THE I-O PROCESSOR

TABLE OF CONTENTS

7 July 1966

FASTRAND IOTS

IOT	AC	IO	MISC	BIT12 40	BIT13 20	BIT14 10	BIT15 04	BIT16 02	BIT17 01
WNIF	ca	rtn da	-	ø	ø	-	3rd	3rd	2 rtn
WNIH	ca	rtn da	-	ø	1	-	3rd	3rd	2 rtn
RAI	ca	da	iopmax	ø	-	-	3rd	3rd	2 rtn
WAI	ca	da	ownwd rewr	ø	held, no checks	in apr	3rd	3rd	2 rtn
EAI	ca	da	ownwd rewr	ø	held, no checks	leave iig item	3rd	3rd	2 rtn
WNBF	ca	rtn da	-	1	ø	zero	3rd	3rd	2 rtn
WNBH	ca	rtn da	-	1	1	zero	3rd	3rd	2 rtn
RAB	ca	da	iopmax	1	-	-	3rd	3rd	2 rtn
WAB	ca	da	-	1	ø	zero	3rd	3rd	2 rtn
EAB	-	da	-	1	ø	-	3rd	3rd	2 rtn
WNIA	ca 4wrds	rtn da	-	-	-	-	-	-	2 rtn
WPID	ca 7wrds	rtn da	-	-	-	-	-	-	2 rtn
RPM	ca 4wrds	-	iopmax	bedspc	no read	da→io	no id	-	2 rtn
MPM	ca 6wrds	-	-	ø	-	-	-	-	2 rtn
EPM	ca 3wrds	-	-	1	-	-	-	-	2 rtn
IVNR+ØØ	-	n, rtn c(n)	-	-	ø	rewr→ac	-	-	2 rtn
IVNR+2Ø	ca	n, rtn c(n)	iopmax	block	1	ø	3rd	3rd	2 rtn
IVNW	ca 3wrds	c(n)	-	ø	-	-	-	-	2 rtn
IVNRW	ca 3wrds	c(n)	-	1	ø	-	-	-	2 rtn
IVNE	ca 3wrds	c(n)	-	1	1	-	-	-	2 rtn
SGI	ca comds	da	?	ø	held	non-adr	3rd	3rd	2 rtn
SGB	ca comds	da	?	1	held	non-adr	3rd	3rd	2 rtn
ITQ+ØØ	ca 3wrds	quant.	-	ø	ø	-	3rd	3rd	2 rtn
ITQ+2Ø	ca 3wrds	ca 3wrds	-	ø	1	-	3rd	3rd	2 rtn
ETQ	ca 3wrds	quant.	-	1	-	-	3rd	3rd	2 rtn
RQTRK	-	da	-	-	-	-	3rd	3rd	2 rtn

THE I-O PROCESSOR

INTRODUCTION

The I-O processor is that portion of the Executive System which handles the manipulation of data on the Fastrand drum and on magnetic tape. An IOT is a trapped instruction which is interpreted by the Executive System. Ordinarily, the major function of an IOT is designated by the middle 6 bits; the user may specify minor variations on the command with the low-order 6 bits. The movement of data between core and the Fastrand or between core and magnetic tape may be accomplished by I-O processor IOT's.

All I-O processor IOT's can be completed successfully or unsuccessfully. Two methods are provided for detecting errors. The first method, (bit 17 of the IOT =Ø), is described as a "trap mode". When an error occurs, the user's extended PC (as of the time the IOT was executed) is placed in lower-core register TRAPPC, and the low-order 17 bits of the PC are set to start the user at register IOPTSU (I-O Processor Trap Start-Up), out of extend-mode. The second method, (bit 17=1), designates two returns to the main sequence following the IOT (similar to the divide instruction): the error return is the register after the IOT, and the successful return is two registers after the IOT.

```
        IOT
        CALL TO ABNORMAL ROUTINE
        GOOD RETURN
```

If there is no error, the program reaches the "good return". In case of error, the PC is stored in TRAPPC, as described

above; the program reaches the "bad return," and the user's "abnormal routine" is called. When an error return is given, a numerical code for the type of error encountered is stored in the first error word in lower core (ERCODE).

FASTRAND HARDWARE

The Fastrand system consists of two rotating drums and a drum controller. These drums revolve approximately every 7Ø. milliseconds. Between the drums is a movable boom on which 64. read/write heads are mounted, 32. for each drum. There are 96. track positions to which the boom can move. The average boom movement time is approximately 60. milliseconds. In addition, there are 8 fixed heads which can be referenced at any time, regardless of the boom position. Under each head, for a given boom position, there are 64. sectors. A sector consists of a tag word and 50. words of data.

In the following pages the two drum cylinders will be referred to as the "drum". (For further information concerning the Fastrand and magnetic tape hardware, see the Data Channel Manual, TN-2ØØ1, 30 November 1964).

DRUM DIVISION

Conceptually, the drum is divided into thirds, each having 32. track positions. The allocation of various types of data to specific thirds is by convention, with the exception that patient records are stored on third Ø and this third is referenced implicitly by APR IOT's.

third	use
Ø	Active Patient Record
1	Library and Programmer's Storage
2	Research

When appropriate, the third to be referenced is designated by bits 15-16 of the IOT; they may be ØØ, Ø1, 1Ø (11 is an illegal combination and will give "illegal specification" error). All Fastrand IOT's must have these bits set appropriately, except for APR IOT's (which automatically reference third Ø) and those invariant number IOT's which only reference the fixed heads. A location on a third is specified by a 17-bit drum address.

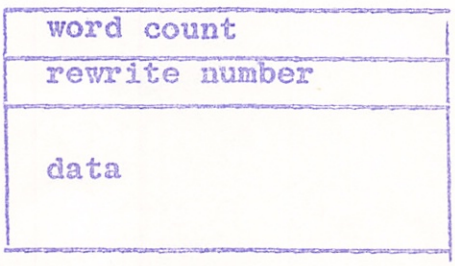
FREE VS HELD INFORMATION

Each third is divided into quarter tracks. (Each quarter track is composed of 2000_8 blocks). A quarter track can be in one of three conditions: available (i.e., empty and unowned), free or held. Common storage items and blocks, such as permanent file structures, libraries and permanent programmers' files are stored as free information. Free quarter tracks may be written on (or expunged from) by all users. A held quarter track is owned by one user; only he can write on or expunge from it, though anyone can read from it. This type of information is for temporary storage: that is, for segments, scratch areas, and internal file storage. When a user halts, his held quarter tracks are returned to available storage. If a user expunges all blocks on a held or free quarter track, the quarter track is returned to available storage.

