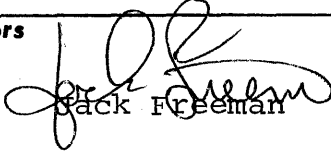


bcc	title	MCALLs on the Model I Sub-Process System	prefix/class-number.revision	
			MISPS/M-7	
	checked Sullivan W. Nampe	authors  Jack Freeman	approval date	revision date
checked Larry L. Barnes	9/11/69			
approved Mel	classification Manual			
			distribution Company Private	pages 59

ABSTRACT and CONTENTS

The attached document describes all the currently implemented MCALLs on the Sub-Process module of the Model I Basic System. It is intended to serve as a system programmer's manual for this part of the system. The portions of document PMTSPT/W-2 which deal with the Sub-Process System are rendered obsolete by the present document.

Table of Contents

	Page
I Introduction	3
II The Sub-Process Table	4
III The Sub-Process Call Stack	9
IV The MCALLs	13
A Creating and Destroying Sub-Processes	16
1 CREATE'SP (MCALL 90)	17
2 DESTROY'SP (MCALL 91)	19
B Passing Control Between Sub-Processes	20
1 SP'CALL (MCALL 111)	21
2 SP'JUMP (MCALL 112)	23
3 SP'TRAP (MCALL 113)	25
4 TRAP'RETURN (MCALL 114)	27
5 SP'BRANCH (MCALL 115)	28
6 SP'RETURN (MCALL 116)	30
7 JUMP'RETURN (MCALL 117)	31
8 MARK'CALL (MCALL 118)	32
C Inspecting and Modifying Sub-Processes	33
1 READ'SPT (MCALL 95)	34
2 READ'SPCS (MCALL 96)	35
3 READ'SPT'FIELD (MCALL 97)	36
4 SET'SPT'FIELD (MCALL 98)	38
5 READ'MAP (MCALL 99)	42
6 SET'MAP (MCALL 100)	43
7 READ'MAP'BYTE (MCALL 101)	45
8 SET'MAP'BYTE (MCALL 102)	46
9 READ'ACCESS'KEY (MCALL 103)	47

	Page
10 COPY 'ACCESS' KEY (MCALL 104)	48
11 COPY 'TAK' (MCALL 105)	49
V Figures	
1 Format of an Entry in SPT	50
2 Format of SB and SCB Fields in SPT	51
3 Format of TM and TCM Fields in SPT	52
4A Normal Format of an SPCS Entry	53
4B Alternative Format of an SPCS Entry	54
VI Appendices	
A SPS MCALLs in Numeric Order	55
B Error Codes Returned by SPS MCALLs	57
C Error Message Numbers Returned by SPS MCALLs	58

I Introduction

The Model I Basic System provides facilities with which a user may include an arbitrarily constituted collection of programs as "sub-processes" of his process, specify relations which are to subsist among the sub-processes, and transfer control from one sub-process to another. The Basic System module which implements these facilities is called the Model I Sub-Process System. Detailed descriptions of the MCALLs through which the user accesses the Sub-Process System are given below, following some explanatory material.

II The Sub-Process Table

The sub-processes of a process are defined by entries in a large table, called the Sub-Process Table (SPT), which occupies a considerable portion of the process' Context Block. SPT can contain as many as eight entries. One or two of these will normally hold definitions of system sub-processes, leaving room for six or seven user-selected programs.

Figure 1 shows the format of an entry in the Sub-Process Table. Of the forty-two words, twenty-four are used to define the sub-process' address space, which may be as large as 128K words. The fields of the entry are given the following significance.

NAME:

This 9-bit field plays two roles. As an identifying mark for the sub-process, it is written into the lock fields of objects, such as ICT, OFT, and PMT entries, which belong to the sub-process. It also functions as a "control lock" on the SPT entry itself, protecting it from modification by other sub-processes which are not properly authorized. NAME always has exactly one bit set--in fact, its value is 2^n , where n is the index of the SPT entry.

CALL MASK:

This is another 9-bit field, whose function is to limit "access" to the sub-process, that is, to prevent it from being called by unauthorized sub-processes. The contents of CM bears no relation to the SPT index or to NAME. It may be any collection of bits.

KEY:

KEY is yet another 9-bit field. Its function is to unlock locks, such as the NAME and CALL MASK of SPT entries and the access and control locks of ICT, OFT, and PMT entries. A KEY fits a lock if the bit-wise AND of it with the lock is non-zero. If a sub-process' KEY fits the lock on some object, the sub-process is authorized to perform certain restricted operations on the object. In particular, if the KEY of a sub-process fits the NAME of an SPT entry, the sub-process is authorized to modify the entry. If its KEY fits the entry's CALL MASK it is authorized to call the sub-process defined by the entry.

FATHER:

When one sub-process creates another sub-process, the SPT index of the creating sub-process is put into the FATHER field of the new sub-process' SPT entry. The restrictions (described below) on subsequent modifications of the FATHER field guarantee that these fields define a tree in the Sub-Process Table. This "FATHER tree" establishes relations of responsibility among the sub-processes. If, for example, a sub-process causes a trap which it is not able to deal with, its FATHER is given a chance to handle it. If it declines, then its FATHER is called, and so on, until the "root" of the FATHER tree is reached. The root will normally be a system sub-process capable of handling any abnormal situations which may arise. The FATHER field of a root is always zero.

UTILITY SUB-PROCESS:

Sub-processes may run in the User Ring of the Model I address space or in the Utility Ring. Since these rings are disjoint and since the CPU provides hardware for communicating between the two rings, it is feasible and efficient to allow one sub-process to reside in the Utility Ring while another occupies the User Ring. It is intended that most programs be run as user-ring sub-processes and that the Utility Ring normally contain a standard system program (the Model I Utility System) with which the user-ring sub-processes can communicate through the hardware UCALL instruction.

The USP field of an SPT entry for a user-ring sub-process contains the SPT index of the sub-process which is to reside in the Utility Ring when the sub-process is active. For convenience, the USP field of a utility ring sub-process' SPT entry always contains the index of the entry itself.

RING:

This field indicates whether the SPT entry describes a user-ring sub-process or a utility-ring sub-process. A value of \emptyset means user-ring, 1 means utility-ring.

ENTRY POINT:

In order to be run as a sub-process a program must contain a specification of its entry points. This is expected to take the form of an array of pointers to function descriptors for the functions that are designed to be called by other sub-processes. The absolute address of the descriptor for this array is kept in the EP field of the sub-process' SPT entry.

The first two pointers in the array are assumed to point to a trap-handling function and an interrupt-handling function, respectively.

ENTRY G-REGISTER:

This field contains an 18-bit number which will be loaded into the CPU's G-register whenever control is transferred to the sub-process by any of the MCALL's described below.

STATUS BITS:

The bits in this field specify the privileges and certain operating characteristics of the sub-process. Details are given in Figure 2.

TRAP MASK:

The bits in TM arm/disarm the various traps which may be generated in the sub-process structure. Figure 3 gives details of this field.

STATUS CONTROL BITS:

This field has the same internal format as the STATUS BITS field (see Figure 2). A sub-process with bits set in SCB is authorized to set and reset the corresponding bits in SB.

