←←400000          ;III bit in line characteristics.

; Data-Disc macros and definitions
; Command word -- alternating commands and parameters:
DEFINE CW(C1,B1,C2,B2,C3,B3) <
        <BYTE (8)<B1>,<B2>,<B3> (3)<C1>,<C2>,<C3>>!4
>

; Command names for DD command bytes
EXCT←←0                         ;Execute
FNCN←←1  ALPHBG←←6  ALPHA←←46   ;Function, usual value bytes
CHNL←←2                         ;Channel select
COLM←←3                         ;Column select
HILIN←←4                        ;Set high 5 bits of line address
LOLIN←←5                        ;Set low 4 bits of line address

DISPLA: RESET           ;A good practice is to start with a RESET.
        SETO    L,      ;Get TTY's line characteristics to find out
        GETLIN  L       ; if TTY is a III or a Data Disc.
        AOJE    L,FINISH        ;Jump if job is detached (no TTY).
        TLNN    L,DDDLIN!IIILIN
        JRST    NOTDPY          ;Jump if TTY is not a display.
```

```
        PPSEL   1               ;Select new piece of paper, and
        DPYPOS  -500            ; position near bottom of screen with
        DPYSIZ  3002            ; 3 glitches and 2 lines per glitch.

        JUMPL   L,DOIII         ;Jump if III (sign bit is IIILIN)
        DDUPG   [[CW FNCN,17,CHNL,0,FNCN,ALPHA↔0]↔2]
                        ;Erase DD screen first.
        MOVE    A,[CW FNCN,ALPHA,CHNL,0,FNCN,ALPHA]
                        ;DD command word to do channel select
                        ; and set up function register for text.
        MOVEM   A,DPPROG        ;Put into display program.

        SKIPA   A,[CW COLM,2,HILIN,2,LOLIN,4]
                        ;DD command word to do column/line selects.
DOIII:  MOVE    A,[BYTE (11)<-777>,640 (3)2,3 (2)1,2 (4)6]
                        ;III long vector word to draw an invisible
                        ; absolute vector to the upper left corner
                        ; of the screen, and select brightness
                        ; of 2, size of 3.
        MOVEM   A,DPPROG+1      ;Store in display program.

        UPGIOT  1,DPHEAD        ;Run display program (on piece of
                                ; glass number 1 if III).

        JUMPL   L,DOIII2        ;Jump if III.
        UPGIOT  DDDHDR          ;Draw line across DD screen.
        JRST    FINISH

DOIII2: UPGIOT  2,IIIHDR        ;Draw vector on III, using POG # 2.

FINISH: EXIT    1,              ;Done.  Don't do normal EXIT which
                                ; would RESET the display.
        EXIT                    ;In case user types CONTINUE.

NOTDPY: OUTSTR  [ASCIZ /
This program only works on displays./]
        EXIT
```

```
DPHEAD: 200000,,DPPROG  ;Double-field mode bit,,address of disp prog
        DPLEN           ;Length of display program
        0
        DPPROG+1        ;Address of low order line select in DD prog

DPPROG: BLOCK   2       ;Space for command and position words.
        ASCID   /This is the text to be displayed on the first line.
And this is the text for the second line.

Note: The ASCID statement turns on the low order bit in each word
generated, thus causing such words to be taken as text words when
they appear in a display program for either III or DD.

Note: For DD, the last line of text must have a CRLF at its end, or
the display program must end with an execute; otherwise the last line
of text will not be displayed.

This is the last text line that will be displayed.
/
        0       ;For DD, a HALT (zero) must end the program, or the
                ; system will zero the last word of the program.
                ; Zero also stops (but is unnecessary in) a III prog.

DPLEN←←.-DPPROG ;Length of display program.

DDDHOR: DDDVEC  ;Address of DD display program to draw line.
        LDDVEC  ;Length of display program.

DDDVEC: CW FNCN,27,CHNL,0,FNCN,27       ;Graphics mode, own channel.
        CW COLM,1,HILIN,27,LOLIN,0      ;Column 1, Line 560 (2*270).
REPEAT =13,<
        BYTE (8)377,377,377,377 (4)2    ;Graphics word, all bits on.
>               ;Draw a horizontal line =52 graphics bytes long.
        CW EXCT,0,FNCN,27,FNCN,27       ;Execute to write out line.
        0       ;End of DD program to draw a line.
LDDVEC←←.-DDDVEC


IIIHOR: IIIVEC  ;Address of III display program to draw vector.
        LIIVEC  ;Length of display program.

IIIVEC: 0       ;All III programs must start with unused word.
        BYTE (11)<-777>,<-400> (3)2,3 (2)1,2 (4)6
        ;Draw invisible absolute vector to left margin near bottom.
        BYTE (11)  1777,0       (3)2,3 (2)0,0 (4)6
        ;Draw visible relative vector to right margin.
LIIVEC←←.-IIIVEC        ;Length of program to draw III vector.

        END     DISPLA
```

14.3 Example of Using Interrupts

Here is a sample program that enables interrupts on the typing of ESC I and on the receiving of a letter through the inter-job mail system (see Section 7).

```
        TITLE   INTERRUPT EXAMPLE

A←1

INTMAIL←←4000   ;Interrupt bit for letter received.
INTTTI←←4       ;Interrupt bit for ESC I typed

INTSAM: RESET                   ;A good beginning.
                ;Note that RESET clears all interrupt enablings.

        MOVEI   A,INTRPT        ;Set up address of routine to
        MOVEM   A,JOBAPR↑       ; handle interrupts.
                ;The symbol JOBAPR is EXTERNed by the ↑.

        SETZM   ESCIFG          ;Initialize ESC I flag.
        MOVSI   A,INTMAIL!INTTTI;Interrupt bits we want to enable.
        INTENB  A,              ;Enable interrupts for these bits.

        <here is main program>  ;Occasionally the main program will
        ...                     ; test ESCIFG and clear it and take
                                ; appropriate action if it is set.
        EXIT                    ;End of main program.

INTRPT: MOVS    A,JOBCNI↑       ;Get bit which is cause of interrupt.
        CAIN    A,INTMAIL       ;Did we get a letter?
        JRST    DOMAIL          ;Yes.  Go process it.
        CAIN    A,INTTTI        ;Did user type ESC I? (He must have.)
        SETOM   ESCIFG          ;Set flag to tell user-level process
                                ; that ESC I has been typed.
        DISMIS                  ;Dismiss the interrupt.

DOMAIL: SRCV    LETTER          ;Skip and read letter if one there.
        DISMIS                  ;No letter (can't happen).  Go home.
        MOVE    A,LETTER        ;Get first word of letter.
        CAME    A,CODE          ;See if letter has right format.
        DISMIS                  ;Wrong format.  Someone unknown
                                ; has sent us a letter--ignore it.

        <process letter here>   ;Interrupt-level process must not
        ...                     ; go into wait state or take more
                                ; than 8 ticks (8/60s sec) to finish.
        DISMIS                  ;Dismiss the interrupt.

LETTER: BLOCK   =32             ;Block for holding received letter.
CODE:   SIXBIT  /INTSAM/        ;Some special code to ensure we
                                ; process only reasonable letters.

