MU5 Hardware Manual

Volume 1

MU5 Hardware Manual

Volume 1

Volume 2

Volume 3

Chapter 2 Instruction Buffer Unit

Chapter 3 — The Primary Operand Unit

Chapter 4 Central Highway

Chapter 5

B-ARITHMETIC UNIT

5.1 Introduction

5.2 Functional description

5.3 Overall diagram

5.4 Mode of operation

5.5 Definitions of interface signals

5.6 Function code

Chapter 10 The Local Store

Chapter 2 Instruction Buffer Unit

2.1 Function of the Instruction Buffer Unit

2.1.1 Introduction

The Instruction Buffer Unit (IBU) supplies functions to the Primary Operand Unit (PROP) in the instruction sequence required by the current process. In particular, the IBU is designed such that the effect that a control transfer has on a process sequence is the minimum compatible with the engineering effort and hardware available for this unit.

It has been shown that for many programs an average of 65 out of every 100 test instructions cause a control transfer, with a subsequent disruption of the instruction flow. If the IBU could 'remember' the results of tests and supply to PROP the same sequence of instructions used the previous time a test was obeyed, then it would be possible to reduce the number of disruptions to the instruction flow.

The ideal solution would be that where PROP Control detected a control transfer, the required out of sequence instruction would be in the pipeline, and be the next to be obeyed. To realise this situation would require an IBU of impractical complexity, so a compromise arrangement has been adopted which still offers a sig ifi-cant improvement. The statistical information which assisted the design of this unit is available in another document (L.A.Taylor), and so will not be discussed here.

2.1.2 IBU Orga isation

The IBU may be considered as consisting of three main sections:-

(a) Data flow and control
(b) Store request system
(c) Jump trace.

The remainder of this section provides a g neral outline of the functions of the above, while detailed information is contained in the succeeding sections of this chapter. The IBU is shown in schematic form on diag am 2/FIG.1.

The data flow is that section of the IBU which is in the path between the SAC highway and PROP. Basically the data flow consists of three 128-bit buffers and associated control logic. Information is received from store in 64-bit biword form (1 word = 32 bits), and assembled in the first IBU buffer to form a 128-bit quadword. Each quadword contains eight 16-bit functions. The control logic unpacks the required functions from the quadword, and holds them in sequential order in the second and third 128-bit registers from where PROP extracts them as required.

The store request system consists of a counter loop which forms the store address of quadwords and issues store requests at a rate matched to that at which PROP takes functions from the data flow. Normally the store request system generates requests in this cyclic manner until a control transfer occurs.

The jump trace is an associatively addressed store within the IBU which holds pairs of jump-from and jump-to addresses. When the store request system produces an address which is identical to one held within the associative field of the jump trace, i.e., a jump-from address, then the normal procedure of issuing a store request is inhibited. Instead the address of the jumped-to instruction is read out of the value field of the jump trace and it is this address which is sent to the store to locate the out of sequence instructions. Obviously there is a limit to the number of jump-from/jump-to addresses which can be stored, and it can be shown that a satisfactory size of jump trace is one having eight lines. The trace is loaded under the control of PROP, a jump being loaded after having caused a control transfer once. The eight line jump trace, loaded in a cyclic manner, has the effect of reducing the number of disruptions per 100 tests from 65 to 35.

Note that once a jump has been entered into the trace the store request system will continue to interrupt the sequence of functions sent to PROP until the entry is over-written. However, although the function sequence may be interrupted, the flow of instructions through the PROP pipeline is not affected. No attempt is made to correct the trace when a predicted jump does not occur, instead PROP issues a store request to obtain the required functions.

2.2 Data Flow and Control

2.2.1 Input Register

Requests to the store are made for a 128-bit quadword which represents one line of one stack, or for the most significant 64-bit biword in the quadword depending on the number of 16-bit functions (half-words) actually required from that quadword.

The IBU is connected to the fast store by a 64-bit highway, thus the quadword will be received from store as two separate 64-bit biwords. The more significant biword (bits 00 to 63 of the quadword) arrives at the IBU accompanied by three extra bits. One of these bits is the IPBS bit and is discussed later in section 2.3.5. The second bit is the IDVA bit which instructs the Unpack Logic to strobe the more significant biword in the more significant half of the 128-bit input register RIP. The third bit is the valid data bit (IDV). Approximately 40ns later the less significant biword is strobed into the l.s. half of RIP.

2.2.2 Unpack Logic

The quadword in RIP may contain a number of functions which are not required for the process, only the required functions are transferred, via the Additional Storage Register (RAS) to the Close Pack register RCP. From RCP the 16-bit functions are taken by PROP at a maximum rate of one every 40ns.

The information required to control the unpacking of functions from RIP is contained in one of the four lines of the Unpack Record (see 2.2.6). One line of the Unpack Record controls the passage of one quadword through RIP. Each function has a sequence (S) bit which accompanies the function across to PROP.

2.2.3 Buffer Control

The Fill Register (RFL) controls the transfer of functions from RIP to RAS via the anti-skew register RIMB. The transfer of functions from RAS to RCP is controlled by the Unfill Register RUNF.

RAM, RBM, RCM and RDM are monitor registers which keep track of the valid functions as they pass through the buffers. RAM and RDM record empty spaces in RAS and RCP, RCM and RCP valid functions in RAS and RCP.

2.2.4 Advanced Control

The function in RCP which is the next to be used by PROP is indicated by the cyclic advanced control register. When a function has been accepted by PROP this register is incremented by 1.

2.2.5 Functions Held Unit

The functions held unit detects the number of valid functions in RCP from the monitor register RBM and produces five signals (GIB17, GIB21, GIB22, GIB23 and GIB24) which indicate there are ≥ 1, ≥ 2, ≥ 3, ≥ 4 or ≥ 5 valid functions respectively. These signals may be sampled by PROP to determine the number of functions held in RCP at any time. Note that in the case of instructions which require more than one function (16-bit half-word), PROP will not commence such an instruction until all the required half-words are available in RCP.

2.2.6 Unpack Record

The information required to receive quadwords into, and unpack them from, the Input Register is contained in the Unpack Record. The contents of the Unpack Record represent the effective number of functions requested but not yet strobed into RAS. The Unpack Record has four lines, each representing the passage of one quadword through RIP, and each comprising five fields as detailed below.

(a) JT bits (3) give the position in RIP of the first function in the quadword which is to be transferred to RAS.
(b) T bits (3) gives the number of valid functions in the quadword which are to be transferred to RAS.
(c) S bit. This bit indicates whether the first function to be unpacked is in (S = 0) or out (S = 1) of sequence.
(d) W bit. This bit indicates the number of passes through RIP. The effect of the W, or same word, bit is described more fully in section 2.4.
(e) C.O. bit. This bit indicates whether a segment overflow has occurred.

The Unpack Record is a shift register i.e. filled at one end and emptied at the other end (see section 2.3). The output is applied to the Unpack Logic for the purposes of unpacking the quadword in RIP.

When the four lines of the Unpack Record are full the Store Request System is prevented from making a store request for a further quadword until such time as a line becomes free.

The JT field is loaded directly from the Store Request register RSR, while the T bits are obtained from the T bit generator, and the S,W and CO are obtained from the Trace Equivalence Logic, the Trace Value Field and the Predicted Control Register respectively.

2.3 Store Request System

2.3.1 General

The Store Request System provides requests for quadwords from the store at the required intervals. The store address is obtained by cycling the previous address through a series of counter loops, incrementing the address by 2 on each cycle. These store addresses are compared with the table of jump-from addresses held in associative field of the jump trace. If the addresses are identical then the corresponding jump-to address is read out of the value field and sent to store. Where the address is not contained in the jump trace the store address is sent to store only when quadword boundaries are crossed. The store request system is shown, together with the jump trace in diagram 2/FIG.1.

2.3.2 Store Address and Counter Loop

Initially the IBU receives from PROP a 31-bit absolute jump-to address, 14 segment bits and 17 line bits, which is loaded into the Store Request register RSR. A priority request is then made to the store for the addressed quadword, and the 31-bit address is also copied to the 31-bit Predicted Control register RPC.

As PROP may use the functions held in the close pack register at a rate of one every 40nsec, it is necessary for IBU to generate store requests at intervals to match this rate. Note that all store addresses produced by the store request system are applied to the associative field of the jump trace. This association takes 40nsec and thus can be performed in parallel with the incrementing of the store address in the counter loop. If, however, a jump pair is detected and the required jump-to address is available in the value field, the reading out of this address occupies a further 40nsec. This would have the effect of increasing the counter loop time to 80nsec, thus giving an average counter loop time of >40nsec.

To overcome this effect, and to bring down the counter loop time to <40nsec, the association for two addresses is performed at the same time. The design of the associative store (jump trace) allows this procedure to be followed. The jump trace is arranged to have a 30-bit associative field, with the extra 31st bit selecting the correct line of the value field. Thus two addresses differing only in their least significant bit may be presented to the associative field at the same time.

If the Line Register RLR indicates that an equivalence has been obtained in the associative field, then the 31st bit of the address is used to select the correct line of the value field. The required jump-to address is then read out to the Store Request register. Note that while the associative phase is in progress the address held in RPC is cycled through the counter and incremented by 2 (on the m.s. 16 bits of the line address). The counter is normally set to the +2 mode, except when loading the jump trace (see section2.4), and the cycle is completed in 40nsec. The new address formed is strobed into RPC and associated.

If RLR does not indicate an equivalence for the first 'double' association it is strobed for the result of the next and the counter cycle repeated. Thus the address in RPC is incremented by 2 for every cycle of the loop. The counter loop continu s in this manner until one of the following conditions occurs.

(a) Jump trace equivalence
(b) Load order signal from PROP
(c) Store request signal from PROP
(d) Store Access Control (SAC) does not accept a request
(e) The Buffers (RAS and RCP) become full
(f) The Unpack Record becomes full.

2.3.3 Non Priority Store Requests

A non priority store request is made when:-

(a) A quadword boundary is crossed
(b) Jump trace equivalence occurs

The crossing of a quadword boundary is detected by RWO in the carry system of the counter loop. This causes the new address formed by the counter loop to be copied into RSR as well as RPC. The output of 'T' bit generator is strobed into the Request Releasing System (see 2.3.4) and to the Unpack Record where it is stored, together with the three least significant bits of RSR, until the requested quadword comes from store. The 'T' bits are used to determine the number of valid functions in the previous quadword and the three least significant bits of RSR, which will all be zero, are used to point to the start of the unpacking of the present quadword.

As stated previously the counter loop is halted if jump trace equivalence occurs, and while the jump-to address is being read from the value field to RSR the Unpack Record is filled as above, but with the 'S' bit set to a 1. A non priority store request is made, using the jump-to address in RSR, unless the W bit is set (see section 2.4 - Jump and Loop Catching). The jump-to address is copied to RPC, and cycled around the counter loop in the normal manner.

2.3.4 Request Releasing System

The rate of issuing requests is related to the rate at which PROP uses the functions held within the IBU. It will be appreciated that should PROP be held up for a period further requests must be delayed owing to the limited storage capacity of the IBU's buffers.

After non priority requests have passed through the CPRs they are prevented from going to store by the Request Releasing System, until there is time and space available to unpack the valid functions held within these request quadwords.

The 'T' (time) bits are a count of the valid functions in any one quadword and by using these the Request Releasing System can determine the time (40nsec per function) and space (16 bits per function) that will need to be available before a request can be released to store.

RCP and RAS can hold a total of sixteen functions. If the total number of functions requested minus the number of functions taken by PROP is greater than sixteen, and PROP was to stop, the extra functions would come to rest in RIP. The releasing of another request at this stage would cause these functions in RIP to be overwritten. Therefore, the rule for releasing requests is:- The total number of functions requested since the buffer were last cleared minus the number of functions taken by PROP is equal to or less than sixteen AND sufficient time has elapsed (T bits X 40ns) since the previous request was released.

2.3.5 Priority Store Requests

An IBU priority request has top priority and is initiated by PROP when:-

(a) An interrupt occurs

(b) A procedure call is made

(c) The IBU supplies to PROP an incorrect sequence of functions.

In cases (a) and (b) the Fixed Instructions are read (see 2.3.6) before PROP initiates a priority request, but in case (c) PROP detects, by use of the sequence (S) bit, the fact that the IBU has predicted incorrectly. In all three cases the IBU's buffers and the PROP pipeline are full of redundant functions, and when the IBU issues a priority request it stops its counter-loop, clears its buffers, and waits until the store answers the priority request. All outstanding non priority requests are ignored until the store answers the priority request with the IPBS bit.

2.3.6 Fixed Instructions

Within the IBU there are four fixed instructions which are:-

(a) Set Link (32 bits)
(b) Exit (32 bits)
(c) Stack Link
(d) Jump D[0].

When an interrupt entry is made (a) and (b) are read, the IBU is cleared (not the jump trace) and a priority request is made. Similarly, when a procedure call occurs (c) and (d) are read, the IBU is again cleared, and a priority request is made.

2.4 The Jump Trace

2.4.1 General

The jump trace is an eight line associativ ly addressed store which holds jump-from (associative field) and jump-to addresses (value field). The jump-from address (31 bit) is the address of the last function (half-word) of the last instruction used by PROP which caused a control transfer. The jump-to address is the address of the first half-word of the jumped-to instruction, i.e., the instruction address. Thus the trace holds eight different control transfers (jump pairs). The loading of jump pairs into the trace is cyclic as described in the following section.

2.4.2 Jump Trace Loading

The onus is on PROP to decide which jumps shall be loaded into the jump trace. When such a predicted jump reaches the PROP control register a second time, its sequence (S) bit is set and this prevents a second loading of this jump into the trace. Note that the jump will now automatically be followed by the out of sequence instructions obtained by IBU.

Two forms of relative jump instructions are handled by PROP:-

(a) Jumps with a literal increment

(b) Jumps with an operand increment.

Type (b) infers a non-catchable jump in that it is unlikely that Control will transfer to the same address again when this particular jump occurs (i.e., multiway jump). If such jumps were to be held in the trace it would be necessary to store and supply to PROP the predicted jump-to address to be checked against the actual jump-to address, with a resultant high time and storage penalty.

To avoid adopting this checking procedure only literal relative control transfers are held in the trace. A control transfer of this type is loaded into the trace after it has caused a control transfer once. After this such an instruction will either jump or not jump again, but when a jump is successful it is always to the same store address. Thus no jump-to address checking is required.

The sequence of functions supplied to PROP may be incorrect for any of the following reasons:-

(a) An unpredicted relative literal increment jump occurs (S = 0)

(b) A predicted jump does not occur (S = 1)

(c) An EXIT or RETURN instruction occurs

(d) A non-literal increment jump occurs.

Condition (a) causes the trace to be loaded with the jump-from and jump-to addresses, and a priority request is made for the quadword containing the first of the out of sequence instructions. Conditions (b), (c) and (d) cause priority store requests only (i.e., no trace loading).

The PROP Control Adder performs two cycles to form the jump-from and jump-to addresses, which are supplied to IBU for loading into the jump trace. First the address in the Control register is incremented by 1, 2, 3 or 5 depending on the number of 16-bit half-words which form the jumped-from instruction. This gives the address of the first function of the next instruction in the normal program sequence, which is not strobed into the Control regist r. During the second adder cycle the jump-to address is formed by applying a literal or non-literal increment, the result being strobed into Control.

When an unpredicted jump occurs (condition (a) above) the result of the first adder cycle is loaded into the associative field of the trace as the jump-from address, after being decremented by 1. This operation, performed by the store request counter loop, ensures that the address loaded is the address of the last function (half-word) of the last instruction before control transfer. While the subtraction is performed the jump-to address is set into RSR. A priority store request is made, using the address in RSR, while the respective fields of the trace are loaded from RPC and RSR.

As stated previously, the conditions (b), (c) and (d) do not cause the trace to be loaded. The first cycle output of the Control Adder is not used, but the result of the second cycle forms the jump-to address which is loaded into RSR. A priority store request is made to locate the out of sequence instructions, while the content of RSR is copied to RPC. The counter loop then commences to form the new address in the normal manner.

In all cases where a store request is made after an incorrect function sequence, the Unpack Record is cleared and then the Unpack Record JT field is loaded from the least significant three bits of the jump-to address in RSR. The Unpack Logic will only recommence unpacking operations when the required priority requested quadword arrives from store.

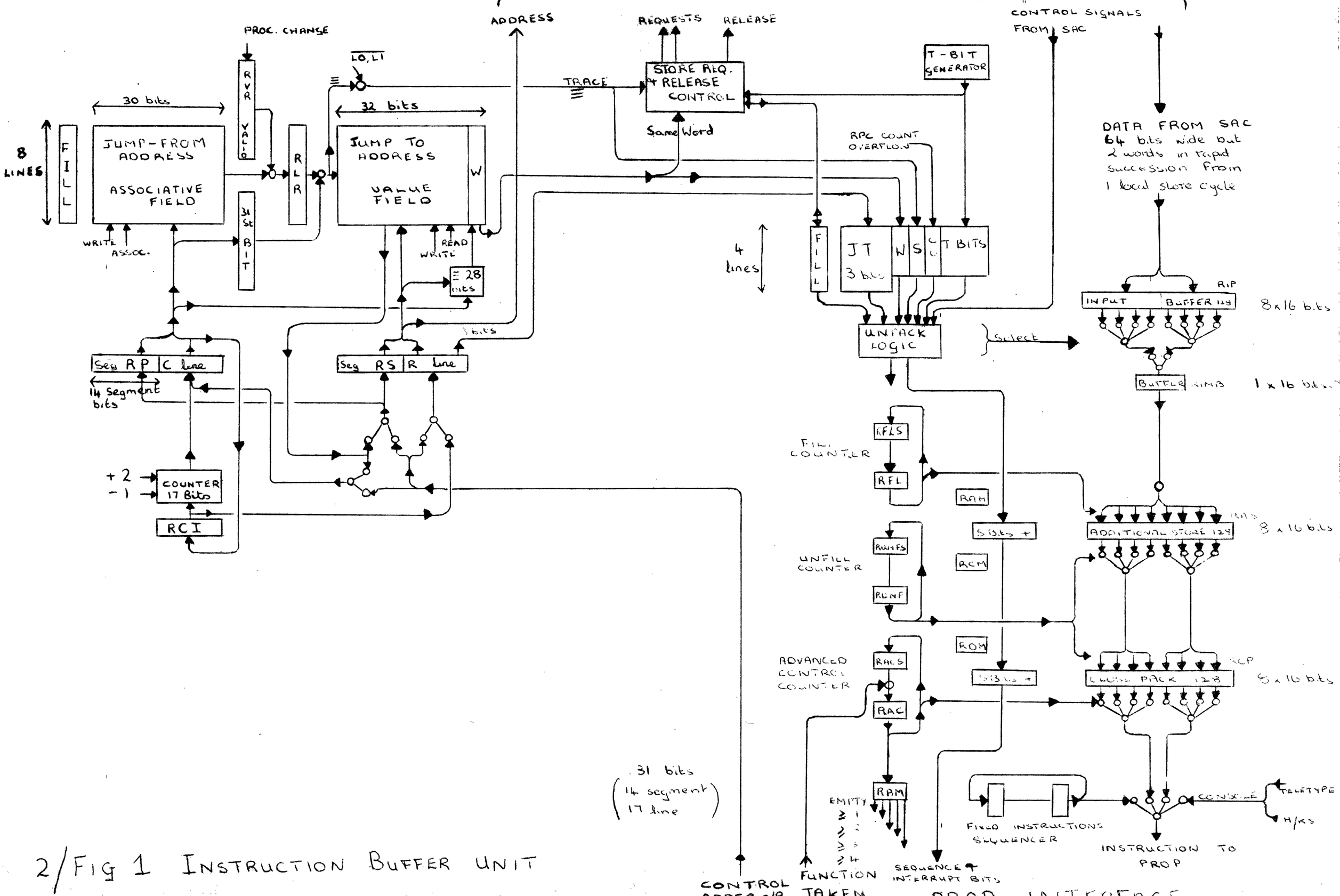
The trace is only cleared on Process Changes OR General Resets.

2.4.3 Jump and Loop Catching

In order to minimise store requests in cases where the jump-from and jump-to addresses are in the same quadword, these addresses are equivalenced on the most significant 28 bits (i.e., store address of a quadword) be ore loading into the trace. Ff equivalence is detected then the same word bit, bit 64, is set and is loaded with the jump-to address into the value field, while the jump-from address is loaded into the associative field, if no equivalence then bit 64 is not set.

Bit 64 indicates to the store request system that when the association predicts a jump to a particular quadword, that request has already been made and so no further request need be produced for that quadword. The normal line loading of the Unpack Record is performed, with bit 64 being set. The T bits retain their original value.

SAC INTERFACE



2/FIG 1 INSTRUCTION BUFFER UNIT

PROP INTERFACE

Chapter 3 Primary Operand Unit

3.1 Introduction

The Primary Operand Unit (PROP) receives instructions from the Instruction Buffer Unit (IBU), and assembles the primary operand required by the instruction in a form suitable for execution of the instruction. The function and associated primary operand are presented to the Central Highway which routes the information to a Central Register for execution of the instruction. PROP is also responsible for the execution of ORGANISATIONAL (f^0) orders.

The PROP is shown schematic form in 3/FIG. 1. Among the features shown are the main working registers involved in the operand assembly process, the major internal and external highways and the Name Store. Not shown are the PROP internal V-store registers, a full list of which is available in Section 3.12. The functions of the various hardware features are discussed under the relevant sections which deal with the operand and organisational processes in detail.

In addition, PROP also contains the Control Register (see Section 3.5), and hardware required to implement the Interrupt feature of the system (see Section 3.6).

3.1.1 Primary Operands - PROP & OBS Name Stores

A primary operand may be any one of the following types:-

(i) Named operand (variable or data descriptor)
(ii) Literal
(iii) Local V-store operand
(iv) Internal Register operand

Named operands are normally obtained from one of two Name Stores in PROP & OBS which are high speed associatively addressed stores forming the lowest hardware level of one-level Virtual Store of the System. PROP Name Store contains 32 lines of 64 bits, 4 of which are reserved as working store for Level 0 Interrupt Routines (see Section 3.11.4); OBS Name Store is part of a larger associative store used for all operands passing through OBS (the Operand Buffer Store) and contains 24 lines of 64 bits.

Named Variable Operands for ACC are kept mainly in OBS Name Store; the remaining named operands are kept mainly in PROP Name Store. A Named Operand cannot be in more than one Name Store at any time; a copy (not necessarily the latest) is always kept in Store (normally Local Store but possibly a Store addressed via Exchange) accessed through the Store Access Control Unit (SAC).

The PROP Name Store is not used for instructions which specify any of the following:-

(a) Literals:- Assembled in the PROP literal registers (see Section 3.10.1).

(b) Local V-store or Internal Register operands:- Obtained by addressing the appropriate hardware unit from PROP (see Sections 3.10.3 & 3.12).

(c) XNB Accesses to Non-Local Names external to Name Segment (see Sections 3.2.5 & 3.10.5).

Conversely to (c), an Operand accessed via a Descriptor may be in PROP Name Store under exceptional circumstances; see Section 3.11.8.

3.1.2 The PROP 'Pipeline'

The PROP consists of six independent stages which an instruction completes as it proceeds through PROP and Central Highway from Instruction Buffer to a Central Register. Each of the six stages can be dealing with a seperate instruction at any one time, thus permitting PROP to act as a 'pipeline' handling six seperate instructions in different phases of assembly.

At no stage before the order is guaranteed to execute is any irreversible action taken by the Pipeline i.e. interrupts can find Control pointing to the order next to be executed. The Pipeline overlap is reduced whenever necessary to ensure that this is so.

At each stage of the assembly process the function bits of an instruction are held in a function register, and are copied through to the next function register as the instruction proceeds along the pipeline. These function registers are shown as RDF, RF1 to RF5 in 3/FIG. 1.

For a simple order (e.g., loading the B-Arith unit with a primary operand in Name Store), the six stages act as below:-

Stage 0; Function decode and shift:

The instruction is decoded, the name shifted relative to the base and the appropriate base selected.

Stage 1; Name-base addition:

The name is added to the base in the Name Adder.

Stage 2; Association:

The operand address is concatenated with Process Number and presented to associative field of the Name Store.

Stage 3; Operand access

The output from each of the 32 lines of the Associative field is copied into the Line Register. Normally one bit position of the Line Register will contain a '1', and this bit position accesses the appropriate line of the Value field of the Name Store.

Alternatively, the required operand may be obtained from an Internal Register or V-store register, etc. as specified by the instruction.

Stage 4; Operand assembly:

The operand (from Name Store, Internal Register, Literal, V-store or Local Store), is available in register RVU or RVF, and appears at the inputs to the appropriate part(s) of the 64-bit Highway (input register RHI), according to the instruction, with sign extension if required.

Stage 5; Execution:

When the previous order in the pipeline has reached a stage where it is guaranteed to complete , the Control Register (RCR) is incremented to point to the order which is now copied into RF5 (see Section 3.1.5); its assembled operand is copied into RHI.

Other actions which the Pipeline is able to perform are described in Sections 3.3 to 3.8.

3.1.3 The PROP 'Juke Box'

The decision at Stage 5 that an order is guaranteed to complete is taken by the Juke Box (see Section 3.9) which then initiates a 'beat' down the pipeline to bring up the next order. To achieve a maximum beat rate of 40 nsec/beat, some complex actions are left until the order requiring them is at Stage 4 or 5 (e.g., V-Store requests, Control Transfers). The pipeline is not advanced until the action is completed.

In some cases, the contents of the pipeline (and thus of the IBU) have to be discarded. This action is initiated by Juke Box and sets all 'valid' orders to 'dummy' orders in the affected Stages and resets certain control bits as necessary.

Due to the Pipeline sometimes advancing more quickly than the IBU can supply new orders, or due to the action of the pipeline and Juke Box, Stage 5 may have neither order nor operand to send to Central Highway. This is known as a 'dummy' order and Juke Box causes a 'beat' to be generated immediately.

3.1.4 The STACK

Operands from any source (central registers, named operands etc.), can be stacked using the Stack Front Register (RSF) as a pointer. Stack Front always points to the current top of the stack location. It is incremented before each stacking operation and decremented after each unstacking operation. Stack locations are set up in the Name Store, but since this is the lowest level of the 'one level' virtual store of the system, the Stack can be extended to any required depth within the Name Segment.

All orders which add an operand to the Stack are Double Orders and are described in Section 3.2.2. Orders which take an operand from the Stack use the Operand UNSTACK (Base =RSF, N=0; SF=SF-1). Orders of the type 'Double Order' using the operand UNSTACK are dummy orders since an operand is switched between a c.r. and Stack and back again. A location above the current top of the Stack is used for writing to and the normal increment/decrement of RSF is inhibited. The order RETURN UNSTACK has the additional action of putting SF=NB before the operand at RSF is used.

If a control transfer or Interrupt occurs while a Stacking or Unstacking instruction is in the pipeline then the position of the Stack Front pointer may not be correct, i.e., if the pipeline contents are discarded the contents of the Stack Front register will not be correct for the next order to be obeyed.

Orders which use the stack are expected to occur frequently, so that to cause all the orders to be executed with the pipeline non-overlapped (see Section 3.2.4) would drastically degrade the performance of PROP.

