User Hardware Handbook – Computer

CENTRAL PROCESSOR UNIT NUCLEUS

GEC COMPUTERS LIMITED
Elstree Way, Borehamwood, Hertfordshire.
Telephone No. 01-953-2030

Holding Company – the General Electric Company Limited of England

December, 1977

CONTENTS

Page

IN-TEXT FIGURES

1. INTRODUCTION TO NUCLEUS

The multiprogramming system is divided into two parts. One part, Nucleus, which is described in this document, is responsible for such tasks as enforcing protection between program units, and short term scheduling of the system. The other part, the Executive, is concerned with longer term operations such as control of the system from the operators console.

Several operating systems can be used and all use the same Nucleus, but differ in the facilities provided by the Executive. The Executives of the Operating Systems are implemented in software, and are described in the appropriate Software Manuals.

Nucleus is implemented in hardware microprogram to reduce time overheads on frequently invoked operations and to reduce space overheads for the system.

1.1 TASKS OF NUCLEUS

Both the Executive and the Applications programs are divided into a set of processes each of which is protected from interference by other processes by Nucleus; the processes are allowed to communicate in a safe manner by use of Nucleus facilities.

The main tasks of Nucleus are:—

(a) To enforce protection boundaries between processes to ensure that processes cannot interfere with each other in an uncontrolled fashion.

(b) To provide safe channels of communication between different processes, and between processes and I/O devices.

(c) To perform the short term scheduling for the system, by ensuring that at any time the most urgent process which has useful work to do is in control of the central processor.

1.2 PROCESS CAPABILITIES

To carry out the protection and communication tasks Nucleus maintains systems tables which define for every process a set of Capabilities each of which allows a process to perform some action or to access some data. Nucleus enforces protection by ensuring that a process is not allowed to perform an action unless it has the corresponding capability.

Two kinds of capability are provided:—

(a) The capability to access an area (segment) of main store.

(b) The capability to communicate with another process or an I/O device. This capability is known as a route.

1.3 NUCLEUS OPERATIONS

The Nucleus performs operations either in response to Nucleus instructions being obeyed (provided the process issuing the instructions has the capabilities appropriate to the operation to be performed) or in response to interrupts arising in the Input/Output subsystem.

The facilities of Nucleus, and the manner in which they are invoked are described in this manual. The major groupings are illustrated in Figure 1 which indicates how Nucleus facilities are inter-related, and the sections of this document in which they are described. They are:—

(a) Segment Manipulation instructions SEG and ICB, described in Section 2 of this document. These permit a process to perform limited manipulations of its segment capabilities.

(b) Inter-Process Message (IPM) instructions, CALL and its variants, described in Section 3 of this document. These permit a process to communicate with any other process for which it has a route capability by sending an IPM, and also allow the process to control reception of messages sent to it by other processes.

(c) Input Output. The CALL I/O instruction and its variants, and interrupts are described in Section 4 of this document. These permit a process to initiate I/O transfers on peripheral devices for which it has a route capability, and also allows a process to control reception of interrupt messages generated by peripheral devices.

(d) Semaphore instructions SEM and its variants, described in Section 5 of this document. These permit a process to manipulate binary semaphores held in one of its segments.

(e) Finally, Section 6 describes the Process Selector which is responsible for carrying out the short term scheduling of the system, and Section 7 describes the action taken when an error condition arises while the system is running.

(f) Section 8 of this manual summarises the system tables used by Nucleus, and the Nucleus instructions described in Sections 2 to 5.

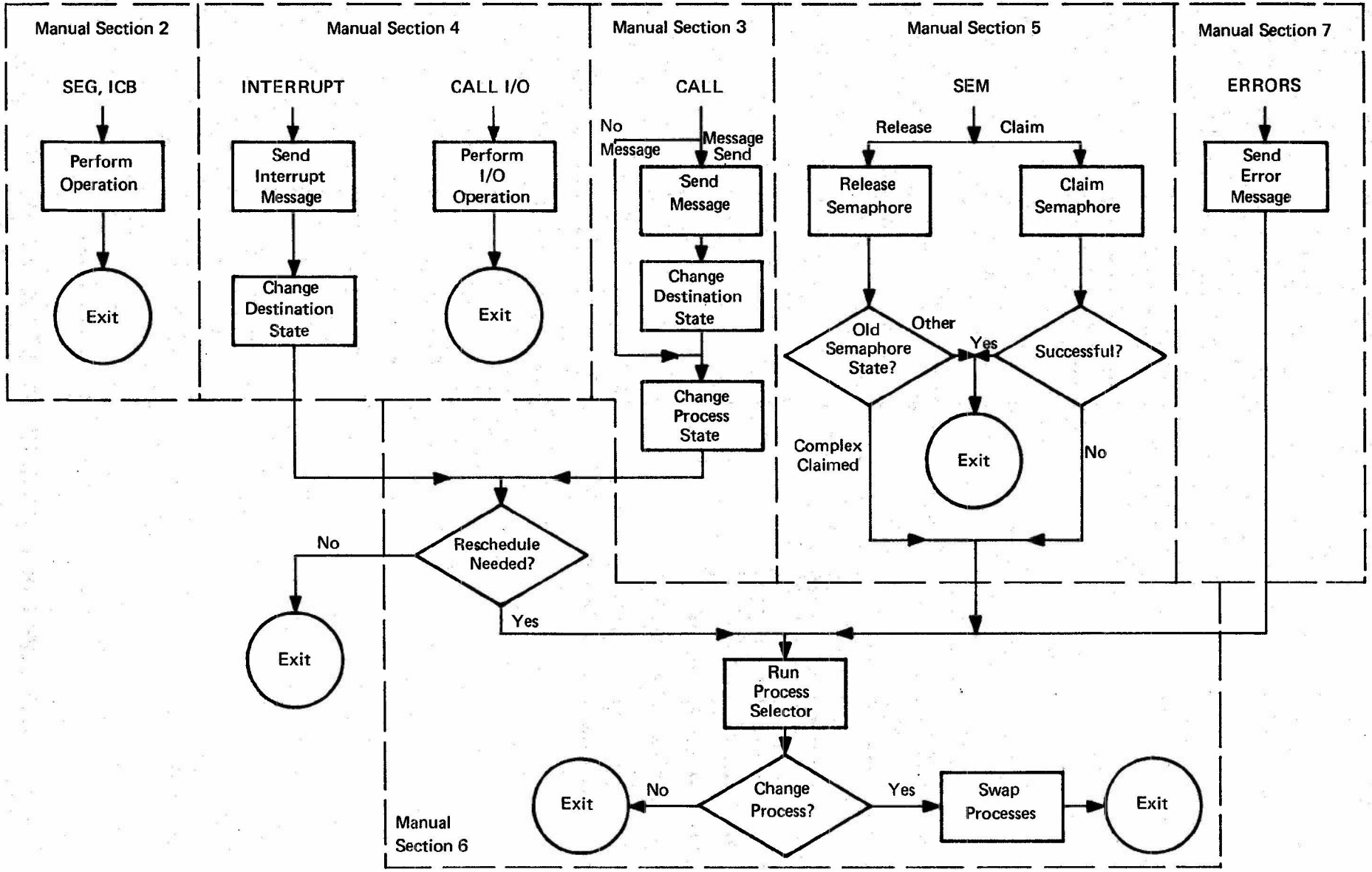


Figure 1: NUCLEUS OPERATIONS

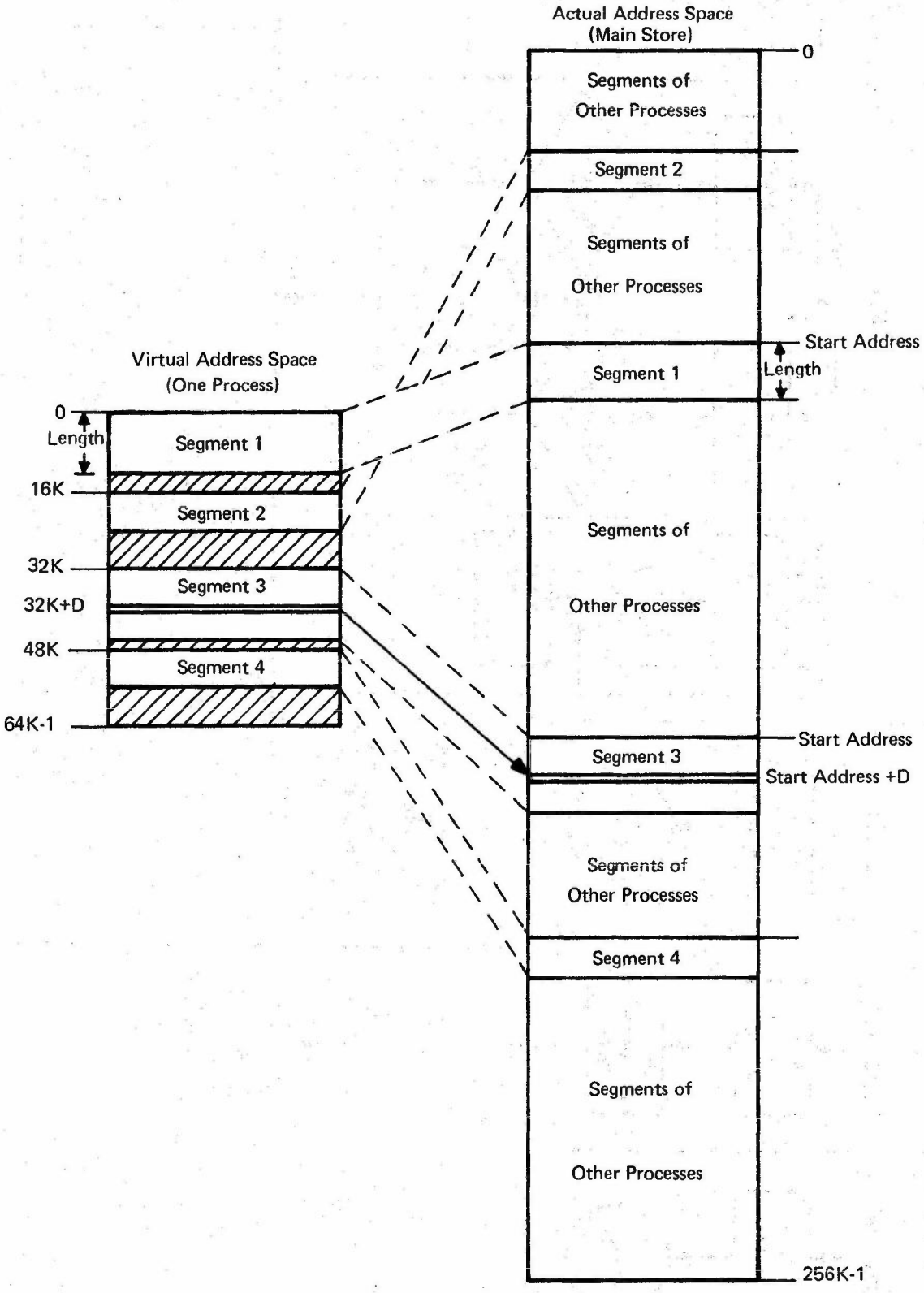


Figure 2: AN EXAMPLE OF VIRTUAL TO ACTUAL ADDRESS RELATIONSHIP

2. SEGMENTATION SYSTEM

The storage used by the CPU is divided into a number of segments. Each segment contains a multiple of 64 bytes of storage, up to a maximum length of 16 Kbytes.

Each segment may be held in main store, where it occupies contiguous store locations starting at an address which is a multiple of 64. In an overlayed system, all segments need not be present in main store. Those segments not in main store are held on backing store, and are allocated space in main store only when they are in use.

Each process in a system may have at any time access to four segments, known as its current segments. Items of storage are accessed by the process using a 16 bit virtual address. The most significant 2 bits of the virtual address define the segment in which the item to be accessed is located. The remaining 14 bits define the displacement of the item from the start of the segment.

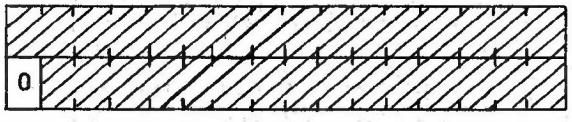
Before the item can be accessed from main store the virtual address must be converted into an 18 bit actual address. This conversion is referred to as mapping. The relationship between the virtual and the actual address is shown in Figure 2. As an example, if it is required to access the item whose virtual address is 32 K + D, the actual address is determined by taking the start address of the third segment of the process and adding the displacement D to it. It is also necessary to ensure that the actual address lies within the segment in question. This is done by comparing the displacement D with the segment length and signalling an error if D is greater than the segment length.

2.1 THE SYSTEM SEGMENT TABLE

To control the segmentation system, Nucleus uses a table in main store, called the System Segment Table (SST). Each segment in the system has an entry in this table, the entry defines whether the segment is currently in main store or not, and if it is present defines its start address and length.

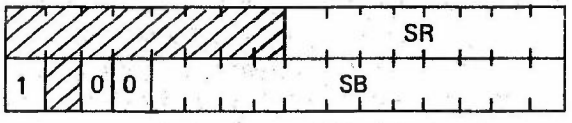
SST entries are all four bytes long, the entry for segment N starting at byte 4N of the table. Two possible formats are used:

(a) *Segment Absent*



The most significant bit of the second halfword of the entry in the presence bit, which is set to zero to indicate that the segment is absent from main store. The shaded bits of the entry may be used as required by system software, and are not changed by or used by Nucleus.

(b) *Segment Present*



The presence bit is set to one to indicate that the segment is present in main store. The SB and SR fields define the position and length of the segment as follows:—

The segment starts at actual byte address 64*SB
The length of the segment is 64*(SR + 1) bytes

SB is referred to as the segment base and SR is the segment range.

The shaded bits of the entry may be used as required by systems software, and are not changed by or used by Nucleus.

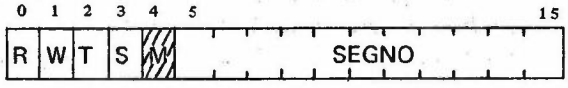
2.2 THE MASTER SEGMENT

In addition to its four current segments, each process has a master segment which contains tables defining the capabilities and status of the process. The master segment for process P is defined by entry P + 4 of the SST.

To control the segmentation system each master segment contains two tables, the Current Segment Table (CST) and the Process Accessible Segment Table (PAST).

The Current Segment Table

The CST contains four entries, defining the four segments which can be accessed by the process. Each entry in the CST has the format:—



SEGNO is an 11 bit field which is the number of the segment in the SST. It points to the entry starting at byte 4*SEGNO of the SST.

The four one bit flags R, W, T, S define the access permissions for the segment as follows:—

R — if set, reading from the segment is permitted for data or instructions.

W — if set, writing to the segment is permitted.

T — if set, the segment may be used for autonomous I/O transfer. See Section 4 for details.

S — if set, the segment may be transmitted with an inter-process message See Section 3 for details.

The M bit may be used as necessary by system software. It is not used by Nucleus but may be reset under some circumstances. See Section 3.5.

The Process Accessible Segment Table

A process may be allowed access to more than four segments, although only four can be in use (i.e. occupy its virtual address space) at any time. The PAST contains entries defining the segments which may be accessed; each PAST entry has the same format as a CST entry. The CST contains copies of the PAST entries for the four current segments. The selection of PAST entries copied in the CST may be changed by hardware instructions. See Section 2.4 for details. The number of entries in the PAST is defined by location PASTMAX of the Master Segment, and the position of the PAST in the Master Segment is defined by location PASTPTR. Location CODESEG of the Master Segment contains the number of the PAST entry which is currently copied into CST[3]. This segment normally contains the code of the process currently being executed.

The relationship between the PAST, CST, and SST is illustrated in Figure 3.

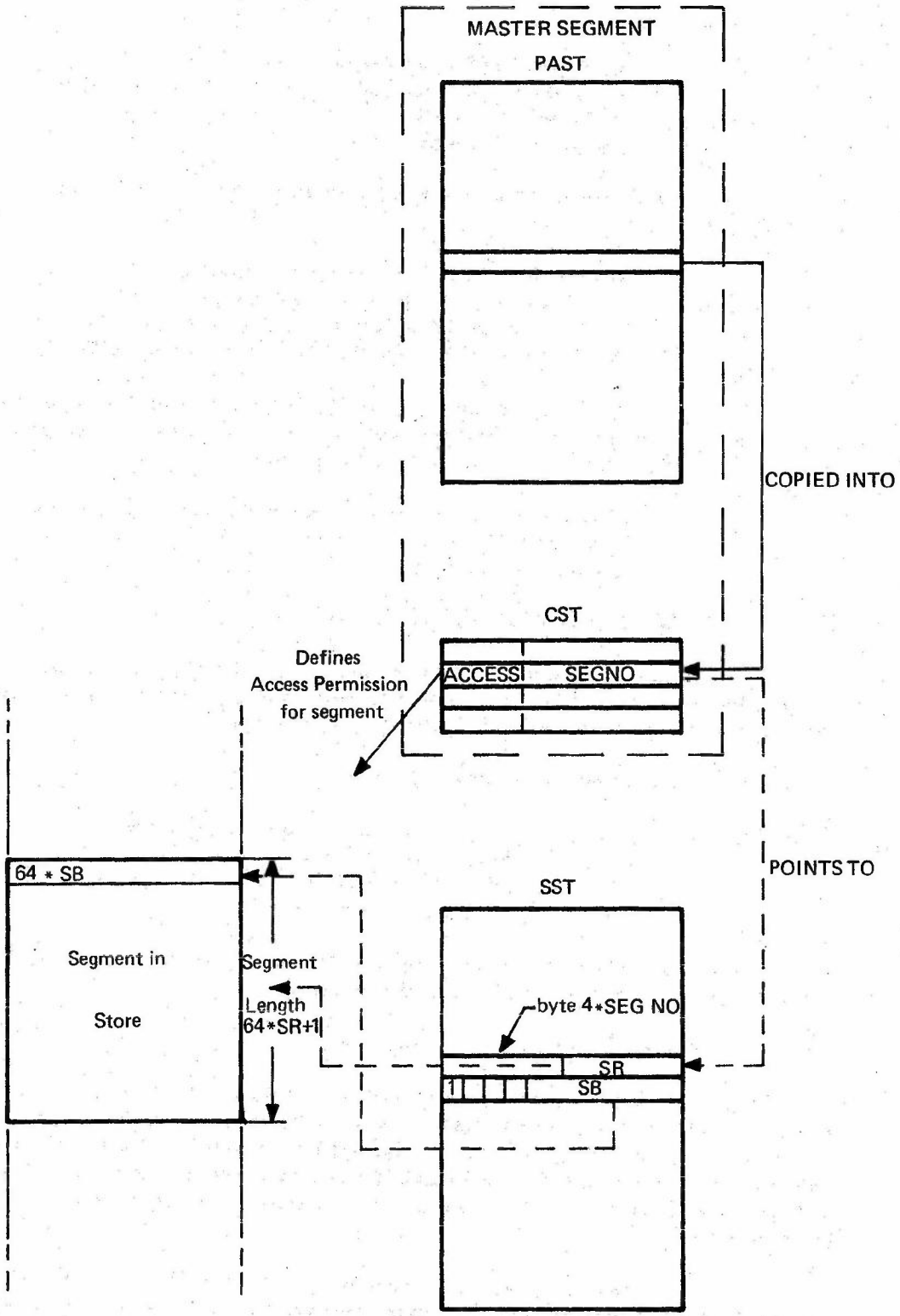


Figure 3: RELATION BETWEEN PAST, CST, AND SST

2.3 NUCLEUS SEGMENTS

To control the operation of the system, Nucleus has access to four segments containing systems information. These segments are accessible only to Nucleus and specially privileged systems processes. The segments are:—

The System Variable Area (SVA)
The System Segment Table (SST)
The Process Vector (PV)
The System Buffer Area (SBA)

To enable systems software to access these segments, the first four locations of the SST are conventionally allocated to them as follows:—

SST[0] defines the SVA. The Segment Base must be zero
SST[1] defines the SST. The entry must be identical to SSTBASE
SST[2] defines the PV. The entry must be identical to PVBASE
SST[3] defines the SBA. The entry must be identical to SBABASE

The SST has already been described. The PV, SBA, and SVA are described in later sections. The SVA segment always starts at location 0 of main store. The other three segments are defined by three entries in the SVA; each entry has the same format as an SST entry. The format of the entries is:—

SSTBASE defining the Base and Range of the System Segment Table
PVBASE defining the Base and Range of the Process Vector
SBABASE defining the Base and Range of the System Buffer Area

2.4 HARDWARE SEGMENT REGISTERS

The mapping of virtual into actual addresses is carried out by a hardware unit which uses eight Hardware Segment Registers, each of 20 bits, to define the position and length of segments in store. Each register contains a segment base and segment range, defined as for SST entries.

The eight registers are used as follows :—

HSR [0—3] define the four current segments of a process
HSR [4] defines the Master Segment for a process
HSR [5—7] define the Process Vector, System Segment Table, and System Buffer Area respectively.

Associated with the first four registers HSR [0—3] are four Hardware Protection Registers, each of 2 bits, which contain the access permissions for the four segments.

Address Mapping

The mapping procedure for a current segment is shown schematically in Figure 4. The 16 bit virtual address is placed in the Virtual Address register. The SN field of the virtual address defines the segment in which the address lies. This is used to access one of the four segment registers (HSR[SN]) which contains the segment base SB and segment range SR. The segment base SB is added to the VP field of the virtual address to form the most significant 12 bits AP of the actual address. The LN bits are then concatenated with AP to form the full 18 bit actual address.

Simultaneously, the segment range SR is compared with VP. A 'range error' is indicated if VP>SR. Also the R and W bits from the Protection Register are matched against the mode of access requested (read or write). A 'protection error' is indicated if the mode requested is not permitted (e.g. a write request with no write permission).

The remaining four HSRs are used in a similar manner by Nucleus for access to the Master Segment, PV, SST, and SBA. In these cases the register to be used is predefined by the Nucleus. There are no associated protection registers; Read and Write access permission being assumed for all segments.