The I-O processor automatically assigns quarter tracks to held or free use, depending on the type of "write" IOT which is executed. The I-O processor keeps track of the locations of available sectors within each quarter track. If a user writes "non-addressed" he leaves it to the Executive System to put the information into the next free sector or sectors; if he writes "addressed", he specifies the location of already existing information to be written over. If a user writes non-addressed held, the processor locates an available address, assigning a quarter track to held use if there are no more sectors available on a currently held quarter track.

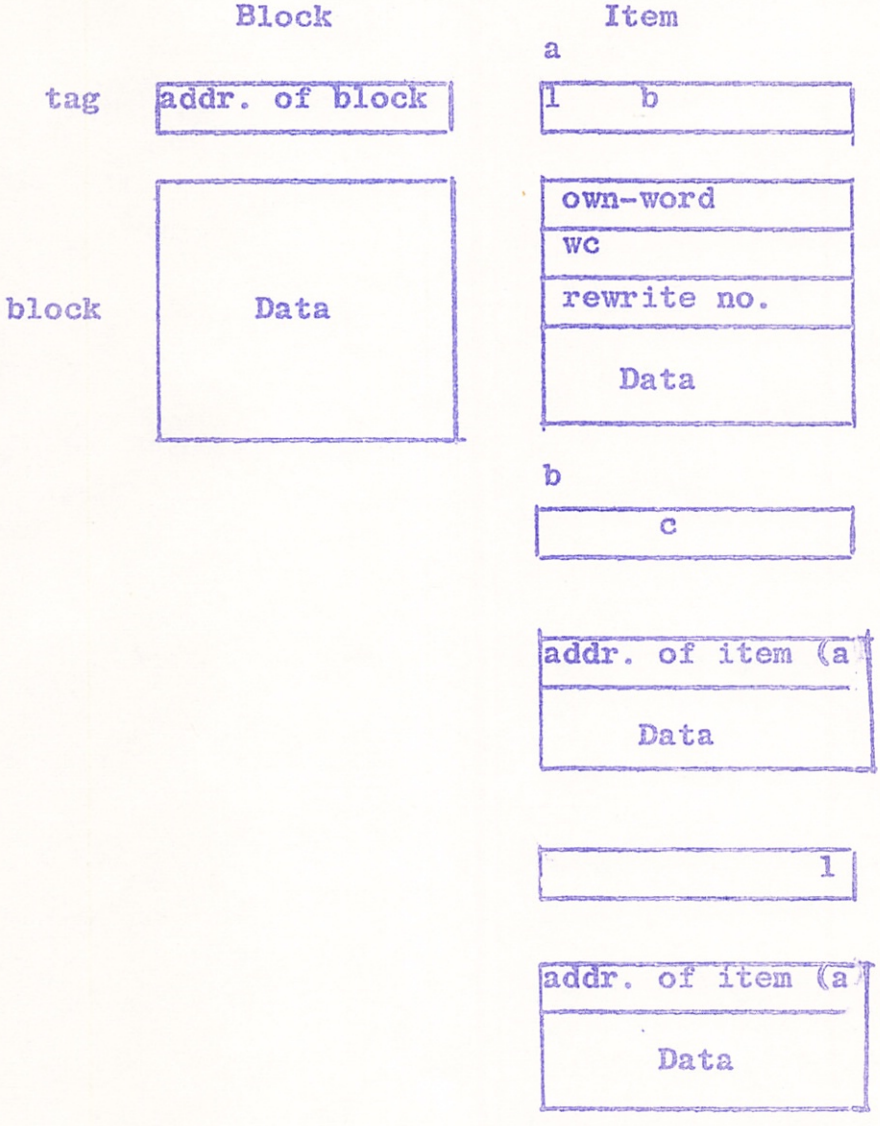
METHODS OF STORAGE

There are two formats for data storage on the drum: the block and the item. A user can write out a fixed length of information, called a block, which consists of precisely 5Ø words. An item is a more general format. Because an item can be of any length, it offers a more flexible means for storing information on the drum. Its format in core is as follows:



The word count equals the number of words in the item. This is the number of data words plus two.

On the drum these formats look as follows:



REWRITE NUMBERS AND OWN WORDS

To prevent certain time-sharing problems and to protect the drum from accidental rewriting or expunging, two mechanisms have

been included: the rewrite number and the own word. The rewrite number follows the word count in each item (as is illustrated in the preceeding diagrams). This number is incremented by one, each time an item is rewritten. When writing or expunging an addressed item, the I-O processor compares the rewrite number of the item in core with the corresponding number on the drum. If they are the same (i.e., no one else has written this item since the current user read it) the rewrite number is incremented by one and the item is rewritten. If the rewrite numbers are not the same (i.e., some one else has written this item since the current user read it) a "rewrite error" is given.

The own-word is another protective device. When an item is written non-addressed, the contents of the lower-core register OWNWD are permanently associated with that item on the drum. Whenever a user attempts to rewrite or expunge any item, his lower-core OWNWD must match the item's own-word. This method of identification helps prevent a program from mistakenly destroying data on the drum. If bit 13 is set in an IOT to write addressed on or to expunge from a held quarter track, the rewrite number and own-word comparisons are not made.

IOP MAXIMUM

Another register in lower core (IOPMAX) may be used to prevent items, blocks or patient parameters from overflowing their core buffers on reads. This register is an inclusive MAXIMUM for all reads into core (except for scatter-gather and magnetic

tape reads). C(IOPMAX)=Ø is the same as C(IOPMAX)=7777. The high-order 6 bits of IOPMAX are ignored. Any attempt to read information into core above the register specified by IOPMAX will give an "IOPMAX" error. There is also a safeguard against information being read into core below register 36_8. If this is attempted a "below bound" error will result.

