PDP-1 COMPUTER
ELECTRICAL ENGINEERING DEPARTMENT
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS
Ø2139

PDP-35

INSTRUCTION MANUAL

PART 5 -- SUPERVISOR CALLS

This memo is the preliminary version of Instruction Memo part 5B. Section numbers in this memo should be ignored.

LAST UPDATE OCCURRED ON

5 JULY 1973

~~21 January 1973~~

I.7.3 Process Hoard

When a process executes a frk, or an enter occurs (see section on entry capabilities), the supervisor allocates a process. If no process is available, the frk waits until a process becomes available. A program may, if it wishes, insure that a process will be available when it is needed by setting its process hoard. The process hoard is the number of processes which the program is guaranteed to be able to have. It may have more than this number if, as is usually the case, the supervisor can allocate space for them. The hoard is simply the number that are reserved and guaranteed to be available at all times, no matter what other programs do. If the hoard is n and there are m processes currently in existence in a program, the next n-m frk's are guaranteed to succeed immediately. Upon logging in, the hoard is initially 1.

mta 406
Read process hoard to A. Does not skip.

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

II.0 Extended System Features

Two powerful features of the PDP-1 time sharing system are the ability to have multiple processes in a single sphere, and the ability to create other spheres which may have their own processes. Since part 4 of the instruction manual is devoted to the uses of multiple processes in a sphere, the related mta's and ivk's will not be discussed here.

The rest of this memo will be primarily concerned with the use of spheres ; intersphere and intrasphere communications, creation and control of spheres, and error handling capacities.

II.1 Purpose of Spheres

It is sometimes useful for a user to create a sphere ; that is, create separate processes running in an address space independent of the address space of the creator, and with a separate set of capabilities. For example, suppose a user program FOO wanted to run some other program DESTROY which, either maliciously or because of program bugs, might attempt to wreak havoc in FOO. FOO could create an inferior sphere, load DESTROY into it, grant it any necessary capabilities, run the program, and be quite safe from whatever might occur. The program ID is an example of this. The typewriter capability and 4K of core the user has upon logging in (the user's initial sphere) is a sphere owned by the user's ID.

Spheres are also frequently used when the user program wishes to run another program in a "nice" environment. For example, "IIID" is a program designed to enable users to debug programs that run in more than one sphere. This is accomplished by creating two inferior spheres, placing ID into one of them, and the user's program into the other. During execution, ID thinks that it is receiving traps directly from the user's sphere as usual, while actually all traps are being filtered by IIID. Similarly, all typeout from ID is passed through an entry into IIID which first generates a process state display and only then directs ID's output to the console. In addition to controlling the input and output streams for ID, IIID monitors all spheres created by the user such that he may point his ID at any one of them in order to follow processes within the selected sphere. Furthermore, by carefully reading and writing ID's core during sphere switching, IIID permits the user to have breakpoints set in many spheres simultaneously. FORTRAN programs wishing to manipulate microtape datafiles provide another example. The opening, closing, reading, and writing of the datafile has been implemented by running the File System of the appropriate tape in an inferior sphere.

Only a user's initial sphere has the ability to have capabilities with PRL off. See section I.5 on PRL.

II.2.1 Ownership and Receiving Traps

When sphere A creates a sphere B, sphere A receives a master sphere capability to B. Ownership of a master capability by A makes A the superior sphere to B, and B the inferior sphere. There may be at most one master capability to a sphere at a time; there may be none. This implies that other copies of a sphere capability created by a mta 405 (copy capability) or a grant or share sphere ivk (described later) will not be master capabilities. A sphere may be its own superior.