To ensure that the correct value may be restored if the Pipeline is discarded a copy of the new Stack Front value created by an order altering RSF is preserved with the order as it proceeds down the pipeline (3/Fig 2). When control is incremented for the order this value of RSF is copied into RS6. Whenever the pipeline is discarded, RS6 is used immediately via the Name Base Adder to restore RSF (RSF=RS6+0). When RSF has been altered by an f' order, RS6 is not used when the pipeline is discarded until RSF is incremented or decremented by a Stacking/Unstacking order. Also RS6 is updated as soon as possible following an f' order since the order altering RSF has already been executed.

3.1.5 The Control Register

The PROP Juke Box ensures that a valid order at stage 5 is guaranteed complete before generating the next beat (see Section 3.9). Should an order fail to execute, it is converted to a dummy order before the beat is generated. Thus when this next beat is generated, Control may safely be incremented to point to the next instruction to be executed which will be strobed into Stage 5 as a valid order by this next beat, or by some subsequent beat.

The Control Register (RCO) consists of 14 Segment bits (RCO 33-46) and a 17 bit address (RCO 47-63) defining the first (or only) ½-word (16 bits) of the next instruction to be obeyed.

Control may be incremented (by 1, 2, 3 or 5 according to the length of the last instruction (Section 3.2.1)) for an order which does not jump: it may be RCO = RCO + Increment (RCI 61-63)
or it may be changed by an arbitrary amount by a Control Tranfer (CT) instruction which jumps:

RCO = Operand (RHO 33-63) if Absolute Unconditional

RCO = RCO + Signed Operand (RHO 33-63) if Relative(Un)conditional

The Control Segment may only be changed by an Absolute Control Transfer, so that RCO 33-46 are only strobed by this action and only an 18 bit adder is required. Its output (slave) register is RCA 46-63 where RCA 46 is used to indicate overflow (OV). One input set to the adder is RCO 47-63 (the input to bit 46=0); the other input set is either the Increment (61-63, input into bits 46-60=0) or the Operand (RHO 46-63), bit 46 being taken as the sign bit for Relative Transfers. Except during Juke Box actions, the Increment input set is enabled.

Each beat down the pipeline strobes RCI 61-63 with the value of increment required for the order entering Stage 5. The beat will also strobe RCO 47-63 = RCA 47-63 if the order leaving Stage 5 is 'Valid & Increment-to-Control'(RF5 17,18,19 = 1). After a delay to staticise the Control Adder outputs, RCA 46-63 are strobed.

Any dummy order in Stage 5 (RF517 = 0) is arranged to have an Increment of zero, so that RCA = RCO (next order to be obeyed) until the first valid phase of this order has reached Stage 5 after which

RCA(I) = RCO(this order) + RCI(this order)

If a Control Transfer is to jump (Unconditional, or Conditional & Test satisfied) then the Control Adder is strobed once more by the Juke Box enabling the Operand (RHO 46-63) input set to the adder, instead of the Increment input set.

Thus RCA(J) = RCO 47-63 + RHO 47-63 for a Relative C.T.;

or = Zero + RHO 47-63 (RCO having first been cleared)

and RCO 33-46 = RHO 33-46 for an Absolute C.T.

Once the operand is available, only Relative transfers may fail to execute due to Segment OV so that the alteration of Control Segment by an Absolute CT prior to the next beat is of no consequence.

A Relative CT is trying to cross the Control Segment boundary if

(a) RCA 46 = 1 after RCA(J)

or (b) RHO 33-46 are neither all zeroes nor all ones, i.e. the Operand magnitude is greater than a segment.

This causes a PROP Internal Interrupt and sets the order to dummy (see Section 3.11.6)

RCA46(OV) is not sampled following an increment since the IBU has a similar adder which is pre-accessing instruction ½-words for PROP and so is always in advance of PROP. The action is described in detail in Section 3.3.2 the result being a PROP Internal Interrupt if the OV order leaves Stage 4 and would otherwise be executed.

When a valid order at Stage 5 fails to execute, RCA(I) is brought back to RCO (this order) by the following beat since the Increment = zero for a dummy order.

All Instruction streams are initiated by a 'new instruction stream' request to IBU. The Process Number (RPN) is sent together with RCO 33-46(Segment) and RCA 47-63 (next instruction - I or J). The request may be the result of a CT order (Section 3.10.7), a Fixed Instruction during the Interrupt sequence (Section 3.11.6) or the end of Manual Instructions whereby instructions are sent to PROP via IBU from the Console (see Section 3.8).

The incrementing action of Control may be inhibited from the Console so that Control is only changed by jumps, i.e. only the RCA(J) strobe is allowed.

3.1.6 The Interrupt System

Various conditions can occur during the running of the processor which require immediate attention by the executive software (see MU5 Basic Programming Manual). These conditions are all combined together to form a single signal, which, when it occurs, causes PROP to stop before obeying the next instruction in sequence. Instead, PROP obeys two fixed Interrupt Entry instructions which cause a transfer of control to the appropriate interrupt routine. The final instruction of each of these routines restores the central processor to its previous state and reactivates the suspended process. The routines fall into two classes, each of four types, Level 0 (System Error, CPR $\pm$, Exchange, Peripheral Window) and Level 1 (Instruction Counter Zero, Illegal Order, Program Fault, Software Message) 3/FIG 10 shows the topmost level of gating into the interrupt signal and indicates the inhibit conditions. The eight interrupt type signals are most significantly encoded in the IBU fixed instruction logic to cause entry to the highest priority routine whenever two or more interrupts occur together. Section 3.11.6 explains the PROP actions required to implement Interrupt Entry.

3.2 Instruction Characteristics

3.2.1 Length

An Instruction from IBU may be 16, 32, 48 or 80 bits long (see MU5 Basic Programming Manual). It is read into PROP 16 bits at a time. The function decode of the first half-word read into RDF indicates the size of the name or literal:

(i) Short Orders are those for which k specifies a 6 bits literal n; an Internal Register; a Variable or Data Descriptor (Local Names). They are all 16 bit instructions.

(ii) Long Orders are those for which the c.r. and k bits specify an extended operand (k') type specifying a Named Variable or Data Descriptor or Local V-Store. Unless a Base is specified for which N=0 (bit 13=1), an additional half-word is required to use as the name N. N = 0 instructions are 16 bits only.

(iii) Long Literals are those using k' = 0 and may be 16, 32 or 64 bits long producing an instruction length of 32, 48 or 80 bits.

The additional half-words required are read into RDN under control of the Microprogram Generator of Stage 0. 32 & 64 bit Literals occuring serially as two or four 16 bit operands in PROP are Assembled at Stage 3 into one 32 or 64 bit operand (see Section 3.6.2).

The length of an order determines the Increment to Control required at Stage 5 (see Section 3.8).

3.2.2 Double Orders

The functions B*=, D*=, ACC*=, STACK, STACK LINK and XC0-XC6 require access to two distinct primary operands. For this reason these orders must pass through the pipeline in two seperate phases. This is implemented by the Microprogram Generator in Stage 0.

For each of these orders one of the operands is a stacked quantity so the content of RSF must be modified according to the order. The other operand required for the instruction may involve the use of a Long Literal or a Long Order. The PROP activity in such cases differs from that for long orders of the normal types.

In order to simplify the actions to be taken at the later stages of the PROP pipeline, and in the Central Highway and Central Registers, the function pattern of the first phase of the orders B*=, D*= and ACC*= are modified in Stage 1 of the pipeline so that they appear as B=>, D=> and ACC=> respectively. An additional function digit (bit 18) is used in the RF registers to distinguish the first phase of a double-order. This digit inhibits the modification of the instruction address in the Control Register until the last phase of the double-order is complete. Thus if an Interrupt occurs mid-way through the order, control has not been incremented and the order may be re-executed from the LINK. The individual double orders are discussed in more detail in Section 3.10.6.

3.2.3. Organisational Orders

An Order with cr bits = 0 uses a 6 bit function and is an Organisational ('f') Order of one of the following types:-

(i) Unconditional Control Transfers: either Absoute (to a new Control Segment) or Relative (within the same Segment).

(ii) EXIT or RETURN which load a new link from any operand. RETURN UNSTACK (loads the link from (NB) and leaves the new top of the Stack at (NB-1). Since a new LINK implies a new value of Control, a Control Transfer occurs.

(iii) STACK-LINK which is equivalent to STACK [MS, NB, CO+OPERAND], accesses an operand & Stacks Modified Link.

(iv) EXEC CALL 0-6 which is a STACK [OPERAND] followed by JUMP [Segment 8193, Line 0-6].

(v) SETLINK or XNB => etc which can store the PROP IR's 0-3 (but not BN) to any primary operand only.

(vi) PROP IR Manipulation e.g. NB=SF+ . MS = allows masked (i.e. bit by bit) alterations of Machine Status Reg. The lower half of MS or the Segment register (SN) are only strobed if RMS (13+14+15)= '1'.

(vii) Boolean(X), Boolean(TEST) and Test Transfer. The one-bit Boolean Register RBN, acessible as PROP IR4 (read only) or as RMS07 allows the 'truth' of complicated Boolean expressions to be computed requiring only one Test Transfer (on BN) at the end.

An element of the expression may be another Boolean operand (X) using the Boolean (X) orders which puts BN = fn(BN,X) where X is the l.s. bit of the Operand, and fn is defined by the order. It may instead be the 'truth' of a 'Test'. The Test Registers (RMS04 - 06) are altered as the result of a COMPare order. The TEST is a Boolean element which is true if the current Test Registers are as required by the Boolean(TEST) order, eg TEST (>0) is true if the Test bits indicate that the last COMPare order to be executed found [cr.r.] - [OPERAND] > 0, but is false if < 0 or = 0 was found. The Boolean(Test) order then puts BN = fn(BN,TEST) where fn is defined by the 4 l.s. bits of the Operand.

The Test Transfer order uses the same set of TEST elements as a Boolean(TEST) order, but executes a Relative Control Transfer (see (i)) if TEST is true, else executes an 'Increment Control only' Order, i.e. PROP looks at the next instruction.

The actions caused in PROP by these orders (including the COMP orders) are described in Section 3.10.7 & 3.11.7.

3.2.4 Non - Overlapped Orders

Orders involving PROP Internal Registers cannot be overlapped in PROP. Stage 0 therefore sets the Z-bit (RDF20) which locks out the pipeline from accepting any more orders from IBU while the order is advancing through PROP (with dummy orders behind it). If and when the order reaches Stage 5 to be executed, the I.R. is strobed if required.

If the order is executed, Stage 5 sends RZC (pulse) which clears the Z bit, else when the pipeline contents are discarded the Z-bit is cleared, to allow more orders to be accepted.

The Relative & Absolute Unconditional Control Transfers are also non-overlapped if they have not been predicted by IBU.

All orders may be executed in a non-overlapped mode in PROP by means of a signal from the Console (RDF29).

3.2.5 Special Features

(a) Normally Names are accessed from the current routine name space within the Name Segment i.e. Local Names. To enable accesses to be made to non-Local Names or direct access to data in other segments, another base register XNB is provided. When the segment of XNB (RYB) is outside the Name Segment, and XNB Access is made to fetch the operand from the OBS Store. If XNB is used in this way to access a Variable for B, ACC or PROP, PROP simply by-passes Name-Store and sends or requests the operand from OBS in a manner similar to a Secondary Operand order. If it is used to access a Variable for SEOP or a Descriptor an extra phase is generated by PROP to access the Operand which is returned to PROP before being sent out to SEOP.

(b) Internal Register orders specifiying IR's outside PROP are implemented in two phases; the first to fetch the operand (from the c.r. or IR); the second to send this operand out to the unit concerned (see Section 3.10.2).

(c) Orders storing to PROP Name Store (except B=>) or V-Lines must not leave Stage 3 until the operand is stored since RLR or RVD is used.

These actions are implemented by the Secondary Microprogram Generator in Stage 3 (see Section 3.6.1).

3.3 Action at Stage 0 - Initial Decode

Functions are taken from RDF00-15 and Names Literals from RDN00-15. A Name, after shifting if necessary (Variable 32), appears on GNM00-16. The selected base (RNB, RXB, RSF) appears on GBS00-14 (GBS15 is always '0'). These provide the inputs to the Name Base Adder of Stage 1.

GNM may instead be zero or ± 1 as required for N = 0 or STACK/UNSTACK orders. Stage 1 is able to gate RIO 00-31 or RHO32-63 through GNM00-15. Stage 1 may also gate a base for PROP IR when the Z bit is set and gate the Boolean Register RBN (=RMS07) through GBS15. GBS00-15 provide inputs to RVU (See Operand Assembly, Section 3.7).

Stage 0 contains the Microprogram Generater and IBU Interface logic(3/Fig. 3)

3.3.1 Microprogram Generater

Due to variable Length , Double & Non-overlapped Orders, it may be necessary to expand or delay an order over a number of Stages in the Pipeline after the initial (Function) half-word has been strobed into RDF. Bits RDF17B, 18, 19 are used to define various phases of an order, e.g. (100) is always the first phase of an order and represents RDF17B=1 RDF(18+19)=0. RF117 is put to '1' to propagate a 'Valid' order (one which activates the actions at each Stage as necessary and will add to Control if executed) through the pipeline, and RF118 is a '1' except for the Valid 1st part of a Double Order to inhibit the increment to Control.

Until the IBU accesses and presents an instruction to PROP, RDF17A = 0 so that the function bits are ignored and 'dummy' orders (ones which are ignored by all Stages except to generate a beat at Stage 5) move through the Pipeline; RDF17B, 18, 19 do not change.

When PROP finds a valid order in RDF/RDN, the following actions occur according to the type of order in RDF; RDF is only strobed on (100).

(a) Short, Single Order : one phase only (100) so that one valid order is put into RF1 [on the next beat]

(b) Short, Double Order : two phases
1st part (100) : valid, no increment to Control - put RF118 = 0
2nd part (111) : valid, increment Control - puts RF118 = 1
For *=, RSF is incremented and then the new top of Stack is used to store the c.r. on (100). (111) carries the primary operand used to load the c.r. Since RDN is not used until (111) for this order, it is not strobed on (111).

For the others, the operand is to be Stacked, thus (100) is used to access the operand and (111) increments SF and uses the new top of the Stack to store the operand.

(c) Long Single Order: two phases

(100) : dummy since no name available

(111) : valid, increment Control,

function and name progressing up Pipeline together; Base indicated by function is used.

For N = 0, still two phases (see next section).

For the operand UNSTACK; (100) causes SF to be decremented by one, (111) then uses RSF + 1 being the (original) top of the Stack. RSF is not strobed being the new top of the Stack.

For RETURN/.UNSTACK, RSF is decremented to RNB-1 in (100).

(d) Long, Double Order:

For *=, two phases only since the order's operand is not required until phase (111) :

1st part (100) Valid, no-increment to Control. RSF is incremented and the new top of Stack used to store the c.r.

2nd part (111) Valid. The base used by the primary operand is selected.

For the other orders, three phases are required:

(100) Dummy (no operand available)

1st part (110) Valid, no-increment to Control. The base required by the primary operand is selected.

2nd part (111) Valid. RSF is incremented and the new top of Stack is used to store the operand accessed by (110).

A double order with an UNSTACK Operand is converted to a ''dummy'' action by inhibiting the strobe of RSF. The Stack is maintained by writing to [RSF + 1] in the store part of the order (1st part of *=, else 2nd part).

(e) Long Literal, not Double Order : two,three or five phases:

16 32 64

(100),(100),(100) Dummy (no literal available)

(111),(110),(110) Valid, Function and (1st part of) literal.

(011),(001) Dummy. Second part of 32/64 literal. For

(010) Dummy. Third part of 64 literal. assembly

(011) Dummy. Last part of 64 literal. into(110)

(f) Long Literal, Double Order:

For *=, two, three or five phases as in (c). Actions of the first two phases are as for a Long Double Order (d).

For the others, three or five phases:

16/32	64	
(100)	(100)	Dummy (no literal available)
1st part(110)	(110)	Valid, no-increment to Control. (1st part of) the literal
2nd part(111)	(101)	Valid. RSF is incremented and the new top of Stack is used to store the literal of the 1st part. Also second part 32/64 literal
	(010)	Dummy. Third part of 64 literal
	(011)	Dummy. Last part of 64 literal

For assembly into (110)

Whenever the pipeline contents are discarded, RDF17B,18,19 are set to (000). This is similar to (100) except RDF17A is ignored; the contents of RDF/RDN are treated as invalid.

Further action is taken in the case of non-overlapped orders. Instead of RDF being strobed on (100) after the last phase of the order, the last phase inhibits RDF and puts RF130 to '1': This sets RDF20 and resets RDF17A on the same beat that puts the last phase in RF1 and also maintains the inhibit on the RDF strobe. RDF20 is known as the Z-bit and inhibits the strobe of RF130, 31 (see next section).

Thus RDF is locked out and contains a dummy order, copies of which follow the order phases through the pipeline.

3.3.2 Instruction Buffer Interface

The IBU has an output buffer of eight 18-bit registers which are filled in rotation by instructions from Store accessed through SAC.

Each register contains one 16-bit ½-word of instruction, an 'Interrupt' bit and an 'Out-of-Sequence' bit (both described at the end of this section). Each register is gated in rotation onto the IBU->PROP instruction highway such that the first register to be filled is the first to be gated out.

Each beat down the pipeline which reaches RDF/RDN (i.e. does not stop at stages 2, 3 or 4) causes a strobe 'JIB' to be sent to IBU.(3/Fig 3) Since each register is filled and used asynchronously, the highway may be invalid, changing or valid according to whether the register being gated out is empty (yet to be filled since it was last gated out), in the process of being filled, or has been filled respectively.

JIB occurs a little in advance of the beat reaching RDF/RDN and so allows IBU to sample the state of its registers. If IBU finds that the highway will be static and valid in time, the input to RDF17A is put to a '1', otherwise it is put to a '0'. This affects PROP in phase(100) as described in the last Section.

If IBU finds that the highway is changing or invalid, it leaves the register gated out to the highway. When IBU finds that the highway is valid, the present register remains gated out if RF131 is set to a '1' by this beat.

Each phase of a valid order in RDF puts RF131 to '1' if the next phase does not require data, i.e. RF131 is set by:

(a) Short, Double Order : (100)

(b) Literal 16 or Long Order : (100) only if N = 0

(c) Long, Double Order : (110) except *=

(d) The last phase of a non-overlapped order.
RF131 remains=1 until the order is executed and the next order is about to be stored into RDF/RDN

(e) 'Waiting' (see below) : (100)

When IBU finds the highway is static and valid and RF131=0, the next register is gated out after RDF/RDN have been strobed in time for the next beat.

The 'Waiting' condition is entered if there is a possibility that the highway could become invalid before all the required $\frac{1}{2}$-words of an instruction of length > 16 bits had been strobed into RDN. It is not possible to insert 'dummy' phases since in most cases (e.g. Assembly of 32/64 literals) this would not work. Having strobed a valid order into RDF, PROP will 'Wait' unless IBU has at least the extra number of words required. Each beat samples and then strobes RDF21-24, where:

RDF(20+n) = '1' shows that IBU has $\geq$ n valid 18-bit words in its buffer registers in addition to the register gated out. Thus for a Literal32, RDF22 must be '1' else PROP will wait by putting RDF26 = 1 and re-sampling RDF21-24 on subsequent beats until RDF21 = '1' (since one 16 bit word is already in PROP and the next word is on the highway. For an instruction 2 words in length, the input to RDF17A is strobed with RDF21-24 into RDF17X to end the 'Wait'.

RF131 is also put to '1' while Stage 0 is Waiting since the highway will not be used by the next beat.

While 'Waiting', the Microprogram Generater is inhibited by not strobing RDF17B, 18, 19.

The 'Interrupt' bit in each buffer register is required for the following reasons. The Pipeline in PROP and further stages of buffering in IBU cause the IBU to access instructions ahead of the present one being executed by PROP. Some or all of these may never be executed if the Pipeline is abandoned, so that if IBU is unable to access an order due to Control Segment Overflow (in IBU) or CPR4, then an Interrupt is not sent to the Interrupt System. Instead IBU sets the 'IBU Interrupt' bit and fills up the buffer registers to PROP with those valid but 'garbage' orders, following the last order IBU was able to access.

The action of the Microprogram Generator is amended when an 'IBU Interrupt' order is sensed by RDF28 as follows:

(a) Phase(100) : If the order in RDF is valid then a dummy order is put into RF1 together with 'IBU Interrupt' (RF122). If the order is invalid, then PROP is either locked out by the Z-bit, or is sampling 'old' 'garbage' orders while IBU is fetching the new order stream requested by PROP, ie no effect (IBU Interrupt is not propagated).

(b) Other phases using RDN : RF122 is set along with the garbage word(s),and that phase of the order is set 'non-valid'.

RDF28 is not strobed while Stage 0 is 'Waiting' for more data in case the highway has become invalid.

Should the Pipeline be discarded, all 'IBU Interrupt' bits are reset at each Stage.

See Section 3.11.5 for actions which occur when an IBU Interrupt Order is ''executed'' by PROP. As soon as IBU is able to access more orders, the 'Interrupt' bits will be over-written as the buffer registers are filled.

The 'Out-of-Sequence' bit in each buffer register is required due to the action of the Jump Trace in IBU(Chapter 2), if an Unconditional or Test Control Transfer using a 6/16/32 bit Literal operand has been executed recently by PROP, IBU will send instructions for the ''Jump to'' address should this instruction (recognised by its ''Jump from'' address) pass through IBU again.

Whenever IBU anticipates the execution of a control transfer the 'Out-of-Sequence' bit is set on the last half-word for that order causing the (possible) transfer. This propagates through the Pipeline (RDF16 -> RF16) and is used in the Control Transfer action (see Section 3.10.7) to decide whether to discard the pipeline.

3.4 Actions at Stage 1 - Addition of Name and Base

In this stage the shifted name in RNM and the appropriate base in RBS are added together using a 16 bit parallel adder. The operand address thus formed is set into the adder register RNA ready to be copied into RIN, the input to the next stage.

Since there are 216 32-bit named operands allocated to a process, and since PROP accesses 64-bit operands, only 15 of the 16 adder outputs are required for the operand address. The 16th, the most significant bit, indicates adder overflow or underflow if = 1, and this indication is preserved with the function bits as digit 23. If this bit is set no operand access is performed, and an interrupt is caused when the Control Register indicates this instruction. (See Section 3.11.6).

Stage 1 also contains the Z - Decode for PROP Internal Registers (see Section 3.10.2), and the decode which modifies 1st part of a *= order so that it appears as a => order (see Section 3.2.2).

3.4.1 Adder Input/Output Alignment

The diagram (Fig.4) shows the alignment of all inputs and outputs associated with the name/base adder.

3.4.2 Z - Decode

This supplies the levels required to select any PROP IR to be read or strobed. As copies of the Non-Overlapped function reach Stage 1, two portions of the decode are enabled:

(a) Select IR (if any) to read. Those outputs which select RNB, RXB, RSF, RBN are fed back to Stage 0. RSN, RYB, RMS are selected immediately and gated through GMS00-15. RMS is masked for MS =, unmasked for EXIT/RETURN.

(b) For f' orders only: Select operand (from RHO) for addition and select IR to be strobed.

Select operand (D1ZH0) is sent to Stage 0. This and the strobe select signals are enabled by D53AZ when the valid order reaches Stage 5 for execution. For an order which stores a PROP IR no signal from (b) is enabled.

3.5 Actions at Stage 2 - Association

At this stage the operand address has been calculated in the Name Adder register RNA and the 15 l.s. bits (m.s. bit being overflow or underflow - 3.3), are copied into the l.s. end of Interrogate Register RIN. The m.s. 4 bits of RIN form the process number P, which may only be altered or read as a V-line. The twentieth digit is used to prevent association succeeding during Level 0 Interrupt Routine. The input to this digit position is a level decoded from the Machine Status Register RMS.

The address (l.s. 15 bits) copied into RIN is also copied into the Non-equivalence Register RNQ together with the 32-bit word address RF222 (for 32 bit variables). This is used during the actions involved in dealing with non equivalence in the Name Store, described in Section 3.11.3 and for output with the order.

The operand address is now ready for presentation to the Name Store. This is an associative store, consisting of two main sections. The two sections, or fields, are as follows:-

Virtual Address field	28 lines of 20 bits
Value field	32 lines of 64 bits

The normal action is to present the virtual address of the named operand to the associative field, and if the operand is available in the Value field it is read out. Association takes place on all 28 lines of the associative field. If the virtual address in RIN is identical with an address contained in one of the 28 lines, a '1' signal appears from that line. This signal, gated with a signal from the corresponding digit position of the Line Used Register RLU, is copied into the Line Register RLR, at the next 'beat' of the pipeline. RLU is 32 bits long and each digit position indicates whether or not the content of the corresponding line of the Value field is valid at any time. The RLU register is updated whenever the contents of the Value field are changed. The RLU bits for the executive lines are $= 1$.

Stage 2 also contains decode providing signals to Stage 3. These are used to control Stage 3 and Juke Box.

A 'Stop at Stage 2' signal is generated if a valid B => order is in RF2 whilst there is a previous one outstanding (see Section 3.11.1).

The contents of the 15 l.s. bits of RIN also provide I/p's to RS3 (copy of Stack Front) and to RVD via the V-decode. RVD00-12 enable the PROP V-Store lines (see Section 3.12).

3.6 Actions at Stage 3 - Read Operand

At this stage one of the 32 digit positions contains a '1' (if the association was successful), and thus access may be made to the corresponding line in the Value field. The addressed 64-bit value is read and appears on the digit lines of the Value field.

While the read access is in progress the content of RLR is checked to ensure that the association did in fact succeed. This test determines whether a '1' actually appeared in any bit position of RLR. If RLR contains no '1' bit then 'non-equivalence' is produced.

The effect of equivalence/non-equivalence is described in Sections 3.10 & 11. If RLR contains more than one '1' bit then 'multiple-equivalence' is produced. This will cause an Interrupt (see Section 3.11.4).

Orders reaching RVF/RVU without operands, but which nevertheless require them, are dealt with by the 'Operand Assembly' mechanisms (see Section 3.7).

Equivalence is forced if the order does not require a named primary operand, or if the order is non-valid or if overflow occurred in the Name Adder.

The virtual address of a named operand specified by an order is preserved in the Virtual Address Buffer RVQ during the read phase of operand access so as to be available in RVA in the event of non-equivalence.

Stage 3 contains the Secondary Microprogram Generator which is used to further expand some orders. 32/64 Literals are also assembled at this Stage.

The contents of RLR can be strobed into RBW which together with an equivalence circuit between RLR and RBW are used to implement the overlapped B=> order (see Section 3.10.4).

3.6.1 Secondary Microprogram Generator

The Secondary Microprogram Generator (3/Fig 5) produces the extra phases required to store operands in PROP, to modify the LINK for STACK LINK, and to produce the extra ''orders'' required to implement Double Internal Register, Executive Call and XNB External Descriptor /SEOP Variable Accesses.

No action is taken for B=> named variable orders, nor for ACC=> named variable orders since these latter expect to find Name Store ≠ (see Section 3.10.4).

A *= order has been expanded and modified to appear in Stage 3 as:

1st part (B+D+ACC) => STACK ie named variable

2nd part (B+D+ACC) = [OPERAND]

B or ACC=> STACK is dealt with as above; D=> STACK is dealt with in (a).

(a) (D+f') => named variable or any => any V-line.

The information to store the operand in RHO is contained in RF3 (type and size of operand), and RVD (PROP V-line) or RLR (the line to which a variable must be written) until the order has reached Stage 5 and has been executed, by means of a 'Stop at Stage 3' signal.