TRAP CONTROL MASK:

TCM has the same relation to TM as SCB has to SB. A sub-process can set/reset bits in TM only if it has the corresponding bits set in TCM.

TEMPORARY ACCESS KEY:
USER NUMBER:
PERMANENT ACCESS KEY 1:
PERMANENT ACCESS KEY 2:

These four 48-bit fields are used to hold access keys for accessing objects through the Basic File System. Normally UNO will contain the User Number of the user to which the sub-process belongs and TAK will hold whatever key is being currently used by the sub-process. PAK1 and PAK2 are provided for saving arbitrary access keys.

USPUN:

This 48-bit field is intended to contain the File System's Unique Name for the file on which the sub-process' Utility Sub-Process resides. If the sub-process is a utility-ring sub-process, this field is the utility program's own Unique Name.

READ-ONLY BITS:

These 64 bits specify the read-only status of the 64 pages in the sub-process address space. They are copied, along with the corresponding PMT indices from the MAP, into the process' map when the sub-process is made active.

PMT INDICES (MAP):

This string of 64 8-bit bytes specifies the sub-process' address space in terms of pointers to page names in the Process Memory Table. Only the first 32 bytes of the MAP (and the first 32 READ-ONLY BITS) have significance for utility-ring sub-processes.

III The Sub-Process Call Stack

The Sub-Process System includes MCALLs by which one sub-process may call a function in another sub-process. In order to allow the called function to return control (by another MCALL) to the sub-process which called it, the system must save some information about the state of the caller. This information consists primarily of the SPT index of the caller and the values to which the P, L, and G registers should be set on return.

Figure 4A shows the format of an entry in the table, called the Sub-Process Call Stack (SPCS), in which this sub-process return descriptor is saved. A sub-process acquires an entry on SPCS (at "stack level -1") when it becomes active, that is, when it is called. At this time, the sub-process' SPT index is put in the entry and certain other fields are initialized. When the called sub-process itself calls another sub-process, its P, L, and G registers are saved in the entry (which becomes stack level \emptyset).

The fields of an SPCS entry hold the following information.

NIS:

This flag is set when the sub-process (in its incarnation at the stack level described by this entry) makes itself non-interruptable. Non-interruptability is described in document IWS/W-11 on the "Interrupt and Wake-up System."

NIC:

This flag is set when the SPCS entry is acquired if the immediately preceding entry has either NIS or NIC set. It means that the sub-process being activated is non-interruptable as a result of being called from a non-interruptable sub-process.

SPNO:

The SPT index of the sub-process being called is recorded here at the time of the call.

CAKF:

If this flag is set in an SPCS entry, the called sub-process is allowed to copy the TEMPORARY ACCESS KEY of its caller into its own SPT entry. The calling sub-process specifies at call time the value to be given to CAKF.

CC:

The Basic System code which actually passes control to the sub-process being called may generate any of a number of traps without completing the transfer. In some cases it may be desirable to call another sub-process to handle the trap and then to try to pass control again when the trap-handling function returns. The Basic System could save its own P, L, and G registers in the SPCS entry and just resume execution at the point they define. This would probably lead to disaster, however, since there is no way to insure that the sub-process called to handle the trap will, for example, preserve the central registers. This problem is solved by saving in the stack entry enough information to

allow the entire function which transfers control to the sub-process to be re-executed. This information is saved in the fields EPNO, PAR1, and PAR2 shown in figure 4B. The CC flag is reset when this information is saved and set when the system detects that the call has been successfully completed. (CC stands for "Call Completed.")

PC, LR, GR:

The P-counter and L and G registers of the sub-process are saved here when it calls another sub-process.

NIET:

This 47-bit field holds the real time at which non-interruptability is to expire. It is loaded from the corresponding field of the immediately preceding stack entry at call time. If the sub-process is interruptable, NIET is zero.

EPNO (Figure 4B):

When $CC=\emptyset$, this field holds an index into the called sub-process' sub-process entry point vector. It has no significance when $CC=1$.

PAR1 (Figure 4B):

When $CC=\emptyset$ and $EPNO=\emptyset$ this holds the trap number of the trap which the sub-process is being called on to handle. When $CC=\emptyset$ and $EPNO=1$ this field holds the interrupt number of the interrupt which caused the sub-process to be called. It has no meaning in any other cases.

PAR2 (Figure 4B):

This field only has meaning in the case $CC=\emptyset$, $EPNO=\emptyset$. It then holds the parameter being passed to the called sub-process' trap-handling function.

IV MCALLs on the Sub-Process System

The following pages describe each of the MCALLs on SPS by giving

- (1) the SPL declaration for the function,
- (2) the values returned when the function succeeds,
- (3) the conditions under which the function will fail and the error codes and error message numbers returned in each case, and
- (4) a description of what the function does.

The function descriptions make reference to the following parameters.

<u>Name</u>	<u>Value</u>	<u>Significance</u>
NSPTE	8	The number of elements in the Sub-Process Table (SPT) and therefore the maximum number of sub-processes which may co-exist in a process.
LSPTE	42	The number of words occupied by each SPT element.
NSPCSE	16	The number of entries in the Sub-Process Call Stack array and therefore the maximum depth to which sub-process calls may be stacked.
LSPCSE	5	The number of words in an SPCS entry.
USER'EP	8	The standard location of the sub-process entry point array descriptor for a user-ring sub-process.
USER'EG	∅	The standard setting of the G register to be used when entering a user-ring sub-process.
MAXEPNO	1∅23	The maximum acceptable index into a sub-process' entry point array. This allows for 1024 entry points, the first two of which are reserved for trap and interrupt functions.

NORMAL'SB	-	The value to which the STATUS BITS word of an SPT entry is initialized. The value is DPNIC + DCWSO + DDWSO + DPNOD. (See Figure 2.)
NORMAL'TM	∅	The value to which the TRAP MASK word of an SPT entry is initialized. A sub-process is expected to arm those traps it wishes to handle.
NORMAL'SCB	-	The value to which the STATUS CONTROL BITS word of an SPT entry is initialized. The value is DPNIC + DCWSO + DDWSO + DPNOD. (See Figure 2.)
NORMAL'TCM	-	The value to which the TRAP CONTROL MASK of an SPT entry is initialized. The value is MACC + PRO + PNIM + PNIC + PI + TI + BLL + ILIM + PNOD + DWSO + CWSO + NEP + DMRD + NILE + SPCSO + PMTO. (See Figure 3.)

The descriptions also make reference to the variables CSP and SPCSL. The first of these is the SPT index of the currently active sub-process, which is imagined to have made the call to the function being discussed. The second is the SPCS index of the stack entry for the currently active sub-process.

Many of the functions take an SPT index as one of their arguments. In every such case, -1 is an acceptable value for this argument. With the single exception of CREATE'SP, -1 supplied as an SPT index is taken to mean CSP, the index of the entry for the current sub-process.

Every one of these MCALLs has at least one failure return. The failure returns uniformly return two integer quantities. The first is a character constant consisting of three 8-bit characters which are intended to be suggestive of the nature

of the error condition causing the failure. These error codes and the phrases they are intended to suggest are listed in Appendix B. The second argument is an index into an array of system error messages which provide a more complete description of the error. Appendix C lists the error message numbers which appear in the SPS MCALLs with tentative specifications of the strings to which they will correspond.

