PDP-1 COMPUTER
ELECTRICAL ENGINEERING DEPARTMENT
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS
02139

PDP-35-1

INSTRUCTION MANUAL

PART 5 -- MTA'S AND IVK'S

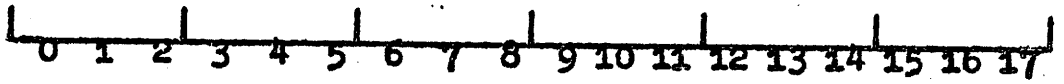
15 April 1971

Introduction

This memo describes the interface between user programs and the PDP-1 timesharing supervisor. Sections identified with an asterisk (*) may be omitted on a first reading.

Needless to say, this memo is subject to change without much notice. Spheres and entries are especially likely to undergo development.

The following convention is used to specify the location of information in registers:



Bit 0 is the sign bit; bit 17 is the "least significant" bit. An expression such as A(n-m) refers to the contents of bits n through m of register A. For example, I(9-17) refers to the right half of the in-out register.

Most supervisor calls are handled by meta-instructions (mta=770070). Information on what action to take is contained in bits 9-11 and 15-17 of the instruction itself, as well as in the live registers.

In addition to the hardware registers A, I, X, F, and G, each process has a W register, which is maintained entirely in software by the supervisor. Certain operations require or supply information in the W register. To facilitate the writing of reentrant procedures, supervisor calls never make use of X.

Instruction	Action
mta 0	Copy A into W.
mta 1	Copy I into W.
mta 2	Copy W into A.
mta 3	Copy W into I.

Since these instructions are interpreted by the supervisor, they are much slower than similar hardware instructions such as lia.

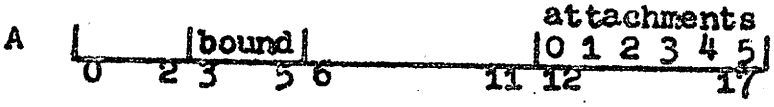
The instructions mta 4, mta 5, mta 6, and mta 7 are special instructions which trap to the program's superior (usually ID). Mta 7 (dsm, "dismiss") is used to signal normal completion to ID. ID interprets mta 5 as a request to perform a command string (see memo PDP-23).

Core Memory

Each program has an address space consisting of those core memory addresses it is permitted to reference. The address space is divided into units of 10000 (octal) words, called "core modules" or simply "cores". The address space consists of "real" cores, numbered from 0 up to (at most) 5, and attachments (see the section on spheres for a discussion of attachments). The memory bound is the lowest address not in a real core. Upon logging in, the memory bound is 10000 and there are no attachments.

mta 206

Read memory bound and attachments to A.



mta 207

Set memory bound. The number of cores is specified in A(3-5) or, if that is zero, A(15-17). The memory bound cannot be lowered to zero or raised above 60000. The content of existing cores is not changed. If the memory bound is raised, any attachments that are in the way are automatically removed. The instruction skips if successful. If unsuccessful, the memory bound and all attachments are unchanged.

(*) mta 205

Detach. The core number is taken from A(3-5) or, if that is zero, A(15-17). The instruction skips unless it is an attempt to detach a real core.

Attempts to reference addresses outside of the address space are illegal, and normally trap to the program's superior (ID prints << and the offending instruction).

(*) A program may intercept these traps by setting its illegal memory reference return. When an illegal memory reference occurs, control is transferred to the address specified by the illegal memory reference return. W(3-17) will contain the program counter at the time of the illegal memory reference. W(0-2) will contain the core which was referenced. The original contents of W are lost. Other registers are unchanged. If AAL (see part 2 of this Instruction Manual) was on when the illegal memory reference occurred, it will be on. The illegal memory reference return may be disabled by setting it to a negative number. It is initially disabled.

mta 202

Read illegal memory reference return to A.

mta 203

Set illegal memory reference return from A.

Illegal Instructions

Certain instructions are illegal. Normally these trap to the program's superior (ID prints >> and the offending instruction).

(*) A program may intercept some of these traps (recoverable illegal instructions) by setting its illegal instruction return. The following instructions are unrecoverably illegal: hlt, bpt, opcode 00, priviledged instructions, and attempts to reference locations 0 through 77 when PRL is on (see below). All other illegal instructions are recoverable. When a recoverably illegal instruction is executed, control is transferred to the address specified by the illegal instruction return. W(3-17) will contain the program counter at the time of the illegal instruction (i.e., the address of the instruction). W(0-2) are cleared. The original contents of W are lost. Other registers, including AAL, are unchanged. The illegal instruction return may be disabled by setting it to a negative number. It is initially disabled.

mta 200

Read illegal instruction return to A.

mta 201

Set illegal instruction return from A.

(*) Low Priority Mode

A process may enter low priority mode by executing a mta 100. Thereafter, the process will run only if no other processes (except other low priority mode processes) want to run.

Capabilities

System resources, such as tape drives, drum fields, and typewriters, are made available to the user through capabilities. A user is permitted to use a resource if he owns a capability to the resource. Each capability is distinguished by an index from 0 to 77. The index of a capability is specified when the capability is created (though it may be changed later by mta 401), and no two capabilities may have the same index. The capabilities are stored in an area of memory known as the C-list.