When the valid phase of the order reaches Stage 3, a valid, no-increment to Control order (bit 19 = 0) is put into RF4 by the next beat. Subsequent beats put 'increment Control only' orders in RF4. When the first phase reaches Stage 5, the Pipeline is as follows:

Stage 5 : Valid, no-increment

Stage 4 : Increment only

Stage 3 : Increment only (no change after first beat)

Stages 0-2 : No change (beat is stopped at Stage 3)

The order at Stage 5 stores the operand and sets Stage 3 to dummy. This releases the 'Stop at Stage 3' condition and leaves one order only (Stage 4) to add to Control not on the next beat, but on the beat after. This delayed increment is required by STACK LINK and Executive Calls (c+d below)

If the order is the 1st part of D*=, then Control is not incremented by the Stage 4 phase (bit 18 = 0). The second part remains in Stage 2 until the Stop condition is removed and will cause Control to be incremented when it is executed.

(b) STACK. This appears as two parts

1st part accesses the operand

2nd part stores the operand in the variable with base RSF (ie to the Stack).

The 1st part is allowed through. The 2nd part is expanded as in (a) above. After the 1st part (valid, no-increment to Control) has been executed, the pipeline is as in (a) above; the operand to be Stacked is in RHI.

(c) STACK LINK

This is similar to (b) except that the 1st part accesses the operand and modifies the LINK = [MS,NB,RCO+OPERAND]. This is used as the operand to be Stacked by the 2nd part. RCO is not directly accessible and instead RCA is used. Normally, RCA = Control required by next beat, ie for a valid order RCA is the incremented value of Control and RCO = RCA if and only if the order is executed. When the 1st part modifies the LINK, RCA = RCO+Operand to be used when the 2nd part in Stage 4 is put into Stage 5. Thus control cannot be incremented until after the following beat.

(d) EXEC CALL : XCO-6

This is similar to (b) except that the 'Increment only' order is, in this instance, a control transfer to Segment 8193 which is executed in the normal manner except that RCA is set to S(8193) + L(f6-8) instead of the operand.

(e) Double Internal Register Order : (B or D or ACC) =/=> (B or D or ACC) IR

A DIR order stops at Stage 3 for one beat only, to produce two phases:

1st phase : Fetch operand to PROP:- from c.r. if c.r. =>
from I.R. if c.r. =

2nd phase : Send operand from PROP:- to I.R. if c.r. =>
to c.r. if c.r. =

Control bits are set to distinguish an order referring to the c.r. bits (I.R. Normal) or to the I.R. bits (I.R. in B, D or ACC Units). The latter change 'store' to 'load' function and vice versa e.g. 'c.r. => I.R.' is 'IR = cr'.

(f) XNB Descriptor or SEOP Variable Access

An XD access must generate an extra phase in front of that part of the order which initiates the required SEOP actions since PROP does not yet have the descriptor or variable to send to SEOP with the order.

Orders (a) above are held up an extra beat while the XD Access reaches Stage 5 and on execution causes the operand to be returned to PROP.

Similarly for (b), (c) and (d), ie the phases produced are:

1st part, XD : Valid, no-increment : Accesses XD operand
1st part (2) : " " : Accesses 2nd y operand from XD
2nd part (1) : " " : Stacks operand or LINK (Modified)
(2) : Increment only or EXEC CALL Control Transfer
(3) : Increment only : set to Dummy when 2nd part(1) executed

3.6.2 Literal Assembly

When a Long Order is in Stage 3, only a 16 bit literal is complete in RL3. A 32 or 64 bit literal is in the 16 bit registers RL3, RL2 and (64 bit only) RL1 and RDN. These are assembled in Stage 3 to produce one 32/64 bit operand to be strobed into RVU. RDN does not become valid until 30 nsec after RL3 due to the beat being delayed 10 nsec per Stage to avoid one Stage changing while the next Stage inputs are still being sampled.
Thus a 'hiccup' is produced to ensure that the next beat does not come until at least 60 nsec after the last beat instead of 40 nsec. The 10 nsec spare for strobing RL3->RVU means that no 'hiccup' is required for RL2 ->RVU (32 bit literal).

The 'IBU Interrupt' and 'Out of Sequence' bits (see Section 3.3.2) have also to be assembled (ORed) with the corresponding literal registers.

A 16 bit signed Literal Order is sign extended at this Stage to 32 bits for RVU.

3.7 Actions at Stage 4 - Operand Assembly

An operand accessed from the Name Store will appear in RVF; from a PROP Internal Register or a Literal it will appear in RVU. Other operands which cannot be picked up by (this phase of) the order as it progresses through the pipeline are put into RVU to be sent from PROP by means of:

(a) VU-WAIT; for all = named operand orders in Name Store Bypass, = Local V-Store lines.

Juke Box fetches the operand from a PROP V-line; else from Store or the Unit containing the V-line via SAC. The operand is strobed into RVU before Control is incremented for the order at Stage 5. Thus the next beat finds Stage 4 with its operand in RVU.

or (b) Prior phases of the order generated by the Microprogram Generator (Stage 0) and the Secondary Microprogram Generator (Stage 3). These cause a similar result to (a) except that an order is generated which leaves PROP for the Unit concerned and causes the operand to be returned to PROP. Juke Box then, having waited for the operand to return to RHO, strobes it into RVU for use by this phase. Examples of this are DIR, XD (see Section 3.10.5), STACK (1st part).

An Operand destined for PROP may either be accessed as in (a) or (b) above and then sent from RHI to RHO via the Central Highway (PROP -> PROP) or by an order reaching Stage 5 and causing the operand to be sent back to PROP from the Unit concerned. An example of the first is NB = SF+, of the latter is D => named variable.

The operand in RVF/RVU may be gated from the m.s. or l.s. half into the l.s. half of RHI for 32 bit operands. To compile the LINK, the m.s. half of RVU (containing RMS, RNB) is gated to the m.s. half RHI and RCA is gated to the l.s. half of RHI.

A 16 (already sign extended to 32) or 32 bit signed literal is sign extended to 64 bits for RHI.

Stage 4 implements the actions required by a descriptor or a non-ACC variable finding Name Store Non-equivalence or an ACC => variable finding equivalence or an order trying to access the same line in Name Store which is used by an outstanding B => variable order. These are described in Sections 3.11.1-3

The presence of various Interrupts in RF4 will over-ride any other action at Stage 4 and produces a no-increment order to OBS (see Section 3.11.6).

Even though an order has successfully reached Stage 4, it may be prevented from going out or executed by two Lock-out conditions:

(a) A B-order at Stage 4 when a B [ ] order is outstanding

(b) A COMPare order at Stage 4 when a COMPare result is outstanding (see sect. 3.11.7).

3.8 Actions at Stage 5 - Order for Execution

An Order appearing at Stage 5 requires one of the following actions

(a) If subject to a Lockout (see Section 3.11.7) it must wait until this is satisfied.

then (b) Wait until accepted by Central Highway (to B or PROP->PROP) or by Dr Highway or both (B Modifier) or neither (no order leaves PROP; there may be action by Juke Box, eg ACC Write Equivalence).

then (c) Control is handed to Juke Box which, by setting bit FBDGN:

(i) Performs any further actions necessary to process the operand (if any) for Stage 5

then (ii) Answers VU-WAIT providing (i) was successful

then (iii) Generates the next 'beat' which increments Control for the order leaving Stage 5 providing that it was a valid order for which the increment was not inhibited, and that (i) was successful. It also resets FBDGN.

Note that a dummy order generates a beat only, which therefore puts the next order into Stage 5. If the pipeline is abandoned, VU-WAIT is not looked - at and PROP cycles on dummy beats until the first valid phase of the new order stream reaches Stage 5. The generation of a beat returns the status of the new order to (a) above.

The value of the increment to Control is the number of 16 bit half words that were read in by Stage 0 to form the instruction (see Section 3.2.1).

The output registers for (b) are:

Function RF500-15	To Central
Operand RHI00-63	Highway & Dr
Virt Add: Process No RPN00-03	
Segment No GSN02-14	To Dr only
(32 bit) Word Add RHQ00-15	

Control levels and pulses are also sent as listed in 3/Fig 6. All operands returning to PROP via Central Highway (from B, D, OBS and ACC) are copied to RHO and are stored away by Juke Box before the next beat is generated.

Operands for SAC may be taken from RVF or RVU. Operands in RHI are first copied to RHO (PROP->PROP) and then RVU. Operands from SAC return to RVU.

In addition to the 'Stop at --' signals by which means orders may be held up at certain Stages of the pipeline, a beat may be delayed following the increment to Control by the following:

(a) Literal 64 or = PROP V-line Order. RHL0 delays this beat if it would otherwise occur less than 60 nsec after the last one. This allows Literal Assembly (see Section 3.6.2) or PROP V-line access (see Section 3.1.2) to complete.

(b) Console: Step Operation

NUM can be switched from 'Auto' to 'Stop' from the Console to be run on 'Prepulse' or from 'IBU' to 'Manual Instruction'.

On 'Prepulse' a single beat is produced following each propulse received.

'Instruction GO' causes beats to be produced following a propulse until an order is executed, ie Control is incremented.

In 'Manual Instruction' 16 bit words are input from the Function handkeys or the Flexowriter into the Instruction Buffer, which feeds them to Prop as for valid orders from Store. For multi-length orders, each 16 bits is seperated by a 'space' or 'single shot' which causes PROP to send a single beat down the Pipeline to read in that word. For the last word, or for a 16 bit instruction, 'newline' or 'Instruction GO' causes PROP to cycle as normal until the order is executed.

Setting RHD0 or RHD1 delays the reset of FBDCN and so holds up, but does not lose, the beat beyond incrementing Control.

(ii) As soon as all System Error interrupts have been cleared, the routine is not protected against a System Error interrupt occurring before the routine exits, so that the Interrupt Entry action would cause the present System LINK to over-write the OLD LINK in System V-Store. Without further action, this LINK would be used to exit both from the later and the earlier System routines to the earlier System routine instead of the interrupted routine, ie a loop condition results. It may be prevented by testing the OLD LINK to see if it is a System Error LINK. If not then it is copied to the same location immediately. Whenever a System Error routine manages to exit after (1), this location is used to restore the LINK, i.e. the original OLD LINK is preservedand the interrupted routine can be resumed.

Neither (i) nor (ii) are required by other L0 or L1 routines since RMS14&15 provide permanent protection and as soon as the OLD LINK has been restored any masked L0 or L1 interrupts would immediately cause an Interrupt Command. Interrupt Entry would then store the OLD LINK again.

3.11.7 Lockouts in PROP

(i) A 'B' Secondary Operand order sets RBLO bit in Stage 5 until Central Highway receives the B order from Dop. No B orders can be sent out from PROP whilst RBLO is set since it may get to B before the other order via SEOP.

(ii) A COMPare order sets RCLO bit in Stage 5 until the Test Registers have been set in Machine Status (RMS04-06). No other COMPARE orders can be sent out from PROP while RCLO is set since there is no assured order in which the test results will return.

Thus should (i) or (ii) occur Stage 5 is in 'Lockout' until the lockout is reset.

Orders which use the Test Registers cannot be executed until RCLO is reset. Thus:

(iii) Interrupt Entry: the Fixed Instructions store RMS. To ensure RMS04-07 are valid Juke Box does not strobe RIE until RCLO has been reset.

(iv) Boolean(Test): the Boolean Register BN is not Set/Reset by Juke Box from the results of fn(BN, TEST) until after JRCLO is reset and TEST is valid.

(v) Test Transfer: the Test Transfer Jump/no-Jump decision is not made by Juke Box until after JRCLO is reset and TEST is valid.

3.9 The PROP 'Juke Box'

The Juke Box provides all the pulses necessary to sample and manipulate information contained in PROP (functions, operands, state-of-PROP). Orders are advanced through the Pipeline by a series of 'beats', each one of whi h puts a new order (possibly a dummy) into Stage 5. When the order has been accepted by any relevant highway(s), a bit FBDGN is set and remains set until the next beat is generated and propagates down the Pipeline (3/Fig 7). Owing to the distance at which sub-units of PROP are placed from Juke Box, their strobes are up to 20 nsec behind the corresponding function registers RDF, RF1-5. Since data for the sub-units is generated from these registers and from RDN (also local to Juke Box) this is of no importance for data flows. Also Central Highway and Dr highway are near or on the correct side of the output registers. It does however limit the response of PROP Pipeline to Name Store conditions and manipulations (see Section 3.11 and 3.12).

3.9.1 Juke Box Routines

Before FBDGN is set, the last beat strobes RWTO-8 which informs Juke Box of any WAIT conditions. More than one can occur and they are dealt with in the following order (unless the pipeline is abandoned)(3/Fig 8):

(a) B => order returned.
RHO contains the operand requested from B by the overlapped B => order. The B => Routine uses RBW and RF3 information to store it in Name Store (see Section 3.10.4).

(b) Interrupt order
PROP has sent out an Interrupt Order. When any outstanding COMPare Order returns, Juke Box strobes RIE which sets Interrupt Entry Mode (see Section 3.11.6) and then abandons the Pipeline and sets Stage 5 to dummy, ignoring other WAIT'S.

(c) DR - WAIT
Exept for named variables and literals which use an advanced handshake (DR guarantees to accept this and the next), Juke Box waits for Dr to say 'Order guaranteed to Complete in Dr' before continuing. Dr may instead say 'Abandon' or 'Procedure Call'. The former implies a STS order is being curtailed, the latter that the descriptor sent by PROP requires fixed instructions from IBU (See Section 3.11.8.) For these last two the pipeline is abandoned.

Dr may also send with 'Order Complete' a level sayinng 'Current Name Segmont Equivalence'. This resets any VU-WAIT and sets '(DA=)CT'

(d) AW=

This is answered (see Section 3.11.2) and any further WAIT'S are ignored since these apply to the order still in Stage 4.

(e) PROP ≠

PROP has sent out a Name Store Access Order. The PROP ≠ action is entered (see Section 3.11.3). Should SAC or OBS be unable to return the operand due to CPR ≠, then (b) is entered; otherwise any further WAIT'S are ignored as for (d).

(f) CPR-WAIT

PROP has sent out an order which requires CPR = to succeed. These are described in Section 3.11.9.

Following a CPR ≠ these would result in PROP otherwise either halting or using an invalid operand. This is avoided with co-operation with OBS; Juke Box enters (b) if the CPR-WAIT is not satisfied.

(g) HO-WAIT

PROP has either sont out an operand from RHI or is waiting for one to return from B, SEOP, OBS or ACC via Central Highway to RHO. When the operand returns, FHORDY is set. If the operand is from B and a B => operand is outstanding, then (a) is entered, holding up (g) until the next operand appears. One of various routines may be entered following the return of the operand:

(i) Boolean or Test (see Section 3.10.7)

(ii) RHO -> RVU (required by Double Orders and XNB Descriptor (Sections 3.10.5 & 6)

(iii) Control Transfer (see Section 3.10.7)

(iv) => Routines using RF3 etc used for => Variable in PROP Name Store; => PROP V-lines; => System V-Store or => variable in NSBP; => External V-Store (see Section 3.10.3)

(v) f'(PROP IR) (see Section 3.10.7)

A Control Transfer may cause the pipeline to be abandoned, as may => NSBP following CPR≠. The latter would also cause Stage 5 to be set to dummy. Stage 3 is set to dummy only if rrequired by the Secondary Microprogram Generator (see Section 3.6.1).

(h) VU-WAIT

PROP has an order at Stage 4 requiring an operand from PROP V-Store or one accessed directly from SAC, i.e. = V-Store, = PROP V-line, = External V-Store or = name in NSBP (see Section 3.10.3). Due to CPR≠, which may occur for = System V-Store or = NSBP, the operand may not be available. If SAC is unable to supply the operand the exit is still normal since a CPR ≠ Interrupt will have occured by the time the next beat is produced. Since EX+LO+L1 is set, this would be a System Error.

(i) (DA=) C.T.

If HO-WAIT did not abandon the pipeline, this forces it by means of a Control Transfer using (g)(iii) to the current (RCA). This is the next order required by PROP and is actually in Stage 4, but it may no longer have an operand (see Section 3.11.8).

3.9.2 Abandoning of Pipeline

If the pipeline is abandoned, Stages 0-4 are set to dummy orders and IBU is asked by Control Transfer, Interrupt Entry or Procedure Call to supply a different order stream (the last two being fixed Instructions from IBU itself). Unpredicted literal (6, 16 or 32 bit) Control Transfer codes which jump, also cause the Jump Trace of IBUto be updated with the ''Jump from'' (= Control for next order if no jump -1) & ''Jump To'' (= Control for next order after jump) Addresses (16 bit boundary).

3.10 Instructions Through PROP - Normal Actions

The following Sections describe the Actions necessary to execute orders from PROP assuming

(i) the order is executable (e.g. the operand is available without software action),

(ii) any PROP Name Store access has produced its expected result,

(iii) No Interrupts occur externally nor are any made to occur by invalid functions in PROP.

(iv) No interactions with other orders or Units occur (e.g. not BW=(order) or DA=(Unit)).

These exceptions are dealt with in Section 3.11.

Certain orders via OBS have to wait for CPR = before Juke Box can execute them. For a list of orders and the action occuring on CPR =, see Section 3.11.9.

B, D and ACC *= orders are dealt with as adjacent => STACK and = orders since the order is split into two by the Microprogram Generator (Stage 0) and the function of the first part modified to => (Stage 1). The relevant Units then interpret the *= function as = only.

Orders expanded by the Secondary Microprogram Generator (Stage 3) are traced in their expanded form.

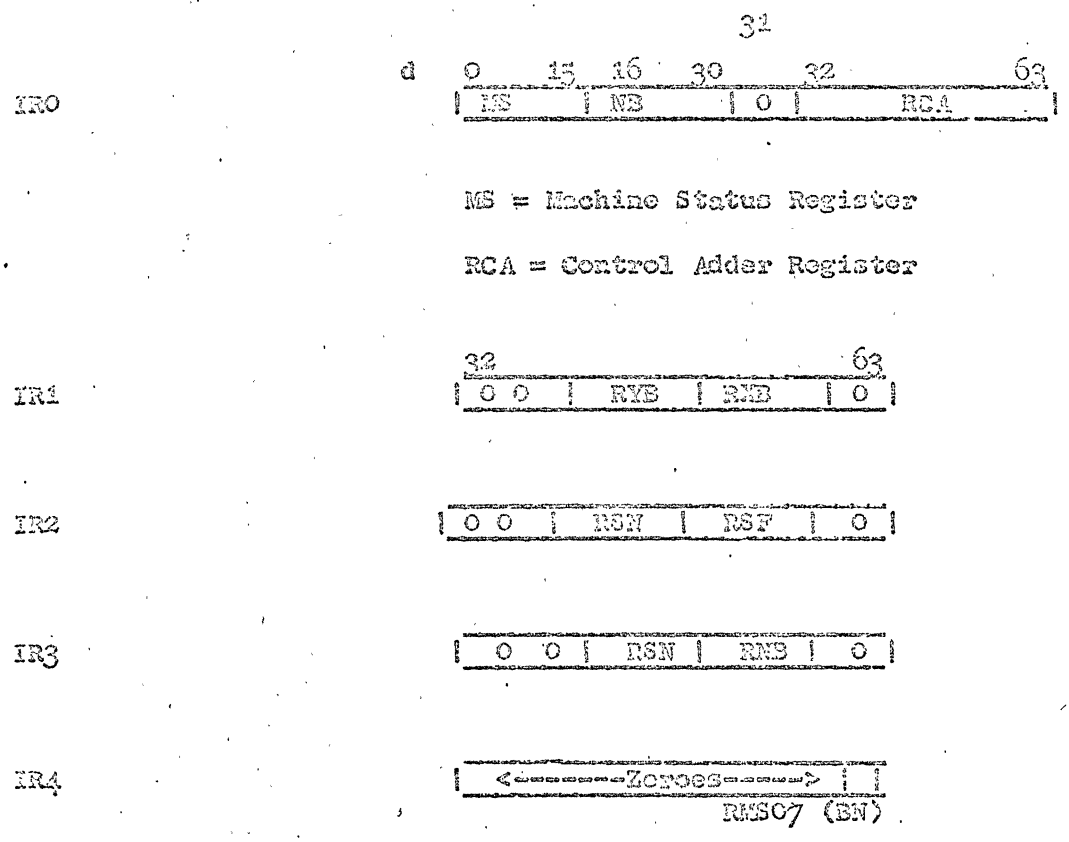
All operands leave PROP from RHI for other Units and return from them to RHO.

3.10.1 Literal Operands

Any => instruction causes a PROP Internal Interrupt and no other action. A = instruction assembles the literal (See Section 3.6.2) from RDN, RL1-3 as required into RVU at Stage 4. An f = [that is bits (0+1+2) of the order code ≠ 0] order leaves Stage 5 to load the c.r. from RHI. An f' = [that is bits (0+1+2) of the order code = 0] order at Stage 5 sends a 'PROP->PROP' request to the Central Highway which copies RHI to RHO. Juke Box then takes the necessary action.

3.10.2 Internal Register Operands

All PROP IR orders [bits (10+11) = 0] are non-overlapped in PROP. Any => instruction specifying a PROP IR causes a PROP Internal Interrupt and no other action. An = PROP IR order picks up the operand via GBS00-15 and GMS00-15 into the ms.½ of RVU at Stage 4. Except for IR0, the IR is shifted to the l.s. ½ of RHI in Stage 5, digits 00-31 being all zero. For IR0, known as the LINK, the l.s. ½ of RHI is loaded with RCA[= RCO (for the next order)]. The PROP IR'S are:


31
d
0
15
16
30
32
63
IR0
MS
NB
0
RCA
MS = Machine Status Register
RCA = Control Adder Register
32
63
IR1
0 0
RVB
RNB
0
IR2
0 0
RSN
RSF
0
IR3
0 0
RSN
RMB
0
IR4
<-------Zones------>
RMSO7 (BN)

Orders for Internal Registers not in PROP [bits (10+11) ≠ 0] are Double IR orders. The 1st part accesses the operand (from a c.r. if a c.r. =>; from an IR if a c.r. =) by sending an order from Stage 5 to the donor Unit (B, D or ACC). The operand is returned to RHO by Central Highway. Juke Box copies this into RVU to provide the operand for the 2nd part in Stage 4. The 2nd part reaches Stage 5 on the next beat and sends the operand (to an IR if a c.r. =>; to a c.r. if a cr =) to the acceptor Unit (B, D or ACC). Control bits with the order are set if PROP is referring to the IR function bits instead of the c.r. bits, ie 'IRA' with B = ACC IR means send the contents of ACC Unit IR (n) back to PROP. Conversely 'IRD' with B=>SEOP IR(n) means load SEOP Unit IR(n) with the operand supplied. Operands from Dr return via OBS and Dop to Central Highway.

From the structure of the order code it is impossible to define an f' order using IR operand. However an IR may be stacked and then used by an Unstacking f' order. The PROP IR'S are also stored and loaded by the f' orders.

3.10.3 V Store Operands

Any V Store order will cause a PROP Internal Interrupt and no other action unless one or more of the Executive Mode, Level 1 Interrupt or Level 0 Interrupt digits are set in Machine Status (RMS 13+14+15 resp.), since these are 'priviledged' operands. Local V-Store is divided into Blocks which PROP divides into three categories:

(i) Block 0: System V-Store, implemented as Segment 8192 of the Virtual Store. An operand is accessed via SAC (3/Fig 9 shows the SAC interface signals) making a normal = or => request as necessary. SAC receives RSN gated to appear as Segment 8192. Although the address is treated as a Virtual Address, the operand is never in the Name Store (Segment 8192 is accessable by Executive routines only and the software never sets RSN = 8192). The Name Store is therefore by-passed.

(ii) Block 1: V-lines within PROP itself. Some of these have special actions which are initiated by a = or => as appropriate (see Section 3.12).

(iii) Blocks 2-127: Local V-Store in other Units. These are accessed via SAC by making a V-Store = or => request as necessary.

A = V-Store order sends a signal to Juke Box which causes a VU-WAIT for an order in Stage 5 while this order is in Stage 4. Juke Box accesses the operand either directly from the PROP V-line for Block 1, else via SAC for Blocks 0, 2-127. The operand is put directly into RVU and enters RHI as the order is put into Stage 5 to go to the Unit concerned (cr or f').

All => V-Store orders stop at Stage 3 to save RVD until the order is executed. An f' => V-Store order is non-overlapped and has picked up the operand from PROP as it passed Stages 4 and 5. The operand is sent from RHI to RHO (PROP -> PROP). A cr => V-Store sends out an order from Stage 5 to the donor Unit which returns the operand to RHO.

For Block 1, Juke Box loads the V-line given by RVD and then enters the PURGE or SEARCH Action if required. For Blocks 0, 2-127, Juke Box stores the operand via SAC using RVA as the address (= RHO since the order is still at Stage 3).

3.10.4 Accesses to PROP Name Store

All 32/64 bit variables and 64 bit descriptors (with Base SF, 0, NB, UNSTACK) use Segment SN and access the Name Store, as do those using Base XNB within Name Segment (RYB = RSN). [No access is made for D[ ] since no descriptor is required - it is already in Dr.] The ''normal'' result for each access (i.e. the condition for PROP to work at maximum speed) is as follows:-

(i) ACC => Variable: expects non-equivalence following l.c. Association since all operands written by ACC must be in (or else be transferred to) OBS. No operand is sent with the order; OBS accesses its own Name Store using the full Virtual address sent with the order from RPN, GSN, RHO. The equivalence result is known as 'AW =' (see Section 3.11.2) and causes the line to be erased from PROP Name store, updating store if necessary.

(ii) ACC = Variable: either equivalence or non-equivalence is 'normal'. If equivalence occurs then a copy of the operand will be sent from PROP (this order does not cause ACC to alter the operand value). If non-equivalence occurs, OBS will access its Name Store as for (i).

(iii) all other accesses expect equivalence since they should be kept in PROP Name Store. If , however non-equivalence is found in PROP (known as PROP $\not=$) and the operand is then found to be in OBS Name Store, it is only written back into PROP and erased from OBS Name Store if the Name Store $\not=$ occurred for a B, D or f' => Variable; otherwise a copy is sent from OBS to be used by this order only. If it is not in OBS, then PROP innserts the operand into Name Store from SAC for all these accesses (see Section 3.11.3).

It is necessary to swop or send copies of operands between PROP and OBS since a name may be used by PROP (f'), B or D or ACC. B, D and f' operands are accessed from PROP, but ACC operands are accessed from OBS. Also 32 bit Variables, even though the programmer should wish to use them specifically by ACC or non-ACC orders, may be mixed in the same 64 bit word. The swopping is minimised by only swopping Name Store entries for => orders finding operands in the other Name Store.

No time is lost for ACC = getting equivalence, but time is lost for B, D or f' getting PROP $\not=$ and OBS $=$, for B, D or f' getting PROP $\not=$ and OBS $\not=$, and for ACC => getting $AW =$

(a) An ACC order with a Named Variable operand reaching Stage 5 will send an order to SEOP causing OBS to make an access using the PROP Virtual Address, unless the operand is being supplied for (ii). No operand returns to PROP.

(b) A B or D = Variable reaches Stage 5 with an operand which is sent to the relavent Unit. An f'= (e.g. MD=) produces a PROP -> PROP order to Central Highway which transfers RHI -> RHC, which Juke Box then uses.