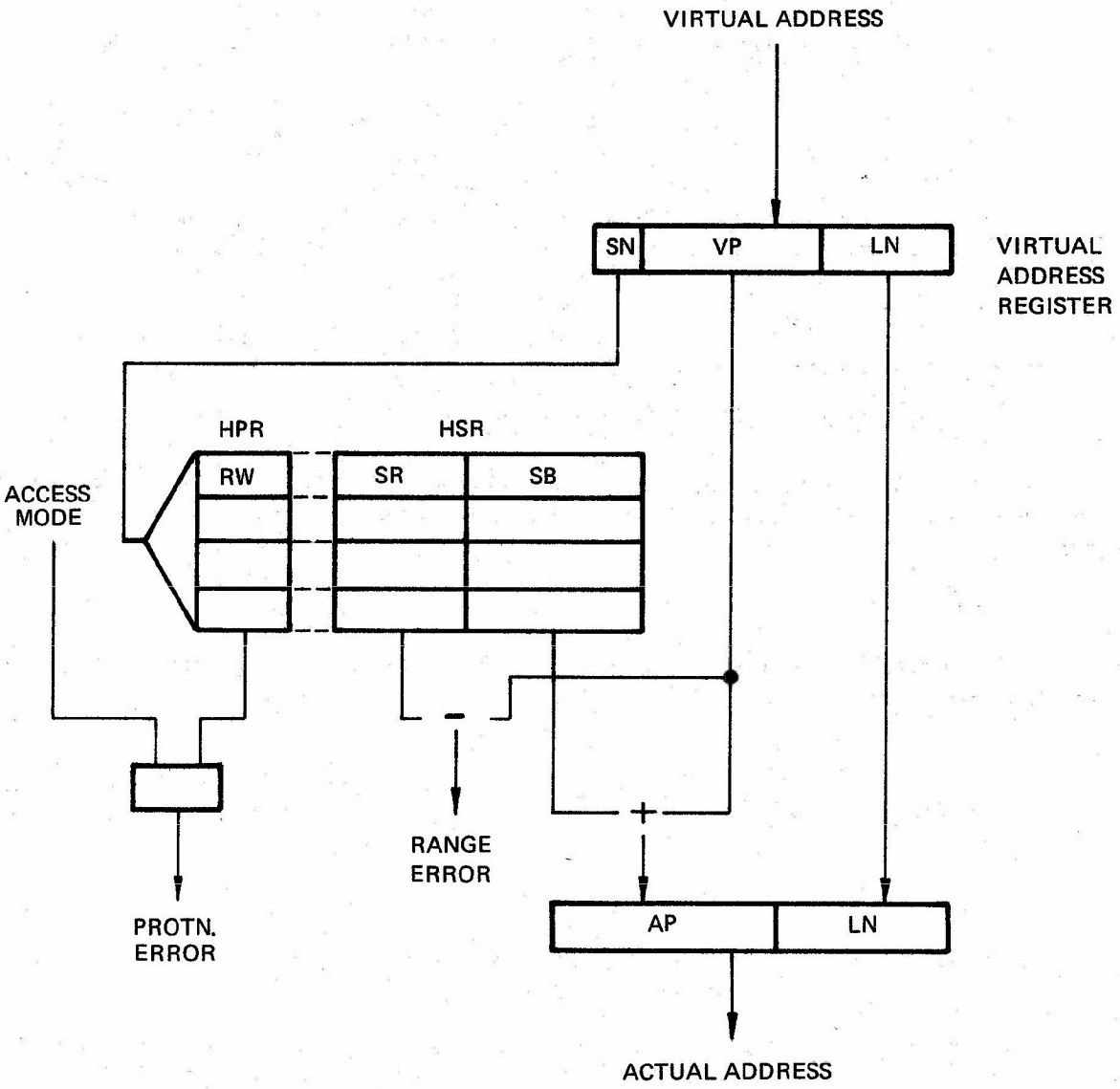


Figure 4: ADDRESS MAPPING

If either a range error or protection error is indicated during address mapping, access to main store is not performed, and an error trap occurs. Error traps are described in more detail in Section 7.

Loading Segment Registers

Segment registers HSR [5–7] are loaded from PVBASE, SSTBASE, and SBABASE respectively when the system is first set running, for example following Initial Program Load (IPL).

Whenever a new process is selected to run, HSR [0–4] are loaded to define the four current segments of the process, and the Master Segment as follows:–

HSR[4] is loaded from SST [P + 4] where P is the number of the process. If the presence bit of the segment is zero, indicating that the segment is absent, the process cannot be run. A special error trap, known as a master segment break occurs and is dealt with as described in section 7.3.

HSR[0–3] are loaded using the four CST entries CST [0–3]. The R and W access permission bits from the CST are copied into the corresponding hardware protection register (HPR). The SEGNO field of the CST is used to access an SST entry, SST [SEGNO]. If the presence bit of the SST entry is zero, the process cannot be run and an error trap, known as a segment break occurs as described in Section 7.3. Otherwise the corresponding HSR is loaded from the SST entry.

Note the consequence of the above is that a process can be run only if its Master Segment and all four current segments are present in store.

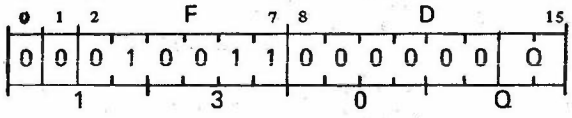
2.5 SEGMENT MANIPULATION INSTRUCTIONS

The Segment Manipulation instructions permit a process to control the allocation of segments in its virtual address space. A single format L instruction is used, the Displacement field specifying the operation to be performed.

LCST

The Load CST instruction loads a specified PAST entry into a specified CST entry, thus giving the process access to the segment in question.

FORMAT:



REGISTERS: Initially Register X contains the number of a segment in the PAST. No registers are changed by the instruction.

EFFECT: The PAST entry defined by register X is copied into the CST entry defined by Q. The Access bits R and W are loaded into the appropriate HPR. The SEGNO field defines an SST entry which is loaded into the appropriate HSR. If Q = 3, then the content of register X is stored also in the CODESEG location of the Master Segment.

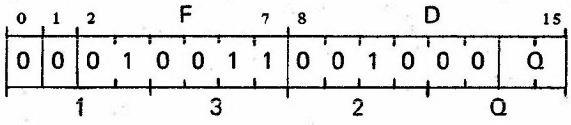
TRAPS: An error trap occurs if the content of register X is greater than or equal to PASTMAX (Code P4).
If the presence bit of the SST entry is zero, a segment break occurs. (Code B9).

NOTE: Although LCST can be used to change CST[3] as noted above, such usage is discouraged if CST[3] contains the code being executed at the time, since an uncontrolled Branch is taken to the code in the new segment. Controlled Branches between different segments can be effected by the ICBR and ICBL instructions described later.

CLCS

The Conditional Load CLCS instruction is used instead of LCST when it is necessary to verify the presence, length, or access permission of the segment before attempting to use it. If the segment is absent, then the instructions has no effect save the setting of conditions bit CZ to 1. If the segment is present, action is as for LCST and in addition CZ is made zero, the length of the segment in bytes is placed in RX and the PAST entry is copied into Register AL.

FORMAT:



REGISTERS: Initially Register X contains the number of a segment in the PAST. CN, CA, COF are all reset to zero by the instruction. If the segment is absent, CZ is set to 1. No other registers are affected.

If the segment is present, CZ is reset to zero, register X is loaded with the length of the segment in bytes, and the PAST entry defined by register X is loaded into register AL. No other registers are affected.

EFFECT: If the segment involved is absent from store, then CZ is set to one. No other action takes place.

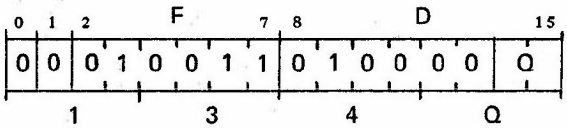
If the segment is present, operation is as described for LCST. In addition, CZ is reset to zero, register X is loaded with the length in bytes (=64*(SR+1))using the SR field of the SST entry involved, and the PAST entry is copied into Register AL.

TRAPS: An error trap occurs if the content of register X is greater than or equal to PASTMAX. (Code P4).

SCST

The Store CST instruction stores a specified CST entry in a specified PAST entry. The Hardware Segment and Protection registers are not affected.

FORMAT:



REGISTERS: Initially Register X contains the number of a segment in the PAST. No registers are changed by the instruction.

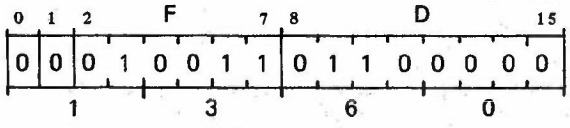
EFFECT: The CST entry defined by Q is copied into the PAST entry defined by Register X.

TRAPS: An error trap occurs if the content of register X is greater than or equal to PASTMAX (Code P4).

LHSR

The Load Hardware Segment Registers instruction is used to load HSR [5–7] from PVBASE, SSTBASE, and SBABASE. A Nucleus Reschedule operation is performed, using the new settings. The instruction normally should be used, only during system initiation, by a privileged system process. Note however that use of the instruction by a non-privileged process has no effect, since it reloads the HSRs with their existing contents.

FORMAT:



REGISTERS: No registers are affected.

EFFECT: HSR[5–7] are loaded from PVBASE, SSTBASE, and SBABASE respectively.

TRAPS: No trap conditions.

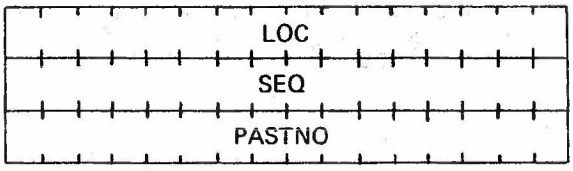
2.6 INTER-CHAPTER BRANCH INSTRUCTIONS

Each process comprises code and data chapters, grouped together to form code segments and data segments. Code segments normally are used as current segment 3, and are normally only given read access permission. Each code chapter has an associated Local Workspace area, which is pointed to by register L when the code chapter is being executed. Normally the local workspace area is held in current segment 0.

The Inter-Chapter Branch instructions are used to transfer control from one chapter to another. They are intended to be used with the code of the two chapters in either the same or different segment in the PAST, but with both executed from current segment 3. If the code of the destination chapter is in a different segment in the PAST from that of the calling chapter, Nucleus loads that PAST segment into CST[3] before executing the destination chapter.

Branch Descriptors

The destination chapter is specified in terms of a chapter descriptor held in main store. This comprises three halfwords, used as follows:–



LOC – defines the virtual address of the local workspace area of the destination chapter. LOC is loaded into register L before the destination chapter is executed.

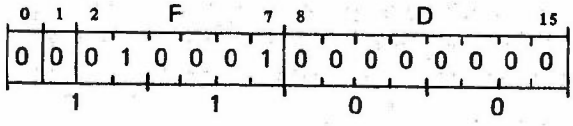
SEQ – defines the entry address within the code of the destination chapter. SEQ is loaded into the sequence register S before the destination chapter is executed. Note that SEQ normally should be a virtual address in current segment 3.

PASTNO – defines the PAST entry of the segment containing the code for the destination chapter. If it differs from the PAST number of the calling chapter's code (which is held in location CODESEG) then the PAST entry is loaded into CST[3].

ICBR

The Inter-Chapter BRanch instruction uses a chapter descriptor pointed to by register Z to enter the destination chapter.

FORMAT:



REGISTERS: Initially Register Z contains a pointer to the destination chapter description.
Register L is loaded from the LOC of the descriptor.
Register S is loaded from the SEQ of the descriptor.
No other registers are affected.

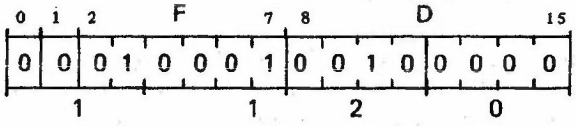
EFFECT: Register L is loaded from the LOC field of a chapter descriptor pointed to by Z. Register S is loaded from the SEQ field of the descriptor. If the PASTNO field is the same as CODESEG, then the calling and called code are in the same segment, and no further action is necessary. Otherwise the segment is loaded into CST [3] as described for the LCST instruction.

TRAPS: An error trap occurs if the PASTNO field of the descriptor is greater than or equal to PASTMAX, (Code P4).
If CST [3] is changed by the instruction, a segment break occurs if the segment loaded is absent from main store, (Code B9).

ICBL

The Inter-Chapter Branch and Link instruction is similar in operation to the ICBR instruction, except that a link descriptor is formed and stored in the first three halfwords of the local workspace of the destination chapter. This link may be used subsequently to return to the calling chapter by means of an ICBR instruction.

FORMAT:



REGISTERS: As for ICBR

EFFECT: The LOC field of the descriptor pointed to by Register Z defines the start of the local workspace of the destination chapter. A link descriptor is formed and stored in the first three halfword of this area as follows:—

LOC contains the content of register L
SEQ contains the content of register S
PASTNO contains CODESEG

Operation is then as described for the ICBR instruction.

TRAPS: As for ICBR

3. INTER-PROCESS MESSAGES

Processes in a system may communicate with each other by means of Inter-Process Messages (IPMs). An IPM is a packet of information which is transmitted from one process (the sender) to another (the destination). Normally the purpose of an IPM is to inform the destination process that the sending process wishes it to perform some task; the information in the message may be used to define the task to be performed and the data to be operated on.

3.1 SENDING A MESSAGE

To send a message, a process must first place the information parameters of the message in registers A, X, and Y, and must load register Z with a route number defining the destination, and then obey a variant of the CALL instruction which causes message transmission to take place.

The route number in register Z selects an entry from a table in the Master Segment of the sender, known as the Route Table. The route table entry defines the destination of the message and contains Flags which are used to further define the operation to be performed.

In particular the Flag bits define the message transmission system to be used, and whether or not a segment reference is to be included as a parameter of the message. This latter feature may be used to give the destination process temporary access to one of the Current segments of the sending process which may for instance contain a table of data to be operated on. If a segment is sent with the message, the most significant two bits of the parameter in register Y are used to select one of the four Current segments of the sender. Then the CST entry corresponding to this segment is included as one of the parameters of the message, and the parameter in register Y is modified by replacing its most significant 2 bits by zeros.

3.2 RECEIVING A MESSAGE

A process in a system which has useful work to do is said to be in the RUN state. In this state, the process cannot receive messages. Other processes may send messages to it, but these will be stored in the System Buffer Area (SBA) by Nucleus until the process is ready to receive a message.

A process may voluntarily leave the RUN state by obeying a variant of the CALL instruction which causes the state of the process to change. It may by this means choose to make itself FREE to receive messages from any source, or it may choose to WAIT for a message from one particular source. When the wanted message arrives the process is moved to a special state, the READY state, indicating that there is work to be done but processing has not yet started. The transition to the READY state is immediate if the message has been transmitted previously and is held already in the SBA. Otherwise transition to the READY state takes place as soon as the wanted message is sent.

When the process is next given control of the processor, the parameters of the message are loaded into its registers and it is placed once more in the RUN state. At the same time an entry number is placed in register Z to identify to the process the source of the message it has just received.

3.3 MESSAGE TRANSMISSION

The task of the message transmission part of Nucleus is to mechanise the transmission of messages from sending to destination processes, and to arrange for their storage in the System Buffer Area (SBA) until their destination processes are willing to receive them.

Two largely independent systems are provided, the Queued message system and the Fixed message system. In the first system, messages are stored in the SBA in dynamically allocated buffers, whilst in the second system the buffers used for storing messages are pre-allocated when the system is set up.

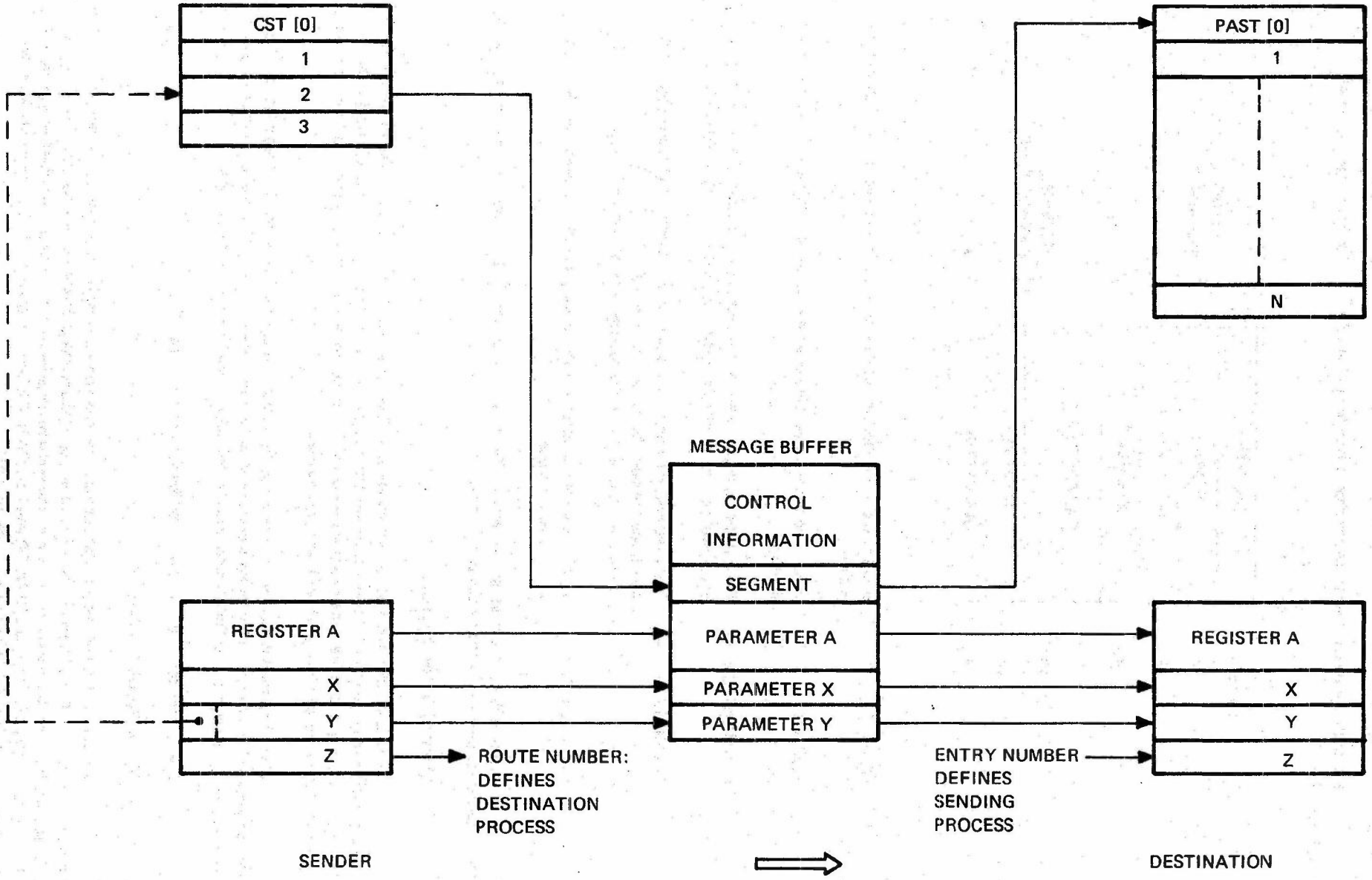
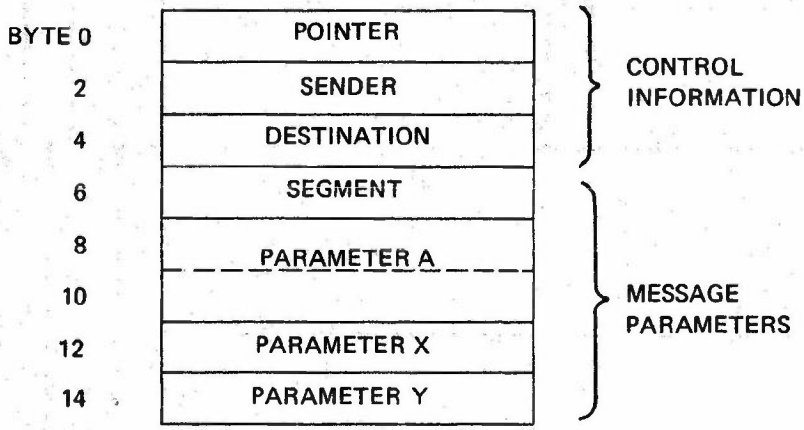


Figure 5: MESSAGE PARAMETERS

Message Buffers

For both transmission systems, messages are stored in the SBA in 16 byte message buffers of identical format:–



POINTER – Is only used in the Queued message system. For a fixed buffer, POINTER is zero.

SENDER – Is the number of the sending process (i.e. the process which generated the message).

DESTINATION – Defines the process number of the destination process, and the entry number which will be used to identify the sender to the destination process.

SEGMENT – Is the segment reference if a segment is being sent with the message. In this case it has the same format as a CST entry.
If no segment is sent with the message, SEGMENT will be zero.

PARAMETER – A, X, Y are the parameters of the message copied from registers A, X, and Y of the sending process.

Figure 5 shows the relationship between the registers of the sending process, the message in a buffer, and the message as received by the destination process.

Queued Buffer System

In the queued buffer system, buffers are allocated by Nucleus from a list of unused buffers (known as the free queue) as they are required. Message parameters and control information are stored in the buffer and the buffer is added to a queue of messages awaiting processing by the destination process. Note that each process in the system has its own independent queue of incoming messages.

When a message is received by a process, the message parameters are loaded into registers and the buffer is removed from the incoming message queue and added to the free queue by Nucleus. Both the free queue and the incoming message queues for each process are maintained on a first in – first out basis.

For further details of the queued buffer system, see section 3.7

Fixed Buffer System

In the fixed buffer system, the buffers to be used for messages on particular routes are pre-allocated at system generation. The address of the buffer to be used for a message, relative to the start of the SBA, is defined by the route table entry for the route over which the messages is to be sent. The control information is preset in the buffer (so that POINTER is zero, and SENDER and DESTINATION are constant). In the case of fixed routes, the ENTRY part of the DESTINATION must be less than 16.

Notes on Usage

As far as the sending and destination processes are concerned, there is no difference visible within the process between the queued and fixed message system. Each system, however, has particular advantages which should be considered before deciding which transmission method to use for any particular route, namely:—

(a) The Fixed message system is faster in operation than the queued message system, and therefore imposes lower time overheads on the running of the system.

(b) With the Queued message system it is possible that at some stage all buffers in the SBA are in use (i.e. the free queue is empty) so that no queued message can be handled. This problem does not arise with the Fixed message system, since buffers are preallocated at system generation.

(c) With the Fixed message system, it is possible that if a number of messages are sent on the same route (therefore using the same buffer) in rapid succession, a later message may be generated before an earlier message has been received by the destination process. In such circumstances the later message simply overwrites the earlier message in the buffer which is lost without trace. If there is a possibility of this situation arising, the queued system is to be preferred since each message is allocated its own buffer and is processed by the destination in time sequence.

