PICTURE SYSTEM 2
Maintenance Manual
Volume 1

Customer Engineering Dept.
Evans & Sutherland Computer Corporation
580 Arapeen Drive
Salt Lake City, Utah 84108

First Edition July 1977

TABLE OF CONTENTS

APPENDIX

1.0 Introduction

This Maintenance Manual was created as a support item for both E&S Customer Engineers and on site maintenance personnel who have maintenance responsibility for the E&S PICTURE SYSTEM 2. To perform maintenance at the component level, a broad understanding of the operation and theory of the machine is a necessity. The PS2 diagnostics were designed to isolate hard failures to a functional unit within the system. To quickly isolate the problem to the component, the troubleshooter must understand the proper operation of the malfunctioning unit and the associated unit in which it is in communication with. The intent of the manual is to serve as both a theory of operation text and a hardware reference text for each functional unit in the standard PICTURE SYSTEM 2. It is assumed that the reader has a maintenance background and experience with troubleshooting TTL circuits.

The Picture Controller computer for PICTURE SYSTEM 2 may be one of many types; however, this manual assumes a Digital Equipment Corporation PDP-11 computer as the Picture Controller.

2.0 Scope of the Manual

Section 3 of the manual deals with installation of the PICTURE SYSTEM 2.

Section 4 outlines the general theory of operation of the standard PICTURE SYSTEM 2.

Section 5 details the theory of operation on a functional unit level and comprises the major portion of the manual. Exact Bit definitions for status and control registers as well as other register definitions are excluded; therefore, many references are made to the Picture System 2 Reference Manual where these definitions exist. When describing the detailed operation of specific circuits, references are made to the logic drawings contained in the PICTURE SYSTEM 2 Drawing Set.

Appendix A and B contain the MAP and Refresh Controller algorithms.

3.0 Installation

Installation of the PICTURE SYSTEM 2 involves the installment of a DEC PDP-11 computer by DEC personnel and the installment of the E&S equipment by an E&S Customer Engineer. The E&S equipment should be unpacked, set in place, and visually inspected for shipping damages.

3.1 Picture Controller Interface Installation

The Picture Controller Interface consists of three cards:

195131-100
195106-100
195105-100

The 195131-100 is built on a DEC hex module and plugs into a peripheral slot in the DEC equipment. This slot should be wired for NPR operation. The 195105 and 195106 cards plug into the PICTURE SYSTEM backpanel as indicated by the stuffing chart, 195101-900 (see the drawing set).

3.2 Picture Display Installation

The Picture Display connects to the scope driver card, 195211-100, by a coax cable set.

3.3 Power Connections

The power connection to the PS power controller should meet the following specifications:

Primary Power
- 115 volts, 60 Hz
- 30 Amp
- Single Phase
- Two wire + ground common to Picture Controller ground
- Hubbel 2610 or equivalent

The Picture Display Power card is plugged into the power control panel. Drawing 195100-100 illustrates the primary power connection to the power control panel and the power distribution to power supplies, clock assembly, etc.

Before applying power to the Picture System, check for shorts between all DC sources and ground. After power is applied, adjust all DC power sources to the correct level at the back-panel.

3.4 Verification of Proper Operation

To verify proper operation of the PICTURE SYSTEM, run the standard PS2 Acceptance Tests. This procedure is outlined in the PS2 Acceptance Tests document.

4.0 General Theory of Operation

The standard PICTURE SYSTEM 2 contains the following functional units:

1. Picture Controller Interface
2. PICTURE SYSTEM Bus, PSBUS
3. PSBUS Arbiter
4. Picture Processor
5. PICTURE SYSTEM Memory, PSMEM
6. Real Time Clock, RTC
7. Refresh Controller
8. Picture Generator

This manual (volume 1) deals with all of the above functional units with the exception of the Picture Generator. Volume 2 deals with the Picture Generator.

4.1 Picture Controller Interface

The Picture Controller Interface provides two data paths between the Picture Controller computer and the PSBUS. The interface also provides an interrupt link between the PICTURE SYSTEM and the Picture Controller. The two data paths are:

1. Direct IO Path (DIO)
2. Direct Memory Access Path (DMA)

4.1.1 Direct IO Path

The DIO consists of a direct interface between the PDP-11 UNIBUS and the PSBUS. It is used for direct (processor controlled) transfers between UNIBUS address space and PICTURE SYSTEM address space. The interface consists of a Direct IO PICTURE SYSTEM Address register (DIOPSA) and a Direct IO PICTURE SYSTEM Data register, (PSDATA), both of which are UNIBUS addressable. Prior to the transfer of data between a UNIBUS address and a PSBUS address, the DIOPSA must be set up to point at a desired PSBUS address. The actual transfer is initiated by reading or writing the PSDATA register. The PSDATA register serves as a buffer to hold the data being either sent to the DIOPSA destination or retreived from the DIOPSA source. The DIO is considered an active device, that is, no other PICTURE SYSTEM device can command it. The DIO, after commanded by the Picture Controller, always initiates either a read or write to a PICTURE SYSTEM passive device. The DIO initiates the transfer by requesting the PSBUS and when granted gates the DIOPSA (address) to the PSBUS, then either gates data to the bus or receives data from the bus depending on the nature of the transfer.

4.1.2 Direct Memory Access Path

The DMA path also consists of an interface between the

UNIBUS and the PSBUS; however, it is used for data block (non processor controlled) transfers between UNIBUS address space and PICTURE SYSTEM address space. The DMA path may be programmed to be either an active or passive device; therefore, it can actively fetch data from UNIBUS address space and relay it to PS address space or it can passively wait for data from a PS device, then relay it to UNIBUS address space. In either mode of operation, it must be previously set up by the Picture Controller for a block transfer, then commanded to "GO". For each word transfer, the UNIBUS must be requested and granted; also, for word transfers to PS address destination, the PSBUS must be requested and granted. The PICTURE SYSTEM DMA Device contains a DMA PICTURE SYSTEM Address register (DMAPSA) which tracks the inter-block PS address. The UNIBUS DMA Device contains a UNIBUS Address register (DMABA) which tracks the inter-block UNIBUS address and a DMA Word Count register (DMAWC) which counts the number of words to be transferred. When the last word of the block has been transferred, the DMAWC register acclaims the DMA path to be READY for another block transfer.

4.2 PSBUS Arbiter

The PSBUS Arbiter controls the use of the PSBUS. More than one PS device may need use of the bus at a given time; therefore, a round robin priority scheme is implemented to grant requesting devices. Eight active requests lines input to the

arbiter. A request line is hardwired to an active device and when the bus is needed, and active device inserts a request line to the arbiter. The eight requests are assigned a priority level and if granted, that request is rotated to the lowest priority level; therefore, all active devices, in general, have equal opportunity for the PSBUS.

4.3 Picture Processor

The Picture Processor is a special purpose vector processor which receives commands and data from a PICTURE SYSTEM device and outputs processed commands and data to another device. Usually output data is directed to the Picture Generator for display; however, output may be directed to PS Memory or to the DMA as data to be stored in UNIBUS address space. Input to the Picture Processor may be sourced by either the Direct IO, DMA, or PS Memory. The Picture Processor input device may be either active or passive, that is, it may actively fetch input data or passively wait for data directed to it. The Picture Processor output may also be either active or passive. Also, input and output may be either 16 or 24 bit precision. These variables are programmably set up by writing a status word in the Picture Processor.

The Picture Processor operates on an instruction set. After inputting a command, appropriate action is performed on either incoming data or internal data. The instruction set is de-

tailed in the PS2 Reference Manual, pp 2-36 through 2-54.

In general the Picture Processor receives commands and point vector data in the Data Space Coordinate system and transforms the point vector data into the Screen Coordinate system for subsequent display by the Picture Generator. Functions performed on the input data include:

1. translation
2. rotation
3. clipping
4. perspective
5. view port mapping
6. zooming
7. matrix concatenation
8. pass data unformated
9. pass data formated

These functions are performed by receiving a command, and traversing states of the MAP algorithm which operates on the data appropriately. The three main units of the Picture Processor are:

1. MAP Input Controller
2. MAP (Matrix Arithmetic Processor)
3. MAP Output Formatter

These units are described in detail in section 5, the detailed theory section of this manual.

4.4 PICTURE SYSTEM Memory

The PICTURE SYSTEM Memory is a dual-port MOS memory (distinct from the Picture Controller's) organized as addressable 16-bit words. This memory is available in increments of 16K words, expandable to 64K words of memory, dependent upon user requirements.

PICTURE SYSTEM Memory may be used in a variety of ways to satisfy the user's application. Typically, a portion of the PICTURE SYSTEM Memory serves as a refresh buffer into which data, still in digital form, is deposited. This data represents information to be shown on the Picture Display. For each frame displayed, the Refresh Controller reads the data from the PICTURE SYSTEM Memory and channels this data to the Line Generator where the data are then converted to analog signals to drive the Picture Display.

4.5 Real Time Clock

The Real Time Clock coordinates the picture update process with the picture refresh process by implementing a programmable interval at which a clock interrupt occurs. This interrupt causes the Picture Controller program to check the condition of the update and refresh process to determine if a new frame may be initiated. The basic timing of the clock is derived from two 60 Hz inputs to produce a 120 Hz

clocking signal which counts two counters. The two counters procide a interrupt interval and a refresh sync interval. At the interrupt interval, a request for interrupt is generated. At the sync interval, a sync pulse is generated for the refresh controller. This pulse is used by the refresh controller during automatic refresh mode.

4.6 Refresh Controller

The Refresh Controller is the unit of the Picture Generator that controls the refreshing of images on the Picture Display. The Refresh Controller reads data from the PICTURE SYSTEM Memory, a refresh buffer, and channels this data to the Line Generator for display. Under program supervision, the Refresh Controller is used to manage the organization of the PICTURE SYSTEM Memory. It also contains special-purpose hardware to facilitate memory segmentation and management.

In single-buffer mode, the entire refresh buffer is used to store a single display frame. In this mode, refresh may be initiated from a partially-updated display frame consisting of some lines from the new frame and some lines from the previous frame.

In double-buffer mode, one-half of the refresh buffer is designated as an old frame and one-half a new frame. Refresh is then initiated from the old frame while the new frame is

being constructed. When construction of the new frame is completed, the frame buffers are swapped and the newly-constructed frame is displayed. The space occupied by the old frame becomes available for new frame construction.

The segmented-buffer mode provides the most general use of the refresh buffer for the display and updating of data. Typically, a frame consists of portions which need not be updated as frequently as others. Ideally, these portions should be updated as separate parts, or segments, of the frame. The Refresh Controller facilitates the use of the refresh buffer in this mode by allowing each of the separate portions of the refresh buffer to be given a name by which the segment may be replaced, appended to, deleted, etc.

The Refresh Controller also improves the utilization of the refresh buffer by providing, in segmented-buffer mode, for the reclamation of unused portions of the refresh buffer that have been left by deleted segments. This prevents fragmentation of the refresh buffer into small, unuseable areas.

5.0 Detailed Theory of Operations

5.1 Picture Controller Interface

The Picture Controller Interface is the communication link between the host computer and the PICTURE SYSTEM. All command and data transfers between the two machines take place through this interface. The interface consists of three devices:

1. Direct I/O Path (DIO)
2. Direct Memory Access Path (DMA)
3. Interrupt Control

A DIO transfer is initiated by host computer control. Also, a DMA block transfer is set up and initiated by host control; however, individual word transfers within the block are initiated by the DMA control logic. This provides a mechanism by which the host computer may set up a block transfer, then execute other tasks while the DMA Controller manages the word transfers of the block. The DMA Controller gains access of the host transfer lines (data bus) for each word transfer, then releases control after each transfer.

The Interrupt Control monitors the DMA Interrupt Request and three interrupt request lines from the PICTURE SYSTEM.

1. Real Time Clock Interrupt
2. Device Interrupt
3. System Interrupt

The DMA Interrupt request, if enabled, requests an interrupt to be serviced when the DMA is ready or when a non-existant memory (NEXMEM) location was referenced by the DMA.

The Real Time Clock Interrupt request, if enabled, requests an interrupt at pre-defined programmable intervals. This program interrupt initiates a status check of the refresh process and update process by the program.

The Device Interrupt request occurs when a PICTURE SYSTEM Device needs servicing. Typical devices include: Function Switches, Keyboard, Tablet, etc.

The System Interrupt request occurs from designated PICTURE SYSTEM control functions such as: "Refresh Stopped", "MAP Stopped", "HIT Request", etc.

A detailed description of the interrupt requesting system exists in the PICTURE SYSTEM Reference Manual, pp. 2-130 thru 2-139.

The Interrupt Control, which monitors the four interrupt

requests, gains control of the host computer data bus, passes the interrupt vector for the honored request to the host computer data bus, and asserts the interrupt line to the host computer thus causing the host to execute the appropriate service routine.

A block diagram of the Picture Controller Interface is illustrated in Figure 5.1-1. The interface is implemented on three cards.

1. 195131 PDP-11/PS Interface
2. 195105 Direct IO/Data In
3. 195106 DMA/Data Out

The 195131 card is located in the host computer and the 195105 and 195106 are located in the PICTURE SYSTEM Backpanel.

A cable connects the 195105 and 195106 in the PICTURE SYSTEM to the PDP-11 Interface card in the host computer with 46 differential pairs of control and data lines.

5.1.1 PDP-11 Interface

The 195131 PDP-11 Interface communicates with the PDP-11 UNIBUS with 57 lines. Communication with the UNIBUS is asynchronous, therefore; no synchronous clock

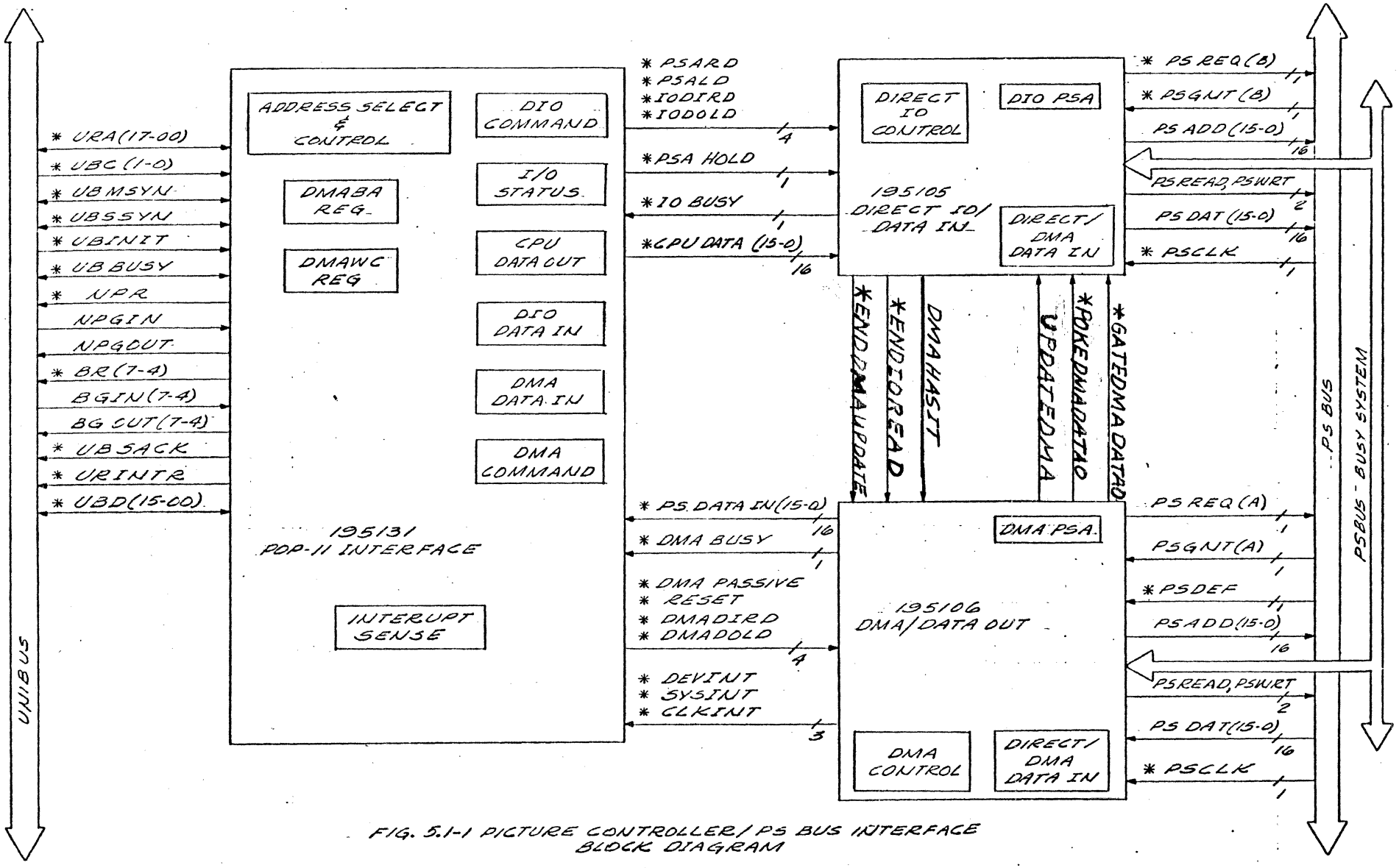


FIG. 5.1-1 PICTURE CONTROLLER/PS BUS INTERFACE BLOCK DIAGRAM

signal exists on the 195131. Likewise communications between the 195131 and the PICTURE SYSTEM cards (195105-195106) is asynchronous; however, all communication between the 195105-195106 cards and the PICTURE SYSTEM Bus (PSBUS) is synchronous. The PSBUS contains a synchronous clock line which is input to both the 195105 and 195106 cards.

The PDP-11 Interface card consists of the following:

1. Address Select & Control
2. DMA Bus Address Register (DMABA)
3. DMA Word Count Register (DMAWC)
4. DMA Data In Register
5. DMA Command Register
6. Direct I/O Command Register
7. I/O Status Register
8. CPU Data Out Register
9. Direct I/O Data In Register
10. Interrupt Sense and Control

5.1.1.1 Address Select & Control

The Address Select & Control monitors the UNIBUS address lines. If one of the five PDP-11 Interface register addresses exist on the UNIBUS the address select decodes which register the address is for and then waits for the UNIBUS MSYN signal which strobes the UNIBUS data in the selected register; else, enables the data out (to the UNIBUS) from the selected register. Whether the UNIBUS

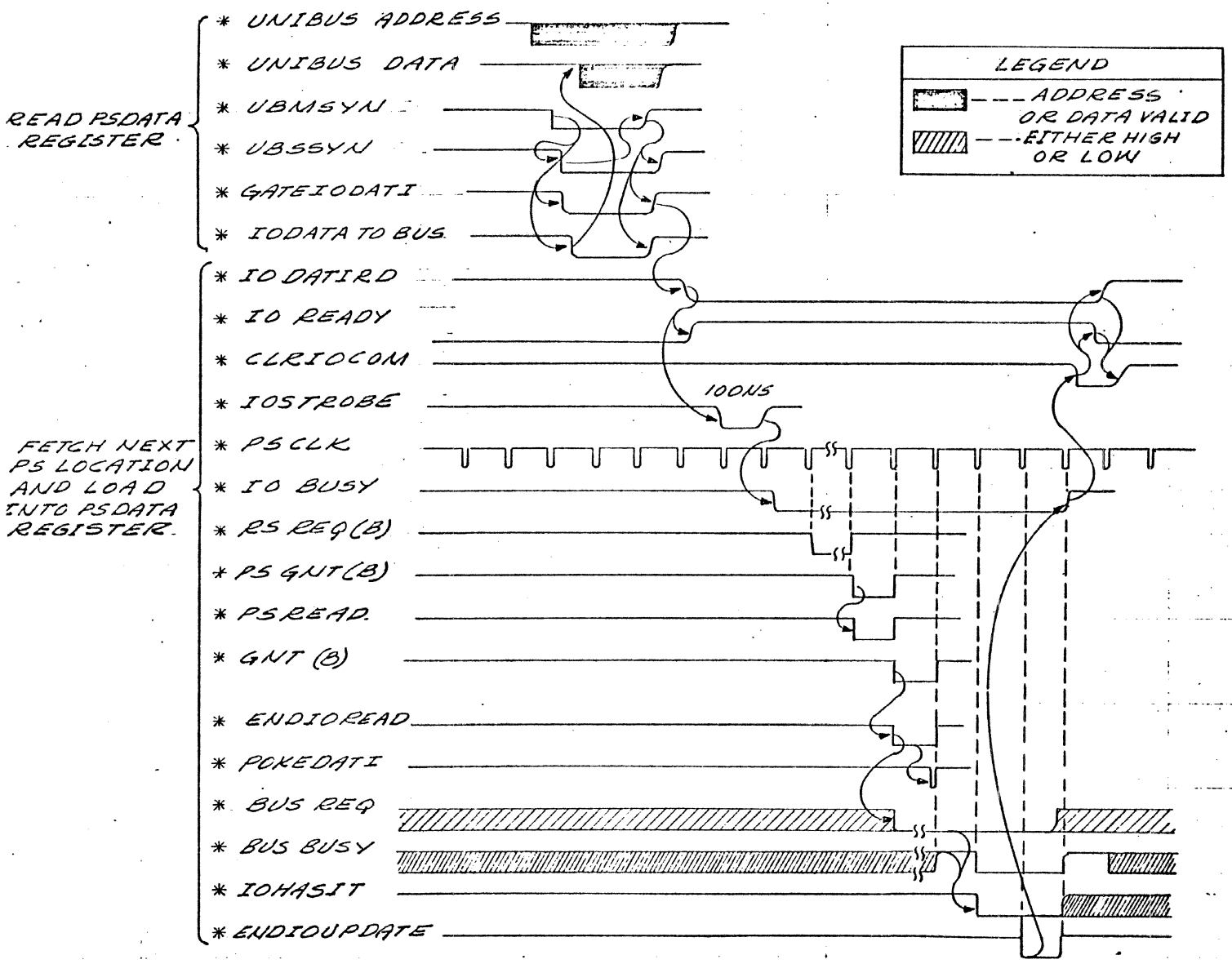


FIGURE 5.1-2 DIRECT IO READ TIMING DIAGRAM

wants to write or read is decoded from the *C1 line of the UNIBUS. This logic exists on sheet 3 of the 195131-600 drawings.

5.1.1.2 DMA Bus Address Register (DMABA)

The DMABA register is used to address the UNIBUS during a DMA word transfer. This register is normally loaded with the beginning address of a data block and then incremented by one upon each word transfer. The DMABA register is only 15 bits and does not drive the LSB of the UNIBUS: therefore, incrementation by one effectively increments the UNIBUS address by two. Only word (not byte) addressing on the UNIBUS takes place during DMA transfers. The DMABA register is implemented on sheet 15 of the logic drawings.

5.1.1.3 DMA Word Count Register (DMAWC)

The DMAWC register keeps track of the word count during a block transfer. It is initially loaded with a negative two's complement number of the words to be transferred. Upon each word transfer it is incremented by one. When the register overflows the last word has been transferred. The DMAWC is implemented on sheet 15 of the logic drawings.

5.1.1.4 DMA Command Register

The DMA Command Register issues a DMA read or DMA write

command to the PICTURE SYSTEM DMA Controller. On the cable between the PDP-11 Interface and the PICTURE SYSTEM, these commands are named:

1. DMADIRD - DMA Data Input Read
2. DMADOLD - DMA Data Out Load

During a DMA write transfer from the UNIBUS to the PICTURE SYSTEM, the PDP-11 Interface performs the following steps:

1. Gains control of the UNIBUS.
2. Receives the UNIBUS data into the CPUDATA OUT register.
3. Releases control of the UNIBUS.
4. Issues a DMADOLD command to the PICTURE SYSTEM.
5. Waits for the PICTURE SYSTEM DMA Controller to go non-busy.
6. When the non-busy is received, goes to step 1 if the block transfer is not complete else goes to step 7.
7. Sets the DMA READY bit in the IOST register indicating the DMA is ready for program initiation of another block transfer.