Upon logging in, the user has only one capability, to his console typewriter, at index zero. Also, only indices from 0 to 17 may be used; to increase the available range to 77, see the description of mta 403 below.

To invoke a capability which he owns, a user executes an ivk instruction (ivk=740000). Bits 12-17 of the ivk instruction are the index of the capability being invoked (e.g., ivk 14 operates the resource at capability index 14). Bits 8-11 of the ivk instruction are called the variant, and, along with A, I, and W, may specify what operation is to take place. The effect of the ivk instruction differs for each type of capability. The remainder of this memo describes the types of capabilities which may be created and how they are used.

Capabilities are created with instructions of the form mta 30n, $0 \leq n \leq 7$. For all such instructions, the index of the capability is specified by A(12-17); if this is zero, the first free index is used. The index is returned in A(12-17) with A(0-11) cleared. The I register and A(0-11) may contain parameters pertaining to the capability to be created. The capability created refers to a new object, not owned by anyone else. (From the point of view of the supervisor, objects are "allocated"; from the point of view of the user, objects are "created".) If successful, the instruction skips. A copy of the capability (what mta 400 (q.v.) reads) is returned in I. If unsuccessful, the instruction does not skip. Either the requested index (or, if 0 was requested, every index) was already occupied, in which case the capability at that index (or, if 0 was requested, the last index) is returned in I, or else the index was available but there were insufficient resources (e.g. drum fields) to create the object, in which case I is cleared.

The following instructions manipulate capabilities:

mta 204

> Delete capability. A(12-17) contains the capability index. If there is no capability at the specified index, no action is taken. A and I are unchanged.

mta 400

> Read capability. A(12-17) contains the capability index. An 18-bit word uniquely specifying the capability is placed in A. If no capability exists at the specified index, A is cleared.

mta 401

Exchange capabilities. The capabilities at the indices in A(6-11) and A(12-17) are exchanged. Null capabilities may take part in the exchange.

mta 402

Turn off PRL. Undoes the effect of mta 403 (q.v.). Capabilities at indices 20 through 77 are deleted. (For spheres not at the top level, capabilities at indices 0 through 17 are also deleted. See the section on spheres for details.)

mta 403

Turn on PRL (program reference list). When PRL is on, capability indices 0 through 77 may be used. The C-list is stored in locations 0 through 77 of the program's address space. To protect against unauthorized modification of this information, the program is not permitted to examine or modify these locations. The state of PRL is read by bit 8 of the cks word (see part 3 of this Instruction Manual).

mta 404

Count capabilities. Return number in A.

mta 405

Copy capability. The capability at the index in I(6-11) is copied into the index in I(12-17) (or the first free index if I(12-17) is zero). The index is returned in A. The instruction skips if successful, and a copy of the capability is placed in I. If unsuccessful, the instruction does not skip. Either a capability already exists at the index in I(12-17) (or at all indices if I(12-17) is zero), in which case that capability (or the last capability) is placed in I, or else no capability exists at the index in I(6-11) or the capability there is an entered process capability, in which case I is cleared.

(*) mta 501

Disown capability. The capability at the index in I(6-11) is deleted without deleting the corresponding object. (For example, if a sphere capability is disowned, the sphere continues to run, occupy storage, own other capabilities, etc.) An index is returned in A(12-17) with A(0-11) clear; this index is to be used to reclaim the capability (see mta 502). If successful, the instruction skips, and a copy of the capability is placed in I.

(*) mta 502

Claim capability. The disowned capability whose index is in I(6-11) is placed in the index in I(12-17) (or the first free index if I(12-17) is zero). The index is returned in A. If successful, the instruction skips, and a copy of the capability is placed in I. If unsuccessful, the same action is taken as for mta 405.

A program is normally given a typewriter capability at index zero by its superior. A typewriter may be created, however, if it is not logged in or otherwise owned. Mta 306 with 05 in A(0-5) creates a typewriter capability. A(6-11) has the desired console number. A(12-17) has the capability index. The instruction skips if successful.

For typewriter ivks, if the variant is zero, A(8-10)+1 is used to specify the operation.

Variant	Operation
1	Type out from A. The flexo code character in A(12-17) is typed out.
2	Type in to A. The character typed is placed in A(12-17) with A(0-11) clear.
3	Type out from I. The flexo code character in I(12-17) is typed out.
4	Type in to I. The character typed is placed in I(12-17) with I(0-11) clear.

(*) It is occasionally desirable to share a typewriter with another program, but in such a way that the original owner may assert control over the typewriter when it wishes, without finding and deleting all copies in the recipient. (For example, ID wants to share its typewriter with the user under it.) An inferior typewriter capability is a typewriter capability with an enable/disable switch. When the typewriter capability is disabled, all ivks on it will hang until the capability is enabled. The recipient of an inferior typewriter capability is not aware of the enabling and disabling. Inferior typewriter capabilities may be created to any reasonable depth, and each has its own enable/disable switch.

Variant	Operation
5	Enable.
6	Disable.
7	Unused.
10	Unused.
11	Turn off enable/disable permit for the ivk'ed capability. Enable/disable permit is a property of each copy of the

capability, not of the inferior typewriter as a whole. This is used to prevent the recipient of an inferior typewriter capability from interfering with the enabling/disabling activities of the superior.