(d) Queued messages are accepted by a destination process in time sequence — that is, the earliest message generated is processed first. Fixed messages are processed at a higher priority than queued messages — that is, all outstanding fixed messages are processed before outstanding queued messages are dealt with. Thus use of the fixed message system allows urgent messages to be dealt with (as fixed messages) before less urgent (queued) messages which might have been generated at an earlier time.

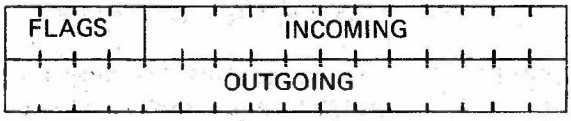
(e) Fixed messages impose a constant space overhead on the system, since the buffers are preallocated in the SBA while the system is running, whether the processes using them are activated or not. In a well designed system queued messages make significantly lower space demands since statistically the number of buffers in use at any time is smaller than the number of different routes in the system. To conserve space, it is desirable to limit the number of fixed buffers as much as is possible.

In view of the above, the use of queued routes is to be preferred in all circumstances except those in which the features of the fixed route system are advantageous and the disadvantages of using fixed routes can be tolerated.

3.4 THE ROUTE TABLE

As stated earlier, the message transmission system is controlled by a table held in the Master Segment of a process, known as the Route Table. This table really consists of two independent tables interleaved, one controlling the sending of messages from a process, the other controlling reception of messages by that process.

The general format of a 4 byte route table entry is:—



The FLAGS are used to define:—

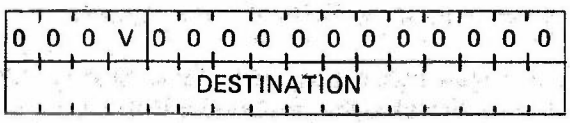
The system (queued or fixed) of transmission of outgoing messages.
The system (queued or fixed) of reception of incoming messages.
Whether a segment is to be sent with outgoing messages.

The INCOMING field is used if necessary to control the reception of incoming messages. Its format depends on whether fixed or queued messages are to be received.

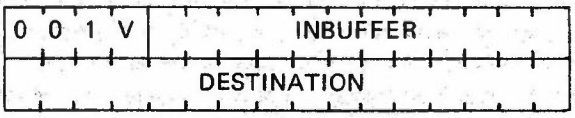
The OUTGOING field is used to control the transmission of outgoing messages. Its format is dependent on whether fixed or queued messages are to be sent.

There are, therefore, four possible formats for route table entries:–

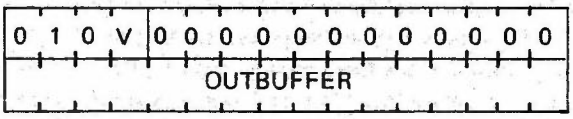
Queued Out – Queued In



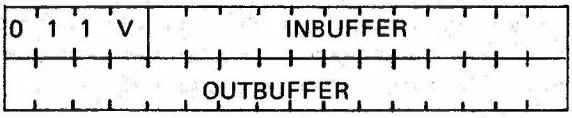
Queued Out – Fixed In



Fixed Out – Queued In



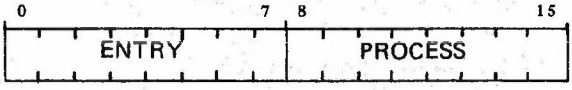
Fixed Out – Fixed In



Where in the above:–

V – is the Segment Send bit. Set to 1 if a segment is to accompany the message, 0 otherwise.

DESTINATION – defines the destination process number and entry number; it has the format:–



INBUFFER – is the address of a fixed message buffer relative to the start of the SBA, right shifted 4 places.

OUTBUFFER – is the address of a fixed message buffer relative to the start of the SBA. The least significant 4 bits of this address must be zero.

Pairing of Routes

It is advised that routes should be set up in pairs. That is, if route X of process A points to entry Y of process B, then route Y of process B should also point to entry X of process A. This convention is not enforced by Nucleus, but has the advantage to the user that the same number is used as the route number for outgoing messages to B and as the entry number for incoming messages from B.

The resulting situation for a pair of queued routes is shown in Figure 6.

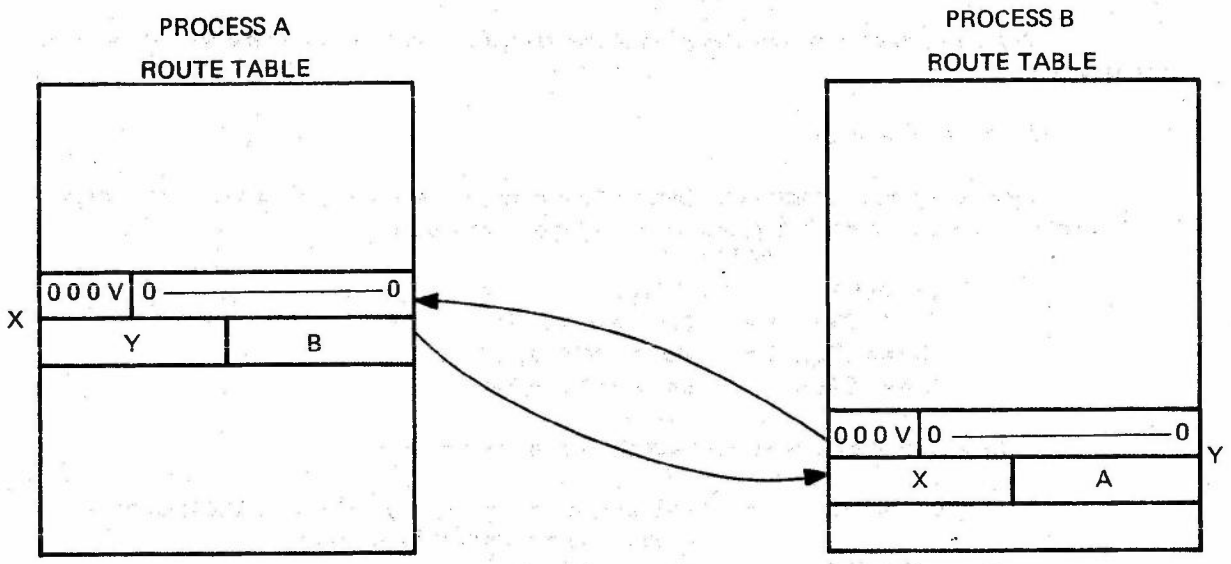


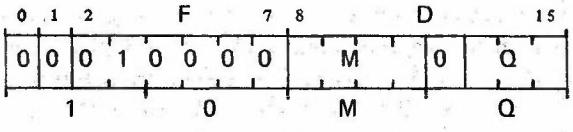
Figure 6: ROUTE PAIRS

3.5 THE CALL INSTRUCTIONS

The sending and reception of IPMs is controlled by a single format L instruction whose D field is used to further define the operation to be performed and the state in which the process should be placed at the completion of the instruction. Variants of the instruction allow a process to either:—

Send a message	SEND
Change its state	STAT
Send a message and then change its state	RSEN

The format of the instruction is:—



M determines whether a message is to be sent

Q determines the next state of the process.

In all cases register Z must contain a valid route number when the instruction is obeyed.

The effect of the CALL instruction for defined values of M and N is described in the following sections. Where M indicates that a message is to be sent and N indicates that a change of process state is to occur, the result is exactly as if a 'send message' followed immediately by a 'change state' instruction is obeyed.

Message Sending

The M field controls the sending of messages in conjunction with the route table entry defined by register Z as follows:—

M = 2: No Message

If M = 2 no message is sent. The effect of the instruction is only to change the state of the process as defined by Q.

M =6 : Send Message

If M = 6 a message is sent to the process defined by the route table entry indicated by register Z. If the V bit of the route table entry is zero, the parameters of the message are:—

SEGMENT	=	0 (Zero)
PARAMETER A	=	Content of A register
PARAMETER X	=	Content of X register
PARAMETER Y	=	Content of Y register

If the V bit of the route table entry is a 1, the parameters are:—

SEGMENT	=	CST entry defined by the m.s. 2 bits of Y. The M bit of the CST entry is always transmitted as a zero.
PARAMETER A	=	Content of A register
PARAMETER X	=	Content of X register
PARAMETER Y	=	Content of Y register with the m.s. 2 bits transmitted as zero.

M =4 : Send Message (Restricted Access)

If M =4 the effect is exactly as for M = 6, except that if a segment is sent (V = 1 in the route table entry) the W and T bits of the CST entry are transmitted as zero. This has the effect of prohibiting the destination process from either writing to or initiating I/O transfers upon the segment sent.

State Change

The Q field of the instruction determines the next state of the process as follows:—

Q = 0: Go Free

The process is placed in the FREE state, its message queue and incoming fixed buffers are examined. If there are any messages awaiting processing, the process is placed immediately in the READY state and in due course enters the RUN state when a message is loaded into registers. If there are any messages in fixed buffers, the message with the lowest ENTRY number is loaded. If there are no messages in fixed buffers, the message at the head of the message queue is loaded.

The Registers which are loaded from the message buffer are shown in the table below:—

REGISTER	CONTENTS
E, B	Undefined
A	PARAMETER A
X	PARAMETER X
Y	PARAMETER Y
Z	ENTRY
C	ZERO
PAST [0]	SEGMENT

If no messages await processing, the process remains in the FREE state until some other process generates a message for it, when the process is placed in the READY state as described above.

Q = 1: Continue Running

If the Q field is equal to 1, the process continues in the RUN state, and is not given a new message to process.

Q = 2: Wait (Pass priority)

The process stays in the WAIT state until a message arrives with an ENTRY number equal to the content of register Z. If a message with the required ENTRY number has arrived, the process is placed immediately in the READY state. Otherwise the process remains in the WAIT state until a message arrives with the required ENTRY number.

While in the WAIT state the priority of the process is passed to the process from which the awaited message is to be sent. This modifies the process selection algorithm in such a way that the awaited process runs at a higher priority in order to expedite the sending of the awaited message. See Section 6 for details.

Q = 3: Wait (No Pass)

The effect is exactly as for Q = 2, except that the priority of the process is not passed.

Q = 4: Conditional Free

If any messages are outstanding for the process, the effect is as described for 'Go Free' (Q = 0). If no messages are outstanding the process remains in the RUN state, but the Conditions bit CZ is set to a 1.

Q = 7: Conditional Wait

If the awaited message has arrived, the effect is exactly as described for 'Wait' (Q = 2). If the awaited message has not arrived the process remains in the RUN state, but conditions bit CZ is set to a 1.

Notes on Usage

(a) From within the process, there is no substantial difference between the use of a CALL instruction with Q = 2 (Wait, Pass priority) and Q = 3 (Wait, no priority pass). Each mode has particular advantages and disadvantages which should be considered before the decision is made whether to pass priority or not in any particular circumstance.

If priority is passed, and the message awaited has not yet arrived, then the process that eventually generates the message is run, until it has sent the message, at whichever priority is the greater of its own priority and the priority passed to it. Thus if process 10 is waiting for a message from process 20, and it has passed priority, process 20 is scheduled temporarily as if its priority is that of process 10. It is run in preference to processes 11, 12 . . . . and so on, and is encouraged to complete its processing and generate the awaited message as rapidly as possible, so that process 10 can resume operation.

Passing priority to a higher priority process has no useful effect on the scheduler. Each instance of priority passing increases the time taken for the reschedule operation to be completed. Unnecessary priority passing serves no useful purpose but may increase the time overhead for the Reschedule operation. This effect only becomes significant if no high priority processes are in a runnable state.

In view of the above, it is recommended that mode Q = 2 (Pass priority) is employed unless it is obvious that under no circumstances does it have a useful effect. If this is so mode Q = 3 (No pass) should be employed.

(b) Modes Q = 4 and Q = 7 (Conditional Free and Conditional Wait) may be used as follows:—

Mode Q = 4 can be used to test whether there is any more work for a process to do (i.e. whether any messages are awaiting processing). After completion of the instruction conditions bit CZ can be tested (with a Branch if Zero or Branch if Non Zero instruction). If CZ is set, no further messages await processing, if CZ is not set at least one message is present. In the latter case the first outstanding message will have been loaded into registers and register Z loaded with the ENTRY number as usual. This

mode should be used only in those few cases where a process has a long processing task to perform, and it is desirable to test periodically whether an urgent message has arrived before that task is completed..

Mode Q = 7 is used similarly to Mode Q = 4, but this time to detect the presence or otherwise of a message with a particular entry number. Again a Branch if Zero or Non-Zero instruction can be used after the CALL to determine whether a message with the correct ENTRY number is outstanding. The mode can be used in a similar way to test whether an urgent message from some specified source has arrived before the processing of a low priority message is complete.

(c) CALL instructions with M = 2 and Q = 3 can be issued where the route number in Z defines an Autonomous Input/Output route. This may be used to control the reception of interrupts, as detailed in section 4.5.

Error Traps

The following error traps may occur when a CALL instruction is obeyed.

(a) The route number in Z is greater than the number of routes of the process. The instruction is abandoned and the sending process is placed in the STOPPED state (Code P3).

(b) For M = 4 or 6 only. The Route Entry defined by Z is not an inter-process route entry. Action taken as above. (Code P3 or P10).

(c) For M = 4 or 6, Queued Route. If the sending of this message causes location QCOUNT of the process vector to become negative, the instruction is abandoned and the process placed in the STOPPED state (Code P2).

(d) M = 4 or 6, Segment Send. If a segment is to be sent with the message but the S bit of the CST entry selected is zero, the SEGMENT parameter of the message is set to zero. The message is sent as usual, but the sending process is placed in the STOPPED state. (Code P3).

(e) Q ≠ 1. The route entry defined by Z is not an inter-process or Autonomous I/O route entry. Action is to abandon the instruction and place the process in the STOPPED state (Code P3).

Instruction Summary

The following tables summarise the operation of the CALL instruction for the defined values of M and Q in terms of the parameters of any outgoing message and the final contents of the programme accessible registers.

Message Parameters

	M = 1	M = 4,6, V = 0	M = 4; V = 1	M = 6; V = 1
PARAMETER A	No	Register A	Register A	Register A
X	Message	Register X	Register X	Register X
Y	is	Register Y	Bits 2:15 of Register Y	Bits 2:15 of Register Y
SEGMENT	Trans-mitted	Zero	CST entry $Y_{0:1}$ with M bit set to Zero	CST entry $Y_{0:1}$ with M,W,T bits set to Zero

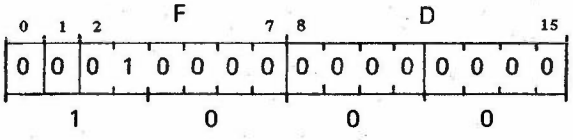
where V is the Segment Send bit in the Route Table entry.

Final Register Contents

	Q = 1	Q = 0, 2, 3 or 4,7 and Message	Q = 4,7 and No Message
S	Unchanged	Unchanged	Unchanged
L	Unchanged	Unchanged	Unchanged
E, B	Undefined	Undefined	Undefined
A	Unchanged	PARAMETER A	Unchanged
X	Unchanged	PARAMETER X	Unchanged
Y	Unchanged	PARAMETER Y	Unchanged
Z	Unchanged	ENTRY	Unchanged
C	Zero	Zero	Zero, except CZ = 1
PAST [0]	Unchanged	SEGMENT	Unchanged

3.6 THE EXIT INSTRUCTION

The EXIT instruction is a special case of the CALL instruction with field M = 0. Its format is:—



Its effect is:—

The process is placed in the STOPPED state, and a message is formed and sent to its OWNER in a similar manner to that used for dealing with error traps and status breaks. See Section 7 for details.

3.7 MECHANISATION

This sub-section describes the detailed mechanisation of the message transmission system over Queued and Fixed routes. The processes involved in sending and receiving messages of both types is described in detail, as is the structure of both the free buffer queue and incoming message queue for each process.

The details given here are invisible from within both sender and destination processes. They are presented as an aid to understanding the operation of Nucleus and to facilitate interpretation of Post Mortem dumps of system tables following catastrophic errors.

The Free Queue

Figure 7 illustrates the organisation of the free queue. Location QFREE in the SVA points to the head of the queue and location QFREND points to the tail. The POINTER fields of the buffers in the queue are used as chain pointers as shown. Buffers are removed for use from the head of the queue and are returned after use to the tail. Location QC of the SVA contains a count of the number of useable buffers in the freequeue. It is incremented by 1 whenever a buffer is returned to the queue and decremented by 1 whenever a buffer is removed. If while the system is running QC becomes negative, indicating that the freequeue is exhausted, a system error trap occurs.

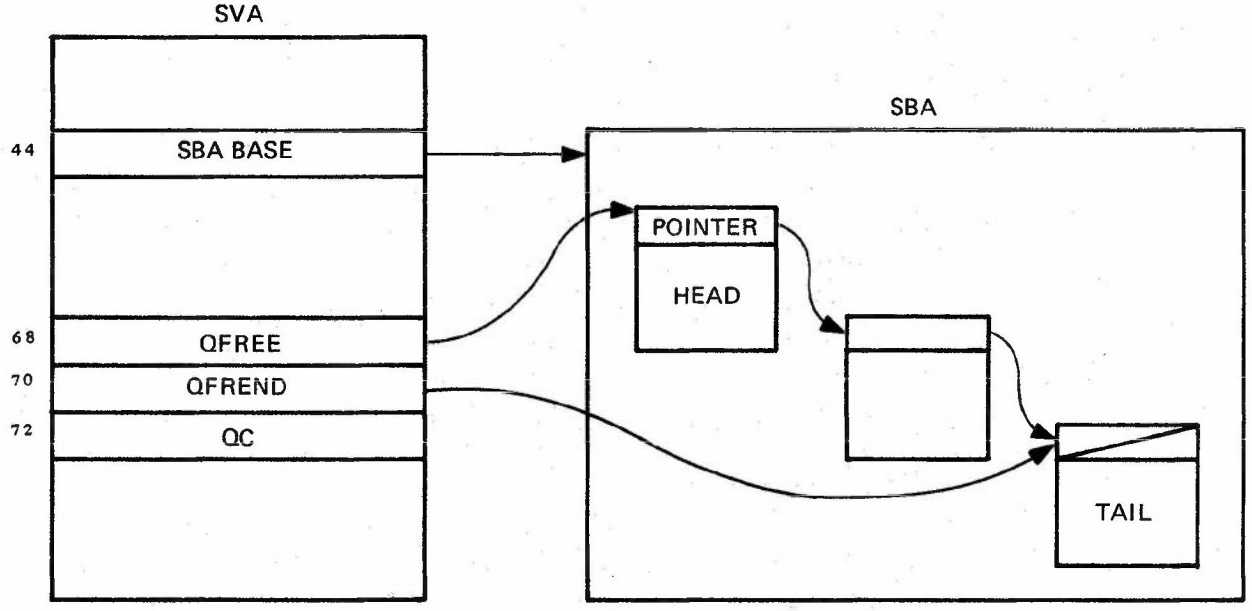


Figure 7: THE FREEQUEUE

The Incoming Message Queue

Figure 8 illustrates the organisation of the incoming message queue for each process. The QEND field of the Process Vector Entry for the process points to the tail of the queue. The POINTER fields in the buffers are used as chain pointers; note that the tail element of the queue points to the head of the queue as shown. If the queue is empty, QEND will be set to zero.

Messages are removed for processing from the head of the queue. New messages are added to the tail of the queue unless the receiving process is in the WAIT state, waiting for this particular message, when the message is added to the head of the queue.

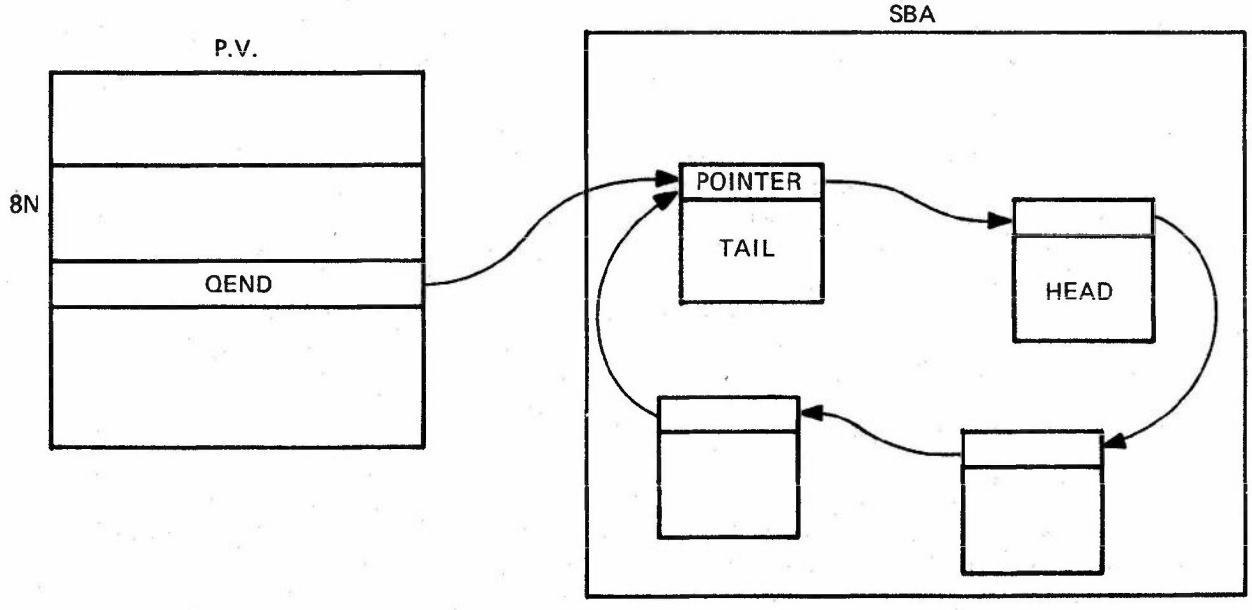


Figure 8: INCOMING MESSAGE QUEUE (PROCESS N)

Queued Message Transmission

Figure 9 illustrates the way in which the control information in the message buffer is related to the registers and tables of the sending and receiving process. The detailed procedures involved in the sending and reception of a queued messages are as follows:—

Sending a Queued Message

When a CALL instruction with the appropriate Mode is obeyed, the following steps are performed to send a queued message:—

(1) The Route number in Z is used to select a Route Table entry. For a queued message this defines the number of the destination process and the ENTRY number which is used to identify the sender to the destination process.