(c) B => Named Variable is overlapped by means of 'B=> outstanding' (Stage 2), RBWregister (Stage 3), and the RBW-bit (Stage 5) which is only set when this order has left Stage 5 for B and been executed (i.e. Control has been incremented). When the operand has returned to RHC it is normally sampled by PROP when the present order is put into Stage 5. Juke Box immediately stores the operand in the Name Store using the RBW register (its 'home' line) and RF2 information as to size (32/64), and if 32-bit which half of the 64 bit line is to be over-written. (A 'B' variable may validly be 64 bit providing bits 00-31 are all zero. Loading B with a 64 bit variable will cause m.s. 32 bits to be ignored.)

When the B => operand has been stored, it is necessary to re-cycle the Name Store to re-associate/read for Stages 2 & 3 since LR is destroyed by => Name Store from the RBW register. The Virtual Address at Stage 3 (RVQ) is put into RIN, associated and the result used to restore RLR. The Virtual Address at Stage 2 (RNQ) is then put back into RIN (destroyed by the last action). Finally, 'B => outstanding' and the RBW bit are reset.

Should the pipeline be abandoned at any time, the RBW-bit is not reset; 'B => Outstanding' is only reset if the RBW-bit is not set (ie the B => order is still in PROP).

(d) D => Local Variable is not overlapped, but stops at Stage 3 (see Section 3.6.1). When the order (Valid, non-increment phase) reaches Stage 5, it is sent to Dr which returns the operand to RHO via OBS, Dop and Central Highway.

The operand is stored in the Name Store using RLR (the 'home' line) and RF3 information as for (c). No recycling is needed since no other order is in Stage 3, and Stage 2 has not been disturbed.

Stage 3 is then set to dummy and Control is incremented after the next beat.

(e) f' => Local Variable (eg SET LINK) stops at Stage 3 and picks up the PROP I.R. in Stages 4 and 5. A PROP -> PROP order loads RHO which is used as in (d) above.

(f) Descriptors are accessed as for D = Variable. SEOP loads the order from Stage 5 into its Descriptor Register and uses it to access the Secondary Operand (Bit, Byte, String,etc) as defined by the descriptor. OBS then accesses and stores the Secondary Operand in one of eight 128 bit lines in its Operand Store. For ACC orders, no further action is required by PROP and Control may be incremented.

For some orders a CPR-WT is necessary (see Section 3.11.9) and Control is not incremented until CPR = is obtained. For a B order it is then necessary to set the RBLO bit which locks out the B until B is loaded by or has stored the secondary operand (See Section 3.11.7). A D= [] order is executed by two passes through Dr in SEOP.

For an f'= [] order, a CPR-WT is necessary. The operand addressed by Dr is accessed by OBS and returned to RHO via Dop and Central Highway to be used by Juke Box. An f'=> [] causes a PROP Internal Interrupt since Descriptor and f' operands would both be in RHI.

If the Descriptor is to be modified by B (eg ACC = S[B]), PROP does not send the order to Dr until a copy of it has been accepted by the Central Highway for 'B'. When both orders have been sent, the order is said to have gone from Stage 5. On receipt of the modifier request, the B sends its value to Dr (via a separate Highway).

For Base 0, D = D descriptor accesses, PROP accesses Name Store but does not require equivalence or non-equivalence on Association. Thus the named operand from Line Zero of Name Segment may appear in RHI. Otear PROP actions are the same. Dr uses the existing contents of the Descriptor Register.

3.10.5 External Names

These use Base XNB and Segment(XNB) (if RYB $\neq$ RSN). They cannot be in PROP Name Store, but may be in OBS Operand Store. No access is made by PROP but an order is forced out to OBS via Dr which uses the supplied full Virtual Address (RYB is gated out, not RSN).

(a) A B External Variable is similar to a B Descriptor order except Dr is not used to form the address.

(b) An ACC External Variable leaves PROP without an operand as for non-equivalence with a Local name.

(c) An f'= External Variable is similar to an f'= Descriptor, the operand being returned to RHO from OBS via DOP and Central Highway. f'=> External Variable causes an Interrupt.

(d) An SEOP External Variable or a Secondary Operand access using an External Descriptor involves the use of SEOP twice. For an SEOP load order or for a descriptor access, an extra phase is produced at Stage 3 (the XD access, section 3.6.1) in front of the first normal active phase. This is a valid, no increment to Control order which goes to OBS via Dr. OBS accesses the operand and sends it back to PROP (RHO) via Dop and Central Highway. Juke Box then copies RHO to RVU to align with the first normal active phase as its operand. All following actions are unchanged.

For an SEOP store order the extra phase accesses the appropriate central register in SEOP (DR or XDR) and returns its content via OBS, Dop and Central Highway to PROP (RHO). Prop does not use the operand but Juke Box can now send the first normal active phase of the order out as a Variable/XNB access. This is treated as a normal store order by SEOP and OBS (i.e. as for a B or ACC => Variable XNB) except that when the order reaches Dop the operand has already been loaded into its buffer by the first phase of the order and it proceeds with its normal store order action immediately.

3.10.6 Double Orders

(a) STACK is split into two parts at Stage 0 (see Section 3.3.1), the second part stopping at Stage 3. The 1st part accesses the operand as for an f'= order. Juke Box then copies RHO to RVU to be used by the 2nd part. This is identical to an f'=> Local Variable using the accessed operand instead of a PROP IR .When the next beat puts the 2nd part (Valid, no-increment to Control) into Stage 5, the operand is stored in the new Stack Front location. RSF has been incremented by the first part in Stage 1. Control is incremented after the following beat.

(b) STACK LINK is expanded as for (a). It is non-overlapped.

The 1st part accesses the operand as for an f'= order. Juke Box then uses RHO to modify Control, i.e. [RCA] = [RCO(this order)] + [OPERAND]. Together with the ms half of RVU (containing RMS, RNB), the 2nd part (valid, no-increment to control) copies RCA into RHI to form the LINK(Modified), c.f. PROP IRO (Section 3.10.2), which is then stored, and Control incremented, as above.

(c) EXEC CALL 0-6 is expanded as for (a):

The 1st part and 2nd part (Valid, no-increment to Control) Stack the Operand as for (a). The 'add to Control' phase of (a) is now a valid JUMP order (see below) which forces Segment 8193 + Line 0-6 into RVU at Stage 4. When this phase reaches Stage 5, a PROP -> PROP order is sent to Central Highway (as for f'=) and Juke Box executes an absolute jump (Control Transfer) to this address. At the same time Executive Mode digit (RMS13) is set to allow IBU to access the Executive Call CPR.

3.10.7 ORGANISATIONAL Orders

STACK LINK, XC0-6 have been described above. The remaining f' orders fall into two sets: f'=> and f'= type orders. The f'= are further divided into PROP IR manipulation, Boolean (X,TEST) and Control Transfers:

(a) f'=> orders are

SET LINK which stores an unmodified, non-incremented LINK to a Name (see IRO Section 3.10.2), ie RCO is for this order.

XNB => which stores the non-Local Name Segment and Base registers (IR1)

SF => which stores the Local Name Segment and Stack front registers (IR2)

NB => which stores the Local Name Segment and Base registers (IR3)

There is no BN=>.

Unlike the PROP IR orders these store to an operand instead of a cr. As for PROP IR these are non-overlapped assembling the registers into RVU and RHI in Stage 4 and 5.

The LINK is non-incremented to a Local Name since it uses a delayed 'increment Control' phase for => Name Store.

(b) PROP IR Manipulation

Thirteen f' orders load or increment registers in PROP:

(i) EXIT and RETURN load the LINK, i.e. [MS, NB, CA] = [OPERAND]. Juke Box thus strobes RMS, RNB and RCA from RHO and then executes a JUMP' to this new value of Control. [RCC] = [RCA] by the next beat as normal. When using RETURN/UNSTACK, the LINK is taken from a reduced Stack where SF = NB. Stack Front register is left at NB-1 (see Section 3.3.1(c)). This returns the (modified) LINK which was Stacked before the Routine was entered.

(ii) MS = permits separate bits of RMS to be changed at will by means of masking bits in the 32 bit operands; the 1s half of RMS is priviledged.

(iii) DL = permits a 32 bit Display register to be loaded from RHO. The display lights are situated on the Console.

(iv) XNB = and XNB+ allow the Non-Local Name Base register to be loaded or incremented.

(v) SF=,+;NB=,+ allow Local Name Base and Stack Front to be loaded or incremented relative to themselves or each other, e.g. SF=NB+3 sets RSF=RNB+3. If a segment boundary is crossed, then an Interrupt occurs.

(vi) SN = allows SN to be loaded if RMS(13+14+15) = 1; if not it is a dummy order.

(c) BOOLEAN (X, TEST) Orders

The l.s. bit or l.s. 4 bits of the operand are used as required (see Section 3.2.3). When the result is valid, BN is set to a '1' if the Boolean function fn is 'true' else to '0'.

(d) Control Transfer Orders

When the operand appears in RHO, RCA is pointing to the next order.

For a TEST Transfer for which the TEST is true, or for the unconditional tranfers, JUMP and ->, Juke Box replaces the RCI input to the Control Adder by RHO and strobes - RCA(J) = RCO (this order) + RHO (relative transfer) or resets RCO and strobes RCA(J) = RHO (absolute transfer).

A relative transfer greater than a Segment or which crosses a Segment boundary (Control Adder overflow) causes an Interrupt.

Whether the pipeline has to be abandoned depends on the following:

(i) If the LU has anticipated the transfer (the Sequence bit = 1 which says the following orders start from RCA(J)) and a transfer is to be executed.

(ii) The IBU has not anticipated the transfer (Sequence bit = 0, therefore next order is from RCA(I)) and a transfer is to be executed.

For (i) the pipeline is correct, but for (ii) it must be discarded and IBU asked for access RCA(J). After the execution is complete, RCO (next order) = RCA(J) and PROP feeds in dummy orders until IBU provides the first instruction. Either way, the next instruction obeyed is from RCA(J).

For a Test Transfer for which the TEST is false, no transfer is to be executed and RCA(I) is left unchanged. Then the following applies:

(iii) The IBU has anticipated the transfer yet a transfer is not to be executed.

(iv) The IBU has not anticipated the transfer and a transfer is not to be executed.

For (iv) the pipeline is correct, but for (iii), the same procedure must be applied as for (ii). Either way, the next instruction obeyed is from RCA(I).

IBU is able to anticipate control transfers by means of a Jump Trace (Chapter 2) which PROP causes to be updated for each case (ii) for Literal 6/16/32 Test Transfers or Unconditional Transfers. Before sending RCA(J) (the Jump To address), PROP sends RCA(I) from which IBU produces RCO(I)-1 (the Jump From address).

The instruction Buffer sends a sequence bit to PROP with the last ¼-word of each instruction. For 6 or 16 bit Literal Transfers, it reaches Stage 5 with the active phase of the ordor. For Literal 32, it is in Stage 4. It will not be set for a 64 bit instruction since Jump Trace is never loaded for this instruction.

Control Transfers are also produced by EXIT and RETURN and by the last phase of XCO-6. RCA is also strobed by the 1st part of STACK LINK to form the modified link. This returns to RCA(I) following the next beat after which RCO = RCA(I) as normal, ie this is <u>not</u> a transfer.

3.11 Instructions Through PROP - Exceptional Actions

These are necessary for the working and updating of the Name Store or arise from interactions between Orders in PROP or in other Units of MU5 and in general impose a time penalty on PROP execution rate.

3.11.1 B => Outstanding

This applies to B => Local-Variable (see Section 3.10.4). If a second B => Local-Variable Order was allowed beyond Stage 2 before the first's operand was Stored, it would interfere with the store action by trying to use the register. A 'Stop at Stage 2' signal is therefore set until ''B=> outstanding'' is reset.

Also any Name Store Access might try to use or write to the line to which the B => operand is writing. This is 'BW =' and arises when ΣRLRj.RBWj = '1' after an access. One of two bits JBWEA/B is set at Stage 4. These are 'Stop at Stage 4' levels and cause the Stage to set dummy orders into Stage 5 until the B => operand is sampled (see Section 3.10.4) as being in RHO. Having stored the operand, Juke Box checks for BW = and if so makes an access using RLR = RBW, i.e. reads out the updated operand into RVF which previously only held the old operand. The RBW register and JBWEA/B are then cleared thus removing the bit in RBW (i.e. no more BW =' s possible until another B => outstanding) and letting the order into Stage 5 as normal on the next beat.

3.11.2 ACC Write Equivalence

This has a similar action to BW = in that 'Stop at Stage 4' is produced by bit JACE following the detection of ΣRLRj = '1' for this order. Only one dummy is produced in Stage 5 which allows Juke Box to write the operand to SAC if if the line is Altered (a bit set for each line which has had a => Name Store action performed on it). RVF is used as the operand and RVA as its Virtual Address in Name Segment. RVA is put into RIN and by Association strobes RLR with the line to be set non-Altered. Non-Used.

The Name Store is then re-cycled to re-form RLR and RIN for Stage 2 and 3. JACE is reset which allows the order to enter Stage 5 without an operand as assumed. OBS will then access SAC for the operand. OBS cannot, as for f'=>, store an operand coming from Dr, so that the AW = is required to update Store before sending the order out.

Should the AW = occur simultaneously with a BW =, then 'AW =' is not actioned until the B => is complete.

3.11.3 PROP ≠

Following the detection of non-equivalence (ΣRLRj = '0') for a Name Store access needing equivalence, bit JNSNE is set which causes a 'Stop at Stage 4' signal. A Valid, no-increment to Control order (the NSNE Order) is produced by Stage 4 which leaves Stage 5 on the next beat and goes to OBS via Dr. This order is similar to the XNB non-Local operand access in that this extra phase in front of the order causes OBS to make an access. This access is to its Name Store, however, and checks for the presence of a 64 bit line that has been used by ACC Local Names. If such a line is found (OBS =), the NSNE Order is put into the By-pass Register until the ACC Queue has run down. The 64 bit line is then sent back to PROP (RHO) via Dop and Central Highway.

If no such line is found (OBS ≠), OBS makes a = Store access to SAC on behalf of PROP. This causes the operand to be returned from SAC directly to PROP (RVU). This is the expected action on PROP ≠. The NSNE Order is discarded by OBS after the request is made.

Before putting the NSNE Order into the Bypass Register (on OBS =), OBS makes a similar request to SAC as for OBS ≠ except that no operand is read from Store and no test is made for CPR = on this order. However, either request causes 'SAC-OK' or 'SAC-NK' to be sent back to both OBS and PROP. Following OBS ≠, 'SAC-NK' implies that no operand will be sent back to PROP (NSNE Order is discarded anyway). Following OBS =, SAC-NK implies that SAC is already in Run-down Mode during which no operands are sent to or from Store (except for forced => requests - see below). The ACC Queue in OBS is also in a Run-down mode in which it would 'jamb' if there is an order which requires an operand from SAC which is not being accessed. Thus the NSNE could be caught in the Bypass Register which is not permitted. This is avoided by OBS discarding the NSNE following 'SAC-NK'.

Whilst the NSNE order is propagating to OBS, PROP prepares its Name Store to receive the operand (either from SAC or OBS). The 'Next-Line Pointer' (RLP) is used to cycle round the Name Store as more entries are required. Obviously the oldest entry is over-written by this action. In case this is the one to which an outstanding B => operand is to return, PROP tests for BW = for this line and skips to the next line (RLP + 1) if so. PROP cannot wait for the B => operand at this stage since should the PROP ≠ operand return to PROP from SAC mid-way during the B=> routine, it would be lost due to gating RHO -> RVU instead of SAC -> RVU.

To ensure that Store holds a valid operand before the line to be over-written is lost, PROP copies the operand back to Store if the line is Altered (i.e. has been written to), and sets the line to non-Altered.

If SAC-NK is sent back to PROP from SAC, then no operand will be forthcoming (due to OBS or SAC actions) and no further action occurs in the PROP ≠ routine. However the NSNE Order has served the same function s the INT Order, so that the Interrupt Routine is entered (see Section 3.11.6).

If SAC-OK is received, then the action can be as follows:
1) The NSNE operand returns from SAC (i.e. OBS ≠) to RVU: Using RLP (pointing to the prepared line), the 64 bit operand is written into Name Store. Since this is a direct copy from store, the line is set unaltered (i.e. Used. non-Altered). An outstanding B => operand returning to RHO is ignored until sampled by the next beat as normal.
2) OBS sends 'OBS =' so that PROP knows (1) cannot occur. Thus if an outstanding B => operand beats the NSNE operand from OBS (via Dop and Central Highway) into RHO it can (and must) be stored in Name Store (at the RBW register'home' line). Should the NSNE operand arrive first, the B => operand waits for RHO to be freed and then its presence is sampled as normal. For an NSNE Order caused by a => Variable, OBS has erased the line from its Name Store; PROP stores the operand at RLP, but the line is set to Used. Altered since an ACC => may have altered it.
3) As (2) but not caused by => Variable: OBS has not erased the line; PROP does not insert the line into Store.

For (1) and (2) only: RLP is then advanced to the next line; Stages 2 and 3 are re-cycled to restore RIN and RLR. Should an overlapped B => order be in Stage 4, RBW is strobed to insert its (new) 'home' line.

Juke Box lastly clears the JNSNE bit which allows the order to proceed to Stage 5 on the next beat. The operand to be used is in RVU not RVF as for equivalence and this is forced by Juke Box until after the next beat.

A => Name Store order for which the operand is already in RHI and has been over-written in RVU by the PROP ≠ operand is executed by inhibiting RHI on the next beat.

3.11.4 Other Name Store Modes

(i) In addition to users' programs (identified in PROP Name Store by Process Number), Executive and Interrupt Level 1 routines use the Name Store. To save changing Process No, their Local Names use the space in front of STACK (which users know is liable to change abruptly and therefore cannot use). Common segments are accessed by XNB as for External Names. In general these routines are caused by user errors or are required to implement user's input/output

(ii) The CPR ≠ Interrupt Level 0 Routine arises from hardware limitations only and so is prevented from using the 28 normal lines of Name Store. All NNB accesses go via OBS (RYB = RSN is inhibited). Four lines in PROP Name Store are reserved for Level 0 Interrupt routines to save all accesses going to Store Via OBS. They are addressed modulo 8 by decoding the 3 l.s. 32-bit address digits and are always 'Used' (i.e. are ignored by the PROP ≠ and PURGE actions). They provide eight working registers for the Level 0 routine are accessed by all Local Name orders. AW = cannot occur, all ACC Variables using these registers as for D orders.
(iii) In addition to the Bypass actions by System (Block 0) V-Store and NNB (not SN) orders, the PROP and OBS Name Stores can be bypassed completely by means of a signal from the Console (NSBP). All Local Names then make accesses to Store via SAC as for System V-Store. If CPR ≠ occurs then control is not incremented for that order.
(iv) Two System Errors may be caused by the PROP Name Store. Should Association (Stage 3) yield multiple-equivalence, then a hardware malfunction is indicated and System Error 'PROP Name Store Mult =' is produced.

Once an operand has been copied successfully from Store into OBS or PROP, that CPR should not be changed by the CPR ≠ routine unless it is no longer used (ie a SEARCH in PROP and OBS shows no line in the Block accessed through this CPR). If all CPR's are used, the Name Store's may be PURGE'd. When an altered line is written back to store by PROP ≠ or PURGE, Juke Box does not wait for CPR = (as in => NSBP) but assumes it. The line is written back using a 'Force' bit which even though SAC is in the Run-down condition, will ensure that this operand reaches and updates Local Store providing this CPR is still set. Should CPR ≠ occur information has been lost and System Error 'CPR ≠' is produced. This will be due to hardware malfunctions, either directly in SAC, or following corruption of the CPR ≠ routine.

3/11/5

3.11.5 Instruction Buffer Interrupts

As described in Section 3.3.2, if the IBU is unable to access an instruction due to Control Segment Overflow or CPR ≠ for an access to SAC, it generates marked 'Valid' orders to be read into PROP.

When an IBU Interrupt reaches Stage 4, PROP's action depends on the cause of Interrupt:

(i) CPR ≠ causes a Control Transfer to this order which causes the pipeline and IBU contents to be abandoned. IBU makes a 'priority' request to SAC which must cause CPR ≠. A ≠ on a priority request does send CPR ≠ Interrupt to the Interrupt System. Prop preserves the occurance of IBU CPR ≠ by setting RIBI. This signal is sent to SAC and causes all IBU requests to be treated as priority requests. This is necessary when the ≠ occurs as a result of a long instruction crossing a page boundary. RIBI is reset on entry to the Interrupt.

The Control Transfer is implemented by an 'Increment Control Only' order in Stage 5 which is prevented from adding to Control by 'Lockout' (see next Section). The Control Transfer routine is then entered which sends RCA(I) to IBU. Since the Pipeline contains 'garbage' orders, its contents are discarded.

(ii) Control Segment O/V reaches Stage 4 and causes a PROP Internal Interrupt as for a Control Segment OV produced by a Relative Control Transfer.

Should the contents of Pipeline and IBU be discarded by a Control Transfer or Interrupt System command before the IBU Interrupt attempts to execute, no action is taken since the ''Interrupt'' was merely due to IBU accessing ahead of the orders actually executed.

3.11.6 The Interrupt Order

PROP acts on Interrupts in Stage 4. The PROP Internal Interrupts are listed below:

(i) Illegal Functions	B, D, ACC or f' => Literal or PROP IR. f' => Secondary operand f' => External Name (/XNB and Seg(XNB)≠SN)
(ii) Name ADDER Overflow:	following Name plus Base by Stage 1 to produce the address of a Primary Operand; following an f' PROP IR Manipulation.
(iii) Control Adder Overflow:	following Control plus Operand during a Relative (Unconditional or Test) Transfer; When an IBU Control Segment O/V order reaches Stage 4.

(iv) illegal 'V' Access: any order specifying a Priviledged operand if RMS(13V14V15) = 0.[Program Fault only]

The Interrupt System 'Interrupt MU5' command is sampled at Stage 4. The period of sampling lengthens indefinitely if the present order in Stage 5 is slow to be executed.

The Interrupt (INT) order is produced at Stage 5 if an order which has produced a PROP Internal Interrupt is attempting to execute or if (irrespective of the order which should be at Stage 5) the Interrupt System command has been sampled. The beat which transfers a PROP Internal Interrupt order into Stage 5 also sets (i) - (iv) into Program Fault or System Error registers if produced by a valid User or System instruction respectively (RMS(13+14+15)=0 or 1 respectively).

The INT order is a valid, no-increment Control order; it leaves Stage 5 for OBS via Dr. OBS makes a 'test' request to SAC and then abandons the order. The purpose of this order is to ensure that the LINK is not set to Interrupt Level 0 (as it may be for some Interrupts) until all SAC request have been made which can cause CPR≠ Interrupt. (The IBU is still accessing SAC, but CPR ≠ is dealt with 'privately' by PROP - see Sections 3.3.2 and 3.11.5.) Otherwise a CPR ≠ could cause a System Error erroneously.

Since the INT order propagates behind all the other orders to OBS, when the 'test' request reaches SAC and sends back SAC-OK or SAC-NK to PROP, Juke Box can enter the Interrupt Routine. A further hold-up could be caused if a COMP result is still outstanding (see next Section).

The Interrupt Routine strobes RIE which encodes the m.s. Interrupt Number for IBU and sets the 'Interrupt Entry Bit' (RIE07). This inhibits any further Interrupt Commands from the Interrupt System. It also causes IBU to abandom its contents and to send two fixed Instructions (32 bits each) to PROP:

1) SET LINK to System V-Store [Block 0, Base 0, Name = 2 x Intp No + 0]

2) EXIT to System V-Store[Block 0, Name = 2 x Intpt No + 1]

PROP meanwhile discards the pipeline contents and strobes in dummies until the first Fixed Instruction arrives which saves the OLD LINK in Segment 8192. Having loaded the NEW LINK from Segment 8192, the second order causes an absolute control transfer to the first instruction of the Interrupt Routine (one per Interrupt No.) and resets the 'Interrupt Entry Bit'. Name Store is by-passed by the System V-Store orders (see Section 3.10.3).

The Interrupt System is also locked out by a signal from the Console. Should a PROP Internal Interrupt be caused, PROP will halt at the entry to the Juke Box Interrupt action.

An INT order is not required before the Interrupt Entry bit is set if SAC-NK occurs instead of 'SAC-OK' for the following:

(i) PROP ≠ order (see Section 3.11.3)

(ii) DR-WAIT and Special Descriptor (see Section 3.11.8)

(iii) CPR WAIT; for orders which PROP or OBS cannot allow to 'jamb' (see Section 3.11.9)

The function of the INT order has already been accomplished and so Juke Box can enter the Interrupt action directly. For CPR ≠ during =/=> System V-Store or NSBP, the INT order must be generated.

By means of Machine Status bits RMS 14, 15, a process based interrupt routine working in level 1 (RMS 14 = 1) cannot be interrupted by any other L1 interrupt, but can still be interrupted by a LO interrupt; a System based interrupt routine working in level 0 (RMS 15 = 1) cannot be interrupted by any other LO or L1 interrupt, excepting a System Error interrupt.

Any System Error routine works in level 0 and so cannot be interrupted by other LO or L1 interrupts; it cannot be interrupted by later System Error interrupts other than PROP internal interrupts due to a pulse Generator and flip-flop (RIE08) being placed between the 'any System Error Interrupt' level and the 'Interrupt Command' OR gates. RIE08 is reset by Interrupt Entry and cannot be set again until all System Error Interrupt levels have been reset by the routine. This leads to the following conditions:

(i) The System Error routine must not exit until it has checked that no further System Error interrupt levels remain set. If this is not so, then any remaining & any further System Error interrupts will remain masked and so will not cause Interrupt Entry if the System Error routine exits.

(vi) = PROP IRQ, SET LINK, STACK LINK, MS=, EXIT, RETURN all load or store MS. The contents of the Test Registers may not be valid following the last COMPare order unless at least one of (iii) - (v) has since been executed.

Due to race hazards in PROP, it is not possible to reset JBLO and JCLO without first causing a 'Lockout' for (i) or (ii), or a hold-up for (iii) - (v).

3.11.8 Interactions with Dr

Each order which sends a Descriptor from Stage 5 to Dr has to wait for one of the following conditions:

(i) Normal action: 'order complete' and may be executed by PROP

(ii) Special Descriptor: Descriptor Types 3.0-2 must not 'jamb' in OBS. A CPR-WAIT is executed from Dr by sending no signals to PROP, but causing OBS to make a 'test' request (c.f. next Section). Juke Box exits DR-WAIT on SAC-OK. SAC-NK causes entry to Interrupt action.

(iii) DA $\neq$. If Dr finds the Secondaty Operand makes a request/or requests to the current Name Segment in PROP, then OBS must check in PROP after accessing its own Name Store (not Operand Store) should it find OBS $\neq$. OBS sends a 'check in PROP' request to SAC which causes it to send the Virtual Address down to PROP. PROP then checks in Name Store for Association; if so the operand is copied to Store if altered (c.f. AV =) and the line set to non-USED. When the check is complete and the operand is not (any longer) in PROP, OBS ACCESSES Store for the valid operand.

This can be repeated as many times as necessary. When Dr has sent the last order to OBS, it sends 'order complete for Segment =' . PROP executes the order when the next (ie last) check is complete. Each time OBS finds OBS =, a signal is sent to PROP to enable exit when waiting for the last check, since no 'check in PROP' order will be generated by OBS.

No DA = action starts until any outstanding B => operand has returned and been stored. This ensures the Name Store operands are valid.