12

Create inferior typewriter. The ivk'ed capability is replaced by an inferior typewriter capability. Skip if successful. The new capability will be disabled, with enable/disable permit on.

Paper Tape Reader

The PDP-1 has a photoelectric paper tape reader capable of reading 320 or 640 lines per second, as selected by a toggle switch on the left side of the reader rack. Eight-hole tape is normally used, although five, six, and seven hole tape may also be read. Mta 306 with 03 in A(0-5) creates a reader capability. A(12-17) has the capability index. A(11) specifies how ivks on the reader will be treated: 0 for alpha mode, 1 for binary mode. The mta 306 skips if successful. For reader ivks, the variant and A are ignored.

Alpha mode

One line of tape is read and placed in A(10-17). A 1 represents a hole, a 0 represents no hole. Channel 8 (the channel farthest from the feedholes) goes into bit 10, channel 7 into bit 11, etc. A(0-9) are cleared. If successful, the ivk skips. If the reader is out of tape, the ivk does not skip.

Binary mode

Three lines of tape are read. Channel 8 must be punched; if not, the line is ignored. Channel 7 is always ignored. Channels six through one of the first line are placed in A(0-5); A(6-11) is filled from the second line, and A(12-17) from the third. If successful, the ivk skips. If the reader is out of tape, the ivk does not skip.

Paper Tape Punch

The PDP-1 paper tape punch punches standard eight-hole tape at a speed of 63 lines per second. Mta 306 with 04 in A(0-5) creates a punch capability. A(12-17) has the capability index. The mta 306 skips if successful.

A punch ivk causes one line of tape to be punched. Tape channels eight through one come from A(10-17) respectively. The feedhole is always punched.

Button Console

The consoles of buttons and switches are described in part 3 of this Instruction Manual. Mta 306, with 01 in A(0-5), creates a button console. A(10-11) has the console number. A(12-17) has the capability index. The mta 306 skips if successful.

Invoking a button console capability hangs until the state of the buttons is different from the contents of A. The new state of the buttons is placed in A (in the same format as rbt) and the instruction completes. Rbt instructions work independently of button console capabilities.

Temporary Clock

Mta 306 with 02 in A(0-5) creates a clock capability. A(12-17) has the capability index. The mta 306 skips if successful; it fails only if the capability index was already occupied.

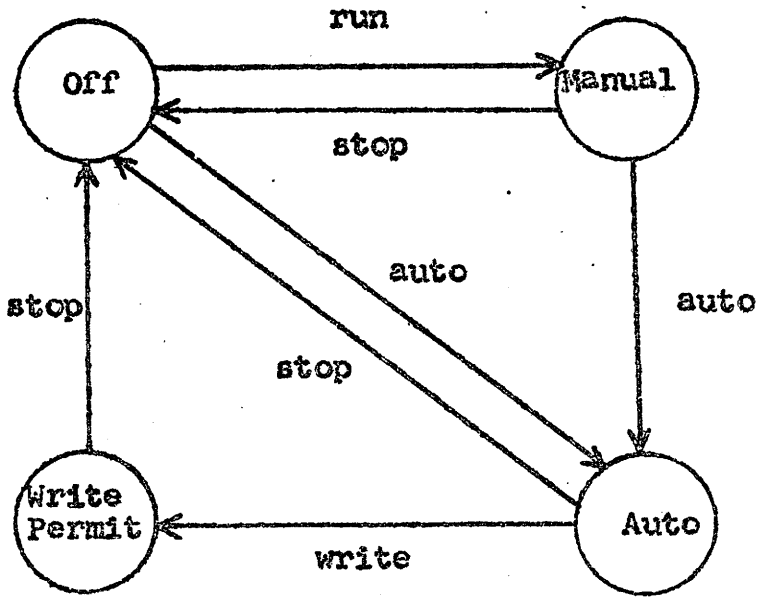
The temporary clock ticks 1760 times per minute (about 30 ticks per second). A clock ivk hangs for -(contents of A) ticks. A is incremented at each tick, and the instruction completes when A is positive or zero.

Microtapes

The PDP-1 has four microtape (Dectape) transports. A standard reel of tape has 1000 (octal) blocks of 400 (octal) words each. There are a number of "public" microtapes providing convenient storage for users.

Mechanical operation of the tape drives

The following state diagram shows how the drives are controlled.



When a drive is off, the motors are not operating, and a tape may be mounted or removed. When in manual mode, the motors are under control of the "fwd" and "rev" buttons. When in automatic or write permit modes, the motors are under control of the computer and the tape may be read or written under program control.

To mount a tape, press it firmly onto the left hub, draw the tape over the head and onto the empty reel on the right hub, and wind it onto that reel one or two turns by hand. Press the "run" button to place the tape in manual status, and run the tape forward for a few seconds. Then lift the "auto" button, followed by the "write" button if desired.

To remove a tape, press the "stop" button, followed by the "run" button. Move the tape in reverse by means of the "rev" button until it comes completely off the take-up reel. Press "stop", stop the coasting tape by hand, and remove it.