(2) The QCOUNT location of the PV entry of the sending process is decremented by 1. If it becomes negative, an error trap occurs and the operation is abandoned.

(3) A message buffer is dechained from the freequeue using the freequeue pointer in the SVA. At this time location QC of the SVA is decremented by 1. If, as a result, QC becomes negative a system error message is generated but the IPM is sent to its destination.

(4) The buffer so obtained is chained onto the incoming message queue for the destination process. Normally the buffer is added to the tail of the queue, using the QEND location of the PV entry of the destination process. If, however, the destination process is in the WAIT state, with WAITROUTE equal to the ENTRY number of the message, the buffer is added to the head of the queue.

(5) The SENDER location of the buffer is loaded with the process number of the sending process, and DESTINATION is loaded with a copy of the outgoing part of the route table entry.

(6) The message parameters are loaded into the buffer as described earlier.

(7) Now the state of the destination process may be changed. Specifically if it was FREE or in the WAIT state, waiting for a message with this ENTRY number, it is put into the READY state. In all other cases its state is unchanged.

(8) Finally, the state of the sending process may be changed depending on the value of the Q field of the CALL instruction.

(9) If this procedure changes the state of either sender or destination process,a Reschedule operation is performed.

Receiving a Queued Message

In due course a queued message reaches the head of the incoming message queue of the destination process, which is made READY to receive it. When the process is next given control of the processor, the following operations are performed:—

(1) The QEND pointer is used to locate the buffer in the SBA. The buffer is dechained from the queue.

(2) PARAMETERS A, X and Y are transferred into registers and SEGMENT is loaded into PAST entry 0. The ENTRY part of the DESTINATION location of the buffer is loaded into register Z.

(3) The QCOUNT location of the PV entry of the sending process is incremented by 1.

(4) The buffer is returned to the freequeue using the QFREND location of the SVA.Location QC of the SVA is incremented by 1.

(5) The destination process is run to deal with the message.

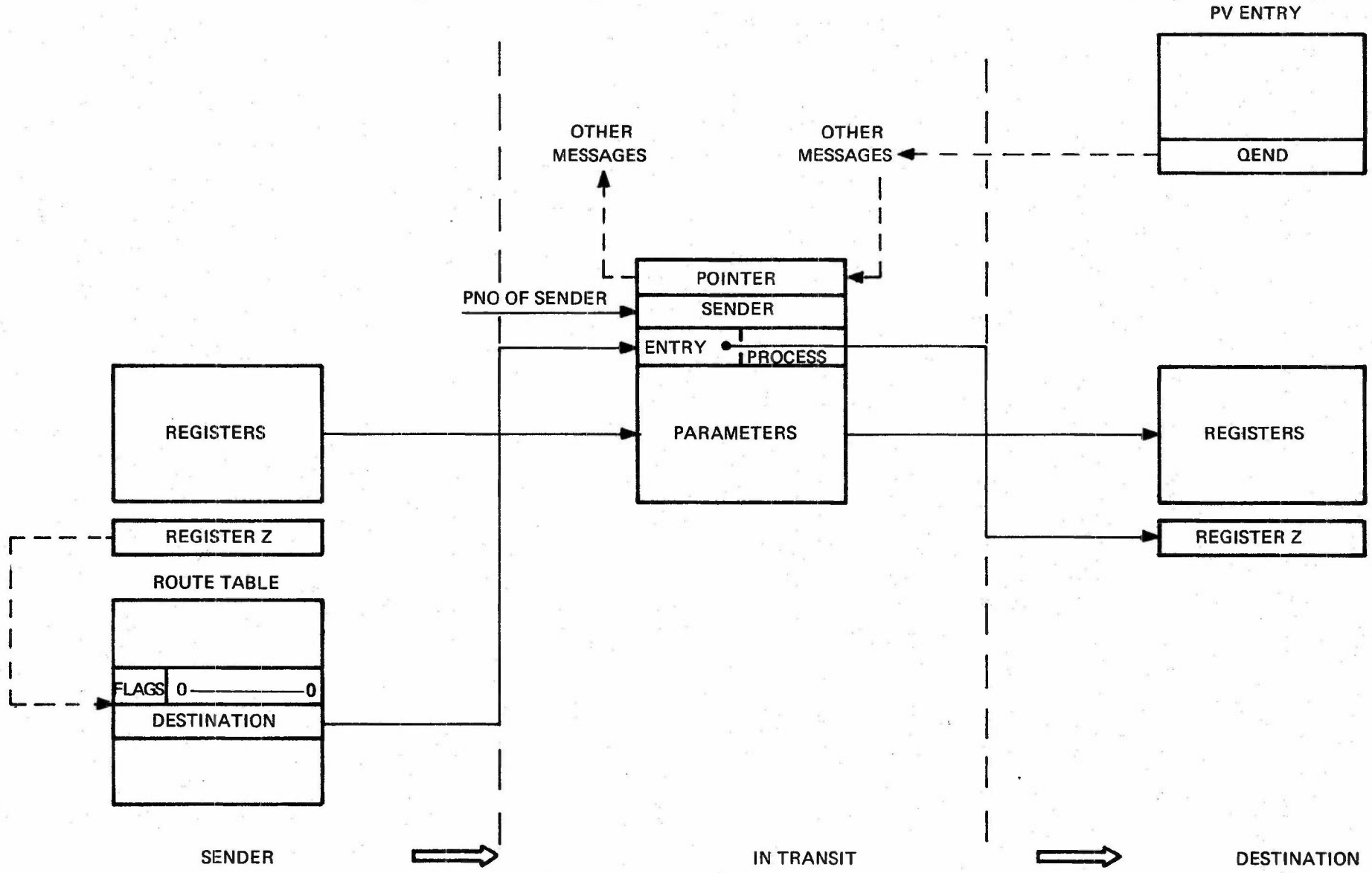


Figure 9: QUEUED MESSAGE TRANSMISSION

STIM Bits

In the fixed message transmission system the presence or absence of a message with a given ENTRY number awaiting processing is recorded by one of the STIM bits of the destination process. Sixteen STIM bits are provided, held as the first halfword of the PV entry for the process. If a message is awaiting processing with ENTRY number i, then STIM bit i of the process is reset to zero. If STIM bit i is set to a 1, it implies that there is no message outstanding with ENTRY number i.

The STIM bits are numbered in the standard way (bit 0 is the most significant bit, bit 15 the least significant). A process with no fixed messages outstanding has all 16 STIM bits set to 1. Clearly the ENTRY number of a fixed message must be less than 16, in consequence of the above.

Fixed Message Transmission

Figure 10 illustrates the relationship between the control information in the message buffer and the registers and tables of the sending and receiving process. The detailed procedures involved in sending and receiving a fixed message are as follows:—

Sending a Fixed Message

When a CALL instruction with the appropriate Mode is obeyed, the following steps are performed in order to send a Fixed message:—

(1) The route number in Z is used to select a route table entry. For a fixed message this defines the address of the first byte of the buffer in the SBA. The DESTINATION location of the buffer is set up already to define the destination of the message, and the SENDER location is set up already to define the sending process.

(2) The message parameters are loaded into the buffer exactly as for a queued message.

(3) The ENTRY part of the DESTINATION word of the buffer is used to reset to zero one of the 16 STIM bits held in the PV entry for the destination process.

(4) The state of the destination process may be changed exactly as for queued messages.

(5) The state of the sending process may be changed as determined by the Q field of the CALL instruction.

(6) If the state of either sending or destination process changes, a Rescheduling operation may be performed.

Receiving a Fixed Message

In due course the fixed message is the highest priority outstanding message, and the destination process is READY to receive it. When the process is next given control of the processor; the following operations are performed:—

(1) The ENTRY number of the message is determined by examination of the STIM bits. The bit number of the leftmost STIM bit which is set determines the ENTRY number.

(2) The STIM bit involved is set to 1.

(3) The Route Table entry defined by this ENTRY number is accessed. It contains the address of the first byte of the incoming message buffer.

(4) PARAMETERS A, X, and Y are transferred into registers and SEGMENT is loaded into PAST entry 0. The ENTRY part of the DESTINATION location is loaded into register Z.

(5) The destination process is run to deal with the message.

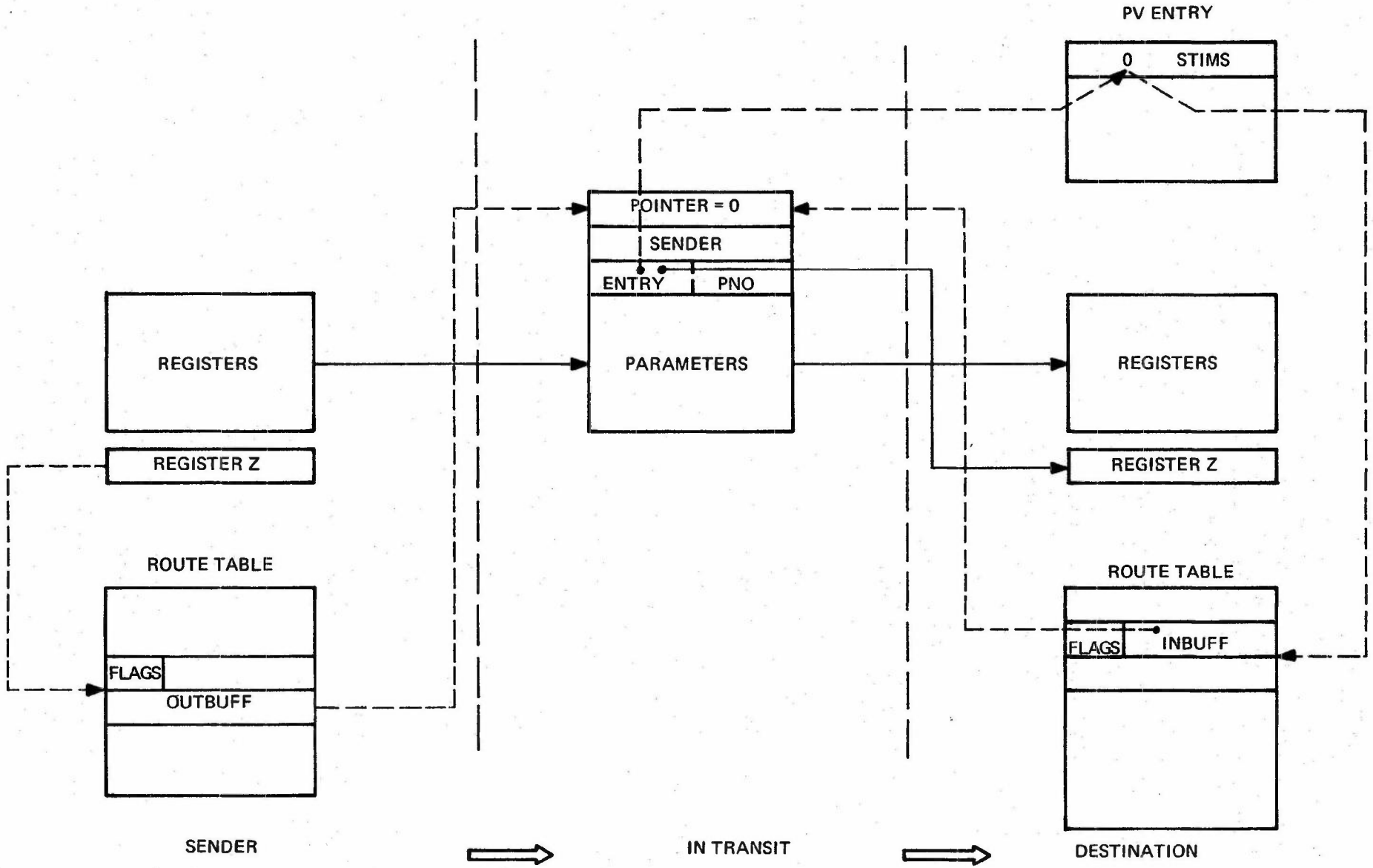


Figure 10: FIXED MESSAGE TRANSMISSION

4. THE INPUT OUTPUT SYSTEM

Input Output operations in a system are performed by Input Output Processors (IOPs) which operate autonomously. A number of IOPs may be present in a system, but one must always be present to provide the basic Input Output (I/O) for the system. This is the Basic Multiplexor Channel (BMC).

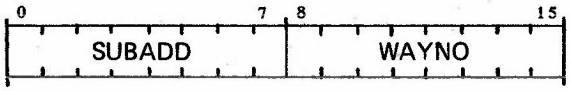
The CPU cannot perform any I/O operations itself, but is capable of stimulating IOPs to carry out I/O Transfers for it. For this purpose the CPU is connected to all IOPs by a special interface, the Command Interface. In conjunction with dedicated locations in main store, this allows the CPU to initiate I/O transfers and also to receive interrupts back from the IOPs.

4.1 PERIPHERAL ADDRESSING

In order to uniquely specify a peripheral device, it is necessary to define:—

(a) The CHANnel number. This is the number of the IOP to which the peripheral is connected. CHAN=0 specifies a device connected to the BMC. CHAN is in the range 0 to 7.

(b) The DEVice ADdress. This is a 16 bit address which defines the peripheral on the selected IOP. DEVAD is treated as two bytes; its format is:—



WAYNO — is normally sufficient to define a particular peripheral device. A peripheral may however be associated with a group of (normally consecutive) WAYNO's. For peripheral devices which are completely specified by WAYNO, the SUBADD field is unused, and may take any value.

A device which performs Autonomous I/O transfers must be completely specified by WAYNO.

SUBADD—is a SUB ADDress which may be used to specify an address or function within a selected peripheral. It is used only by devices which perform only Programmed I/O Transfers.

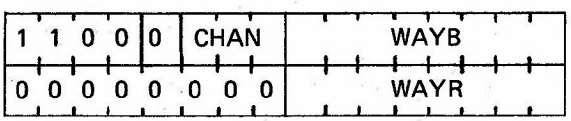
4.2 PROGRAMMED I/O INSTRUCTIONS

Programmed I/O Instructions are used to perform Programmed I/O operations. A single halfword of data is transferred between register AL in the CPU and a peripheral device connected to one of the IOPs in the system. A single format L instruction is provided, the displacement field being used to determine whether data is to be input from or output to the IOP.

In both cases register Z contains a Route number, defining one of the entries in the Route Table of the Master Segment. The Route Table Entry indicates a Programmed I/O Route; the various fields of the entry define the CHANnel number and a group of one or more peripherals with consecutive Addresses. A SELECT parameter held in register X is used to select which device of the group the Programmed I/O instruction refers to.

Programmed I/O Route

The format of a Programmed I/O Route Table Entry is as follows:—



CHAN — defines the IOP involved in the transfer.

WAYB and WAYR are used to define the Device Address, in conjunction with the SELECT parameter in register X as follows:—

The ls byte of X (X_L) is checked against WAYR. An error trap occurs if $X_L >$ WAYR.

The ls byte of X is then added to WAYB to produce the WAYNO byte of the Device Address. Any carryout is ignored.

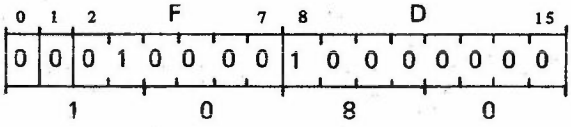
The ms byte of X is used direcly as the SUBADD byte of the Device Address.

It can be seen that WAYB and WAYR define a group of devices with consecutive WAYNOs. The address formation process is shown in Figure 11 . The X register is used to select one of the devices from the group to be used for the transfer.

PIN

The Programmed INput instruction is used to input a single byte or halfword of data from a peripheral device on one of the IOPs in the system to the registers of the CPU. At the completion of the instruction, the C register is set with a SIGNAL code indicating whether the instruction was successfully completed or not, as described later.

FORMAT:



REGISTERS: Initially Register Z contains a route number and register X a SELECT parameter. Finally, registers are set as shown in the table below.

EFFECT: The specified IOP is caused to cease operation by use of the Command Interface. The device address formed from the WAY field of the route table entry and register X is sent to the IOP.

The IOP takes the necessary action and returns a halfword of Data, and a SIGNAL code to the CPU. The data is placed in the AL register and the SIGNAL code in the C register.

Finally, the IOP is released to resume operation.

TRAPS: Error traps occur if the route number in Z is greater than or equal to the number of routes in the route table of the process, or if the route is not an I/O route, or if the least significant byte of register X is greater than RANGE (Code P5).

NOTE: If the route defined by register Z is an autonomous I/O route, the instruction is interpreted as an RN instruction.

REGISTER TABLE:

REGISTER	INITIALLY	FINALLY
AM	—	Zero
AL	—	Data Input*
X	SELECT	N.A.
Y	—	N.A.
Z	Route	N.A.
C	—	SIGNAL
B,E	—	Undefined

* If the transfer is unsuccessful (SIGNAL ≠ 0) then register A is undefined.

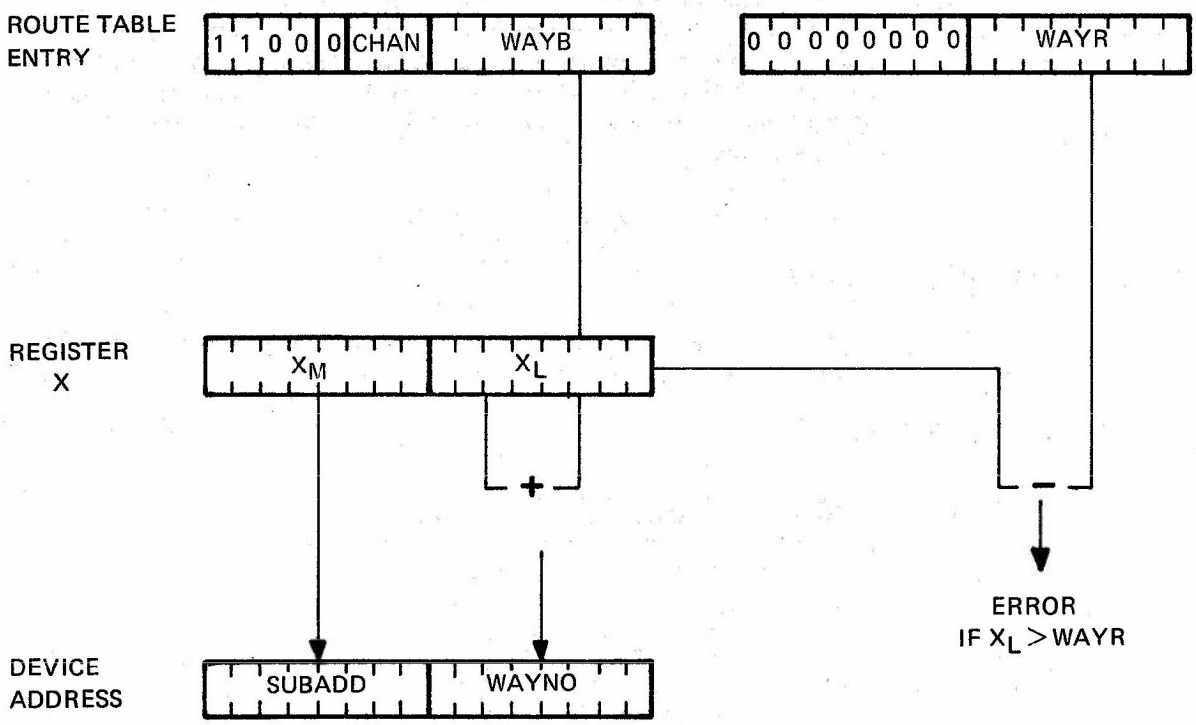
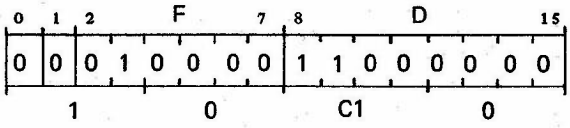


Figure 11: PROGRAMMED TRANSFER DEVICE ADDRESSES

POUT

The ProgrammedOUTput instruction is used to output a single halfword of data from the registers of the CPU to one of the IOPs in the system. At the completion of the instruction, the C register is set with a SIGNAL code indicating whether the instruction was successfully completed or not.

FORMAT:



REGISTERS: Initially Register Z contains a route number, register X a select parameter, and register AL the Data to be output. Finally, registers are as shown in the table below.

EFFECT: The specified IOP is caused to cease operation, by use of the Command Interface.

The device address formed from the WAY field of the route table entry and register X, and the Data held in register AL, are sent to the IOP. The IOP then takes the necessary action and returns a SIGNAL code to the CPU. The IOP also may return a halfword of Data at this time. The SIGNAL code is placed in the C register and if Data was returned it is placed in register AL. Otherwise AL is left unchanged.

Finally, the IOP is released to resume operation.

TRAPS: As for PIN

NOTE: If the route defined by register Z is an autonomous I/O route, the instruction is interpreted as a CT instruction (see sub-section 4.3).

REGISTER TABLE:

REGISTER	INITIALLY	FINALLY
AM	–	Zero
AL	Data	May Change
X	SELECT	N.A.
Y	–	N.A.
Z	Route	N.A.
C	–	SIGNAL
B,E	–	Undefined

The SIGNAL Code

On completion of Programmed I/O operations, and some Autonomous I/O operations (see later sections), a SIGNAL code is written into the CA, COF, CN, and CZ bits of the C register, and the FM bit is reset to zero. The meaning of the possible codes is dependent on the IOP. For the BMC the codes used are:–

CN	CZ	COF	CA	MEANING
0	0	0	0	Transfer accepted
0	0	1	1	Unspecified device rejection
0	1	1	1	Invalid Function
1	0	1	1	Device busy

CN	CZ	COF	CA	MEANING
0	0	1	0	Device timeout
1	1	1	0	Store parity failure
1	1	1	1	Store timeout
0	1	1	0	Overall timeout

A PIN or POUT instruction which is not successful may leave register A in an undefined state.

4.3 AUTONOMOUS INPUT/OUTPUT

Autonomous Input/Output operations involve the transfer of bytes or halfwords of information between peripheral devices connected to the IOPs in the system, and buffer areas in main store. Such transfers are carried out entirely by the IOPs, and do not involve use of the CPU.

Each IOP is provided with up to 256 WAYs for autonomous transfers. Each peripheral device connected to an IOP which may perform autonomous I/O transfers is allocated one or more WAY NUMBERS (WAYNOs). As described earlier (section 4.1) the 8 bit WAYNO forms the ls 8 bits of the DEVAD for the device.