The first Virtual Address for PROP from SAC sets up DA = (Control Transfer) which causes a Control Transfer to the next required instruction on exit. The pipeline is abandoned since Orders in Stage 2 - 5 may suddenly find themselves without entries in the Name Store. Any NO-WAITs are first actioned (ie an operand returned following the DA = is used as normal).

A Special Descriptor may also cause a DA = in which case there is a single check in PROP which occurs before the 'test' request; Dr sends no signal; PROP exits on SAC-OK/NK.

(iv) Procedure Call (Type 3.4)

When Dr recognises this descriptor, a 'Procedure Call' signal is sent to PROP. Juke Box sets the 'Procedure Call Bit' which causes IBU to send two Fixed Instructions (c.f. Interrupt Entry):

STACKLINK

JUMP, D[0] (Jumps to location given by Dr)

The Procedure instructions exit by setting RMS00('D Set Bit') in the stored LINK and then obeying an EXIT. The order causing the Procedure Call will be re-obeyed as a D = D order by PROP (due to RMS00), the descriptor now in Dr pointing to the value of the evaluated expression. RMS00 is reset by PROP following the re-execution of the order.

(v) Abandon STS

PROP sends Dr the Interrupt command signal from the Interrupt System. This allows PROP to respond to Interrupts during a long STS (eg String -> String) order by Dr curtailing it at a convenient point. PROP then sets Stage 5 to dummy to leave Control pointing to this order. When the order is re-executed following Interrupt Entry and the Interrupt routine, the order carries on from where it left off, Dr having left the Descriptor pointing to the next element to be processed. Should a STS order produce multiple DA =, the last check in PROP is completed before abandoning the order.

3.11.9 Interactions with OBS

While OBS is fetching or passing through operands, orders follow one of two paths from Dr to Dop:

(i) ACC Function Queue can contain up to 6 or 8 functions yet to be executed by ACC. OBS has made requests to SAC for any operands that are neither supplied (by PROP or DR) nor are in OBS Store. Should SAC be in Run-down Mode (or enter it due to a request finding CPR ≠)for any of these, no operand will be returned and the Queue will 'jamb' or Run-down with the first operand-less function at the Head of the Queue.
If a request from PROP or a Priority (ie Control Transfer) request from IBU has caused the Run-down Mode in SAC, then OBS Queue need not necessarily jamb, but depends on a racing condition.

The OBS Queue can be un-jambed by causing OBS to re-access SAC after the CPR ≠ routine has finished by the RESTART V-Store Action. Alternatively the Queue may be copied to Store by DUMP and Run-down Queue for another Process copied from Store by UNDUMP and this one RESTARTED if a Process change is necessary.

(ii) The Bypass Function Register can contain one function for a non-ACC order ie an order from PROP which does not load or store the c.r.'s or IR's in the ACC Unit. The operand is held in the OBS Name, Secondary or Literal [operand supplied] Store as for (i)

An order in the Bypass Register is not allowed out until the ACC Queue is either empty or Rundown. The former ensures valid operands in OBS Store, the latter is required for software action in the CPR ≠ routine (see below).

Orders going from PROP to OBS may not be executable when SAC is in (or enters for that order) Rundown Mode for one or more of the following reasons:

(i) Following a CPR ≠ Interrupt and Interrupt Entry (which resets Rundown Mode in SAC), the CPR ≠ Routine must access all Names using /XNB via OBS Bypass Register (see Section 3.11.4). For this reason, no order using the Bypass Reg [non-ACC order from PROP] must be left in the Bypass Register unless it is guaranteed an operand (ie OBS = or CPR =). Bypass Orders are allowed through despite a jambed queue for this purpose: if the Block required is in Local Store, a CPR can be changed and the Queue RESTARTed without changing process or DUMP/UNDUMP to the same Queue. Level 0 Routines (of which this is one) must get CPR = (else a System Error is caused and their operands are not used by other routines. Thus the jambed Queue cannot cause any invalid operands (cf (ii) and (iii) below).

(ii) An operand is required by PROP from OBS Store or from ACC. If OBS needs to access SAC, then no operand can be supplied. If no access is required (operand in OBS Store or in ACC Unit) then the operand may still not be returned to PROP due to the OBS Queue jambing. A Bypass Order (operand in OBS) must not be allowed to return an operand to PROP since if the Queue jambs such that a jambed order could have written to the operand requested by the Bypass order, PROP would receive an invalid operand.

(iii) A Lockout would be set in PROP (see Section 3.11.7 should PROP execute [add to Control] for this order, but for the same reasons as (i) an ACC COMP order may jamb or a B COMP [ ] order may either jamb or use an invalid operand. Thus either PROP would halt, or the Test Registers be set with a doubtful COMPare result.

(iv) A Special Descriptor (see Section 3.11.8) has produced an order which is not re-executable from the ACC Queue due to special conditions in OBS, eg Real Address access would, when RESTARTed, cause a normal Virtual Address access.

PROP and OBS guard against those possibilities by means of the CPR-WAIT in PROP, and by PROP-, D-CHECK in OBS.

PROP waits for CPR $=$ on all orders sent to OBS which:

(i) Will use the Bypass Register in OBS

or (ii) require an Operand from OBS (via Dop) or ACC to be returned to PROP from Central Highway

or (iii) would cause a 'Lockout' to be set in PROP.

PROP also sets the 'PROP CHECK' bit which causes OBS to also wait for CPR $=$ and to send a 'test' request to SAC. If the operand is in OBS (OBS $=$ or operand supplied) or not required by OBS (ACC => to PROP orders) this causes SAC to send SAC-OK, SAC-NK to both PROP and OBS. If the operand is required by OBS but is not in OBS (OBS$\neq$), a named request with 'test' action is sent to SAC.

If SAC-OK, then OBS inserts the order into the Queue or Bypass as normal; PROP continues its normal action (e.g. enters NO-WAIT if an operand is expected).

If SAC-NK, then OBS discards the order; PROP enters Interrupt action in Juke Box which also sets Stage 5 to dummy, i.e. the order can be re-executed from PROP after the CPR $\neq$ software routine.

For a Special Descriptor, PROP is forced to do a CPR-WAIT by DR (see Section 3.11.8) which sends no signal. If no signal from DR, then PROP waits for SAC-OK or -NK before continuing as above.

As described in Sections 3.11.3 & 6, PROP $\neq$ and Interrupt actions by PROP also produce orders which result in SAC-OK, -NK to PROP. This is caused by setting CPR-WAIT for these orders. PROP $\neq$ needs to know whether an operand is to return from OBS or SAC or not. This is a true CPR-WAIT action and the NSNE order is either put in the Bypass Register (OBS $=$ SAC-OK) or abandoned (OBS $\neq$ or SAC-NK). PROP then either prepares to insert the operand (SAC-OK) or enters Interrupt action (SAC-NK). The INT order is used to empty MU5 of SAC requests before Interrupt Entry. OBS always abandons it and PROP enters Interrupt Action (SAC-OK or -NK).

3.12. PROP V-Lines and Actions (see also MU5 Basic Programming Manual)

These are lines addressed by orders using the V-Store Operand in Block 1 of the Local V-Store. The address from RIN drives the V-Decode in Stage 2 which provides inputs to the V-line register RVD in Stage 3. RVD is only strobed by alid V-Store order.

Hex	RVD	= access	=> access	Name
00	00	= PF	Reset PF	Program Fault Register
01	01	= SE	Reset SE	System Error Register
02	02	= PN	=> PN	Process Number
03	03	= IC	=> IC	Instruction Counter
08	04	= Zero	=>PA & SEARCH *	Purge Address
09	05	= Zero	=>PM	Purge Mask
10	08	= LCU	Reset LP->Zero *	Line Copy (ms half)
11	09	= LCL	Reset LU & LA *	Line Copy (ls half)
18	10	= AP & * LP->LP+1	Purge *	Proc No. in line LP
19	11	= AX	No Action	Line Add in line LP
1A	12	= PW Decode	=> DL	Display Lamps
1B	13	= SI	=> SI	Software Interrupt

Outputs from the V-Decode are used by Stage 2 to distinguish between Block 0, 1 and 2-127 V-Store Actions required in Juke Box. Section 3.10.3 describes the read/write actions taken by PROP.

All read accesses use RVD to gate the V-line through GINTOO-15. The VU-WAIT Action by Juke Box for PROP V-lines puts RLP -> RLR and reads (LR). This is only used by = AP and = AX for which either the process number (4bits) or line Virtual Address (15bits) is read out and then strobed into RVU. Lastly Stages 2 & 3 of the Name Store are re-cycled to restore RLR and RIN.

All write accesses produce a strobe JVW which can strobe or reset the required register (if any).

Accesses marked by * have ancillary actions described below:

(i) Reset LP->ZERO.

The line pointer register RLP always contains one & only one bit. This order sets RLP00='1' & the rest = '0'. It points to the next line to be used by PROP≠ action or by (ii) below.

(ii) = AP & LP->LP+1

Having put RLP->RLR, LP is advanced by $RLR_j \rightarrow |RLP_{j+1}|28$. By using = AX & = AP etc alternately, the Name Store Virtual Address Field contents may be read out sequentially. By obeying Reset LP -> ZERO first, the read-out will start at line 0. By also reading Line Copy, the bit(s) of LR (if any) set on Association by the last valid Name Store access may be read out to test the operation of the Associative Field.

(iii) Reset LU & LA

The registers containing the Vectors of lines which are Used (RLU) or Altered (RLA) are reset to all zero's by this access. This scraps the contents of the Name Store. The Local Store is not updated for altered lines.

(iv) => PA & SEARCH

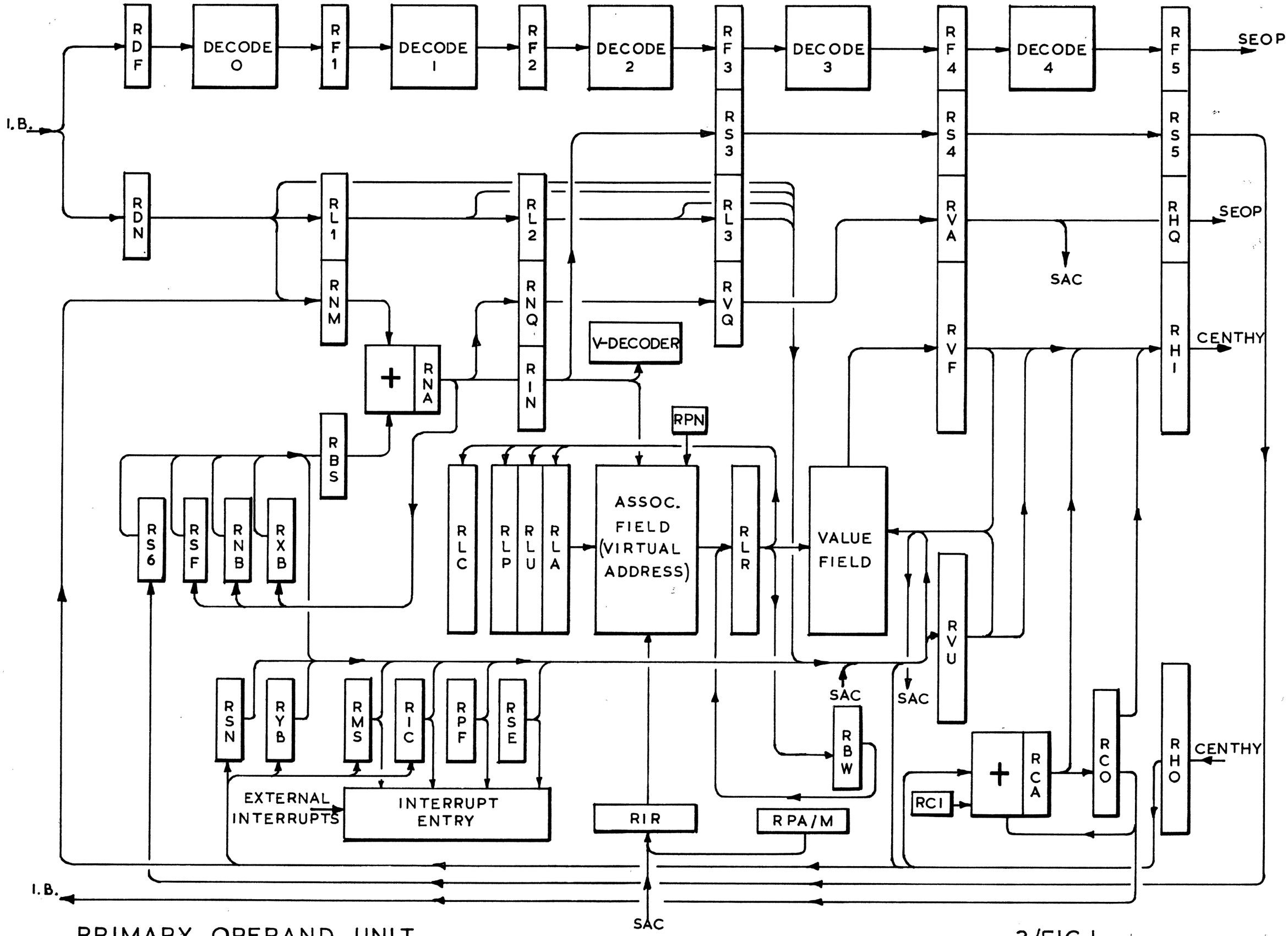
When a CPR ≠ has occurred, the CLR≠ routine will normally have to de-allocate a CPR. Before doing so the PROP & OBS Name Stores must be SEARCHed to find out if any CPR's are unused. Normally a => PM is obeyed first, PURGE then causes a Masked Association to be carried out by Juke Box on the Virtual Address in RPA down to the Block size defined be RPM. The l.s. three bits are always masked by PROP.

If the CPR is used by PROP there will be (multiple) equivalence in RLR. This causes the Test Register ' ≠ 0' (RSM05) to be set. Non-equivalence in PROP causes no change in RMS05 since searching the OBS Name Store puts RMS05 = Association Found. Thus if RMS05 = 0 after Searching OBS & then PROP, the CPR is used by neither Name Store and may be altered.

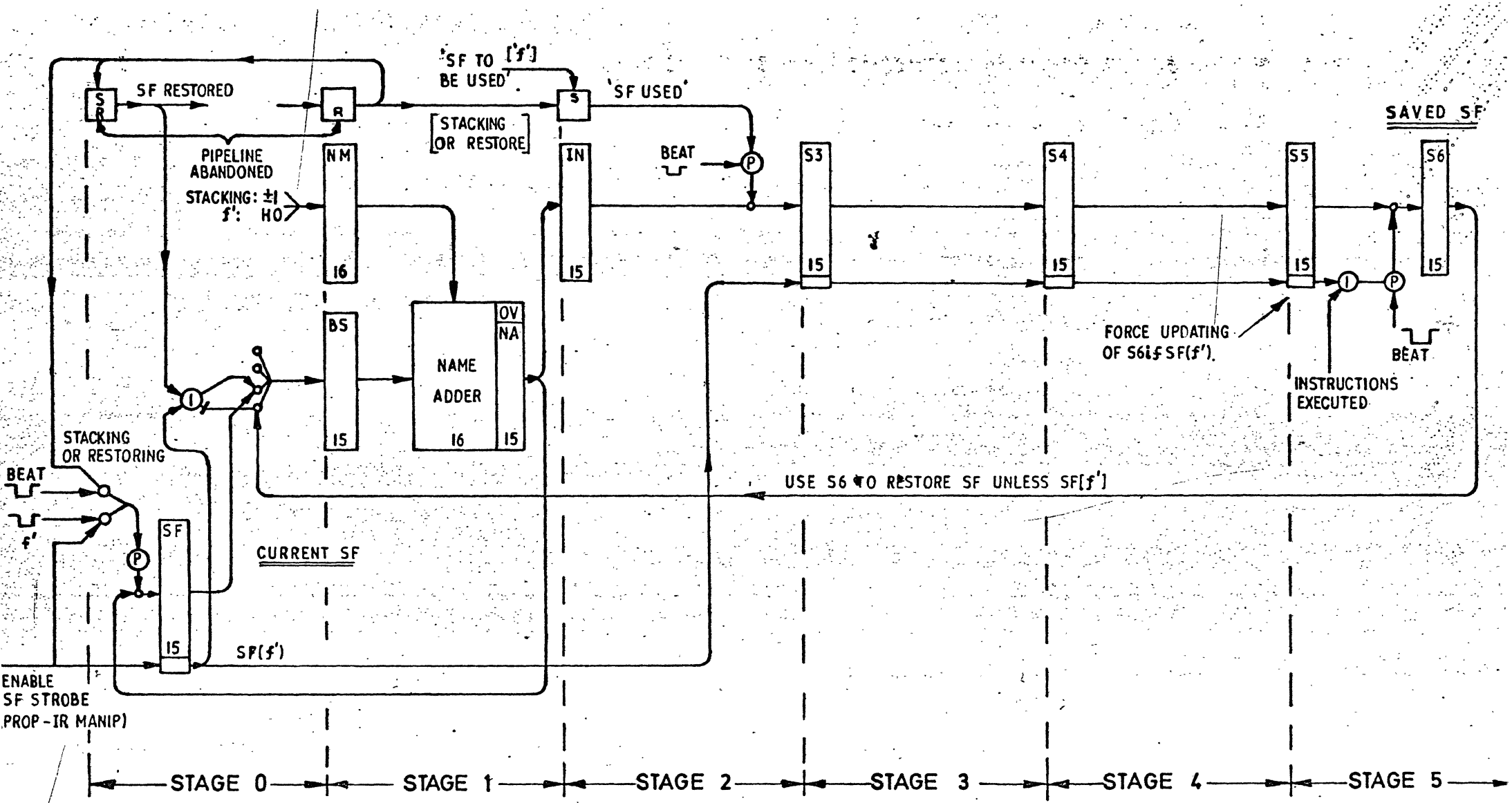
(v) Purge

If repeated use of (iv) above shows that all CPR's not required by System routines are used, then PROP & OBS must be PURGEd.

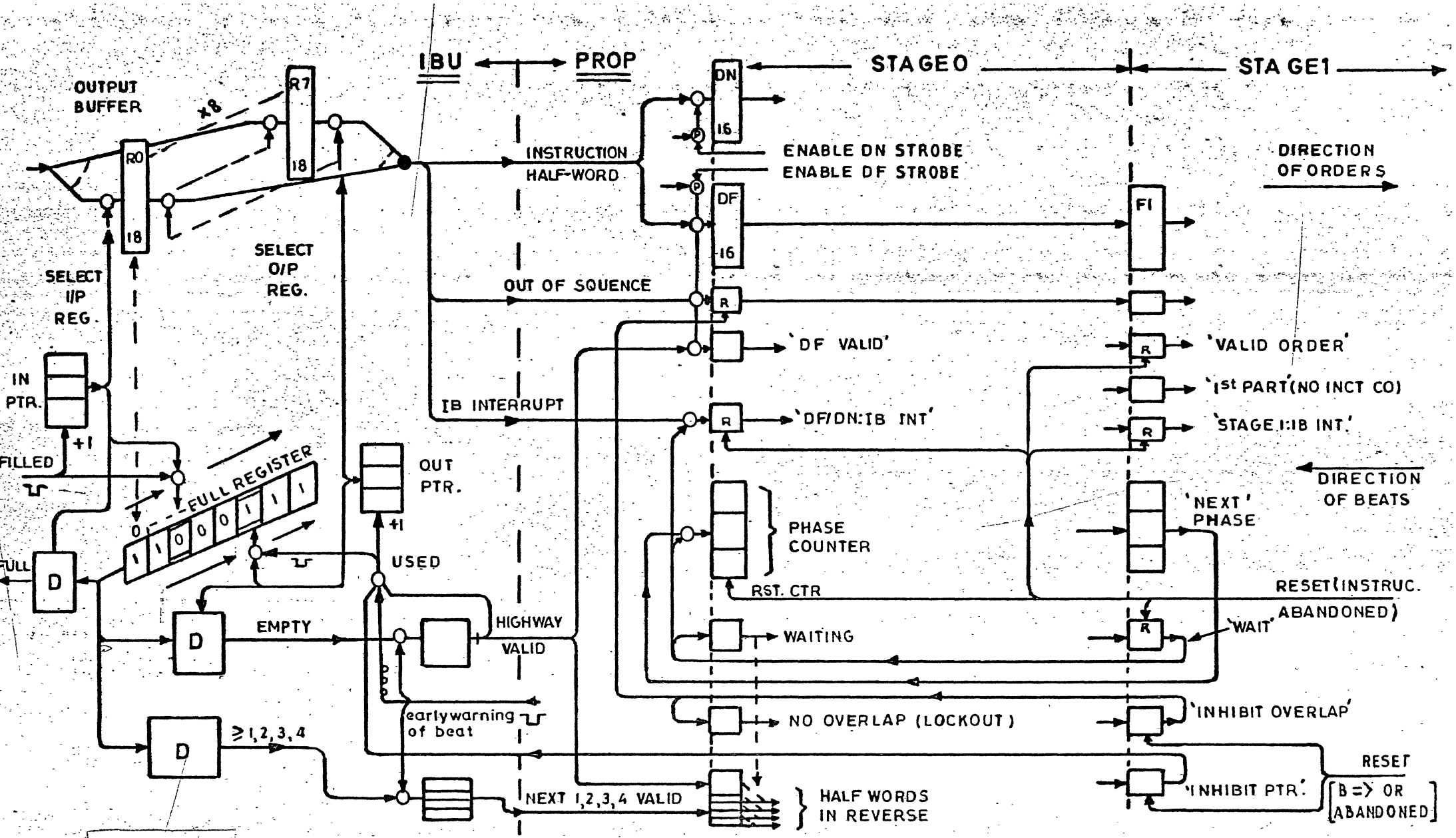
PURGEing PROP Name Store sets LP -> ZERO after which Juke Box examines each line in turn until line LP = 27 is reached. Each line is copied back to Store via SAC if Altered. All lines are lastly set to non-Used.non-Altered to leave the Name Store empty and Store updated. There is a similar action in OBS.



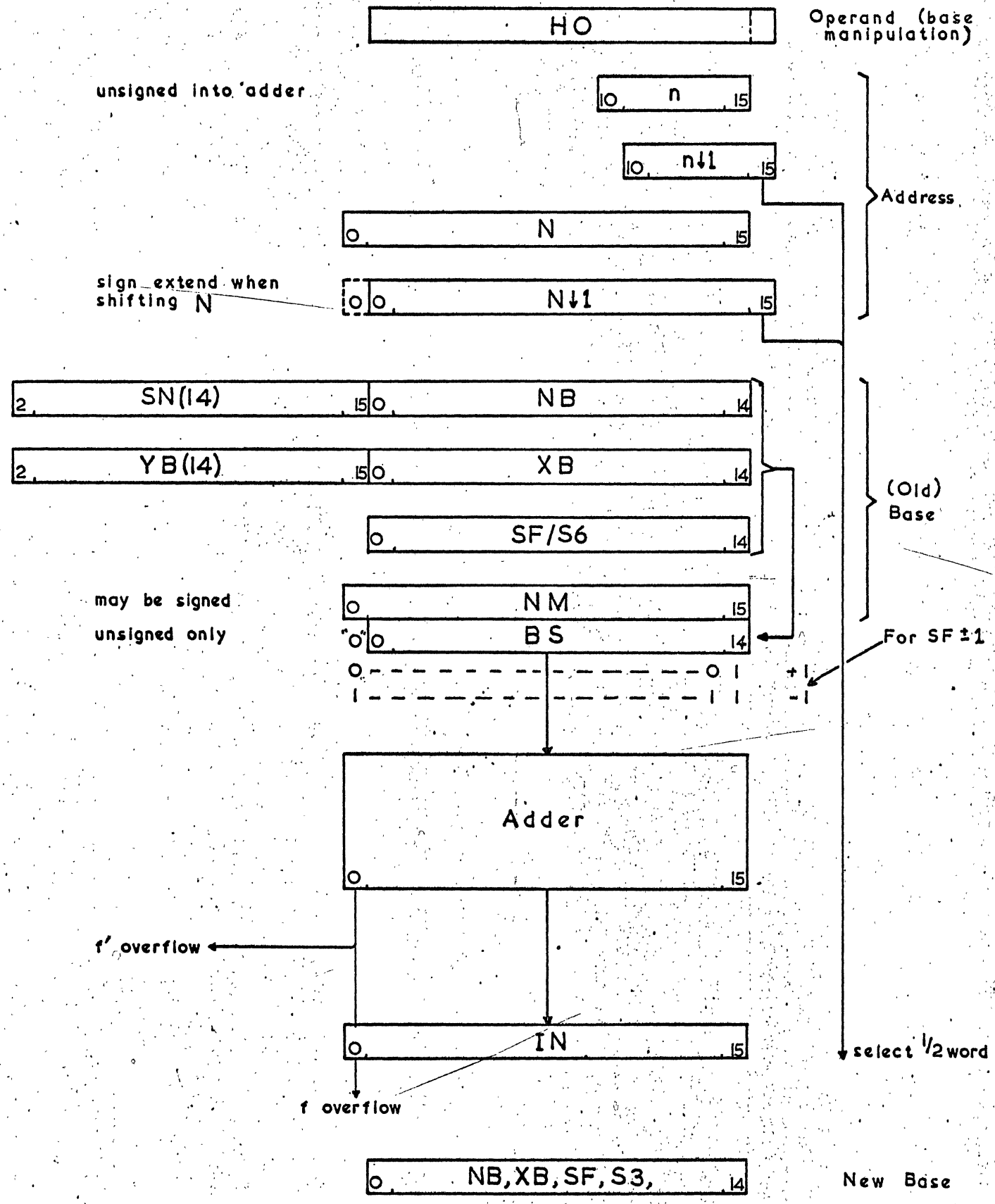
PRIMARY OPERAND UNIT



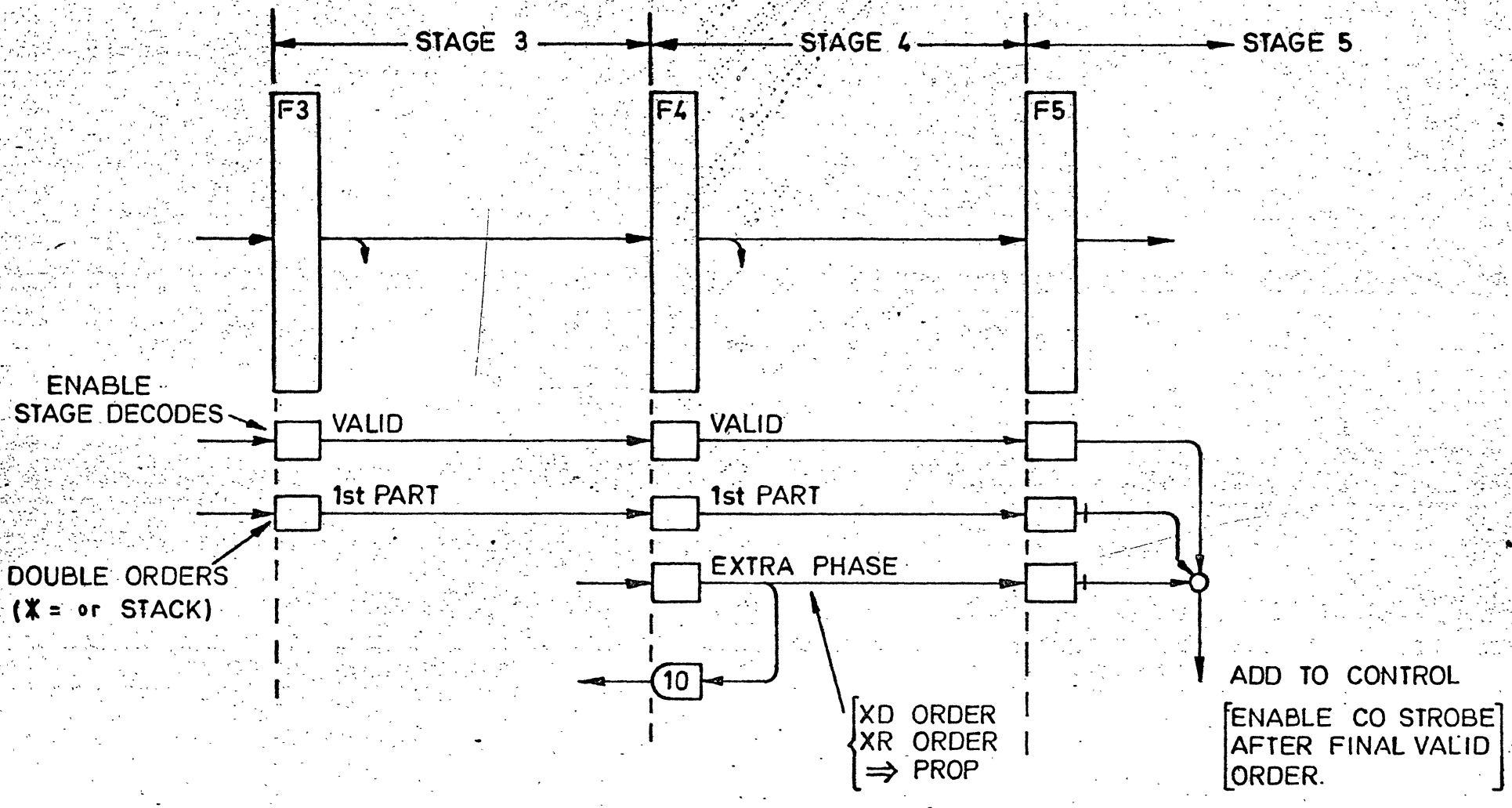
3/Fig. 2 STACK PIPELINE



3/Fig. 3 IBU/PROP INTERFACE AND MICROPROGRAM GENERATOR-1



NAME ADDER INPUTS — "SIGNIFICANCE" OF REGISTERS



3/Fig. 5

MICROPROGRAM GENERATOR – 2

Prop	Highway
F31SL1, ST1/4/5	C36SB3/4/5, SR6
	Operand (64)
F33SB4/5, SR5/6	C39 SB1-4
Operand(64)	
F26SB5	C39SR5
Function digits 3-6, 12-15	
Function digit 18	
Function digit 19	
Internal Register B	
Modifier Request	
B long order	
F23SB1	C39SB5
Prop to B	
Prop to Prop	
Prop Complete	
Initial Reset	
	Highway Complete
	B Store Complete
	Highway Ready
	B loaded by Dop