Being a sphere's superior implies that if some process in the inferior sphere encounters difficulties, the superior will be informed. Processes that encounter problems in a sphere having no superior will simply cease executing instructions but will not otherwise affect any remaining process in the sphere. Difficulties are:

0	illegal instruction; irrecoverable or return not enabled
1	lock fault
2	ESI trap
3	I/O function busy trap
4	bpt trap
5	unused
6	illegal memory reference, return not enabled (except for illegal reference to PRL, which is an illegal instruction and therefore code 0).
7	unused
10	mta 4 (hlt)
11	mta 5
12	mta 6
13	mta 7 (dsm)

Informing the superior is done by an entered process as diagrammed below.

```
              sphere B                          sphere A

      \                     \<          \    C-list X    \
      \         entering    \           \_   C-list Y   _\
      \         process     \           \                \
      \       \ A, G, \<    \           \                \
      \       \ I, X, \     \           \                \
      \       \ F, W  \     \           \                \
      \                     \           \                \
      \    <instruction>    \           \                \
      \                     \           \                \
```

When the process executes whatever instruction causes the trap, the following steps occur:

1) the process executing the instruction is "hung"; no further instructions may be executed unless the process is explicitly restarted.

2) A system "pointer" to this process state is placed in the first unused C-list location provided that the superior's run indicator is on, that an unused C-list location exists, and that step 3 can occur. This pointer is called an entered process capability.

3) A new process is created in the superior sphere. This process will have in its AC the index of the entered process capability created in step 2. The I register will contain the octal number Ø to 13 from the above list describing the reason for the trap.

The fault entry address specified when the sphere was created or subjugated is put into G(3-17) and G(Ø-2) is cleared. All other registers of the created process contain zero. The fault entry address indicates where a process created to respond to a "fault" in the inferior should begin executing.

In the diagrams above, C-list index "y" implies the entered process capability added in step 2, and C-list index "x" represents the index of the master sphere capability. Reversing this entire entry mechanism will be discussed later.

A sphere that wishes to guarantee an enter can occur should set its process hoard. Hoards are more fully described in part 4 of the Instruction Manual.

Mta-Trap Mode

A sphere may choose to handle all mta's ≥ 200 executed by an inferior rather than have them handled by the Administrative Routine. To trap such mta's, first the superior must set run status. (See sphere ivk 512.) Then, whenever any process in the inferior attempts to execute mta's ≥ 200, the following occurs:

1 and 2) Identical as for a fault (above).

3) A new process is created in the superior sphere. Its AC will contain the index of the entered process capability created in step 2. I will contain the mta number. The mta number in I is compressed; e.g. mta 306 appears as 36.

The fault address specified when the sphere was subjugated in put in G(3-17) and G(0-2) is cleared. All other registers of the created process contain zero.

II.2.2 Relevant Instructions

Instruction Action

mta 302
Create sphere. Creator receives a master sphere capability. A(12-17) contains the desired index or 0 if first free capability to be used. I(3- 17) should contain the fault entry address. Skip if successful. The sphere will initially have:

```
core 0          (memory bound = 10000)
PRL off         (sphere may not own capabilities)
run indicator off
(the following have not yet been described)
breakpoint variables disabled (777777,0,0)
illegal instruction return disabled (777777)
illegal memory reference return disabled (777777)
no attachments
```

mta 4
This is the hlt instruction. Causes a mta 4 trap to the superior.

mta 5
Causes mta 5 trap to superior. ID will interpret this as a request to perform a command string (see ID memo).

mta 6
cause mta 6 trap.

mta 7
This is the dsm instruction. ID interprets this as a "normal completion".

Sphere ivks.

If A(14) is zero, a read/write sphere operation is specified. The operation greatly resembles the read/write drum field ivks given earlier. A(Ø-5) is ignored. The count of the number of words to be transferred/4Ø is given in A(6-12). If this count is zero, 1ØØØØ words will be transferred. If A(13) is zero, data will be moved from the invoked sphere into the sphere executing the ivk. If A(13) is a one, data will be written onto the invoked sphere from the invoking sphere. The contents of A(15-17) should be = 2, but under the current system this field is ignored. I(3-17) specifies the extended core address in the invoked sphere; since this address must be a multiple of 4Ø words, I(13-17) will be zero. W(3-17) gives the extended core address in the sphere executing the ivk. W(Ø-2) and I(Ø-2) are ignored. The instruction skips if successful. Core locations Ø-77 of a sphere with PRL on may not be read or written.

```
A     \///////////\    count     \W\ Ø\       \
       Ø          5 6          12 13 14 15    17

I     \/////\   sphere address  \ Ø Ø Ø Ø Ø\
       Ø   2 3                12 13          17

W     \/////\          core address           \
       Ø   2 3                                17
```