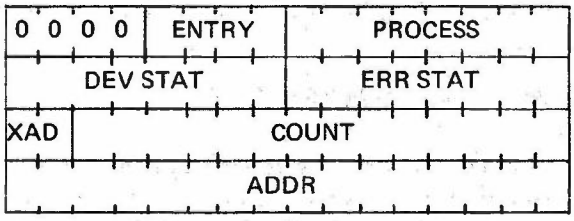
Associated with each WAY is an eight byte item of main storage, which contains parameters which control autonomous I/O transfers on that WAY. This item is known as the Way Control Block (WCB). The WCBs for an IOP occupy contiguous locations in main storage, starting at an address which is preset for each IOP individually. This address is known as the WAYBASE for the IOP. The WCB for WAY N on an IOP starts at actual byte address:—

WAYBASE + 8*N

Note that all WCBs for all IOPs must be located in the first 64 Kbyte of main store. For the BMC, WAYBASE is 128.

Format of the WCB

The WCB has the format:—



The first four bytes are used for handling interrupts from the device which uses the WCB. The second four bytes define the area of main store to be used as a buffer for the transfer.

The fields of the WCB are as follows:—

COUNT — Defines the size of the buffer area in bytes. As a transfer progresses, COUNT is normally decremented by 1 for each byte that is transferred. At any time during a transfer, COUNT defines the number of bytes of the buffer that have not been transferred.

XAD,ADDR — Together form an 18 bit actual address, XAD forming the ms 2 bits and ADDR the ls 16 bits of the address. This is the address of the last byte in the buffer. It is known as BUFFAD.

Note that COUNT, XAD, ADDR between them define a buffer in store, where
First address in buffer = BUFFAD — COUNT + 1
Last address in buffer = BUFFAD
Number of bytes in buffer = COUNT

If COUNT = 0, the buffer is said to be exhausted and no further transfer may take place.

ENTRY,
PROCESS — Form the DESTINATION halfword of the WCB. This is used to route interrupts arising on the WAY to the process which should be run to deal with the interrupt condition. For further details see Section 4.6.

DEVSTAT,
ERRSTAT– Form the STATUS halfword of the WCB. This is used to carry status information in the event of an interrupt. For further details see Section 4.6.

Autonomous Transfer Operation

Autonomous Transfers generally are carried out in 4 phases:–

(1) Initialisation – During this phase, the WCB for the WAY in question is initialised to define the buffer area to be used in the transfer.

(2) Initiation – During this phase a Command is issued to the IOP to cause the peripheral device involved to commence the transfer.

(3) Transfer – During this phase the IOP transfers data between the buffer area and the peripheral device.

(4) Interrupt – At the end of the transfer, or in the event of an error condition(s) arising, the IOP signals an interrupt to the CPU, so that the appropriate process can take the necessary action.

The Transfer phase is carried out by the IOP, and does not involve the CPU at all. It is not considered further here. For details refer to the description of the appropriate IOP.

The Interrupt phase is described in detail in Section 4.6.

The Initialisation and Initiation phases are performed by Autonomous I/O Instructions, which are described in the following section.

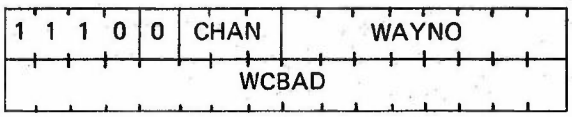
4.4 AUTONOMOUS I/O INSTRUCTIONS

Autonomous I/O Instructions are used to perform the Initialisation and Initiation phases of Autonomous I/O Transfers. Also an instruction is provided to allow a process to monitor the progress of an ongoing transfer. A single format L instruction is provided, the Displacement field being used to further define the operation to be performed.

In all cases, register Z contains a Route Number, defining one of the entries in the Route Table of the Master Segment. The Route Table Entry must indicate an Autonomous I/O Route; the various fields of the entry defining the CHANnel number of the IOP and the WAYNO of the device to be used for the transfer.

Autonomous I/O Route

The Format of an Autonomous I/O Route Table Entry is as follows:–



CHAN – defines the IOP involved in the transfer.

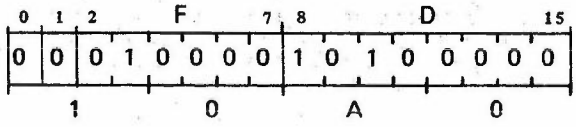
WAYNO – defines the device address. The SUBAD field of the Device Address used is set to zero.

WCBAD – defines the actual address of the first byte of the WCB for the WAY in question.

LWCB

The Load Way Control Block instruction is used to initialise a WCB in preparation for an Autonomous Transfer. The instruction does not cause the transfer to be initiated. The number of bytes in the transfer and the position of the data buffer in virtual store are defined by the content of registers X and Y. These registers are used to initialise the WCB appropriately.

FORMAT:



REGISTERS: Initially Register Z defines a route number, Register X defines a transfer length (in bytes) and Register Y the virtual address of the start of a data buffer in store. Finally, registers are set as indicated in the table below.

EFFECT: The IOP specified by the CHAN field of the route table entry is caused to cease operation, by use of the Command Interface.

The WCB defined by the route table entry is set up as follows:—

The least significant 14 bits of register X are loaded into the COUNT field of the WCB.

The virtual address given by Y + X -1 is computed, and mapped by Nucleus into the corresponding 18 bit actual address. The two most significant bits of this address are loaded into the XAD field of the WCB, the remaining 16 bits are loaded into the ADDR field. Finally, the IOP is released to continue operation.

TRAPS: Error traps occur if:—

Register Z contains a route number greater than or equal to the number of routes of the process.
The route defined by Z is not an Autonomous I/O route.
The most significant 2 bits of register X are non-zero initially.
Virtual addresses Y and Y + X - 1 are not valid.
The virtual addresses Y and Y + X– 1 are in different segments.
The segment involved does not have Transfers permitted (e.g. the T bit is not set in its CST entry (Code P5).

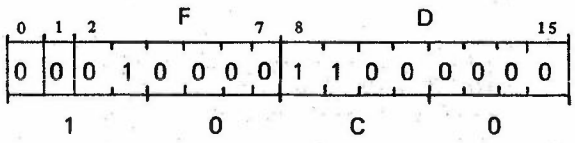
REGISTER TABLE:

REGISTER	BEFORE	AFTER
AM	–	N.A.
AL	–	N.A.
X	Transfer Length	N.A.
Y	Buffer Address	N.A.
Z	Route Number	N.A.
C	–	Zero
B,E	–	Undefined

TRIP

The Transfer Input/Output instruction is used to initiate an autonomous I/O transfer in circumstances where it is unnecessary to initialise the WCB. It causes a Command halfword to be sent to the IOP, for subsequent output to the peripheral device it is required to initiate. At the completion of the instruction, the C register is set with a SIGNAL code indicating whether the transfer was successfully initiated or not, as described in Section 4.2.

FORMAT:



REGISTERS: Initially Register Z contains a route number and Register AL a halfword Command. Finally, registers are set according to the table below.

EFFECT: The specified IOP is caused to cease operation by use of the Command Interface. The IOP is sent the Command from register AL, and the device address given by the WAY field of the route table.

The IOP then takes the necessary action, and returns a SIGNAL code to Nucleus. The SIGNAL code is loaded into the C register.

Finally, the IOP is released to resume operation.

TRAPS: An error trap occurs if the route number in Z is greater than or equal to the number of routes of the process, or if the route is not an I/O route. (Code P5).

NOTE: If the route specified is a programmed I/O route, the instruction is interpreted as a POUT instruction.

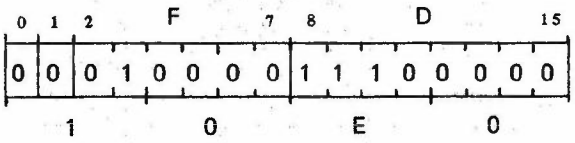
REGISTER TABLE:

REGISTER	BEFORE	AFTER
AM	–	Zero
AL	Command	May Change
X	–	N.A.
Y	–	N.A.
Z	Route	N.A.
C	–	SIGNAL
B,E	–	Undefined

LWT

The Load Way control block, command Transfer instruction combines the functions of a LWCB instruction followed by a TRIP instruction. The effect of the instruction is to Initialise a WCB in preparation for an autonomous transfer, and to send a Command halfword to the IOP involved to initiate the transfer.

FORMAT:



REGISTERS: Initially Register Z defines a route, Register X and Register Y a buffer area in store (as for LWCB) and Register AL a Command (as for TRIP)

Finally, registers are set as indicated in the table below.

EFFECT: The IOP involved is caused to cease operation by use of the Command Interface The relevant WCB is loaded using registers X and Y as described for the LWCB instruction. Finally a Command halfword and device address are sent to the IOP which takes action as described for the TRIP instruction. The IOP is released to resume operation.

TRAPS: As for LWCB

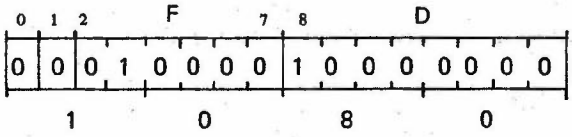
REGISTER TABLE:

REGISTER	BEFORE	AFTER
AM	–	Zero
AL	Command	May Change
X	Transfer Length	N.A.
Y	Buffer Address	N.A.
Z	Route Number	N.A.
C	–	SIGNAL
B,E	–	Undefined

RW CB

The Read Way control block instruction is used to determine the status of a specified peripheral transfer. It causes the DEV STATUS, ERROR STATUS, and COUNT fields of the WCB to be loaded into registers. The COUNT determines how far a transfer has progressed by indicating the number of bytes of data remaining to be transferred, while the STATUS information is placed in the WCB as a result of interrupts, as described in Section 4.6.

FORMAT:



REGISTERS: Initially Register Z defines a route number. The route must be an Autonomous I/O route.

Finally, registers are set as indicated in the table below.

EFFECT: The IOP specified by the CHAN field of the route table entry is caused to cease operation, by use of the Command Interface.

The DEV and ERROR STATUS bytes are copied from the relevant WCB into register AL. Then these bytes of the WCB are cleared.

The COUNT field of the WCB is copied into register C. Finally the IOP is released

TRAPS: An error trap occurs if register Z is greater than or equal to the number of routes of the process.

An error trap occurs if the route is not an I/O route. (Code P5).

NOTE: If the route involved is a Programmed I/O route the instruction is interpreted as a PIN instruction.

REGISTER TABLE:

REGISTER	BEFORE	AFTER
AM	–	Zero
AL	–	STATUS
X	–	COUNT
Y	–	N.A.
Z	ROUTE	N.A.
C	–	Zero
B,E	–	Undefined

4.5 STATE CHANGE INSTRUCTION

As already described, the CALL instruction with M = 2 can be used to change the state of a process. The CALL instruction is used in conjunction with a route number held in register Z. The effect of the instruction has been defined where the route defined by Z is an inter-process route, but also it can be used in conjunction with I/O routes as defined below:—

Q = 0 (GO FREE) can be used with any valid route number in register Z. The type of route is irrelevant.

Q = 1 (Continue to RUN) can be used with any valid route number in Z.

Q = 2 (WAIT, PASS priority) is meaningful for inter-process routes only. It can be used for Autonomous I/O routes, when it has the same effect as Q = 3. It must not be used with a Programmed I/O route. An error trap occurs if this is attempted.

Q = 3 (WAIT) can be used in conjunction with an Autonomous I/O route, when it causes the process to WAIT until a specific I/O interrupt arrives.

Q = 4 (Conditional FREE). Comments as for Q = 0.

Q = 7 (Conditional WAIT). Comments as for Q = 3.

4.6 INTERRUPTS

Interrupts are used to inform a process that a condition has arisen in the I/O subsystem that requires its attention. The usual condition is either that a peripheral device has completed an operation previously initiated by that process, or that some error condition (such as a parity error on the peripheral's recording media) has arisen which the process should be aware of.

Interrupts are handled in three distinct phases which may be separated in time from each other.

Interrupt Generation Phase

When an IOP is made aware that one of the devices it controls wishes to generate an interrupt, it writes a status byte into DEV STAT location of the WCB for the device in question. This byte normally defines the reason for which an interrupt has been generated, and normally is input to the IOP from the device. The IOP also records the address of the WCB in a store location reserved for this purpose, called the INTADDR location, and then uses the Command Interface to signal an interrupt to the CPU. The IOP now continues operations while waiting for the CPU to respond, but must not generate any further interrupts until it has done so.

Interrupt Recording Phase

At the next interruptable point, normally when the IOP has completed the instruction or Nucleus operation it was performing when the interrupt was signalled to it, Nucleus is entered to deal with the interrupt.

Nucleus uses the INTADDR location previously set up by the IOP to locate the WCB, and uses the DESTINATION halfword of the WCB to determine the destination process and the ENTRY number of the interrupt message. From this point, operation is much the same as it some other process had generated a fixed message for the destination process. The STIM bit corresponding to the ENTRY number is reset to zero, and the state of the destination process is changed to READY if it was previously FREE or in the WAIT state waiting for this interrupt message. If the state of the destination process changes, a Nucleus reschedule operation is performed. Otherwise control reverts to the previously operating process. In any case, the CPU signals acceptance of the interrupt to the IOP, which may now generate further interrupts if necessary.

Interrupt Completion

Either immediately or some time later the destination process is selected to run to process the interrupt message. The process of loading an interrupt message is very much the same as the process of loading a fixed message. Immediately after the RUN state has been entered, the program accessible registers are set as follows:—

REGISTER	CONTENTS
S	Unchanged
L	Unchanged
E,B	Undefined
A	AM = 0, AL = STATUS halfword from WCB
X	bits 0 : 1 = Zero, bits 2 : 15 = COUNT field from WCB
Y	Undefined
Z	ENTRY
C	Zero

No other registers or Master Segment entries are changed. Note in particular that unlike reception of a fixed message, PAST entry 0 is unaffected.

After the message parameters have been loaded into registers, the STATUS halfword of the WCB is cleared by Nucleus.

4.7 INTERRUPT MECHANISATION

The interrupt process is illustrated in Figure 12 which should be compared with the similar figure (Figure 10) for Fixed Message transmission.

To summarise, the operation is as follows:—

Generating an Interrupt

(1) The IOP writes a status byte to the WCB.

(2) The IOP places the address of the WCB in the INTADDR location, and signals an interrupt to the CPU.

(3) The CPU locates the WCB using the INTADDR pointer.

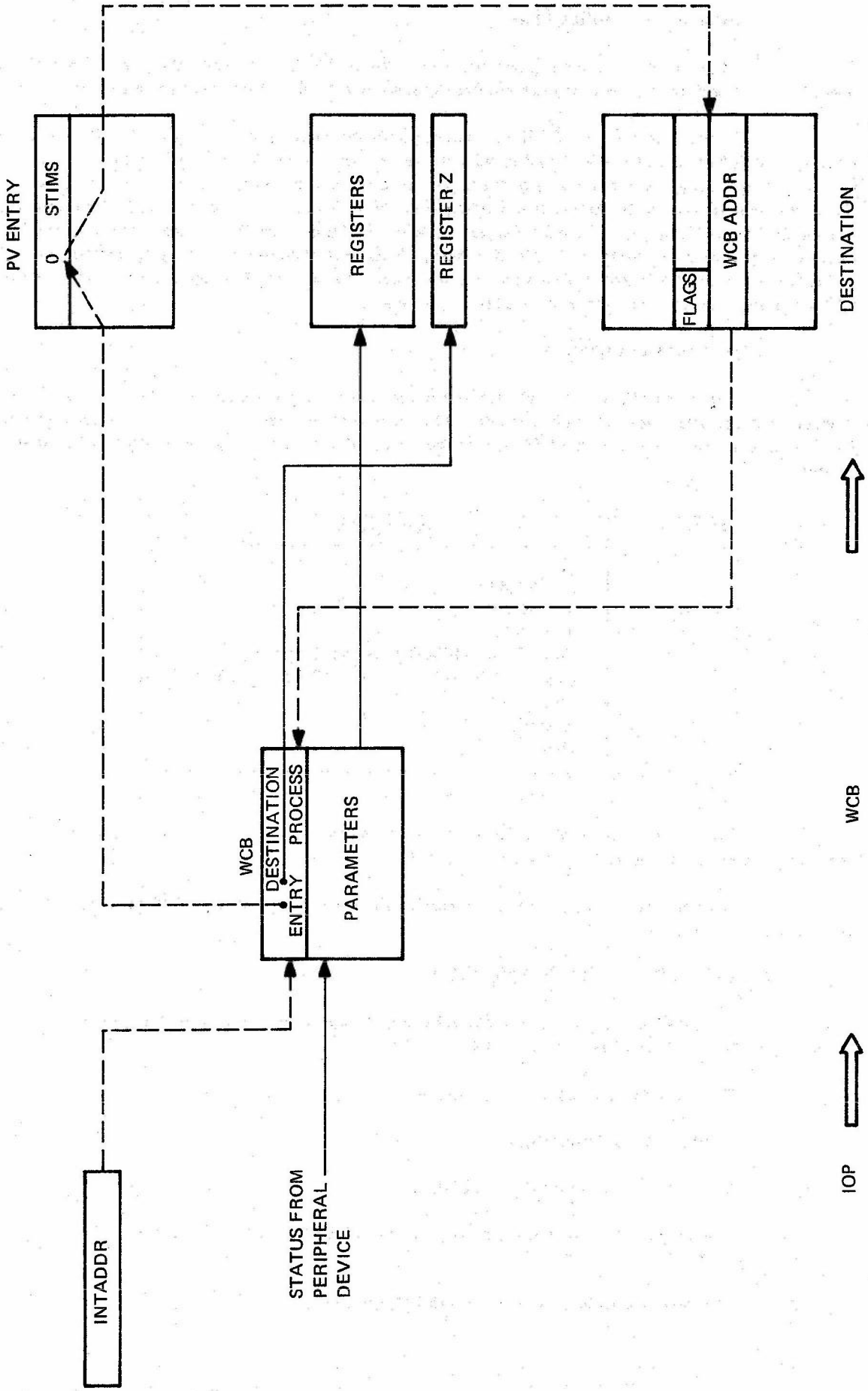


Figure 12: SENDING AN INTERRUPT MESSAGE

(4) The ENTRY part of the DESTINATION word of the buffer is used to reset to zero one of the 16 STIM bits of the destination process.

(5) The state of the destination process may be changed exactly as for fixed or queued inter-process messages.

(6) If the state of the destination process changes, a Reschedule operation is performed.

Receiving an Interrupt

In due course the interrupt message becomes the highest priority outstanding message, and the destination process is READY to receive it. When the process is next given control of the processor the following operations are carried out:—

(1) The ENTRY number of the message is determined by examination of STIM bits. The bit number of the leftmost STIM which is set determines the ENTRY.

(2) The STIM bit involved is set to 1.

(3) The Route Table Entry defined by this ENTRY number is accessed. It contains the address of the first byte of the interrupt WCB.

(4) The STATUS halfword and COUNT field are transferred from the WCB into registers. The ENTRY part of the DESTINATION location is loaded into register Z.

(5) The destination process is run to deal with the message.

4.8 SPECIAL EFFECTS

Whilst the majority of Input Output operations conform to the pattern laid down in Section 4.3. (Initialisation – Initiation – Transfer – Interrupt), variations on this pattern are possible and may be used as noted in this section.

(a) It is possible to re-initialise a WCB for a WAY on which a Transfer is already in progress. This has the effect of instantaneously changing either the position or the size (or both) of the buffer in main store used for the transfer. Interlocks are provided to ensure that no data items are lost or corrupted under such a circumstance. This feature may be used, for instance, to ensure that a continuously running input device never runs out of buffer area in main store.

(b) It is possible to Initiate a transfer on a peripheral which is busy on a transfer. The effect of doing this depends on the IOP and the peripheral device. In many cases the effect is to cause the peripheral to stop the transfer it was engaged on. This feature can be used to stop a transfer under program control with an appropriately designed peripheral.

(c) Finally, it is possible that a device may generate Interrupts before it has completed a transfer. Arrangements must be made to intercept such interrupts and ensure that all earlier interrupts have been received before later interrupts are generated.

(d) The IOP, in the course of its operation, reads from and writes to the WCB, and sometimes holds fields from the WCB in its registers. Nucleus causes the IOP to cease operations at the earliest convenient moment, and restore all WCB's, before Nucleus makes any access to a WCB. Access to a WCB using other than Nucleus instructions is possible but can give unexpected results if the IOP is operating also on the same WCB.

5. SEMAPHORE SYSTEM

Nucleus provides for the operation of binary semaphores. Normally semaphores are used to prevent two or more processes from performing critical operations, such as accessing a shared data segment, at the same time.

A semaphore is a halfword data item held in any segment shared by all the processes which use it. A segment containing a semaphore must have both Read and Write access permission. The semaphore can be in one of two states; in the released state, it indicates that no process is performing a critical operation, whilst in the claimed state, it indicates that one of the processes is so engaged.

Two operations can be performed on semaphores by a process, a claim operation and a release operation. If a semaphore is in the released state and a process performs a claim operation on it, then the semapho is put in the claimed state and the process continues to run. The semaphore remains in the claimed state until a corresponding release operation is performed, normally (but not necessarily) by the process which made the original claim.

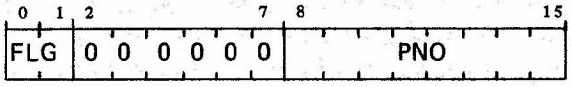
If a process attempts to claim a semaphore which is already in the claimed state, the claim operatio does not succeed. The process is put into a special state, the HELD state, and it is not allowed to run until the semaphore has been returned to the released state. At this time the unsuccessful process is returned to the RUN state, and may again attempt to claim the semaphore.

It can be seen that if several processes attempt to claim the same semaphore, the first is successful, and is allowed to continue running, whilst the others are HELD by the semaphore, and so are not allowed to run.

5.1 SEMAPHORE STRUCTURE

Thus far, two states of a semaphore have been distinguished, the released state and the claimed state. To mechanise the semaphore system it is necessary to further subdivide the claimed state into two substates, claimed-simple and claimed-complex. In the simple substate, the semaphore has been claimed, but no further attempts have been made to claim it. In the complex substate, the semaphore is claimed and other processes have attempted to claim it unsuccessfully and, therefore, are in the HELD state.

The format of a semaphore is as follows:—



FLG — is a two bit field defining the state of the semaphore. FLG = 0 is not a permitted combination.

PNO — is an eight bit field used to hold a Process Number. Its usage is defined below.