3/Fig 6(a) Prop - Highway Interface

F21ST2

Segment Number (14)
Process Number (4)
Exec/L1/LO

F1/ST1

32 bit word Address (16)

F33SR1-4

Descriptor / Operand (64)

F26SB6

Function Digits 0-15
Function Digit 18
Function Digit 19
32 bit Variable
OBS Delete

F23SL4

Start
Prop to DN
Prop to Dr
Interrupt Dr
Interrupt Order
Valid Name Access
Initial Reset
Internal Register D
Internal Register A
D [ ] request
Modified request

CPR Check

F38SR4

D interrupts

F38SR3

F36SL1-4

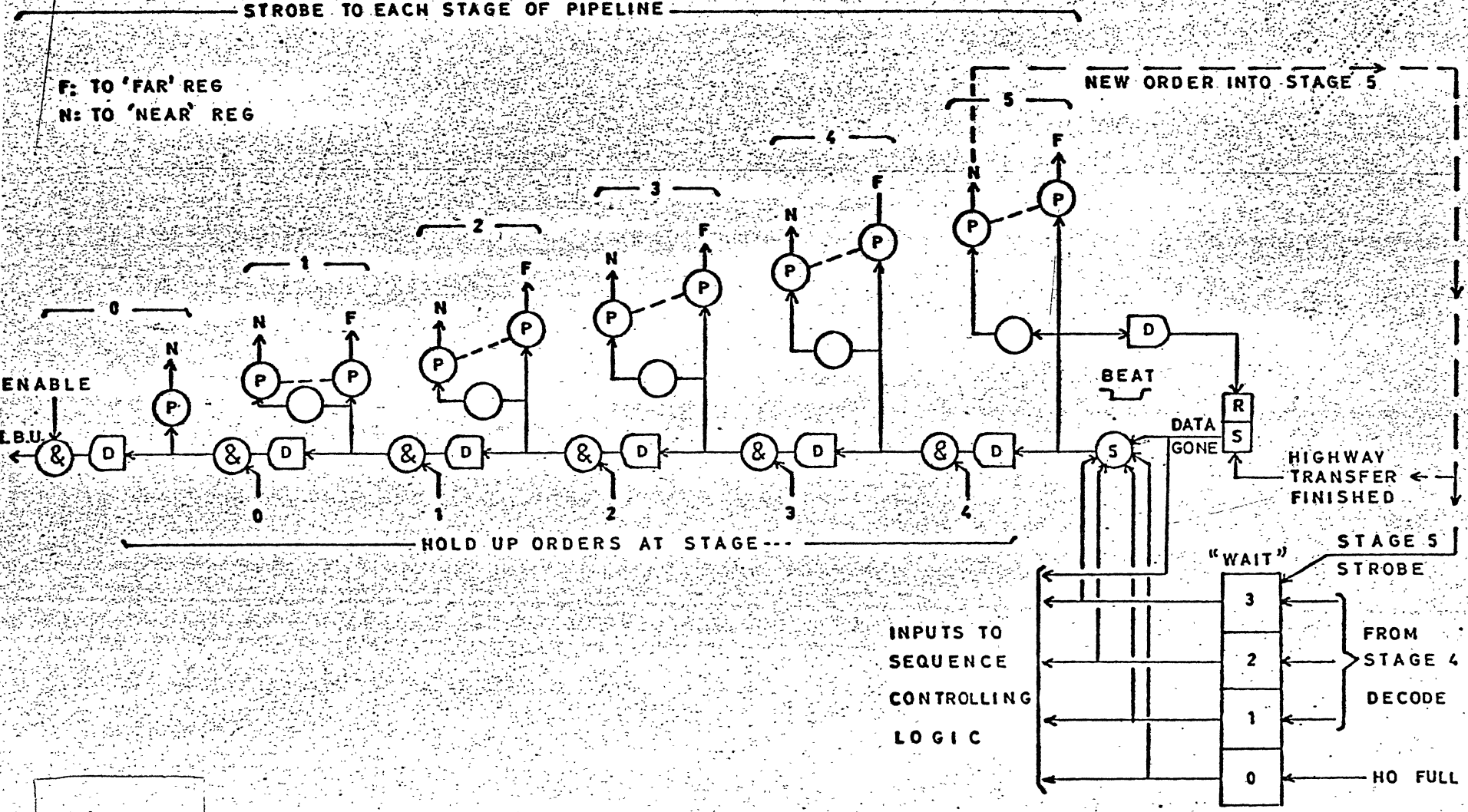
F30SB5

F36SR5

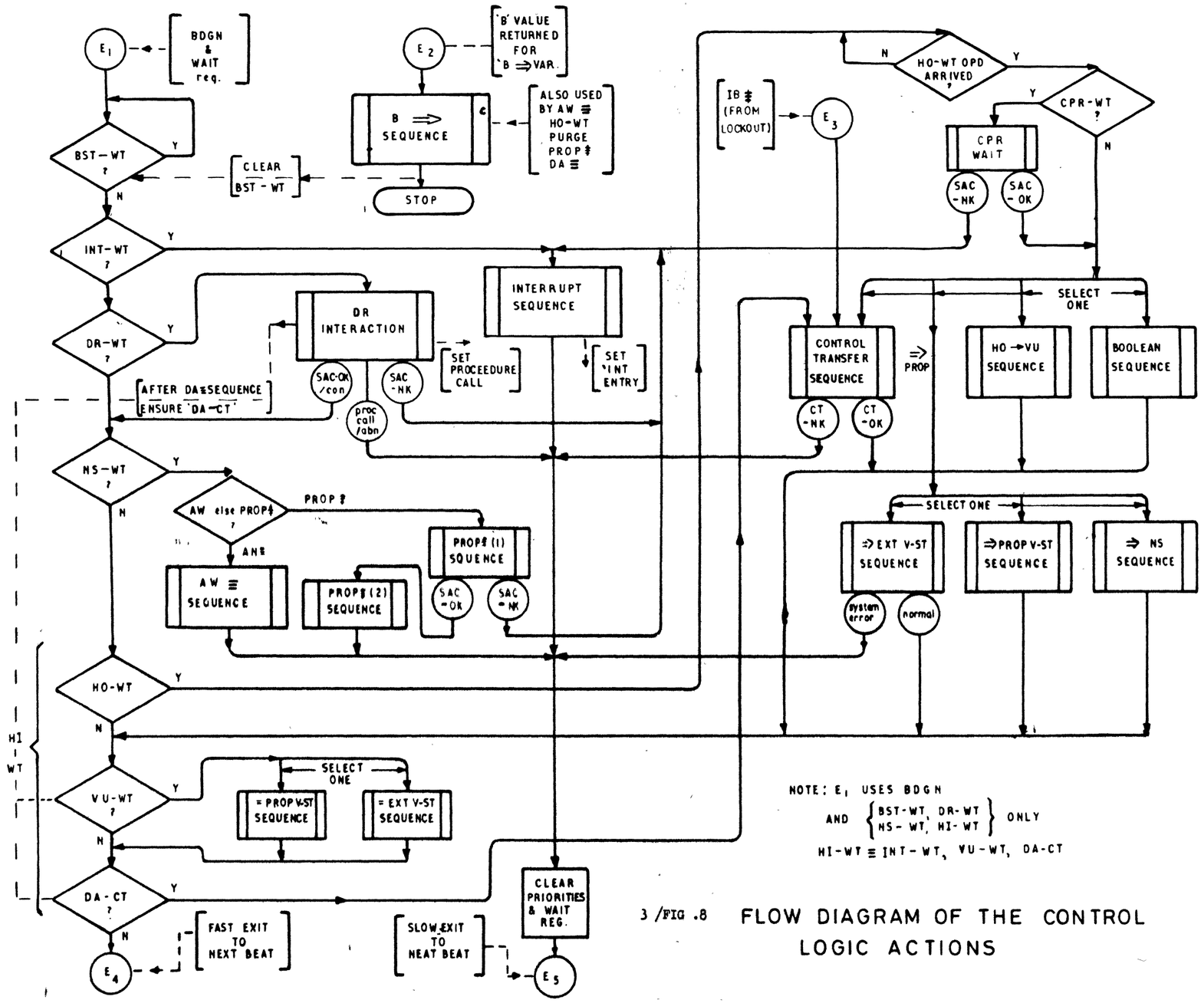
Dr Complete (Slow)
Name Segment =
Procedure Call
Dr Complete (Fast)

Increment Inst. Counter

3/Fig 6(b) Prop - Dr Interface



3/Fig. 7 PIPELINE DELAY CHAIN



3 /FIG .8 FLOW DIAGRAM OF THE CONTROL LOGIC ACTIONS

Prop

F32SL3, SB1/3

Operand (60)

F31SL5/6, 32SL5/6

Operand (4)

F24ST2

Segment Number (14)
Process Number (4)
Interrupt Request
Exec/L1/L0

F11 SR6

32 bit word Address (16)

F11SR5

Process Number (4)

F23SL1

Read Request
Write Request
Force Bit
V Request
32 bit Request
IBU Requests -> Priority
CPR By-Pass

SAC

F46 SB3/5, 45SR6

F44 ST1/2/4, SL5

Operand (64)
Data Available

F43SL4

F43SL6

64 bit word Address (3)

F43SR5

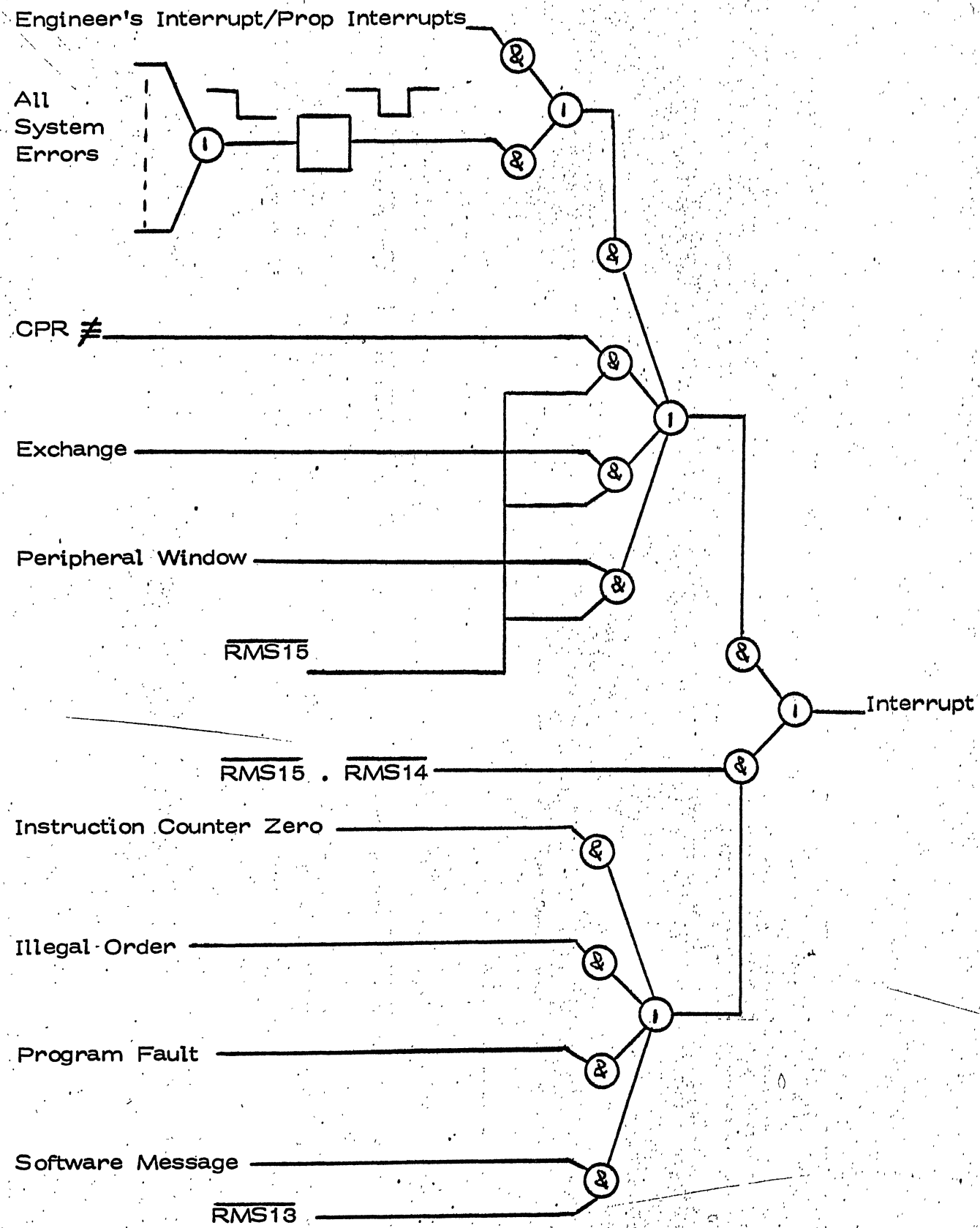
64 bit word Address (12)

F43SB1

Name Segment = Request

SAC OK
SAC NK
Request Accepted
Data Available

3/Fig 9 Prop - SAC Interface



3/Fig. 10 - Interrupt Structure

Chapter 4 Central Highway

4.1 Introduction

4.1.1 Highway function

The Central Highway (or CENTHY) provides some of the non-dedicated data paths within the MU5 CPU. The CPU is shown in block diagram form on Fig. 4/1, with the routes provided by CENTHY indicated by bold lines. These are:

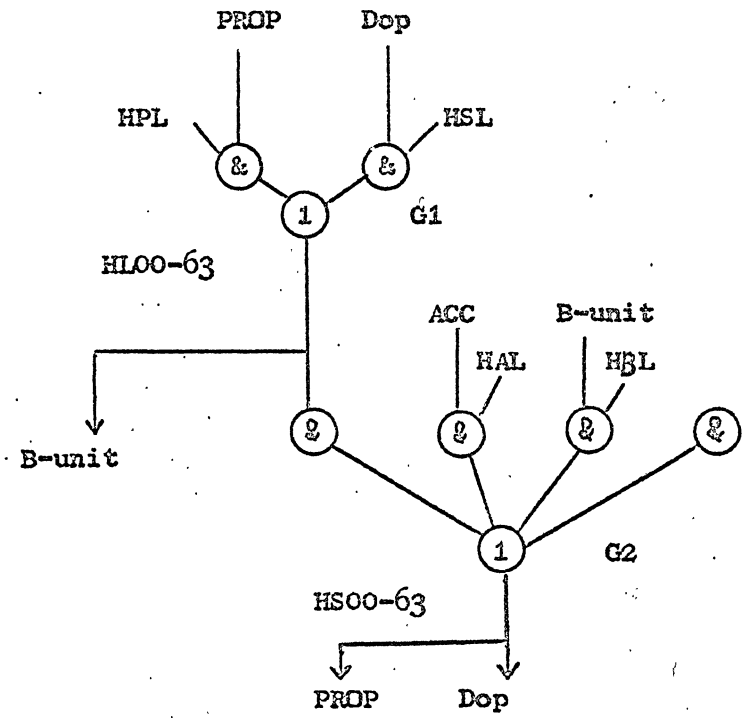
1. A route from PROP or SEOP into the B-unit: provided for B load-type orders.
2. A route from the B-unit or ACC into PROP or SEOP: provided for the transfer of ACC or B store order operands.
3. A route from PROP or SEOP into PROP : provided for various PROP orders.

The chapter provides a generalised description of the CENTHY action for the various types of orders sent through it, with particular reference to the interface requirements.

4.1.2 Data path

Basically the CENTHY consists of two sets of 64 gates, providing a 64-bit wide data highway, plus the control for selection of the right gate (path) at the right time. These two sets of gates are termed G1 and G2, situated on platters C39 and C36 respectively. Note that although the path is 64 bits wide, in the case of the B-unit 16 of these bits are not required (bits 16 - 31)

The data path is as shown below :



4.1.3 Highway interfaces

The highway interfaces with PROP, the B-unit, SEOP and ACC. These interfaces are shown in Figs. 4/2 - 5 inclusive, together with a list of data, function and control signals.

4.2 B load-type orders

4.2.1 General

A B order is specified when the three cr bits of the function are decoded as 001. A B load-type order is specified when cr = 001 and the four f bits are other than 0011 (=>).

These orders can be either of two types; direct orders which require a primary operand or indirect orders requiring a secondary Operand.

4.2.2 Direct orders (Figs. 4/2 and 3).

All primary operands are received from PROP. The table below shows the control signals and data passing through the CENTHY interfaces between PROP and the B-unit, for a normal direct load-type order.

PROP -->	<-- CENTHY -->	<-- B
		BRDYS B-unit ready
PHØØ-63 data PFØ3-Ø6 function PBH PROP-> B order	HLØ0/-15,32-63 data HFØ3 -HFØ6 function HPRPB 'PROP' order HCP data transfer complete	HRDYB input strobe

Initially PROP sends the pulse PBH to the CENTHY, at the time when the PROP highway input register RHI is about to be strobed with an operand which must go to the B-unit. The CENTHY control checks that:

(a) the B-unit input buffer is available to receive the operand (BRDYH), and

(b) the data path from PROP to B is free (HPL).

If these conditions are satisfied CENTHY sends the strobe HRDYB to B. This strobe loads the operand and function into the B input buffers. When the input buffers are about to become free, the B-unit again sends the pulse BRDYH to CENTHY.

4.2.3 Indirect orders (Figs. 4/2, 3 and 4).

In this case the B-unit receives its operand from the Dop section of SEOP. The specification and function are sent directly to the Dr section of SEOP from PROP. The Dr process of operand address assembly then commences, and access is made to the store for the operand. When CPR equivalence is obtained and the modifier request has been sent, the B lock-out flip-flop in PROP is set. This inhibits further direct orders being sent to B from PROP out of sequence.

The following table shows the interface signals and data for an indirect load-type B order.

SEOP ->	<- CENTHY ->	<- B-unit
SBH SEOP ---> B transfer	HRDYB input strobe	BRDYH B-unit ready
SHØØ-SH63 DATA	HL ØØ-HL15 HL 32-HL 63 data	
SFØ3-SFØ function	HCS transfer complete	HFØ3 - HFØ function HPRPB SEOP order
PROP		
	HLOMP reset B lock-out	

Initially SEOP sends the pulse SBH to CENTHY, and the general procedure is then similar to that for a direct order, but in this case the operand comes from SEOP. A pulse, HLOMP, is sent to PROP to reset the B lock-out. When the data transfer is complete HCS is sent to SEOP.

4.3 B store orders

4.3.1 General

B store (=>) orders are specified by cr = 001 and f = 0011. The order is routed to the B-unit in the same manner as for a direct (PROP) or indirect (SEOP) load-type order, although the operand for a store order is not significant. B store orders require the content of the 32-bit register RB to be set onto the 1.s. end of the highway and routed to either PROP or SEOP.

The function bits are accompanied to the B-unit by HPRPB, which is at logic 1 for a direct order or at logic Ø for an indirect order.

4.3.2 B -> PROP store order (Figs. 4/2 and 3)

The table below shows the interface signals required for a B -> PROP store order.

PROP ->	<- CENTHY ->		<- B-unit
PRDYH PROP ready to take operand	HSØØ-HS63 data		BPH data is for PROP
	HRDYP input strobe		BHØØ-15, BH32-63 data
	5HBLF operand from B	HTRCB data transfer	data available

When the B-unit is ready to perform the store order, the pulse BH is sent to CENTHY. A signal, BPH, which is a copy of HPRPB sent with the order, is also sent. BPH controls the routing of the operand when the operand is received from B.

The CENTHY checks that the PROP highway output register RHO is free, and then issues a pulse HRDYP, to strobe the operand into RHO. In order that PROP will be able to determine the origin of the operand, CENTHY sends the pulse 5HBLF. When the data transfer is complete CENTHY sends HTRCB to the B-unit.

4.3.3 B -> SEOP store orders (Figs. 4/3 and 4)

In this case the function is accompanied to B by HPRPB = ø. It is not necessary for CENTHY to check the SEOP input buffer as SEOP will be ready to receive the store order operand. The interface signals are as shown below.

SEOP ->	<- CENTHY ->		<- B-unit
	HRDYS input strobe HSøø - HS63 data	HTRCB data transfer is complete	BH data available BPH = ø data is for SEOP BHøø - BH63 data

In this case the operand is routed to SEOP as BPH=ø (HPRPB). When the data is available the B-unit sends BH to CENTHY, which sends the input strobe HRDYS to SEOP. When the data transfer is complete HTRCB is sent to the B-unit.

4.4 ACC orders

4.4.1 General

ACC load-type orders utilise a route through a special buffer in the SEOP Dr system, and so are not considered in this section on the CENTHY.

ACC store orders are of two types; those where the operand is destined for PROP and the more usual case where the operand goes to SEOP.

4.4.2 ACC PROP store order (Figs 4/2 and 5)

It should be noted that the operand is not sent to PROP for all ACC direct store orders, but only when the operand is to be written to a V-line or Internal Register or when Name Store By Pass Mode is set on the Console. The following table shows the interface signals for an ACC -> PROP store order.

PROP ->	<- CENTHY ->	<- ACC
PRDYH PROP ready to take operand	HS00-HS63 data HRDYP input strobe HTRCA transfer complete	AH data available AH00-AH63 data

ACC sends the signal AH to CENTHY to initiate the transfer of the operand to PROP, similar to the procedure used for a B -> PROP store order. PRDYH and HRDYP are active as before, and when the data transfer is complete HTRCA is sent to ACC.

4.4.3 ACC SEOP Store Order (Figs 6/4 and 5)

In this case the SEOP initiates the transfer of the operand, the interface signals being as below.

SEOP ->	<- CENTHY ->	<- ACC
ACC SEOP order	HRDYS input strobe HS00-HS63 data HTRCA transfer complete	AH00-AH63 data

The pulse SAH, from SEOP, initiates the transfer, and CENTHY sends the input strobe HRDYS to SEOP. When the transfer is complete HTRCA is sent to the ACC.

4.5 Modifier Request

The modifiers for SEOP data descriptors are held in the B register. There are two cases in which a modifier is sent from the B-unit to Dr, and these are discussed below. In each case the modifier request is overlapped in the same manner as normal orders.

4.5.1 Modifier for indirect order.

When an indirect order is sent from PROP to SEOP, PROP determines whether modification is required. If this is the case then PROP initiates a B-order by sending PBH to the CENTHY (see 4.2.2). In addition to the function bits the signal HMODB is sent. The effect of this is for the B-unit to ignore the function received from PROP and to send the modifier in RB to Dr.

4.5.2 Store to store order modifier

It is possible that Dr will require a modifier whilst executing a long store to store order. In this case SEOP initiates the modifier transfer as for an indirect load-type order (see 4.2.3). Accompanying the function bits to B is the signal SMRB, which caused the modifier to be routed to Dr after completion of the order specified by the function bits.

4.6 Internal Register Orders

All Internal Register (IR) orders are interpreted as double orders. Thus:

(i) A load-type order is converted to (a) an Internal Register read order and (b) a load-type order

(ii) A store order is converted to (a) a store order and (b) an Internal register write order.

The transfer of the operand between the two phases is effected via the PROP.

4.6.1 Read from IR

To read one of the B-unit IRs, the procedure is similar to that for a direct B store order. The function is sent to B as described in 4.2.2, together with the Internal Register address (PF12-PF15), and accompanied by HIRB. This signal causes the B-unit to initiate a store order if the function bits indicate load-type. Instead of the content of RB being gated onto the highway, the content of the IR specified by PF12-PF15 is sent.

4.6.2 Write to IR

Writing to an IR in the B-unit involves similar action to that for a direct B load-type order (see 4.2.2), except that PF12-PF15 and HIRB are sent as above. The order may be overlapped in the normal manner as the complete operand can be buffered in the B-unit.