AC Code	Action
12	Suppress processing in the ivk'd sphere. That sphere's run indicator is turned off.
32	Permit processing. The run indicator is turned on. If a sphere's run indicator is on, processes in that sphere may run and any pending enters will occur.
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 which is not the core Ø of a sphere with PRL on) and the attaching core is not a real core. If the attached core is itself an attachment, the real core the attached core is an attachment to will be used. If the sphere to receive the attachment already has an attachment at the area specified by I(3-5), the previous attachment will be removed. An attachment may be made and maintained whether the run indicator of the invoked sphere is on or off. Skip if successful. Refer to section on attachments.

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. The sphere's run indicator must be off. 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 bp1, bp2, bp3 are read into three consecutive words beginning at the address in I(3-17). Does not skip.

172

Write breakpoint state. Similar to 152. Refer to Breakpoints and ESI. Does not skip.

412

Create process. A new process is created for the sphere, and becomes the highest numbered process. Its process number is returned in A. The run

ANNOUNCING → NEW SPHERE IVK

AC Code	Action
472	Set/clear assignment number. If A(0) = 0, assign the same console number as that of the ivk'er. If A(0) = 1, assign 0 as console number. This governs which sense switches, display lever, and run light are selected by the hardware when the sphere is running. Ordinarily (and initially) user-created spheres are assigned to console 0 (sense switches on bay 11, no display lever or run light).

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 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.

472 Set/clear console assignment number. If A(0)=0, assign the same console # as ivk'er. If A(0)=1, assign console # 0. This determines which sense switch display lever, & run light are used. Ordinarily user-created spheres are assigned to console 0 (sense switches on bay 11).

512

Set/Clear run status. Three status bits to be associated with the ivk'd sphere are read from I(3-5). I(3) = 1 and I(3) =0 respectively enable and disable mta-trap. (See section describing mta-trap). I(4)=1 causes all faults normally trapping to the ivk'd sphere to trap to its superior instead, provided the superior is enabled, has an unused C-list index, and that a process can be created. I(5)=1 prevents this ivk from having any effect if executed in the sphere. The ivk is still legal however, in the sense that it will not cause an illegal instruction trap.

532

Read fault entry address and superior. If the invoked sphere has a superior, the fault entry address for the invoked sphere in that superior is returned in A, and I will indicate the superior sphere (low twelve bits of sphere capability). If the invoked sphere has no superior, then I will contain zero, and A will be unchanged.

552

Subjugate. The sphere in which the ivk was executed becomes the superior of the invoked sphere. I 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. If the sphere executing the ivk is already the invoked sphere's superior, then I must contain the new fault entry address. This instrucion fails if the sphere already has a superior other than the one executing the ivk. Skip if successful.

572

Execute mta. The W register is moved to A, and the meta-instruction whose code was originally in A(Ø-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 ≥ mta 2ØØ may be executed. The ivk is illegal if the meta-instruction is illegal. Skip if meta-instruction would skip.

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 but not 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, of the first free index otherwise. If successful, skip and put the index of the capability created in the receiver in A and a copy of the capability in I. If unsuccessful, then

1) a capability already exists at the index in I(12-17) (or at all indices if I(12-17)=Ø) in which case the interfering capability (or the last capability) is placed in I,

2) no capability exists at the index I(6-11) in which case I is cleared, or

3) the capability is an entered process capability in which case I is cleared.

II.2.3 Processing entered process capabilities

Having been informed that a fault has occurred, some action should be taken. This action may take two forms. Either one may use the powerful sphere ivks and incur the cost of stopping processing in that sphere, or one may choose to use the less powerful entered process ivk's which affect only the process that caused the fault (the enterer). Having discovered why the fault occurred, and taken whatever action is desired, the entering process must be restarted. This can only be done by executing an entered process ivk with any of 51, 71, 111, or 131 in the AC. Execution of any of these four ivk's will cause the following to occur:

1) the entering process is restarted according to the description below,

2) the entered process capability in the superior is deleted,

3) the process executing the ivk will remain running. If this process has no more to do, it should execute a "qit."

An entered process capability may be granted or moved, but not shared or duplicated. Only one copy may exist at a time.