Note that bits 2 to 7 must all be zero.

The three semaphore states are:—

(a) *Semaphore RELEASED*

This is denoted by FLG = 1.

In this state the PNO field is unused and normally contains zero.

(b) *Semaphore CLAIMED - SIMPLE*

This is denoted by FLG = 3.

In this state, the PNO field of the semaphore contains the number of the process which has claimed it successfully. This process is said to be holding the semaphore.

(c) *Semaphore CLAIMED - COMPLEX*

This is denoted by FLG = 2

In this state, the PNO field of the semaphore contains the number of the process which last attempted (unsuccessfully) to claim it. As will be seen later, this points to a chain of HELD processes, ending with the process holding the semaphore.

Semaphore Corrupt

Before any semaphore operation is performed, the location to be used as a semaphore is checked to determine whether it has the structure of a semaphore. The semaphore is said to be corrupt if either the FLG field is zero, the PNO field contains a number greater than the total number of processes in the Process Vector, or bits 2 to 7 are non-zero.

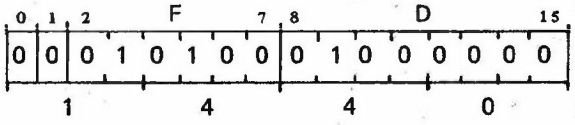
5.2 SEMAPHORE INSTRUCTIONS

The Semaphore Instructions permit a process to operate semaphores as described above. A single format L instruction is used, the Displacement field being used to specify the operation to be performed.

CLM

The semaphore CLM instruction is used to attempt to claim a semaphore. If the claim is unsuccessful, the process is placed in the HELD state and is not allowed to run until the semaphore has been released.

FORMAT:



REGISTERS: Initially Register Z contains the virtual address of a semaphore.
No registers are affected by the instruction.

EFFECT: The semaphore pointed to by Register Z is examined. If it is in the released state, it is put into the claimed-simple state, and the PNO field is set to the number of the process issuing the instruction.

If it is in the claimed (simple or complex) state, it is put into the claimed-complex state. The process is put into the HELD state, and is added to the list of HELD processes associated with the semaphore as described later. If the CLAIM is unsuccessful, the register S is decremented by two, so that when the process is next run, the CLM instruction is obeyed again in a further attempt to claim the same semaphore. Figure 13 illustrates the procedure.

TRAPS: An error trap occurs if the content of Register Z is not a valid virtual address, or if the segment containing the semaphore does not have Read and Write access permission. (Code PO). An error trap occurs if the semaphore is corrupt. (Code P7).

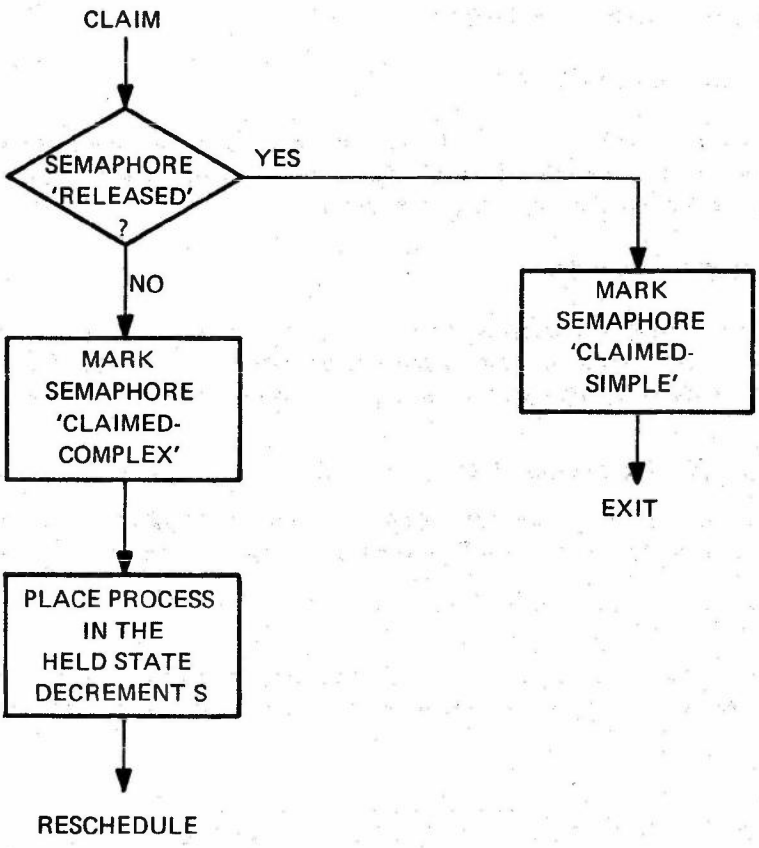
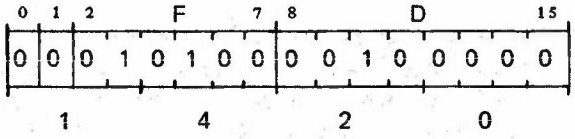


Figure 13: SEMAPHORE CLAIM

CCLM

The CLM Conditionally instruction is used instead of CLM where it is important not to hold up the process if the claim is unsuccessful. If the claim is successful, operation is as for the CLM instruction. If the CCLM is unsuccessful, the semaphore is not changed, but conditions bit CZ is set to a one.

FORMAT:



REGISTERS: Initially Register Z contains the address of a semaphore. CN, CA, COF are all reset to zero by the instruction. If the semaphore is in the released state, CZ is reset to zero. If the semaphore is in either claimed state, CZ is set to one. No other registers are affected.

EFFECT: The semaphore pointed to by Register Z is examined. If it is in the released state, the effect is as for CLM, and also the conditions bits CZ, CN, CA, COF are all reset to zero.

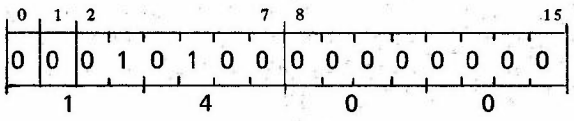
If the semaphore is in either claimed state, then conditions bits CN, CA, COF are reset to zero, and bit CZ is set to one. The state of the semaphore is not changed.

TRAPS: As for CLM.

REL

The semaphore RELease instruction is used to return a claimed semaphore to the released state. Any processes HELD on the semaphore are allowed to run again.

FORMAT:



REGISTERS: Initially Register Z contains the address of a semaphore. No registers are affected by the instruction.

EFFECT: The semaphore pointed to by Register Z is examined. If it is released already, or is claimed-simple then the semaphore is placed in the released state with bits 2 to 15 zeroed.

If it is claimed-complex then it is placed again in the released state, but also the associated list of HELD processes is used to return these processes to the RUN state, Figure 14 illustrates the above procedure.

TRAPS: As for CLM.

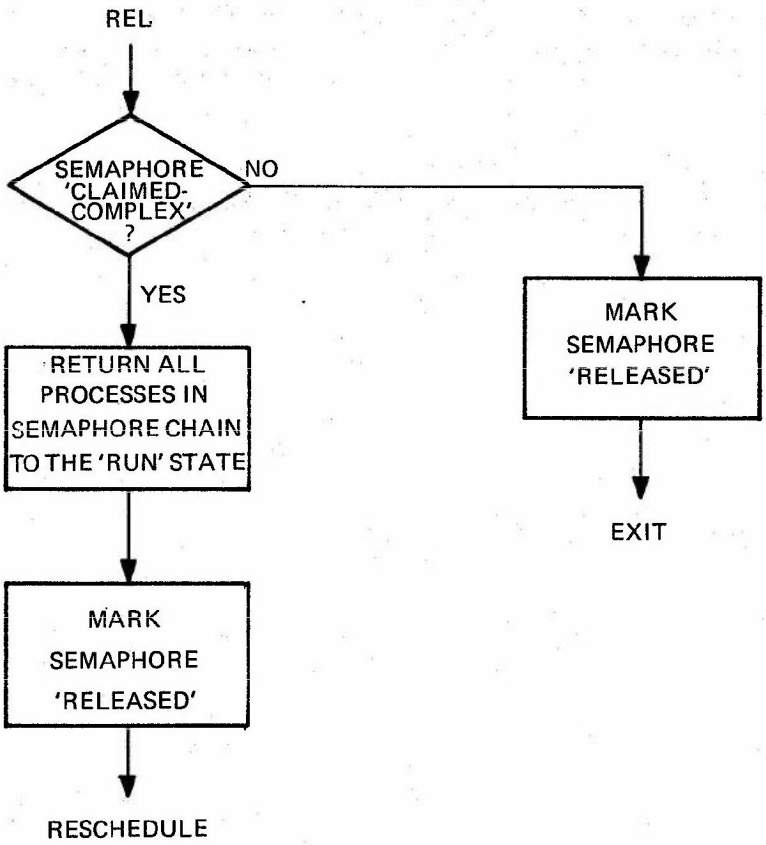


Figure 14: SEMAPHORE RELEASE

5.3 THE SEMAPHORE CHAIN

Consider a semaphore, originally in the released state. If a process (say process A) CLAIMS it, the semaphore is set to the claimed-simple state (FLG = 3) with the PNO field set to the number of the holding process (A).

If a second process, process B, now attempts to CLAIM the semaphore, its claim is unsuccessful. The semaphore is set to the claimed-complex state (FLG = 2) with the PNO field set to (B). Also, process B is put into the HELD state, and the TRANSFER field of its Process Vector entry is set to point to the process holding the semaphore (A). This is shown in figure 15.

Any subsequent process (say process C) attempting to CLAIM the semaphore is dealt with in the same way, that is:—

(1) The semaphore is set to the claimed-complex state with the PNO field set to the number of the process (C).

(2) Process C is put into the HELD state, and the TRANSFER field of its process vector entry is set to the number originally contained in the PNO field of the semaphore.

(3) The process vector entry for the first unsuccessful claimant, B, is distinguished by a marker bit from the entries for C and subsequent claimants. This bit is called the "Last" marker, L.

This chain of processes, starting with the process defined by the PNO field of the semaphore and ending with the process holding the semaphore, is called the semaphore chain, shown in Figure 15.

The chain is used for two purposes:—

(1) During scheduling, the process selector uses the TRANSFER fields of the PV entries to follow the chain and so pass priority from the HELD processes to the holding process. This is important, since the HELD processes are normally of higher priority than the holding process. See Section 6 for details.

(2) When the semaphore is released (by the holding process, normally) the chain is used to discover all processes which were HELD on the semaphore. All the processes are returned to the RUN state so that they may again attempt to CLAIM the semaphore.

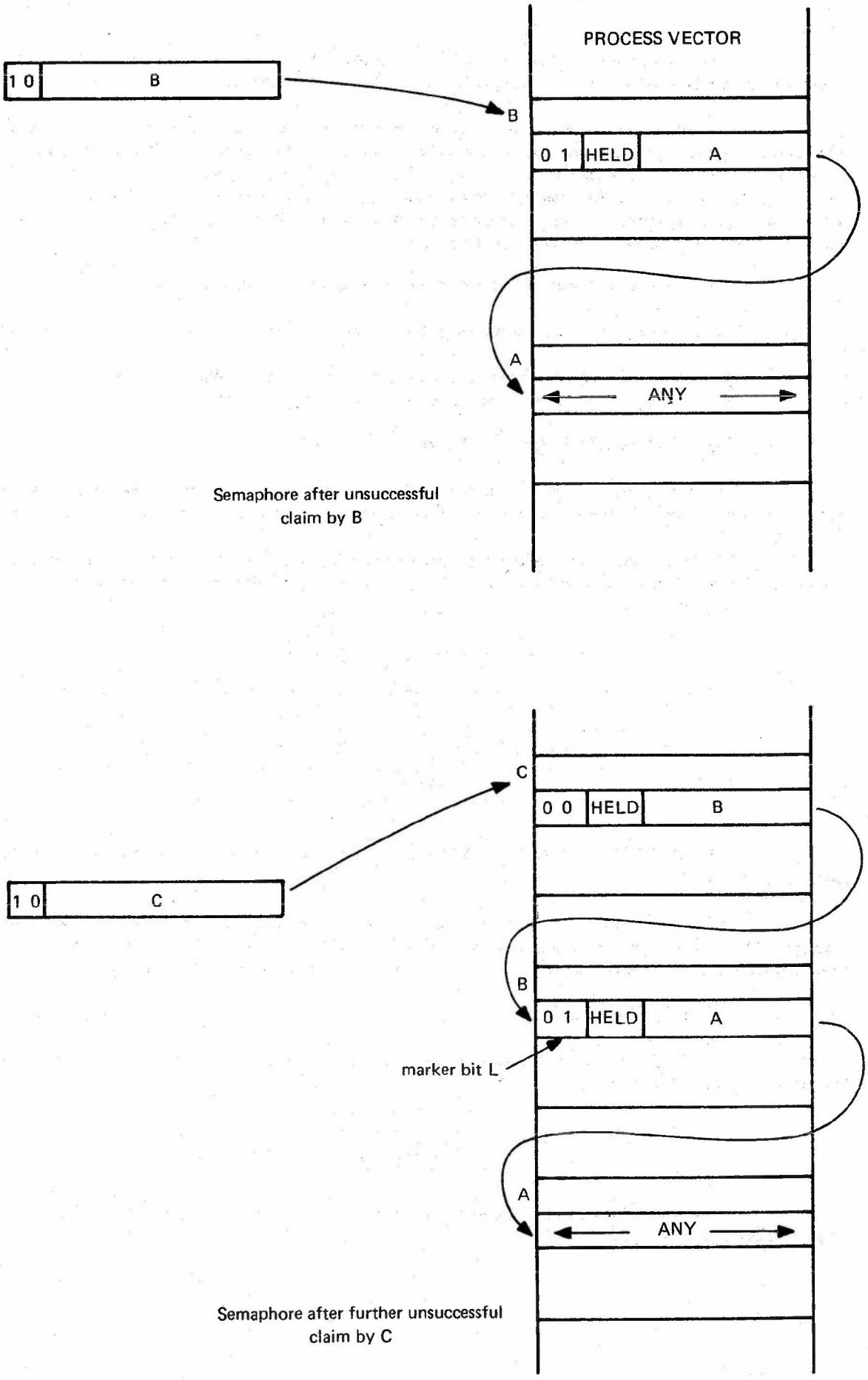


Figure 15: SEMAPHORE CHAINS

The function of the Process Selector part of Nucleus is to ensure that at any time the most urgent process which has useful work to do (i.e. is 'active') is in control of the processor.

The Process Selector is invoked to carry out a reschedule operation whenever an event occurs which either makes a previously inactive process active or an active process inactive. The result of a Process Selector operation is to select a new process to have control of the processor. This new process may be the same as that previously operating, in which case it continues in control of the processor; more usually, a different process is chosen to run, and the process change mechanism is used to save the working registers of the previously operating process and load working registers for the newly selected process.

The main circumstances under which the process selector is invoked are:—

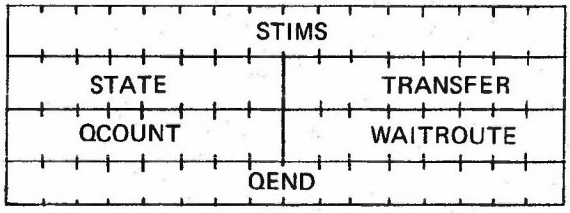
(1) The currently operating process obeys a CALL instruction which results in it becoming inactive.

(2) A process other than the current process becomes active. This may happen as a result of a message being sent to it by the current process, or an I/O interrupt arising for it.

6.1 PROCESS VECTOR AND SCAN BITS

The Process Vector is a table in main store held in a single segment. Each process in the system has an entry in the process vector. All entries are 8 bytes long, and the entry for process N starts at byte 8*N of the PV.

The PV contains information defining the state of all processes in the system. This information is used by the Process Selector in choosing the correct process to operate. The format of a PV entry is:—

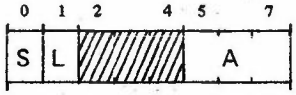


Only the STATE, TRANSFER, and QEND fields of the PV entry are of concern to the Process Selector and the process change mechanism.

In addition to the Process Vector, a set of SCAN bits are maintained in the first few bytes of actual storage. SCAN bit i of byte j corresponds to process 8*j+i. The SCAN bits duplicate in summary form the state information in the process vector, and are used by the Process Selector to speed up its operation.

6.2 STATES OF A PROCESS

The STATE field of the PV entry for a process defines the state of the process as described below. The structure of the STATE field is:—



The shaded field is unused by and unaffected by Nucleus operation. The remaining fields define the process state as follows:—

S	L	A	TRANSFER	PROCESS STATE	WAITROUTE	SCAN	NOTES
0	0	0	–	FREE	255	1	1
0	0	1	–	RUN	–	0	
0	0	2	0	WAIT	ENO	1	2
0	0	2	PNO	WAIT/PASS	ENO	0	2,3
0	0	3	PNO	HELD	–	0	3
0	1	3	PNO	HELD/LAST	–	0	3,4
0	0	4	–	READY	ENO	0	5
1	X	X	X	STOPPED	X	1	6

NOTES:

1 For a process in the FREE state, the STIMS consist of 16 ones and QEND is zero.

2 For a process in the WAIT or WAIT/PASS state, WAITROUTE contains the ENTRY number (ENO) of the awaited message.

3 For a process in the WAIT/PASS, HELD or HELD/LAST states, TRANSFER contains the Process Number (PNO) of the process to which priority is to be passed.

4 The L bit is only used to distinguish the last process in a Semaphore chain. It is of no importance to the Process Selector.

5 In the READY state, the ENO in WAITROUTE determines the message to be picked up on entry to the RUN state. If ENO=255, entry is from the FREE state. Hence the first message to hand should be processed next. If ENO≠255, entry is from the WAIT state, in which case the message with that ENTRY number should be processed next.

6 If S=1 the process is in the STOPPED state. Normally the L,A, TRANSFER and WAITROUTE fields of a STOPPED process conform to one of the entries given above.

Non Standard SCAN Values

The SCAN bits can be changed by software to assume different values from those given. The effect of this is:–

(a) If SCAN is zero and its table value is one, the Process Selector takes longer to operate but its operation is not changed.

(b) If SCAN is forced to a one and its table value is zero, the Process Selector does not select that process to run (whatever its state) unless priority is passed to it.

6.3 STATE TRANSITION

Figure 16 illustrates the various states of a process and the state transitions that may be affected by Nucleus. It should be noted that the WAIT and WAIT/PASS states, and HELD and HELD/LAST states have both been amalgamated to simplify the diagram.

The letters against the transitions indicate the cause of the transition as follows:–

A RUN to FREE or WAIT. The process obeys a CALL instruction indicating that the next state of the process is to be FREE or WAIT.

B FREE or WAIT to READY. A message arrives for the process. In the FREE state the arrival of any message causes the transition, while in the WAIT state the arrival of a message with a correct ENTRY number is required.

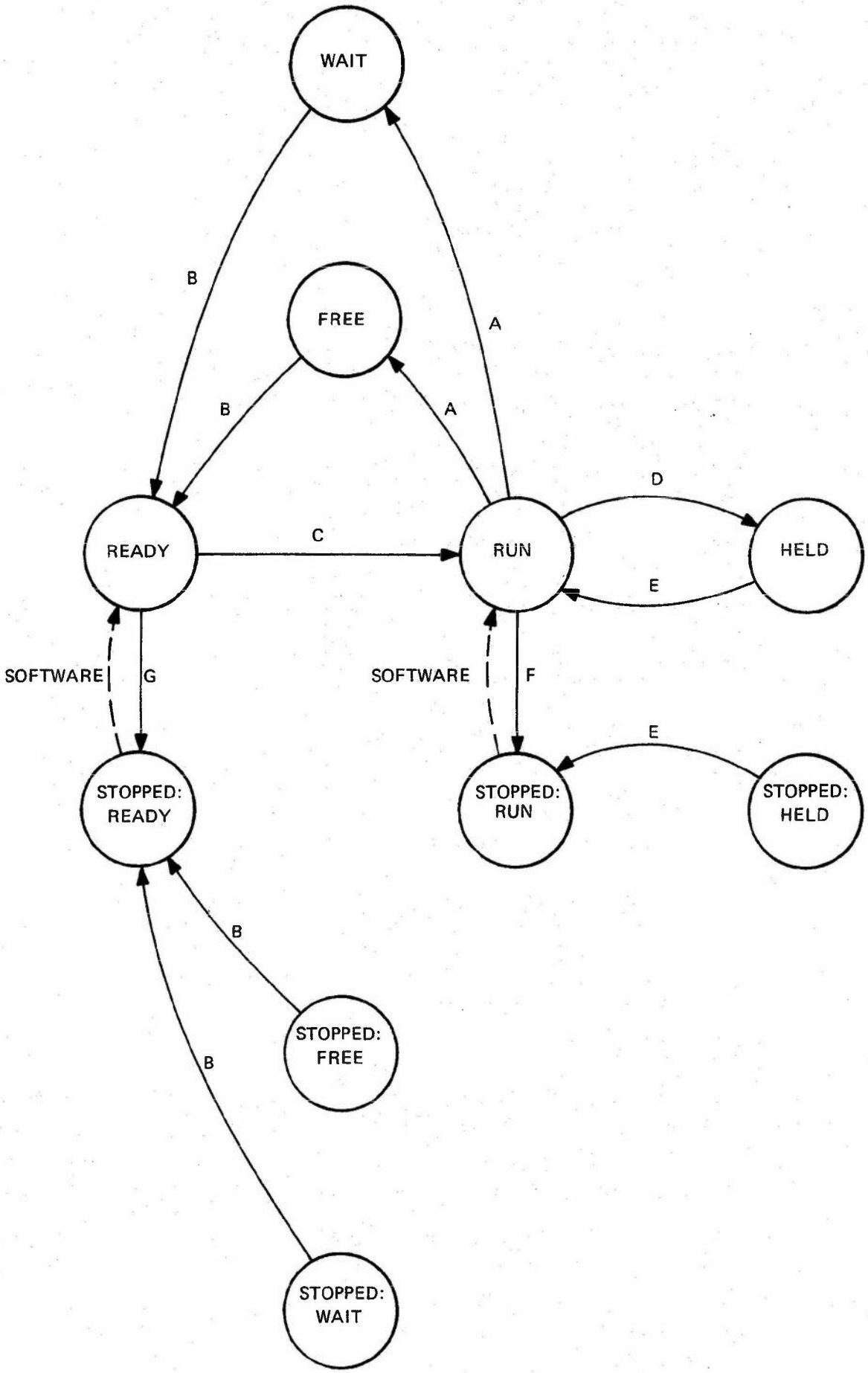


Figure 16: STATE TRANSITIONS

C READY to RUN. The process in the READY is selected to run by the Process Selector.