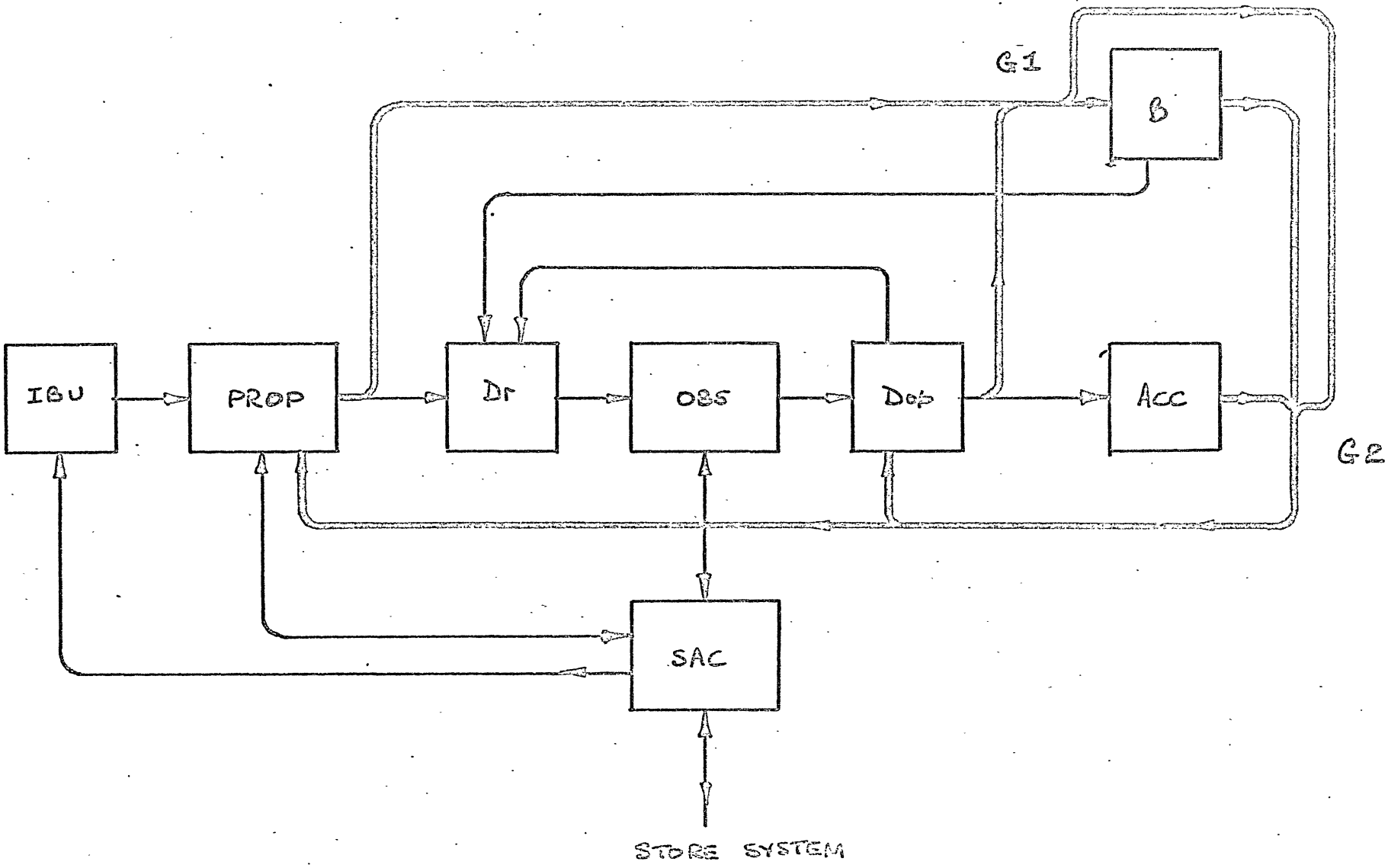
4.7 Other Orders

The route from Dop to PROP shown on Fig.1 is used for the following:

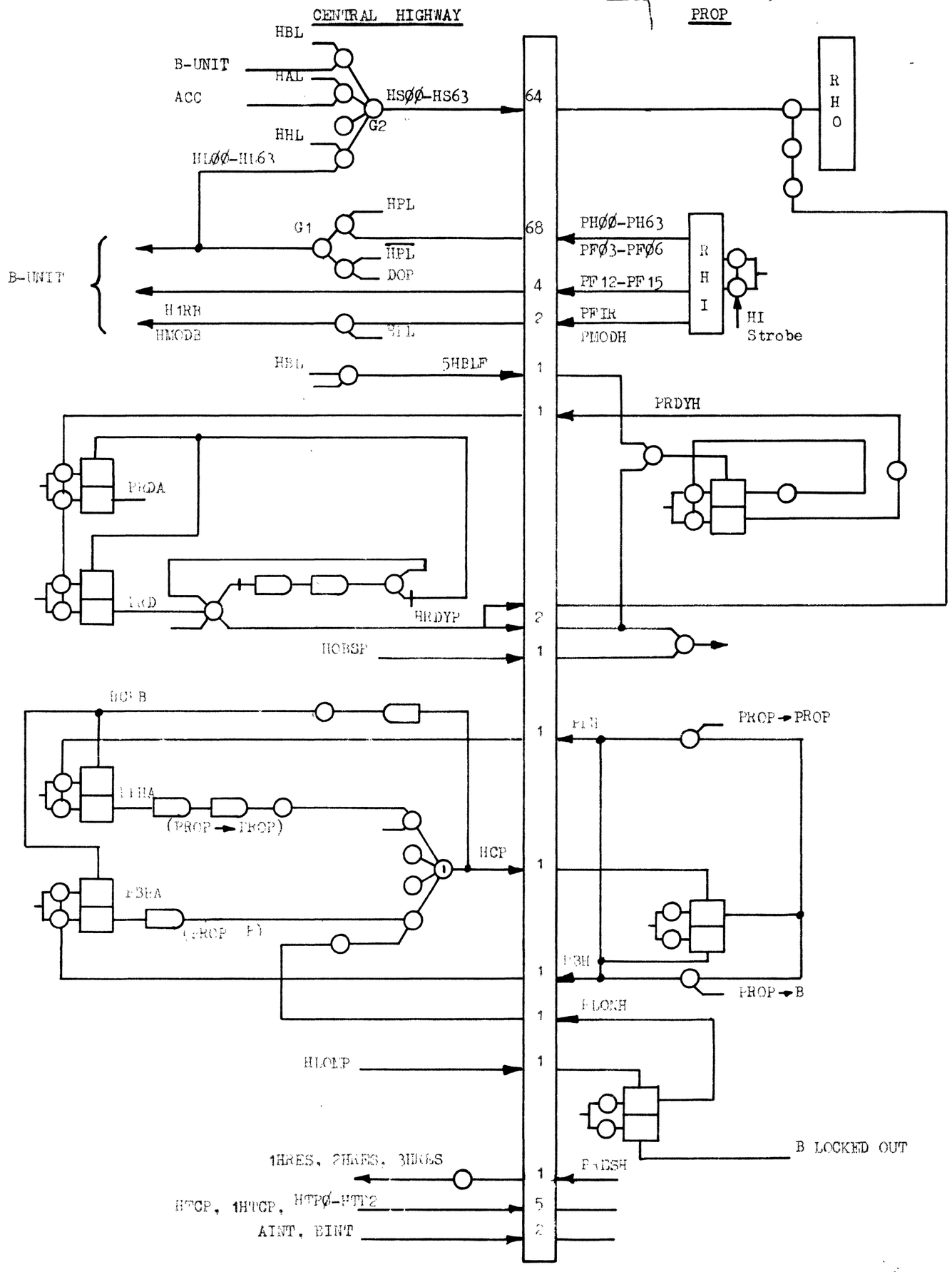
1. To send the contents of the Internal Registers in SEOP to PROP
2. To load registers in PROP with an indirect operand.
3. To send operands from OBS to the Name Store. In this case CENTHY sends a signal HOBSP to PROP, indicating that the operand has originated from the OBS system.

4.8 Test Bits

Various orders executed in B, ACC and SEOP (e.g. COMPARE) are required to set the Test Bits in PROP (RMS 04, 05, 06). CENTHY receives four signals from each unit, 3 result and 1 ''result available'' signal. The respective signals from each unit are OR'ed together and sent to PROP. Interrupt conditions in B or ACC are also routed to PROP by CENTHY.



CPU – BLOCK DIAGRAM



CONTROL OF PROP AS A RECEIVING UNIT

CONTROL OF PROP AS A SENDING UNIT

NOTE: SIGNAL LIST ON FOLLOWING SHEET

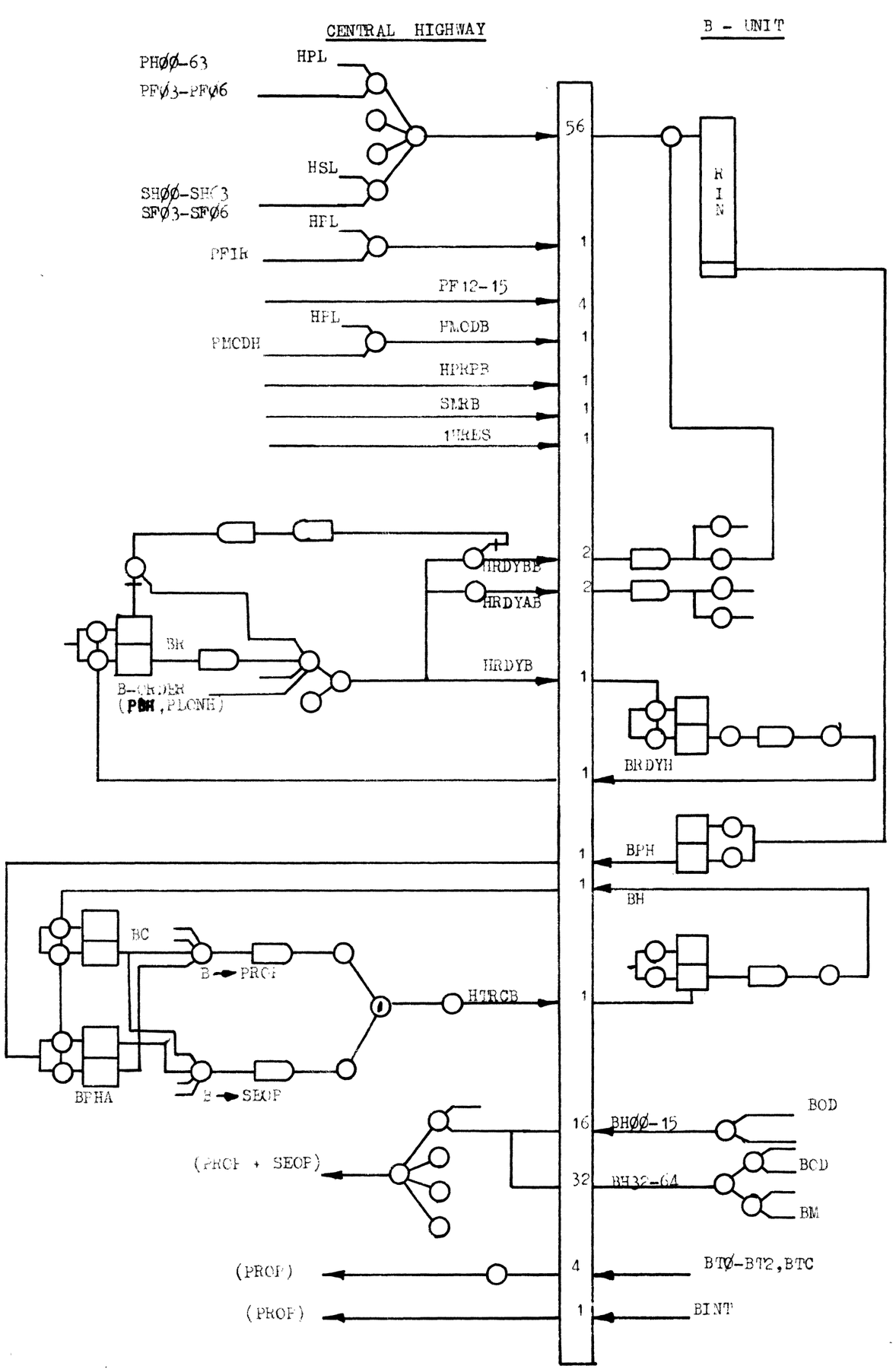
INTERFACE: CENTHY-PROP

4/ FIG. 2a

INTERFACE SIGNAL LIST		CH → PROC	
CENTRAL HIGHWAY →		← PROC	
HSØØ - HS63 data	64	64	PHØØ - PH63 data
SHBIF data is from B-unit	1	4	PFØ3 - PFØ6 function
HRDYP. 1HRDYP input strobe	2	4	PF12 - PF15 IR bits
HCBSP operand from CBS	1	1	PFIR IR phase
HCP data transfer from PROP is complete	1	1	PMODH modifier request
HLOLP reset B-unit lockout	1	1	PRDYP PROC ready to receive an operand
HPPØ - PPP2 test bits	3	1	PSH HSCF to PROC order
HPCP. 1HPCP test bits strobe	2	1	PSH HSCI to S order
ALPP ACC interrupt	1	1	PLONH B-unit lockout not set
BINT B interrupt	1	1	PRESH initial reset
	77	77	

INTERFACE : CENTHY–PROP

4/ FIG.2b

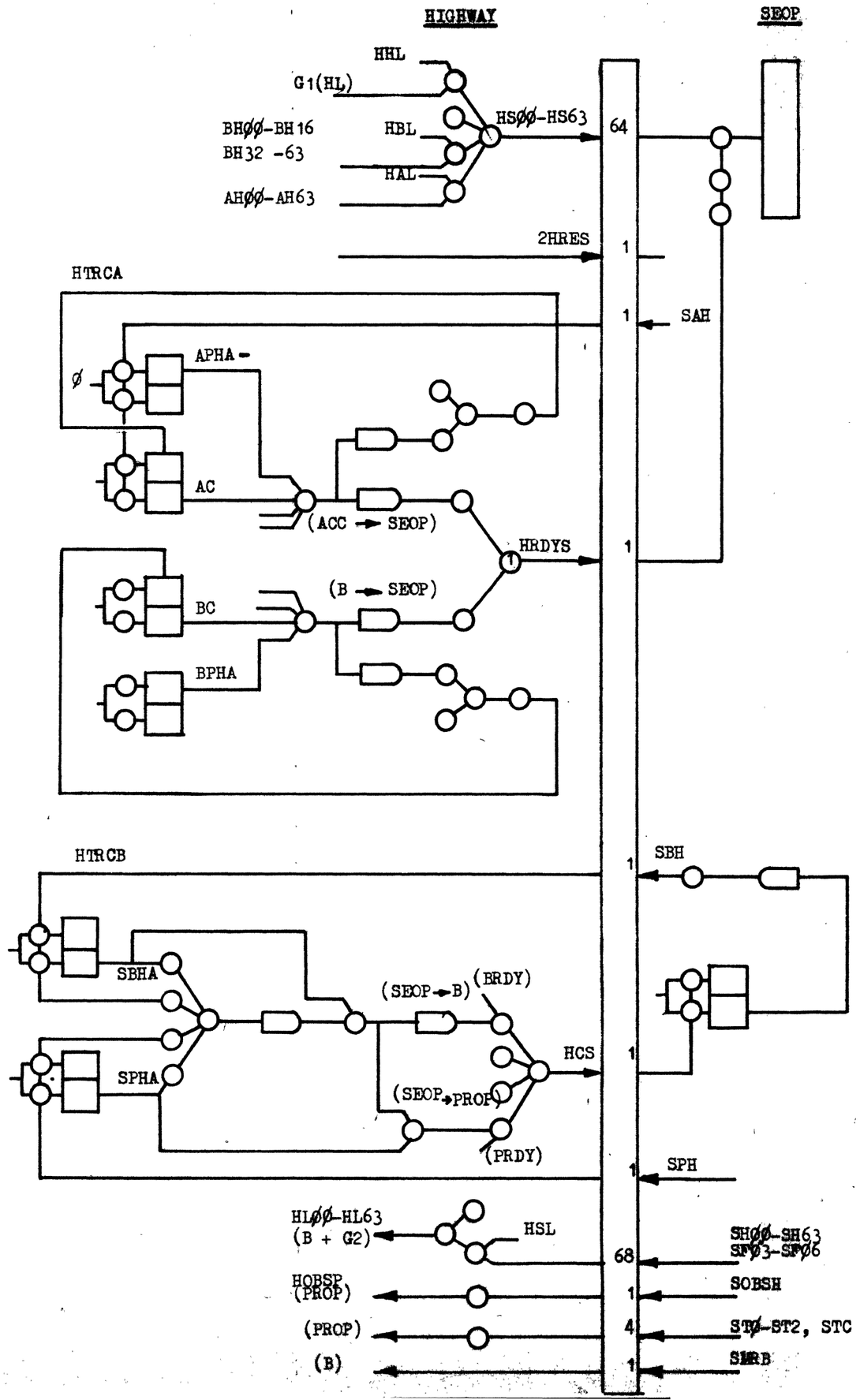


INTERFACE SIGNAL LIST CH → B

CENTRAL HIGHWAY →		← B-UNIT	
HLØØ-HL15	16	16	BHØØ-BH15
HL32-HL63	32	32	BH32-BH63
data			data
HFØ3-HFØ6, 1HFØ3-1HFØ6	8	1	BH
function			data available
PF12-PF15	4	1	BrH
internal register address			data to be sent to PROP
HIRB	1	1	BRDYH
internal register order			B ready for next order
HRDYB	1	3	BTØ-BT2
HRDYAB, 1HRDYAB	2		test bits
HRDYBB, 1HRDYBB	2		
input strobe		1	BTC
HMCDB	1		test bits strobe
modifier request		1	BINT
HPRPB	1		interrupt
'primary operand' order			
HIRCB	1		
transfer of data from			
B-unit complete			
1 HRES	1		
initial reset			
SMRB	1		
IMXT modifier request			
	71	56	

INTERFACE : CENTHY – B-UNIT

4/ FIG.3



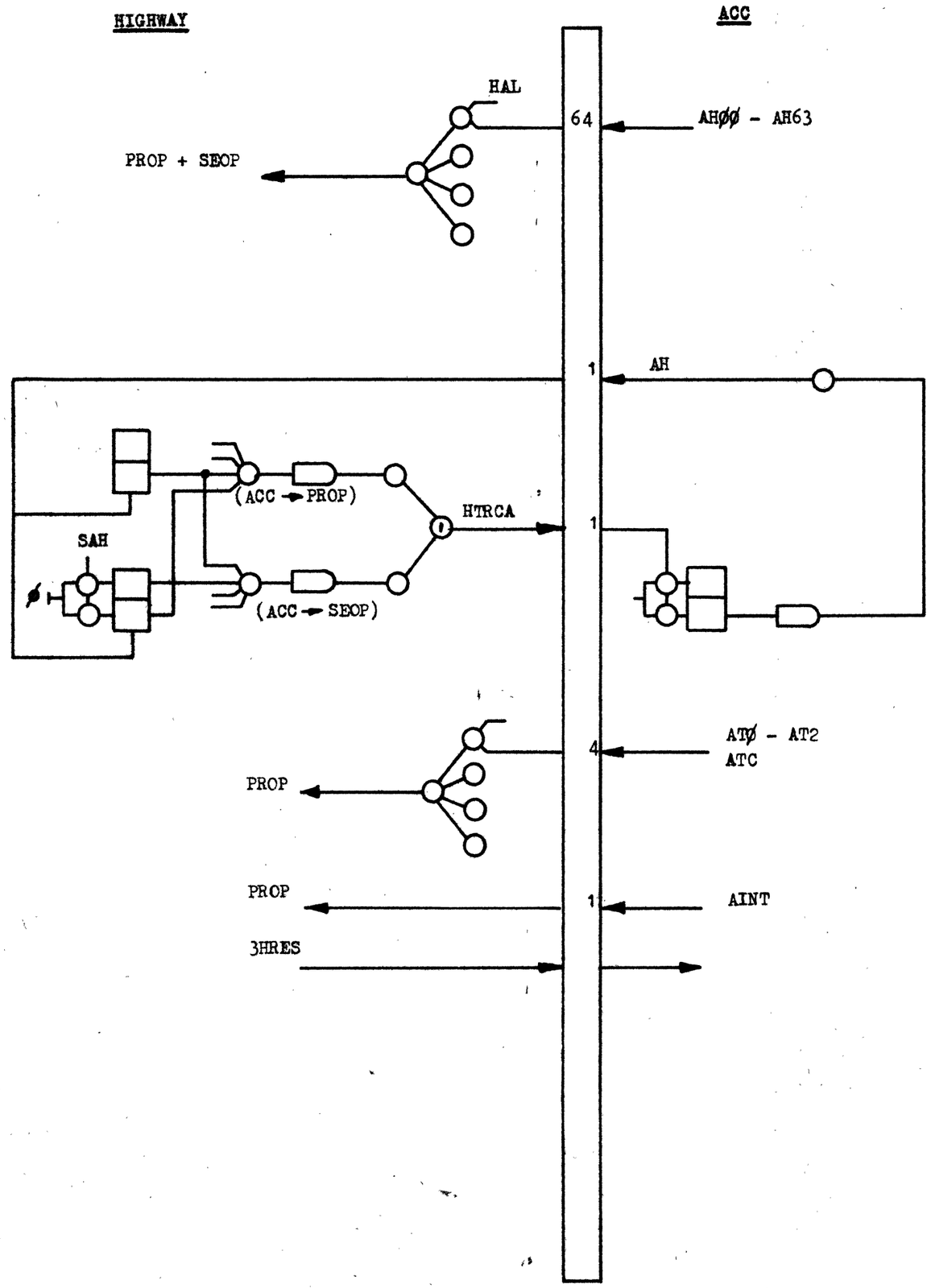
INTERFACE SIGNAL LIST

CH → SEOP

CENTRAL HIGHWAY →		← SEOP	
1HSØØ - 1HS63 data	64	64	SHØØ - SH63 data
HRDYS input strobe	1	4	SFØ3 - SFØ6 function
HCS Transfer of data from SEOP complete	1	1	SAH Acc → SEOP transfer
		1	SBH SEOP → B-unit transfer
2HRES initial reset	1	1	SPH SEOP → PROP transfer
		3	STØ - ST2 test bits
		1	STC test bits strobe
		1	SOBSH operand from OBS
		1	SMRB modifier request
	67		77

INTERFACE : CENTHY - SEOP

4/ FIG.4



INTERFACE SIGNAL LIST : CH → ACC

CENTRAL HIGHWAY →		← ACC	
HTCRA Transfer of data from ACC is complete	1	64	AHØØ - AH63 data
		1	AH data available
		3	ATØ - 2 test bits
		1	ATC test bits strobe
3HRES Initial reset	1	1	AINT Interrupt
	2	70	

INTERFACE : CENTHY–ACC

4/ FIG.5

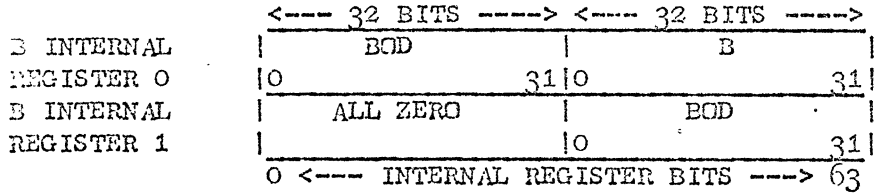
Chapter 5 B-32 Arithmetic Unit

5.1 Introduction

The 32 bit B-Arithmetic Unit performs load, store, logical, compare and all signed arithmetic funtions except divide and reverse divide which are treated as dummy orders. Load and decrement by 1, compare and increment by 1, and shift functions are also available.

5.2 Functional Description

The B-unit receives instructions and operands from the central highway. Operands are taken from the least significant 32 bits of the 64 bit wide highway. The unit contains two of the machine internal registers as shown below. When writing to or accessing these registers all 64 bits of the highway are significant. When reading an internal register, the overflow flag (BOD 0) is reset to logical 0. When requested, the B-unit sends modifiers to SEOP with an associated control signal, or information to PROP with a control signal.



FORMAT OF B INTERNAL REGISTERS

5.3 Overall Diagram (5/Fig.1)

The unit consists of an input operand buffer RDIN 0-63 and a function input buffer RFIA 0-3. For reasons of fanout limitation this register is duplicated as RFIB 0-3. The internal register decode bits are strobed into buffer register IRIN 0-3. If the B-unit is empty, or when the current instruction reaches the end of its execute phase, the contents of RDIN are gated to both operand registers RD, RDA 0-31 and RS 0-31 (bit 32 of RS is set to zero). At the same time the decode function bits are used to set one of the sixteen bistables in register RFD 0-15 (the one associated with the instruction to be executed). Each of registers RD, RS has its own slave register (RD*, RS* respectively), forming two independent shift loops RD-RD* and RS-RS*. The output of RD (shifted or inverted as necessary) can be presented to the inputs of both functional units. Because of the large amount of fanout needed to do this, register RD is duplicated by register RDA, which is similar in all respects, and is strobed at the same time as RD. The arithmetic unit is a 32 bit full adder with an output register RA 0-31. The logic unit is a 32 bit gated register which (by selecting various phases of input and output signals) can be made to perform logical AND, OR, NOT EQUIVALENCE functions. Its output register is RL 0-31. In all cases the other input to the arithmetic and logical units is gated (inverted if required) from the B register RB 0-31. The value strobed into RB on completion of the dataflow cycle can be selected from either phase of the logic units input, the adder output, or the output of the slave register RD*.

5.4 Mode of Operation

With the B unit inactive, BIDLE and BEMT are true. Initially BRDY is formed, and when HRDYB is received, XIN is made to strobe input data from the highway into registers RFIA, RFIB, RDIN, IRIN and certain control bistables. On receipt of HRDYB, BIDLE and BEMT are reset to logical zero. This completes the first phase, i.e., buffering of the instruction. Since the control bistables show that no function is in execution, the instruction may now enter the second phase (execution of the instruction). The individual function bistables (RFD) are strobed, and one of the outputs will become a logical 1. This will initiate one branch of a timing control chain, and the instruction proceeds as described in section 5.6. On entering the execute phase BRDY will be formed in readiness for the next instruction to enter the buffer registers. If HRDYB is not received within the time necessary to ensure its entering the execute phase immediately following completion of the previous instruction, BIDLE is formed. If the previous instruction is complete and the B unit is inactive before the next HRDYB is received, i.e., no descriptor requests etc., are outstanding, then the signal BEMT is formed. Since most of the B order code consists of single beat instructions, the unit may work continuously buffering instructions, entering the execution phase while requesting the next instruction, accepting and buffering the next instruction, and so on, at a maximum rate limited by the delay time around the loop from RB to Adder to RB.

5.5 Definitions of Interface Signals (FIG. 2)

(a) Input Signals

HLB00-15,32-63 Data input to RDIN from central highway. Bits 16-31 of the interface are always zero, and in order to minimise the number of connectors required, they have been omitted. The same remarks apply to outputs BHIN00-15,32-63.

HRDYB This signal marks the arrival on the input from the highway of the next instruction, which was requested by the previous BRDY pulse. The HRDYB pulse is used to form XIN, which strobes data into the buffer and control bistables. HRDYAB, HRDYBB are the same signals as HRDYB, but HRDYB is sent one level earlier, to simplify the control logic.

BTRC The data placed on the output lines from the B unit to the highway has been accepted, and the data may now be changed, allowing the next instruction to proceed.

HPRPB In the case of a store order originating in the PROP this bit is logical 1 (logical 0 for SEOP). This signal is gated to BPH while the data to be stored is present on the output lines to the central highway.

HF3-6 The four function bits decoded to give one of sixteen functions.

HF12-15 The four internal register bits, decoded to give one of sixteen internal registers. Only registers 0 and 1 are used.

HMODB If this bit is a logical 1, at the end of the instruction associated with it the value of B is sent to SEOP, and the output signal BD is formed.

IRO An instruction associated with a 1 in this position is treated as a 'LOAD', if it is a store instruction, or as a 'STORE' instruction for any other case. The data transferred is the content of internal register 0 or 1 as defined by the HF 12-15 decode (any other decode of the four bits is treated as 0 or 1, defined by the least significant bit).