ESCIFG: 0                       ;Flag indicating if ESC I was typed.

        END     INTSAM
```

APPENDIX 1

III DISPLAY PROCESSOR

The III display processor is itself a small computer. The instructions that it executes form the display. In order for a user to use the display processor, he must compile display instructions into his program and ask the system to give these instructions to the display processor. See Section 4.1 and Section 4.4 for details on getting the system to run your display program. This appendix explains the instructions that can be executed by the III display processor.

TSS Instruction

Test, set, and skip Opcode 12

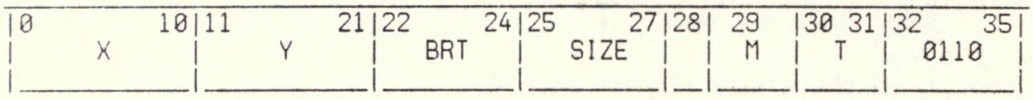
A skip condition is generated if at least one of the eight flags is on and the corresponding bit in the TEST field is on. If the exclusive or of the skip condition and bit 31 is true, the next instruction is skipped. The flags are then set or reset according to the set and reset field. If both set and reset bits are on, the corresponding flag is complemented. The flags are as follows:

Bits	*Octal*	*Flag meaning*
0,8,16	401002,,0	Control bit. This bit may be set, reset, and tested but has no other meaning to the processor.
1,9,17	200401,,0	Light pen flag. The bit is set if the light pen is seen.
2,10,18	100200,,400000	Edge overflow flag. This bit is set if the beam is ever positioned off the screen by any means.
3,11,19	40100,,200000	Wrap-around flag. This bit is set if overflow occurs in incremental vector mode.
4,12,20	20040,,100000	Not running mask. If this bit is on, the processor will interrupt if a halt is executed. This bit cannot be set or reset by this instruction.
5,13,21	10020,,40000	Light pen mask. If this bit is on, the processor will interrupt if the light pen flag comes on.

6,14,22	4010,,20000	Edge overflow mask. If this bit is on, the processor will interrupt if the edge overflow flag comes on.
7,15,23	2004,,10000	Wrap-around mask. If this bit is on, the processor will interrupt if the wrap-around flag comes on.

LVW Instruction

Long vector word — Opcode 06



The long vector word draws one vector with mode, type and brightness as specified by the M, T, and BRT fields respectively. A 0 in the BRT field indicates no change in brightness. 1 is the dimmest intensity and 7 the brightest. The brightness affects all vectors and characters until reset by another long vector word.

Mode 0 indicates relative mode and 1 indicates absolute mode. In absolute mode, the new position is given by the X and Y components relative to the center of the screen. In relative mode the components are added to the current position to give the new position.

Type	*Meaning*
0	Visible.
1	End point.
2	Invisible.
3	Undefined, currently end point.

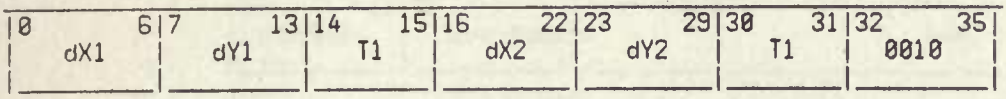
A visible vector is drawn from the current position to the new position; the invisible vector moves the beam to the new position without displaying; the end point vector moves the beam to the new position and then displays a point.

The size field sets the character size. The selected size is used for all characters until reset by another long vector word with a non-zero character size field. The sizes are:

Size	*Chars per line*	*Lines per screen*
0	no change	no change
1	128 (smallest chars)	64
2	85	42
3	73	36
4	64	32
5	42	21
6	32	16
7	21 (largest chars)	10

SVW Instruction

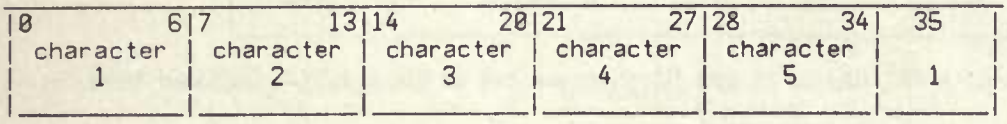
Short vector word Opcode 02



The short vector word always draws two vectors in relative mode. The type of each vector is specified by the corresponding T field (see the long vector word above). The high order bits of the dX and dY fields are extended left to give 11-bit quantities.

CHR Instruction

Character word Opcode 1

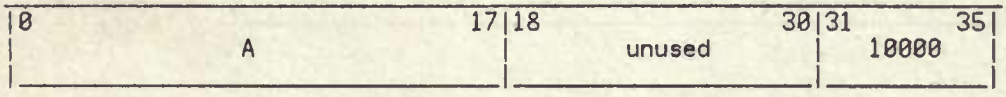


The character word displays the five characters in order from left to right with automatic spacing. All characters are displayed as printed on the line printer with the following exceptions:

Code	*111 character*
011	None.
013	Integral sign.
014	Plus-or-minus sign.
177	Backslash.

JMP Instruction

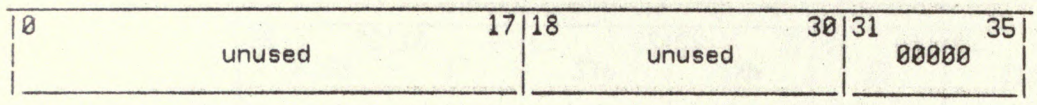
Jump Opcode 20



The processor jumps to location A and continues executing.

HLT Instruction

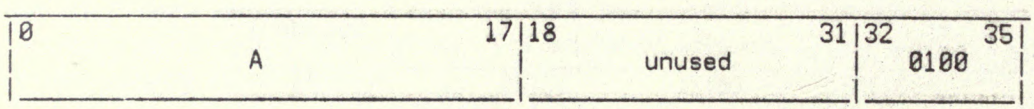
Halt Opcode 00



The processor stops with its MA pointing to the location following the HALT. The not running flag is turned on.

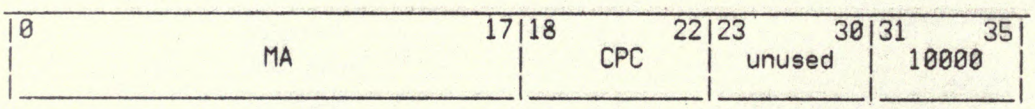
JMS Instruction

Jump to subroutine and save Opcode 04



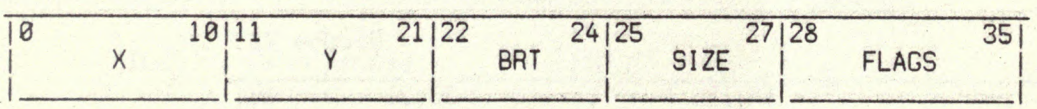
The JMS instruction is not allowed in user III programs, but its description is included here for completeness.

The following word of information is written into location A:



where CPC is the contents of the CPC buffer register. This register is loaded whenever the processor discovers an interrupt condition while processing a character word or short vector word. It is set to the number of the character being displayed (0-4) or the number of the vector of the short vector word (0-1). It is reset by a CONO 430, with the clear flags bit on.

The following information is written in location A+1:



Here are the meanings of the flag bits (see also the TSS instruction above).

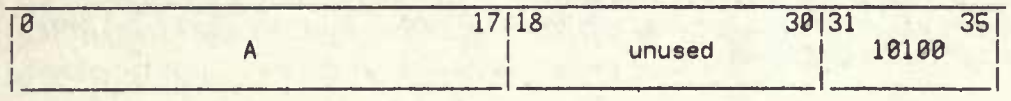