FASTRAND IOT'S FOR ITEMS AND BLOCKS

WNIF = IOT 63ØØ (write non-addressed item, free)
WNIH = IOT 632Ø (write non-addressed item, held)
C(AC) = Core address of item

An item is written out starting at the location specified in the AC. The drum address is placed in the IO.

Errors

a) Total system error or data error
b) Illegal specification (word count ≤2; illegal third)
c) No more quarter tracks.

WAI = IOT 64ØØ (write addressed item)
C(AC) = Core address of item
C(IO) = Drum address of item

The item in core replaces the one on the drum if that position on the drum indeed contains an item whose OWNWD and rewrite number agree with those in core. The rewrite number in core and on the drum are incremented by 1. If bit 14. is set, the item which is being rewritten is considered an APR item, and therefore the two linking pointers are compared with those on the drum to be sure they are the same. If bit 13. is set and the item is on a held quarter track, the rewrite number and own word are not checked, thus saving a revolution of the drum.

Errors

a) Total system error or data error

b) Illegal specification (illegal drum address; illegal third; word count ≤2

c) Rewrite error (OWNWD or rewrite number does not agree)

d) Illegal item

RAI = IOT 61ØØ (read addressed item)

C(AC) = Buffer area in core

C(IO) = Drum address of item

The item is read into core beginning at the location specified.

Errors

a) Total system error or data error

b) Illegal item or illegal drum address

c) IOP maximum exceeded

d) Below bound

EAI = IOT 62ØØ (expunge addressed item)

C(AC) = Location in core of item

C(IO) = Drum address of item

The own-word and rewrite number of the item are compared with the appropriate core locations. If they agree the sectors on which the item was written are marked as "available" and their contents are zeroed. If bit 14 of the IOT is set, the first sector of the item is not returned to available storage. Instead, it is re-written in a form that can be referenced neither as a block nor an item. (Its tag is left containing the address of the sector, with the sign bit set).

WNBF = IOT 634Ø (write non-addressed block, free)

WNBH = IOT 636Ø (write non-addressed block, hold)

C(AC) = Location in core

A 5Ø. word block is written on a free or a held quartertrack, as specified. The address is returned in the IO. If bit 14 of the IOT is set, the AC is ignored, an all-zero block is written, and the address in returned in the IO:

Errors

a) Total system error or data error
b) No more quarter tracks

WAB = IOT 644Ø (write addressed block)

C(AC) = Core location of block
C(10) = Drum address

The block is written at the sector specified by the IO, provided the following is true: The sector may not be marked as "available". If the sector is on a held quarter track, the user must own, and if it is on a free quarter track, it must contain a block (rather than part of an item). If bit 14 of the IOT is set, a zero block is written.

Errors

a) Total system error or data error
b) Illegal block
c) Not your quarter track

RAB = IOT 614Ø (read addressed block)

C(AC) = Buffer area in core
C(10) = Drum address of block

The block is read into core beginning at the location specified, and the contents of the tag word are placed in the IO. (In the normal case, the tag contains the address of the block and the IO is effectively unchanged).

Errors

a) Total system error or data error
b) IOP maximum exceeded
c) Illegal block or illegal drum address
d) Below bound

EAB = IOT 624Ø (expunge addressed block)

C(10) = Drum address of block
(block need not be in core)

The sector specified by the IO is marked "available" and its contents are zeroed, provided the following is true: it must contain a block (rather than part of an item); and if it is on a held quarter track, the quarter track must be owned by the user.

Errors

a) Total system error or data error
b) Illegal block
c) Not your quarter track

RQTRK = IOT 6ØØØ (release quarter track)

C(IO) = Any drum address in quarter track

If this quarter track is held by the user, the quarter track is returned to free storage and its contents are zeroed.

Errors

a) Total system error or data error
b) Not your quarter track

ACTIVE PATIENT RECORD

The APR is the file in which patient records are stored. It is specifically designed for this purpose, and its structure is built into the Executive System. APR IOT's automatically refer to the third Ø of the drum.

The file is organized and accessed by bed space. A bed space code is an 18 bit word, in which the left 9. bits are care unit number and the right 9. bits are bed within care unit. For every legal bed space code, the I-O processor maintains a three-word block on the fixed heads:

1. Pointer to parameters item (-Ø if none)
2. Unit number (first word)
3. Unit number (second word)

The first of these points to an item (on third Ø) which contains the patient's parameters.

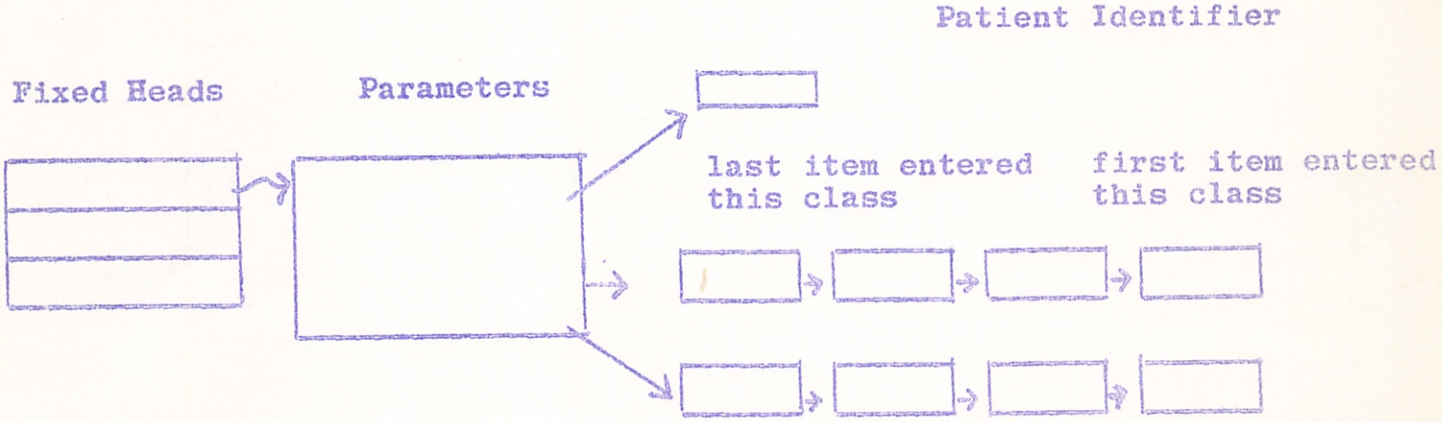
44 (word count)
rewrite number
addr of patient identifier
last item this patient
addr of last item class 1
addr of last patient class 32