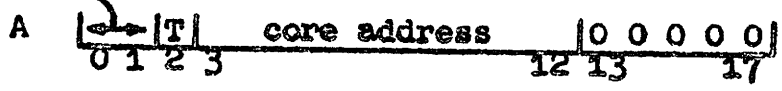
Use of Microtapes

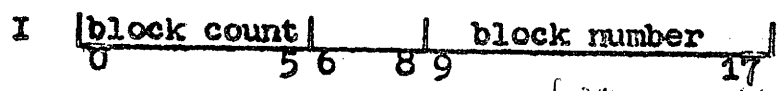
Normally, microtapes are used through the Microtape File System (see memo PDP-42). All tape instructions operate directly on microtapes, without reference to any file structure. Mta 306,

with 00 in A(0-5), creates a microtape capability. A(8-11) has the desired transport number (0-3). A(12-17) has the capability index. It skips if successful.

On a microtape ivk, the variant is ignored. A(0-1) is decoded as follows:

00 - read
10 - write
01 - rewind. The instruction completes when the rewind is done.
11 - rewind. The instruction completes immediately. Any subsequent tape instruction will override the rewind.





If zero, one block is transferred.

For read and write operations, I(0-5) contains the number of blocks to be transferred; I(9-17) contains the block number of the first block to be transferred. Consecutive blocks are transferred to consecutive areas in core. Block 0 is considered to follow block 777. The core address of the first word of the first block is given by A(3-17); it must be a multiple of 40 words. No block may be transferred partly into one core and partly into another, but different blocks may be transferred to different cores in the same operation.

If A(2)=0, the block number in I(9-17) is translated into a physical block number. This is the normal case, as it allows consecutive block numbers to be read in one pass across the tape. Block numbers 0, 1, ..., 377, 400, 401, ..., 777 translate to physical blocks 1, 3, ..., 777, 776, 774, ..., 0. If A(2)=1, the physical block number in I(8-17) is used without translation. This allows tapes with more than 1000 blocks to be read.

If all blocks are transferred successfully, the instruction skips, leaving the address of the last word transferred+1 in A(3-17) with A(0-2) unchanged, the number of the last block transferred+1 in I(9-17), and zero in I(0-5). If an error occurs on any block, the instruction does not skip, I(9-17) contains the number of the block in which the error occurred, and I(0-5) contains the number of blocks remaining, including the one that was in error. Furthermore, an error code is placed in As and w

0 - tape unit is not in automatic status
1 - block cannot be found (probably bad tape)
2 - illegal core address
3 - checksum error (the data transfer took place anyway)
4 - mark track error (probably bad tape)
5 - data channel error (serious hardware malfunction)
6 - no write permit

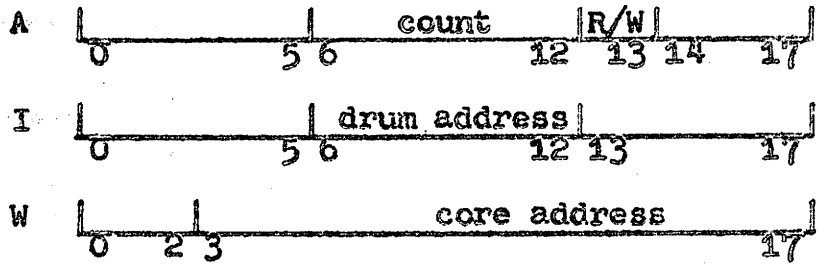
For rewind operations, the instruction skips unless the tape unit is not in automatic status, in which case A is cleared.

Drum Field

A drum field is a 10000 (octal) word block of medium-speed memory. Transfers take place between the drum and core memory. The initial drum address and count (number of words transferred) must be multiples of 40 (octal) words. A drum field, like the drum, is circular; if a transfer extends past the end of the field, it will "wrap around" to the beginning of the same field. A transfer may not cross a core boundary. The core address need not be a multiple of 40 words. If the count is zero, 10000 words will be transferred.

Mta 300 creates a drum field. The capability index is specified by A(12-17). If A(0) is 1, the absolute field number in A(5-11) is used, and writing on that field will be illegal (any absolute field may be created in this "read-only" mode). The mta 300 skips if successful.

The core address for the transfer is specified by W(3-17), the drum address/40 by I(6-12), and the word count/40 by A(6-12). Note that the drum address and word count appear in the "normal" position in the word. A(13) is 0 to read from the drum, 1 to write. The drum ivk skips if no drum error occurred.



Mta 104 and mta 105 are used to read any absolute drum address. The format is the same as for a drum ivk, except that the drum field comes from I(0-5). Mta 104 reads fields 0-77, mta 105 reads fields 100-177. Mta 104 and mta 105 skip if no drum error occurs.

Programmed Queue

Queues are described in Part 4 of this Instruction Manual (Multiprocessing).

(*) Hardware I/O

The design of the PDP-1 allows certain I/O operations to be performed directly by users. An extensive and somewhat accurate description of this facility may be found in memo PDP-33, Input/Output in the PDP-1-X. PRL must be on to operate hardware directly (see description of mta 403).

Mta 306, with 07 in A(0-5), creates a hardware I/O capability. A(6-11) has the I/O device number. A(12-17) has the capability index. The mta 306 skips if successful.