A few terms need definition:

An SPT entry is free if its NAME field is \emptyset .

A sub-process is called a utility sub-process if it is designed to run in the Utility Ring.

A sub-process is called a user sub-process if it is designed to run in the User Ring.

The ancestors of a sub-process are all the sub-processes which appear below it in its FATHER tree.

The root of a FATHER tree is the (unique) sub-process in the tree which has no FATHER.

One sub-process controls another if its KEY includes the other's NAME.

One sub-process has access to another if the bit-wise AND of its KEY with the other's CALL MASK is non-zero.

Creating and Destroying Sub-Processes

A sub-process creates another one by making an entry in the Sub-Process Table. The first step in making an SPT entry is to call the function CREATE'SP. This function "acquires" a free SPT entry and initializes it to certain standard values. If these standard values are what are desired, it only remains for the creating sub-process to specify the new sub-process' address space (with SET'MAP). If necessary, the standard sub-process definition may be modified, by the use of SET'SPT'FIELD. It will be noticed that there is no way to create a utility sub-process with the MCALLs described in this document. This limitation will be removed shortly.

A sub-process which controls another may delete it (remove its definition from SPT) by the use of the function DESTROY'SP.

CREATE'SP - Acquire and Initialize an SPT Entry

Declaration:

```
FUNCTION CREATE'SP(SPTX), FRETURN, MONITOR  $\leftarrow$  9 $\emptyset$ ;
```

Success Return:

```
RETURN SPTX;
```

Failure Returns:

- (1) FRETURN('SPI', 138) unless
 - (a) $1 \leq SPTX \leq NSPTE$, or
 - (b) $SPTX = -1$.
- (2) FRETURN('SPF', 1 \emptyset 3) if
 - (a) $SPTX = -1$ and there are no free SPT entries, or
 - (b) $SPTX \neq -1$ and $SPT[SPTX]$ is not free.

Action:

If $SPTX = -1$, SPT is searched for a free entry and the index of the first one found is assigned to SPTX.

$SPT[SPTX]$ is cleared, and then initialized as follows.

- (1) $NAME \leftarrow 2 \uparrow SPTX$
- (2) $CM \leftarrow NAME(CSP)$
- (3) $KEY \leftarrow NAME$
- (4) $FATHER \leftarrow CSP$
- (5) $USP \leftarrow USP(CSP)$
- (6) $RING \leftarrow \emptyset$
- (7) $EP \leftarrow USER'EP$
- (8) $EG \leftarrow USER'EG$
- (9) $SB \leftarrow NORMAL'SB \text{ AND } (SCB(CSP) \text{ AND } SB(CSP))$
- (1 \emptyset) $TM \leftarrow NORMAL'TM \text{ AND } (TCM(CSP) \text{ AND } TM(CSP))$

CREATE 'SP (continued)

- (11) SCB ← NORMAL'SCB AND SCB(CSP)
- (12) TCM ← NORMAL'TCM AND TCM(CSP)
- (13) UNO ← UNO(CSP)
- (14) USPUN ← USPUN(CSP)

SPT[SPTX] is incorporated into the sub-process control structure by merging its NAME into the KEYS of all its ancestors.

SPTX is returned as the value of this MCALL.

DESTROY'SP - Delete the Contents of an SPT Entry

Declaration:

```
FUNCTION DESTROY'SP(SPTX), FRETURN, MONITOR ← 91;
```

Success Return:

RETURN, or

SP'RETURN in the case SPTX = CSP.

Failure Returns:

- (1) FRETURN('CSE', 162) if SPTX = CSP and the Sub-Process Call Stack is empty.
- (2) FRETURN('SPI', 138) unless $1 \leq SPTX \leq NSPTE$.
- (3) FRETURN('SPC', 160) unless SPT[SPTX] is controlled by CSP.
- (4) FRETURN('SPS', 161) if SPT[SPTX] appears on the Sub-Process Call Stack at any level except the very top.

Action:

If SPTX = -1, it is taken to refer to CSP.

NAME(SPTX) is removed from all CM, KEY, and USP fields of SPT entries and from all Access Locks and Control Locks in ICT, OFT, and PMT. Any entries in ICT, OFT, and PMT whose Control Locks become zero as a result of the removal of NAME(SPTX) are DELETED.

Any SPT entry which has SPTX as its FATHER is given FATHER(SPTX) instead.

SPT[SPTX] is cleared and the MCALL returns, either in the normal manner, or through SP'RETURN, in the special case where SPTX = CSP.

Passing Control Between Sub-Processes

This section describes the MCALLs by which one sub-process may pass control to another. The general procedure is for the current sub-process to call the Basic System and for the Basic System to then call a specially prepared function (designated as an SP'ENTRY) in the other sub-process. The MCALLs SP'CALL, SP'JUMP, and SP'BRANCH allow the calling sub-process to specify the sub-process to be called and the point at which it is to be entered. The sub-process called as a result of SP'TRAP or TRAP'RETURN is determined by the calling sub-process' FATHER tree and the TRAP MASKs therein and is not under control of the caller. There is one additional mechanism, the inter-process interrupt system, which causes control to be passed to a new sub-process. This mechanism is described in document IWS/W-11.

With the exception of SP'TRAP and its variant, TRAP'RETURN, the functions about to be described do not allow for the passing of arguments from one sub-process to another. A later version of the Sub-Process System will have this deficiency removed.

SP'CALL - Call a Sub-Process

Declaration:

```
FUNCTION SP'CALL(SPTX, ENTRYNO, FLAG), FRETURN,  
MONITOR ← 111;
```

Success Return:

This MCALL does not return to its caller if it succeeds. Control will be returned to the caller when the called sub-process returns (with SP'RETURN).

Failure Returns:

- (1) FRETURN('SPI', 138) unless $1 \leq SPTX \leq NSPTE$.
- (2) FRETURN('EPN', 173) unless $2 \leq ENTRYNO \leq MAXEPNO$.
- (3) FRETURN('SPA', 140) unless CSP either controls SPT[SPTX] or has access to it.
- (4) FRETURN('RCS', 141) unless there is sufficient room on the Sub-Process Call Stack to record the call of SPTX and a subsequent series of calls on all the sub-processes which are ancestors of it. This is to insure that any software trap generated by SPTX can be handled in an orderly manner.
- (5) FRETURN('ARG', 172) unless FLAG is \emptyset or 1.

Action:

This MCALL transfers control to the sub-process defined by SPT[SPTX] by calling the ENTRYNOth function in that sub-process' SP'ENTRY transfer vector. If SPTX = -1, the sub-process called is the currently active one, CSP. No arguments are passed, but the central registers are preserved across the call. The final

SP'CALL (continued)

argument, FLAG, determines whether the called sub-process will be allowed to execute the special function COPY'TAK. If FLAG = 1 execution will be allowed; if it is \emptyset , execution will be illegal.

The P, L, and G registers of the calling sub-process are saved so that execution can be continued when the called sub-process returns (with SP'RETURN or an equivalent).

If the calling sub-process is non-interruptable with a non-interruptability expiration time, NIET, the called sub-process is made non-interruptable with the same value for NIET.