The first parameter is the drum address of the (special) patient identifier item. The parameters item is created when the patient identifier is written. Until the parameters item is written, any attempt to write items for a bed space will give an error. The other parameters are pointers to the most recent items of chains: APR items for each patient are chained together in a list structure in reverse chronological order. Each item in the list points at the one written just previously, and the end of a chain is denoted by a -Ø pointer. There is one main chain, threading all the items for each patient. In addition, there are 32. sub-chains, and each item is a member of one of these sub-chains (as well as being a member of the main chain). The particular sub-chain to which an item belongs is determined by its class, a number that is stored (together with the chaining pointers) in the overhead of the item. The APR item format contains

these three words, in addition to the normal item overhead.

word count
rewrite number
previous item this patient
previous item this class
class in bits 2-7
DATA

Thus we have structure as follows:



WPID = IOT 674Ø (write patient identifier item)
C(AC) = Pointer to 7 word block

Core adr of P.I.D. item
old bedspace
OLD UNIT NUMBER
waste
NEW UNIT NUMBER

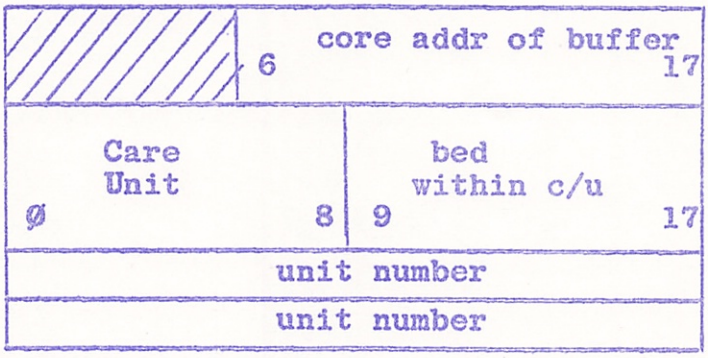
The old unit number on the drum is compared with that in core. The new unit number then replaces the old one on the drum. If this bedspace has no parameters, the patient identifier item is written (in standard item format), and the parameters are created; the drum address is indicated in the IO. If the parameter item already exists, the patient identifier is rewritten.

Errors

a) Total system error or data error
b) Illegal specification
c) No more quarter tracks
d) Bedspace does not exist
e) Unit number does not match

RPM = IOT 55ØØ (Read Parameters)

C(AC) $_{6-17}$ = Pointer to 4 word block in core



The user can read parameters by specifying either bed space or unit number. The I-O processor will furnish the other identifier (by placing it in the 4-word block), and the parameters will be read into the buffer specified.

Options in IOT

bit 12 — 1 means read by bed space
Ø means read by unit number

bit 13 — 1 means do not read parameters item
Ø means read parameters item

bit 14 — 1 means place address of parameters item in IO
Ø means leave IO transparent

bit 15 — 1 means do not place other identifier in core
Ø place other identifier in core

Errors

a) Total system error or data error
b) No such bed space
c) No patient in this bed or no such unit number
d) IOPMAX exceeded

EPM = IOT 574Ø (expunge parameters)

C(AC) = pointer to three word block

Ø	care unit 8	9	bed within c/u 17
unit number			
unit number			

If the unit number on the fixed heads agrees with that in core, then the parameters item is expunged and the three words on the fixed heads are replaced by -Ø.

Errors

a) Total system error or data error
b) No parameter item exists or the unit numbers disagree
c) Illegal bedspace

MPM = IOT 57ØØ (move parameters)
C(AC) = Pointer to 6 word block

Ø	care unit					8	9	bed within c/u									17
unit number																	1
unit number																	1
Ø	care unit					8	9	bed within c/u									17
unit number																	2
unit number																	2

If the unit numbers in core agree with those stored on the drum, the parameters and unit numbers of these two patients (as specified by bed space codes) are switched.

Errors

a) Total system or data error
b) Unit number disagreement
c) No such bed space

WNIA = IOT 67ØØ (write non-addressed item on APR)
C(AC) = core address of 4 word block

						6	core address of item										17
Ø	care unit					8	9	bed within c/u									17
unit number																	
unit number																	

A check is made to assure that the parameter item exists and that the unit number in core matches that on the fixed heads. Then the third and fourth words of the item in core are replaced, respectively, by the parameters pointing to the "last item, this patient" and the "last item, this patient, this class." Next the item is written, as if by WNIF on third Ø, and its drum address is put in the user's IO. Finally the parameters are up-dated: the drum address of the item replaces both the pointer to the last item, this patient, and the pointer to the last item, this class.

INVARIANT NUMBERS

An invariant number is a number by which a user may address a one-word storage area on the Fastrand. These storage areas (with their associated own-words and rewrite numbers) are kept on the fixed heads (and are accessed by the I-O processor using address arithmetic.) Currently there are 21ØØ invariant numbers, numbered Ø-2Ø77. They are, in effect, one-word items, and there are IOT's to write, read, rewrite, and expunge their contents. In addition, since it is anticipated that these storage areas will be used principally for storage of the drum addresses of frequently accessed items, it is possible to use the invariant number as the "indirect address" of an item or block to be read. Note that there is no "indirect" feature for writing, rewriting, or expunging items or blocks via invariant numbers.

IVNR = IOT 65ØØ (invariant number read)
C(AC) = Core address of buffer if bit 13 of IOT set
C(IO) = Invariant number

The contents of the storage area addressed by the invariant number is placed in the IO.

Options

bit 12. 1 means (if bit 13. set) contents of storage area addresses block to be read, rather than an item.

bit 13. 1 means place contents of storage area in IO and handle exactly as read item or block.