Bits	*Octal*	*Meaning*
28	0„200	Control bit.
29	0„100	Light pen flag.
30	0„40	Edge overflow flag.
31	0„20	Wrap around flag.
32	0„10	Wrap around mask.
33	0„4	Light pen mask.
34	0„2	Edge overflow mask.
35	0„1	This bit is always 1.

After storing the above words at A and A+1, the program continues executing at A+2.

Note that the word stored in A is in the form of a jump instruction. This permits the subroutine to return by jumping to A.

JSR Instruction

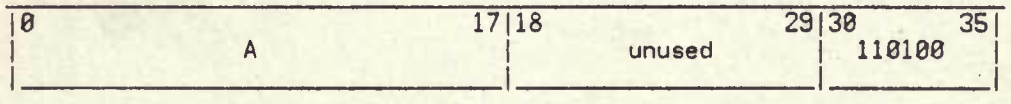
Jump to subroutine Opcode 24



The JSR instruction saves a PC word into location A and then executes code from location A+1. The PC word is in the same format as the word stored in location A by the JMS instruction. The word is a jump instruction so that the subroutine return can be simply a jump to A.

SAVE Instruction

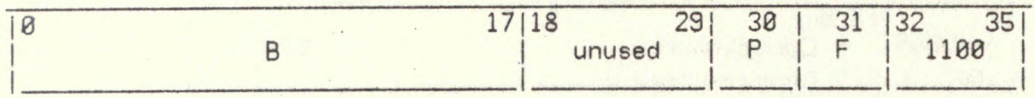
Save Opcode 64



The save instruction saves a position word in location A. This word is in the same format as the word put into A+1 by the JMS instruction and is in the correct format to be used by the REST instruction below.

REST Instruction

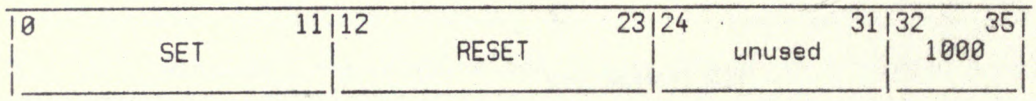
Restore Opcode 14



The contents of location B are assumed to be in the format of the word stored in location A+1 by a JMS or the word stored in location A by a SAVE. If bit 30 is a 1, the X and Y position registers and the size and brightness registers are reloaded from the corresponding fields of this word. If bit 31 is a 1, the flags are restored.

SEL Instruction

Select (displays) Opcode 10



If any of bits 0:11 (777700,,0 bits) are on, the corresponding displays are selected. If any of bits 12:23 (77,,770000 bits) are on, the corresponding displays are deselected. If both the select and deselect bits are on for a given display, the state of selection of that display will be complemented. Note that we have only 6 III displays at Stanford; only bits 0:5 (770000,,0 bits) and 12:17 (77,,0 bits) are relevant in the SET and RESET fields.

APPENDIX 2

DATA DISC DISPLAY SYSTEM

The Data Disc display system uses a =64-track disc rotating at =60 r.p.s. to store TV images. The terminals used with this system are standard television monitors. Each picture on the disc uses one *channel*, which is two disc tracks. The tracks are switched alternately every 60th of a second--one track for each field of the standard television raster. Characters and graphics are sent to a one-line buffer and then commanded to be written. Characters must be written twice; once for each field.

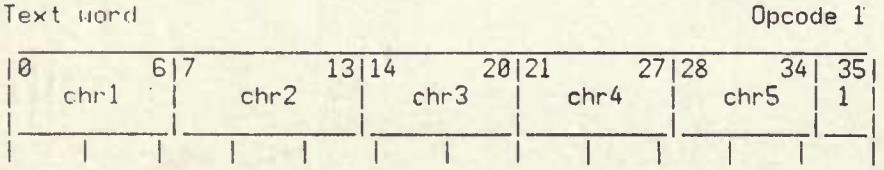
There is an interface that takes display instructions from main memory and sends data to the Data Disc. The display instructions are used to change the pictures stored on the disc. A user can have his own display programs executed by the Data Disc interface by using the UUOs explained in Section 4.4; see also Section 4.2. This appendix describes the display instructions, how they are handled by the interface and their effects on Data Disc pictures.

The Data Disc system itself operates in two modes: text and graphics. In text mode, each byte sent to the Data Disc is interpreted as a character and will be displayed in the current column on the current line; each column position is one character wide (6 bits wide in single width character mode, 12 bits wide in double width character mode). Characters must be written on the disk twice, once for each track of the given channel. In graphics mode, each byte is interpreted as eight bits to be written on the disk exactly as sent. The bits are written on the current line at the current column position. Each column position in graphics mode is 8 bits wide.

Normally, in both graphics and text modes data is sent to the Data Disc line buffer and is not written onto the disc until an execute is received by the Data Disc system. It is possible, however, to write directly onto the disc without use of the line buffer. See the Write Directly command in the command word description below.

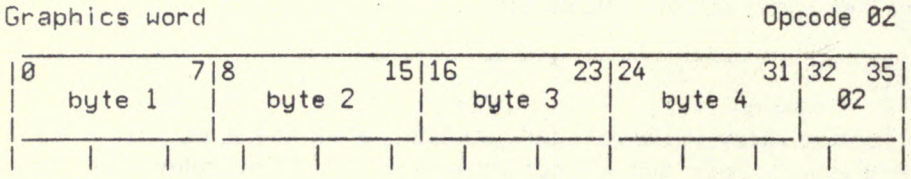
Now here are the display instructions which the interface will accept.

Text Word



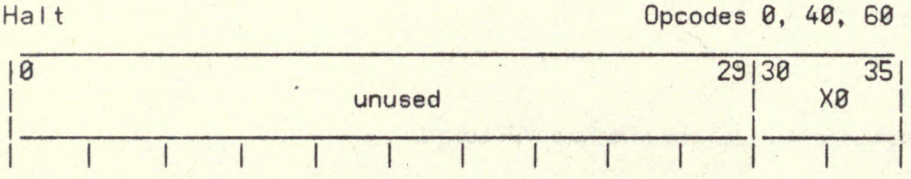
A text word causes the interface to send five bytes to the disc's line buffer. Tabs (011) and backspaces (177) are ignored unless preceded by a backspace, in which case a special character is sent to the line buffer (e.g., a small "tb" will be printed for 177&011). Nulls are always ignored. Carriage return and linefeed are specially processed to do the right thing: If any characters have been transmitted to the line buffer since the last execute command (see command word below), an execute is generated. (Execute causes the line buffer to be written onto the disc.) Carriage return then causes a select to column 2, and linefeed increments the line address by 14. (If you are displaying double-height text, then each line should be ended with a carriage return and *two* linefeeds in order for subsequent lines to be positioned properly.) Both carriage return and linefeed print special characters instead of the above functions when preceded by a 177. Note: The Data Disc must be in text mode for the bytes sent to it to be written onto the disc as text. In graphics mode, these same bytes will be written as graphic bits.