SP'JUMP - Jump to a Sub-Process

Declaration:

```
FUNCTION SP'JUMP(SPTX, ENTRYNO, FLAG), FRETURN,  
                MONITOR ← 112;
```

Success Return:

This function does not return to its caller if it succeeds.

Failure Returns:

- (1) FRETURN('SPI', 138) unless $1 \leq SPTX \leq NSPTE$.
- (2) FRETURN('EPN', 173) unless $2 \leq ENTRYNO \leq MAXEPNO$.
- (3) FRETURN('SPA', 140) unless CSP either controls SPT[SPTX] or has access to it.
- (4) FRETURN('RCS', 141) unless there are as many unused Sub-Process Call Stack entries as there are ancestors of SPTX.
- (5) FRETURN('ARG', 172) unless FLAG is \emptyset or 1.

Action:

Like SP'CALL, SP'JUMP transfers control to sub-process SPTX by calling the ENTRYNOth function in its array of SP'ENTRY functions. As usual, SPTX = -1 is taken to mean the current sub-process, CSP. The central registers are passed unchanged from the calling sub-process to the one being called, but no other arguments are transmitted.

SP'JUMP differs from SP'CALL in that no provision is made for the called sub-process to return to its caller. In fact, the calling sub-process completely

SP'JUMP (continued)

disappears from the Sub-Process Call Stack. It will look to the called sub-process as if it were called by the caller of CSP instead of CSP itself.

The called sub-process will be allowed to execute COPY'TAK iff the calling sub-process was authorized to do so and the value of FLAG is 1.

If the caller of the calling sub-process is non-interruptable with non-interruptability expiration time NIET, the called sub-process is made non-interruptable until the same time.

SP'TRAP - Generate a Soft-ware Trap

Declaration:

```
FUNCTION SP'TRAP(TRAPNO, PARAMETER, FLAG), FRETURN,  
MONITOR ← 113;
```

Success Return:

This function doesn't return to its caller if it succeeds. Control will return to the caller when the sub-process which handles the trap executes an SP'RETURN.

Failure Returns:

- (1) FRETURN('TNO', 139) unless $\emptyset \leq \text{TRAPNO} \leq 23$.
- (2) FRETURN('ARG', 172) unless FLAG is \emptyset or 1.

Action:

Starting with sub-process CSP and following its FATHER chain in SPT, we search for an SPT entry with bit TRAPNO set in its TRAP MASK (TM). Let SPTX be the SPT index of the first such entry, or the root of CSP's FATHER tree if there are none.

Control is transferred to sub-process SPTX by calling its SP'TRAP'ENTRY function, which is the \emptyset^{th} entry in its array of SP'ENTRY functions. TRAPNO and PARAMETER are passed by the call and the central registers are transmitted unchanged.

The P, L, and G registers of the calling sub-process are saved, so that execution can be resumed when the trap-handling function returns (with SP'RETURN).

If FLAG = 1, the sub-process which fields the trap

SP'TRAP (continued)

will be authorized to execute COPY'TAK. If FLAG = \emptyset , it will not.

The algorithm for finding the sub-process to which the trap is to be sent is in reality slightly more complicated than what was described above. In searching down the FATHER tree, we ignore SPT entries whose distance from the root of the tree is greater than the number of free Sub-Process Call Stack entries. In particular, this means that if there is only one free SPCS entry, we always call the root of the tree.

If the sub-process which generates the trap is non-interruptable with non-interruptability expiration time NIET, the sub-process which fields the trap is made non-interruptable until the same time.

When the Monitor wants to reflect a ring-independent hardware trap to a sub-process, it uses SP'TRAP. Primarily for this reason, bit TRAPNO in the TRAP MASK of the called sub-process is reset before the call is made.

TRAP'RETURN - Pop up the Sub-Process Call Stack and Generate
a Software Trap

Declaration:

```
FUNCTION TRAP'RETURN(TRAPNO, PARAMETER, FLAG), FRETURN,  
MONITOR ← 114;
```

Success Return:

This MCALL doesn't return to its caller if it succeeds.

Failure Returns:

- (1) FRETURN('TNO', 139) unless $\emptyset \leq \text{TRAPNO} \leq 23$.
- (2) FRETURN('ARG', 172) unless FLAG is \emptyset or 1.
- (3) FRETURN('CSE', 152) if the Sub-Process Call Stack
is empty.

Action:

This MCALL is provided to allow a sub-process to convert a call on itself into a software trap. The system deletes all record of the call and proceeds as if the sub-process which made the call had instead executed an SP'TRAP. See the description of SP'TRAP for the details of the action taken. There is one small anomaly. The sub-process which gets called to handle the trap is given authority to use COPY'TAK iff the current sub-process has that privilege and FLAG is 1.

SP'BRANCH - Branch into a Sub-Process

Declaration:

```
FUNCTION SP'BRANCH(SPTX, PC, LREG, GREG, FLAG), FRETURN,  
MONITOR ← 115;
```

Success Return:

This function doesn't return to its caller if it succeeds, but the sub-process into which it branches may return to the calling sub-process by executing an SP'RETURN.

Failure Returns:

- (1) FRETURN('SPI', 138) unless $1 \leq SPTX \leq NSPTE$.
- (2) FRETURN('SPC', 142) unless CSP controls SPT[SPTX].
- (3) FRETURN('RCS', 141) unless there is enough room on SPCS to record calls to SPTX and all its ancestors.
- (4) FRETURN('RNG', 173) unless PC, LREG, and GREG are addresses in the ring specified by the RING field of SPT[SPTX].
- (5) FRETURN('ARG', 172) unless FLAG is \emptyset or 1.

Action:

Control passes to sub-process SPTX at location PC. The L and G registers are set from LREG and GREG. The central registers are passed unchanged. As usual, SPTX = -1 is taken to refer to the current sub-process, CSP.

With the one difference that the new sub-process is simply entered at a specified location instead of being called through one of its SP'ENTRY functions, this MCALL is identical with SP'CALL.

SP'BRANCH (continued)

Whether the "called" sub-process will be allowed to execute COPY'TAK is determined by the value of the final argument, FLAG. A value of 1 allows execution, a value of \emptyset forbids it.

The calling sub-process' P, L, and G registers are saved so that the called sub-process may return control to it by executing an SP'RETURN.

The non-interruptability status of the calling sub-process is carried over to the new sub-process. That is, if the calling sub-process is non-interruptable until a certain time, NIET, the new sub-process is made non-interruptable until the same time.

SP'RETURN - Return to Calling Sub-Process

Declaration:

```
FUNCTION SP'RETURN( ), FRETURN, MONITOR ← 116;
```

Success Return:

This MCALL doesn't return to its caller if it succeeds.

Failure Returns:

(1) FRETURN('CSE', 152) if the Sub-Process Call Stack is empty.

Action:

A sub-process which has been called as the result of SP'CALL, SP'JUMP, SP'TRAP, TRAP'RETURN, or SP'BRANCH can return to the sub-process which called it by executing SP'RETURN. Control is returned to the old sub-process with P, L, and G restored to the values which were saved when the current sub-process was called. The central registers are preserved over the return.