The following device numbers are currently used:

1 - new drum A
2 - new drum B
16 - teletype input
17 - teletype output
20 - microtape unit monitor
21 - microtape data control
22 - speech controller
23 - ditto
24 - ditto
77 - microtape motion control

Some of these devices are described in separate memos. Most require special turn-on procedures. Devices 20, 21, and 77 cannot be assigned; microtapes are referenced by other means.

(*) Sphere

One of the important concepts in timesharing is that of extensibility. A timesharing system is extensible if it is possible, from a console, to construct another timesharing system underneath the first. It should be possible to change the characteristics of the system; for example, one might want to cause what would normally be typewriter output to become spoken speech. Extensibility goes a long way toward making a timesharing system both useful and general. The PDP-1 timesharing system is extensible. The objects which make this possible are spheres and entries.

Everything which has been said about "programs" in reality applies to spheres. The user gets a sphere when he logs in. Each sphere has its own address space, processes which run in that address space, and capabilities which those processes may invoke.

The initial user sphere may have capabilities 0 through 17 even when PRL is off. Other spheres may not; they must have PRL on to have any capabilities.

Mta 302 creates a sphere. A(12-17) has the capability index. I(3-17) has the fault entry address (see below). The mta 302 skips if successful. The sphere in which the mta 302 is executed becomes the superior of the sphere which is created. The initial state of the sphere is:

memory bound	10000
attachments	none
PRL	off
processes	none
process hoard	0
run indicator	off
breakpoint variables	-0,0,0 (disabled)
illegal instruction return	-0 (disabled)
illegal memory reference return	-0 (disabled)

There are two kinds of sphere capabilities, master and non-master. Mta 302 creates a master sphere capability. There is at most one copy of a master sphere capability, owned by the sphere's superior. Whenever a master sphere capability is granted or shared (by mta 405 or an appropriate sphere ivk), the new copy is a non-master sphere capability. When a master sphere capability is deleted (either explicitly or by granting it), the sphere no longer has a superior. The sphere itself is deleted only when all capabilities referring to it, master and non-master, are deleted. Mta 501 and mta 502 are considered grants.

Core modules that are not assigned as "real" cores may be made into attachments. An attachment is a core module that is shared among several spheres. It is a "real" core module in one

of the spheres and an attachment in each of the others. Memory references to an attachment are directed (efficiently) to the attached "real" core module. Cores which are attachments need not be consecutively numbered. An attachment will disappear without warning if the attached core is deleted. The core 0 of a sphere with PRL on cannot, unfortunately, be attached; the read/write sphere ivk must be used to reference such a core.

Each sphere has an enable or run indicator. Processes in the sphere may run only if the indicator is on. When the indicator is off, no processes may run or be created by an enter. When the run indicator is turned off, processes that are in "soft" waits (all waits except enter, queue, and hardware I/O waits) will be removed from the wait. They will re-execute the instruction when the run indicator is turned back on. Processes that are in enter, queue, or hardware I/O waits may have the wait complete, but they will not resume running. Certain operations on spheres, such as manipulation of processes, may be done only when the run indicator is off.

Certain instructions, including all unrecoverably illegal instructions, are treated as enters into the superior sphere. (See the section on enters.) Recoverable illegal instructions and illegal memory references are treated as enters if they are not handled by the illegal instruction or illegal memory reference returns. If the sphere has no superior, the instruction waits until the sphere gets a superior (possibly forever).

When a superior enter happens, the entered process starts executing at the fault entry address. Its A register has the index of an entered process capability. Its I register has the reason for the trap:

0 - illegal instruction
1 - lock fault
2 - ESI trap
3 - I/O function busy trap (hardware devices only)
4 - bpt trap
5 - hlt
6 - illegal memory reference, return not enabled.
7 - unused
10 - mta 4
11 - mta 5
12 - mta 6
13 - mta 7 (dsm)