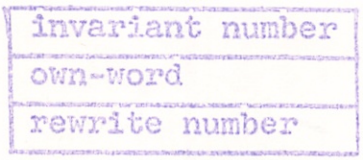
bit 14. 1 means place rewrite number for storage area in AC. Note that, since this operation happens first, bit 13 (the "indirect" feature) cannot be used in the same IOT, since the buffer address in the AC will be lost before the item or block is read.

Errors

a) Total system error or data error
b) Illegal specification (invariant number $\geq$ 2100)
c) All errors for read item or block (in case bit 13. set)

IVNW = IOT 6600 (invariant number write)
IVNRW = IOT 6640 (invariant number rewrite)
IVNE = IOT 6660 (invariant number expunge)

C(AC) = Pointer to 3-word block in core
C(IO) = Value to be placed in storage area (on write or rewrite)



Write stores the contents of the IO in the storage area and copies the own-word and rewrite number from the 3-word block in core. Write is permitted only if the current contents of own-word, rewrite number, and storage area on the drum are all -Ø, indicating availability. If the rewrite number is written as Ø, rewrite number comparison will be omitted on later rewrites and on expunge.

Rewrite stores the contents of the IO in the storage area, provided there is no disagreement of own-word or rewrite number. (If the rewrite number is non-zero, it is incremented in such a way that it can not end up as zero.) A zero rewrite number is not incremented. If bit 14.is set on IVNRW, the 3-word block is read into the 3-word buffer pointed at by the AC, i.e., IVNRW+10 reads the 3-word block instead of rewriting the block.

Expunge changes the own-word, rewrite number, and contents of the storage area to -Ø, provided there is no disagreement of rewrite number or own-word.

SCATTER-GATHER

Scatter-Gather makes it possible for a user to dynamically alter the core address involved in a data transfer or comparison while processing one continuous item or block on the drum. The functions that may be used are logically equivalent to the data channel commands. They are: read, write, skip reading, skip writing, continue on compare, and continue on non-compare. Several functions may be combined in any IOT.

Read and write are self-evident (note that Scatter-Gather does not reference IOPMAX). Skip reading is in effect a read into a fixed non-existent location (an information sink). The command

to skip in reading will result in the specified amount of material being held in the data channel without passing into core. In skip writing, unwanted information is zeroed. Continue on compare will cause words on the drum to be compared with a corresponding number of words in core, with a non-comparison being defined as an error. If the specified words match exactly, the I-O processor will continue executing commands; otherwise data transfers will cease. Continue on non-compare is identical to continue on compare except that the condition for continuation is that there be at least one mismatch.

With scatter-gather one may operate on items or blocks. Functions may be used singly or in any combination in any block or sector of an item. This feature is unlike the hardware in that the hardware can handle only compatible groups of commands within a physical block. The compatible groups are: 1) write and skip write, 2) read, skip read, continue on compare, and 3) read, skip read, continue on non-compare. Note that the two kinds of compare are not compatible with each other.

When the I-O processor senses an incompatible sequence of commands within a physical block it executes them interpretively. This is costly both in central processor time and in terms of total time that the I-O processor is busy, and should be avoided unless absolutely necessary.

There are a few restrictions in the use of scatter-gather:

When an item is to be changed by writing or zeroing, the rewrite number (word 2 of the item) must be written.

When the length of an item is to be changed, the word count (word 1 of the item) must be written.

Neither the word count nor the rewrite number may be zeroed.

A scatter-gather command table consists of two word blocks:

Ø - 5	6 - 17
Command	Core Address
word count	

The command is in bits: Ø-5

ØØ means read
Ø1 means skip over words on drum
Ø2 means continue on compare, i.e., when all words of this segment are equal between core and drum
Ø3 means continue on non-compare, i.e., when there is at least one disagreement between core and drum
Ø4 means write
Ø5 means write zero
77 is the terminator for command table

The core address is the first location in the core segment to be read into or written-from, and this field is ignored on skip, zero and terminator.

The word count specifies the number of words to be dealt with by the command. Word count of Ø is legal* The word following the scatter-gather table terminator is ignored.

* except in the first command pair of a write-non-advanced sequence when the first command must be a write!

FASTRAND IOT's for SCATTER-GATHER

SGI = IOT 53ØØ (Scatter-Gather item)
SGB = IOT 534Ø (Scatter-Gather block)
C(AC) = Pointer to scatter-gather table
C(IO) = Drum address if any

The user may write non-addressed by setting bit 14 of the IOT. Then bit 13 is set to specify held quarter tracks, or clear, to specify free. The drum address is returned in the IO. Note: when writing non-addressed, write and skip write are the only legal commands. Also, if write or skip write are included anywhere in the command sequence, the sum of all word counts must equal the first word of the item. (for blocks, the sum of the word counts must be 50.)

Errors:

a) System failure or data error
b) Illegal item
c) Attempt to read off end of core (analogous to IOPMAX)
d) Rewrite number or CWNWD do not agree
e) Comparison, or non-comparison, error (depending on the user's definition).
f) Illegal specification (drum address $>2^{17}-1$; word count ≤ 1 or $>2^{17}-1$; attempt to read below bound; trying to zero a word count -- i.e., specifying skip writing as first command; incompatible sequence -- for example, write and skip write are the only compatible IOT with writing non-addressed; indeterminate command).

INDEXING

The index IOT's allow the user to store, in a file on the Fastrand, a relationship, or "mapping", between a 36.-bit argument, or "index code", and an arbitrary 18.-bit number (ordinarily a drum address). The organization of the file allows another user, at a later time, to use the same 36. bit index code as a "handle" to retrieve the stored 18.-bit number from the file. Suppose, for example, whenever a patient identifier item was filed, the drum address of the item was stored in the index, using the patient's unit number as the index code. Then, whenever someone typed in a unit number, the user program could find the associated patient identifier by obtaining its address from the index.

Currently there are two entirely separate index files, identical in structure. One is on third Ø of the Fastrand, and the other is on third 2. The user can address either of these files by indicating "third" in the usual manner, in bits 15.-16. of his IOT's. The remainder of this discussion will describe one of these files, with the understanding that everything said applies to both.