D RUN to HELD. The process attempts unsuccessfully to CLAIM a semaphore.

E HELD to RUN. The semaphore which the process tried earlier to claim is released by the process holding it.

F RUN to STOPPED: RUN. The process commits some violation as a result of which an error trap occurs, or attempts to reference an absent segment, so that a segment break occurs.

G READY to STOPPED: READY. An absent segment is encountered when Nucleus attempts to move the process into the RUN state, so that a segment break occurs.

Other Transistions

Systems software with access to the Process Vector may change the state of any process between the UNSTOPPED and STOPPED variants of any state simply by changing the S bit of the PV entry. While in the STOPPED state, transitions can occur legally between states as shown in the diagram.

More drastic state changes can be carried out by software intervention. It is evident that any such changes must be performed with extreme caution if unpredictable effects are to be avoided.

6.4 SELECTOR OPERATION

Figure 17 illustrates in simplified form the operation of the Process Selector. The process is as follows:—

(1) The SCAN bits are examined in process number order, until a zero scan bit is found.

(2) The state of the corresponding process is examined. If it is READY or RUN, select this process to operate, and exit to the process change mechanism. If it is WAIT/PASS or HELD, the TRANSFER field contains the number of the process to which priority is to be transferred, and the state of this process is examined next. Otherwise, return to examination of the SCAN bits from the point previously reached.

It is seen from the above that the effect of the SCAN bits is to allow the Process Selector to rapidly 'skip over' processes which are neither runnable nor passing priority.

Limits on Priority Passing

It is possible by poor system design for 'deadly embrace' situations to arise, in which process A passes priority to B, which (directly or indirectly) passes priority back to A. In such a circumstance the Process Selector loops endlessly around the priority pass loop.

To prevent this situation, location PASSMAX in the SVA defines the maximum number of TRANSFER operations which can occur consecutively. If this limit is exceded, control reverts to step (1) of the selector operation.

Termination

Note that there is no test in the selection procedure to terminate the procedure when none of the processes in the system are in a runnable state. Nucleus eventually attempts to access a Process Vector entry beyond the end of the PV segment.Then an error message is generated to system software and the Process Selector is reactivated.

To prevent this situation occurring, it is recommended that every system should include as its lowest priority process a dummy 'idler' process which is always in the RUN state. If there is no useful work for the system to do, this process is selected preventing the scheduler error condition occurrring.

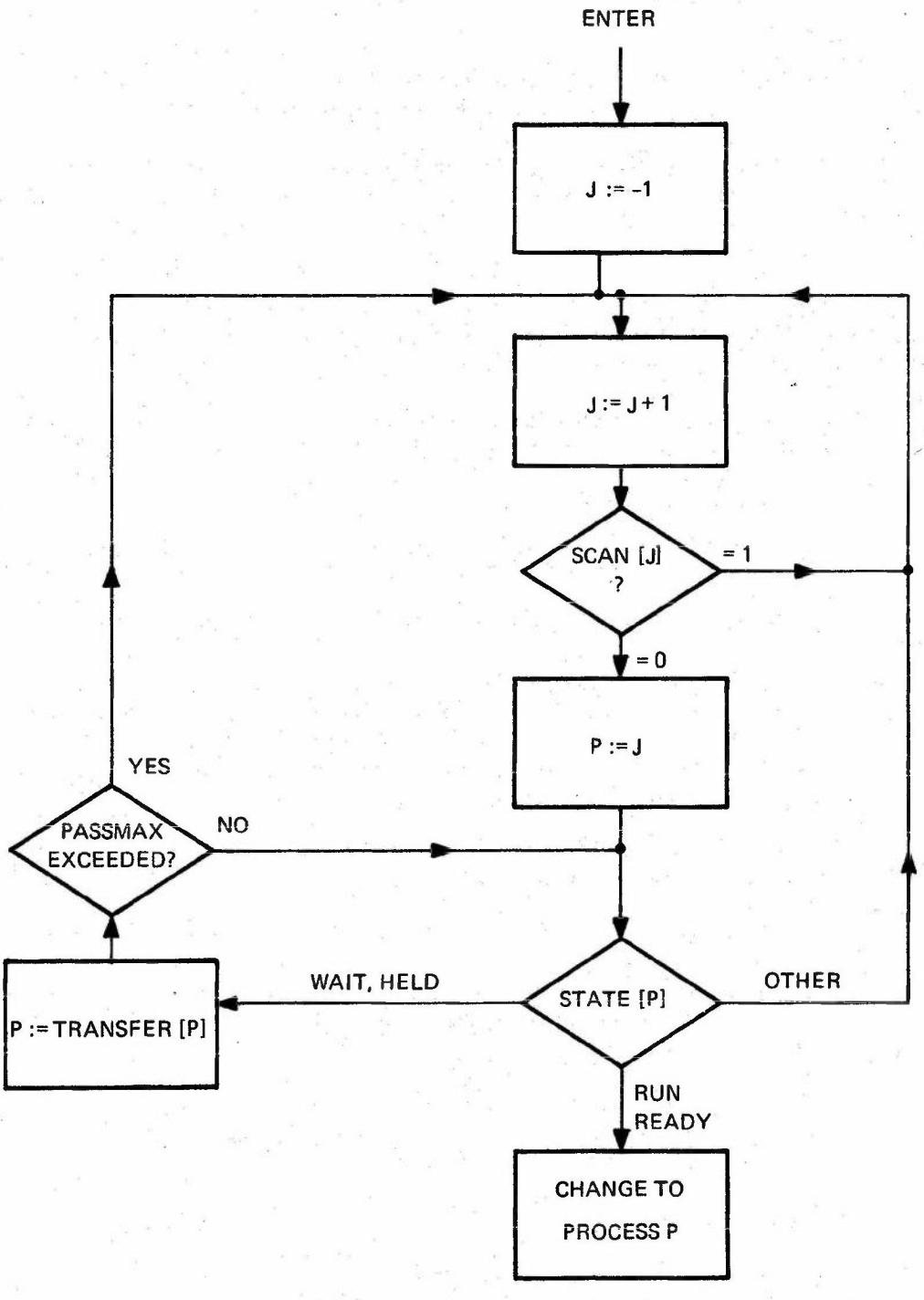


Figure 17: PROCESS SELECTOR OPERATION

6.5 THE PROCESS CHANGE MECHANISM

When the Process Selector has chosen a process to control the processor, the Process Change mechanism is invoked to save the registers of the previously operating process if necessary, and to load the registers of the newly selected process.

Figure 18 illustrates the Process Change mechanism. In this figure and in what follows the 'old' process is that previously in control of the processor and the 'new' process in that newly selected by the Process Selector.

First the process numbers of the old and new process are compared. If they are equal, the same process remains in control of the machine. Otherwise the process is changed by a swap operation.

Same Process

If the new process is the same as the old, the next step is to test the state of the process. If it is in the RUN state, no further action is necessary, and Nucleus operation is terminated and execution of the process coninued. If it is in the READY state, a message is loaded into its registers by the 'Load Message' operation described later.

Swap Operation

If the process is to be changed, a swap operation is performed as follows:—

(1) The working registers of the old process are saved in the Register Save area of its Master Segment. If the old process is in the RUN or HELD state all its registers (S,L,B,A,X,Y,Z,E,C) are saved. Otherwise only the S and L registers are saved.

(2) The hardware segment registers for the new process are loaded as described in sub-section 2.4. If any segment is absent from store, a segment break occurs, and the process is placed in the STOPPED state. The S and L registers are now loaded from the Save area of the process' Master Segment.

(3) The state of the new process is tested. If it is READY, a message is loaded by the 'Load Message' operation. Otherwise, if it is in the RUN state, its remaining registers (B,A,X,Y,Z,E,C) are loaded from the Save Area of its Master Segment. Nucleus operation is terminated and execution of the new process commences.

Load Message Operation

If the new process is in the READY state, the required message is loaded into registers as follows:—

(1) WAITROUTE is tested to determine whether the new process was either FREE or in the WAIT state before being made READY. If WAITROUTE=255 the process was originally in the FREE state, otherwise it was in the WAIT state.

(2) If the new process was in the FREE state, the appropriate message is selected by first examining the STIM bits. If any of these are zero, the message to be processed is the fixed message with ENTRY number equal to the lowest zero STIM bit. Otherwise the message is taken from the head of the incoming message queue.

(3) If the new process was in the WAIT state, WAITROUTE defines the ENTRY number of the appropriate message. Note that if the message is a queued message it is at the head of the message queue.

(4) In both cases (2) and (3) above, if the selected message is on a Fixed route, the corresponding STIM bit is now set to one.

(5) Finally the parameters of the message are loaded into registers A, X, and Y, PAST entry 0 is loaded with the SEGMENT parameter, and register Z is loaded with the ENTRY number. Note that as a result of the operation registers E and B are undefined, and the C register is reset to zero. See Section 3 for details.

(6) The process is now placed in the RUN state, Nucleus operation is terminated, and execution of the new process commences.

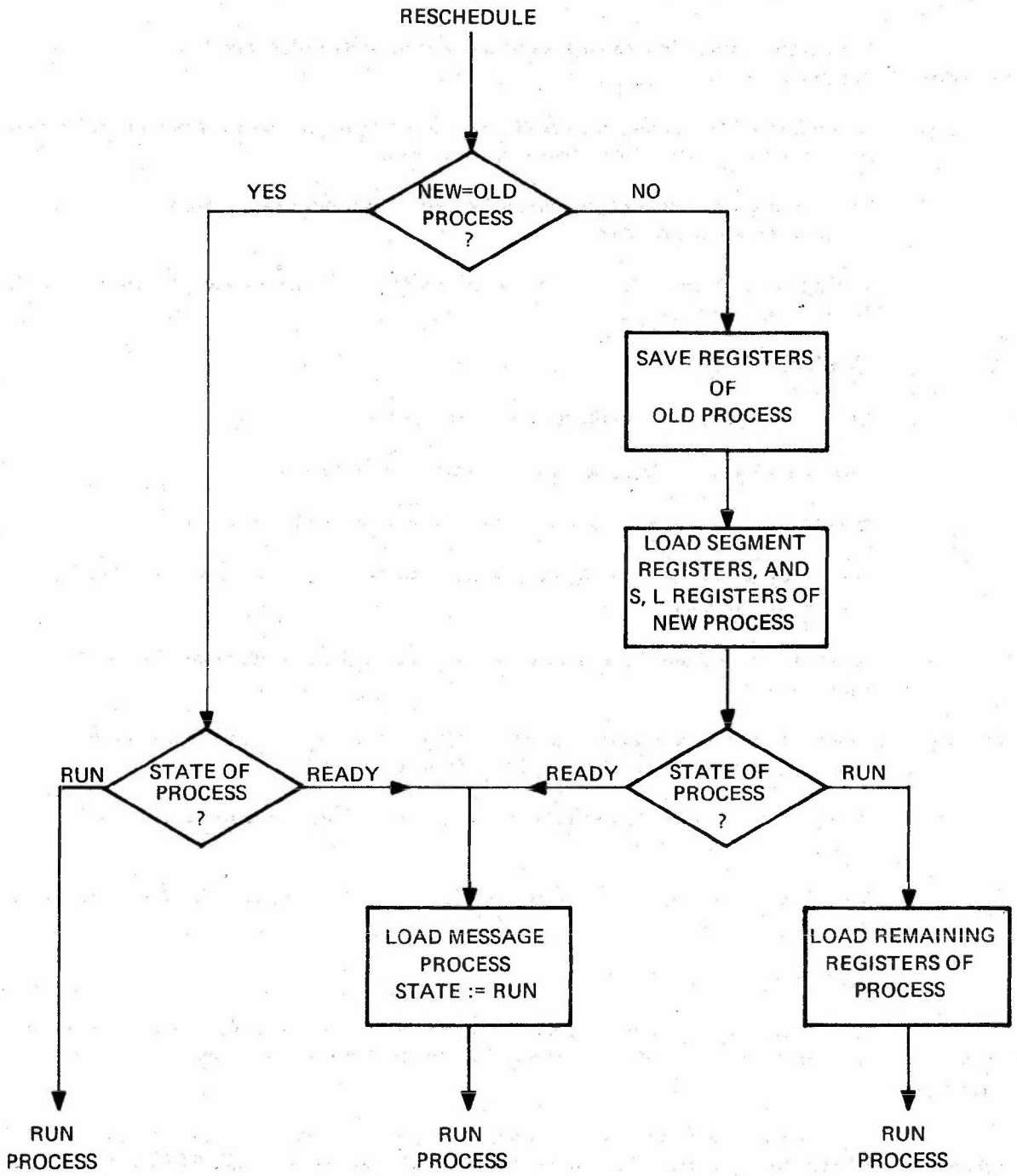


Figure 18: PROCESS CHANGE MECHANISM

7. ERROR HANDLING

The general philosophy of error handling is first to take any immediate action which may be necessary to prevent the effects of the error propagating through the system (e.g. by placing an erring process in the STOPPED state), and secondly to form an error message and send it to a process which may take further action.

The process to which the error message is sent depends on the category of the error. Three main categories are distinguished:—

(a) System Errors. This category includes all errors which may be attributed to hardware malfunctions or erroneous entries in Systems Tables used by Nucleus.

(b) Programming Errors. This category includes all errors which may be attributed to programme faults in a single process in the system.

(c) Segment Breaks. Not strictly error conditions; caused by the attempted use of a segment which is flagged as 'out of main store'.

7.1 SYSTEM ERRORS

The following conditions give rise to system error traps:—

(a) Store Parity Failure: Detected following a Store Read operation.

(b) Store Timeout: No response was received from store when it was last accessed.

(c) Command Interface Timeout: No response was received from an IOP when the Command Interface was last used.

(d) Power Failure: Failure of the power supply of the CPU or a vital independently powered module was detected.

(e) Protection Violation of the Master Segment, SST, PV, or SBA: An attempt was made to access beyond the permitted range of one of these Segments.

(f) QC Error: Indicated if when a message buffer is removed from Free queue, QC becomes negative after being decremented.

(g) STIM Error: Indicated if an attempt is made to reset a STIM bit using an ENTRY number greater than 15.

Error Action

For all system errors an error message is formed by Nucleus and sent to the System Error Process, Process 0 on Entry 0 via Fixed buffer 0 of the SBA. For all errors other than Power Failure (see below) action is also taken as follows.

If at the time the error occurred, the central processor was not executing a Nucleus operation (i.e. was running a Process) then the Process which was running at the time is placed in the STOPPED state. If Process 0 was itself running at the time the error occurred, a dynamic microprogram stop occurs.

After the message has been sent to Process 0 and, if appropriate, Process 0 has been made READY to receive it, a Reschedule operation is performed.

Power Failure

Action on Power Failure is slightly different. Whatever Process or Nucleus operation was being performed, action is always the same. An error message is formed and sent to Process 0 which is forced into the READY state to receive it, regardless of its original state. The processor is now halted.

When power is resumed, and the processor restarted (automatically if in AUTO, otherwise by manual operation), the standard startup procedure is followed. This results in Process 0 being selected to run to process the message planted before power failed.

The Error Message

The parameters of the error message are:—

PARAMETER A	AM – Undefined AL – Location Code (See Table below)
PARAMETER X	Error Code (See Table below)
PARAMETER Y	Process Number
SEGMENT	Master Segment of Process in Y, with full access permission.

(a) *Location Code*

The location code defines the operation taking place at the time the error occurred. The location codes used and their meanings are:—

CODE	MEANING
0	Within a Process (i.e. Not in Nucleus)
1	First part of CALL, SEG, ICB instruction handling
2	First part of SEM instruction handling
3	Nucleus reschedule or process change mechanism
4	Loading a message into registers
5	In the System Error handling section of Nucleus
6	In the Process Error handling section of Nucleus
7	First part of Interrupt Handling

Location codes 1,2, and 7 cover Nucleus operations up to the start of the reschedule operation, when the code becomes 3 in all cases.

Note that location code 5 is never received in a message, since if a system error occurs when the Nucleus is in the System Error handling phase, a dynamic halt occurs.

(b) *The Error Code*

The error code determines the cause of the error trap as defined overleaf:—

CODE	CAUSE	NOTES
0	Protection Violation of the Master Segment.	
1	Protection Violation of the PV	
2	Protection Violation of the SST	
3	Protection Violation of the SBA	
4	Store Parity Failure – Module 0	A
5	Store Parity Failure – Module 1	A
6	Store Parity Failure – Module 2	A
7	Store Parity Failure – Module 3	A
8	Store Time Out	A
9	STIM Error	B
10	QC Error	B
11	Power Failure	C
12	Command Interface Timeout	B

NOTES:

A In these cases, if the Location Code = 0, the process running at the time of the error is placed in the STOPPED state.

B The location code is never 0 in these cases.

C The location code may be 0, but no process is STOPPED.

(c) *The Process Number*

The significance of the Process Number in Y depends on the value of the location code. In some cases it is undefined. It is interpreted according to the table below:

LOCATION CODE	PROCESS NUMBER	NOTES
0	Current Process	A
1	Current Process	B
2	Current Process	B
3	either Old Process or New Process or -1	C
4	New Process	C
6	either Process which caused error or -1 (Segment break error)	C
7	Undefined	

NOTES:

A The process running at the time of error.

B The process which issued the Nucleus instruction

C Old Process = process previously running on the processor.
New Process = process newly selected to run.

7.2 PROGRAMMING ERRORS

These are the errors which arise as a result of illegal operations being attempted by a process. The possible error conditions are as follows:—

(a) Protection Violation: Either the process has attempted to access a segment using a virtual address which lies beyond the end of the segment or the process has attempted to access the segment in an unauthorised manner (e.g. a write attempted to a Read only segment).

(b) Undefined Instruction: The process attempted to issue an illegal (undefined) instruction.

(c) Route Trap: A CALL instruction was used to attempt to send a message over a route whose flag bits were set to '10XX' (Break to Owner).

(d) EXIT Instruction: The process issued an EXIT instruction. This is not strictly an error condition.

(e) QCOUNT Error: The process attempted to send more queued messages than was permitted by the value of QCOUNT.

(f) CALL instruction Error: A CALL instruction was issued with incorrectly specified Route or Parameters. See later for details of possible error conditions.

(g) SEG Instruction Error: A SEG instruction was issued which specified a PAST entry number greater than the number of PAST entries.

(h) CALL I/O Instruction Error: A CALL instruction for either Programmed or Autonomous I/O was issued with an incorrectly specified Route or parameters. See later for details of possible error conditions.

(j) SEM Instruction Error: A SEM instruction was issued which used a corrupt semaphore. See later for details of possible error conditions.

(k) Process Counter Trap: The process' PPCOUNT location was decremented past zero. See later for details.

Error Action

For all errors an error message is formed and sent to the owner of the process via a queued buffer. The erring process is placed in the STOPPED state.

If Process 0 is the erring process, Nucleus enters a dynamic halt, and no message is generated.

The Message Parameters are:—

PARAMETER A	AM — Undefined AL — See *below
PARAMETER X	Error Code (see overleaf)
PARAMETER Y	Process Number
SEGMENT	Master Seg of Process, with full Access

* For Error Code 0, AL contains the CST number of the Segment to which erroneous access was attempted. For all other cases, AL is undefined.

The error code defines the type of error:—

CODE	CAUSE
0	Protection Violation
1	Undefined Instruction
2	QCOUNT error
3	CALL Instruction Error
4	SEG Instruction Error
5	CALL I/O Instruction Error
6	EXIT Instruction
7	SEM Instruction Error
8	Process Counter Trap
10	Route Trap

CALL Errors

The following conditions result in CALL error traps:—

(a) Route Number in Register Z greater than number of routes of the process.

(b) Route entry defined by Register Z is of an incorrect type.

(c) Segment send attempted using a segment without send permission (S bit of CST entry zero).

CALL I/O Errors

The following conditions result in a CALL I/O error trap:—

(a) Route number in Register Z greater than number of routes of the process.

(b) Route entry defined by Register Z is of an incorrect type.

(c) (Programmed I/O only) register X content is greater than the permitted RANGE of Way Number specified by the Route table entry.

(d) (Autonomous I/O Only) the COUNT parameter in register X has its ms two bits non-zero.

(e) (Autonomous I/O Only). The address of the start and end of the data buffer are in different segments, or either (or both) are not valid virtual addresses.

(f) (Autonomous I/O Only). The segment containing a data buffer does not have Transfer permitted (T bit of CST entry zero).

SEM Errors

(a) The virtual address of a semaphore is illegal.

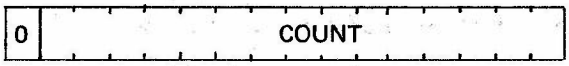
(b) A segment containing a semaphore does not have Read and Write access permission.

(c) A semaphore is Corrupt.

The Process Counter

The process counter is a halfword location of the Master Segment of a process called PPCOUNT. It can be in one of three states:—

(a) *Counting*



In this state it contains a positive integer (COUNT) which defines the number of instructions the process can execute while remaining in the RUN state. If PPCOUNT contains the integer N, N+1 instructions may be obeyed. COUNT is decremented by one after each instruction is obeyed.

(b) *Error*

An attempt to access instructions when PPCOUNT is in the error state causes an immediate error trap with Code 8. If a process is in the RUN state when PPCOUNT is zero (in the Counting state) it is decremented to -1. Therefore the next attempt instruction access causes an Error trap.

(c) *Disabled*

In this state, PPCOUNT is not decremented when instructions are obeyed and never generates an Error interrupt.

7.3 SEGMENT BREAKS

A segment break arises if, when Nucleus selects a process to run it finds that either the Master Segment or one of the four current segments of that Process is absent from main store, or if during a LCST or ICB instruction it is found that the segment involved is absent.

In both cases, the process is placed in the STOPPED state, and a message is formed and sent to a process to deal with the condition, via a queued buffer. The destination of the message is MSBREAKER if the Master Segment is absent, or is the BREAKER of the process if one of the Current Segments is absent.

The parameters of the message are:—

PARAMETER A	AM — Undefined AL — SST Number of missing segment
PARAMETER X	Break Code
PARAMETER Y	Process Number
SEGMENT	(Current Segments Only) ms of Process, with full Access.