Graphics Word



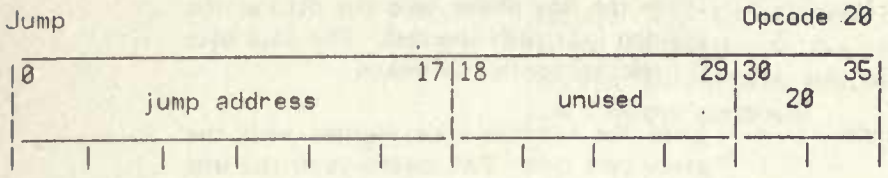
A graphics word causes the interface to send 4 8-bit bytes directly to the disc's line buffer with no modification. Note: The Data Disc must be in graphics mode for the bytes sent to it to be written onto the disc as graphics bits. In text mode, these same bytes will be written as characters.

Halt Instruction



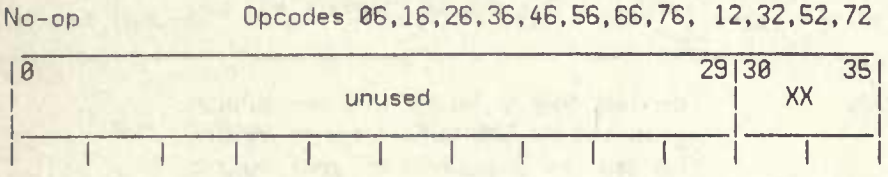
A halt stops the interface, terminating the display program.

Jump Instruction



A jump word causes the interface to take subsequent display instructions starting at the address contained in the left half of the jump word.

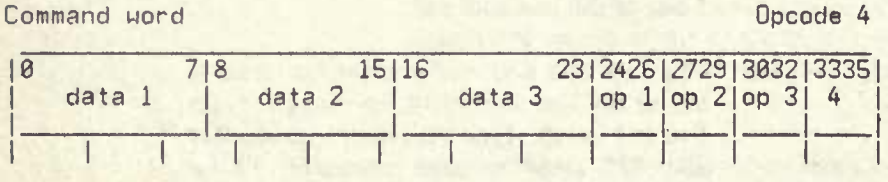
No Operation



Opcodes ending in 6 or 12 have no function. The interface ignores them and proceeds.

Warning: Opcodes ending in 10 (10, 30, 50, 70), previously advertised as no-ops, are not no-ops at all; they are actually command words, although their use as such is not recommended because the 10 bit is part of the third command code (*op 3* in the command word below).

Command Word



A command word causes the interface to send the Data Disc three commands, as indicated above by "op 1," "op 2" and "op 3." Sent with each command is an 8-bit data byte. The commands possible are as follows:

Command	*Name*	*Meaning*
0	Execute	Write the line buffer onto the disc at the position previously specified. The data byte is irrelevant for this command.
1	Function code	Load the function code register with the given data byte. The meanings of the bits in the function code register are explained below.
2	Channel select	Select the channel specified in the data byte for writing. If the erase bit is on and the graphics mode bit is set, the channel selected is erased to the background selected by the dark/light bit. Only the first channel select in a DD display program has any effect.
3	Column select	The data byte is loaded into the column register and the line buffer address register. This sets the X-position of your output. Column 0 is illegal and will hang the controller. Column =85 is the last column to be displayed with normal size characters; characters sent for columns =86 through =128 are flushed, over =128, you wrap around. A column select greater than =85 will also hang the controller. The last graphics column is =64 and columns greater than that will hang the controller.
4	High order line address	The data byte is loaded into the high order 5 bits of the line address.
5	Low order line address	The data byte is loaded into the low order 4 bits of the line address. Line range is from 0 to 737 (octal). Line addresses between 740 and 777 cause execute commands to be ignored. Line addresses above 777 wrap around.
6	Write directly	Data is written directly on the disc at the location previously set up; the line buffer is not used. The column address is automatically incremented. Executes are not necessary.

7	Line buffer address	Data is loaded into the line buffer address only. This allows some of the line buffer contents to be changed and the rest retained. The first character displayed will be the one specified by the column address, and the last character will be the last one sent after this command.

The 8-bit function code register, which is loaded by command 1 above, has the following format:

Bit# 0	1	2	3	4	5	6	7	
unused	unused	single height	nospace (add)	2 wide (erase)	dark back	write enable	graphic mode	1
unused	unused	double height	space (rep)	1 wide	light back	display direct	text mode	0
MSB							LSB	

Bits	*Octal*	*Meaning*
2	040	Single height/double height. (Text mode only.) This bit determines what height displayed characters will have. Single height characters are 12 lines tall; 10 lines above the "base" line and 2 lines below. The top line of the character prints on the line addressed. This bit has no effect in graphics mode. If you are displaying double-height text, then each line should be ended with a carriage return and *two* linefeeds in order for subsequent lines to be positioned properly.
3	020	Space/nospace. (Text mode only.) When this bit is on, each character written is substituted on top of the character previously written; when this bit is off, the remainder of the line is erased.
3	020	Additive/replacement. (Graphics mode only.) When this bit is on, only 1 bits are written, ORed with the bits already written; when this bit is off, 1's and 0's are written over previous data. CAUTION: When replacing, the bits at the beginning and end of the line segment you are writing should be the same as the previous data or bit lossage may occur.

4	010	Double width/single width. (Text mode only.) This bit determines what width displayed characters will have. With this bit on, characters are 5 bits wide with a 0 bit on the end (total 6 bits); when this bit is off, characters are =10 bits wide with two 0 bits on the end. CAUTION: When using double width characters, do not exceed =43 characters in a line or the controller will hang. Double width characters often fail to get displayed correctly anyway.
4	010	Erase. (Graphics mode only.) If this bit and the graphics mode bit are on, a channel select will cause the screen to be erased to the background indicated by the dark/light bit below.