JUMP'RETURN - Return to a Specified Level in the Sub-Process
Call Stack

Declaration:

```
FUNCTION JUMP'RETURN(STKL), FRETURN, MONITOR < 117;
```

Success Return:

None, unless STKL = -1. This rather strange case is
discussed below.

Failure Returns:

- (1) FRETURN('CSL', 153) unless $-1 \leq \text{STKL} \leq \text{SPCSL} - 1$.
- (2) FRETURN('SPC', 154) unless the calling sub-process,
CSP, controls all the sub-processes which are
entered in the Sub-Process Call Stack at levels -1
through STKL.

Action:

This function pops the Sub-Process Call Stack up to
level STKL and returns control to the sub-process
whose SP'RETURN descriptor (P, L, and G) was stacked
at that level. The central registers are preserved by
this function. If STKL = -1, the function just returns
to its caller. Note that the function can fail even
in this case if the calling sub-process doesn't control
its own SPT entry.

MARK'CALL - Record a Function Call on the Sub-Process Call
Stack

Declaration:

```
FUNCTION MARK'CALL(PC, LREG, GREG), FRETURN,  
MONITOR ← 118;
```

Success Return:

```
RETURN;
```

Failure Returns:

- (1) FRETURN('RCS', 141) unless there is at least one more free Sub-Process Call Stack entry than there are ancestors of CSP.
- (2) FRETURN('RNG', 155) unless PC, LREG, and GREG are all addresses in a ring accessible from the ring from which the MCALL was made.

Action:

This function is provided primarily to allow utility-ring functions which have been entered by a UCALL to record the call on the Sub-Process Call Stack as if they had been entered with SP'CALL.

Inspecting and Modifying Sub-Processes

This final section describes the MCALLs which are used to read SPT and SPCS entries and to read and modify the various fields of an SPT entry. In general, a sub-process may modify an SPT entry iff it controls it. There are no restrictions placed on reading entries or parts of them.

READ'SPT - Read an SPT Entry

Declaration:

```
FUNCTION READ'SPT(SPTX, Array SPTE, Scalar NW), FRETURN,  
MONITOR ← 95;
```

Success Return:

```
RETURN;
```

Failure Returns:

(1) FRETURN('SPI', 138) unless $1 \leq \text{SPTX} \leq \text{NSPTE}$.

Action:

If SPTX = -1, CSP is used in its place.

The first M words of SPT[SPTX] are copied into the caller's array, SPTE, where M is the minimum of LSPTE and NW.

READ'SPCS - Read an SPCS Entry

Declaration:

```
FUNCTION READ'SPCS(STKL, Array SPCSE), FRETURN,  
MONITOR < 96;
```

Success Return:

```
RETURN;
```

Failure Returns:

(1) FRETURN('CSL', 153) unless $-1 \leq \text{STKL} \leq \text{SPCSL} - 1$.

Action:

STKL is interpreted as a stack level number, not as an SPCS index. Stack level -1 refers to the SPCS entry for the current sub-process, stack level \emptyset to the entry for the sub-process which called it, stack level 1 to that for its caller, and so on. The LSPCSE words of the SPCS entry selected by STKL are copied into the caller's array, SPCSE. The CC field of the entry returned determines which of figures 4A and 4B should be used to interpret it. CC = \emptyset refers to figure 4B, CC = 1 to figure 4A.

READ 'SPT' FIELD - Read Selected Field of an SPT Entry

Declaration:

FUNCTION READ 'SPT' FIELD(SPTX, FLDNO), FRETURN, MONITOR ← 97;

Success Return:

RETURN FLD; where FLD is the contents of the field specified by SPTX and FLDNO.

Failure Returns:

- (1) FRETURN('SPI', 138) unless $1 \leq \text{SPTX} \leq \text{NSPTE}$.
- (2) FRETURN('ARG', 165) if FLDNO is not an acceptable field specification (see below).

Action:

SPTX is used to select an SPT entry. If SPTX = -1, the entry selected is the one which describes the currently active sub-process, CSP.

FLDNO selects one of 12 fields within SPT[SPTX], in accordance with the following table.

<u>FLDNO</u>	<u>Field Selected</u>
∅ or 'NAM'	NAME
1 or 'CM'	CALL MASK (CM)
2 or 'KEY'	KEY
3 or 'FTH'	FATHER
4 or 'USP'	UTILITY SUB-PROCESS (USP)
5 or 'RNG'	RING
6 or 'EP'	ENTRY POINT (EP)
7 or 'EG'	ENTRY G-REGISTER (EG)
8 or 'SB'	STATUS BITS (SB)
9 or 'TM'	TRAP MASK (TM)

READ 'SPT' FIELD (continued)

10 or 'SCB' STATUS CONTROL BITS (SCB)

11 or 'TCM' TRAP CONTROL MASK (TCM)

READ 'SPT' FIELD returns the contents of the selected field as its value.

SET'SPT'FIELD - Set the Value of a Specified Field of an SPT
Entry

Declaration:

```
FUNCTION SET'SPT'FIELD(SPTX,FLDNO,DATA), FRETURN,  
MONITOR ← 98;
```

Success Return:

```
RETURN;
```

Failure Returns:

- (1) FRETURN('SPI', 138) unless $1 \leq \text{SPTX} \leq \text{NSPTE}$.
- (2) FRETURN('SPC', 143) unless SPT[SPTX] is controlled by the current sub-process, CSP.
- (3) FRETURN('ARG', 167) if FLDNO is not an acceptable field specification. See the description of READ'SPT'FIELD for a list of acceptable values of FLDNO and the SPT fields they select.
- (4) In addition to the three general errors just described, there are others having to do with the validity of DATA as a new value for the field selected by FLDNO. We list these for each of the 12 fields which FLDNO may select.
 - (a) NAME -
 - (1) FRETURN('MNM', 168) unless DATA is identical with the current NAME field of SPT[SPTX].
 - (b) CALL MASK (CM) -

Any value of DATA is acceptable.
 - (c) KEY -
 - (1) FRETURN('SPC', 148) if DATA contains any bits which are in neither the KEY of SPTX nor the

SET 'SPT' FIELD (continued)

KEY of CSP (i.e., if CSP is trying to give SPTX control of SPT entries which it does not itself control).

(d) FATHER -

- (1) FRETURN('SPI', 138) unless $1 \leq \text{DATA} \leq \text{NSPTE}$.
- (2) FRETURN('SPC', 149) unless SPT[DATA] is controlled by CSP.
- (3) FRETURN('FHL', 150) if
 - (a) DATA = SPTX, or
 - (b) SPTX is an ancestor of DATA.
- (4) FRETURN('RCS', 151) if SPTX appears on the Sub-Process Call Stack at a level, CSL, such that the number of ancestors of DATA is greater than NSPCSE - (CSL + 3).

(e) UTILITY SUB-PROCESS (USP) -

- (1) FRETURN('MUS', 170) unless DATA is identical with the current USP field of SPT[SPTX].

(f) RING -

- (1) FRETURN('MRG', 171) unless DATA is identical with the current RING field of SPT[SPTX].

(g) ENTRY POINT (EP) -

- (1) FRETURN('RNG', 157) unless DATA is an address in the ring specified by the RING field of SPT[SPTX].