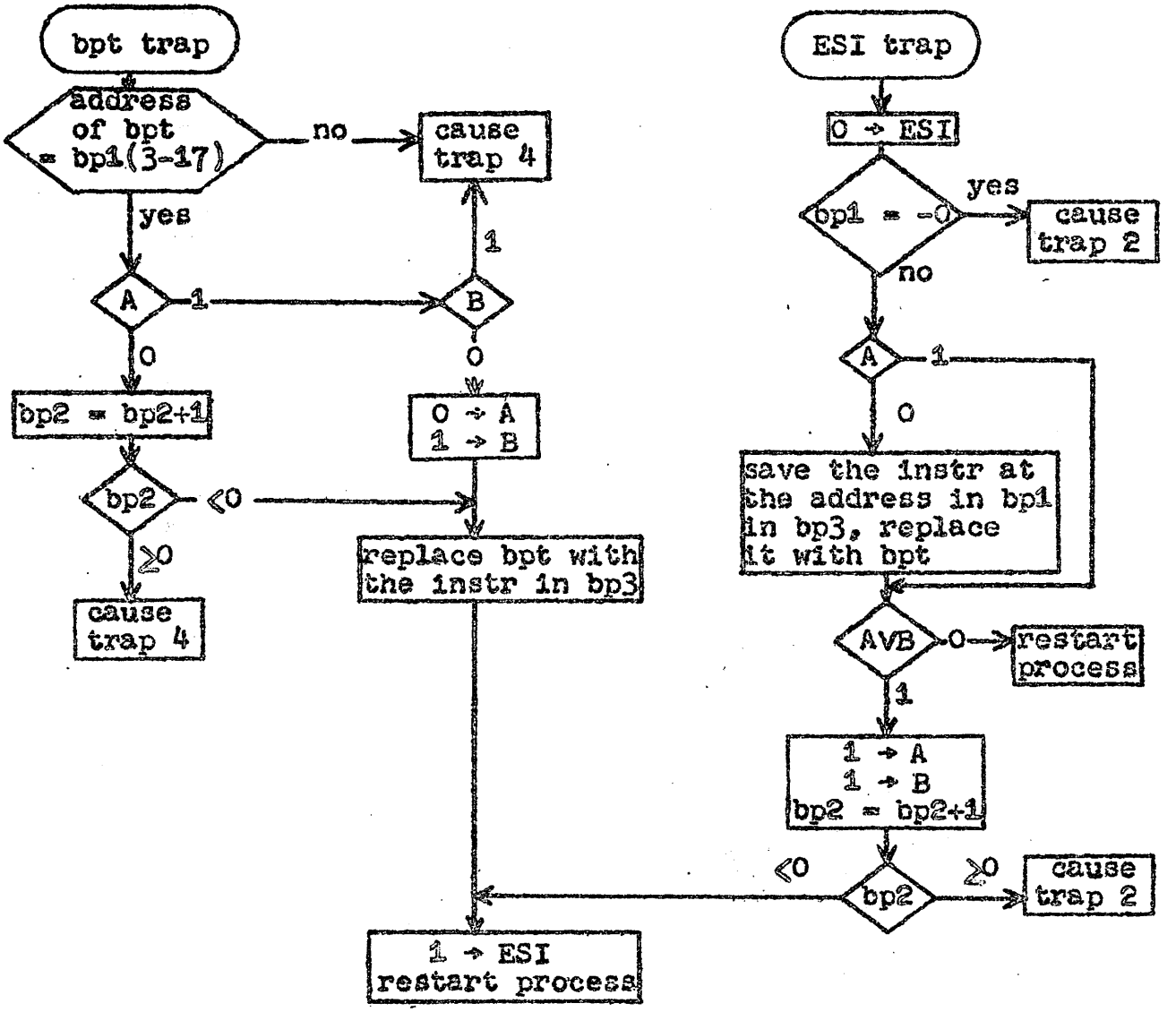
If the entering process is "restarted", it will enter again (immediately, unless its run indicator is off) without re-executing any instructions, unless its registers have been written on with a sphere ivk.

There are three "breakpoint" registers associated with each sphere hat enable the execution of instructions to be counted at (reasonably) high speed. The registers will herein be called bp1,

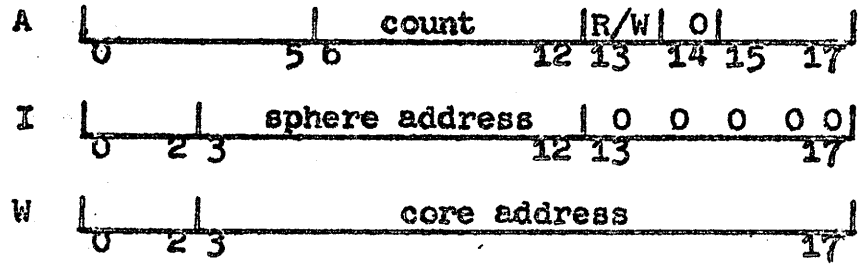
bp2, and bp3. Bit 6 of the F register is called ESI (execute single instruction). Whenever it is on, a special ESI trap occurs to the supervisor at the end of each instruction (including supervisor calls, enters, etc.) executed by the process. In addition, the bpt instruction ("breakpoint", 770044) causes a special bpt trap. The breakpoint mechanism can be disabled by setting bp1 to -0.

(**) Whenever bpt or ESI traps occur from any process (no checking is done to see that it is the same process each time), the following action is taken. Bp1(0) will be called A, bp1(1) will be called B.

"Meaning" of A and B:
AB = 00 : not a superproceed
10 : superproceed, haven't hit bpt yet
01 : hit bpt, proceeding under ESI
11 : proceeded, further bpts illegal



If A(14)=0, the operation is a read/write core. A(13) is 0 to read the invoked sphere's memory, 1 to write. Either of the core modules taking part in the transfer may be an attachment. The word count and sphere address must be multiples of 40. The transfer may not cross a core boundary in either sphere. If PRL is on in either sphere, locations 0 through 77 of that sphere may not be read or written on. The word count/40 is in A(6-12); if it is zero, 10000 words will be transferred. The address in the invoked sphere is in I(3-17), and the address in the invoking sphere is in W(3-17). The instruction skips if successful.