II.2.4 Relevant instructions

Entered process ivks

If A(14) is zero, a read/write core operation is specified. A(Ø-5) is ignored. The number of words to be transferred/4Ø is specified in A(6-12). If a count of zero is found, 1ØØØØ words will be transferred. If A(13) is a zero, data will be read into the sphere executing the ivk. If A(13) is a one, data will written from the sphere executing the ivk. A(15-17) should contain 1 to be compatible with future modifications, but is currently ignored. I(3-17) specifies the address in the enterer and will contain zeros in I(13-17) since this address must be on a 4Ø word boundary. The address of the data in the sphere executing the ivk is given in W(3-17). W(Ø-2) and I(Ø-2) are ignored. Skip if successful. Locations Ø-77 of core Ø of a sphere with PRL on may not be read or written.

```
A     \////////////\    count     \W\ Ø\        \
       Ø          5 6         12 13 14 15   17

I     \/////\ address in enterer\ Ø Ø Ø Ø Ø\
       Ø   2 3                  12 13       17

W     \/////\          core address         \
       Ø   2 3                              17
```

AC Code 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 but it will not have its PC advanced or AAL indicator cleared. Hence, unless the sphere is suppressed, the process will trap again.

71
Return. The entering process is restarted and will have its PC advanced and AAL indicator cleared. If its ESI indicator (execute single instruction) is on, an ESI trap will occur. This makes the enter appear to "complete."

111
Cause recoverable illegal instruction trap. 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. If the sphere does not have its illegal instruction return set, an illegal instruction trap to the superior will occur.

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 process number of the entering process is read into the AC, 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 vary.) This instruction skips unless the entering process was deleted (e.g. by logout or deletion of the sphere) since it entered.

171

Read capability. Reads the entry capability that the entering process invoked into A. Skip unless process has been deleted or unless index of capability invoked no longer exists. Effect is undefined if entry was a fault entry to a sphere's superior.

II.3

Illegal Instruction Return

To avoid some of the time consuming work of letting faults trap to a superior, a program may wish to handle its own errors. For example, suppose a program executes an ivk 16 expecting a microtape capability to be there. If index 16 contains a null capability, then the program would get a recoverable illegal instruction error. In this particular case it might be nice to be able to catch this error, assign the needed microtape, and continue. In another case, a user might wish to catch his attempts to reference an illegal memory location, and either assign needed additional core or perform some error function. These functions are done respectively by setting the illegal instruction return or the illegal memory reference return.

A program may intercept only recoverable illegal instructions. Only the following are unrecoverably illegal:

hlt	(77ØØ74)	dsm	(770077
bpt	(77ØØ44)	mta 5	
opcode ØØ		mta 6	
privileged instructions		illegal iots	

attempts to reference Ø-77 of core Ø when PRL is on.

If a recoverably illegal instruction is executed, the process attempting the execution is modified as follows: W(3-17) contains the address of the illegal instruction, W(Ø-2) are cleared, and G(3-17) is set to the illegal instruction return location. All else, including AAL, is not modified. This may require the first instruction of the error routine to be a nop or rpf.

Instruction	Action
mta 2ØØ	Read illegal instruction return location into A.
mta 2Ø1	Set illegal instruction return from A. Any negative number in A will disable this feature.

Illegal Memory Reference Return

A program may intercept any illegal memory reference. Note that referencing locations Ø-77 of core Ø of a sphere with PRL on is NOT treated as an illegal memory reference, but is an unrecoverably illegal instruction. The method of catching the illegal memory reference is the same as that of intercepting recoverable illegal instructions above.

Instruction	Action
mta 2Ø2	Read illegal memory reference location into A.
mta 2Ø3	Set illegal memory reference return from A. Any negative number in A will disable this feature.

II.4 Attachments

When a sphere is first created, it has a core address space consisting of a single 4K region. In the course of execution of a program, this quantity of allocated core space might be raised or lowered in blocks of 4K words by a mta 207. Memory space acquired in this fashion is said to be owned by the sphere and is referred to as a real core.

