MISCELLANEOUS IOT'S

The IOT's described here are:

Reader		
	GRDR = IOT 4100	/get reader
	RRDR = IOT 4300	/release reader
	RPA = IOT 4200	/read character
	RPB = IOT 10200	/read binary punch
	ERIM = IOT 10300	/enter readin mode
Punch		
	GPUN = IOT 4400	/get punch
	RPUN = IOT 4600	/read character
	PPA = IOT 4500	/punch character
	PPB = IOT 10400	/punch binary word
Soroban		
	SOT = IOT 5200	/type on Soroban
Time and Date		
	GTD = IOT 3400	/get startup time and date
	GTD+1	/actual time and date
Real-Time Clock		
	RCK = IOT 4700	/read clock
P-Pointers		
	WPP = IOT 3200	/write location
	PEEK = IOT 3700	/read location
List Pairs		
	ADDLP = IOT 1300	/EVD
	ADDLP+1	/TOT
	GETLP = IOT 1400	/get list pairs
Restart Mode Control		
	RSMC = IOT 1200	/leave
	RSMC+1	/enter
	RSMC+2	/debreak
	RSMC+3	/debreak and enter restart mode
Miscellaneous		
	DELAY = IOT 1600	/delay
	DELAY+40	/delay for n seconds
	HALT = IOT 12401	/halt
	HOLD = IOT 2000	/hold a number
	PSU = IOT 1500	/program start up
	DDTGO = IOT 10063	/start program under DDT
	C16RET = IOT 4000	/misc. Exec functions
	SUPGO = IOT 14100	/full core segment IOT

PAPER TAPE READER

A program must own the reader in order to reference it. If a program executes an RPA, RPB, or ERIM without owning the reader, the IOT is considered illegal and the program crashes. A program may "get" the reader even if it already owns it.

Use of reader IOT's.
Get the reader and clear the reader buffer:

```
        GRDR
        R1                          /owned by another program
        R2                          /gotten
```

There is no trap mode for this IOT's error return. Option: GRDR+4Ø does not clear the reader buffer.

Release the reader:

```
        RRDR
        R1                          /released
```

There is no error return for this IOT. If a program executes an RRDR and does not own the reader, the IOT is ignored.

Read an alphanumeric character:

```
        RPA
        R1                          /character in low order 8
                                    bits of IO
```

RPA takes the next character from the reader buffer and places it in the low-order 8. bits of the IO. The remainder of the IO is cleared. If no characters are available, the user is "reader-bung" until the reader buffer fills.

Read paper tape binary:

```
          RPB     /read a binary word from paper tape
```

RPB reads a binary word into the IO from paper tape. A binary word consists of three binary characters assembled in the following way.

low-order 6 bits of 1st character	low-order 6 bits of 2nd character	low-order 6 bits of 3rd character

A binary character is one that has the 8th hole punched. The 7th hole (normally Ø) is ignored. The IOT continues reading tape until it has found three such characters, skipping over any that do not have the 8th hole punched. RPB is a core 16 IOT that executes RPA's.

Enter readin mode:

```
          ERIM    /enter simulated readin mode
```

ERIM is a core 16 IOT which calls RPB repetitively. The tape in the reader is assumed to be punched in "readin mode" format, and this IOT simulates the action of the PDP-1 readin mode hardware. The readin mode paper tape format consists of alternate "address" words and "data" words. Each address word (if positive) indicates the address in which to store the data word that follows. When an address word is encountered which has its sign bit set, readin mode terminates and the IOT returns to the address in user core indicated by the low-order 12. bits of this negative address word.

PUNCH

A program must own the punch in order to reference it. If a program executes a PPA or PPB without owning the punch, the IOT is considered illegal and the program crashes. A program may "get" the punch even if it already owns it.

Use of punch IOT's

Get the punch:

```
          GPUN
          R1        /owned by another program
          R2        /gotten
```

There is no trap mode for this IOT's error return.

Release the punch:

```
          RPUN
          R1        /released
```

There is no error return for this IOT. If a program executes an RPUN and does not own the punch, the IOT is ignored.

Punch an alphanumeric character:

```
          LIO       /low order 8. bits
          PPA
```

PPA takes the character in the low-order 8. bits of the IO and puts it in the punch buffer. (The IO is unchanged and the high-order 10. bits are ignored.) If there is no room in the punch buffer, the user is "punch-hung" until the buffer empties.

Punch paper tape binary

```
          LIO  (BINARD WORD
          PPB       /punch a binary word on paper tape
          R1
```

PPB punches the 18.-bit contents of the IO on paper tape as a binary word (see description of RPB for binary word format). It is a core 16 IOT and executes three PPA's. Note that this IOT, unlike the hardware PPB, is the inverse of RPB.

SOROBAN

SOT allows the user to type on the Soroban. It is called with the AC pointing to a concise code text string to be printed. The string terminates with a character code 56, which acts as "end of message."