5	004	Dark/light. When this bit is on, an erase causes the screen to go dark and characters and graphic 1 bits are displayed as light. When this bit is off, an erase makes the background light and characters and graphic 1 bits are displayed as dark.
6	002	Write/display directly. When this bit is on, operations go to the disc (normal mode). When off, data is displayed once on the selected channel and then goes away and previous data remains on the disc.
7	001	Text/graphics mode. When this bit is on, you are in graphics mode; when it is off, you are in text mode.

APPENDIX 3

GENERAL INFORMATION ABOUT THE PDP-10

The PDP-10 is a 36-bit word, single address machine. Normal instructions (excluding I/O instructions) take the following form.

Bits	*Octal*	*Meanings of fields in an instruction word*
0:8	777000,,0	Instruction code.
9:12	740,,0	Accumulator.
13	20,,0	Indirect bit.
14:17	17,,0	Index register if non-zero.
18:35	0,,777777	Memory address.

For I/O instructions, the following format is used:

Bits	*Octal*	*Meanings of fields in I/O instructions*
0:2	700000,,0	This field holds a 7 in all I/O instructions.
3:9	77400,,0	Code of I/O device.
10:12	340,,0	I/O instruction code.
13:35	37,,777777	Same as in normal instructions.

Machine I/O instructions are generally privileged in that only the monitor is allowed to issue such instructions. The opcodes corresponding to these instructions trap to the monitor, which normally interprets them as UUOs. However, the user is permitted to give machine I/O instructions when he is in IOT-USER mode, which is described below.

For every instruction, an effective address calculation is carried out in the following manner using bits 13:35 of the instruction. If the index register field (bits 14:17) is non-zero, the contents of the specified index register are added to the memory address found in bits 18:35 of the instruction. Then if the indirect bit (bit 13) is 0, this result is the effective address for the instruction. If the indirect bit is 1, then another word is picked up from the location specified by the result so far. Then the effective address calculation starts over using bits 13:35 of this new word. This process continues until the indirect bit in some word used in this calculation is 0; the address (including any indexing) specified by that word is then the effective address of the instruction.

PC Flags

The PDP-10 has several flags that indicate the state of the program running. These flags are commonly called the PC flags because their values are stored in the left half of the PC word stored by the PUSHJ, JSR and JSP instructions. The PC itself is stored in the right half of this PC word. The bit positions and meanings of the flags are explained below. For further detail, see the PDP-10 instruction manuals.

Note that there are no flags in bits 13:17; these bits are always stored as zero in the PC word so that when the PC word is addressed indirectly neither indexing nor further indirecting will take place.

Bits	*Octal*	*Meanings of flags in the PC word*
0	400000,,0	Overflow. An arithmetic operation has resulted in an incorrect result because of some kind of overflow.
1	200000,,0	Carry 0. An arithmetic operation has caused a carry out of bit 0. This does not necessarily indicate an overflow condition.
2	100000,,0	Carry 1. An arithmetic operation has caused a carry out of bit 1. This also does not necessarily indicate an overflow condition.
3	40000,,0	Floating overflow. A floating point instruction has resulted in an incorrect result because of some kind of overflow or underflow.
4	20000,,0	Byte-increment suppression. The next ILDB or IDPB instruction will be executed without incrementing the byte pointer. This flag is set when an ILDB or IDPB instruction is interrupted after the byte pointer has been incremented but before the byte has been moved. This flag can also be set by a user program with the "JRST 2," instruction.
5	10000,,0	User mode. The processor is in user mode.
6	4000,,0	User in-out. The processor is in IOT-USER mode. This mode is explained below.
11	100,,0	Floating underflow. The exponent of the result of a floating point operation was less than -128 (decimal).
12	40,,0	No divide. A division operation was not carried out because either the divisor was zero or an overflow would have resulted.

IOT-USER Mode

On the PDP-10, usually only the monitor is allowed to use the machine's I/O instructions. In user programs, the opcodes (700:777) of these instructions are usually interpreted as UUOs. However, there exists a special mode, called IOT-USER mode, in which the opcodes 700:777 are executed not as UUOs but as machine I/O instructions.

A user program can get into IOT-USER mode by giving the EIOTM UUO (see page 135), which does nothing but put you into this mode. Also, an interrupt-level process will be started up in IOT-USER mode after a new style interrupt if bit 6 (the 4000,,0 bit) is on in JOBAPR; see Section 9. Finally, spacewar processes are always started up in IOT-USER mode; see Section 8.

The simplest method for getting out of IOT-USER mode, whether at user, spacewar or interrupt level, is to execute a