It is often found useful to be able to communicate data between two spheres, and there exist several means by which this may be accomplished. Two of the possible solutions are to let both spheres have capabilities to one or more common objects (such as an entry capability or drum field), and to give one sphere a capability to the other sphere such that the first may employ read/write sphere ivks. Both of these cases are frequently used, especially when the data is primarily to be transmitted only one way.

A third alternative is to share a portion of addressable core space between the two spheres. In this sharing, one sphere owns the core space as a real core and the other sphere receives an attachment. An attachment acts exactly like a real core except for two major things. First, having an attachment does not imply any degree of ownership of the associated real core. Attachments will be deleted without notice when the real core is deleted. Second, the memory bound of a sphere is not modified when a core is attached; memory bound refers only to the real core owned by a sphere. However, a mta 206 will indicate the existence of attachments if they are present (see section I.3 for further description and a definition of memory bound).

Consider the following example. Suppose sphere A having an initial memory bound of 10000 octal creates an inferior sphere B, and then attaches B's core 0 as its core 2. B would perceive no change in its address space, but A's address space would look like the following:

```
     legal             legal
     - real  illegal B's core 0 illegal illegal  illegal
   0      10000   20000     30000    40000     50000    60000
```

Thus if a process in A should change A's core location 21567, sphere B would find that location 1567 in its core 0 has been changed. Memory references to an attachment are directed (efficiently) to the attached real core region.

One use of attachments would be two separate spheres that wish to have a variables area in common, without the aggravation of sharing a drum field or creating entry capabilities for communication. Another example would be a sphere which creates an inferior sphere and finds attaching the inferior's core is a simpler method of entering data to the inferior than performing read/write sphere ivks.

Core Ø of a sphere with PRL on may not be attached. Read/write sphere or process ivks must be used instead, although reading or writing the C-list is, of course, illegal.

Attaching a sphere's core requires invoking a capability to the sphere with codes in the AC and I as noted below.

AC Code	Action
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 which is not the core Ø of a sphere with PRL on) and the attaching core is not a real core. If the attached core is itself an attachment, the real core the attached core is an attachment to will be used. If the sphere to receive the attachment already has an attachment at the area specified by I(3-5), the previous attachment will be removed. An attachment may be made and maintained whether the run indicator of the invoked sphere 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.

Instruction	Action
mta 2Ø5	Detach. The attached area to be detached is given in A(3-5), or, if that is zero, A(15-17). The instruction skips unless it is an attempt to detach a real core.

Breakpoints and ESI

Sometimes it is useful to determine some statistics about a program's behavior. These might be such things as how frequently a loop in the program is executed, how many instructions are executed, or how often a particular branch in the program is chosen. This ability is implemented by associating with each sphere a set of three "breakpoint" registers, and associating with each process an ESI (execute single instruction) flag. The ESI flag is F(6) and may be set only by a write process state sphere ivk instruction.

Breakpoint registers allow the equivalent of ID's super proceed (-nP) and its multiple proceeds (nP). The breakpoint registers will be symbolically referred to as bp1, bp2, and bp3. Use of the breakpoint feature would involve approximately the following steps.

1) Since the setting of breakpoint registers and ESI are sphere ivks, it is necessary for the user to have a capability to the sphere in question. In the case where it is desired to measure a program's performance, it will be necessary to put the program in a sphere, grant it the necessary capabilities, etc.

2) A bpt (770044) instruction must be explicitly put into the sphere at whatever location is desired.

3) The breakpoint registers are interpreted as follows:

```
    bp1(0-1)        00   not a superproceed
              10    superproceed, haven't hit bpt yet
              01    hit bpt, proceeding under ESI
              11    proceeded, further bpts illegal
    bp1(2)    ignored
    bp1(3-17)         address of bpt instruction

    bp2              -[number of times to proceed]

    bp3              instruction to replace bpt, if needed
```

The chart below should be consulted for the myriad possibilities available to the user.

4) A write breakpoint state sphere ivk is executed, and then the sphere's run indicator is turned on.

Multiple proceeds over a breakpoint that has been set do not try to distinguish between processes that may hit the breakpoint.

Superproceeds are set up in a similar manner. It is not necessary to set a bpt in the sphere, however, nor is it necessary to specify an address in bp1(3-17) or a replacement instruction in bp3. A write process state sphere ivk must be performed to initially turn on ESI. This will cause the process to trap to the system after each instruction completes, and the number of instructions executed will be - the number specified in bp2. Arithmetic on bp2 is in ones complement.