The index file contains 48. identical indexes, which may be assigned various functions by convention. For example, index Ø may map unit numbers into patient identifier addresses, while index 1 maps soundex-encoded drug names into drug item addresses, etc. These 48. indexes, though stored in the same file, are essentially independent of each other and are treated as such here. For time efficiency, the indexes are ordered by index code; and for space efficiency, only those "slots" which actually contain indexed 18.-bit numbers take up room on the drum.

The filing scheme is explained below; the user need understand only enough of the details to be able to use the index "file" and "delete" IOT's and the "index retrieve" subroutine package. The 36.-bit index code is interpreted as three 12.-bit sections. For each section, the combined knowledge of which section it is, (1,2, or 3), the numerical value of the 12,-bit section, and which index is involved, (Ø-47.), provides enough information to locate a list containing a particular subset of all the 18.-bit numbers that have been indexed. This subset consists of all those 18.-bit numbers that have been associated in the index with any 36.-bit index codes whose corresponding 12.-bit sections contain exactly the 12.-bit number under consideration. Thus, given a 36.-bit index code and an index number (Ø-47.), one can find three lists: one list for each 12.-bit third of the index code. The process of index filing places the 18.-bit number in all three lists.

When index retrieving for a 36.-bit index code, one examines the three lists to which it leads. Any 18.-bit number which is simultaneously present in all three lists must have been filed previously under the index code in question. (This conclusion is based on the assumption that it is not permissable to index the same 18.-bit value twice, using different index codes.) Any 18.-bit number found in the first list must have been indexed under some 36.-bit code whose first 12.-bit section exactly matches the first section of the code for which one is retrieving. Further, any number found in the second list must have been indexed under some code whose second section matches; and similarly for the third list. Thus, any number found in all three lists must have been indexed under a code whose three sections, taken one at a time,

exactly match the corresponding sections of the code for which retrieval is being done -- and this, consequently, must be the very same code.

The first 12.-bit section of an index code, combined with the index number, will lead to one of 48. x $2^{12.}$ possible lists. The second and third 12.-bit sections, however, each lead to one of only $2^{12.}$ possible lists. All 18.-bit numbers under a given 12.-bit value for the second section are placed in the same list, regardless of which index they are filed in; and similarly for the third section. The reason this does not lead to confusion between indexes is that, for any given value of the first 12.-bit section, of an index code, there are 48. separate lists, one for each index. (If the same 18.-bit number is indexed twice, even in different indexes and with different index codes, spurious results may be retrieved.)

The foregoing has assumed index codes that require the full 36.-bit capacity of the indexing system. For some applications it may be more efficient to index 18.-bit numbers against index codes of 24. or 12. bits. This may be accomplished by storing the 18.-bit number in the first one or two lists only, and not in all three. Similarly, it may be desirable to retrieve numbers found in one list or common to two, either because they were filed against shortened index codes or in order to obtain numbers filed under index codes that disagreed only in the last 12. or 24. bits. The index IOT's and "index retrieve" subroutine provide these facilities.

Some of the details are included below.

Track Ø, permanently held by Exec, contains $2^{12.}$ blocks. Each of the 50. x $2^{12.}$ words on this track is (potentially) a pointer to a list of the kind described. (If the list is empty, this word contains -Ø.). Given the numerical value of a 12. -bit section of a 36. -bit index code, together with a knowledge of whether it is the first, second, or third section, and a knowledge of which index is involved, one can locate a single pointer word on track Ø via the following address-arithmetic process: treat the 12. -bit section of the argument as a drum address within track Ø, and read the block at that location. Next, if the 12. -bit section is the first third of the argument, choose the word (Ø-47.) within the block corresponding to the index number; alternatively, if it is the second section, choose word 48. of the block; or, if it is the third section, choose word 49. The word chosen contains a pointer to the list to be located.

In order to obtain the list itself, given this pointer to it, one must understand the manner in which these lists are stored. They are kept on quarter-tracks that are permanently held by Exec, in a conventional list structure (to allow them arbitrary length). The cells of this list structure are stored 4 to a block and are 12. words long. The last word of a cell points at the next cell of its list (or contains -Ø), and each list ends with an 18. -bit data word of -Ø. The data words in each list are kept in numerical order. The I-O processor keeps track of available cells and assigns

and list-structures a new quarter-track whenever necessary. The quarter-block pointers can be converted to drum addresses by a process involving both address-arithmetic and table-lookup in Exec core. The table-lookup is accomplished by user programs (including "index retrieve") via the IOT "PEEK".

The index IOT's are executed with the information to be indexed or expunged in the I-O. This may be any 18.-bit number other than -Ø. The AC points to a three-word block in core:

Ø 5	6 11	12 17
Index code length (1-3)	Index number of "B#" (0-47.)	
First third of code		Middle ...
...Third of code	Last third of code	

The code length or "pass count" indicates how many 12.-bit sections of the index code are relevant. Codes shorter than 36. bits are left-adjusted.

ITQ = IOT 54ØØ (index this quantity)
ETQ = IOT 54ØØ (expunge this quantity)
C(AC) = Pointer to 3-word block
C(I-O) = Quantity to be indexed or expunged

ITQ places the quantity in one or more lists, as described above.
ETQ removes it from these lists.
In addition, ITQ provides an option useful to "index retrieve". This IOT finds the three quarter-block pointers on track Ø. Then, if bit 13. is set, it places them in a 3-word buffer designated by the I-O and returns to the user.

ITQ+2ϕ = IOT 542ϕ (finds 3 quarter-block pointers)
C(AC) = Pointer to 3-word block, as in ITQ
C(IO) = Pointer to 3-word buffer

Errors:

a) Total failure or data error
b) Illegal specifications
c) No more quarter-tracks

TIMING

To give the user an idea of how long write items and read items take, the following list will be helpful.

IOT

write item 49. words	85 millisecs (1ØØ millisec = 1 second)
read item 49. words	1Ø5millisecs
write block	85 millisecs
read block	1Ø5 millisecs

For every extra 49. words in an item add 2 millisec. To figure how long drum activity will take, the user should also assume there are 4-5 other users ahead of him on the drum waiting list, so, in effect his IOT will take 5 times as long.