During a DMA read transfer from the PICTURE SYSTEM to the UNIBUS, the PDP-11 Interface performs the following steps:

1. Issues a DMADIRD command to the PICTURE SYSTEM.
2. Waits for the PICTURE SYSTEM DMA Controller

to fetch the word. (waits for DMA BUSY TO go low).

3. Strobes the data from PSDATAIN (15-0) into the DMA DATA IN register.
4. Gains control of the UNIBUS.
5. Gates the DMA Data In register onto the UNIBUS and releases control of the UNIBUS when the destination device takes the data.
6. Goes to step 1 if the block transfer is not done; else goes to step 7.
7. Sets DMA READY in the IOST register, indicating ready for program initiation of another block transfer.

5.1.1.5 DMA Data In Register

The DMA Data In Register, in the PDP-11 Interface, buffers data received from the PICTURE SYSTEM. During a DMA IN word transfer, the DMA Controller issues a DMADIRD command to the PICTURE SYSTEM's DMA Controller, which gains control of the PS BUS and fetches the data word pointed to by the DMAPSA register. The data is put on the PSDATA (15-0) lines to the PDP-11 Interface. The PICTURE SYSTEM clears DMA BUSY which strobes the data into the DMA Data In Register. The PDP-11 Interface issues a request for the UNIBUS and eventually, when the request is granted, gates the DMA Data In Register onto the UNIBUS.

The DMA Data In Register is on sheet 14 of the logic drawings.

5.1.1.6 I/O Status Register (IOST)

The I/O Status Register implements status indicators of the DIO and DMA operations. It also contains a reset bit to initialize the DIO and DMA logic to their initial states and a GO bit to transfer control of a DMA block transfer to the DMA Controller after a block transfer has been set up under program control. A detailed description of the IO status bits exists in the PS2 Reference Manual, pp. 2-12 through 2-15.

DIO status bits are implemented on sheet 4 and DMA status bits on sheet 5 of the logic drawings.

5.1.1.7 Direct IO Command Register

The Direct IO Command Register is updated when a new command from the UNIBUS is sensed by the Address Select & Control logic. Four Direct IO commands are possible:

IOPSARD - Read the Direct IO PICTURE SYSTEM Address Register (DIOPSA)

IOPSALD - Load the DIOPSA

IODATIRD - Read the Direct IO Data Input Register

IODATOLD - Load the CPU DATA OUT Register via the Direct IO.

The Direct IO Command Register is implemented by four F/F's on sheet 4 of the logic drawings. The four commands F/F's drive four differential command line pairs on the cable to the Direct IO/Data In Card, 195105. These drivers are on sheet 10 of the logic (195131-600).

5.1.1.8 CPU Data Out Registers

The CPU Data Out Register buffers either DIO data or DMA data being sent to the PICTURE SYSTEM. This buffering enables the appropriate controller to store the data gathered from the UNIBUS and release control of the UNIBUS while waiting for the PICTURE SYSTEM to take the data. If the input data were not buffered, the UNIBUS would be tied up until the appropriate DIO or DMA logic interfaced to the PS BUS could take the data. The PICTURE SYSTEM Reference Manual refers to this register as the PSDATA Register when describing Direct IO write to the PICTURE SYSTEM.

The CPU DATA OUT Register is implemented on sheet 8 of the logic drawings.

5.1.1.9 Direct IO Data In Register

The DIO Data In Register buffers data from the PICTURE SYSTEM to the UNIBUS upon completion of a IODATIRD command. The register is referred to in the PICTURE

SYSTEM Reference Manual as the PSDATA register. If a user wants to read a PS Memory location, he must first point the DIOPSA to the PS Memory location desired.

```
MOV #MEMLOC, DIOPSA
```

Execution of the PDP-11 instruction sets up the DIOPSA and initiates a read of the PS Memory location specified by DIOPSA. The contents of the specified memory location are fetched and stored in the DIO Data In (PSDATA) Register. To actually retrieve the desired data the user must read the PSDATA register.

```
MOV PSDATA, SAVE
```

This PDP-11 instruction moves the contents of the PSDATA register to the specified UNIBUS address (SAVE) and initiates a IODATIRD command to the PICTURE SYSTEM DIO Interface which increments the DIOPSA (unless inhibited) and fetches data from the next PS Memory location pointed to by the DIOPSA.

Successive PS Memory locations may be retrieved by successive reads of the PSDATA register.

While the DIO path is busy, another PSDATA read cannot be initiated; therefroe, the user must test the IO READY bit in the IOST Register before initiating a read.

The DIO DATA IN Register is on sheet 14 of the 195131-600 logic drawings. The DIOPSA Register is on sheet 6 of the 195105-600 drawings.

5.1.2 Direct IO Path

The Direct IO path interfaces the UNIBUS to the PICTURE SYSTEM to enable programmable Direct IO transfers between UNIBUS address space and PICTURE SYSTEM address space. The actual hardware for the Direct IO path consists of:

1. Direct IO control and buffering on the PDP-11 Interface Card (195131-100).
2. Direct IO control and "Data In" buffering on the 195105-100 card in the PS backpanel.
3. "Data Out" buffering on the 195106-100 card in the PS backpanel.
4. Cable driver and receiver lines between the PDP-11 Interface and the 195105-195106 cards.
5. Control lines on the PS backpanel between the 195105 and 195106 cards.

A Direct IO transfer can take place during a DMA block transfer. If the DMA is writing and the DIO is reading, or visa versa, there is no conflict of data line usage between the PDP-11 Interface and the PICTURE SYSTEM; because, there are two separate

groups of data lines. (See Fig. 5.1-1)

1. CPUDATA (15-0) - Data from UNIBUS to PICTURE SYSTEM.
2. PSDATAIN (15-0)- Data from PICTURE SYSTEM to UNIBUS.

Also, if both DMA and DIO are writing, there is no conflict in usage of the CPUDATA (15-0) lines because UNIBUS mastership arbitration insures only one user of the lines at one time.

For example, if the DMA controller is bus master and the program encounters a direct IO write instruction to the PICTURE SYSTEM, the requesting UNIBUS device must wait for the current DMA word transfer to complete before it becomes bus master. While it is master, the DMA process is held up and the CPUDATA (15-0) lines are solely dedicated to the UNIBUS master device using the Direct IO path.

However, if both the DMA and DIO are reading, the PSDATAIN (15-0) lines must be arbitrated between two devices.

For example, a DMA may be performing a block of read transfers and a UNIBUS device may desire to read the PSDATA register in the Direct IO path. The UNIBUS device again will not be granted bus

mastership until the current DMA word transfer is complete. Once bus master the UNIBUS device relays the read command to the PDP-11 Interface's Direct IO Controller. The controller responds by enabling the PSDATA register onto the UNIBUS, issues a read command to the PICTURE SYSTEM, and acknowledges the UNIBUS master. The master releases control of the UNIBUS and the DMA controller regains bus mastership; however, the read command to the PICTURE SYSTEM causes its Direct IO controller to increment the DIOPSA, (if not inhibited), and retrieve the data in PS address location pointed to by the DIOPSA. After fetching the desired data off the PSBUS, the controller must transfer it across the PSDATA (15-0) lines to the PSDATA register. Remember the DMA may still be performing read transfers on these lines; therefore, an arbiter in the PICTURE SYSTEM's Direct IO controller must arbitrate these lines between the DMA and DIO as necessary. This arbiter exists on the 195105-600, sheet 4.

5.1.2.1 Direct IO Read

To read a PS address location(s) the user points the DIOPSA to the desired location:

```
MOV PSADD, DIOPSA
```

Loading the DIOPSA causes the PICTURE SYSTEM's Direct IO controller to retrieve the PS address data pointed to by the new DIOPSA contents and buffer it in the PDP-11 Interface PSDATA register. When the DIO becomes ready then the user may examine the PSDATA register:

```
MOV PSDATA, SAVE
```

Each read of the PSDATA register increments the DIOPSA by one (unless inhibited) and causes the Direct IO controller to retrieve the next sequential PS address contents and buffer the data in PSDATA. This enables the user to examine successive PS address locations by issuing successive PSDATA read commands.

Execution of the above instruction causes the following sequence of events to occur. The sequence is illustrated in Direct IO Read Timing Diagram of Figure 5.1-2.

1. The UNIBUS master device asserts the address on PSDATA and sets up control lines C0 and C1 for a full word read onto the UNIBUS.

2. The UNIBUS master device asserts *UBMSYN requesting the addressed device to respond.

3. The PDP-11 Interface's Direct IO control decodes the UNIBUS address and control lines and receives the master's request, *UBMSYN, (195131-600 sheet 3).

4. The Direct IO control enables the DIO PS DATAIN register (PSDATA) through the mux on sheet 16 and *IODATATOBUS asserts the data onto the UNIBUS.

5. *UBSSYN acknowledges the UNIBUS master and *GATEIODATI gets ready to issue a read command to the PICTURE SYSTEM. (sheet 4)

6. The trailing edge of *GATEIODATI sets IODATIRD which clears IOREADY F/F indicating to the program that the DIO path is busy. IODATIRD drives a cable line to the PICTURE SYSTEM's Direct IO control on the 195105 card.

7. The Direct IO control in the PICTURE SYSTEM receives IODATIRD and generates *IOSTROBE pulse of 100ns. (sheet 3 of 195105-600).

8. The trailing edge of *IOSTROBE asserts the *IOBUSY F/F which indicates to the PDP-11 Interface that the Direct IO is accessing a PS location for the PS DATA register. The trailing edge of *IOSTROBE pulse also increments the contents of DIOPSA by one (if not inhibited).

9. IOBUSY sets the *PSREQ(B) F/F in sync with the next *PSCLK pulse which issues a request for the PSBUS.

10. The PS arbiter eventually responds (in sync with *PSCLK) with *PSGNT which gives the PSBUS to the Direct IO controller.

11. *PSGNT will gate the contents of DIOPSA (PS address) and *PSREAD to the PS BUS (sheet 6).

12. The next clock buffers *PSGNT by setting *GNT(B) F/F. (During the clock period with *GNT(B) is set, the data from the destination location location is on the PS data lines of the PS BUS).

13. *GNT(B) asserts *ENDIOREAD which issues a request, BUSREQ, for the PSDATAIN (15-0) lines and the next clock pulse generates the *POKEDATI pulse which strobes the data from the PSBUS into the DIO/DMA Data In register file on sheet 6 of the 195106 card which drives the PSDATAIN (15-0) lines.

14. When the PSDATAIN (15-0) lines are not busy (may be in use by the DMA) the IOHASIT F/F (sheet 4 of 195105-600) is set indicating that the lines now belong to the DIO.

15. IOHASIT enables the DIO word in the DIO/DMA Data In register file out to the PSDATAIN (15-0) lines to the PDP-11 Interface.

16. The next clock asserts *ENDIOUPDATE which clears *IOBUSY on the next clock.

17. The PDP-11 Interface's Direct IO control, which has been waiting for the trailing edge of *IOBUSY, sets *CLRIOCOM and the data on PSDATAIN (15-0) is strobed into the PSDATA register. (195131-600, sheet 14)

18. *CLRIOCOM sets IOREADY, clears the read command F/F, IODATIRD (sheet 4), and the Direct IO path is ready for another transfer.

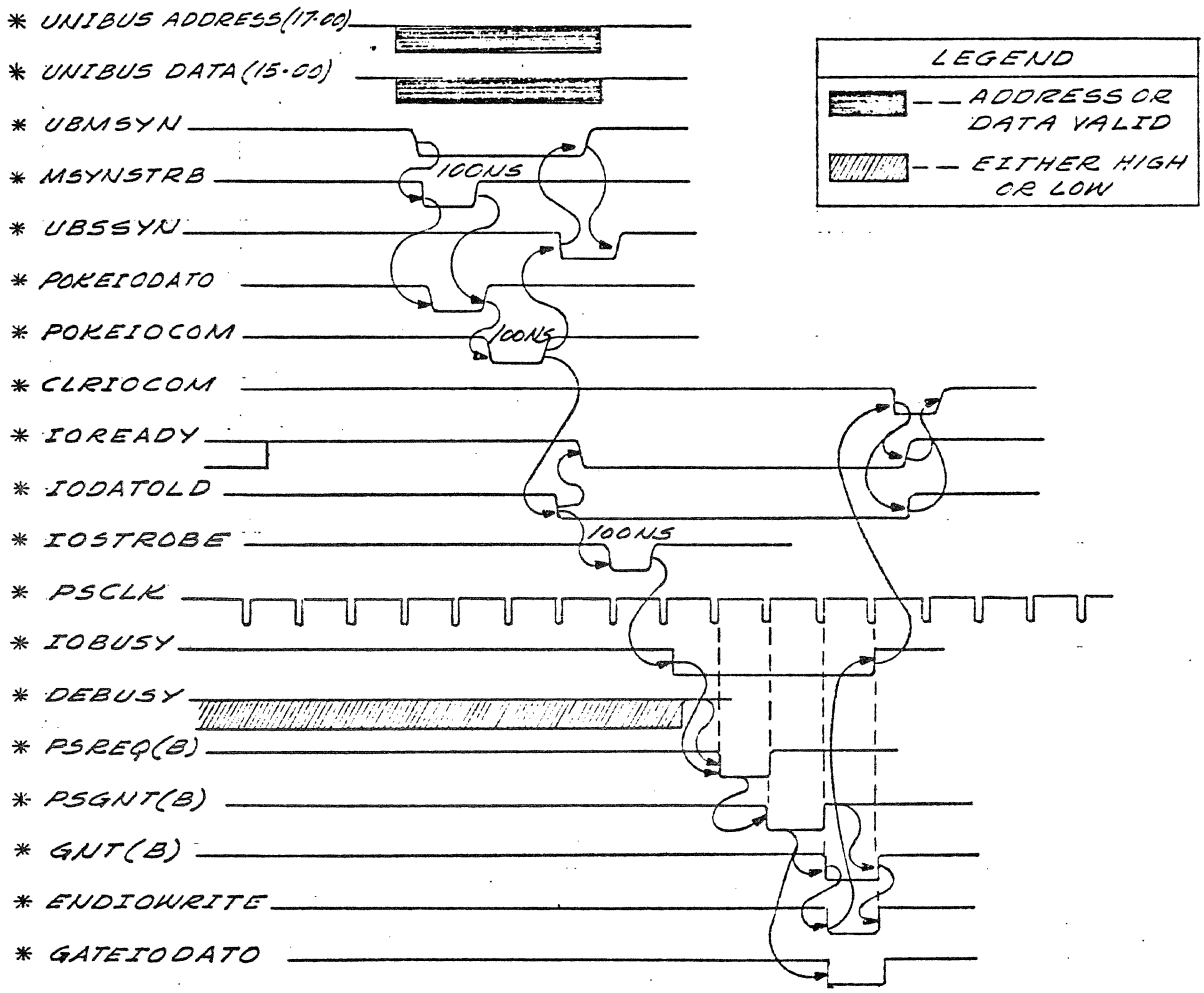


FIGURE 5.1-3 DIRECT IO WRITE TIMING DIAGRAM

5.1.2.2 Direct IO Write

To write a PS address pointed to by DIOPSA, the user simply writes into the PSDATA register; therefore, data is moved to the PSDATA register.

```
MOV DATA, PSDATA
```

Execution of the above instruction causes the following sequence of events to occur. This sequence is illustrated in the Direct IO Write Timing Diagram of Figure 5.1-3.

1. The contents of location DATA are fetched and put on the UNIBUS data lines.

2. The address of PSDATA is asserted on the UNIBUS address lines.

3. UNIBUS control lines C1 and C0 are set up for a full word write.

4. After the data, address, and control lines of the UNIBUS have settled, the UNIBUS master asserts *UBMSYN requesting the slave to respond.

5. The Address Select & Control logic on the PDP-11 Interface, 195131-600 sheet 3-4, received *UBMSYN which generates *MSYNSTRB pulse of 100ns duration.

6. *MSYNSTRB enables the address decode's output *POKEIODATO.

7. The trailing edge of *MSYNSTRB terminates *POKEIODATO which strobes the UNIBUS data into the CPU DATA OUT register on sheet 8

and generates *POKEIOCOM pulse of 100ns.

8. The trailing edge of *POKEIOCOM asserts *UBSSYN, which acknowledges the UNIBUS Master, and sets the IODATOLD (IO DATA OUT LOAD) command F/F on sheet 4.

9. IODATOLD clears the IOREADY F/F which indicates to the program that the Direct IO path is busy.

10. IODATOLD and the data in the CPU DATA OUT register drive the cable drivers (on sheet 10) to the PICTURE SYSTEM Direct IO card.

11. The PICTURE SYSTEM DIO controller receives the command and data lines on the 195105 card, sheet 1 and 2.

12. *IODATOLD generates the IOSTROBE pulse on sheet 3 which sets the IOBUSY F/F and strobes the incoming data into the DIO/DMA Data Out register, sheet 5.

13. IOBUSY sets the *PSREQ(B) to the PSBUS if the destination device (location) pointed to by the DIOPSA is not busy. (This device busy monitor is on sheet 7 of the drawings).

14. The PSBUS Arbiter acknowledges the bus request with PSGNT(B) when the DIO controller can have the bus.

15. The PSGNT(B) gates the DIOPSA (destination address) onto the PSBUS address lines (sheet 6)

and the next clock buffers the PSGNT(B) by setting the GNT(B) F/F (sheet 4) and also sets the *GATEIODATO F/F.

16. *GATEIODATO gates the DIO/DMA DATA OUT register (destination data) onto the PSBUS and GNT(B) asserts *ENDIOWRITE.

17. On the next clock, *ENDIOWRITE clears *IOBUSY indicating that the destination device has received the data.

18. *IOBUSY is monitored back on the PDP-11 Interface (195131) and sets the *CLRIOCOM F/F (sheet 4) which sets IOREADY and clears the command, IODATOLD F/F.

19. The Direct IO path is now ready for another DIO transfer.

5.1.2.3 Reading the DIOPSA

The DIOPSA is a write only register and cannot be read directly. Any attempt by the user to read the DIOPSA causes the contents of DIOPSA to be transferred to PSDATA register which is directly readable. The following code may be used to read the DIOPSA.

```
MOV DIOPSA, SAVE        ; Try to read the DIOPSA
TST IOST                ; Test the DIOREADY bit
                          in IOST
BPL .-4                 ; If not ready test again
MOV PSDATA, SAVE        ; Read contents of PSDATA
                          into UNIBUS address SAVE.
```

5.1.3 Direct Memory Access PATH (DMA)

The Direct Memory Access Path Interfaces the UNIBUS to the PICTURE SYSTEM to enable programmable DMA block transfers between UNIBUS address space and PICTURE SYSTEM address space. The actual hardware for the DMA path consists of:

1. DMA control and buffering on the PDP-11 Interface card (195131-100).

2. DMA control and "Data Out" buffering on the 195106-100 card in the PS backpanel.

3. "Data In' buffering on the 195105-100 card in the PS backpanel.

4. Cable driver and receiver lines between the PDP-11 Interface and the 195105-195106 cards.

5. Control lines on the PS backpanel between the 195105 and 195106 cards.

A DMA block transfer between UNIBUS address space and PS address space is set up and initiated under program control. The transfer may be programmed to take place in either direction by setting or clearing the DMAIN bit of the IO status register (IOST).

5.1.3.1 DMA Block Transfer Initialization

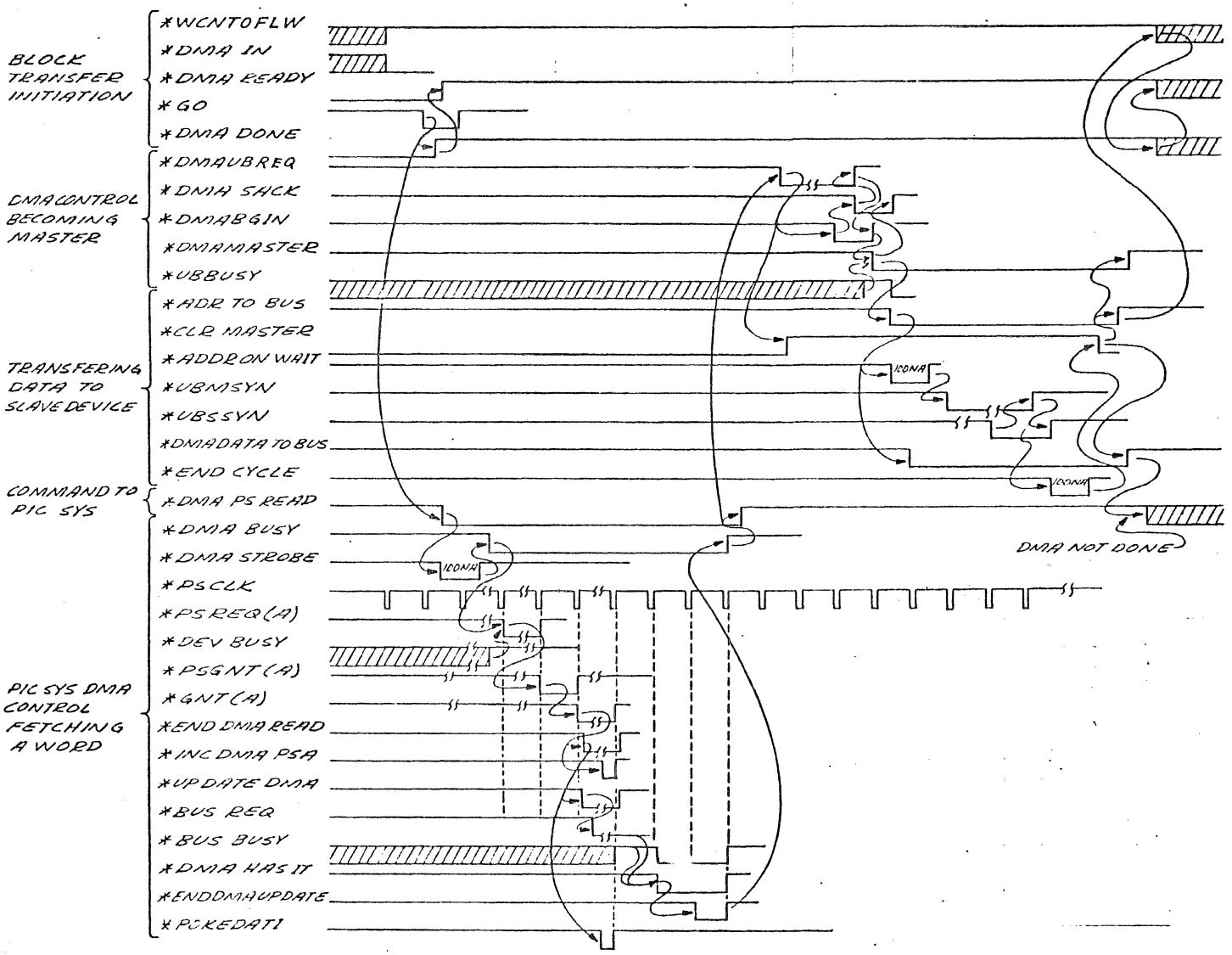
The program must interagate bit 7 of the IOST register

to see if the DMA is ready (see page 2-13 of PICTURE SYSTEM Reference Manual) before manipulating any of the DMA registers. Upon finding the DMAREADY, (195131-600, sheet 5) the program may set up a data block transfer between UNIBUS address space and PS address space. A negative two's complement number of words in the block to be transferred is written into the word count register, DMAWC (sheet 15). The starting UNIBUS address of the block to be sent (or where the block is to be received) is written into the bus address register, DMABA (sheet 15). The direction of the block transfer is set up by writing a 1 or a Ø in the DMAIN bit of the IOST (sheet 5). The block transfer is started by setting bit Ø of the IOST (GO bit, sheet 5).

5.1.3.2 DMAIN Transfer

A DMAIN block transfer reads a block of data (one word at a time) from PS address space and stores it in UNIBUS address space. Assuming the block transfer has been set up according to section 5.1.3.1, the block transfer is executed as a series of word transfers. The timing sequence of a DMAIN word transfer is illustrated in Figure 5.1-4. The sequence is described as follows:

1. The *GO pulse, generated by the program setting the GO bit in the IOST, clears *DMADONE which



DMAIN TIMING DIAGRAM

FIGURE 5.1-4

clears *DMAREADY indicating the DMA path is busy (195131-600, sheet 5). *GO also sets the *DMAPSREAD command F/F which drives the *DMADIRD (sheet 10) line to the PICTURE SYSTEM.