SET'SPT'FIELD (continued)

(h) ENTRY G-REGISTER (EG) -

- (1) FRETURN('RNG', 157) unless DATA is an address in the ring specified by the RING field of SPT[SPTX].

(i) STATUS BITS (SB) -

- (1) FRETURN('MSB', 144) if the exclusive or of DATA with the SB field of SPT[SPTX] is not a subset of the SCB field of SPT[CSP]. This means that CSP cannot set or reset a status bit which it doesn't control.

(j) TRAP MASK (TM) -

- (1) FRETURN('MTM', 147) if the exclusive or of DATA with the TM field of SPT[SPTX] is not a subset of the TCM field of SPT[CSP]. This means that CSP cannot set or reset a trap bit which it doesn't control.

(k) STATUS CONTROL BITS (SCB) -

- (1) FRETURN('MSC', 145) if the exclusive or of DATA with the SCB field of SPT[SPTX] is not a subset of the SCB field of SPT[CSP]. This prevents CSP from modifying status control bits it doesn't control.

(l) TRAP CONTROL MASK (TCM) -

- (1) FRETURN('MTC', 146) if the exclusive or of DATA with the TCM field of SPT[SPTX] is not a

SET'SPT'FIELD (continued)

subset of the TCM field of SPT[CSP]. The effect of this is to prohibit CSP modifying trap control bits it doesn't control.

Action:

SPTX is used to select an entry in SPT. As with most of the MCALLs which take an SPT index as an argument, SPTX = -1 is taken to refer to the SPT entry which defines the currently active sub-process, CSP. The second argument, FLDNO, specifies a field within SPT[SPTX] in exactly the same way as is detailed in the description of READ'SPT'FIELD. The third argument, DATA, is copied into the selected field of SPT[SPTX], if the relevant validity checks succeed.

READ'MAP - Read the Map of an SPT Entry

Declaration:

```
FUNCTION READ'MAP(SPTX, String MAP), FRETURN, MONITOR ← 99;
```

Success Return:

```
RETURN;
```

Failure Returns:

(1) FRETURN('SPI', 138) unless $1 \leq \text{SPTX} \leq \text{NSPTE}$.

Action:

If SPTX = -1, this MCALL reads the MAP of CSP.

Let ML = 32 if SPTX is a utility sub-process or 64 if it is a user sub-process, and let M be the minimum of ML and LENGTH(MAP). This function writes the first M bytes of MAP from the first M PMT indices in SPT[SPTX] and its first M READ ONLY BITS. Twelve bit quantities are written into each byte. The last 8 bits are a PMT index and the first bit a READ ONLY BIT.

SET'MAP - Set the Map of an SPT Entry

Declaration:

```
FUNCTION SET'MAP(SPTX, String MAP), FRETURN, MONITOR ← 100;
```

Success Return:

```
RETURN;
```

Failure Returns:

- (1) FRETURN('SPI', 138) unless $1 \leq \text{SPTX} \leq \text{NSPTE}$.
- (2) FRETURN('SPC', 137) unless SPT[SPTX] is controlled by CSP.
- (3) FRETURN('SPM', 100) unless
 - (a) BYTESIZE(MAP) is 12 or 24, and
 - (b) $0 \leq \text{LENGTH}(\text{MAP}) \leq \text{LM}$, where LM = 32 if SPTX is a utility sub-process or 64 if it is a user sub-process.
- (4) FRETURN('SPM', 101) unless the PMT index field, PMTX, of every byte in MAP satisfies at least one of the following conditions.
 - (a) PMTX = 0.
 - (b) PMTX is identical with the PMT index currently in the corresponding byte of MAP(SPTX).
 - (c) $1 \leq \text{PMTX} \leq \text{NPMTE}$ and CSP has access to PMT[PMTX].

Action:

If SPTX = -1, the map referred to is that of CSP.

The bytes of MAP are used to set the first LENGTH(MAP) PMT indices of SPT[SPTX] and the corresponding READ ONLY BITS. The remaining PMT indices and READ ONLY BITS are cleared. Each byte of MAP is expected

SET'MAP (continued)

to contain a PMT index, PMTX, and a read only bit, RO.

If BYTESIZE(MAP) is 12, RO is taken from bit 0 of the bytes and PMTX is taken from bits 4-11. If BYTESIZE(MAP) is 24, RO is taken from bit 12 and PMTX from bits 16-23.

If SPTX is the current sub-process in either the User or Utility ring, the Process Map is updated to correspond to the new MAP(SPTX) and the Physical Map is cleared.

READ 'MAP' BYTE - Read a Byte in the Map of an SPT Entry

Declaration:

```
FUNCTION READ 'MAP' BYTE(SPTX, BYTENO), FRETURN, MONITOR ← 101;
```

Success Return:

```
RETURN BYTE; where BYTE contains in bits 16-23 the BYTENOth  
PMT index from the MAP of SPT[SPTX], and in  
bit 12 the corresponding READ ONLY BIT.
```

Failure Returns:

- (1) FRETURN('SPI', 138) unless $1 \leq SPTX \leq NSPTE$.
- (2) FRETURN('MBN', 163) unless
 $\emptyset \leq BYTENO \leq LM$, where $LM = 31$ if SPTX is a utility
sub-process or 63 if it is a user sub-process.

Action:

SPTX is used to select an entry in SPT (-1 selects CSP's entry) and BYTENO to choose a PMT index from the entry's map. The value of the MCALL is a word with this PMT index in bits 16-23 and the corresponding READ ONLY BIT in bit 12.

SET'MAP'BYTE - Set a Byte in the Map of an SPT Entry

Declaration:

```
FUNCTION SET'MAP'BYTE(SPTX, BYTENO, BYTE), FRETURN,  
MONITOR ← 102;
```

Success Return:

```
RETURN;
```

Failure Returns:

- (1) FRETURN('SPI', 138) unless $1 \leq \text{SPTX} \leq \text{NSPTE}$.
- (2) FRETURN('SPC', 137) unless SPT[SPTX] is controlled by CSP.
- (3) FRETURN('MBN', 163) unless $\emptyset \leq \text{BYTENO} \leq \text{LM}$, where LM = 31 if SPTX is a utility sub-process or 63 if it is a user sub-process.
- (4) FRETURN('SPM', 101) unless the PMT index, PMTX, in BYTE satisfies at least one of the following conditions
 - (a) PMTX = \emptyset .
 - (b) PMTX is identical with the PMT index currently in the byte of MAP(SPTX) selected by BYTENO.
 - (c) $1 \leq \text{PMTX} \leq \text{NPMTE}$ and CSP has access to PMT[PMTX].

Action:

If SPTX = -1, it is taken to refer to CSP.

The byte of MAP(SPTX) selected by BYTENO is set from bits 16-23 of BYTE and the corresponding READ ONLY BIT is set from bit 12.

If SPTX is the current sub-process in either the User or Utility ring, the Process Map is updated to correspond to the new MAP(SPTX) and the Physical Map is cleared.

READ 'ACCESS'KEY - Read an Access Key from an SPT Entry

Declaration:

```
LONG FUNCTION READ 'ACCESS'KEY(SPTX, KEYNO), FRETURN,  
MONITOR < 1Ø3;
```

Success Return:

RETURN AKY; where AKY is the value of the access key selected by SPTX and KEYNO.