GET TIME AND DATE

Standard time and date format is two words: date in the first (or AC) and time in the second (or IO). The time is represented in minutes since midnight. The date is represented in days since 1 January 1849, which is defined as day Ø.

Use of time and date IOT's:

The following two IOT's have only one return - date is in the AC and time in the IO.

```
        GTD                         /get time and date of the
                                     startup of this program
        R1                          /AC = Date;  IO = time

        GTD+1                       /get actual time and date

        R1                          /AC = Date;  IO = time
```

CLOCK IOT

The clock counts milliseconds within current minute. It is in sync with time and date.

Use of the clock IOT:

```
        RCK                         /read millisecond clock
        R1                          /AC or IO holds "time"
Option:
        Bit 13. = 1                 /place time in IO
                = Ø                 /place time in AC
```

P-POINTER IOT'S

P-Pointer IOT's allow a program to reference a memory location in any of the cores. (The term "p-pointer" is a carry-over from Exec II).

Use of p-pointer IOT's:

Read a word, given a 16.-bit address.

```
        LAC or LIO (ADDRESS         /16. bits
        PEEK
        R1                          /word in AC or IO
```

PEEK has the following options:

```
        Bit 12 (PEEK+4Ø):           off means take address from AC
                                    on means take address from IO
        Bit 13 (PEEK+2Ø):           off means place word in AC
                                    on means place word in IO
```

Write P-pointer; store 18.-bit word via 16.-bit address

```
        LAC (ADDRESS                /16 bits
        LIO (Q                      /quantity to be deposited
        WPP
```

It is illegal to try to write into locations Ø - 7 of core Ø or in any of the Exec cores unless sense switch five is up.

LIST PAIRS IOT'S

Programs may communicate with Type Out Text and Event Detector by means of the List Pairs IOT's. A complete description of the why and how of these IOT's is contained in their respective memos. Add List Pairs adds a pair of words to a small Exec buffer to wait for the "Get List Pairs" call from Event Detector or Type Out Text.

Use of Add List Pairs IOT

The two-word "message", is in the AC and IO. The AC contains a number which may be, for example, a Teletype number or a class and item type. The IO is a drum address or -Ø to indicate this pair does not contain a drum address. There are two returns for this IOT. R1 says there is no more room in Exec's buffer. Note: If bit 16. is set, instead of R1 for "full", the user gets RØ; instead of R2 for "added", the user gets R1.

Add List Pair - Event Detector

```
     LAC NUM
     LIO DRA                        /or -Ø
     ADDLP                          /add list pair
     R1                             /no room
     R2                             /added
```

Add List Pair - Type Out Text

```
     LAC TTNUM                      /Teletype number
     LIO DRA                        /DRA of message
     ADDLP+1                        /add list pair
     R1                             /no room
     R2                             /added
```

There is no trap mode for error returns for this IOT.

Use of Get List Pairs IOT's

Bit 17 of the IOT designates the buffer from which a list pair is to be taken: Event Detector if Ø, Type-Out-Text if 1. If one or more list pairs are available in Exec's buffer, the first of these is placed in the user's AC and IO (in the same order in which it was received by ADDLP) and the user is given R2. If the buffer is empty, two options are available, selected by bit 14 of the IOT.

Bit 14=Ø. An indicator in Exec core is set whenever a condition occurs which would "un-hang" EVD or TOT (e.g., a TT alarm or a 5-minute-clock alarm). There are two such indicators, one for EVD and one for TOT. When one of these programs executes a GETLP"U'1Ø and the appropriate buffer is empty, it is given R1 right away if the corresponding alarm indicator is set. Otherwise, the program is "Get-List-Pairs hung."

Bit 14=1. If no list pairs are available, the user is given R1.

RESTART MODE CONTROL

A program running in restart mode is automatically interrupted and removed from restart mode when one of its Teletypes sends a break. (Note that a program is not interrupted until the end of an I-O processor command). The PC is saved in RESTPC and the program is started up at RESTSU. If the program wishes to resume after it has handled the break, it may execute a debreak if it has not executed any common routine IOT's, or re-entered restart mode since the break. (These conditions cause "debreak" to be illegal).

Use of restart mode IOT's:

```
        RSMC                    /leave restart mode
        RSMC+1                  /enter restart mode
```

Two added to either of these IOT's will cause a debreak also.

Four combinations are possible:

enter restart mode and debreak

debreak

enter restart mode

leave restart mode

MISCELLANEOUS

DELAY IOT

When a program executes the Delay IOT it is dropped one queue level and its priority word is set to zero, which is lowest priority.

```
        DELAY                   /delay
        DELAY+4Ø                /delay for n seconds
```

Delay+4Ø sets the user to "clock-hung" status for the number of seconds in the AC. (Ø in AC = Ø to 1 second; 1 in AC = 1 to 2 seconds, etc). The hung status will end when the time is up, when a break is received on a Teletype owned by the user, or when the user is examined by Exec DDT. At this time the user will be put on high queue. Delay+4Ø does not change the user's priority.