2. The PICTURE SYSTEM's DMA control receives *DMADIRD (195106-600, sheet 2) and generates *DMASTROBE pulse which sets the *DMABUSY F/F (sheet 3). *DMABUSY is monitored back on the PDP-11 Interface card.

3. *DMABUSY generates a request for the PSBUS, *PSREQ(A), in sync with the *PSCLK when the source device is not busy, indicated by *DEVBUSY (sheet 3).

4. The PS Arbiter eventually grants the request with *PSGNT(A) which gates the DMAPSA register (source address) and PSREAD signal onto the PSBUS (sheet 5).

5. On the next clock, *PSGNT is buffered by setting *GNT(A) F/F, which is set for one clock period and enables *ENDMAREAD. During the clock period when *GNT(A) is set, the source device enables the data onto the PSBUS data lines, and *ENDMAREAD enables *UPDATEDMA which generates a *BUSREQ (195105-600, sheet 4), requesting use of the PSDATAIN (15-0) lines to the PDP-11 Interface (see block diagram Figure 5.1-A).

6. The next clock pulse generates *INCDMAPSA, which bumps the DMAPSA register by one (if not inhibited), and generates *POKEDATI pulse which strobes the data off the PSBUS into

the Direct IO/DMA Data In register file (195106-600, sheet 6).

7. When *BUSBUSY (195105-600, sheet 4), indicates the data lines between the PICTURE SYSTEM and the PDP-11 Interface are not in use, *DMAHASIT is set in sync with the clock. *BUSBUSY is asserted indicating the data lines are now in use by the DMA.

8. *DMAHASIT stays asserted for two clock periods and gates the DMA source data from the input register file onto the data lines to the PDP-11 Interface. After the first clock period, *ENDMAUPDATE is asserted and the next clock pulse clears the *DMABUSY F/F (195106-600, sheet 3).

9. The PDP-11 Interface has been monitoring the *DMABUSY F/F in the PICTURE SYSTEM. The trailing edge of *DMABUSY clears the read command F/F, *DMAPSREAD (195131-600, sheet 5) and the data from the PICTURE SYSTEM is strobed into the DMA DATA IN register (sheet 14). The DMA control generates a request for the UNIBUS, *DMAUREQ (sheet 6).

10. The UNIBUS eventually responds with a grant, *DMABGIN, which generates *DMASACK acknowledging the grant and clearing the request.

11. The cleared request causes the UNIBUS to clear the grant which sets the *DMAMASTER F/F when the UNIBUS becomes "not busy".

12. *DMAMASTER clears *DMASACK and asserts *UBBUSY indicating the DMA is master of the UNIBUS. *DMAMASTER gates the contents of the DMABA (destination address) onto the UNIBUS by asserting *ADRTOBUS (sheet 13).

13. *ADRTOBUS asserts *DMADATATO BUS, which gates the data in the DMA DATA IN register onto the UNIBUS, and generates *ADDRONWAIT pulse (sheet 7). The trailing edge of this pulse sets the MSYNOUT F/F which asserts *UBMSYN to the UNIBUS.

14. The destination device on the UNIBUS eventually responds with *UBSSYN, indicating it has taken the data. *UBSSYN clears *UBMSYN which generates *ENDCYCLE pulse indicating the completion of a UNIBUS cycle.

15. The trailing edge of *ENDCYCLE sets the *CLRDMAMASTER F/F (sheet 7) which clears the *DMAMASTER F/F and unasserts *ADRTOBUS and *DMADATAOBUS.

16. The rising edge of *ADRTOBUS increments the DMAWC (word count register, sheet 15). If the word count overflows (*WCNTOFLW), the *DMADONE F/F is set which asserts *DMAREADY, indicating the DMA path is ready for another block transfer. The rising edge of *DMADATATOBUS issues another command to the PICTURE SYSTEM (*DMAPSREAD) if the block transfer is not complete, which commands another word to be fetched from the PICTURE SYSTEM.

5.1.3.3 DMAOUT Transfer

A DMAOUT transfer reads a block of data (one word at a time) from UNIBUS address space and stores it in PS address space. Assuming the block transfer has been set up according to section 5.1.3.1, the block transfer is executed as a series of word transfers. The timing sequence of a DMAOUT word transfer is illustrated in Figure 5.1-5. The sequence is described as follows:

1. The *GO pulse clears *DMADONE which clears *DMAREADY, indicating to the program that the DMA path is busy. *GO also sets the *DMAUBREQ F/F (195131-600, sheet 6).

2. *DMAUBREQ requests mastership of the UNIBUS and is eventually granted with *DMABGIN which sets *DMASACK acknowledging the grant.

3. *DMASACK clears the request and eventually the UNIBUS grant is cleared which sets the *DMAMASTER F/F when UNIBUS mastership is released by the current master.

4. *DMAMASTER asserts *ADRONBUS which gates the DMABA (source address) onto the UNIBUS and generates *ADDRONWAIT pulse (sheet 7).

5. The trailing edge of *ADDRONWAIT sets MSYNOUT F/F which asserts *UBMSYN to the UNIBUS.

6. The slave device (usually memory) eventually responds with data on the UNIBUS by asserting *UBSSYN which generates *POKEDMADATA pulse and clears *UBMSYN.

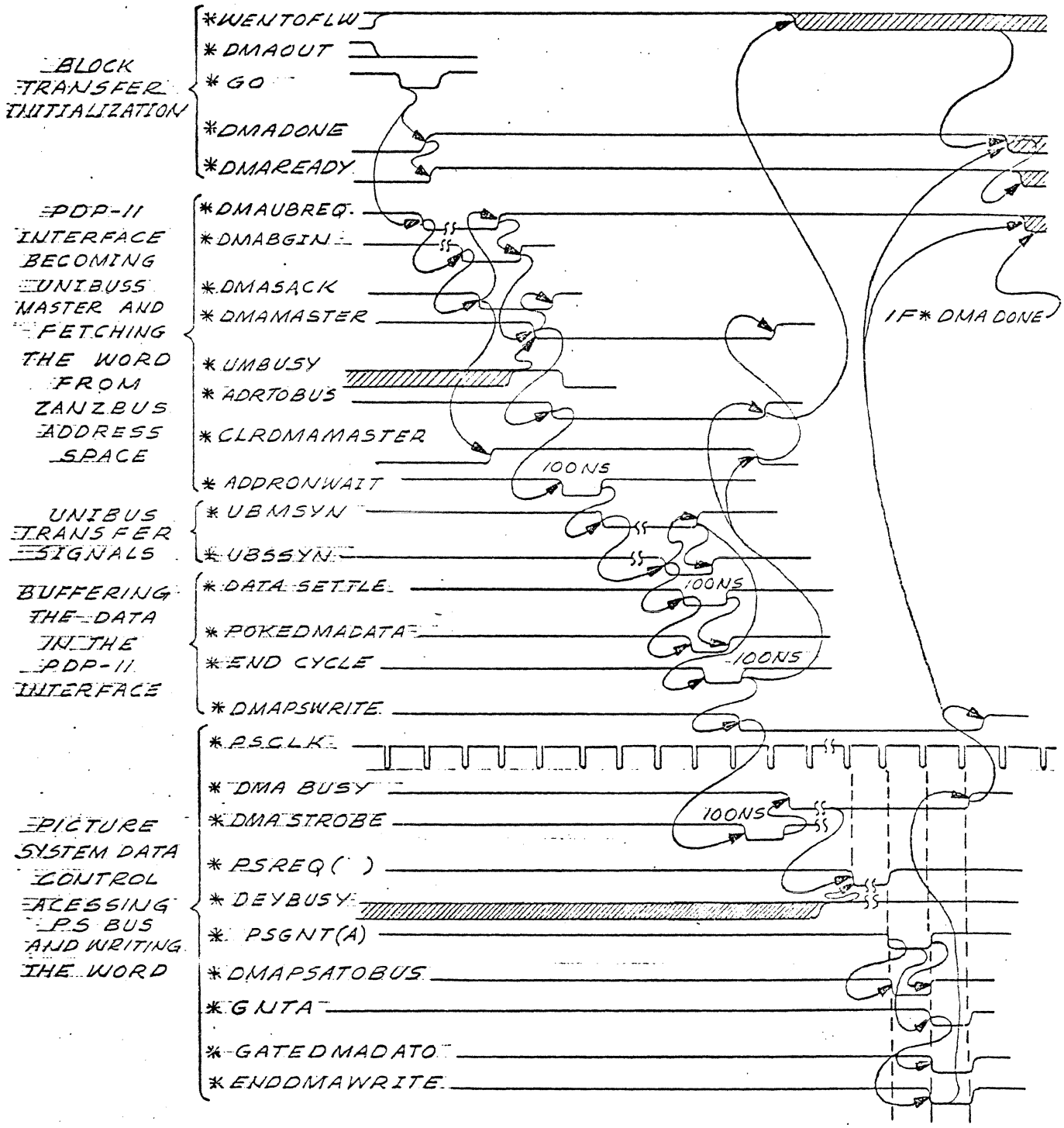


FIGURE 5.1-5 DMAOUT TIMING DIAGRAM

7. The trailing edge of *POKEDMADATA strobes the data off the UNIBUS into the CPUDATAOUT register (sheet 8) and the trailing edge of *UBMSYN generates *ENDCYCLE pulse indicating the completion of a UNIBUS cycle.

8. The trailing edge of *ENDCYCLE sets *CLRDMAMASTER which clears *ADRTOBUS and *DMAMASTER.

9. The trailing edge of *ADRTOBUS increments the DMAWC (word count) and DMABA (source address) and sets *DMAPSWRITE (sheet 5) which is the write command to the PICTURE SYSTEM. *DMAPSWRITE drives the DMADOLD line and CPUDATAOUT drives the data lines to the PICTURE SYSTEM.

10. *DMAPSWRITE and the data is received (sheet 1 and 2, 195106-600) and generates *DMASTROBE pulse which sets *DMABUSY and strobes the data into the Direct IO/DMA data out register file (195105-600, sheet 5).

11. When the destination device is not busy (*DEVBUSY), a request for the PSBUS is generated in sync with *PSCLK.

12. The request is eventually granted with *PSGNT(A) which gates the DMAPSA (destination address, 195106-100, sheet 5) onto the PSBUS.

13. The next clock buffers the grant by setting *GNTA which gates the data from the DIO/DMA register file to the PSBUS and asserts *ENDDMAWRITE.

14. The next clock clears *DMABUSY which is the signal the PDP-11 Interface is monitoring.

15. The rising edge of *DMABUSY clears the write command (*DMAPSWRITE) and sets *DMAUBREQ to request another UNBIUS cycle if the block transfer is not done. If *WCNTOFLOW (word count overflow) is asserted, then *DMADONE is set which asserts *DMAREADY indicating the DMA path is ready for another block transfer (195131-600, sheet 4).

5.1.4 Interrupt Sense and Control

The PICTURE SYSTEM is equipped with an intrerupt facility to enable program intervention by a requesting device. The possibility of more than one request for service at one time requires an arbitrator to decide which device is serviced first. A requesting device issues an interrupt request to the Interrupt Sense and Control which causes the following sequence of events:

1. The sense logic (195131-600, sheet 11) examines all interrupt request lines.

2. When one or more interrupt request are present, the Interrupt Control requests use of the UNIBUS (sheet 12).

3. Eventually, the UNIBUS is given to the Interrupt Control and it becomes bus master.

4. The Interrupt Control recognizes the interrupt request with the highest priority (sheet 11, bug 92), passes the appropriate interrupt vector address to the UNIBUS, and asserts the interrupt line, *UBINTR, on the UNIBUS (sheet 12).

5. The PDP-11 Processor pushes the contents of the Processor Status Word and Program Counter (PSW,PC) onto the program stack and loads into the PC and PSW the first two consecutive words (interrupt vector) from memory pointed to by the interrupt vector address.

6. The Processor does a jump to the location in the PC and begins executing the interrupt service routine.

7. The last instruction of the routine is a RTI, return from interrupt, instruction. When executed, the RTI causes the Processor to pop two words from the program stack into the PC and PSW registers.

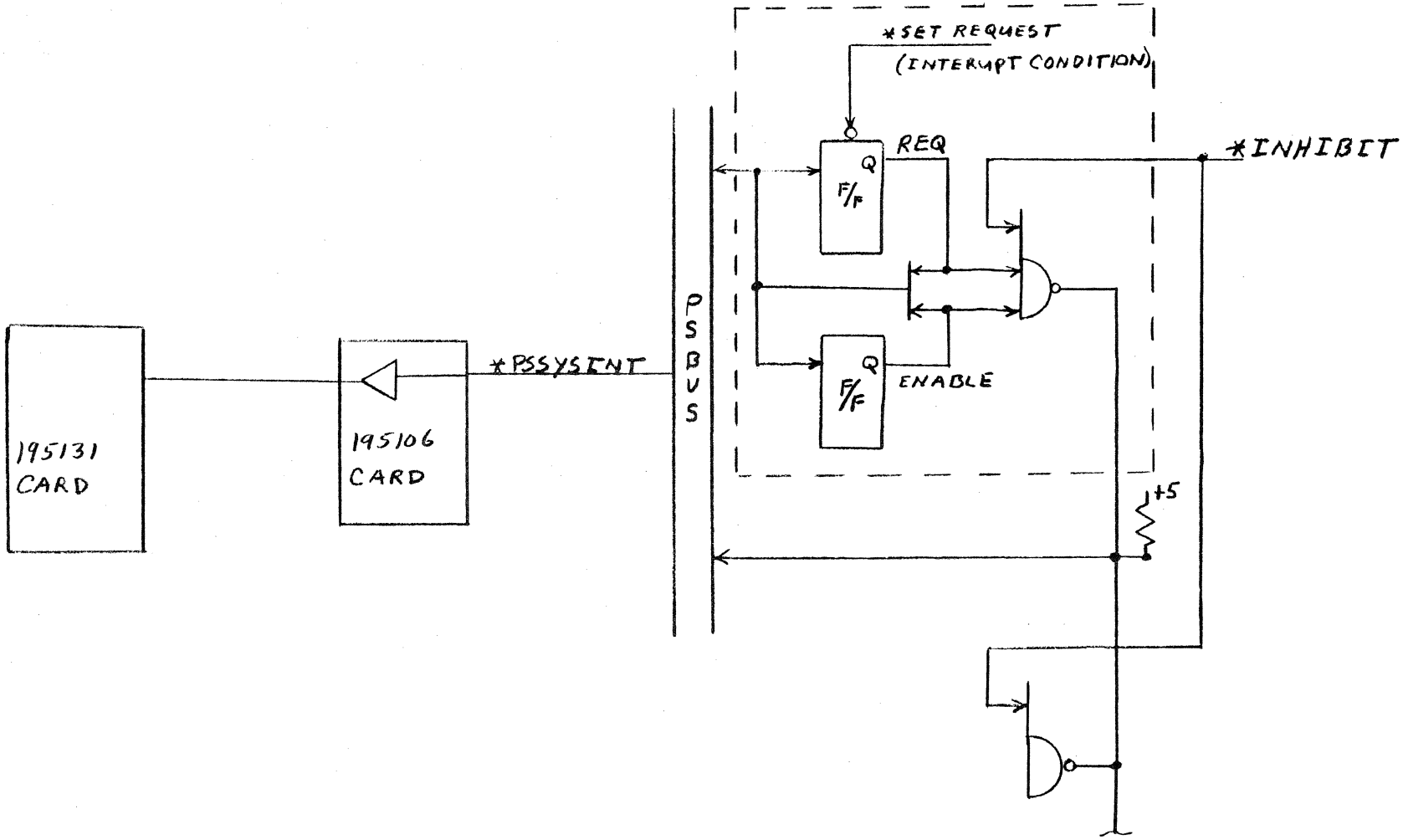
8. The Processor jumps to the new PC location and resumes execution of the interrupted task.

5.1.4.1 System Interrupts

Any one of the following seven conditions will generate a system interrupt if the corresponding interrupt enable F/F is set.

1. MATCH REQ
2. WBSTOP REQ
3. RFSTOP REQ
4. MOSTOP REQ
5. JUMP REQ
6. HIT REQ
7. HALT REQ

These seven conditions are defined in the PS2 Reference Manual, pages 2-134 through 2-136. A circuit similar to the one surrounded by dashed lines in figure 5.1-6 exists for each system interrupt condition. *PSSYSINT is the system interrupt line of the PSBUS driven by each system interrupt circuit. This line is relayed to the Interrupt Controller on the 195131 card by a differential driver on the 195106 card. A high to low transistion of this line initiates action by the Interrupt Controller which will gain access of the UNIBUS and interrupt the Picture Controller (provided the PSIE bit of the IOST register is set). In order for a system interrupt circuit to assert *PSSYSINT, both the interrupt enable F/F



SYSTEM INTERUPT SCHEME

FIGURE 5.1-6

and the interrupt request F/F must be set. *INHIBIT is a common line to all system interrupt circuits which inhibits any pending interrupt requests from asserting *PSSYSINT while the program is clearing the particular request bit which generated the last interrupt. After the request bit is cleared, any other system interrupt request bit still set will generate another high to low transition of *PSSYSINT; therefore, generating another system interrupt.

The interrupt circuits are found on the 195121-100 and 195151-100 cards.

5.2 PICTURE SYSTEM BUS - PSBUS

All PICTURE SYSTEM 2 devices connect and interact with each other on a single, high-speed, synchronous data bus. Memory, device registers, control and status registers all exist and are addressable memory locations on the PSBUS. By means of this PSBUS, coordinate data may be transferred from the Picture Controller host computer to the Picture Processor while data may concurrently be transferred to PS Memory. During the process, data may also be transferred from PS Memory to the Picture Generator for display. In addition, data may be entered from the data tablet, alphanumeric keyboard, function switches, etc., and read by the control program of the host computer. Data flow is supervised by a bus arbitration system which is an integral part of the PSBUS.

5.2.1 PSBUS Structure

The PSBUS consists of the following lines:

1. *PSDAT (15-0) - 16 data lines
2. PSADD (15-0) - 16 address lines
3. *PSREQ (7-0) - 8 request lines
4. *PSGNT (7-0) - 8 grant lines
5. *PSCLK (5-1) - 5 clock lines
6. *PSDEF - 1 memory defer line
7. *PSMEM - 1 memory busy line (FIFO Line)

8.	*PSRST	- 1 reset line
9.	*PSREAD/PSWRT	- 2 memory command lines
10.	*PSDEVINT/*PSSYSINT/ *PSCLKINT	- 3 interrupt lines
11.	*PSBSY (7-0)	- 8 device busy lines
12.	*PSSYNC	- 1 sync line

TOTLA 70 lines

The five clock lines are all in sync and were implemented for driving capability. The *PSSYNC line is used to arbitrate use of the Refresh Memory by the Refresh Contoller and possible future console keyboard devices.

5.2.2 Active/Passive Devices

Data transfers from device to device in the PICTURE SYSTEM are performed by an active device transferring data to or from a passive device via the PSBUS. The active device initiates and controls the transfer, and the passive device accepts or provides the data as commanded.

All devices are either active or passive; however, some devices may be programmed to be either active or passive at a given time. No device can be active and passive simultaneously.

A typical data transfer between active and passive devices involves the following sequence of events if the passive

device is not PS Memory:

1. The active device requests and is eventually granted use of the PSBUS.

2. When granted, the active device gates the address of the passive device and a read/write command onto the PSBUS.

3. The passive device is given one bus clock period (150 ns) to decode the address and command.

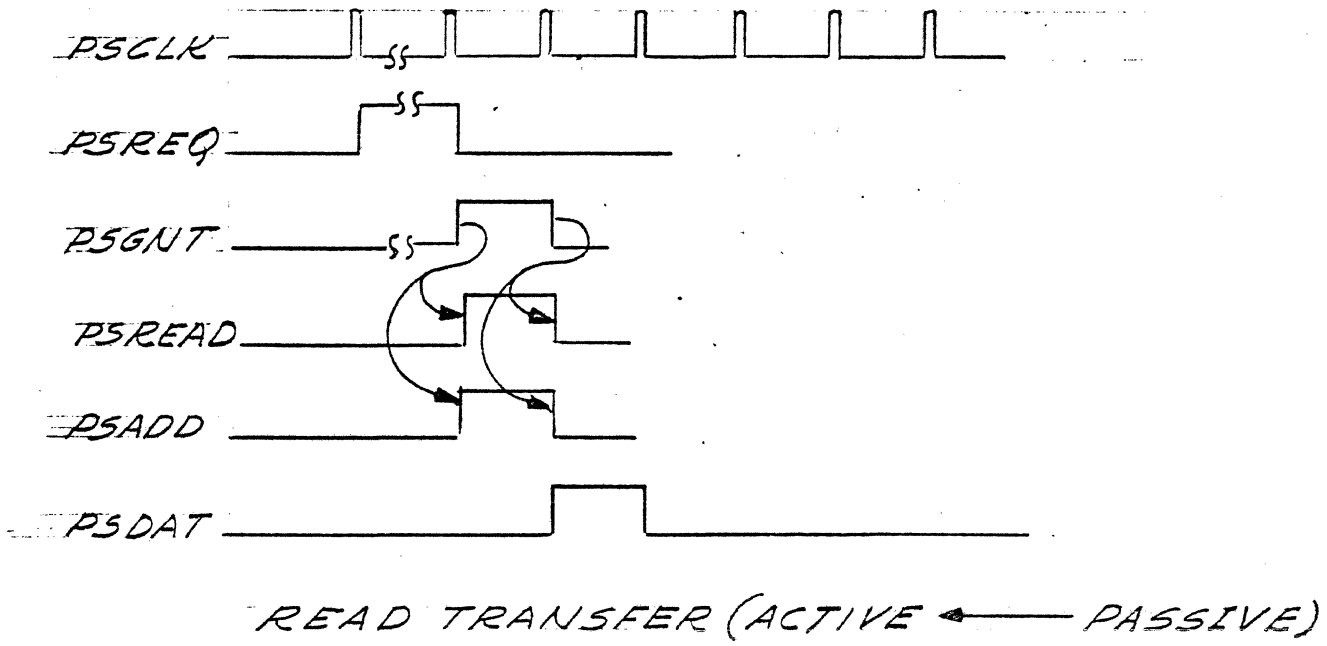
4. If the transfer is a write (active to passive), the active device gates data to the PSBUS. If the transfer is a read, the passive device gates data to the PSBUS. This data is held for one clock period.

5. The data on the PSBUS is taken by either the active or passive device depending on the direction of the transfer.

5.2.3 PSBUS Timing

A timing diagram of typical non memory data transfers between active and passive devices as described in section 5.2.2 is illustrated in Figure 5.2-1. A special case exists when the passive device is the PICTURE SYSTEM Memory.