(b) Output Signals

BHI00-15,32-63 Data output to central highway. Used for store operations and internal register transfers. Bits 16-31 are omitted, see HLB00-15,32-63.

BH This signals that data is present on the output to the highway, and stays true until BTRC is received.

BRDY This pulse to the central highway signals that the B unit is ready to receive another instruction into its buffer registers.

BPH A copy of HPRPB , used during transfers to the highway.

BDIN32-63 Data to SEOP (permanently wired from RB).

BD Data is ready on BDIN lines. After this 20 nS pulse the data may be removed.

BIDLE The B unit buffer is empty, i.e., no HRDYB has been received within 25nS of BRDY being formed.

BEMT Signals B unit inactive, i.e., no function being buffered, no function in execution, and no Internal Register requests, etc., outstanding.

BFAULT Arithmetic overflow has occurred, setting BOD0, and the mask bit (BOD4) was not set. BFAULT = BOD0, $\overline{\text{BOD4}}$.

BTO-2 Test lines giving the result of compare operations.

BTS Strobe for BTO-2.

5.6 Function Code

LOAD (=)

The operand (RD) is loaded to RB via the adder RA.

LOAD and DECREMENT (='')

The operand (RD) is sent to the adder RA where it is decremented by adding -1. The result is loaded into RB. If overflow has occurred BOD 5 is set. If INHIBIT (BODO) is not set a program fault interrupt, B FAULT, is produced.

STACK and LOAD (*=)

This order is seen in the B unit as two distinct orders from PROP. Firstly a STORE order (see below) is given, in which the content of RB is fed onto the least significant 32 bits of the central highway. Finally the operand (RD) is loaded to RB via the adder RA, as in a LOAD order.

STORE (=>) The content of RB is fed onto the least significant 32 bits of the central highway and zeros placed on the most significant 32 bits.

ADD (+)

The operand (RD) is added to the content of RB in the adder RA and the result loaded into RB. If overflow has occurred BOD5 is set and B FAULT may be produced (see above).

SUBTRACT (-)

The operand (RD) is gated to the adder where it is subtracted from the content of RB and the result loaded into RB. The algorithm for subtracting RD from RB uses the fact that RB-RD $= \overline{\overline{RB} + RD}$.

REVERSE SUBTRACT (Θ)

This is identical to the subtract order except that the content of RB is subtracted from the operand (RD).

MULTIPLY (*)

To decide which operand is to be made the multiplicand a test is carried out on the sign of the operand (RD) and the content of RB. If they have the same sign a subtract function is performed, forming the result in RA. If they are of differing signs an add function is performed, (RB being unchanged). If bit 0 of RA = bit 0 of RB then $|RB| \geq |RD|$, the content of RB is gated to RD as the multiplicand and BOD 12 is set to a 1. Otherwise, the content of RB is gated to RS as the multiplier. RB is then cleared ready to receive the first partial product.

During the multiplication process the multiplicand is shifted left two places at a time round the loop formed by the registers RD and RD*. Similarly the multiplier is shifted right two places through the registers RS and RS*.

The two least significant bits of the multiplier are applied to the multiplier decode together with the 'sign' bit of the previous pair of multiplier bits (i.e., bit 32 of RS). From the decode a signal is produced which gates a function of the multiplicand in RD to the adder RA, (i.e., Zero, $\pm$ multiplicand, $\pm$ 2 x multiplicand).

MULTIPLIER DECODE

RS30	RS31	RS32	Multiple of RD added
0	0	0	0
0	0	1	+1
0	1	0	+1
0	1	1	+2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

The partial product in RB is added to this function to produce the cumulative partial product in RB. The process continues until all significant bits of the multiplier have been applied to the decode, or until the multiplicand becomes zero, at which time the multiplication is complete and RB now contains the true product. If multiply is terminated for multiplicand (RD) = 0, BOD 13 is set to a 1. If overflow occurs BOD5 is set and BFAULT may be produced (see above).

DIVIDE and REVERSE DIVIDE (/) (ø)

A dummy instruction which is accepted and then ignored.

NEW EQUIVALENCE / AND / OR (#) (&) (V)

The operand (RD) is fed to RL with the content of RB and the logical function specified is performed. The result is loaded into RB.

(Note:- RB V RD ≡ (RB & RD) V (RB $\underline{\#}$ RD) and therefore for the V operation both & and $\underline{\#}$ are specified to the logic.)

SHIFT (a)

The shift is arithmetic and the shift count is held in RC. The count is specified by the least significant 5 bits of the operand (RS). The most significant 4 bits of RC form the shift counter. The operand to be shifted is loaded from RB to RD.

The shifting operation, two places left or right (specified by BIT 0 of RS), is performed in the loop formed by the registers RD and RD*. The shift count is decremented in the most significant 4 bits of RC until the count is zero. (This applies for left shifts only. In the case of right shift a negative number is incremented to an all 1's condition). There is now only one operation left to be performed. The operand in RD is gated to RD* shifted 1 place left or right, if the least significant bit of RC is set, and is gated directly (zero shift) if this bit is unset. The final result in RD* is gated to RB.

Overflow can occur on left shifts only. If overflow occurs BOVF is set and B FAULT may be produced (see above).

COMPARE (COMP)

The operand (RE) is gated to the adder RA where it is subtracted from the content of RB (RB unchanged). The result is formed in RA. If overflow occurs BOVF is $\underline{\text{NOT}}$ set. The content of RA is then tested.

1. BOVF is gated to T0
2. Content of RA ≠ 0 then T1 set

However, should overflow occur in the adder the sign of RA is incorrect and is inverted before the test.

3. If answer is < 0 the T2 is set

i.e., T2 := sign of RA

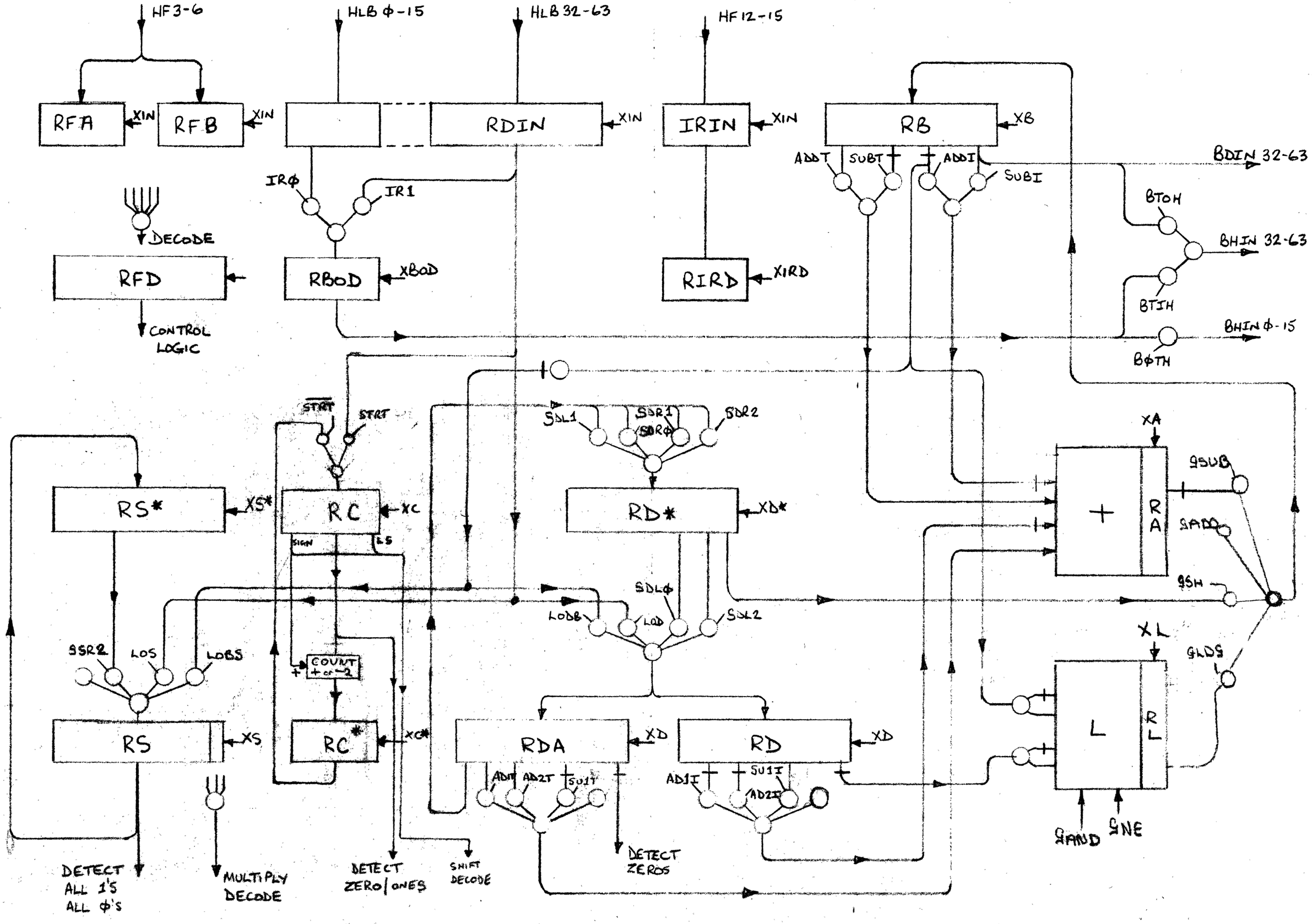
:= [RA00≠ACVR]

A strobe is generated when the lines are true and all four signals are sent to PROP.

COMPARE and INCREMENT (COMP & INC)

The compare order is carried out (see above). Then the content of RB is gated to the adder RA, where it is incremented by 1. The result is then loaded into RB.

If the overflow has occured BCD5 is set but this is not transmitted to the TO line. B FAULT may be produced (see above).



B-ARITH UNIT - OVERALL DIAGRAM

5/FIG 1

Chapter 10 The Local Store

10.1 PLESSEY STORE

10.1.1 INTERFACE SIGNALS FOR EACH STACK:-

INPUT SIGNALS

LINES VALID — The inner and outer of this cable must be shorted together to indicate to the memory that the control lines etc. have reached sensible logic levels, and that store cycles may begin when initiated.

D.C. RESET — Resets the store control circuits when taken to a '1'.

INITIATE — Starts a store cycle (according to the state of the other control lines), when taken to a '1'. This time is defined as TO.

READ INTO REGISTER (RIRG)) These two lines indicate whether a read, write, or read
LOAD REGISTER (LDRG)) and mark cycle is to be performed.

MASK BITS (8 IN ALL, 1 PER BYTE) — During a read and mark operation, bytes for which the mask bit is a '1' are written to, others are unchanged.

ADDRESS — 12 bits of address specify which 64 bit location of 4K such locations is to be accessed. The stack itself is organised as 2K locations each two words wide, and the least significant address bit specifies which word (of the two accessed in each cycle) is actually wanted.

DATA — 64 data and 8 parity bits constitute 72 input lines to each stack. No parity checking is performed either in the store or in the local store interface: no distinction is made between data and parity bits. The store is thus 2K words of 144 bits internally.

READ ODD DATA REGISTER (ROD) — During each access two words are actually read from the stack as explained above. If an even addressed location is read, and this signal (READ ODD) is made a '1', after the even address has been output from the store, the contents of the other half of that location in the store, (i.e. the next 64 bit address) is automatically placed on the output lines of the store.

OUTPUT SIGNALS

BUSY — This signal is a '1' from 40ns after TO (INITIATE) until the end of the write phase of that cycle. When BUSY becomes a '0' the next cycle may be initiated. There is an interlock within the store such that if INITIATE is held at '1' continuously the store will cycle at its maximum rate as determined by the BUSY signal.

ORDER CAN CHANGE (OCC) — This approx 40ns pulse indicates that the inputs to the store have been noted, and may be changed.

DATA AVAILABLE (DA) — This 40-80ns pulse can be adjusted to occur between TO and TO + 130ns during a read operation.

READ TIMING (RTIM) — This 40-80ns pulse can be adjusted to occur between TO and TO + 150ns during a read operation.

WRITE TIMING — This signal is not used.

DATA OUTPUT — 64 bits + 8 parity, 72 lines in all.

ODD DATA AVAILABLE (ODA) — TO' is defined as setting time of even data on the store output buffer, or time at which READ ODD becomes a '1' - whichever is later. The ODD DATA AVAILABLE pulse (40-80ns) can be adjusted to occur between TO' + 20ns and TO' + 40ns.

GENERAL POINTS

Data inputs must be present for $\geq$ 35ns.
O.C.C. occurs 20ns after TO and is a 40-80ns pulse.
Access time is nominally 130ns.
Cycle time is nominally 250ns (300ns on read and mark).

10.1.2 REMOTE SELF TEST SIGNALS

INPUT SIGNALS

PERFORM SELF TEST (PST) This signal must not be taken to '1' while the store is busy. When taken to a '1' self test is performed according to the signals below:

FUNCTION 0 = CLEAR/WRITE 1 = READ/RESTORE

REMOTE SELF TEST PATTERN BITS (P1 P2 P3)

WORST PATTERN	P1	P2	P3
WORST PATTERN	P1	P2	P3
ALL LOW (1's)	P1	P2	P3
ALL HIGH (0's)	P1	P2	P3
ADDRESS PATTERN	P1	P2	P3

POWER SUPPLY MARGINS MARG1 = 1 ON MARGINS MARG1 = 0 ON MARGINS
MARG2 = 1 gives margins according to V. Store.
MARG2 = 0 gives inverse of V. Store.

TRANSFER ADDRESS (TAD) Used after the store has stopped cycling after failing while on self test. When this signal is taken to '1', the contents of the address register are placed on the data output lines (bits 52-63) within 250ns.

FIX ADDRESS BITS (F1,F2) The least significant but 1 address bit (specifying 128-bit words) is held or allowed to count as below:

hold at 0	F1	F2
hold at 1	F1	F2
counting	F1	F2

SEQUENCE ONCE If this bit is a '0' self test is performed continuously. If it is a '1' the store will stop after one cycle of all the address bits.

OUTPUT SIGNALS

FAULT This is taken to '1' to indicate that a fault has occurred while on self test. It is forced to '0' when self test (PST) is taken to a '0'.

FAILING ADDRESS is output on data bits 52-63 from the store when TAD = 1.

10.2 Local Store Interface

10.2.1 Introduction

Within the MU5 system the Local Store may be accessed by either the Store Access Control (constituting a direct link with the MU5 central processor) or the Exchange (in response to the request of any other unit in the system, including MU5). The Local Store itself consists of four stacks of 4096 seventy two bit words, and its properties are defined in the Plessey sales specification. The Local Store Interface enables each Stack of store to be cycled independently under the control of either the Store Access Control (SAC) or the Exchange. Thus successive store accesses may be interleaved between the four stacks with consequent improvement in rate of access. Self test functions can be performed on any stack under manual or processor control. Should any of the four stacks not be available there are twelve fail-soft modes through which certain address bits are interchanged. A four way interleaved system breaks down should any stack fail, and the fail soft modes enable two way interleaving when only two stacks are available or continuous addressing of 4K, 8K or 12K words when 1, 2, or 3 stacks are operating.

10.2.2 Format of Data

A maximu eight (eight-bit) bytes of data (with 8 associated parity bits) can be transferred in one request. This is organised as most significant four bytes, four associated parity bits, followed by least significant four bytes and four parity bits, 72 bits in all. Half-width transfers can be made, i.e. m.s. four bytes only or l.s. four bytes only. Odd parity is employed, i.e. all zero data would require a '1' parity.

RIN=1, WIN=1, SINGIN=1, RODIN=0. 72 bits are transferred from the addressed location to the output buffer. If EVENIN= 1, the most significant four bytes from the input buffer are written to the m.s. half of the addressed location. The least significant half is unchanged. The converse applies if EVENIN = 0.

RIN=1, WIN=1, SINGIN=1, RODIN=1. 72 bits are transferred from the even location addressed to the output buffer. The 72 bits from the associated odd location are next transferred. If EVENIN=1 the most significant four bytes from the input buffer are written to the m.s. half of the addressed even location.The least significant half is unchanged. The converse applies if EVENIN = 0.

b) VSTAK = 1 Accesses to VSTORE

RIN = 1, WIN = 0. The store itself is not accessed. A register within the Local Store Interface is selected by address bits 10, 11, and 12 from SAC. Its contents are gated to the appropriate output buffer, depending on the state of EIN.

RIN = 0, WIN = 1. The store itself is not accessed. Address bits 10,11, and 12 from SAC select a register within the Local Store Interface, to which the contents of the input buffer are loaded (Certain of the lines in VSTORE have read-only or write-only access). On completion of the request an SVTREE pulse is sent to SAC.

In order to perform SELF TEST under processor control, certain lines in VSTORE must be set up to specify the type of test to be carried out, and on which stack they will operate. VSTORE lines are defined in the MU5 Basic Programming Manual.

10.2.4 Mode of Operation

a) When initiating a store request SAC provides data bits (DISAC 00-71) and address bits (AISAC 00-15) in addition to the control waveforms listed below:-

INLSIF This signal initiates a request according to the state of the other relevant waveforms. In order to reduce transition effects this signal is sent as a change of level from '0' to '1' or '1' to '0'.

EIN=1 indicates a request originating from Exchange. Data from a read is sent to Exchange with a signal SO. EIN=0 indicates a SAC request, and data from a read operation is sent to SAC with a signal DASAC (which is sent as a change of level, as is INLSIF).

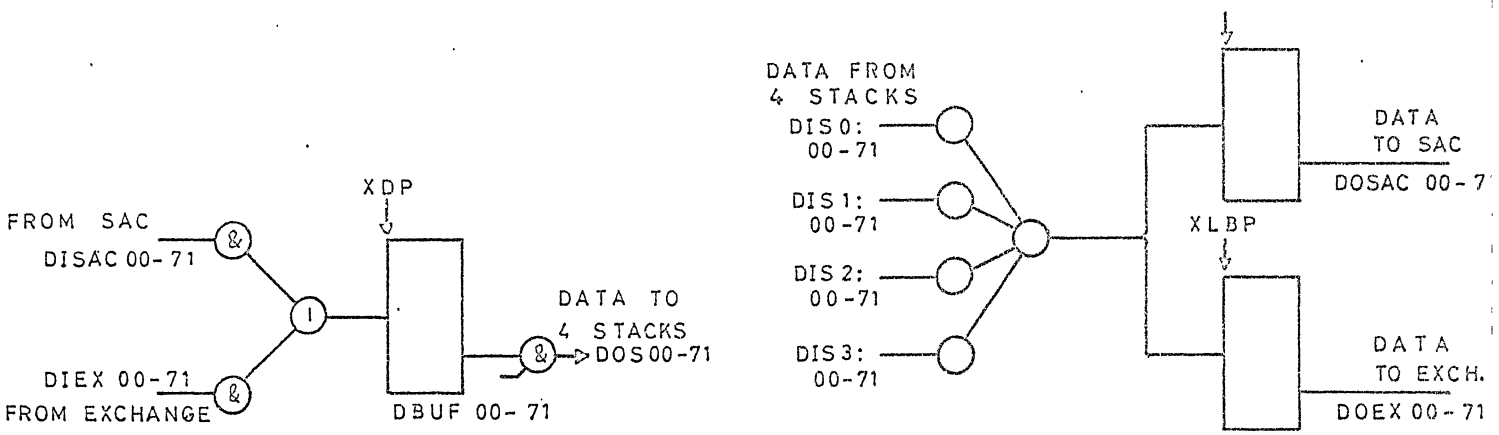
RIN, WIN, SINGIN, EVENIN, RODIN, VSTAK. The interpretation of these signals is described in detail in section 3. If SINGIN=1 at initiate time of a write operatio a 'read and mark' operation is forced: data from the store is read (but not sent to Exchange or SAC), the appropriate part of the word is written to (as required by the request), the rest of the word being written back unchanged.

ENSTAKO,1 These two bits are decoded to access one of stacks 0-3 if VSTAK=0.

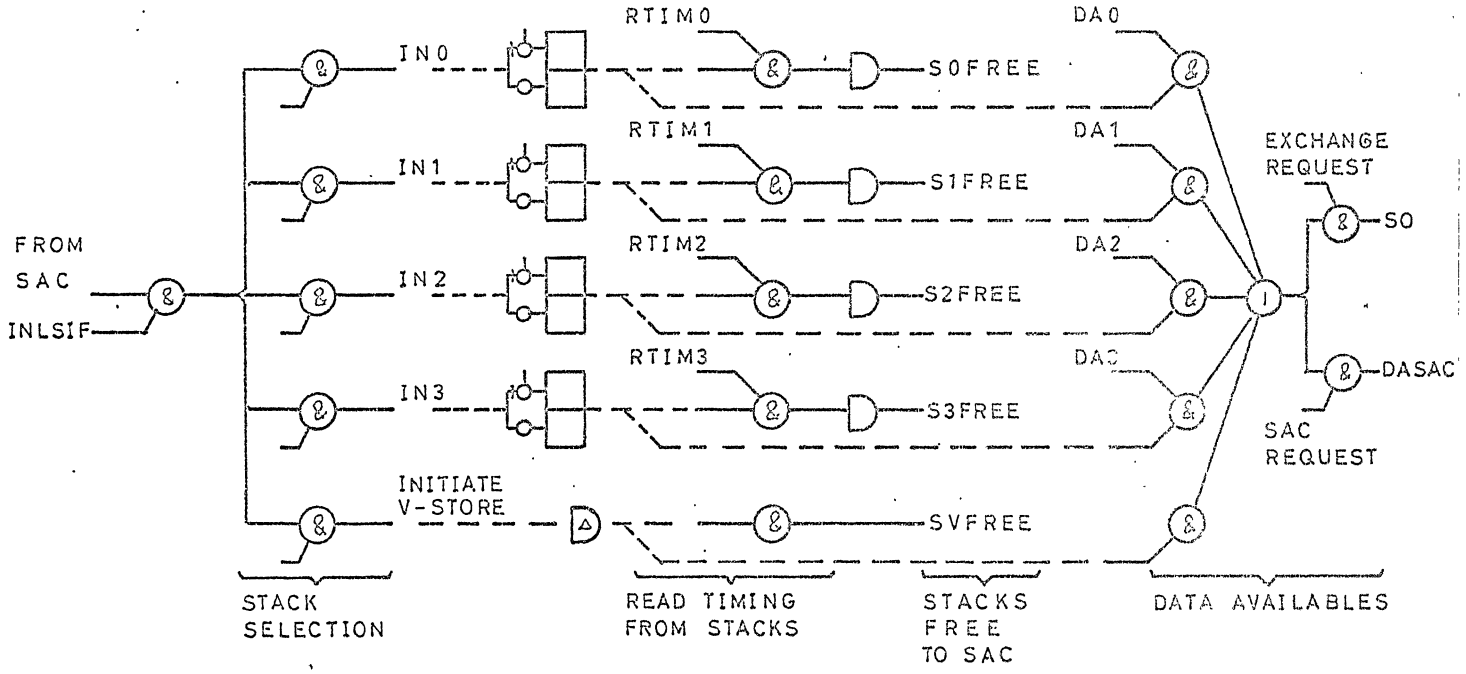
b) Assuming a read and read odd function has been specified, after the above waveforms have been decoded and staticised within the Local Store Interface, the appropriate stack is initiated. The address is placed on the store address line (modified if necessary by the FAIL SOFT MODE bits in the VSTORE). When the stack acknowledges receipt of initiate for a read cycle another signal is sent to the stack to specify that the associated odd location is to be read in addition to the even location already addressed. Some short time later 'data available' from the even location will be received, and data is strobed into the appropriate output buffer. Approximately 40ns later 'data available' from the odd location is received, and the appropriate output buffer is again strobed. A DASAC or SO signal is sent on both occasions. On completion of the request the appropriate stack free signal is sent to SAC (SOFREE to S3FREE). The cycle time for each stac is about 250ns. Therefore, requests can be interleaved between the four stacks approximately every 65ns, and four separate timing loops are provided, one for each stack. As each DASAC is sent, two signals are supplied to SAC to indicate which of the four stacks is responding, (HAL0, HAL1).

c) The Local Store Interface receives requests from Exchange via the pulse SI (supplied inverted by Exchange). Apart from the UNIT and TAG bits, and incoming data, the control and address bits are passed onto SAC, with a pulse INEXSAC. SAC then arranges a timeslot for the request, which is then carried out as described previously (EIN=1 for Exchange requests). Should Exchange access the area of store defined by AI1=1, the request is not passed onto SAC but is executed entirely within the Local Store Interface. A write operation has no effect except to signal SBFREE to Exchange. During a read operation the Exchange output buffer is forced to zero (with 1's in parity bits) and SO is signalled to Exchange.

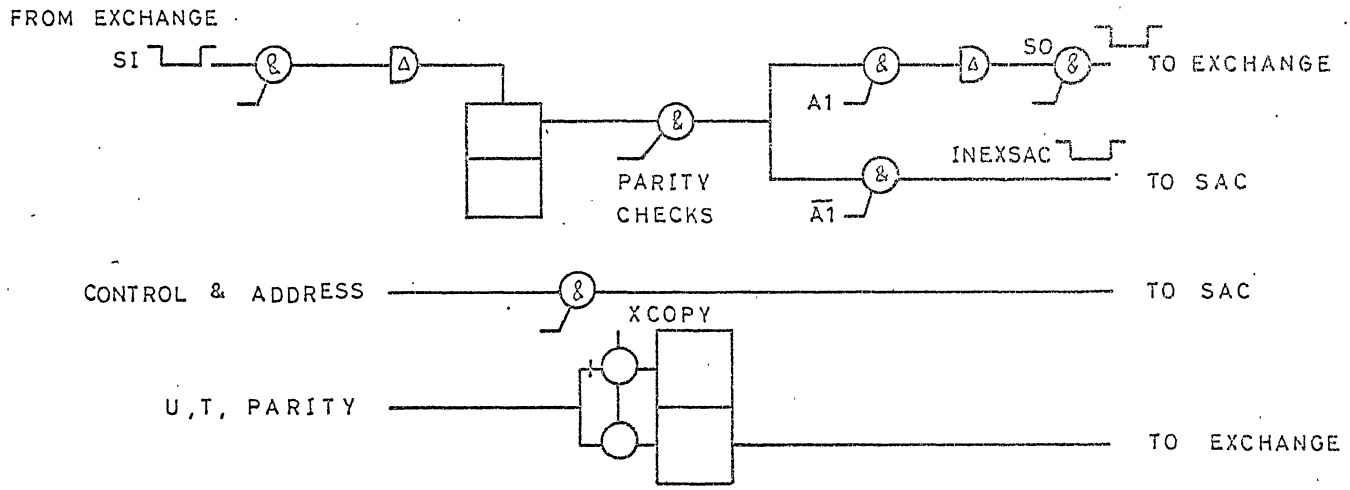
d) VSTORE requests. VSTAK=1. Address bits 10,11,12, select a line within the VSTORE. Read accesses take approximately 500ns (including the 'transfer failing address' line), and write accesses about 200ns. VSTORE accesses may not be overlapped with <u>any</u> succeeding requests. i.e. nothing further may be initiated until SVFREE has been sent to SAC. This is because the FAIL SOFT MODE is contained within the VSTORE, and any attempt to alter the mode while accessing the store may have disastrous results.



DATA PATHS AROUND THE LOCAL STORE



CONTROL PATHS AROUND THE LOCAL STORE - SAC INTERFACE



CONTROL PATHS AROUND THE LOCAL STORE - EXCHANGE INTERFACE

10/FIG.1 LOCAL STORE INTERFACE