The Break code determines the type of segment involved:—

CODE	TYPE OF SEGMENT
-1	Master Segment Break
9	Current Segment Break

7.4 CONTINUATION AFTER ERRORS

In a number of cases where errors occur, one of the processes in the system is placed in the STOPPED state. The question arises, can the process in question be meaningfully continued by removing it from the STOPPED state. The answer depends on whether the Register Save Area in the Master Segment of the process contains a consistent record of the result of the last instruction obeyed by the process.

Three categories of 'continuability' are recognised:—

(a) 'Directly Continuable'. A process is said to be directly continuable if the instruction being obeyed at the time of the error trap was completed normally, and the first instruction which would be obeyed after leaving the STOPPED state would be the instruction following that.

(b) 'Continuation Possible'. Continuation is possible if the instruction being obeyed at the time of the trap was aborted before any program registers were changed, but with the S register incremented to point to the following instruction. It may be necessary to decrement the S register of the process by 2 before the process is unstopped, so that the erring instruction can be retried.

(c) 'Not Continuable'. A process is said to be not continuable if its program registers are inconsistent, or if the final state of the S register is not simply established.

System Errors

Store Parity Failures (Error Code = 4-7) and Store Timeout errors (Error Code = 8) can occur while a process is running in all cases; the process is Not Continuable.

Programming Error

The Continuability depends on the error, as defined by the following table:—

CODE	MEANING	CONTINUABILITY
0	Protection Violation	Not Continuable
1	Undefined Instruction	Continuation Possible
2	QCOUNT Error	Continuation Possible
3	CALL Instruction Error	Continuation Possible
4	SEG Instruction Error	Continuation Possible
5	CALL I/O Instruction Error	Continuation Possible
6	EXIT Instruction	Directly Continuable
7	SEM Instruction Error	Continuation Possible
8	Process Counter Trap	Directly Continuable
10	Route Trap	Continuation Possible

Segment Breaks

Under all circumstances, a segment break leaves the process in a directly continuable state.

8. NUCLEUS SUMMARY

This section contains a summary of Nucleus tables and Nucleus instructions referred to in other sections of this manual.

Nucleus tables described are:

(a) The System Variable Area (SVA)

(b) The System Segment Table (SST)

(c) The Process Vector (PV)

(d) The System Buffer Area (SBA)

(e) The Master Segment for a process.

Three tables of the Master Segment are also described:—

(i) The Current Segment Table (CST)

(ii) The Process Accessible Segment Table (PAST)

(iii) The Route Table

In addition the various groups of Nucleus instruction are also presented in summary form.

8.1 THE SYSTEMS VARIABLE AREA

The Systems Variable Area (SVA) is located at the low addressed end of actual store, from location 0 upwards. It may be regarded as a Segment of information which unlike all other segments is fixed in absolute main store addresses.

The information in the SVA is accessed normally only by Nucleus, and comprises data items used by Nucleus to control fundamental aspects of the system, and entries which enable all other information in the system to be accessed by Nucleus.

Its layout is as given overleaf in Figure 19. The usage of the items in the SVA is described in detail in other sections of this manual.

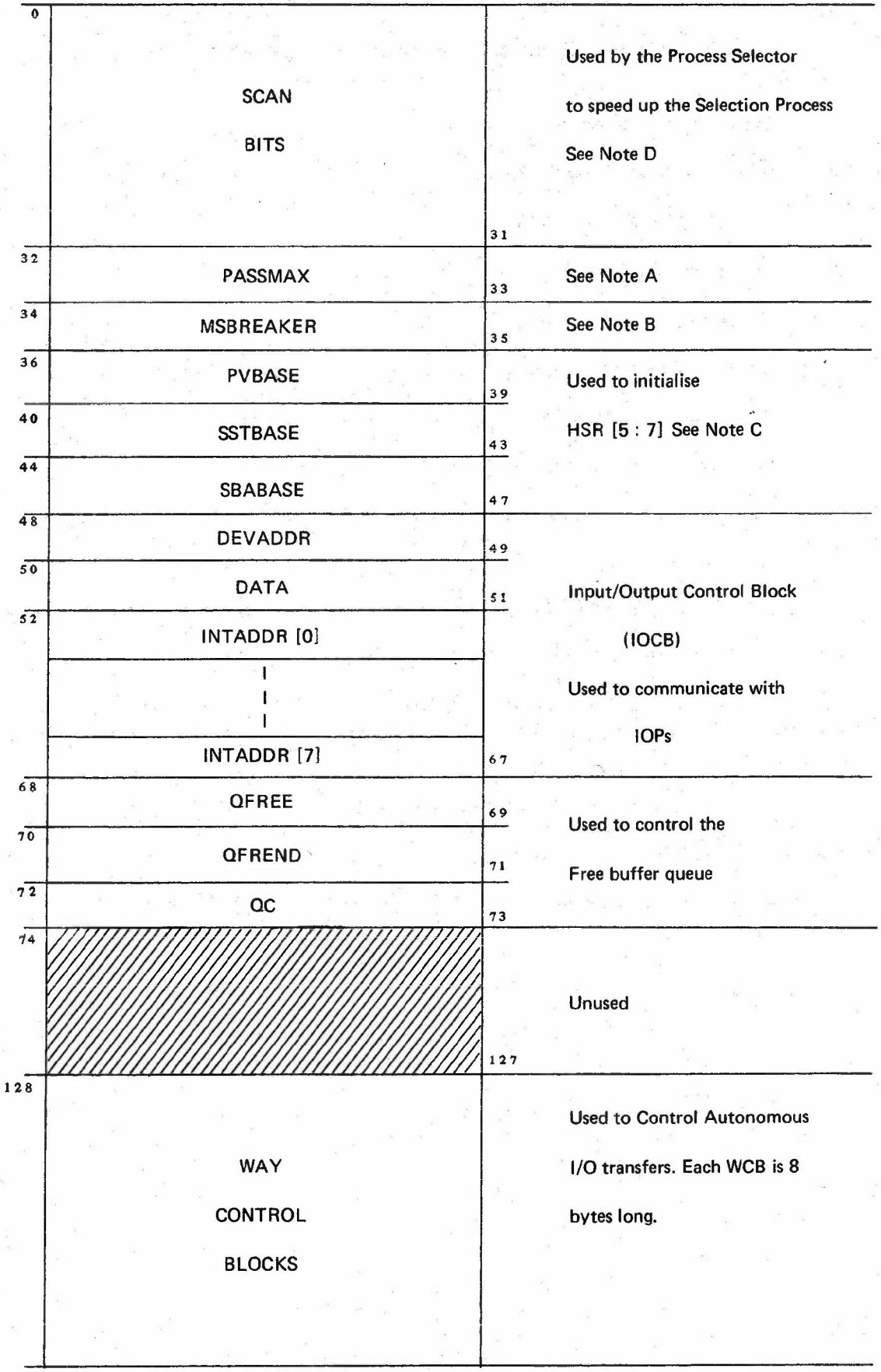
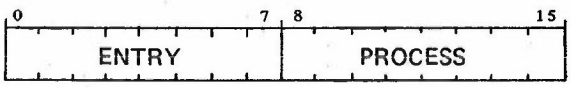


Figure 19: THE SVA

NOTES ON FIGURE 19

A PASSMAX defines the maximum length of a Priority pass chain. It is used by the Process Selector to prevent 'deadly embrace' conditions stopping the entire system.

B MSBREAKER has the format of a DESTINATION viz:—



and defines the Process to be alerted if a Segment Break occurs when a Master Segment is being loaded.

C PVBASE, SSTBASE, SBABASE are used to set up Hardware Segment registers which give access to the PV, SST, and SBA respectively. All have the same format as SST entries (q.v.).

D The SCAN bits provide a rapid accessible summary of the STATE information (q.v.). Bit i of byte j refers to process 8*j+i.

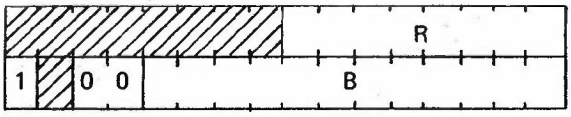
8.2 THE SYSTEM SEGMENT TABLE

The position and extent of the SST in store is defined by location SSTBASE of the SVA. It contains one entry for each segment in the system; each entry is 4 bytes long, and the entry for segment N starts at location 4*N of the SST. Conventionally entries 0, 1, 2, and 3 are used to define the SVA, SST, PV, and SBA respectively.

Format of an Entry

The two possible SST entry formats are:—

(a) *Segment Present in Main Store*



Segment starts at byte address 64*B
Size of segment in bytes is 64*(R + 1)

Shaded fields may be used by systems software.

(b) *Segment Absent*

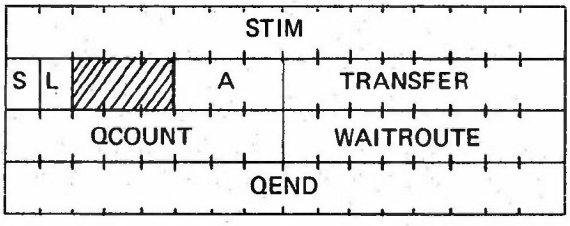
Segment not present in Main Store. Shaded fields may be used by system software.

8.3 THE PROCESS VECTOR

The Process Vector is a segment whose position and size are defined by location PVBASE of the SVA. It contains one entry for each process in the system; each entry is 8 bytes long, and the entry for process N starts at byte 8*N of the PV.

Format of an Entry

The format of each entry is:—



The fields of the entry are:

STIM — A set of 16 independent bits used to record the presence of fixed messages and interrupts awaiting processing by this process.

S,L,A — Collectively define the STATE of a process, according to the table given below.

TRANSFER — The number of a process to which priority is passed if the process is in the WAIT/PASS, HELD, or HELD/LAST state.

WAITROUTE — The ENTRY number of the awaited message if the process is in the WAIT state. In the FREE state it is set to 255.

QCOUNT — At any time defines the number of queued messages that the process can generate. Decremented by 1 whenever a message is generated by this process, and incremented by 1 when the message is received by some other process. An error trap occurs if QCOUNT ever becomes negative.

QEND — A pointer to the queue of incoming messages awaiting processing by this process. Set to zero if the queue is empty.

States of the Process

The following table indicates how the S, L, A and TRANSFER fields are used to define the state of the process and, for each state, the corresponding values of the SCAN bit for the process, and WAITROUTE.

S	L	A	TRANSFER	STATE	WAITROUTE	SCAN
0	0	0	—	FREE	255	1
0	0	1	—	RUN	—	0
0	0	2	0	WAIT	ENO	1
0	0	2	PNO	WAIT/PASS	ENO	0
0	0	3	PNO	HELD	—	0
0	1	3	PNO	HELD/LAST	—	0
0	0	4		READY	ENO	0
1	X	X	X	STOPPED	X	1

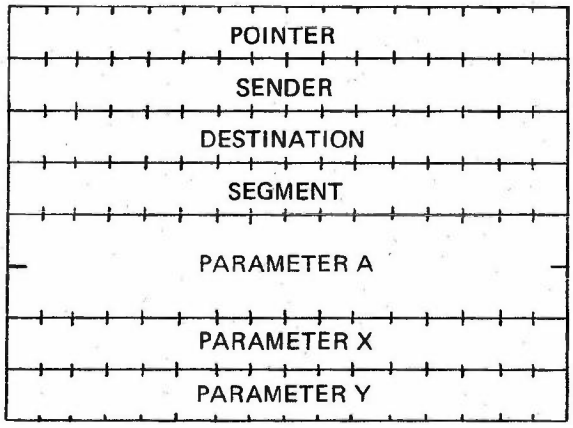
8.4 THE SYSTEMS BUFFER AREA

The SBA is a segment whose position in store, and size, are defined by location SBABASE of the SBA. The SBA contains a number of buffers which are used by Nucleus in the mechanisation of the Inter-Process Message system.

Two types of buffer are provided, Fixed buffers and Queued buffers. Both have the same structure, but are used in different ways. In general, Fixed buffers are located at the low addressed end of the SBA and Queued buffers at the high addressed end of the SBA. In particular the first buffer (i.e. the buffer starting at location 0) must be a Fixed buffer.

Each buffer is 16 bytes long. Therefore a maximum of 1K buffers can be provided, in a maximum sized segment.

The format of a buffer is:—



The fields of the buffer are used as follows:—

POINTER — For a Fixed buffer it is always zero.
For a Queued buffer it is used as a chain pointer in a queue containing an address relative to the start of the SBA.

SENDER — Process number of the originator of the message. It is fixed in a fixed buffer.

DESTINATION — Defines the ENTRY and Process number of the destination of the message. It is fixed in a fixed buffer.

SEGMENT — If a segment is sent with the message it contains a copy of a CST entry in the same formatbut with the M (and possibly also the W and T bits) forced to zero. If a segment is not sent, it contains zero.

PARAMETERS A,X,Y — Parameters of the message, copied from the registers of the originating process.

8.5 THE MASTER SEGMENT

Each process in the system is provided with a special segment, called a Master Segment. The Master Segment contains information to enable Nucleus to control the operation of the Process. The position and size of the Master Segment for process N is determined by entry N + 4 of the SST.

The information in the Master Segment is accessed normally only by Nucleus while the process is in control of the processor. It is necessary, however, for privileged systems processes to be able to access the Master Segments of other processes on occasions. A process is not normally given access to its own Master Segment.

The layout of the Master Segment for a process is given in Figure 20. The usage of the various items in the Master Segment are described in detail in other sections of this Manual.

0	CODESEG | PNO	1	See Note A
2	PPCOUNT	3	See Note B
4	CST[0]		CST. Contains 4 two byte entries
	CST[3]	11	
12	S		Register Save Area. Registers stored here whenever Process is not in control of the processor.
	L		
	BM		
	BL		
	AM		
	AL		
	X		
	Y		
	Z		
	E | C	31	
32	PASTPTR	33	Defines start and extent of the PAST
34	PASTMAX	35	
36	OWNER	37	Define processes which deal with errors and segment breaks. See Note C.
38	BREAKER	39	
40	ROUTE TABLE		Each entry occupies 4 bytes. See Note D.
		PASTPTR-1	
PASTPTR	PAST		PAST. Each entry occupies 4 bytes See Note E.
		PASTPTR+ 2*PASTMAX-1	
PASTPTR +2*PASTMAX	AUXILIARY AREA		
		SEGEND	

Figure 20: THE MASTER SEGMENT

NOTES ON FIGURE 20

A CODESEG is the PAST number of the segment currently in use as CST[3]. It is used to mechanise the Inter Chapter Branch instruction.

B PPCOUNT is a count of the number of instructions which may be obeyed by the process. Each time an instruction is obeyed, PPCOUNT is decremented by 1. When it goes negative an error trap is taken and the process placed in the stopped state. Facilities are available to disable the effect of this mechanism.

C OWNER has the format of a DESTINATION. It defines the process to which a message should be sent if a process error trap occurs. BREAKER also has the format of a DESTINATION, and defines the process to which a message is sent if a Segment Break occurs.

D The Route table is defined in detail later. The table contains $\frac{\text{PASTPTR}}{4}$ -10 entries, each of 4 bytes.

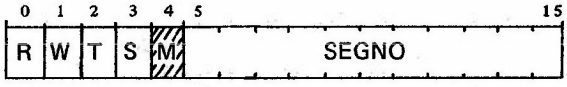
E The PAST is described in detail later. It contains PASTMAX entries each of 2 bytes. The format of PAST and CST entries is identical.

F The Auxiliary area extends from location PASTPTR + 2*PASTMAX to the end of the Master Segment. It is available for use by system software.

8.6 THE CST AND PAST

The PAST contains one 2 byte entry for each segment which may be accessed by a process. The CST contains copies of the PAST entries of the four segments currently in use by a process. CODESEG contains the PAST number of the segment currently in use on CST[3].

The formats of PAST and CST entries are identical:—



R, W, T, S, M are Access Permission bits used as follows:—

R — Set if read permitted from segment

W — Set if write permitted to segment

T — Set if I/O transfers may be performed to segment

S — Set if Segment may be sent with an Inter-Process message

M — Marker bit used by system software

SEGNO is the number of the segment in the SST.

8.7 THE ROUTE TABLE

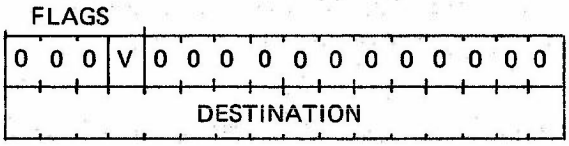
The route table defines the possible destinations for inter-process and I/O messages generated by the process. One four byte entry is used for each possible destination.

Seven different types of route are provided, each with a different format of route table entry. The type (and therefore format) of the route are defined by Flag bits contained in the first byte of the entry.

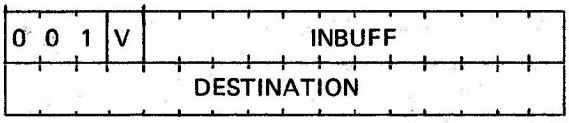
Inter-Process Routes

Four types of inter-process route are provided:—

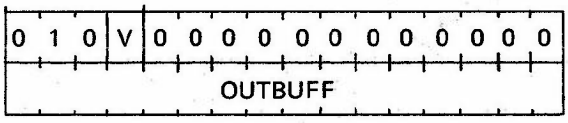
(a) Queued Out — Queued In



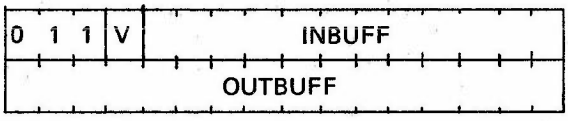
(b) Queued Out — Fixed In



(c) Fixed Out — Queued In



(d) Fixed Out — Fixed In



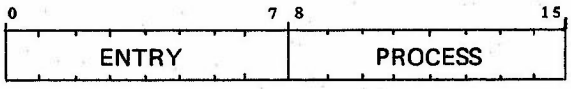
Where in the above:—

V is 1 if a segment is to accompany the message, and is 0 otherwise.

INBUFF is the address of the fixed buffer holding an incoming message. It is a byte address right shifted 4 places.

OUTBUFF is the address of a fixed buffer to be used to hold an outgoing message. It is a byte address.

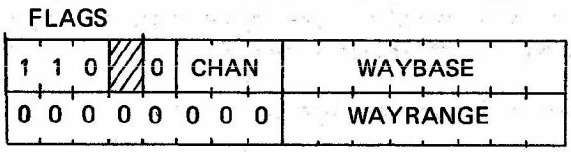
DESTINATION has the standard format of a destination, viz:—



Where PROCESS is the Number of the process to which a queued message is to be sent

ENTRY is the incoming entry number by means of which the destination process recognises the source of the message.

Programmed I/O Route



The shaded bit can take either value, but is conventionally zero.

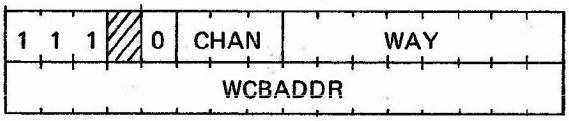
CHAN is the number of an IOP in the system.

WAYBASE is a Way number which may be indexed by register X.

WAYRANGE is the maximum value of the ls byte of X.

Autonomous I/O Route

The format of the entry is



The shaded bit can take either value. Conventionally it is zero.

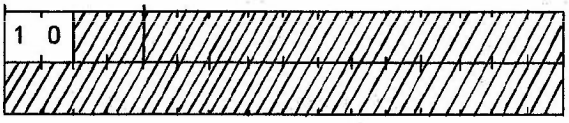
CHAN is defined as above.

WAY is the Way number of the autonomous device.

WCBADDR is the address of the WCB to be used for the transfer.

Break to Owner

An attempt to send a message over a route whose entry has the format below causes an error to be sent to the OWNER of the process, and the process is placed in the STOPPED state.



8.8 SIZE LIMITATION OF NUCLEUS

Maximum Number of Processes	256
Maximum Number of Segments (in total)	2K
Maximum Number of buffers	1K
Maximum Number of IOPs	8 (inc. BMC)
Maximum Number of Ways per IOP	256
Maximum Number of Segments/Process	255

8.9 INSTRUCTION SUMMARY

This section summarises Nucleus instructions which have been introduced in the previous sections. In each case the hexadecimal code of the instruction is given together with a Mnemonic, a brief description of its effect, and notes concerning relevant hardware registers.

Inter Process Message (Section 3)

HEX	MNEMONIC	EFFECT	REGISTERS
1000	EXIT	Send Exit message to owner	–
102Q	STAT	Change state as defined by Q	Z = Route Number
104Q	RSEN	Send Message (Segment with Restricted Access)	Z = Route Number
106Q	SEND	Send Message (Segment with same Access)	Z = Route Number

Where Z defines an Inter Process Route

Where Q = 0 Next state:– FREE
1 RUN
2 WAIT/PASS
3 WAIT
4 FREE (Conditional)
7 WAIT (Conditional)

Programmed I/O (Section 4)

HEX	MNEMONIC	EFFECT	REGISTERS
1080	PIN	Perform Programmed Input	Z = Route Number
10C0	POUT	Perform Programmed Output	Z = Route Number

Where Z defines a Programmed I/O route.

Autonomous I/O (Section 4)

HEX	MNEMONIC	EFFECT	REGISTERS
1080	RWCB	Read WCB Status	Z = Route Number
10A0	LWCB	Load WCB	Z = Route Number
10C0	TRIP	Command Transfer	Z = Route Number
10E0	LWT	Load WCB, Command Transfer	Z = Route Number

Where Z defines an Autonomous I/O Route.

Inter Chapter Branches (Section 2)

HEX	MNEMONIC	EFFECT	REGISTERS
1100	ICBR	Inter Chapter Branch	Z = Ptr. to Chapter Descriptor
1120	ICBL	Inter Chapter Branch and Link	Z = Ptr. to Chapter Descriptor

Segment Manipulation (Section 2)

HEX	MNEMONIC	EFFECT	REGISTERS
130Q	LCST	Load CST from PAST	X = PAST Number
132Q	CLCS	Load CST from PAST (Conditional)	X = PAST Number
134Q	SCST	Store CST in PAST	X = PAST Number
1360	LHSR	Load Hardware Segment Registers	—

Where Q = 0:3 defines a CST entry.

Semaphore Manipulation (Section 5)

HEX	MNEMONIC	EFFECT	REGISTERS
1400	REL	Release Semaphore	Z = Ptr. to Semaphore
1420	CCLM	Claim Semaphore (Conditionally)	Z = Ptr. to Semaphore
1440	CLM	Claim Semaphore	Z = Ptr. to Semaphore