Failure Returns:

- (1) FRETURN('SPI', 138) unless $1 \leq SPTX \leq NSPTE$;
- (2) FRETURN('ARG', 166) if KEYNO is not an acceptable Access Key specification (see below).

Action:

SPTX is used as an index to select an entry in SPT. SPTX = -1 selects the entry corresponding to the current sub-process. KEYNO selects one of the four access keys in SPT[SPTX] according to the following.

KEYNO	Access Key Selected
Ø or 'TAK'	TEMPORARY ACCESS KEY (TAK)
1 or 'UNO'	USER NUMBER (UNO)
2 or 'PK1'	PERMANENT ACCESS KEY 1 (PAK1)
3 or 'PK2'	PERMANENT ACCESS KEY 2 (PAK2)

The (48-bit) access key is returned as the value of the MCALL.

COPY'ACCESS'KEY - Copy an Access Key from one SPT Entry to
Another

Declaration:

```
FUNCTION COPY'ACCESS'KEY(SPTX1, SPTX2, KEY1, KEY2),  
                        FRETURN, MONITOR ← 104;
```

Success Return:

```
RETURN;
```

Failure Returns:

- (1) FRETURN('SPI', 138) unless SPTX1 and SPTX2 are in the interval [1, NSPTE].
- (2) FRETURN('SPC', 143) unless the calling sub-process, CSP, controls both SPT[SPTX1] and SPT[SPTX2].
- (3) FRETURN('ARG', 166) unless KEY1 and KEY2 are acceptable access key selectors. See the description of READ'ACCESS'KEY for an explanation of how these arguments are interpreted.

Action:

This MCALL sets the access key selected by SPTX2 and KEY2 from the one selected by SPTX1 and KEY1. SPTX1 and SPTX2 are used as SPT indices, with the usual convention that when -1 is offered as an SPT index, it refers to the SPT entry for the current sub-process. KEY1 and KEY2 select one of the four access keys in SPT[SPTX1] and SPT[SPTX2] in exactly the same way that KEYNO selects an access key in the function READ'ACCESS'KEY.

COPY'TAK - Copy TEMPORARY ACCESS KEY

Declaration:

```
FUNCTION COPY'TAK( ), FRETURN, MONITOR < 105;
```

Success Return:

```
RETURN;
```

Failure Returns:

(1) FRETURN('CNA', 174) unless the CAKF flag is set in the SPCS entry at stack level -1.

Action:

This MCALL copies the TEMPORARY ACCESS KEY of the sub-process which called the current sub-process into the TEMPORARY ACCESS KEY field of the current sub-process.

Format of an Entry in SPT

0	0 23	NAME	11 12 14 15	CALL MASK (CM)	23
1	0 23	KEY	11 12 15 16	FATHER USP	23
2	0 12 56	RING	ENTRY POINT (EP)		23
3	0 56	ENTRY G-REGISTER (EG)			23
4	0	STATUS BITS (SB)			23
5	0	TRAP MASK (TM)			23
6	0	STATUS CONTROL BITS (SCB)			23
7	0	TRAP CONTROL MASK (TCM)			23
8	0	TEMPORARY ACCESS KEY (TAK)			23
9	0				23
10	0	USER NUMBER (UNO)			23
11	0				23
12	0	PERMANENT ACCESS KEY 1 (PAK1)			23
13	0				23
14	0	PERMANENT ACCESS KEY 2 (PAK2)			23
15	0				23
16	0	UNIQUE NAME OF UTILITY SUB-PROCESS (USPUN)			23
17	0				23
18	0	READ ONLY BITS			23
19	0				23
20	0	PMT INDEX 0			23
21	0	PMT INDEX 1	PMT INDEX 2	PMT INDEX 3	23
41	0	PMT INDEX 61	PMT INDEX 62	PMT INDEX 63	23

Figure 1

Format of SB and SCB Fields in SPT

0	1	2	3	4	5	6	7	8	9	10	11	12	23
	P	S		9	D	D		D	D	D			
	R	Y		4	P	P	W	C	D	P	M	S	
	P	S		0	N	N	F	W	W	N	S	D	
				M	I	I	I	S	S	O	P		
				C	M		O	O	D				

- PRP - Sub-process is a proprietary program
- SYS - Sub-process has system privileges
- 940M - Sub-process runs in 940 Mode
- DPNIC - The system is to automatically put referenced pages into the Core Working Set
- DPNIM - The system is to create new pages when the sub-process refers to pages not in its map
- WFI - The sub-process is allowed to open files for output in certain cases where this is not normally allowed
- DCWSO - The system is to automatically correct Core Working Set Overflow
- DDWSO - The system is to automatically correct Drum Working Set Overflow
- DPNOD - The system is to automatically put referenced pages into the Drum Working Set
- MSP - The sub-process has Master Sub-Process privileges
- SD - The sub-process has System Diagnostic privileges

Figure 2

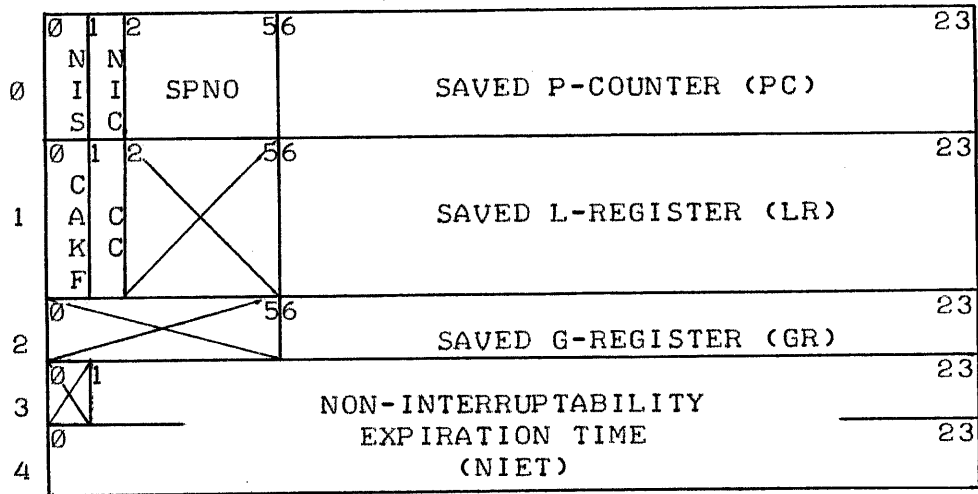
Format of TM and TCM Fields in SPT

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	23
M		P	P				I	P	D	C		D	N	S	P		
A	P	N	N	P	T	B	L	N	W	W	N	M	I	P	M		
C	R	I	I	I	I	L	I	O	S	S	E	R	L	C	T		
C	O	M	C			L	M	D	O	O	P	D	E	S	O		