Any data transfer to or from PICTURE SYSTEM Memory requires



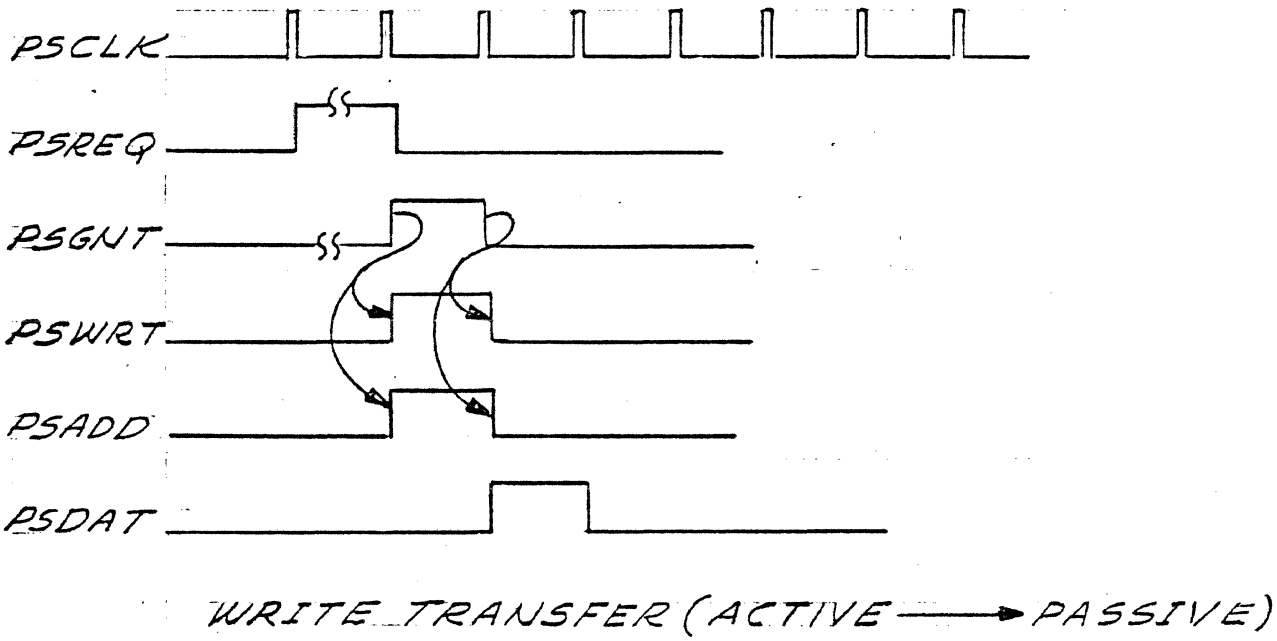


FIG. 5.2-1 TIMING DIAGRAM FOR TYPICAL ACTIVE PASSIVE DATA TRANSFER

an active device initiating the transfer to or from the passive memory. More than one active devices may request concurrent use of the memory. The PSBUS Arbiter arbitrates use of the PSBUS, however, a memory read cycle is 450 ns and a read-write cycle is 750 ns; therefore, data from memory cannot be gated to the PSBUS during the clock after the memory receives the address. Also, the memory may be busy when addressed by an active device.

The Bus Arbiter always responds with a grant to a requesting (active) device when the PSBUS is not busy; however; if the memory access is a read, the memory controller simultaneously responds to the active device with a deferred signal (*PSDEF). The deferred signal instructs the active device to wait till later for the data. If the memory is busy, it asserts the busy line (*PSMEM) the same time it defers the active device. The device then waits for a second grant to give the memory the address, is again deferred and waits for a third grant which signifies data is valid on the next bus cycle. Typical data transfers between an active device and PICTURE SYSTEM Memory are illustrated by the timing diagrams in Figure 5.2-2.

5.2.4 PSBUS Arbitration

An Arbitration system for the PSBUS manages data

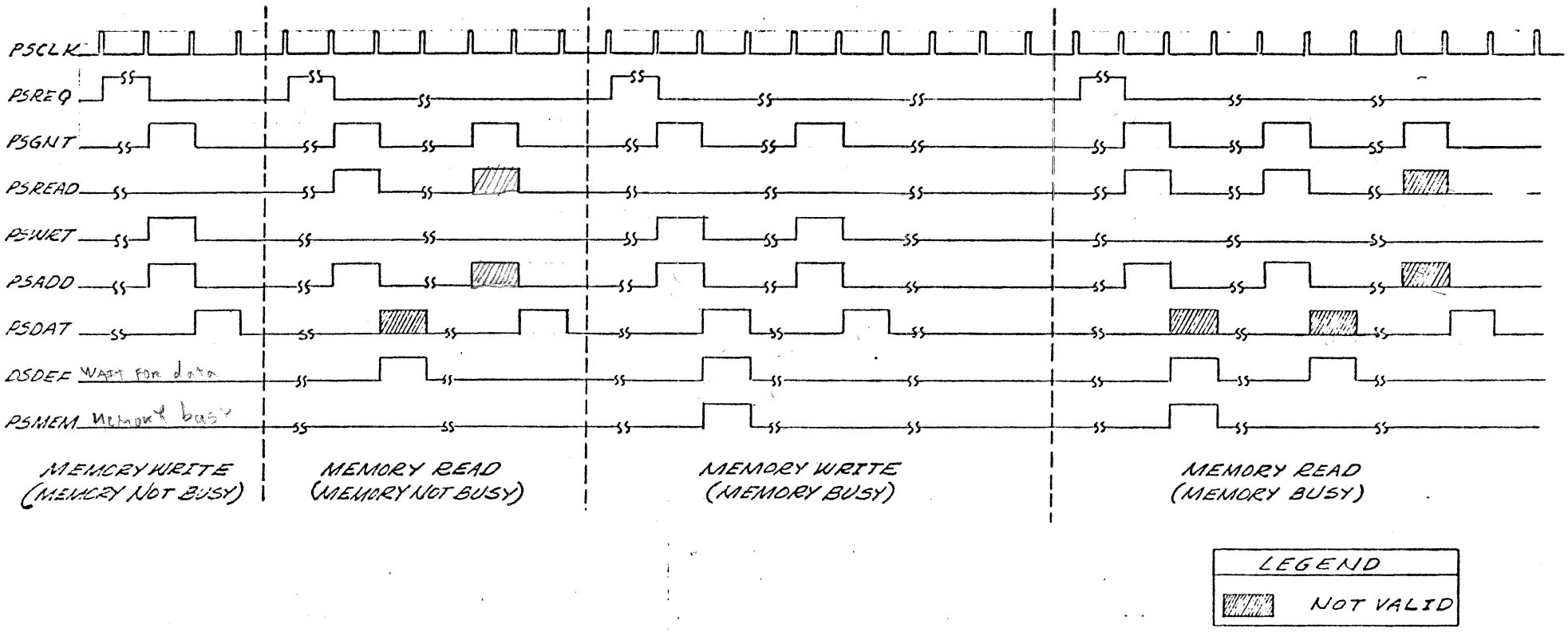


FIGURE 5.2-2 MEMORY DATA TRANSFER TIMING DIAGRAM

transfers between devices in such a way as to utilize the full bandwidth of the bus. Since the PSBUS is synchronous, a data transfer may take place every clock period if active devices are constantly requesting use of the bus; therefore, the bus can transfer 6.6 meg-a-words per second. The PS memory operates at a slower rate of 2.5 meg-a-words per second; therefore, a device accessing memory must wait for data; however, the device releases control of the bus to other devices while waiting. The arbitration system must recognize when memory is ready to output data, then give memory highest priority for use of the bus. This implies that memory has higher priority than the eight active request priority levels, *PSREQ(7-0). Memory is always granted use of the bus the next bus cycle after requesting the bus.

The arbiter actually rotates priority of the eight active request levels. When one request is granted, that request level is changed to lowest priority, and the other seven levels are rotated appropriately for the next grant cycle; however, remember memory can override the highest requesting priority level if memory is in a data output state.

The PSBUS arbitration system receives eight active

requests from PICTURE SYSTEM devices. These requests are input into a priority rotater as illustrated in Figure 5.2-3. The rotator examines WHO, the last request level who was granted, and rotates the eight requests input to the priority encoder. The request corresponding to WHO is input to the encoder at the lowest priority level. The encoder outputs a 3 bit code corresponding to the highest level request on its inputs to the NEWWHO adder. The adder sums the result of the priority encoder with the number of places the encoder inputs are rotated to determine which Active Request is to be honored. This honored request, NEWWHO, waits for one of two events to occur before being saved in the WHO latch. The two events are:

1. A valid WHO does not exist in the WHO latch; therefore clock the latch until a valid NEWWHO is saved.

2. A valid WHO exists in the latch, a memory port does not need the PSBUS (not in data output state), and a free memory port does not need the FIFO.

Either event gates the decoded WHO out to the PSBUS as a grant, and on the next clock latches NEWWHO into the WHO latch which determines the rotation for the

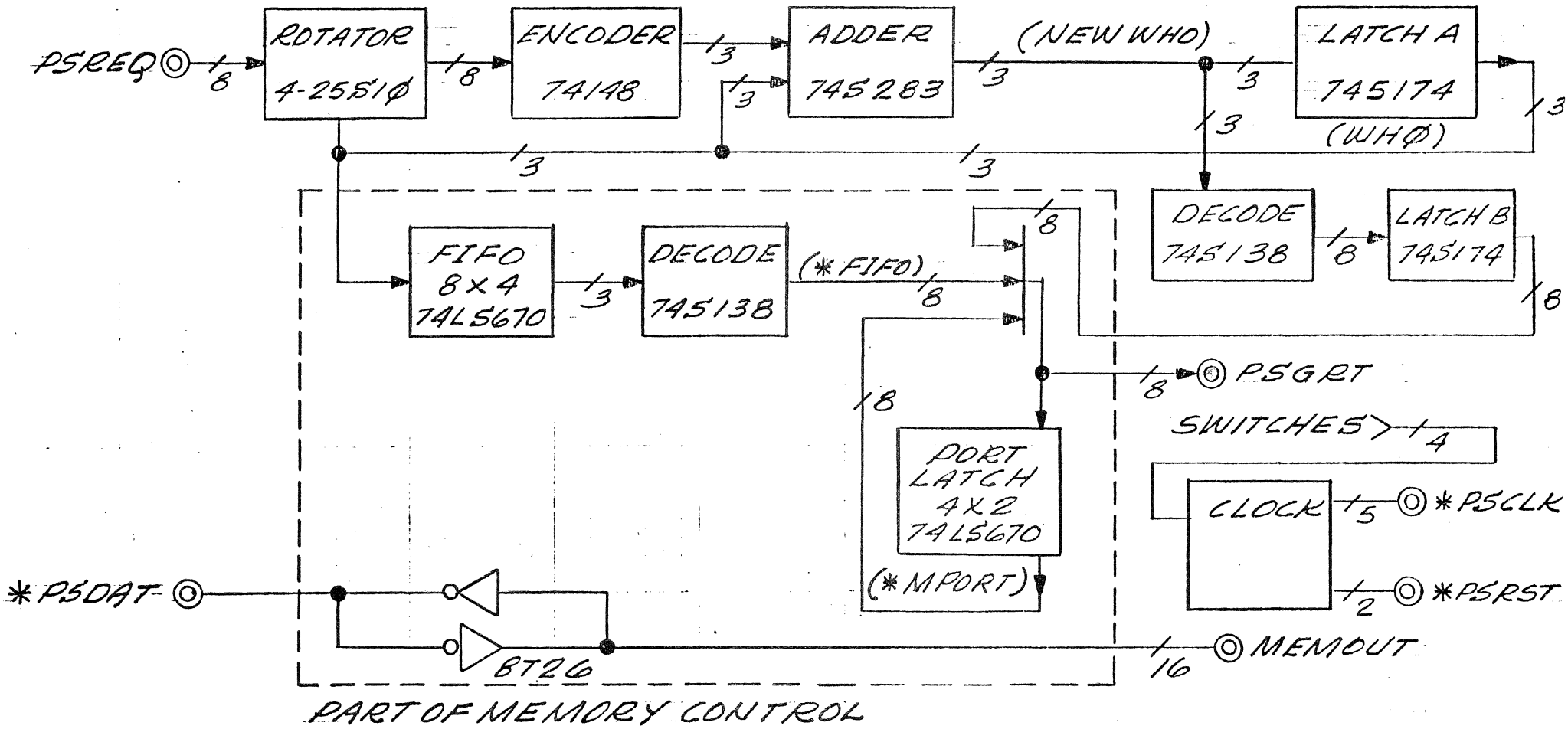


FIGURE 5.2-3
PSBUS ARBITRATION SYSTEM BLOCK DIAGRAM

next clock.

Further examination of event 2 leads to a clearer understanding of the request/grant timing on the PSBUS. Suppose that a valid WHO exists in the WHO latch, the memory FIFO contains a pending grant, and both memory port controllers are busy; however, neither port controller is in a data output state. This means event 2 is true and the PSBUS is free for a grant cycle. The decoded WHO which corresponds to the original active request that generated WHO, is gated out to the PSBUS as a grant. This grant is received by the active device which in turns gates the address and a read or write command to the passive device as depicted in the timing diagrams of Figure 5.2-2. The WHO latch is updated on the next clock, and if the address on the bus was for memory and the access is a read, the PS Memory Control asserts the deferred line which tells the active device that data will not be valid during this clock period. (If passive device is not memory data will be valid). Also, if the memory is busy, the PS Memory Control asserts *PSMEM which tells the active device that the address was not accepted by the memory and will be asked for at a later time. If the active device is writing to memory, the device asserts data to the PSBUS during

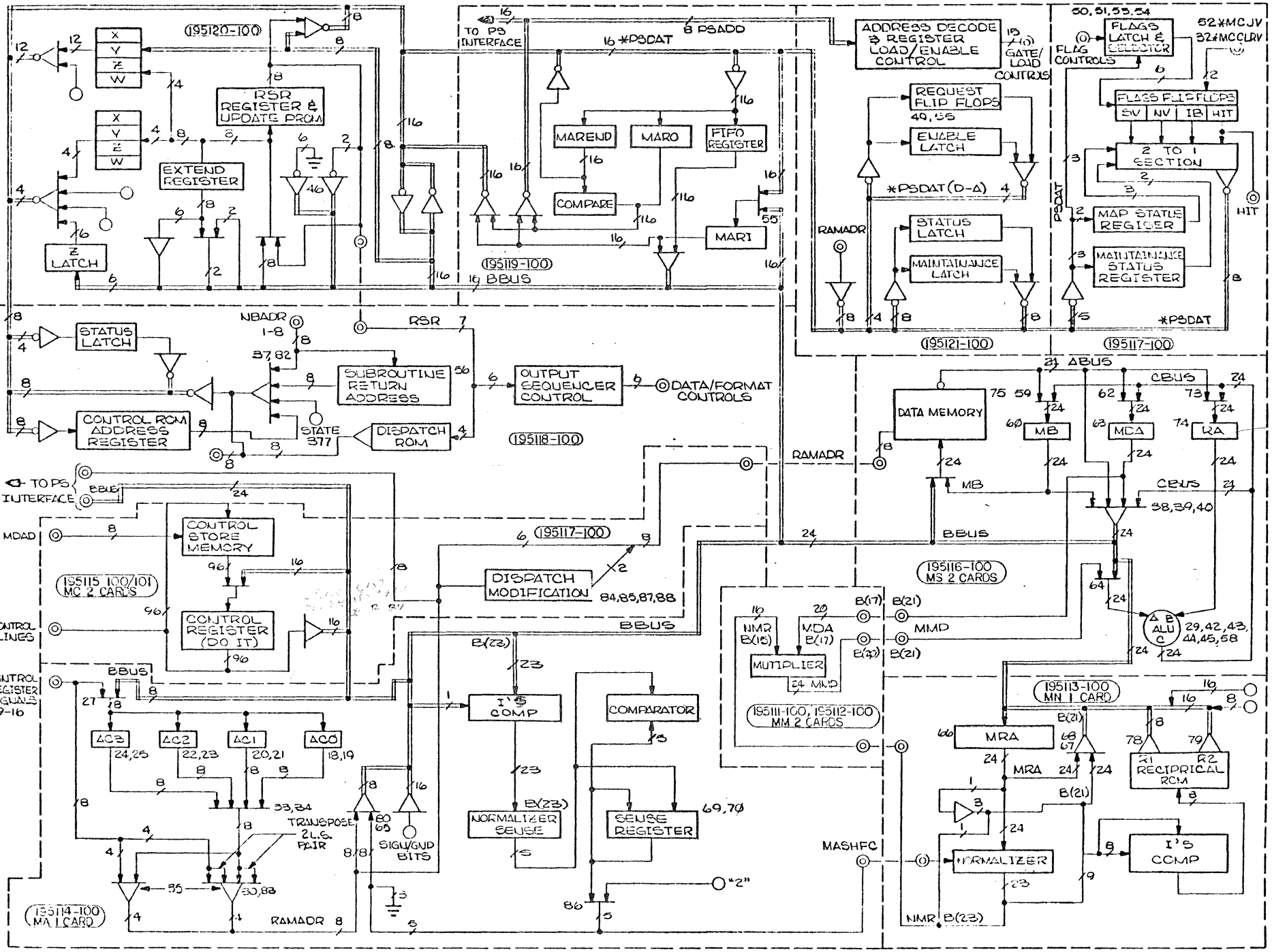


FIGURE 5.3-2 PS2 MAP BLOCK DIAGRAM

the clock period after memory latches the address.

If an active device receives the memory busy signal the device will receive a second grant at a later time from the memory. This implies that the memory must save the first grant. It does so in the memory FIFO, and if the FIFO is not empty when a memory port controller becomes idle, the FIFO is output to the grant lines as the second grant to a waiting active device. This second grant signals the device to repeat the address to the bus sense a free port controller can now begin access of the address. The PS Memory Control is also equipped with a port latch. This latch saves the particular grant that signals a device of the beginning of a read access by a port controller. When the port controller has data from memory (in data output state) it repeats the grant from the port latch which signals the device that data will be on the PSBUS during next clock cycle.

5.3 Picture Processor

The Picture Processor receives commands and data from the PSBUS, decodes the commands, operates on the data, and outputs data in a specified format to the PSBUS. Output from the Picture Processor may be directed to PS Memory, the Picture Generator, or the Picture Controller. The source of Picture Processor input and destinations of Picture Processor output is program controlled.

The Picture Processor consists of three units:

1. MAP Input Controller
2. Matrix Arithmetic Processor (MAP)
3. MAP Output Formatter

The three units interface to an internal bus, the BBUS. Data transfers between the three units occur on this bus; however, input and output occur on the PSBUS. Figure 5.3-1 illustrates a simplified block diagram of the Picture Processor interfaced to the PSBUS. A detailed block diagram is illustrated in Figure 5.3-2.

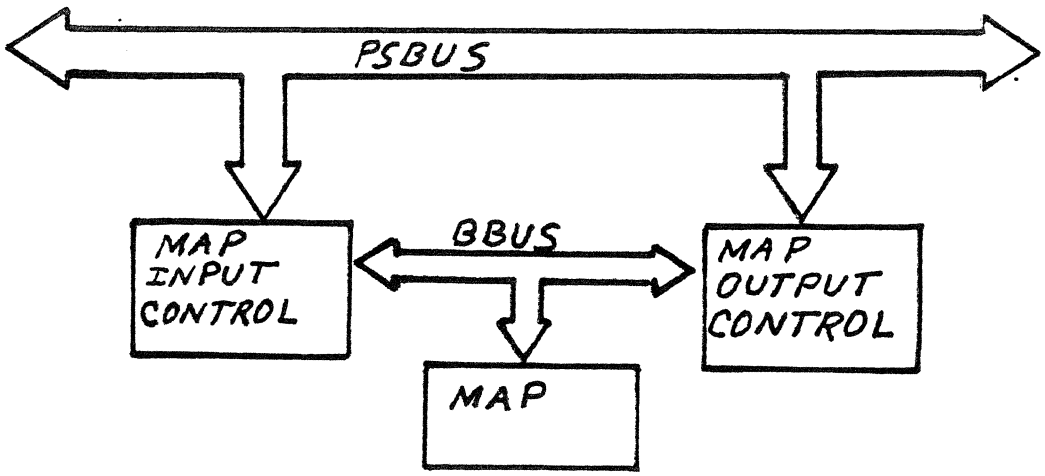


Figure 5.3-1
Simplified Block Diagram of Picture Processor

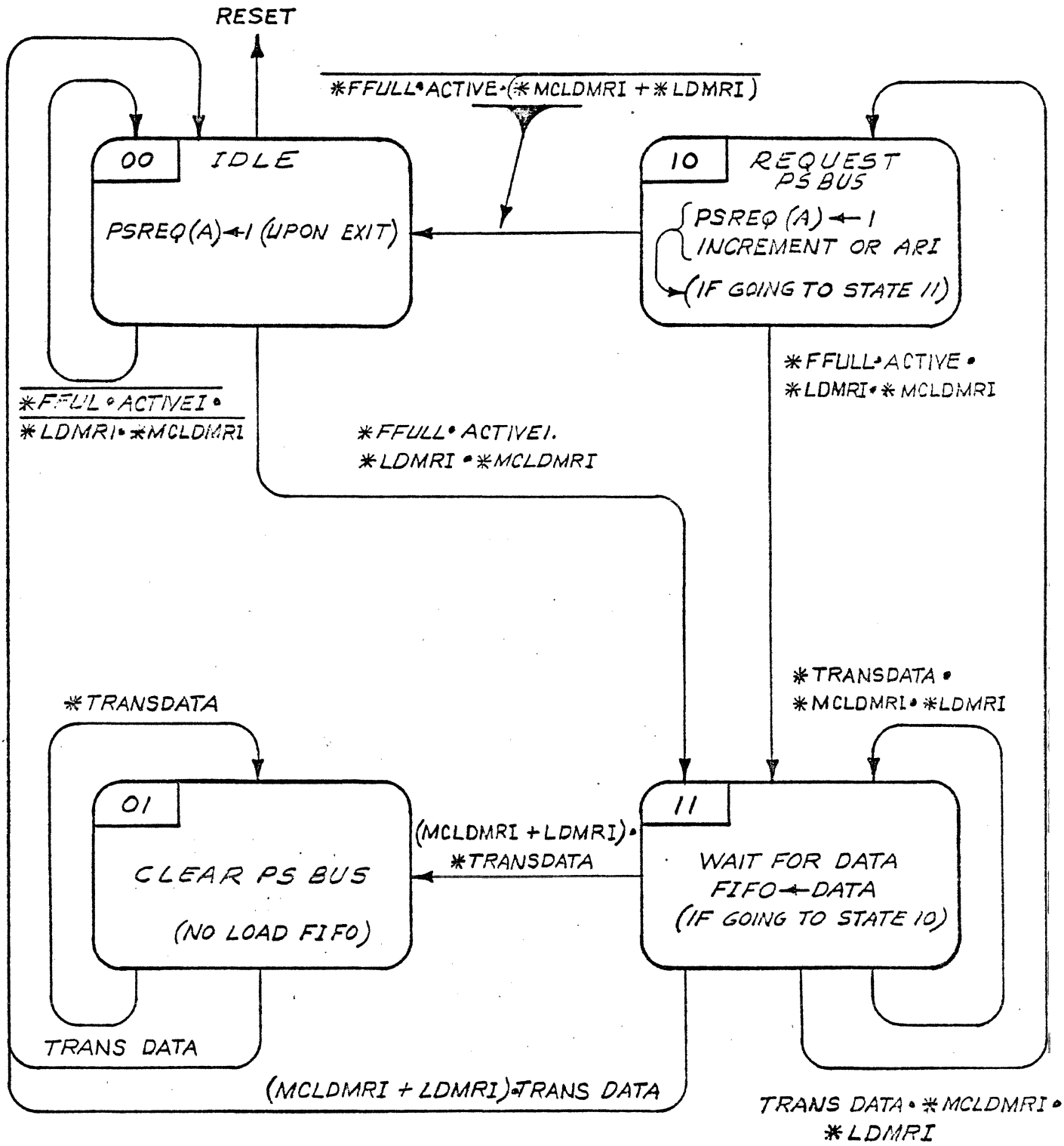


FIGURE 5.3-3

MAP INPUT STATE DIAGRAM

The Picture Processor contains eight, PSBUS, addressable, control registers which control status, input, and output. These eight registers are:

1. MAOL - MAP Active Output Limit
2. MAOA - MAP Active Output Address
3. MAIA - MAP Active Input Address
4. MSR - MAP Status Register
5. MMSR - MAP Maintenance Status Register
6. MMRSR- MAP Maintenance Repeat Status Register
7. MMPAR- MAP Maintenance Prom Address Register
8. MMBUS- MAP Maintenance BBUS

The eight control registers are defined in Section 2.3 of the PS2 Reference Manual.

In addition to the eight control registers, the Picture Processor contains two additional PSBUS-addressable port registers. These registers act as input and output ports when input or output is functioning as a passive PSBUS device. The two passive port registers are:

1. MPIP - MAP Passive Input Port
2. MPOP - MAP Passive Output Port

The MPIP and MPOP are described in the PS2 Reference Manual on pages 2-28, 2-55 and 2-56.