The breakpoint and ESI features may be disabled by setting bp1 to 777777 and writing the breakpoint state.

The following instructions are the sphere ivks needed to perform the read/write breakpoint state registers.

AC Code	Action
152	Read breakpoint state. The three words of breakpoint state bp1, bp2, bp3 are read into three consecutive words beginning at the address in I(3-17).
172	Write breakpoint state. Similar to 152.
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. The sphere's run indicator must be off. 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.

For clarification of all this material, a few examples are in order. First is a program that will set a breakpoint in an inferior sphere (at capability index 14) which will allow the inferior to hit the set breakpoint three times before trapping. It would be equivalent to the code ID would need to execute the commands

```
       "foobleB
       3P   "   .
```

It assumes processing has already been suppressed on the sphere.

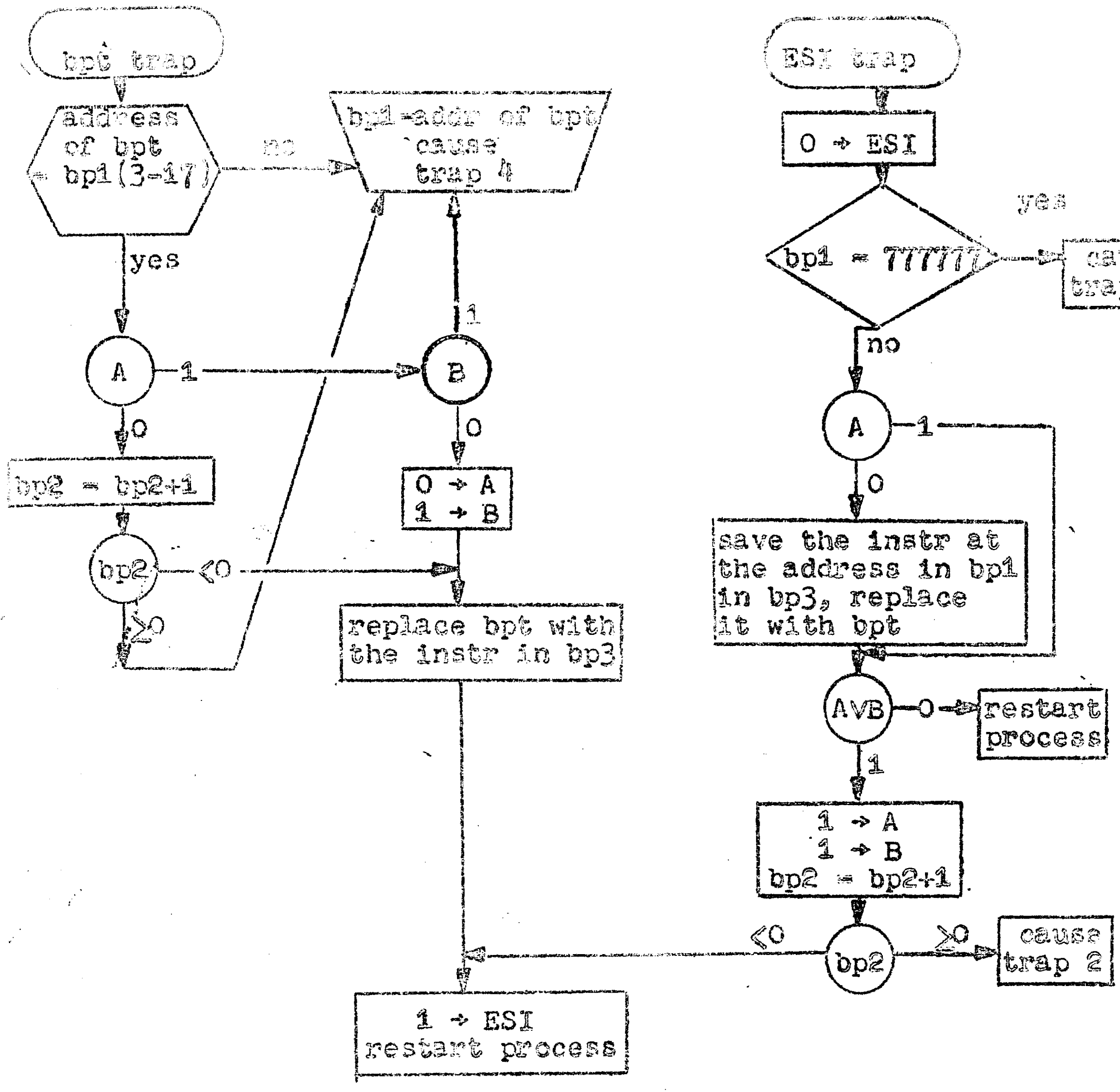
```
       dimension bp(3)
       dimension end(4Ø)          / space for reading in core
       lio (fooble&77774Ø         / 4Ø word block containing fooble
       law end           / adr in this sphere for data
       mta               / put in W register
       law 4Ø            / read sphere, 4Ø words to be read
       ivk 14            / perform read sphere ivk
       hlt               / if read lost
       lac fooble&37 end          / get old contents of fooble
       dac bp 2                   / bp3
       law i -bpt        / equivalent to lac (bpt
       dac fooble&37 end          / insert breakpoint
       law 6Ø            / write sphere, 4Ø words to be written
       ivk 14            / I, W unchanged from previously
       hlt               / write failed
       law i 3           / bp2 arithmetic is ones complement
       dac bp 1                   / bp2
       law fooble        / address of fooble, core Ø assumed.
       dac bp            / bp1(Ø-1) set to ØØ; bp1(3-17) -> fooble
       lio (bp           / address of three words of bpt state
       law 172           / code to write breakpoint state
       ivk 14            / write it
       law 32            / code to enable processing in sphere
       ivk 14            / processing is enabled.
```