Other sphere ivks:

code in A	Operation
12	Suppress processing. The run indicator is turned off.
32	Permit processing. The run indicator is turned on.
52	Attach. The core module of the invoked sphere (the attachee) specified by I(15-17) becomes attached to the sphere executing the ivk as the core module specified by I(3-5). This instruction will succeed if the attached core exists (as an attachment or a real core) and the attaching core is not a real core. If the attached core is itself an attachment, its attachee will be used. If the attaching core is a previous attachment, it will be removed. An attachment may be made and maintained whether the run indicator is on or off. Skip if successful.
72	Reverse attach. Similar to attach. The core module of the invoking sphere specified by I(15-17) becomes attached to the invoked sphere as the core module specified by I(3-5). Skip if successful.

112

Read process state. The registers of the process whose number is in I are read and stored in six consecutive words beginning at the address in W(3-17). The order is A, G, I, X, F, and W. Processing must be suppressed. Processes are numbered beginning with 1. This instruction will fail if the numbered process does not exist or the run indicator is on. Skip if successful.

132

Write process state. Similar to read process state. This will fail if the numbered process does not exist, the run indicator is on, or the process is in a wait. Skip if successful.

152

Read breakpoint state. The three words of breakpoint state are read into three consecutive words beginning at the address in I(3-17).

172

Write breakpoint state. Similar to read breakpoint state.

412

Create process. A new process is created for the sphere, and becomes the highest numbered process. The process number is returned in A. The run indicator must be off. The instruction fails if no process is available. Skip if successful.

432

Delete process. The process whose number is in I is deleted. All higher numbered processes are renumbered. If the process is in a wait it will be deleted anyway, but the process hoard will diminish by 1. This instruction fails if the numbered process does not exist or the run indicator is on. Skip if successful.

452

Count processes. The number of processes in the sphere is returned in A.

552

Subjugate. The sphere in which the ivk was executed becomes the superior of the invoked sphere. The I register contains the new fault entry address. The invoked capability becomes a master sphere capability. Any processes waiting to enter will enter immediately unless the run indicator is off. This instruction fails if the sphere already has a superior. Skip if successful.

572

Execute meta. The W register is moved to A, and the meta-instruction whose code was originally in A(0-8) is executed as if by a process in the sphere. Values returned by the meta-instruction will be placed in A and I. Only instructions $\geq$ mta 200 may be executed. The ivk is illegal if the meta-instruction is illegal. Skip if meta-instruction skips.

612

Reverse share. A capability in the invoked sphere is copied and the copy placed in the C-list of the sphere executing the instruction.

632

Share. A capability in the sphere executing the instruction is copied and the copy placed in the C-list of the invoked sphere.

652 Reverse grant.

672

Grant. Grant and reverse grant are similar to share and reverse share, except that the donor's copy is deleted. Entered process capabilities may be granted (they may not be shared).

Format of (reverse) share and grant

The capability in the donor specified by I(6-11) is read and placed in the C-list of the receiver at the index in I(12-17) if non-zero, or the first free index otherwise. If successful, the index of the capability created in the receiver is placed in A and a copy of the capability in I. If unsuccessful, either the requested index in the receiver (or, if 0 was requested, every index) was already occupied, in which case the capability at that index (or if 0 was requested, the last index) is returned in I, or else the index in the receiver was available but no capability (or, in the case of (reverse) share, an entered process capability) existed at the specified index in the donor, in which case I is cleared. Skip if successful.

(*) Entry

An entry is a user-programmed capability. The action which occurs when an entry capability is invoked is entirely under the control of a program. For example, an entry can be used to simulate any of the other kinds of capabilities.

There are two kinds of entry capabilities, master and non-master. Invoking a master entry capability performs operations relating to the entry itself. Invoking a non-master entry capability sets in motion the entry mechanism to be described below.

For each entry there is at all times exactly one copy of a master entry capability, owned by the sphere which created the entry. Whenever a master entry capability is granted or shared (by mta 405 or an appropriate sphere ivk), the new copy is a non-master entry capability. When a master entry capability is deleted (either explicitly or by granting it), the entry becomes unuseable; invoking non-master entry capabilities corresponding to the same entry will be illegal.

Each copy of an entry capability has a 6-bit transmitted word. When an entry capability is granted or shared, the transmitted word is copied. The transmitted word of a master entry capability may be changed by invoking the capability with the desired transmitted word in A(12-17) and A(0-11) clear. The transmitted word of a non-master entry capability cannot be changed. The transmitted word is a tool for distinguishing several similar entry capabilities without creating a separate entry object for each one.

Mta 307 creates a master entry. The transmitted word is initially zero. The I register contains the entry address (explained below). A(12-17) contains the capability index. The mta 307 skips if successful.

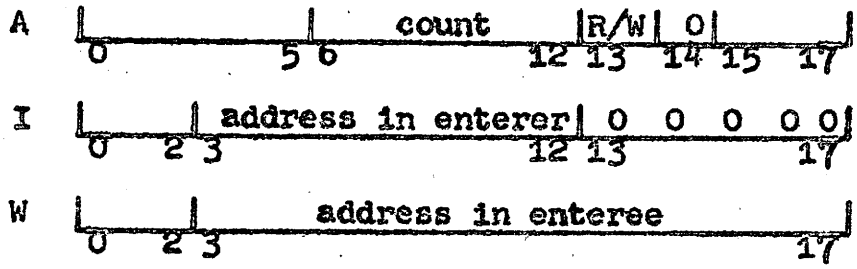
When a non-master entry capability is invoked, a process is created in the sphere which created the entry (the enteree). It starts executing at the entry address which was specified when the entry was created. Its A is initialized to the index of an entered process capability which is created when the entry is invoked. Its I(12-17) contains the transmitted word of the invoked capability. Its I(8-11) contains the variant of the ivk.