The Picture Processor also includes 256, non-PSBUS-addressable, internal registers. These registers are

implemented as RAM Memory. They are used as parameters and working registers by the MAP during point vector transformation functions. They may be loaded from the PSBUS during MAP execution of LOAD, or LOAD STACK commands and may be output to the PSBUS during MAP execution of STORE or STORE STACK commands. These internal registers are described in the PS2 Reference Manual on pages 2-30 through 2-35.

The heart of the Picture Processor is the Matrix Arithmetic Processor (MAP). This processor is used to perform point-vector transformations, matrix concatenations, clipping, viewpoint mapping, perspective calculations, hit testing, etc. The MAP receives commands from the MAP Input Controller, operates on data as specified by the command, and outputs data to the MAP Output Formatter.

The MAP Output Formatter receives data from the MAP and formats it as specified by MSR bits 9 and 8 (See PS2 Reference Manual, page 22) in preperation for output to a PSBUS device.

5.3.1 MAP Input Controller - Active

The MAP Input Controller is the device by which the Picture Processor inputs data from the PSBUS. It can be programmed to be either an active device or a passive device by modifications of bit 4 of the MSR. When operating

as an active device, the controller inputs data from the PS address specified by the MAIA register. The MAIA is incremented upon each PS address access; therefore, commands and data from a PSBUS-addressable file will stream into the MAP Input FIFO.

5.3.1.1 MAP Input FIFO

The MAP Input FIFO consists of a four word register file which is simultaneously writeable and readable. The FIFO is monitored for full or empty status by the MSR; therefore, the program may sense the conditions of FIFO by testing bits 14 and 13 of the MSR. When the Input FIFO is full, further access to PS address locations by the Input Controller is inhibited. When the FIFO is empty further access to the FIFO by the MAP is inhibited.

The FIFO is implemented on sheet 2 of the 195119-600 drawings of the MAP BUFFER Card. Note the separate read and write address inputs.

5.3.1.2 MAP Input Sequencer

When operating as an active device, the MAP input is controlled by the MAP Input Sequencer. The sequencer is a four state machine which implements the following five tasks:

1. requests use of the PSBUS
2. passes the address specified by the MAIA to the PSBUS
3. loads data from the PSBUS into the MAP Input FIFO
4. controls automatic incrementation of the MAIA register
5. allows a current PSBUS cycle to clear if the program or the MAP changes the contents of the MAIA

The state diagram for the sequencer is illustrated in Figure 5.3-3. The four states are:

State 00 - IDLE
State 01 - PSBUS Clear
State 10 - Request PSBUS
State 11 - Wait for Data

State 00 is the initial state of the sequencer. When the MAP Input Controller is passive, this state is maintained; however, when functioning as an active device, exit from the "IDLE" state sets a request for the PSBUS. The dispatch from the "IDLE" state to the "Wait for Data" state occurs if the FIFO is not full, assuming the MAIA is <u>not</u> to be loaded. The MAIA may be loaded with a new address by command from the MAP or by command from the user program.

The "Wait for Data" state is maintained until the passive

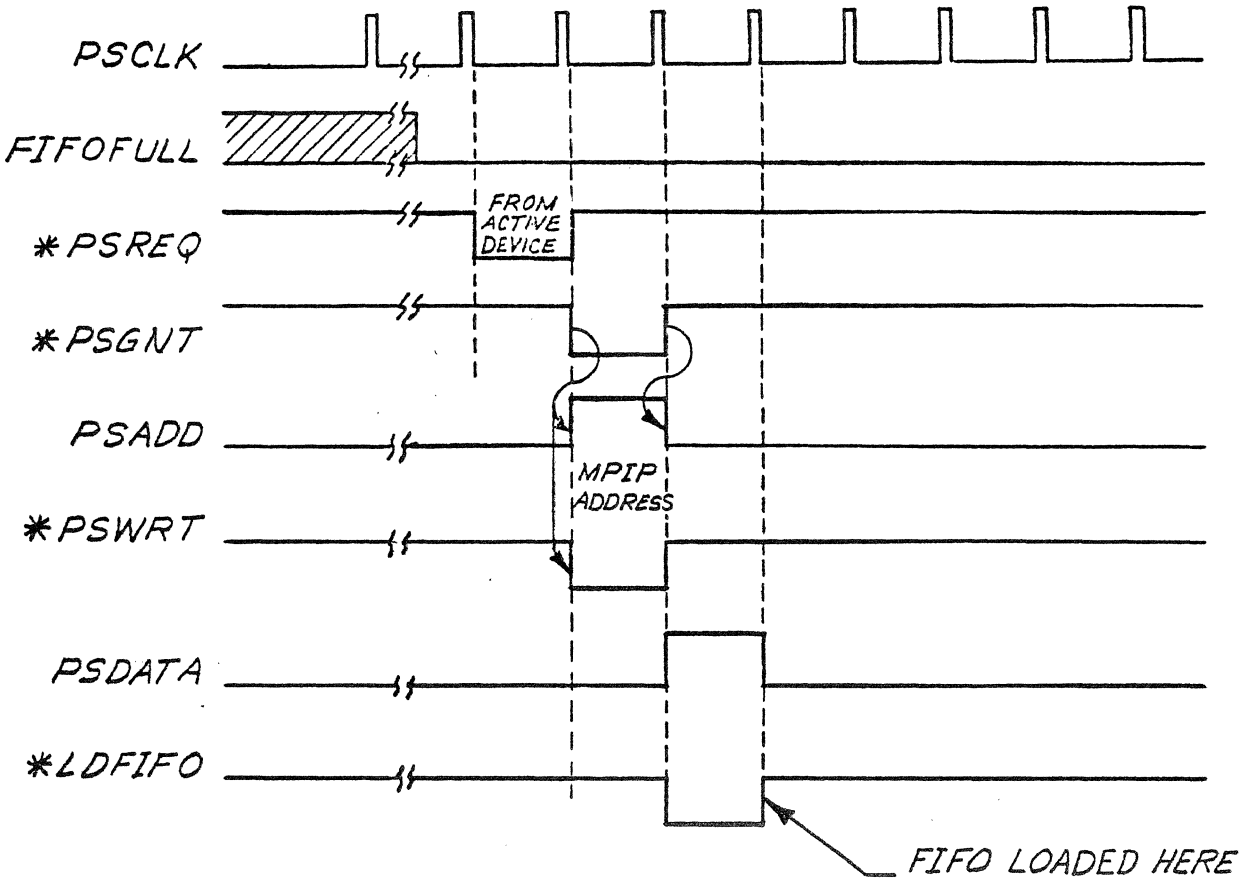


FIGURE 5·3-6

ACTIVE DEVICE WRITE TO MAP PASSIVE INPUT PORT

device has data for the active Input Controller, unless the MAIA is to be loaded. During the time the sequencer remains in the "Wait for Data" state, a PSBUS cycle is in progress. The Input Controller gates the MAIA to the address lines of the PSBUS upon receiving a grant from the Bus Arbiter; the passive device gates the data onto the PSBUS; and, the data is strobed into the FIFO upon exit to the "Request PSBUS" state. If a command to change the MAIA content occurs when the passive device has data on the BUS, the data is not loaded into the FIFO and the "IDLE" state is entered. If the command to change the MAIA occurs while in the "Wait for Data" state and the passive device does not have data on the PSBUS, the "Clear PSBUS" state is entered. The "Clear PSBUS" state is maintained until the current PSBUS cycle is complete. Upon completion of the bus cycle the "IDLE" state is entered and the data from the bus cycle is not strobed into the FIFO.

The "Request PSBUS" state is maintained for only clock period and, upon exit, sets another request for the PSBUS only if going to the "Wait for Data: state. The "Wait for Data" state is entered only if the FIFO is not full and the MAIA is not to be loaded; else, the "IDLE" state is entered.

An example of the MAP Input Controller functioning as an

active device and fetching commands and data from a PS Memory file is described in the next few paragraphs.

Suppose the data file as depicted in Figure 5.3-4 exists in PS Memory.

Partition A	1000	MOVE	RSR instruction with
	1001	Ax	a count field of 12_8
	1002	Ay	and a draw sequence
	1003	Bx	of M,D,D,D,D (061764)
	1004	By	
	1005	Cx	
	1006	Cy	
	1007	Dx	
	1010	Dy	
	1011	Ax	
	1012	Ay	
	1013	JUMP	RSR Jump command (002000)
	1014	4000	
Partition B	4000	MOVE	RSR instruction with
	4001	Ex	a count field of 6
	4002	Ey	and a draw sequence
	4003	Fx	of M,D,D,D (061772)
	4004	Fy	
	4005	Gx	
	4006	Gy	
	4007	HALT	RSR Halt command (000000)

Figure 5.3-4
PS Memory Data File

Assume the data words Ax.....Gy define the points of a square and a triangle as depicted in Figure 5.3-5.

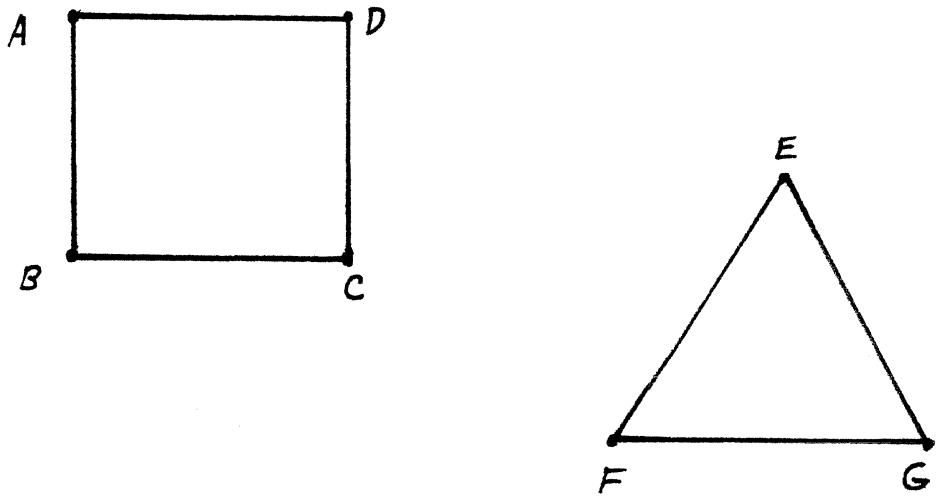


Figure 5.3-5
Displayed Result of Data File

Notice that the file consists of two partitions. Partition A from locations 1000 to 1014 contains an RSR, 2-dimensional draw, sequence command of Move, Draw, Draw, Draw,... and a count of 12_8. This partition also contains a JUMP command to the MAP. When executed it will cause the MAP Input Controller to fetch data starting at location 4000. Partition B contains the same RSR, 2-dimensional draw sequence command but with a count of only 6. The 12_8 data words of partition A define the points of the square and the 6 data

words of partition B define the points of the triangle.

The program initiatized the MAIA to point at PS Memory location 1000. At the beginning of a frame update, the MAP Input Controller fetches the PS Memory location pointed to by the MAIA and puts the contents in the MAP Input FIFO. To do this the input sequencer requests the PSBUS upon exit of state 00 to state 11. In state 11 the sequencer waits for the word from PSMEM location 1000 to settle on the data lines of the PSBUS. Upon exit from state 11 to state 10 the word is strobed from the PSBUS into the FIFO. The MAP itself waits in the Load RSR State (see Appendix A, MAP Alogrithm) until the FIFO is not empty. Since a word <u>is</u> in the FIFO, the MAP loads this word into its Repeat Status Register (RSR). The Input Sequencer tries to keep the FIFO full by accessing PS Memory locations and loading the data into the FIFO. This is accomplished by the input sequencer executing the loop consisting of state 10 and state 11. The MAP tries to empty the FIFO by unloading the FIFO contents and operating on the data. There will be a time when the MAP fetches the JUMP command from the FIFO. This JUMP command will be loaded from the FIFO into the RSR thus causing the MAP to execute the JUMP states of its algorithm. The JUMP command causes the MAP to fetch the next word from the FIFO and load it into the MAIA register; therefore, 4000 is fetched from the FIFO and loaded into the MAIA. The input will be in either state

10 or state 11 when the MAP issues the MAIA load signal; thus, dispatching the input sequencer to either state 00 or state 01 depending on whether the PSBUS is clear or not. The load signal, *MCLDMRI, also clears the FIFO, therefore the input sequencer has an empty FIFO for the next command fetched from the PS Memory location 4000. At this point the MAP has received the data for the square and is waiting for the RSR Move command and data for the triangle. The Input Sequencer fetches the RSR Move command, data, and the Halt command, and the MAP unloads the commands and data from the FIFO. Upon execution of the Halt command, the MAP waits for the program to initiate the next frame update process.

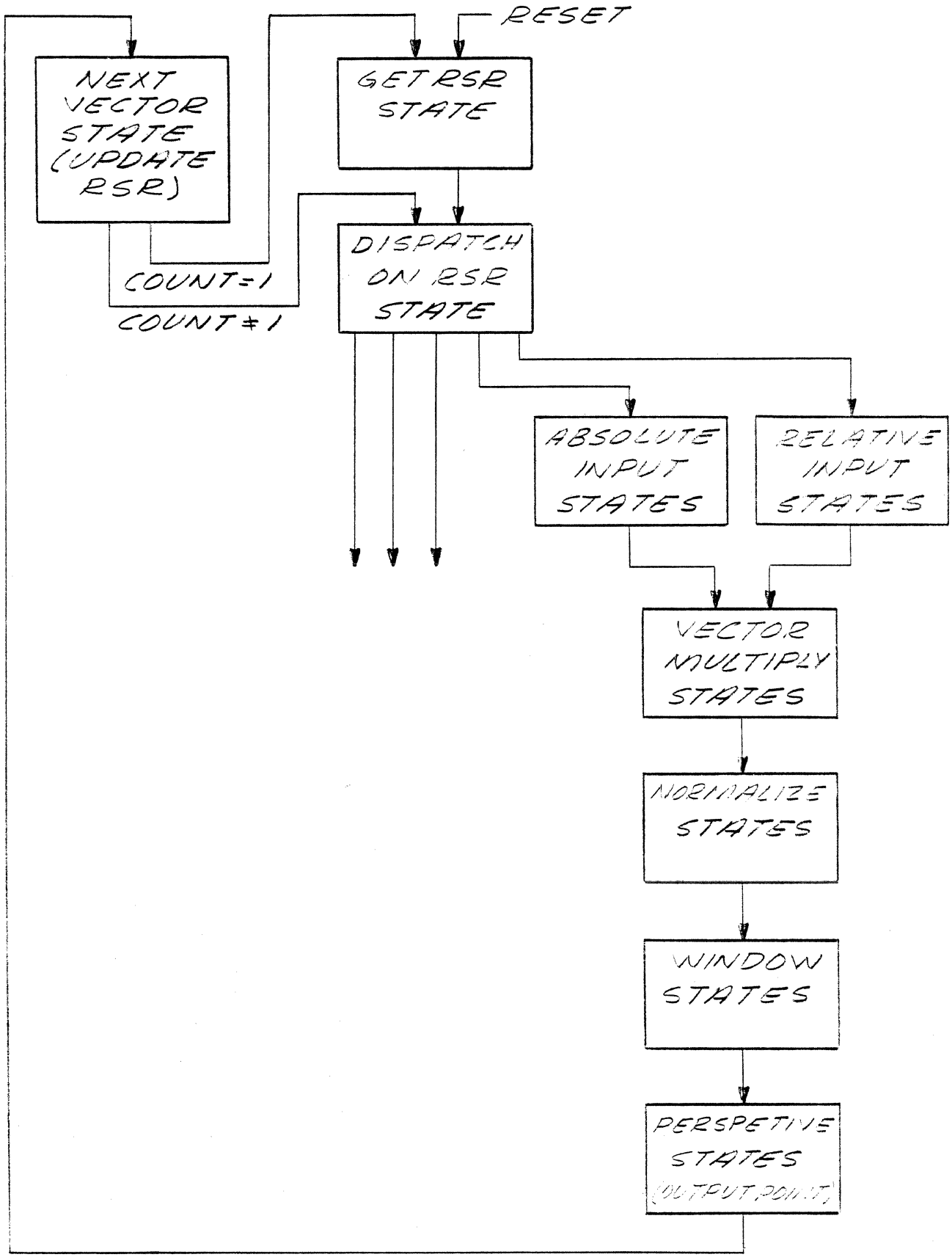
The implementation of the MAP Input Sequencer is shown on the MAP I/O Sequencer drawing, 195118-600, sheet 3. The four states are derived from two bits of the register in location 41. ISTATEA is the LSB and ISTATEB is the MSB (Refer to Figure 5.3-3, MAP Input Sequencer Diagram).

5.3.2 MAP Input Controller - Passive

When operating as a passive device, the MAP Input Sequencer is always "IDLE": Data transfers to the MAP Input Controller occur by an active device writing data to the MAP Passive Input Port (MPIP=177777). The MAP Input Controller passively waits for data directed to the MPIP and synchronously loads

the FIFO during the data cycle of the PSBUS. The timing diagram of Figure 5.3-6 illustrates an active device writing data to the MPIP.

The PSWRT line from the PSBUS is input on sheet 1 of the 195121-600 drawing. The MPIP address is decoded by bug 15 on sheet 2 and the PASIN bit of bug 26 is set at the beginning of the PSDATA period of the data transfer cycle. PASIN is input to the MAP I/O Sequencer Card, 195118-600, sheet 3, and generates LDFIFO. LDFIFO is input to the MAP Buffer Card, 195119-600, which contains the FIFO.



SIMPLIFIED PARTIAL NIAP STATE DIAGRAM

FIGURE 5.3-9

5.3.3 RSR Register and Update PROM

The Repeat Status Register (RSR) is the command register to the MAP. The register contains three fields as described on page 2-36 and 2-50 of the PS2 Reference Manual. The three fields are:

1. COM----------Command field (bits 15,14)
2. FUNCTION/FSM1-FSM2--------Function field for "Control" commands, or Finite State Machine fields for "DRAW" commands (bits 13-8)
3. OPERAND/COUNT----Operand field for some "Control" commands, or Co-ordinate Count field for "DRAW" commands (bits 7-0)

The Update PROM sequences the Finite State Machine fields through patterns defined by the contents of the RSR at the beginning of a new DRAW command to the MAP.

5.3.3.1 GET RSR State

In the MAP algorithm exists a state which enables the FIFO to the BBUS and loads the RSR with BBUS data. This state is called the "Get RSR" state (See Appendix A, MAP Algorithm). Upon "getting" an RSR, the MAP algorithm decodes the COM and FUNCTION/FSM fields, then branches appropriately.

5.3.3.2 Control and DRAW Commands

There are two basic types of RSR Commands:

1. RSR "Control" Command
2. RSR "DRAW" Command

The COM field of the RSR word determines the command type. If COM = Ø, the command is "Control". If COM ≠ 0, the command is a "DRAW".

Thirteen "Control" commands are defined by the FUNCTION field of the RSR word. These Control commands are described in detail on pages 2-37 through 2-48 of the PS2 Reference Manual.

The three "DRAW" commands are:

1. 2DDRAW - Two dimensional draw command
2. 3DDRAW - Three dimensional draw command
3. 4DDRAW - Four dimensional draw command

The "DRAW" commands are defined in detail on page 2-51 of the PS2 Reference Manual.

The 2DDRAW command operates on data defining vector points in an X,Y plane. The 3DDRAW operates on data points in X,Y,Z space; as also, does the 4DDRAW; however, 4DDRAW

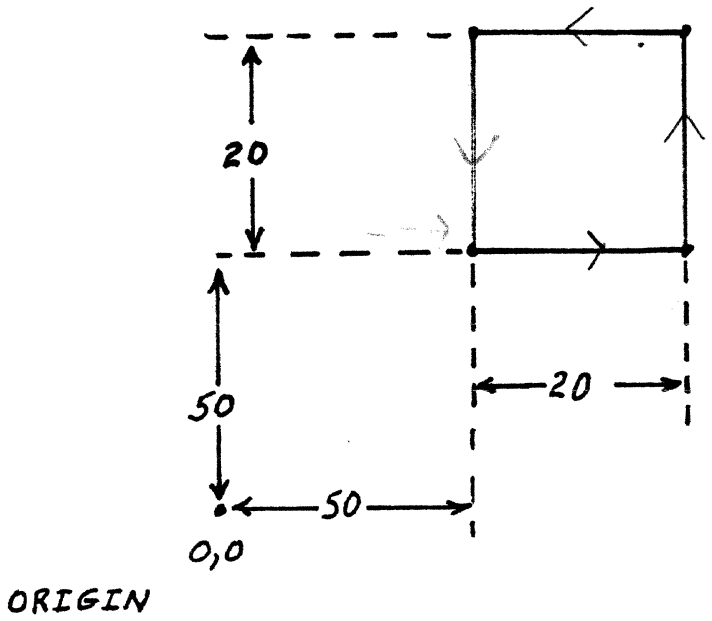
vector points are defined by elements X,Y,Z and W. The W element of each point is a scaler for that point. When the MAP processes 4DDRAW data, it eventually devides the X,Y and Z elements by the scaler, W.

5.3.3.3 Drawing Sequences and Data Interpretation

All RSR "DRAW" commands have an associated drawing sequence and coordinate count as defined by the FSM and COUNT fields. Each time the "DRAW" command is performed on a point vector, the FSM1 and FSM2 fields are updated while the COUNT field is incremented. This provides the capability of performing pattern sequences of Moves and Draws (M,D,D,D, ... or M,D,M, D ..., etc.) as well as interpret data as Absolute or Relative points on a given number of vectors defined by the COUNT field. An original RSR "DRAW" received from the FIFO by the MAP contains a negative number (in Two's Complement form) residing in the COUNT field. The FSM2 field contains a number defining the sequence pattern in which the MAP will interpret incoming vector points (either as Absolute or Relative data points). Example: A,R,A,R... A,R,R,R... R,R,R,R, etc. The FSM1 field contains a number defining the sequence pattern of Move or Draw commands and the MAP will operate on point vectors accordingly; thus outputting data to the Picture Generator in a sequence defined by the Move-Draw pattern.

5.3.3.4 Drawing Sequence Example

Consider the square of Figure 5.3-7 with sides 20 units each and the lower left vertice offset from screen center by 50 units.



Square Offset From Screen Center

Figure 5.3-7

The RSR command and associated data file input to the MAP may be defined as illustrated in Figure 5.3-8.

051773	2DDRAW (A,R,R,R... M,D,D,D...COUNT = - 5)
50	Ax (Absolute)
50	Ay (Absolute)
20	Bx (Relative)
0	By (Relative)
0	Cx (Relative)
20	Cy (Relative)
-20	Dx (Relative)
0	Dy (Relative)
0	Ax (Relative)
-20	Ay (Relative)
0	Halt

Figure 5.3-8
RSR Command and Associated Data File for a Square

A simplified diagram of the MAP states executed is illustrated in Figure 5.3-9.

Execution of the RSR command and input of data by the MAP is described in the following sequence of events.

1. The MAP inputs the RSR, decodes the COM and FSM fields, and executes the "Absolute Input States" where Ax and Ay are input as absolute data defining point A.

2. Point A is transformed in the "Vector Multiply States" checked against a defined window, put in perspective, and output. Point A is output with a Move command to the Picture Generator.

3. The FSM and COUNT fields of the RSR are updated in the "Next Vector State", the updated RSR is decoded and the "Relative Input States" are executed to input data for point B.

4. The MAP loops through the "Relative Input, Vector Multiply, Perspective, and Next Vector" states until Draw commands with points B,C,D, and A are output to the Picture Generator.

5. After the Draw from D to A which closes the square, the MAP is in the "Next Vector State" with a COUNT = - 1; therefore, the command has been executed the specified number of times. The MAP enters the "Get RSR State" to input the next RSR command from the FIFO.

5.3.3.5 RSR/Update PROM Hardware Description

The COM field and FUNCTION/FSM field of the RSR (bits 15-8) are implemented by the dual input registers in bug locations 24 and 33 on the 195120-600, sheet 4 drawings. The Update PROM is in location 23. The FSM1 and FSM2 fields out of the register address the Update PROM. The Update PROM outputs a modified LSB for each of the FSM fields. The Operand/COUNT field (bits 7-0) of an RSR word input to the MAP are loaded into a counter (AC3) during execution of the "Get RSR" state

of the MAP algorithm. This counter exists on the 195114-600 drawing, sheet 3, implemented in locations 52 and 47.