HALT

HALT /halt

Halt types out the time, the day if it differs from the startup day, the year if it differs from the startup day and executes an HLT.

HOLD A NUMBER IOT'S

HOLD allows a user program to specify one or more 2Ø. bit numbers to be held or released.

The IOT saves 18. bits taken from the AC, IO, or from a list, and includes bits 15-16 of the IOT itself (the IO processor "third" bits) for its 2Ø. bit number. When operating in list mode, HOLD takes a pointer to a list in either the AC or IO, and the length of the list in the other live register. HOLD will ignore extend bits in the pointer to the list, but the list cannot go off the end of core.

Options:

HOLD: hold the number in the AC, two returns.
R2 → number held.
R1 and IO = Ø → number is already held by someone else.
R1 and IO = -Ø → you should have gotten the number but the Exec table overflowed.

HOLD+1: hold the number in the IO, two returns as in HOLD.

HOLD+4Ø: release the number in the AC, one return.
(Note: It is not an error to release a number you don't own).

HOLD+41: release the number in the IO, one return.

HOLD+2Ø: release all numbers held by this user, returns the number of numbers released in the AC, one return.

HOLD+1Ø: hold the list specified by a pointer in the AC and a length in the IO, two returns as in HOLD but in the case of R1, also returns a pointer to the offending number in the AC.

HOLD+11: like HOLD+1Ø except that it takes the pointer in the IO and the length in the AC.

HOLD+5Ø: releases the list specified by a pointer in the AC and the length in the IO, one return.

HOLD+51: like HOLD+5Ø except that it takes the pointer in the IO and the length in the AC.

PROGRAM START-UP

The PSU IOT enables one running program to start up another program, running simultaneously with the first and with a different program number. To do this, the original program first appends two words to the item addressed by invariant number 6 and rewrites the item. These two words are, respectively, the starting address and drum address of the library program to be started up. Next, the original program executes a PSU, thereby completing its task.

Whenever a PSU is executed, Exec finds a free program number, reads into user core the item addressed by invariant number 6, and starts up the short program which is contained in the first part of this item. (The list of word-pairs for pending start-ups is at the end of the item). This program shortens the item by one word-pair and rewrites it. Then, if there are any more word pairs in it, the program executes <u>another PSU</u>, thus continuing the chain. Finally, it executes a SUPGO IOT which reads into core (beginning at register 36) the item designated by the second word of the pair and starts the program contained in this item at the address designated by the first word of the pair.

SUPGO

SUPGO is a core 16 IOT which reads into core (beginning at register 36) an item whose drum address is in the IO and starts the program contained in the item at the address specified in the AC. When the startup program uses SUPGO to read a library program into a DDT core-image <u>without</u> starting it, the starting address is complemented to indicate this to SUPGO, which then returns to register 36 of user core.

DDTGO

DDTGO is a Core 16 IOT used by DDT to start a core-image segment running. It writes DDT out on the Fastrand, reads in the segment, and executes a C16RET+2, which sets the "running under IDDT" bit in the bits p-pointer and starts the segment.

CORE 16 EXECUTIVE IOT'S

Certain Executive functions are best performed by coding within Core 16, running as a user. C16RET facilitates communication between these routines and other parts of Exec. The low-order 6 bits are dispatched upon to provide as many as 64. distinct functions. Use of this IOT is entirely internal to Exec, and it is illegal if executed in user-core.

Use of C16RET IOT'S

C16RET /normal return from Core 16

C(C16AC) → user's AC
C(C16PC) → user's PC
C(C16IO) → user's IO
C(C16FLA)→ user's flags

C16RET+1 /abnormal return from Core 16

C(C16AC) → user's AC
C(C16PC) → user's PC
C(C16IO) → user's IO
C(C16FLA)→ user's flags

After these registers are transfered, this becomes an "illegal IOT", causing a crash or a return to IDDT.

C16RET+2 /halt horrible return

If C(AC) Ø set bit Ø of word Ø.

C(HH16AC) user's AC
C(HH16PC) user's PC
C(HH16IO) user's IO
C(HH16FLA) user's flags

And restart user at location specified by user's PC.

C16RET+3 /halt horrible type out

Bit 5 of register Ø of user core is set (indicating a "halt horrible"). Then program number C(BILLTT)$_{12-17}$, C(PROGAD)$_{9-17}$, C(AC), C(IO) are typed on Soroban. When IOT is executed, C(AC) = contents of trap buffer and C(IO) = drum address of halt horrible item.

C16RET+4 /Fastrand swap in
C16RET+5 /Fastrand swap out
C16RET+6 /Fastrand swap

Core is written out on Fastrand slot specified by C(AC)$_{9-17}$ if swap out or swap. Slot is read in from address specified by C(AC)$_{\varnothing-8}$ if swap in or swap. If successfully completes operation on Fastrand C(PC) = location C16SWF in Core 16, otherwise C(PC) = location C16SWF-1 in Core 16.