A process trying to enter will wait until all of the following are true: 1) Processing is enabled in both the entering and entered spheres. 2) A process is available. The process is taken from the entered sphere's hoard. A program which wants to be certain it can be entered should set its hoard. 3) A capability index in the entered sphere is available for the entered process capability.

An entered process capability is the enteree's "link" to the enterer. There is exactly one copy of an entered process

capability; it may be granted but not shared. The following ivks may be performed on an entered process capability.

If A(14) is zero the operation is a read/write core operation. A(13) is 0 to read the enteree's core, 1 to write. The number of words to transfer/40 is in A(6-12); if this is zero, 10000 words will be transferred. The address in the enterer is in I(3-17) and must be a multiple of 40 words. The addrss in the enteree is in W(3-17). The transfer must not cross a core module boundary in either sphere. The instruction skips if successful.



Other entered process ivks:

code in A — Action

11

The A, I, and W registers of the entering process are read into three consecutive words at the address given in I(3-17).

31

The three consecutive words at the address given in I(3-17) are written onto the A, I, and W registers of the entering process.

51

Restart. The entering process is restarted and the entered process capability is deleted. The entering process will not have its PC advanced or AAL indicator cleared, hence, unless something has changed, it will execute the same instruction again.

71

Return. The entering process is restarted and the entered process capability is deleted. The PC of the entering process is advanced to the next instruction, its AAL indicator is cleared, and, if its ESI indicator is on, an ESI trap will occur. This makes the enter appear to have completed.

111

Cause illegal instruction. The ivk that the entering process executed is treated as a recoverable illegal instruction. The entered process capability is deleted. The process's PC is not advanced, nor is AAL cleared, so that the process will appear to have not yet executed the illegal instruction.

131

Return and skip. Similar to return (71), except that the PC is advanced one more time, making the enter appear to skip.

151

Read process number. The number of the entering process in its sphere (1, 2, 3, ...) is read into A, and the sphere to which it belongs (low 12 bits of sphere capability) is read into I. (Note - Unless processing is suppressed, the process number may not be fixed.) This instruction skips unless the entering process was deleted (e.g. by logout) since it entered.

SUMMARY OF META-INSTRUCTIONS

mta 0	Copy A into W.
mta 1	Copy I into W.
mta 2	Copy W into A.
mta 3	Copy W into I.
mta 4	Cause program trap 10.
mta 5	Cause program trap 11 (ID executes commands).
mta 6	Cause program trap 12.
mta 7	Cause program trap 13 (dsm, normal completion).
mta 100	Enter low priority mode.
mta 104	Read absolute drum fields 0 through 77.
mta 105	Read absolute drum fields 100 through 177.
mta 200	Read illegal instruction return to A.
mta 201	Set illegal instruction return from A.
mta 202	Read illegal memory reference return to A.
mta 203	Set illegal memory reference return from A.
mta 204	Delete capability in A(12-17). No skip.
mta 205	Detach. Core number from A(3-5) or A(15-17). Skip if ok.
mta 206	Read memory bound to A(3-5), attachments to A(12-17). No skip.
mta 207	Set memory bound from A(3-5) or A(15-17). Skip if ok.
mta 300	Create drum field. If A(1)=1, use absolute field in A(5-11). If A(0)=1, read only. Skip if ok.
mta 302	Create sphere. Fault entry address from I. Skip if ok.
mta 303	Create queue. Initial population is -|I|. Skip if ok.

mta 306 Create I/O device. Skip if ok.

A(0-5)	Device	A(6-11) contains
0	microtape	transport no.
1	buttons	console no.
2	clock	
3	reader	0-alpha, 1-bin
4	punch	
5	typewriter	console no.
7	hardware IO	device no.

mta 307 Create entry. Entry address from I. Skip if ok.

mta 400 Read capability in A(12-17). No skip.

mta 401 Exchange capabilities in A(6-11) and A(12-17). No skip.

mta 402 Turn off PRL. No skip.

mta 403 Turn on PRL. No skip.

mta 404 Count items in C-list, return number in A.

mta 405 Copy capability at I(6-11) to I(12-17). Skip if ok.

mta 406 Read process hoard to A. No skip.

mta 407 Set process hoard from A. Skip if ok.

mta 500 Assign and deassign external equipment.

A(0-5)	Operation
0	Assign external levels. For all i, $1 \leq i \leq 7$, external level i is assigned if A(i+10)=1. Skip if ok.
1	Deassign external levels as above. No skip.
2	Assign external register, shared. Skip if ok.
3	Assign external register, private. Skip if ok.
4	Deassign external register. No skip.

mta 501 Disown capability in I(6-11). Return index in A. Skip if ok.

mta 502 Claim capability at index in I(6-11). Index in self is in I(12-17). Skip if ok.