5.3.3.6 Reading the RSR

For a user to read the RSR, the MAP must be in either the "Get RSR" or "Next Vector" state. While in either of these two states, the counter, AC3, is gated to the MAP RAMADR lines (MAP internal storage address). During the read cycle the RSR address from the PSBUS is decoded on the MAP Address Card, 195121-600 sheet 2. The decoded RSR address (177755) generates *GATRSR if the read line of the PSBUS is inserted. *GATRSR enables RAMADR to the lower *PSDATA lines (see sheet 5 of 195131-600). Also, the COM and FUNCTION/FSM fields of the RSR register are enabled to the upper address lines (see sheet 6 of 195120-600). An example of reading the RSR via the Direct I/O is illustrated by the flow chart of Figure 5.3-10.

5.3.3.7 Writing the RSR

When writing the RSR via an active device such as the Direct I/O, the MAP Input Controller must be passively waiting for data in the "Get RSR" state. The data is directed to the MPIP; therefore the first word to the MPIP is loaded into the RSR. The actual signal (*MCRSRLOAD) which loads the RSR is generated by the MAP algorithm controller during the "Get RSR" state. If the RSR is loaded with a "Control" command,

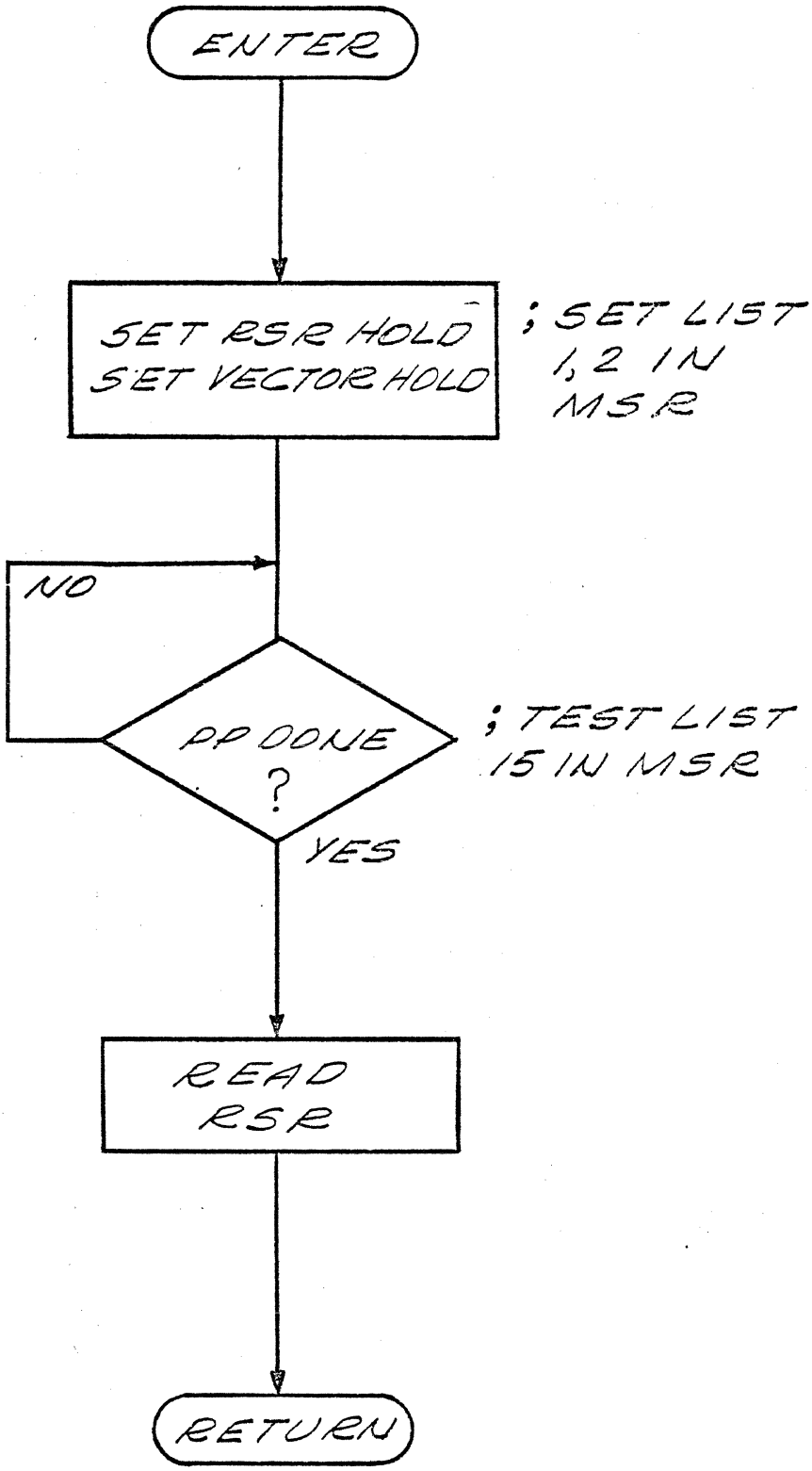


FIGURE 5.3-10 READ RSR FLOW DIAGRAM

the algorithm will return to the "Get RSR" state; however, if loaded with a "DRAW" command, the MAP will wait in certain states for drawing data. An example of writing the RSR via the Direct I/O is illustrated by the flow chart of Figure 5.3-11.

5.3.4 Extend Register

The MAP is a 24 bit machine which operates on point vectors. The elements (X,Y,Z,W) of the point vectors are defined each by a 16 bit word. During arithmetic operations, 16 bit precession will not insure picture quality; therefore, the MAP maintains 24 bit precession throughout all calculations, and all MAP parameters, working register, etc...are 24 bit registers. The user has the option of loading MAP parameters in either 16 or 24 bit precession. Since the PSBUS and MAP Input FIFO accommodate only 16 bit words, the Extend Register is used to format data from two, 16 bit, input words into a 24 bit word for loading MAP parameters. Also, the user has the options to store MAP parameters in 16 or 24 bit precession. In the 24 bit case, the Extend Register is used to format data from a 24 bit MAP word into two, 16 bit words for output to the PSBUS. Figure 5.3-12 illustrates formatting for a 24 bit load and 24 bit store.

To load a MAP parameter in 24 bit precession, the MAP inputs word 1 from the FIFO and loads bits 15-10 and bits 1-0 into

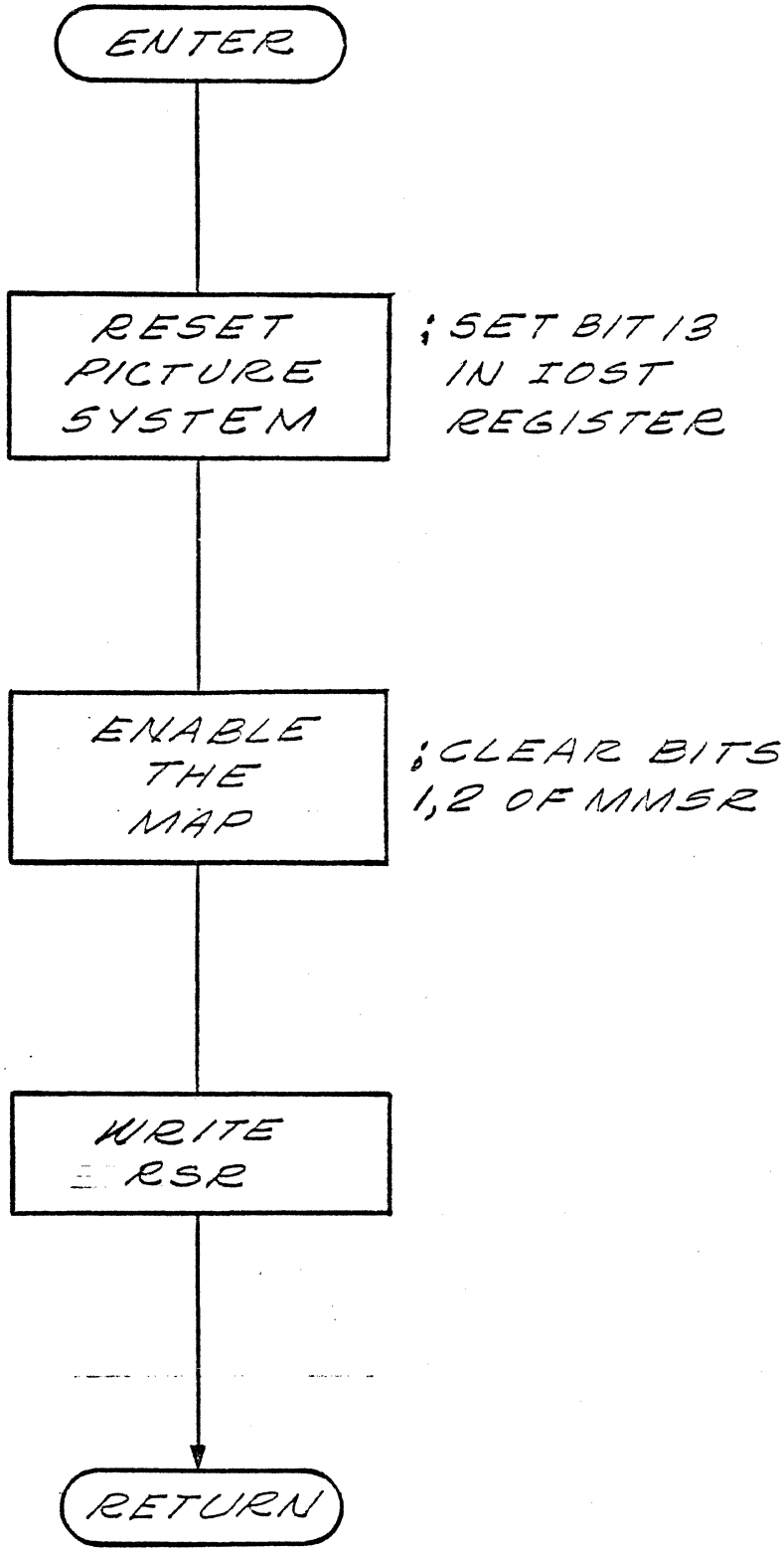


FIGURE 5.3-11 WRITE RSR FLOW DIAGRAM

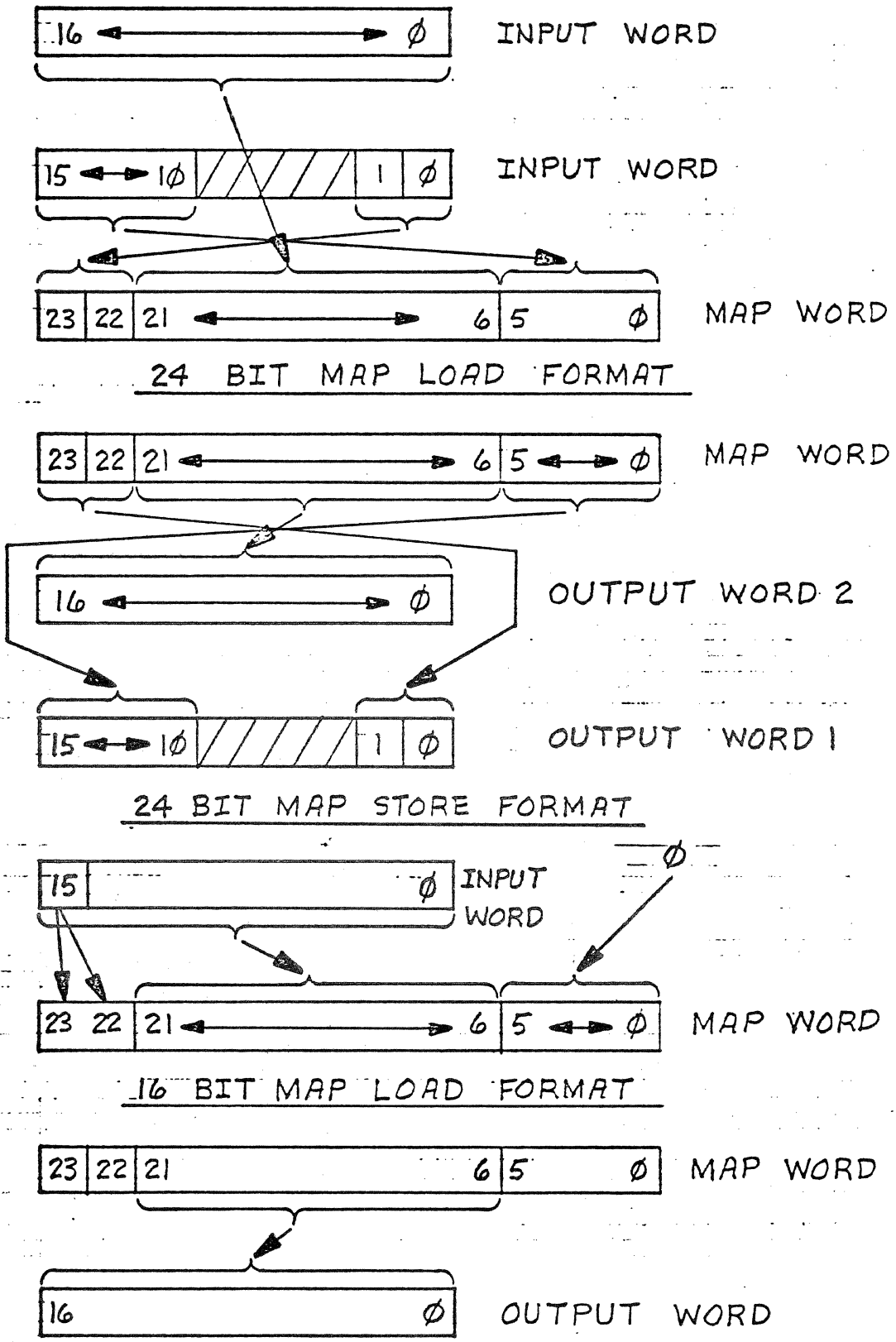


FIGURE 5.3-12 MAP LOAD/STORE WORD FORMAT

the Extend Register; therefore the Extend Register contains the upper two and lower 6 bits for the 24 bit MAP word. Next, the MAP inputs word 2 from the FIFO, gates the Extend Register and word 2 onto the BBUS, and strobes the 24 bits into the specified MAP parameter. The MAP performs a similar process when outputting a 24 bit word to be stored in two 16 bit words. Data flow for a 24 or 16 bit I/O is illustrated in Figure 5.3-13.

If a 16 bit precession load is executed, the data word from the FIFO is gated to BBUS bits 21-6, bit 21 is smeared into bits 23 and 22, and the Extend Register output is disabled to Ø. The result is a 16 bit sign extended word with trailing zeros on the BBUS. The Extend Register circuitry is implemented on the 195120 card, (See sheet 6 of the drawing).

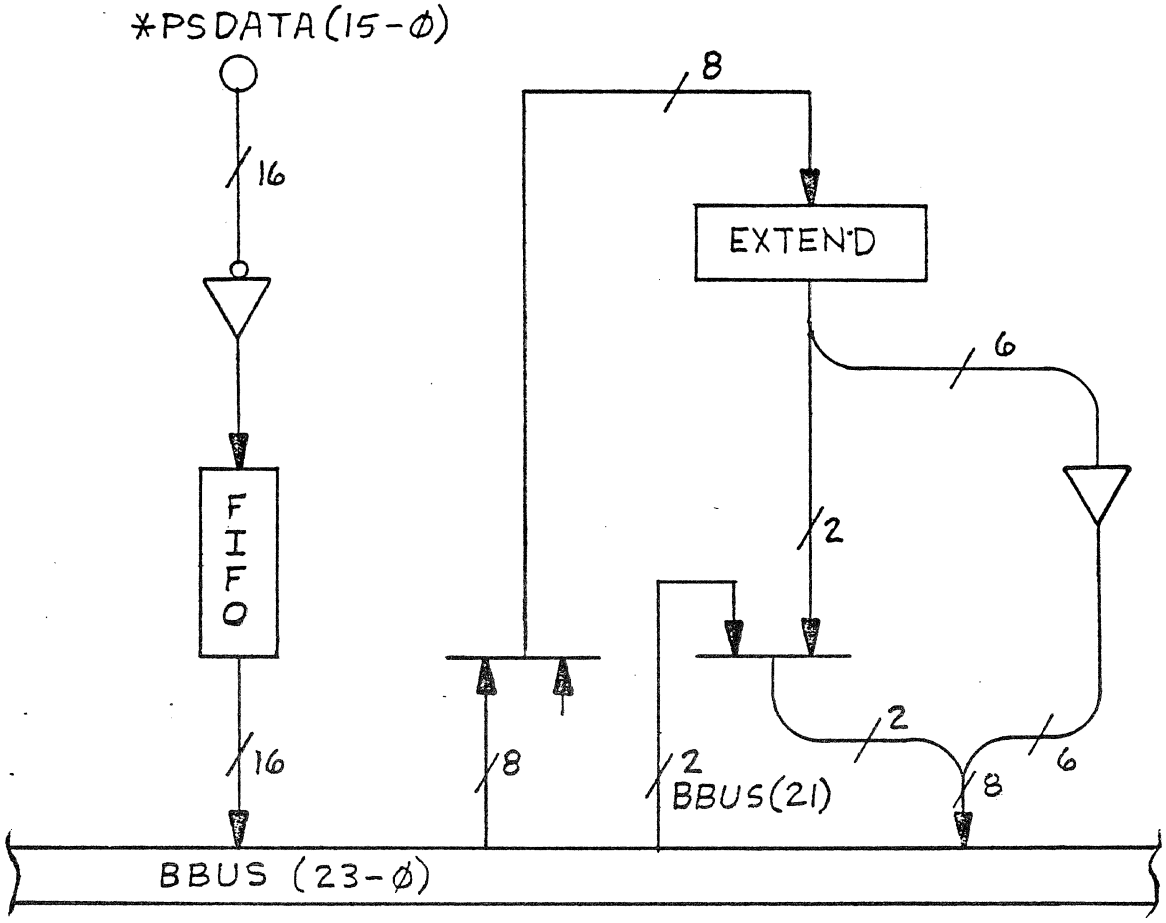


FIGURE 5.3-13 DATA FLOW BLOCK DIAGRAM OF EXTEND CIRCUIT

5.3.5 Matrix Arithmetic Processor (MAP)

The MAP is a semi-specialized arithmetic processor unit capable of receiving data in 16 bit words, packing data into 24 bit words, and performing the following arithmetic functions:

1. addition
2. subtraction
3. multiplication
4. reciprocation
5. normalization

These arithmetic functions are used to implement the necessary equations which transform defined point vectors from the Data Space Coordinate system to the Screen Coordinate system for output to the Picture Generator.

The MAP is classified as semi-specialized since it contains the necessary bus structure along with the necessary data storage area, counter, comparators, registers, multiplexers, programmable control, etc... to be micro programmed for implementation of a specialized algorithm.

Typically the MAP is loaded with an initial transformation matrix. This initial matrix is usually the Idenity Matrix; therefore, if used to transform a point, the transformed point is identical to original point.

$$[P] \cdot \begin{bmatrix} I \end{bmatrix} = [P'] \qquad \text{and } P=P'$$

The user may restrict display of the visual results to a defined area on the CRT by setting up a viewport in Screen Coordinates. The viewport is defined by loading six parameters in the MAP. These six parameters are:

Viewport x 1/2 size
Viewport x center
Viewport y 1/2 size
Viewport y center
Viewport z size
Viewport z front

The six parameters define a rectangle as well as a beam intensity range in the z axis (see Figure 5.3-14).

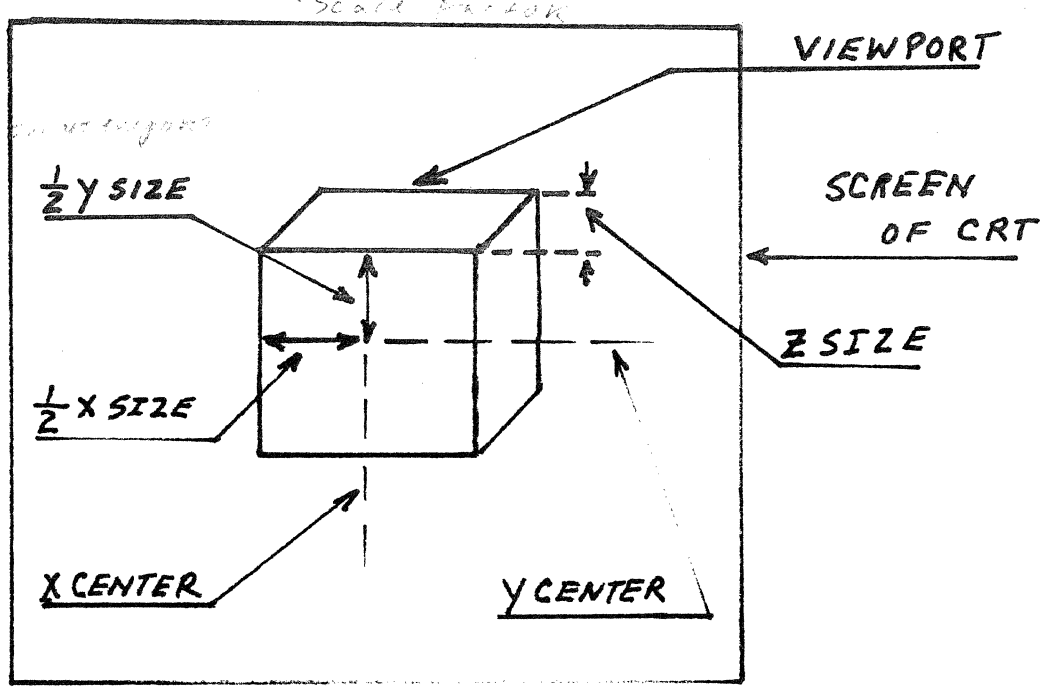


Figure 5.3-14
Viewport Example

When the MAP is pre-loaded with an Idenity Matrix and a defined viewport, all transformed data is perspectively mapped into the defined viewport; and, since the viewer's eyepoint default to minus infinity, the MAP output to the Picture Generator consists of an orthographic view of the complete set of vector points defined by the input data.

The user may not want to visualize the entire set of point vectors defined in the data space but only those within a defined area. In this case, the MAP transformation matrix is concatenated with a WINDOW matrix which defines a truncated pyramid known as the Frustrum of Vision or window. All vectors falling in the window are to be visable and all vectors falling outside the window are to be non-visable. This enables the user to translate objects or zoom the eyepoint and visualize only data seen through the window. Figure 5.3-15 illustrates a Frustrum of Vision defined by six window parameters.

Transformation of a point vector by a WINDOW matrix scales the point such that the x,y, and z element of the point can be checked against the element w to determine if the point is in or out of the Frustrum of Vision. Figure 5.3-16 illustrates transformation of a point P and the six window checks performed on the transformed point P.