```
JRST 2,@[.+1]
```

This turns off all of the PC flags, except the user-mode flag. See the JRST instruction in the PDP-10 manuals and note that DEC calls IOT-USER mode "user in-out".

In summary, opcodes 700:777 are treated as UUOs unless the program executing such instructions is in IOT-USER mode, in which case these opcodes are machine I/O instructions.

Text Representations

There are several common representations used for storing text on the PDP-10. These include ASCII, ASCIZ, ASCID and SIXBIT. In each of these four representations, characters are stored left-justified in a block of one or more words. The octal codes for characters are given in Appendix 7.

ASCII — In this representation 7-bit characters are packed 5 to a word with the low order bit (bit 35--the 0,,1 bit) always being zero. A word count or character count is needed to determine the length of an ASCII string.

ASCIZ — This is the same as ASCII except that a null (zero) byte follows the last character in the string to mark its end.

ASCID — This is the same as ASCII except that the low order bit of each word is always a one.

SIXBIT — This representation packs 6-bit characters 6 to a word. The characters representable in SIXBIT are those with ascii representations from 40 to 137.

Assembler Features Relevant to this Manual

Here is an abbreviated list of PDP-10 assembler features, including those used in the UUO writeups in this manual.

1. The form "[...]" is a literal whose value is the address at which the code and/or data specified by "..." is placed in the program. The code/data can consist of any number of words and can include other literals.

2. The form "A,,B" represents a word whose left half contains the value A and whose right half contains the value B. This is equivalent to the form "XWD A,B".

3. The form "IOWD A,B" is exactly equivalent to the form "-A,,B-1".

4. The forms "ASCII /.../", "ASCIZ /.../", "ASCID /.../" and "SIXBIT /.../" represent blocks of words containing the string "..." in the ASCII, ASCIZ, ASCID and SIXBIT representations respectively. See these representations above.

APPENDIX 4

JOB DATA AREA

The first 140 words of each core image are reserved for storage of various parameters for that job. This block is called the user's Job Data Area. The data here can be examined by the user, and he can change some of it directly simply by storing new values in the appropriate words.

Part of this data, however, is important to the system and is protected from any attempt by the user to change it. In particular, the block from JOBHCU to JOBPFI (see table below) is copied into the monitor when the job is run so that the user cannot change it. When the job is not running, this block is stored back in the user's job data area to conserve space in the system.

References to locations in the job data area should be by their symbolic names rather than by their absolute addresses. It is possible that some of these locations might be moved around, and if that happens, programs that refer to these locations symbolically will need only to be reloaded. All of the symbols for these words are defined in a system library file; their definitions will automatically be retrieved from there by the LOADER if the symbols are declared EXTERNAL.

The table below explains the names and uses of the locations in the job data area.

Word	*Symbol*	*Explanation*
0	JOBAC	The user's accumulators are stored here over UUO calls.
20	JOBDAC	The user's accumulators (whether Exec mode or user mode) are stored here when a clock trap occurs.
40	JOBUUO	User UUOs (opcodes 001:037) encountered are stored here after the effective address calculation has been done. See Section 1.4.
41	JOB41	The instruction in this location is executed whenever a user UUO (opcodes 001:037) is encountered. This instruction is often a JSR to the user's UUO-handling routine. See Section 1.4.
43	JOBENB	The user's APR trap enablings (old style interrupt enablings) are stored here. See Section 9.
44	JOBREL	The job's protection constant is stored here. This is the highest address in the user's core image (excluding any upper segment).
45		This is the first of several words used for temporary storage by the system.

71 JOBINT If this word is non-zero, it specifies the address in the user's core image of a block of three words to be used in place of JOBCNI, JOBTPC and JOBAPR, respectively, for new style interrupts. See Section 9.

72 JOBHCU (Also know as JOBPRT.) The number of the highest I/O channel in use by this job is stored here. This is the first location of the block copied into the system when the job is run.

73 JOBPC Your program counter is stored here when your program is not running.

74 JOBDDT If DDT or RAID is present in your core image, its starting address is stored here. When you type the DDT monitor command, your program is started at the address specified in this word, unless this word contains zero. Also, when RAID is present, bits 0:12 (the 777740,,0 bits) of this word hold the version number of RAID. This is first word written into a dump file by the SAVE monitor command.

75 JOBJDA This is the first word of a 20-word block in which are stored the system addresses of the device data blocks (DDBs) for the devices you have open on the 20 logical I/O channels. The address of the DDB for channel 6, for instance, is stored in JOBJDA+6.

114 JOBPFI This is the last word of the 20-word block starting at JOBJDA and is also the last word of the block (starting at JOBHCU) that is copied into the system when you run.

115 JOBHRL The right half of this word contains the highest address in your upper segment (e.g., 401777 for a 1K upper), and the sign bit (400000,,0 bit) is on if your upper is write protected. If you have no upper, this word is zero.

116 JOBSYM A pointer to your symbol table is placed here. If you have no symbols in your core image, this word is zero. Otherwise, the left half of this word contains the negative of the number of symbols in the table, and the right half contains the address of the first word of the table.

117 JOBUSY A pointer to the table of undefined globals encountered is placed here by the LOADER. This word has the same format as JOBSYM above (i.e., -<word count>,,<table>). If there were no undefined globals, the left half of this word will contain zero.

120 JOBSA The right half of this word contains the starting address of

your program. The START and CSTART monitor commands cause your program to be started at that address (unless it is zero). The left half of this word contains the address of the first location above your symbol table (or program if no symbol table). The RESET UUO (see page 126) causes this number to be stored in JOBFF (see below).

121 JOBFF The address of the first free (unused) location in your core image is stored here. The words from that address up to that in JOBREL are not used. Whenever the system sets up an I/O buffer for you, it is put at the address specified by JOBFF and JOBFF is then increased to point to the first word beyond the buffer. The RESET UUO causes JOBFF to be reset to the value stored in the left half of JOBSA (see above).

122 JOBS41 The contents of JOB41 (see above) are stored here just before the core image is written into a dump file by a SAVE or SSAVE monitor command or by the SWAP UUO. JOB41 is restored from this word when the file is read by a GET, RUN or R command or by the SWAP UUO.

123 JOBEXM This is a temporary cell used by the EXAMINE and DEPOSIT monitor commands.

124 JOBREN This word contains the address at which the program should be started when the REENTER monitor command is given. A zero in this word means REENTER will not work.

125 JOBAPR This word should contain the address at which you wish your interrupt routine started when an interrupt occurs. See Section 9.

126 JOBCNI When an interrupt occurs, a bit indicating the cause is stored here. See Section 9.

127 JOBTPC When an interrupt occurs, the program counter word is stored here before control is given to your interrupt routine. See Section 9.

130 JOBOPC When you start the program with a START, CSTART, DDT or REENTER monitor command, the program counter word (picked up from JOBPC) is stored here.

131 JOBCHN This word is used for FORTRAN chaining.

132 JOBFDV This word is used for temporary storage by the FINISH command.

133	JOBCOR	This word is used for temporary storage by the SAVE and GET commands.