FASTRAND ERRORS

The following is a list of Fastrand errors. Data errors are not handled separately, since this I-O processor will repeat data transfers up to 4 times in case of data errors. The I-O processor error codes are bit-oriented and start at bit ZERO. The second error word contains information useful for handling this particular occurrance of the error. The interpretation of the contents of the second error word will be specified with each IOT. Contents of first error word:

Bit Ø=Ø — Fastrand drum error. This bit may be used by a general error routine to distinguish Fastrand errors from tape errors.

Bit 1=1 (2ØØØØØ) — Some condition which cannot be handled by the Executive System is present. For example, persistent trellis errors and data errors are included here.

Bit 2=1 (1ØØØØØ) — The user requested data to be read into a region above the bound set by the contents of the register 'IOPMAX' in lower user core.

Bit 3=1 (4ØØØØ) — Some condition in the calling sequence of the IOT was in error, i.e., an illegal specification. Also included here are attempts to read into protected memory (below register 36). This bit indicates errors in coding which will not come up in debugged programs.

Bit 4=1 (2ØØØØ) — The user attempted to write (held) on a quarter-track not owned by him.

Bit 5=1 (1ØØØØ) — There are no more available quarter-tracks on this third on the drum.

Bit 6=1 (1ØØØØ) Having defined non-comparison as an error, the user requested that a certain section of words in core be compared with a corresponding number of words on the drum, and they were not the same. (Bit is also used, when successful comparison is defined as an error, to indicate a comparison was successful, thus causing an error return.

Bit 7=1 (2ØØØ) Rewrite number or OWNWD disagree on rewrite on Fastrand.

Bit 8=1 (1ØØØ) User attempted to reference an illegal item, block, or drum address.

Bit 9=1 (4ØØ) User attempted to access non-existent bedspace.

Bit 10=1 (2ØØ) No such unit number exists. Also used when user attempts to access a legal but empty bedspace.

MAGNETIC TAPE IOTS

FUNCTION	IOT	AC	IO	BIT12 40	BIT13 20	BIT14 10	BIT15 04	BIT16 02	BIT17 01
get tape	MCNT+00	-	reel no, rtn unit	0	-	0	-	-	2 rtn
get tape and space	MCNT+10	blks/ files	reel no, rtn unit	0	files	1	oddpar	lowden	2 rtn
release tape at load pt	MCNT+40	-	unit	1	-	-	-	-	2 rtn
rewind to load pt	MREW+00	-	unit	0	-	-	-	-	2 rtn
release at change tape pt	MREW+40	-	unit	1	-	-	-	-	2 rtn
space	MSPC	blks/ files	unit	back	files	-	oddpar	lowden	2 rtn
write eof	MWEOF	-	unit	-	-	-	-	-	2 rtn
erase	MERS	inch/5	unit	-	-	-	-	-	2 rtn
read	MRPR length	ca	unit	fixed length	0	scatter gather	oddpar	lowden	2 rtn
write	MWPR length	ca	unit	fixed length	1	scatter gather	oddpar	lowden	2 rtn

MAGNETIC TAPE SYSTEM HARDWARE

The magnetic tape hardware consists of two UNIVAC III-C tape drives connected to a BBN modified magnetic tape controller. Information can be read or written on a magnetic tape at a low density of 2ØØ bits per inch or a high density of 556 bits per inch. The system will perform such typical tape manipulations as:

1. Read records
2. Write records
3. Erase tape
4. Space blocks forward
5. Space blocks backwards
6. Write end of files
7. Rewind to load point
8. Rewind with interlock

CONVENTION ON LOW ORDER BITS OF IOT

Bit 17=1 Give error return to register following the IOT
Bit 17=Ø Trap on error
Bit 16=1 Low density
Bit 16=Ø High density
Bit 15=1 Odd parity (binary)
Bit 15=Ø Even parity (BCD)

Conventions on bit 17 apply to all IOT's; the conventions on density and parity only apply to reading, writing, and spacing IOT's. Density indicates the quantity of information which can be stored on a magnetic tape; parity provides an error check. Once density and parity have been designated for a particular tape, (i.e., when first written) that tape must be read and spaced at the specified density and parity.

TAPE TIMING CONSIDERATIONS

The tape system is radically more sophisticated than Exec II and requires some explanation about what can be done. Non-data transferring IOT's do not hold the data channel in Exec III. These IOT's are handled by the tape controller independent of the Data Channel I-O processor structure. Non-data transfers are:

1. Spacing tape
2. Rewinding tape
3. Erasing tape
4. Writing end of files.

Therefore, one user can be using the Fastrand at the same time as another is spacing tape.

MAGNETIC TAPE HANDLING

Magnetic tape reels are stored (in the computer room) by reel number. The user program must specify the reel number desired when it first requests the tape. If the user program wishes to write on a tape, it must request the tape with the reel number complemented.

In order to gain access to a tape drive, the user executes a specific IOT which assigns a tape drive to a program and signals the computer room operator to mount the tape. If the requested reel is already in use a "NOT YOURS" error return is given. If no tape drive is available a "FULL" error return is given. If a drive is available, the user is hung until the operator mounts the tape. When the tape is at load point, the user is unhung. A complemented number in the IO means mount the tape with the write ring in. After a successful IOT, the C(IO) = tape drive unit number.

Two other basic IOT's are available which contain the command to get a tape and after it is mounted to space a certain number of blocks or files. This enables the user to locate himself, or his program, at a certain position on the tape. During the time a tape unit is held, no other user can reference it.

IOT's FOR GETTING AND SPACING A TAPE

MCNT=IOT 7ØØØ

C(AC) = Ignored

C(IO) = Tape reel number for reel wanted.

Errors

a) Full: no more tape units

b) Not Yours: the reel requested is in use

MCNT+1Ø (Get a tape and space blocks)

MCNT+3Ø (Get a tape and space files)

C(AC) = Number of files / blocks to space

C(IO) = Tape unit

BLOCKS AND FILES

A block is the smallest amount of data which can be put on the tape. A file is composed of a number of blocks; its organization and length is decided by the user. An end-of-file mark can be used to indicate a logical division of data, i.e., it separates files; it can also be used to mark the end of information on the tape. The end-of-file mark is used as a means of locating data or free area on a tape. By spacing blocks and files the user can manipulate his position on a tape. He can space himself to an unused area if he wants to write on the tape; he can locate a certain block of data if he wants to read it. Rewind is another means

of spacing, or changing location on a tape. The program is inactive until it arrives at load point when it is automatically unhung.