The checks are performed simultaneously on both the previous point and the new point; therefore, for each check the MAP

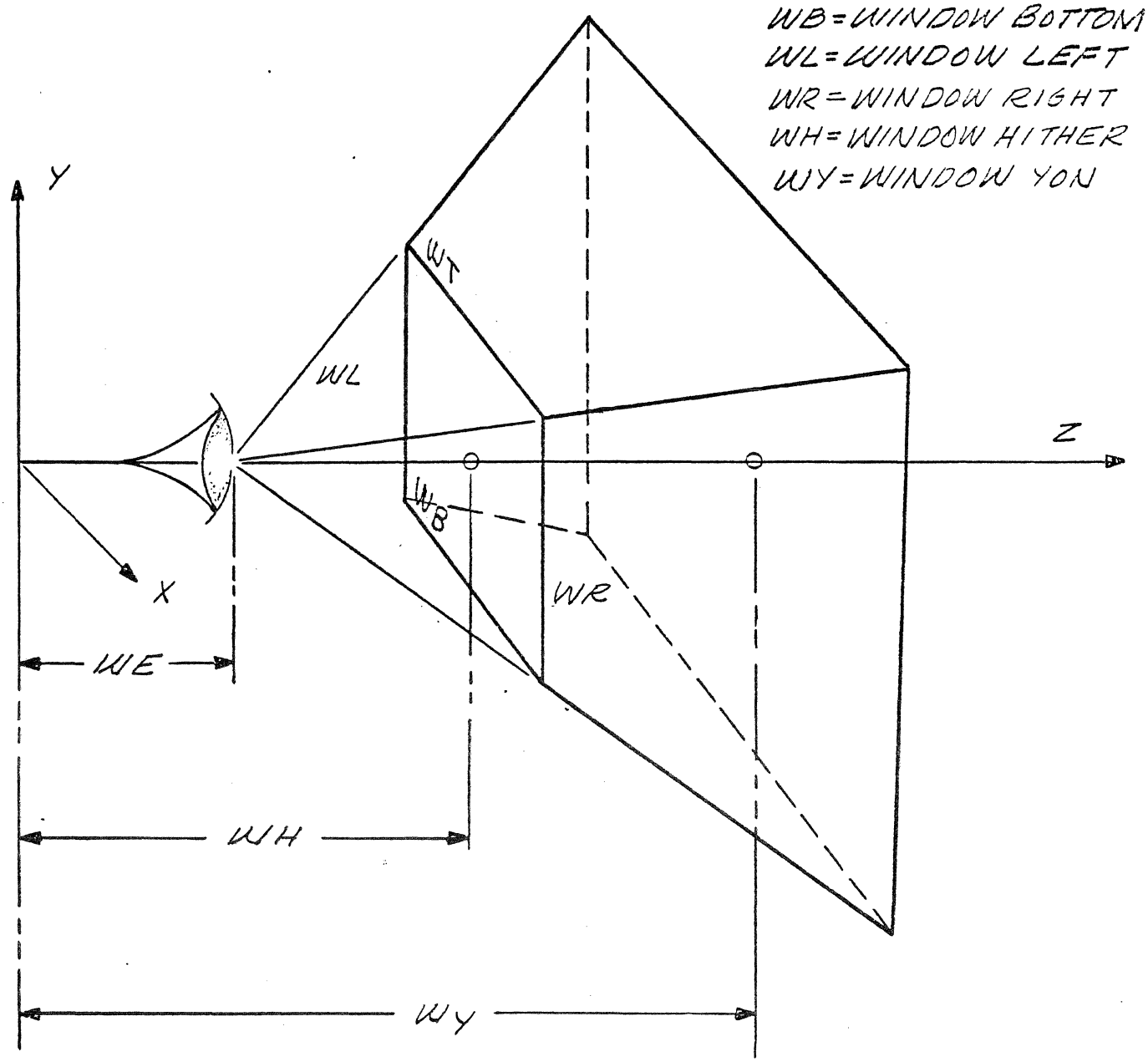


FIGURE 5.3-15

FRUSTRUM OF VISION SHOWING THE EYE POSITION IN RELATION TO AN ARBITRARY COORDINATE AXIS

determines whether the line between the two points needs clipping.

$$[P]\begin{bmatrix} \\ W \\ \\ \end{bmatrix} = [P']$$

$P = x,y,z,w$

$P' = x',y',z',w'$

1.	$x' - w' < \emptyset$	in side of Right plane
2.	$x' + w' \geq \emptyset$	in side of Left plane
3.	$y - w' > \emptyset$	in side of Top plane
4.	$y' + w' \geq \emptyset$	in side of Bottom plane
5.	$z' - w' < \emptyset$	in side of Yon plane
6.	$z' > \emptyset$	in side of Hither plane

Figure 5.3-16
WINDOWING a Point, P

Figure 5.3-17 illustrates a line between two points, Previous and New, with a defined window. Four cases are illustrated. For each of the six equality checks, the MAP observes the following rules.

1. If the x,y, or z element of both points is outside the window plane, then both points are outside the Frustrum of Vision. Input the next point.

2. If the checked x,y, or z element of both points is inside the window plane, then check the next element.

3. If the checked x,y, or z element of one point is inside and the element of the other point is outside, the line intersects with the window; therefore clip the line against the appropriate window plane, then check the next element.

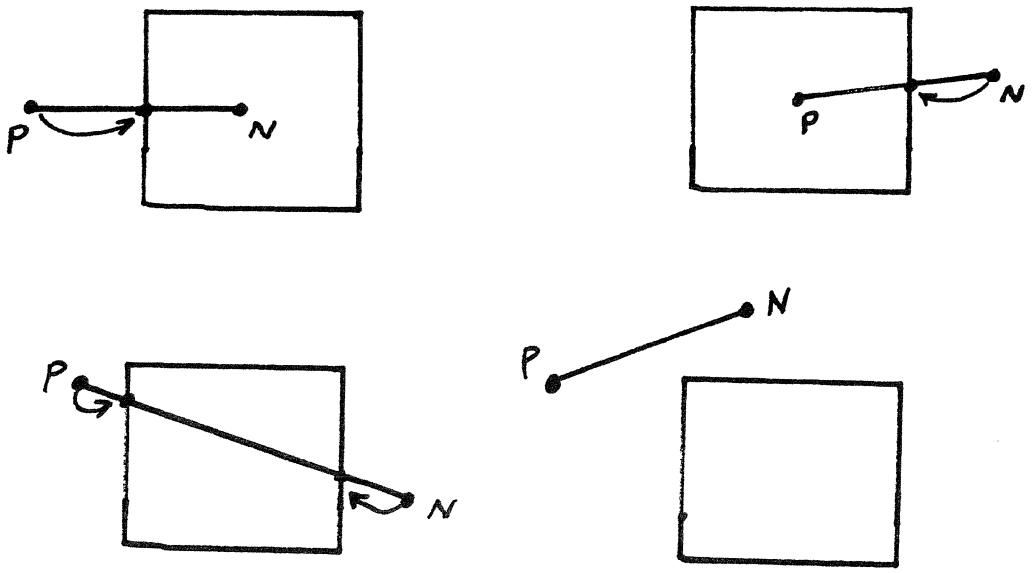


Figure 5.3-17
Lines between Previous Point and New Point with a Defined Window

Upon finding a line segment within the window, the MAP saves the New Point, perspectively maps the previous point into

the viewport, maps the New point into the viewport, assigns the saved New point to Previous point, and inputs the next point vector.

The MAP receives commands and data from the MAP Input Controller. An input command received in the RSR register is decoded and appropriate action taken by the MAP which sequences through states as defined in the MAP algorithm (see Appendix A). Execution of command specific and data specific states is determined by the COM and FUNCTION/FSM fields of the RSR.

"Control" commands direct the MAP as follows:

1. Load or store MAP parameters.
2. Initiate jumps or subroutine jumps within a MAP input file.
3. Relocate data within MAP internal storage.
4. Save or recall MAP transformation matricies.
5. Concatenate the MAP's transformation matrix with input matricies.

"DRAW" commands direct transformation of input point vectors by initiating the following MAP functions and transformation:

1. Windowing
2. Clipping
3. Rolation
4. Translation

5. Scaling

6. Viewport Mapping

Data associated with a DRAW command is input to the MAP in 2,3, or 4 dimensional space. The FSM2 field of a DRAW command directs MAP interpretation of input data with respect to the data space origin.

The FSM1 field directs MAP interpretation of point vectors as to being either "set point" or lines.

The MAP outputs data as screen coordinates to the MAP Output Formatter. There is a special mode of operation, where some MAP functions are ignored and input is simply passed to the formatter. This mode if operation is used to pass character data and Line Generator status through the MAP.

5.3.5.1 MAP Data Store and ALU Unit

The MAP Data Store and ALU consist of the following:

1. 256 x 24 bit RAM Memory Data Store
2. Three MAP Support Registers

   MAP Buffer (MB)
   Multiplicand Register (MDA)
   Accumulator (RA)

3. Arithmetic Logic Unit (ALU)

The Data Store and ALU Unit is implemented on two 195116 cards each comprising a 12 bit slice of the unit. Figure 5.3-18 illustrates a block diagram of the Data Store and ALU.

The 256 x 24 bit RAM Memory is addressable only by MAP control; therefore, to write or read the memory, the user must issue LOAD or STORE commands to the MAP. The memory is used by the MAP as a parameter and register file for MAP calculations. These parameters and registers are:

1. Input Registers (x,y,z,w)
2. Base Registers (x,y,z,w)
3. Output Registers (x,y,z,w)
4. Save Register (x,y,z,w)
5. Viewport Parameter
6. Working Register

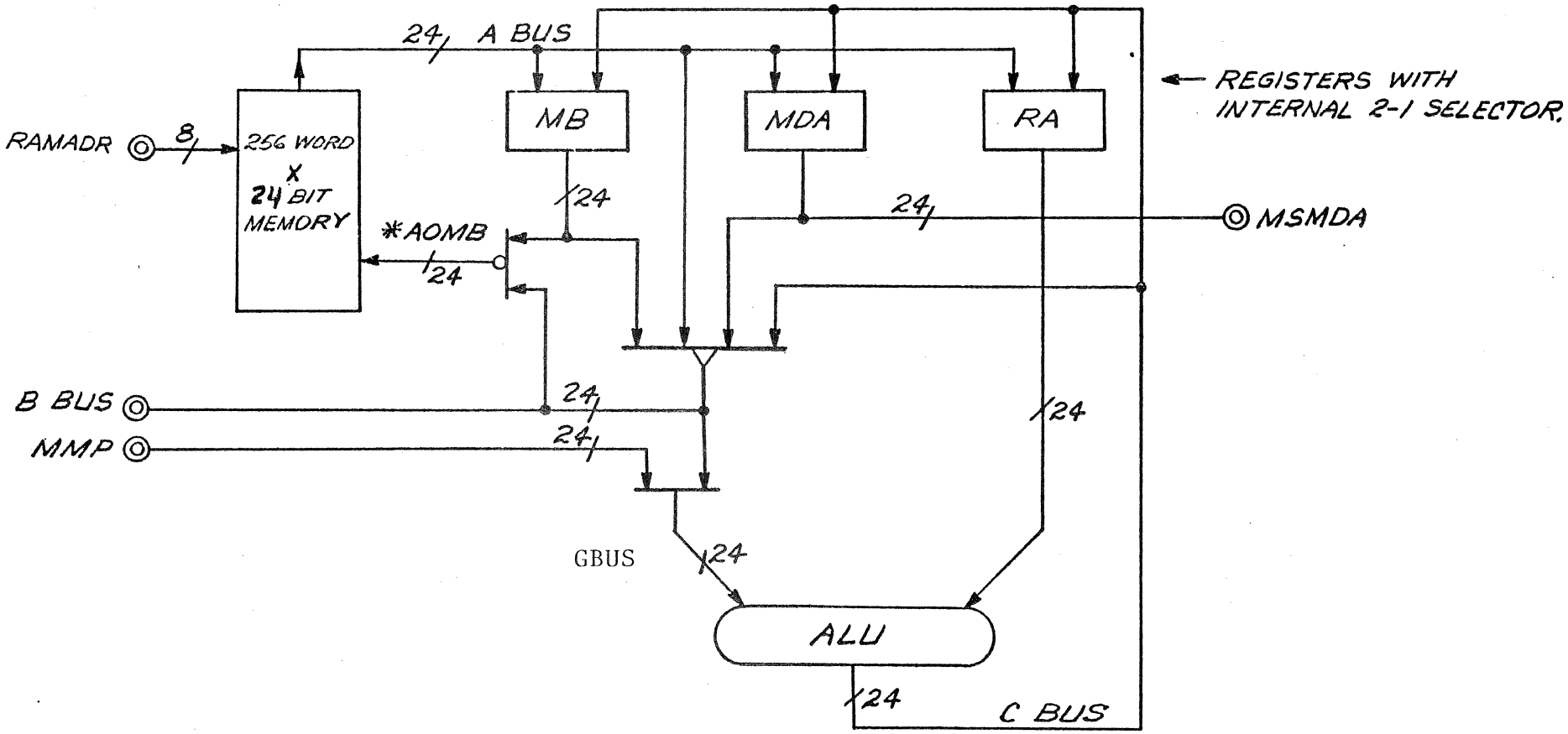


FIGURE 5-3-18

DATA STORE AND ALU BLOCK DIAGRAM

7. Transformation Matrix Address Register
8. New Clip Register (x,y,z,w)
9. Clip Save Register (x,y,z,w)
10. Transformation Matrix Registers
11. Matrix Stack Register

These parameters and registers are defined in detail in the PS2 Reference Manual, pages 2-32 through 2-35. The three MAP support registers are used to buffer operands into the ALU. The MDA register also buffers the multiplicand operand to the MAP's multiplier. The other operand is buffered elsewhere. During MAP operation, parameters and data are output from the RAM Memory to the ABUS. Note that the ABUS drives the input to the MB, MDA, RA, and also may be gated to the BBUS. The ALU receives one operand from the GBUS: the multiplier's output (MMP), the RAM Memory, the (MB), or the MDA. The other operand to the ALU is input from the RA. This structure provides a versatile mechanism which is used to process MAP arithmetic functions. For example to transform a point P, with elements x,y,z, and w, the MAP multiplier outputs 16 products on the MMP lines. The 16 products are summed in groups of 4 to calculate the transformed point x',y',z', and w'.

$$(x\ y\ z\ w)\begin{bmatrix} T_{00} & T_{01} & T_{02} & T_{03} \\ T_{10} & T_{11} & T_{12} & T_{13} \\ T_{20} & T_{21} & T_{22} & T_{23} \\ T_{30} & T_{31} & T_{32} & T_{33} \end{bmatrix} = (x'x'z'w')$$

$$x' = x \cdot T_{00} + y \cdot T_{10} + z \cdot T_{20} + w \cdot T_{30}$$

$$y' = x \cdot T_{01} + y \cdot T_{11} + z \cdot T_{21} + w \cdot T_{31}$$

$$z' = x \cdot T_{02} + y \cdot T_{12} + z \cdot T_{23} + w \cdot T_{32}$$

$$w' = x \cdot T_{03} + y \cdot T_{13} + z \cdot T_{23} + w \cdot T_{33}$$

The first product from the Multiplier is passed through the ALU and stored in the accumulator, RA. The second and third products are summed with the RA then restored back in the RA. The fourth product is summed with the RA then the total sum of four products stored in RAM Memory as x'. The remaining elements (y',z', and w') are calculated and stored similarly.

Another example of the flexability of the Data Store/ALU unit is the perspective calculation performed by the MAP. To put the x' element of a transformed point into viewport perspective, the following equation is implemented.

$$x_s = \frac{x'}{w'} \cdot VSX + VCX$$

where: x_s = x in screen coordinates

$VSX = \frac{\text{Viewport size in x}}{2}$

VCX = Viewport center in x

In actural implementation, the MAP fetches x' from RAM Memory, stores it in MRA, reciprocates w, and stores it in MDA. MRA and MDA are inputs to the Multiplier. The Multiplier re-

turns $\frac{x'}{w'}$, which is gated to the GBUS, passed through the ALU to the CBUS, and loaded in MDA. The Viewport parameter VSX is fetched from RAM Memory and loaded into MRA. The Multiplier turns the product $\left(\frac{x'}{w'} \cdot VSX\right)$ which is gated to the GBUS as one input to the ALU. Simultaneously, the MAP fetches the Viewport parameter, VCX, from RAM Memory and loads it into the RA as the other operand to the ALU. The ALU returns the sum

$$\frac{x'}{w'} \cdot \quad VSX + VCX$$

on the CBUS which is loaded into MDA. MDA now contains x_s which is a viewport mapped element in the Screen Coordinate system. All register loading, data multiplexing, RAM Memory addressing, etc... is controlled by the MAP algorithm implemented as Micro code in the MAP Control Store.

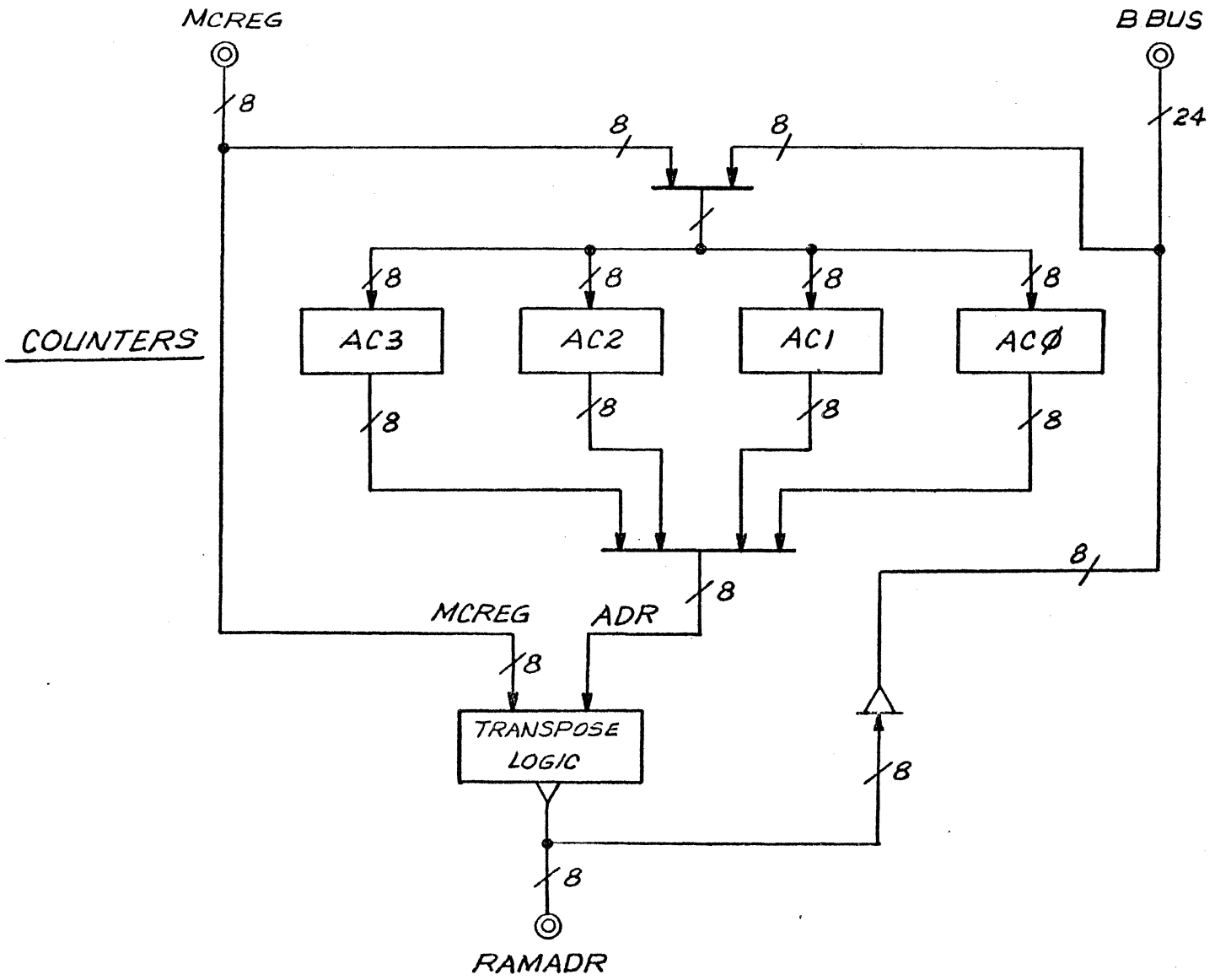
5.3.5.2 MAP Address Counters

Four MAP address counters (ACØ - AC3) implement the following functions:

1. control loops in the MAP algorithm
2. track the current coordinate count for a given RSR command
3. control addressing of the MAP's internal RAM Memory

The address counters and associated data paths are illustrated in the block diagram of Figure 5.3-19.

Use of an address counter to control a loop in the MAP algorithm is accomplished by loading the counter with a negative number from the MAP Control's DOIT Register upon entrance into a loop, incrementing the counter within the loop, and checking the counter for -1 as a dispatch condition out of the loop. Note on the block diagram, MCREG is eight bits from the DOIT register. The counters are implemented with 93S16 chips which have a look ahead carry output; therefore, examinations of the carry output determines if the count is equal to a -1. For an example of an address counter (AC2) used as a loop controller, see the "Output New Vector" routine in the MAP Algorithm (Appendix A).



ADDRESS COUNTERS AND ASSOIATED DATA PATH BLOCK DIAGRAM

FIGURE 5-3-19

Also, in the MAP Algorithm, AC3 is used to track the current coordinate count of an RSR command. During the "Get RSR" state, AC3 is loaded via the BBUS with the Count field of an incoming RSR command. AC3 contents are incremented and checked against a -1 upon exit from the "Next Vector" state. If the coordinate count has run out (AC3 = -1) the "Get RSR" state is entered to input a new command. If more coordinates remain to be processed (AC3 $\neq$ -1) the "Dispatch" state is entered to reiterate executions of the command for another set of coordinate data.

Use of an address counter to sequence addressing in the MAP's internal RAM Memory consist of loading a counter with a beginning address via the MAP Control's DOIT register, gating the counter to the RAM address lines (RAMADR), and incrementing the counter to the next address. For an example, see the "Absolute Input" routine in the MAP Algorithm. Note that in state 2 of the routine, a Ø from the DOIT register is not only gated as input to ACØ, but also gated to the RAM address lines. This enables simultaneous addressing of the RAM and initialization of the address counter. The counter is incremented in state 3 and gated to the RAM address lines again in state 4. In this example, AC2 is also used as a loop counter since it was initially loaded during the "Dispatch" state with the number of words to be input for an incoming vector.

The counters and their associated data path are implemented on the 195114 card.