134	HINAME	The name of your upper segment, if any, is stored here just before your core image is written onto a dump file by a SAVE or SSAVE monitor command or by the SWAP UUO.
135	HILOC	The location within the dump file of your upper segment is stored here prior to writing a dump file with the SSAVE monitor command.
140	JOBDA	This is the first word in your core image that is not part of the job data area.

APPENDIX 5

LOCATIONS OF USEFUL POINTERS IN THE MONITOR

The table below lists the contents of some absolute locations in the monitor. Most of these locations contain pointers to system tables that user programs occasionally need to examine. Any word in memory may be examined by use of either the PEEK UUO (see page 93) or the SETPR2 UUO (see page 92).

To get the address of a particular job's entry in a job table, add the job number to the base address of the table.

Word	*Contents*	*Explanation*
210	JBTSTS	This is the address of the job table of status words. The bits in each entry are explained with the JBTSTS UUO on page 90.
211	PRJPRG	This is the address of the job table of project-programmer names.
212	JBTSWP	This is the address of the job table of swapper data. The left half (bits 0:17--the 777777,,0 bits) of each entry contains the logical band number used to swap this job. Bits 18:26 (0,,777000 bits) contain the size of the job (in 1K blocks) as stored on the disk. Bits 27:35 (0,,777 bits) contain the size of the job (in 1K blocks) as it will appear when it is swapped in.
213	SPWGO	This is the address of the job table of PDP-10 spacewar processes. The right half (bits 17:35--0,,777777 bits) of each entry contains the user-specified starting address of the spacewar process. Bits 14:17 (17,,0 bits) contain the number of ticks between startups, and bits 10:13 (36,,0 bits) contain the number of ticks until the next startup. This word is zero for jobs that do not have PDP-10 spacewar processes running. See Section 8.
214	TTIME	This is the address of the job table of total run times (in ticks). The time for job 0 is the null time (idle time) since the last system reload or restart.
215	UPTIME	This is the address of a word that contains the length of time in ticks since the last system reload or restart.
216	CORMAX	This is the address of a word that contains the largest size (in 1K blocks) a user program can be and still fit in core.

217 DEVLST This is the address of the header word for the list of all device data blocks (DDBs). The left half of the header word contains the address of the first DDB. DDBs are described in Appendix 6.

220 TTYTAB This is the address of the table of pointers to TTY DDBs. The entry for a TTY in use contains the TTY line number in the left half and a pointer to the TTY's DDB in the right half. The entry for a TTY not in use is zero. Index into this table with the TTY line number.

221 BYTE (9) SCNNUM,DPYNUM,DDNUM,PTYNUM These four quantities are the numbers of 1) teletype lines, 2) III display lines, 3) Data Disc display lines and 4) pseudo-teletype lines, respectively.

222 JOBN-1 This is the highest possible job number.

223 JBTADR This is the address of the job table of protection-relocation constants. The left half of each entry contains the job's protection, which is the highest location addressable in the job (not counting any upper segment). The right half of each entry contains the job's relocation, which is the physical memory address where the job's core image is actually located. When a job is swapped out, its entry in this table is zero.

224 JBTQ This is the address of the job table of entries in the job queues. The queues are circular with each entry containing a forward pointer in the right half and a backward pointer in the left half. The pointers are all relative to JBTQ: a pointer that is positive points to another job's entry in the table; a pointer that is negative points to the queue's header word, which itself is just another entry in the circular queue and which contains both forward and backward pointers. The magnitude of a negative pointer indicates the number of the queue the entry is in.

225 JOBNAM This is the address of the job table of sixbit job names.

226 JOB This is the address of a word that contains the number of the currently running job.

227 CONFIG This is the address of an ASCIZ string that gives the title of the current system.

230 SP2GO This is the address of the job table of PDP-6 spacewar processes. The format of this table is the same as that of the SPWGO table; see word 213 above.

231 JOBQUE This is the address of the job table of queue numbers. Each entry in this table contains either the queue number or the negative of the queue number for the particular job.

234 NQUES This is the number of different queues. The queues are numbered from 1 to this number.

235 QNAMS This is the address of a table of the ASCII names of the various queues. Index into this table with the queue's number.

236 JBTLIN This is the address of the job table of attached terminals. The entry in this table will be -1 for a detached job. For an attached job the entry will contain the TTY line number in the right half and the permanent TTY line characteristics in the left half. There may be bits on for non-permanent characteristics, but these bits are not kept up to date in this word. See LINTAB in word 302 below for the current line characteristics; see also the GETLIN UUO on page 40.

237 LETAB This is the address of the table of pointers to the displays' line editor headers. The right half of each entry contains a pointer to the free storage block for that display. The left half holds various flags used by the line editor. When a display is not in use, its entry here is zero. Index into this table with the number of the display line minus 20.

250 JBTKCJ This is the address of the job table of kilo-core-jiffies (KCJs) used by each job. A jiffie is a tick, i.e., 1/60th of a second. One KCJ represents a job running for one tick with 1K of core.

251 JBTBTM This is the address of the job table of login times. Each entry in this table contains the date and time when the particular job logged in, with the date (in system date format) in the left half and the time (in seconds after midnight) in the right half.

257 SHFWAT This is the address of a word that contains the number of the next job to be shuffled in core.

265 SYSTOP This is the address of a word that contains the physical address of the first word in memory that can be allocated to user programs.

266 CORTAB This is the address of a table that indicates the usage of each 1K block of core. The entries in this table are 9-bit bytes packed 4 to a word. The entry for a given block

contains either 1) the number of the job occupying the block, 2) a 101 if the block is part of the system, 3) a 105 if the block is part of free storage or 4) a 0 if the block is unused.

270 PTYJOB This is the address of the table of owners of pseudo-teletypes (PTYs). Each entry contains the number of the job that owns that PTY. Index into this table with the PTY line number minus 121.

271 JBTPRV This is the address of the job table of privileges.

272 UCLLEN*1000+UCLDLN,,UCLTAB The right half of this word contains UCLTAB which is the address of the CALL UUO name table. This table is made up of two parts. The first UCLDLN words contain the names of DEC CALLs, and the next UCLLEN-UCLDLN words contain the names of the special Stanford CALLs (numbers from 400000 up). Thus the total length of the table is UCLLEN, which is in bits 0:8 (777000,,0 bits) of word 272; the length of the first part of the table, UCLDLN, is in bits 9:17 (777,,0 bits) of word 272. The CALL names within each part are in their expected order by CALLI number.

273 DSKPPN This is the address of the job table of Disk PPNs (ALIASes). If a job has no ALIAS, its entry in this table is zero. Disk PPNs are explained on page 16.

274 FTIME This is the address of the job table containing each job's time last run. Each entry contains the date and time when the particular job was last run, with the date (in system date format) in the left half and the time (in seconds after midnight) in the right half.

275 NJOBS This is the address of the job table that gives the number of users for each upper segment. Each entry contains the number of jobs attached to the given upper segment. Index into this table with the upper segment's job number.