- MACC - Memory Access Trap
- PRO - Page Read Only Trap
- PNIM - Page Not in Map Trap
- PNIC - Page Not in Core Trap
- PI - Privileged Instruction Trap
- TI - Trapped Instruction Trap
- BLL - BLL error
- ILIM - Indirection Limit Exceeded
- PNOD - Page Not on Drum Trap
- DWSO - Drum Working Set Overflow
- CWSO - Core Working Set Overflow
- NEP - Non-Existent Page
- DMRD - Drum Read Error
- NILE - Non-interruptability Limit Exceeded
- SPCSO - Sub-Process Call Stack Overflow
- PMTO - Process Memory Table Overflow

Figure 3

Normal Format of an SPCS Entry



NIS - Non-interruptability set

NIC - Non-interruptability copied from calling sub-process

SPNO - Index of SPT entry which defines this sub-process

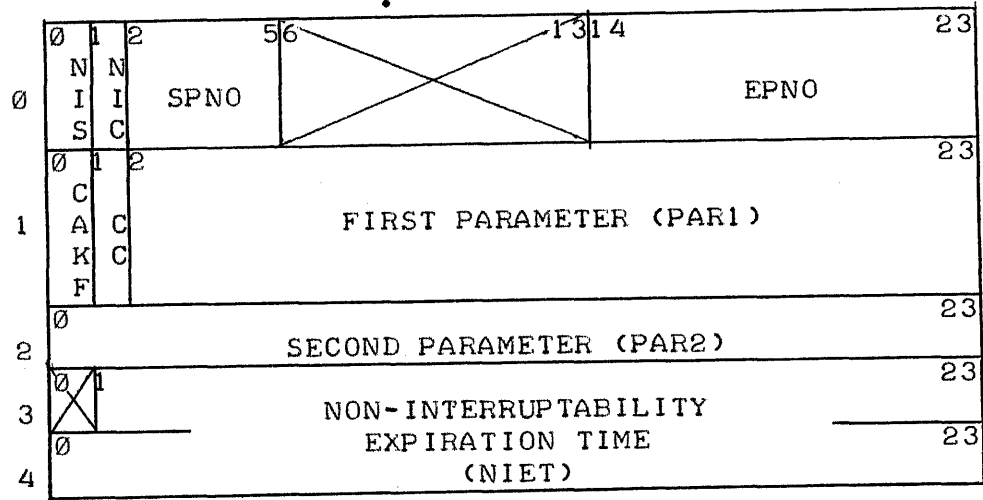
CAKF - Copy caller's Temporary Access Key

CC - Call Completed

NIET - Real time at which non-interruptability status will expire

Figure 4A

Alternative Format of an SPCS Entry



- NIS - Non-interruptability set
- NIC - Non-interruptability copied from calling sub-process
- SPNO - Index of SPT entry which defines this sub-process
- EPNO - Entry Point Number
- CAKF - Copy Caller's Temporary Access Key
- CC - Call Completed
- PAR1 - Interrupt or trap number
- PAR2 - Trap parameter
- NIET - Real time at which non-interruptability status will expire

Figure 4B

Appendix A:

SPS MCALLs in Numeric Order

<u>Number</u>		<u>Name</u>	<u>Page</u>
90	-	CREATE ' SP	17
91	-	DESTROY ' SP	19
92	-		
93	-		
94	-		
95	-	READ ' SPT	34
96	-	READ ' SPCS	35
97	-	READ ' SPT ' FIELD	36
98	-	SET ' SPT ' FIELD	38
99	-	READ ' MAP	42
100	-	SET ' MAP	43
101	-	READ ' MAP ' BYTE	45
102	-	SET ' MAP ' BYTE	46
103	-	READ ' ACCESS ' KEY	47
104	-	COPY ' ACCESS ' KEY	48
105	-	COPY ' TAK	49
106	-		
107	-		
108	-		
109	-		
110	-		
111	-	SP ' CALL	21
112	-	SP ' JUMP	23
113	-	SP ' TRAP	25

Appendix A: (continued)

<u>Number</u>		<u>Name</u>	<u>Page</u>
114	-	TRAP 'RETURN	27
115	-	SP 'BRANCH	28
116	-	SP 'RETURN	30
117	-	JUMP 'RETURN	31
118	-	MARK 'CALL	32

Appendix B:

Error Codes Returned by SPS MCALLs

'ARG' - Argument value unacceptable
'CNA' - Call not allowed
'CSE' - Call stack (SPCS) empty
'CSL' - Call stack (SPCS) level out of bounds
'EPN' - Entry point number illegal
'FHL' - Fatherhood loop in SPT
'MBN' - Map byte number out of bounds
'MNM' - Modify sub-process NAME
'MRG' - Modify sub-process RING
'MSB' - Modify STATUS BITS
'MSC' - Modify STATUS CONTROL BITS
'MTC' - Modify TCM
'MTM' - Modify TM
'MUS' - Modify sub-process USP
'RCS' - Room on SPCS
'RNG' - Ring error
'SPA' - Sub-process access needed
'SPC' - Sub-process control needed
'SPF' - SPT (or SPT entry) full
'SPI' - SPT index out of bounds
'SPM' - Sub-process map
'SPS' - Sub-process on call stack
'TNO' - Trap number out of bounds

Appendix C:

Error Message Numbers Returned by SPS MCALLs

- 100 - byte size or length of string descriptor to SET'MAP
- 101 - value of byte offered to SET'MAP unacceptable
- 103 - attempt to acquire occupied SPT entry
- 137 - attempt to set map of uncontrolled sp
- 138 - SPT index out of bounds
- 139 - trap number out of bounds
- 140 - attempt to call inaccessible sp
- 141 - not enough room on SPCS for SP'CALL or MARK'CALL
- 142 - attempted SP'BRANCH on uncontrolled sp
- 143 - attempt to set uncontrolled SPT entry
- 144 - attempt to modify uncontrolled STATUS BITS
- 145 - attempt to modify uncontrolled SCB bits
- 146 - attempt to modify uncontrolled TCM bits
- 147 - attempt to modify uncontrolled TM bits
- 148 - attempt to set uncontrolled bits in sp KEY
- 149 - attempt to set FATHER to uncontrolled sp
- 150 - attempt to create FATHERhood loop in SPT
- 151 - lack of room on SPCS forbids setting FATHER
- 152 - attempted SP'RETURN from empty stack
- 153 - call stack level for JUMP'RETURN or READ'SPCS out of bounds
- 154 - attempt to JUMP'RETURN over un-controlled sub-process
- 155 - offered P,L,G for MARK'CALL violate ring restrictions
- 157 - offered EP or EG violates ring restriction
- 160 - attempt to delete un-controlled SPT entry

Appendix C (continued)

- 161 - attempt to delete SPT entry which appears on SPCS
- 162 - attempt to commit suicide when SPCS empty
- 163 - map byte number out of bounds
- 165 - unacceptable parameter code offered to READ'SPT'FIELD
- 166 - unacceptable access key code (READ/COPY'ACCESS'KEY)
- 167 - unacceptable parameter code offered to SET'SPT'FIELD
- 168 - attempt to modify the NAME of a sub-process
- 170 - attempt to modify the USP of a sub-process
- 171 - attempt to modify the RING of a sub-process
- 172 - offered value for CAKF is neither \emptyset nor 1
- 173 - illegal entry point number (SP'CALL or SP'JUMP)
- 174 - un-authorized call on COPY'TAK