MSPC=IOT 7200 (space n blocks forward)

C(AC) = Number of blocks to space

C(IO) = Tape unit

The tape unit spaces the number of blocks (down the tape) specified by the AC.

Errors

a) Total system error

b) End of file occurred while spacing tape. The second error word contains number of blocks left to space.

c) Illegal specification if C(AC)< 0

d) End of tape occurred

e) Not your tape unit

MSPC+40= (space n blocks backward)

C(AC) = Number of blocks to space

C(IO) = Tape unit

The tape unit spaces toward load point the number of blocks desired. (Note that the hardware cannot see end of files while spacing backward and end of files are counted as blocks).

Errors

a) Total system error

b) Illegal specification if C(AC) <0

c) Load point error if load point reached before count of blocks is finished. The unit is then positioned at load point.

d) Not your tape unit

MSPC+2Ø (Space n files forward)

C(AC) = Number of files to space

C(IO) = Tape unit no.

The tape unit spaces forward the number of end of files indicated by AC. The unit is positioned immediately after the last end of file requested.

Errors

a) Total system error

b) End-of-file error: two end of files occurred in a row. The unit is positioned after the last of the two end of files.

c) Illegal specification: C(AC) <Ø

d) End of tape encountered

e) Not your tape unit

Note: MSPC+6Ø is legal and will position you at load point with load point error.

MWEOF=IOT 73ØØ (Write end of file)

C(AC) = Ignored

C(IO) = Tape unit

One end-of-file mark is written on the tape.

Errors

a) Total system error or data error

b) Write ring is out

c) End of tape encountered

d) Not your unit

MREW=IOT 71ØØ (Rewind)

C(AC) = Ignored

C(IO) = Tape unit number

Wherever the tape is in position, the tape is rewound to load point. When the tape is at load point, return is given to the user.

Errors

a) Total system error
b) Not your tape drive

RELEASING THE TAPE

When the user is finished with his tape he may release it with one of three commands:

1. Release the tape at load point - MCNT+4Ø
2. Release the tape at change tape point - MREW+4Ø
3. Halt

Each of these commands releases the tape in a different manner. Normally, MCNT+4Ø is used by a program which frequently wants this tape for a short amount of time. This IOT rewinds the tape to load point and releases the unit. If this user is the next person to want the tape unit, his tape is still mounted and is immediately assigned to him.

If a user is completely finished with a tape, he rewinds the tape with interlock, i.e., he releases the tape at change tape point. At this time the operator will demount the tape and store it away. The halt program releases magnetic tape units by rewind with interlock IOT's.

MCNT+4Ø (Release tape)

C(AC) = Ignored
C(IO) = Tape unit no.

The tape unit is no longer assigned to this user and the tape is rewound to load point.

Errors

a) Total system error (unit deassigned but error in rewind)
b) Not yours in first place

MREW=4Ø (Rewind with interlock)

C(AC) = Ignored

C(IO) = Tape unit number

The tape on the specified unit is released. Control is returned to the user immediately. The tape is rewound to change-tape position.

MRPR=IOT 74ØØ (Read physical record)

C(AC) = Core address to start at (bits 6-17)

C(IO) = Tape unit number

C(word after IOT)=Length of physical record (bits 6-17)

The record is read into core starting at location in core specified by AC. If the length of the record is greater than that specified in the calling sequence, only the number of words specified in the calling sequence will be read. If the calling sequence specifies a count greater than the actual length of the record, only the words in that record will be read. The length (number of words read) is left in ERCODE + 1.

Errors

a) Total system crash or data error
b) Not your tape drive
c) Illegal specification: C(AC)<36
i.e., attempted to read below bound
d) End of tape encountered
e) End of file encountered

MRPRF=IOT 744Ø (Read physical record fixed)

C(AC) = Core address to start read at (bits 6-17)

C(IO) = Tape unit

Same comments as above except the length of record is assumed to be 2Ø8, if binary, or 28. if BCD.

Same error comments as RPR

MWPR=IOT (write physical record)
C(AC) = Address to start writing from (bits 6-17.)
C(IO) = Tape unit
C(word after IOT) = length of record (bits 6-17.)

Errors
a) Total system error or data error
b) Not your tape drive
c) End of tape encountered
d) Write ring out

MWPRF=IOT 746Ø (write physical record fixed)
C(AC) = Address to start writing from (bits 6-17.)
C(IO) = Tape unit
Comment as in WPR except length of record is assumed to be 2Ø8. if binary, 28. if BCD.
Errors as in MWPR

MERS=IOT 57ØØ (erase tape)
C(AC) = Erase (No. of inches of tape) divided by 5
C(IO) = Tape unit
Erase the appropriate number of inches of tape.

Errors
a) Total system failure
b) Write ring out
c) Illegal specification: C(AC) ≤ Ø

MAGNETIC TAPE ERRORS

Bit Ø=1 Magnetic tape error. This bit is for a general error routine to specify what type of activity was involved.

Bit 1=1 Some condition which cannot be handled by the Exec-
6ØØØØØ utive System is present. For instance, persistent trellis errors. Data errors are included here.

Bit 2=1

Bit 3=1 Some condition in the calling sequence of the IOT was
44ØØØØ meaningless. For example, the user specified space -1 block. Included here are attempts to read into protected memory (below register 36). This bit means there are errors in coding which will not come up in debugging programs.

Bit 4=1 The tape drive you attempted to use is not held by you.
42ØØØØ You have not received permission to use this drive by executing a 'MCNT' IOT.

Bit 5=1 There are no available tape drives.
41ØØØØ

Bit 6=1

Bit 7=1 End of file encountered while spacing tape or
4Ø2ØØØ reading a record (ERCODE +1)

Bit 8=1 End-of-tape mark, seen during previous operation.
4Ø1ØØØ

Bit 9=1 Load point encountered while back-spacing tape.
4ØØ4ØØ That is, user requested to back-space more blocks than were on the tape before the point at which the tape rested.

Bit 1Ø=1 User attempted to write on a tape with the write
4ØØ2ØØ ring out.