276 DSKOPS This is the address of the job table containing the number of disk operations each job has done. The entry for job 0 is the total number of disk operations since the system was reloaded.

277 INITIM This is the address of a word that contains the date and time of the last system reload or 203 restart. The date (in system date format) is in the left half and the time (in seconds after midnight) is in the right half.

300 -DISPL,,COMTAB COMTAB is the address of the table of monitor commands names in sixbit, and DISPL is the number of commands in that table.

302 LINTAB This is the address of the table of TTY line characteristics. Each entry in this table corresponds to a particular TTY. The TTY's line characteristics are in the left half (see the GETLIN UUO on page 40 for the meanings of these bits). The right half is used to store ESCAPE arguments (for TTYs that are displays). Index into this table with the TTY line number of interest.

303 ASTAB This is the address of the audio switch connection table. There is a word here for each display; index into this table with the display's line number minus 20. The data in each entry of this table is as follows (for more details about the audio switch, see Section 4.7):

Bits	*Octal*	*Values of fields in ASTAB entry*
0	400000,,0	One if a UUO is waiting for a temporary connection to finish on this display.
1	200000,,0	One if the current connection is a temporary connection.
2	100000,,0	One if a page interruption is happening at this display.
3	40000,,0	One if a delayed beep is pending.
4	20000,,0	One if the permanent audio channel is not page interruptible.
5	10000,,0	One if the permanent channel is not beep interruptible.
6:7	6000,,0	Beep disposition for temporary channel.
8:9	1400,,0	Page disposition for temporary channel.
10:13	360,,0	Temporary channel number.
14:17	17,,0	Permanent channel number.
18:35	0,,777777	Remaining duration of temporary connection (0 for infinite).

APPENDIX 6

DEVICE DATA BLOCKS (DDBS)

For each device used, there is a Device Data Block (DDB) in the system in which is kept a collection of data pertinent to the device. Certain devices (including the disk, the IMP, and TTYs) do not have DDBs when they are not in use; other devices' DDBs are permanently built into the list. With sharable devices such as the disk, there is a DDB for each user I/O channel with the device open.

All the DDBs are linked together in one big list. The header for this list is located at DEVLST; the left half of the word at DEVLST points to the first DDB. (The word DEVLST is pointed to by the word at absolute memory address 217; see Appendix 5.)

The data common to all DDBs is explained below. Other data is device dependent.

Word	*Name*	*Contents*
0	DEVNAM	This is the device's physical name in sixbit.
1	DEVCHR	Bits 0:5 (770000,,0 bits) of this word contain the number of the job the device belongs to, or zero if the device is unused; if this field contains zero and the device is ASSIGNed (see the DEVCHR UUO on page 33), then the device has been detached from the system (probably for maintenance). Bits 6:11 (7700,,0 bits) contain the current hung time count down in seconds. Bits 12:17 (77,,0 bits) contain the hung time in seconds. Bits 18:23 (0,,770000 bits) contain the unit number for multiple unit devices like dectapes and magnetic tapes. Bits 24:35 (0,,7777 bits) contain the size in words of the buffer this device uses.
2	DEVIOS	This is the device's I/O status word. See Section 2.6.
3	DEVSER	The right half of this word contains the address in the system of this device's UUO dispatch table. The left half contains the address of the next DDB in the list, or zero if this is the last DDB.
4	DEVMOD	This is the word returned by the DEVCHR UUO (see page 33).
5	DEVLOG	This is the device's logical name in sixbit, if any.

6	DEVBUF	The left half of this word contains the user address of the output buffer header, if any. The right half contains the user address of the input buffer header, if any.
11	DEVFIL	For directory devices like the disk and dectapes, this word contains the sixbit name of the file that is currently open, or zero if no file is open.
12	DEVEXT	For directory devices the left half of this word contains the file name extension of the file currently open.
13	FILPRO	For disk files this word contains the protection, mode and date/time written. For dectape files this word contains the date written. See the LOOKUP UUO on page 21.
14	FILPPN	For disk files this word contains the project-programmer name of the file currently open.

APPENDIX 7

STANFORD CHARACTER SET

The table below gives the octal codes for characters in the Stanford ascii character set. The octal code for a character is three digits and is obtained for a character in the table by adding the label of the character's row to the label of the character's column. For example, the ascii code for "G" is 100+7, or 107.

To get the sixbit code for a character, find the ascii code and subtract 40. Note that the only characters with sixbit representations are those with ascii codes in the range 40:137.

The abbreviations used for special characters in the table are explained below.

	0	1	2	3	4	5	6	7
000	NUL	↓	α	β	∧	¬	∈	π
010	λ	TAB	LF	VT	FF	CR	∞	∂
020	⊂	⊃	∩	∪	∀	∃	⊗	↔
030	_	→	~	≠	≤	≥	≡	∨
040	SP	!	"	#	$	%	&	'
050	(	)	*	+	,	-	.	/
060	0	1	2	3	4	5	6	7
070	8	9	:	;	<	=	>	?
100	@	A	B	C	D	E	F	G
110	H	I	J	K	L	M	N	O
120	P	Q	R	S	T	U	V	W
130	X	Y	Z	[	\	]	↑	←
140	`	a	b	c	d	e	f	g
150	h	i	j	k	l	m	n	o
160	p	q	r	s	t	u	v	w
170	x	y	z	{	|	ALT	!	BS

NUL (0) is a null.
TAB (11) is a tab.
LF (12) is a linefeed.
VT (13) is a vertical tab.
FF (14) is a form feed.
CR (15) is a carriage return.
SP (40) is a space.
ALT (175) is an altmode.
BS (177) is a backspace.

APPENDIX 8

UUOS BY NUMBER

-----UUOs-----	
Opcode	Name
040	CALL
041	INIT
043	SPCWAR
047	CALLI
050	OPEN
051	TTYUUO
055	RENAME
056	IN
057	OUT
060	SETSTS
061	STATO
062	GETSTS
063	STATZ
064	INBUF
065	OUTBUF
066	INPUT
067	OUTPUT
070	CLOSE
071	RELEAS
072	MTAPE
073	UGETF
074	USETI
075	USETO
076	LOOKUP
077	ENTER
701	DPYCLR
702	PPIOT
703	UPGIOT
704	UINBF
705	UOUTBF
706	FBREAD
707	FBWRT
710	MAIL
711	PTYUUO
712	POINTS
713	UPGMVE
714	UPGMVM
715	PGIOT
716	CHNSTS
717	CLKINT
720	INTMSK
721	IMSKST
722	IMSKCL
723	INTUUO

----CALLIs----	
Number	Name
0	RESET
1	ODTIN
2	SETDDT
3	ODTOUT
4	DEVCHR
5	ODTGT
6	GETCHR
7	ODTRL
10	WAIT
11	CORE
12	EXIT
13	UTPCLR
14	DATE
15	LOGIN
16	APRENB
17	LOGOUT
20	SWITCH
21	REASSI
22	TIMER
23	MSTIME
24	GETPPN
25	TRPSET
26	TRPJEN
27	RUNTIM
30	PJOB
31	SLEEP
32	SETPOV
33	PEEK
34	GETLN
35	RUN
36	SETUWP
37	REMAP
40	GETSEG
41	GETTAB
42	SPY
43	SETNAM
44	TMPCOR
400000	SPWBUT
400001	CTLV
400002	SETNAM
400003	SPCWGO
400004	SWAP
400005	EIOTM
400006	LIOTM
400007	PNAME

----CALLIs----	
Number	Name
400010	UFBGET
400011	UFBGIV
400012	UFBCLR
400013	JBTSTS
400014	TTYIOS
400015	CORE2
400016	ATTSEG
400017	DETSEG
400020	SETPRO
400021	SEGNUM
400022	SEGSIZ
400023	LINKUP
400024	DISMIS
400025	INTENB
400026	INTDRM
400027	INTACM
400030	INTENS
400031	INTIIP
400032	INTIRQ
400033	INTGEN
400034	UWAIT
400035	DEBREA
400036	SETNM2
400037	SEGNAM
400040	IWAIT
400041	USKIP
400042	BUFLEN
400043	NAMEIN
400044	SLEVEL
400045	IENBW
400046	RUNMSK
400047	TTYMES
400050	JDBRD
400051	DEVUSE
400052	SETPR2
400053	GETPR2
400054	RLEVEL
400055	UFBPHY
400056	UFBSKP
400057	FBWAIT
400060	UFBERR
400061	WAKEME
400062	GETNAM
400063	SNEAKW
400064	SNEAKS
400065	GDPTIM

----CALLIs----	
Number	Name
400066	SETPRV
400067	DDCHAN
400070	VDSMAP
400071	DSKPPN
400072	DSKTIM
400073	SETCRD
400074	CALLIT
400075	XGPUUO
400076	LOCK
400077	UNLOCK
400100	DAYCNT
400101	ACCTIM
400102	UNPURE
400103	XPARMS
400104	DEVNUM
400105	ACTCHR
400106	UUOSIM
400107	PPSPY
400110	ADSMAP
400111	BEEP

---MAIL UUOs--	
Number	Name
0,	SEND
1,	WRCV
2,	SRCV
3,	SKPME
4,	SKPHIM
5,	SKPSEN

----INTUUOs---	
Number	Name
0,	INTJEN
1,	IMSTW
2,	IWKMSK
3,	INTDMP
4,	INTIPI
5,	IMSKCR

----PGIOTs----	
Number	Name
0,	PGSEL
1,	PGACT
2,	PGCLR
3,	DDUPG
4,	PGINFO

----TTYUUOs---	
Number	Name
0,	INCHRW
1,	OUTCHR
2,	INCHRS
3,	OUTSTR
4,	INCHWL
5,	INCHSL
6,	GETLIN
7,	SETLIN
10,	RESCAN
11,	CLRBFI
12,	CLRBFO
13,	INSKIP
14,	INWAIT
15,	SETACT
16,	TTREAD
17,	OUTFIV

----PTYUUOs---	
Number	Name
0,	PTYGET
1,	PTYREL
2,	PTIFRE
3,	PTOCNT
4,	PTRD1S
5,	PTRD1W
6,	PTWR1S
7,	PTWR1W
10,	PTRDS
11,	PTWRS7
12,	PTWRS9
13,	PTGETL
14,	PTSETL
15,	PTLOAD
16,	PTJOBX

----PPIOTs----	
Number	Name
0,	PPSEL
1,	PPACT
2,	DPYPOS
3,	DPYSIZ
4,	PPREL
5,	PPINFO
6,	LEYPOS
7,	PPHLD

INDEX