At this point, a process or processes in the inferior sphere may hit the bpt three times before a trap to the above program will occur. If any other bpt's are hit, or other illegal instructions, etc., occur, they will produce the normal variety of trap.

The next example program will cause process 1 in the sphere at capability index 14 to execute 1ØØØ instructions and then produce an ESI trap. Any other processes in the sphere will execute normally, without ESI. Once again, it is assumed that the sphere's run indicator is already off, and that a process exists in the sphere that is ready to run. Additionally it is assumed that process 1 is not in a wait.

```
        dimension prcstt(6)     / space for process state
        dimension bs(3) / space for breakpoint state
        lio (1          /process 1 to be referenced
        law prcstt      / address of prcstt
        mta             / put in W
        law 112         / code to read process state
        ivk 14          / get process state
        hlt             / some lossage has occurred
        law 4ØØØ                / ESI bit in Flag word
        adm prcstt 4    / Process's F register. turn on ESI
        law 132         / code for write process state
        ivk 14          / write process state
        hlt             / some lossage
        law i 4         / put A(Ø-1) = 11 and A(Ø-17) ≠ 777777
        dac bs          / bp1 set
        law i 1ØØØ      / number of instructions to be executed
        dac bs 1                / bp2
        lio (bs         / address of breakpoint state words
        law 172         / write breakpoint state.
        ivk 14          / write breakpoint state
        law 32          / code to enable processing in sphere
        ivk 14          / enable processing
```

At this point process 1 in the sphere will be allowed to execute at most 1ØØØ instructions before causing a trap. If the process does not otherwise cause some trap because of executing the program in the sphere, then exactly 1ØØØ instructions later an ESI trap will occur.



What are A and B ?

what are their initial values?

when do they get reset ?

III.1 Communication between spheres; Entry Capabilities

An entry is a user programmed capability. It is a means of communication among any group of spheres. Use of entry capabilities involves many of the same things involved in handling entered processes. This mechanism is described in sections II.2.1 and II.2.3. When creating an entry, one must specify the entry address, analogous to the fault entry address, which specifies where the process created in the entered sphere, analogous to the superior sphere, should begin executing. The entry capability initially acquired is a master capability, and all other granted or shared copies or duplications result in a non-master capability. Non-master capabilities are useful only while the master exists. If the master is deleted, or by granting converted into a non-master, all ivks on the associated non-master capabilities become illegal.

Each entry capability has a six-bit transmitted word. The transmitted word of a master entry capability may be changed but those of a non-master may not. Duplication, granting, and sharing operations copy the transmitted word without modification.

When a non-master capability is ivk'd, the process created in the entered sphere has its registers initialized as for a fault entry trap (see II.2.1), except that I(12-17) contains the transmitted word and I(8-11) contains the variant of the ivk executed by the entering process (process that the entered process capability points at). The transmitted word is useful for distinguishing between several similar non-master entry capabilities without creating a separate master entry capability for each distinguishable non-master desired. For example, suppose sphere A creates an entry, gives B a non-master with transmitted word Ø1 and C a non-master with transmitted word Ø3. By the nature of an entry capability, when a process in either B or C ivk's the non-master entry capability, a new process is created in A that gets started at the entry address. The newly created process in A can quickly tell whether B or C has caused the enter by looking at the transmitted word in I(12-17).

Any of the entered process ivks listed in section II.2.4 may be used with entered process capabilities generated by invoking entry capabilities.

A sphere that wishes to guarantee an enter can occur should set its process hoard.

Instruction Action

mta 3Ø7
Create master entry. A(12-17) contains desired capability index; if Ø the first free index is used. I(3-17) contains desired entry address. Skip if successful.

ivk <master entry>
Change transmitted word of master entry capability. A(12-17) becomes new transmitted word and A(Ø-11) must be clear.

ivk <non-master entry>
Hangs the process executing the ivk, creates an entered process capability pointing to the state of the hung process's registers at the time of the ivk, creates a new process in the sphere containing the master entry capability and initializes registers as described above. Section II.2.1 gives the three conditions necessary for this trap to occur.

C-list index mm in sphere A is the master entry capability. C-list index xx in sphere B is a non-master capability associated with entry mm. The number nn in the ivk nnxx may vary from Ø to 17 (4 bits) and is the variant of the ivk. C-list index cc is the entered process capability created after the ivk nnxx was executed.

```
       SPHERE B                              SPHERE A

   \     c-list     \                    \     c-list     \
   \  \           \ \                    \  \           \ \
   \  \           \ \                    \mm\  master   \ \
   \  \           \ \                    \  \entry cap.\  \
   \  \           \ \                    \  \           \ \
   \xx\  entry    \ \                    \  \           \ \
   \  \capability\  \                    \  \           \ \
   \  \           \ \                    \cc\ent. proc.\  \
   \  \           \ \                    \  \capability\  \
   \  \           \ \                    \  \           \ \
   \                \                    \                \

       ivk nnxx
```

IV. Disowned Objects

An object in a sphere's C-list may be deleted without deleting the corresponding object. If an object exists in the system, some sphere somewhere must own it, and there must be a path from the "root node", the C-list of the timesharing system, to the object in question. If such a link does not exists, or is broken, the system will free the resources represented by that object.

One very well known program, Spacewar, in many of its versions performs the following: creates an inferior sphere, copies all of core into the inferior, and, passes the number of ships playing to the inferior, starts the inferior running, disowns the sphere, then dismisses. At this point the console could be logged out and Spacewar would continue to run, even though the original owner has been deleted from the system. Disowning actually is equivalent to a grant of the specified capability to the Administrative Routine of the system so that in fact the "disowned" object is owned by the system.

Any sphere may request ownership of any disowned object. Absentee computations must be performed using this technique.

Instructions	Action

mta 501

Disown capability. Capability in I(6-11) is disowned. An index is returned in A which should be used to reown the object when desired. Skip if successful and copy of the capability is placed in I. The copy of the capability at the index given in I(6-11) is removed from the C-list.

mta 502

Own capability. Disowned object whose index is in I(6-11) is reowned onto capability I(12-17) (or first free index if I(12-17) is zero). New index of the capability in the sphere executing the mta 502 is returned in A. Copy of the capability is placed in I. Skip if successful. If not successful, the same action is taken as for a mta 405.

integers starting at 20

If a disowned sphere attempts to reown itself, assuming no other sphere has a capability to the disowned sphere in question, then the sphere will be deleted from the system for the reason noted above.

What Reason?!? — it's very "far away"??