5.3.5.3 Normalize Sense

MAP arithmetic operations treat all numbers as two's complement, signed (fixed point) fractions. The 24 bit MAP word is an extention of a two's complement fraction. Figure 5.3-20 illustrates the 24 bit MAP word extended from a basic, 16 bit, PICTURE SYSTEM word. Three sign bits are maintained in the MAP word to preserve sign during MAP transformation operations which sum four products from the array multiplier. In the process of summing four binary numbers, overflow of two bits may occur. Consider the transformation of the point P by the matrix T resulting in the transformed point P'.

$$[P]\begin{bmatrix} T \end{bmatrix} = [P']$$

where: $P = x,y,z,w$

and $P' = x',y',z',w'$

The transformed elements of P' are defined as:

$$x' = x \cdot T_{00} + y \cdot T_{10} + z \cdot T_{20} + w \cdot T_{30}$$

$$y' = x \cdot T_{01} + y \cdot T_{11} + z \cdot T_{21} + w \cdot T_{31}$$

$$z' = x \cdot T_{02} + y \cdot T_{12} + z \cdot T_{22} + w \cdot T_{32}$$

$$w' = x \cdot T_{03} + y \cdot T_{13} + z \cdot T_{23} + w \cdot T_{33}$$

x',y',z', and w' may result in very small positive or negative fractions such as the two following examples:

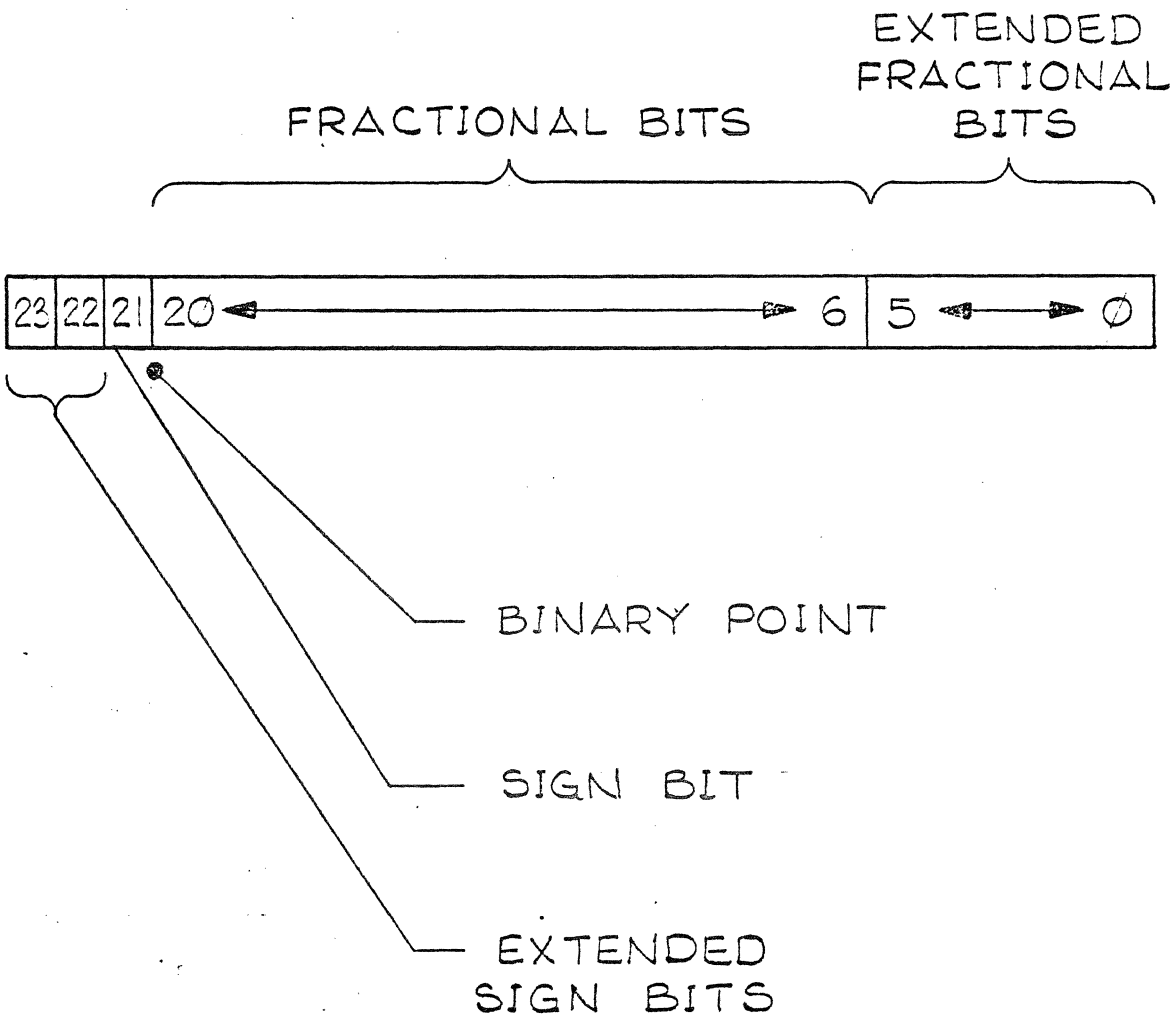


FIGURE 5.3-2Ø
SIGN EXTENDED 24 BIT MAP WORD

000.00000000000000100101

111.11111111111111011001

Unless normalized, accuracy will be lost during later viewport and perspective calculations. The perspective equation divides x',y', and z' by w'; therefore, x',y',z', and w' may be left shifted all the same amount without effecting the perspective results; however, more accuracy will be maintained.

Upon summing four products of the transformed element x', the Normalize Sense logic determines the left shifts required to properly normalize x'. This shift code is saved and compared with the shift code to normalize y'. The minimum of the two is saved and compared with the code for z' and so on.... Upon completing the transformation of the point, the Sense register contains the minimum shift code of the four elements. All four elements will be normalized by the MAP normalizer according to the minimum shift code in the Sense register.

Figure 5.3-21 is a block diagram illustrating the Normalize Sense logic. The 1's complementor examines the upper sign bit (bit 23) and complements bits 22-0 if the number is positive; therefore, both positive and negative numbers are treated the same by the sense logic. The sense logic calculates the left shift code required to properly normalize

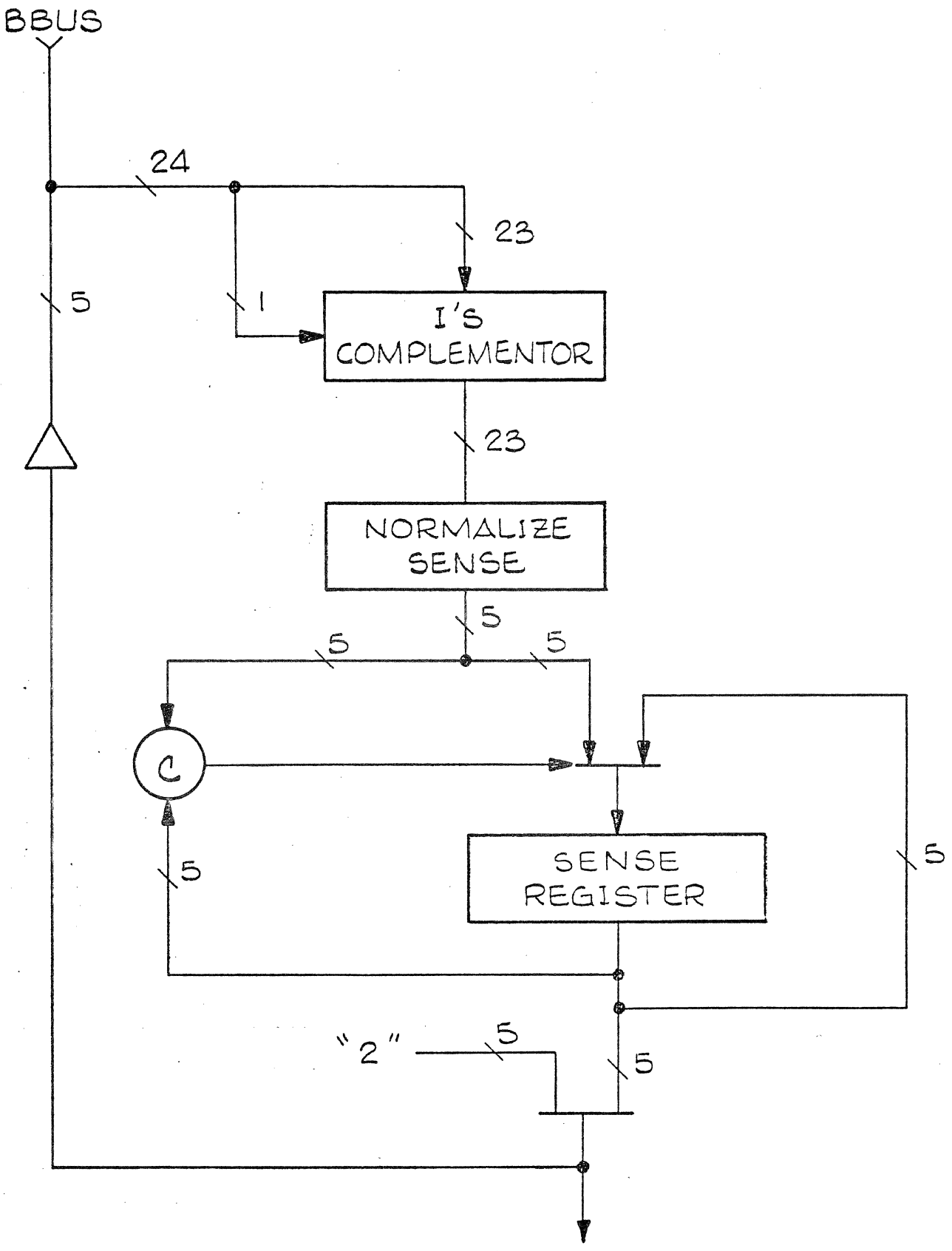


FIGURE 5.3-21
NORMALIZE SENSE BLOCK DIAGRAM

the number. Notice the comparator and Sense register mentioned in the preceeding paragraph. The Normalize Sense logic is implemented on the 195114-100 card (see sheets 1,2,4, and 5 of the logic drawings).

5.3.5.4 Normalizer

The MAP's normalizer receives a 5 bit shift code and a 24 bit number to be normalized. The shift code is decoded, and the input number from the MRA register is shifted left accordingly. Therefore, a shift code of 5 will left shift a 24 bit number 5 places and bits 4-0 of the result (NMR) will contain zeros. The normalizer is implemented on the 195113-100 card (see sheets 1-5 of the logic drawings).

5.3.5.5 Array Multiplier

The Array Multiplier achieves high-speed multiplications by using purely combinational logic. This circuitry consists of an array of multiplier and adder chips. When presented a multiplicand and a multiplier, the circuitry produces a product without use of any additional control signals. The multiplier chips (AM25S05) implements Booth's Algorithm which is discussed thoroughly in the Schottky and Low-Power Schottky Bipolar Memory, Logic and Interface Manual published by Advanced Micro Devices, Inc. For an in depth understanding of binary multiplication read pages 5-54 through 5-63 of the above mentioned manual.

The Array Multiplier receives a 16 bit multiplier (NMR) and a 20 bit multiplicand (MDA). The product is truncated to 24 bit. This 24 bit product includes 3 sign bits (bits 23-21).

The multiplier is input to the y(15-0) lines of the 195111-100 and 195112-100 cards. The multiplicand is input to the X(19-0) lines. The Array Multiplier block diagram (195111-900) illustrates the MSB and LSB sections of the array. Also note the partial product adders illustrated in the block diagram.

5.3.5.6 Reciprocation

The Matrix Arithmetic Processor (MAP) must be capable of performing division. Consider the perspective equation:

$$x_s = \frac{x'}{w'} \cdot \frac{Vr-Vl}{2} + \frac{Vr+Vl}{2}$$

where: x_s = element of the transformed point (x', y', z', w') put into perspective screen coordinates.

Vr, Vl = right and left viewport boundaries in screen coordinates.

The division required to calculate $\frac{Vr-Vl}{2}$ and $\frac{Vr+Vl}{2}$

can be accomplished simply by performing a right shift of one bit on the quantities Vr-Vl and Vr+Vl. However, x' and w' are not whole numbers; therefore, the arithmetic division function must be employed to calculate x_s.

The equation can be rewritten as follows:

$$x_s = \frac{x'-w'}{-2w'} \cdot (Vl-Vr) + Vr$$

In general, to implement the divison of $\frac{a}{b}$, the MAP takes the reciprocal of b and multiplies it by a:

$$\frac{1}{b} \cdot a = \frac{a}{b}$$

Therefore, to solve the perspective equation, the MAP finds the reciprocal of -2w and multiplies it by (x'-w').

Newton's Method of Reciprocation is used by the MAP to find the reciprocal (R) of the number (D). The equation is:

$$R2 = 2R1 - R1^2.D$$

Where R1 = a first guess for the reciprocal R.

R2 = the calculated, more accurate, second approximation of R.

In order to ensure convergence, D must be normalized:

$$.5 \leq D < 1 \text{ or } -.5 \geq -D > -1$$

The numerator must be normalized the same amount as D to ensure the proper answer. That is, if D is shifted N bits in normalization, the numerator A must also be shifted N bits since:

$$\frac{A}{D} = \frac{A.2^N}{D.2^N}$$ (normalization is equivalent to multiplication by a power of 2)

Now the equation used in Newton's Method can be derived as follows:

$$R = \frac{1}{D}$$

$$D = \frac{1}{R}$$

therefore:

$$\frac{1}{R} - D = 0$$

and we can write:

$$f(R) = \frac{1}{R} - D = 0$$

Now using Taylor' method of expanding polynomials:

$$F(x) = f(a) + \frac{f'(a)}{1!} \cdot (x-a) + \frac{f''(a)}{2!} \cdot (x-a)^2 + \cdots\cdots$$

$$+ \frac{f^{(n-1)}(a)}{(n-1)!} \cdot (x-a)^{n-1} + \frac{f^n(a)}{n!} \cdot (x-a)^n$$

and disregarding all terms higher than the first order and also equating a=R1 (first guess), we can write f(R) as:

$$f(R) = f(R1) + f'(R1) \cdot (R-R1) = 0$$

Now the derivative of f(R1) is f'(R1).

$$f'(R1) = f' \frac{1}{R_1} - D = \frac{-1}{R_1^2}$$

and if we equate $R=R_2$ (R_2= second approximation), then we can write:

$$f(R_2)= \frac{1}{R_1} -D + \frac{-1}{R_1^2} \cdot (R_2-R_1)=0$$

$$= \frac{1}{R_1} - D - \frac{R_2}{R_1} + \frac{1}{R_1}$$

$$= \frac{2}{R_1} - D - \frac{R_2}{R_1}_2 = 0$$

$$=2R_1 - R_2 - R_1^2 \cdot D=0$$

and solving for R_2 we have:

$$R_2 = 2R_1 - R_1^2 \cdot D$$

which is Newton's equation.

The hardware uses two ROMs, the R_1 ROM and the R_1^2 ROM. Each ROM is 256 words. Both ROMs are addressed by D; therefore, to find the reciprocal (R) of the number D, the hardware simply has to address the ROMs by D, form two terms $2R_1$ and $-DR_1^2$, and sum these terms. The ROMs are located on the 195113-100 card (see sheet 6 of the logic drawing).

5.3.5.7 MAP Control Store

The MAP Control Store consists of a ROM (RAM optional) controller which implements the MAP Algorithm. Figure 5.3-22 is a block diagram illustrating the MAP Control Store. The 256 word x 96 bit ROM output is buffered in the DOIT register which drives the control lines in the MAP. Sixteen bit sections of the DOIT register may be gated to the BBUS to enable reading the contents of the DOIT. In the case of the optional RAM controller, the Control Store may be loaded during maintenance mode by loading the DOIT and writing the DOIT contents into the RAM.

The next state of the ROM algorithm is determined in one of six ways:

1. If going to the next vector state, 377 gated to Control Store address.

2. If in subroutine return state, subroutine address gated to Control Store address.

3. If in main Dispatch state, Dispatch ROM gated to Control Store address.

4. If in a non-dispatch state, Next Buffered Address field of the DOIT gated to the Control Store address.

5. If in a normal dispatch state, dispatch

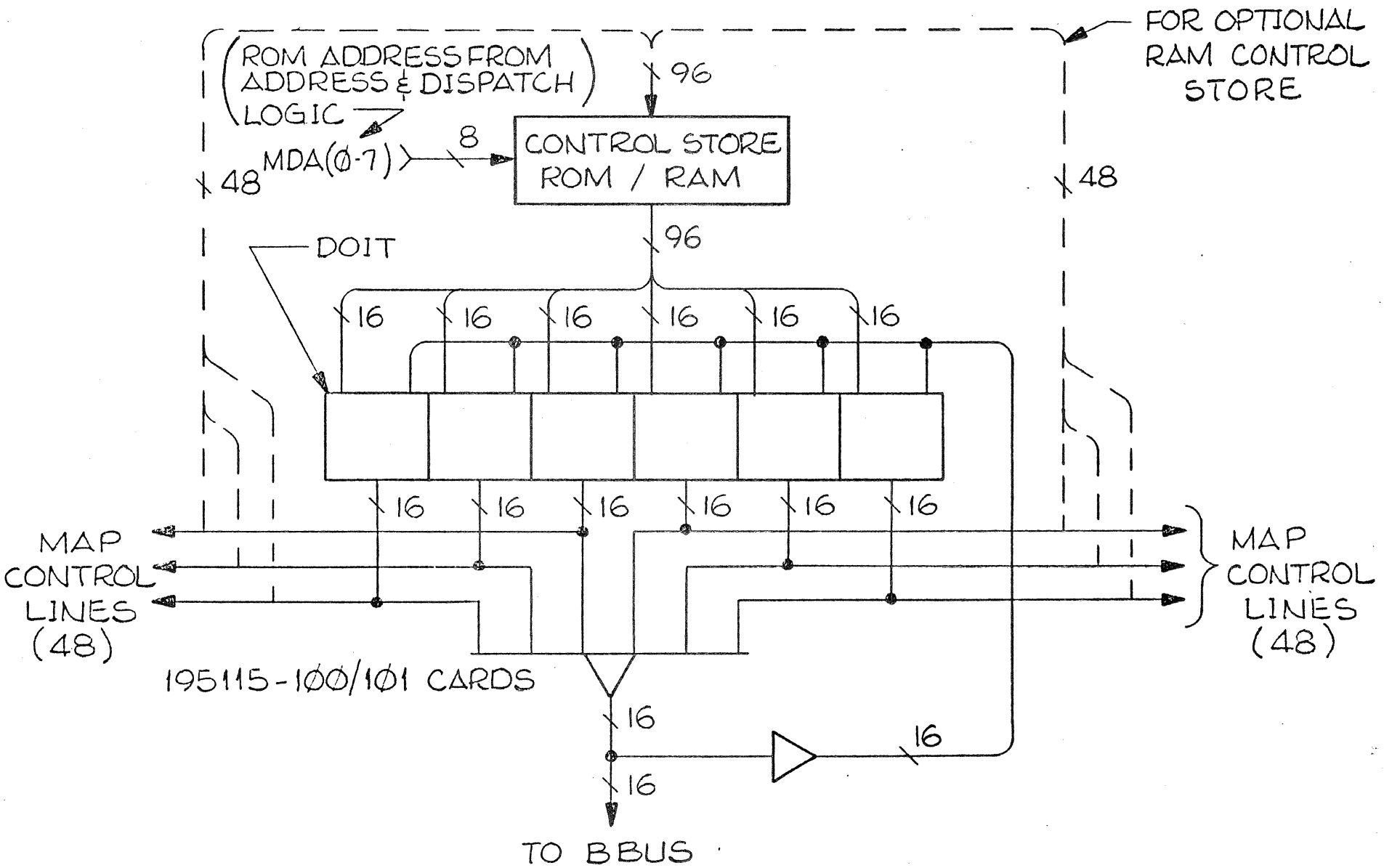


FIGURE 5.3-22 CONTROL STORE BLOCK DIAGRAM

condition (one bit) exclusive ored with LSB of DOIT's (NBADR) and results gated to Control Store address.

6. If in maintenance mode, MAP Maintenance PROM address register gated to Control Store address.

The next state determination scheme is illustrated by the Control Store address and Dispatch Block Diagram of Figure 5.3-23.

If the MAP is in a non dispatch state, the next state base address (NBADR) from the DOIT is selected to address the Control Store's next state. If the MAP is in a dispatch state, the LSB of NBADR is exclusive ored with a selected dispatch condition, then the result plus the other seven bits of NBADR are selected to be the next state. The dispatch condition is selected by four DOIT bits determined by the present state. The MAP algorithm calls subroutines by dispatching to the subroutine and saving the NBADR in the Subroutine Return Address register. To return from the subroutine, the saved NBADR from the return register is selected to be the next state address into the Control Store. State 377, the Next Vector state, is hard wired into the 4 to 1 selector. To enter the Next Vector state, 377 is selected to address the Control Store. A main dispatch ROM is selected when dispatching from the main dispatch state where the current RSR command is decoded. During the

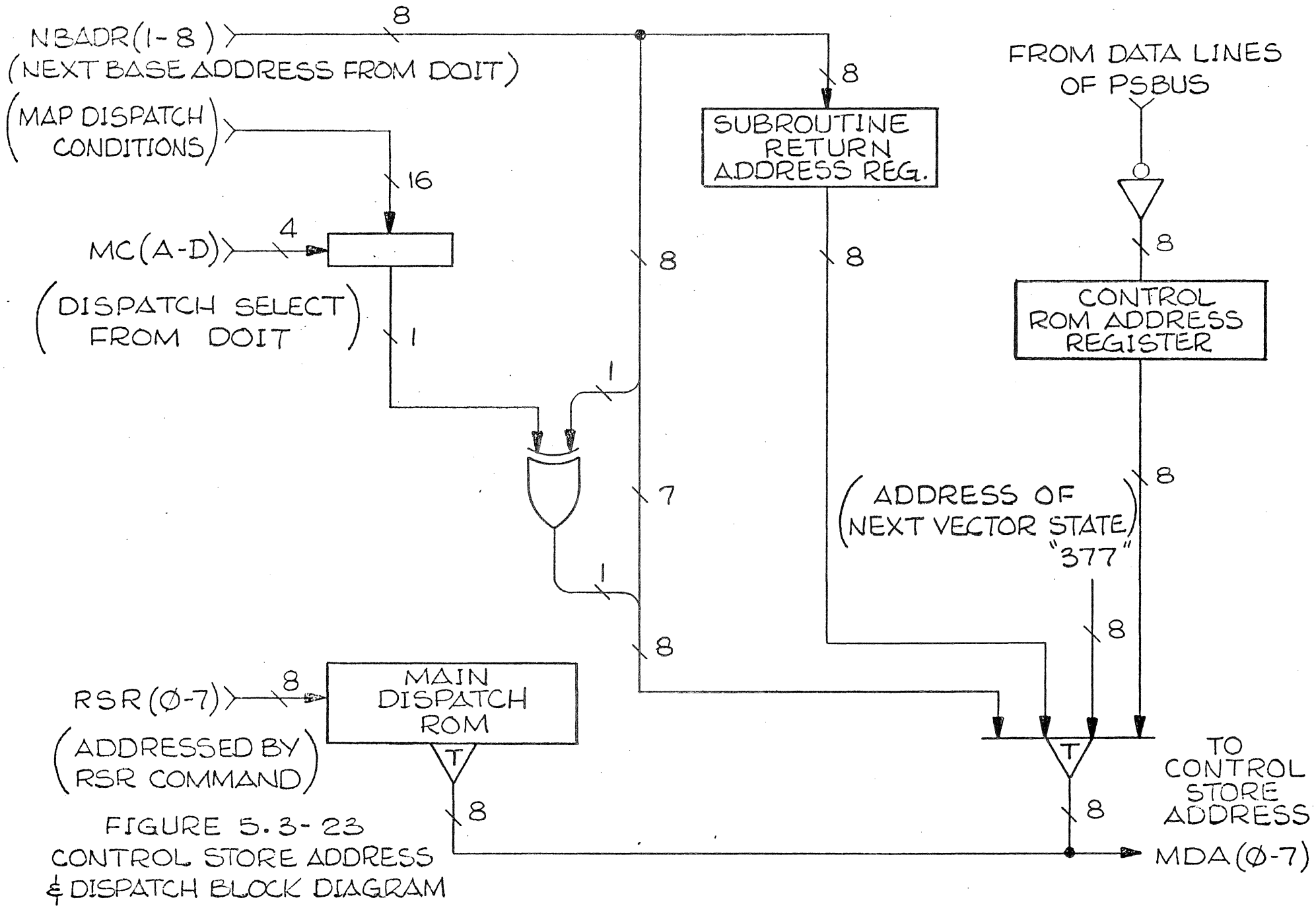


FIGURE 5.3-23
CONTROL STORE ADDRESS
& DISPATCH BLOCK DIAGRAM

main dispatch state, the output of the 4 to 1 selector is disabled while the dispatch ROM drives the address to the Control Store. During maintenance mode, the next state is determined by selection of the Control ROM Address register which is loadable from the PSBUS.

The logic which implements the dispatch scheme except for the dispatch selector is illustrated on the 195118-600 drawing, sheets 4 and 5. The 16 to 1 dispatch selector is located on 195117-600, sheet 3.

5.3.5.8 MAP System Clock

The MAP System Clock runs in sync with the Picture System's clock; however, unlike the Picture System's Clock, the MAP clock can be stopped by a hold condition. There are seven hold conditions which will halt the MAP clock. They are:

1. MAP Input FIFO is empty and the MAP is in a read FIFO state.

2. JUMP command executed while the MAP is in Passive input mode.

3. Upon a HIT Request while the HIT HOLD bit of the MSR is set.

4. The MAP in the "Get RSR" state and the RSR HOLD bit in the MSR is set.

5. Upon MAP execution of a RSR HALT command.

6. The MAP in the "Next Vector" state and the VEC HOLD bit of the MSR is set.

7. The MAPHIT bit of the MMSR is set.

Conditions 2,4, and 5 halt the MAP clock in the "Get RSR" state. Conditions 3 halts the clock in either the "Next Vector" or "Get RSR" state.

These programmable halt conditions enable the troubleshooter to check and examine the MAP at key points in the MAP algorithm.

For example, condition 1 may be used to examine the input FIFO's output to the BBUS. Condition 2 is an error condition, since the MAP should not be programmed to execute JUMP commands unless the MAP Input controller is active. Condition 3 may be set up to check which data caused a HIT, where in the input file the hit occurred, etc... Conditions 4 and 6 may be used to examine the current RSR command. Condition 7 may be used to single step the MAP as detailed in section 5.3.5.9.

The logic, which implements the MAP clock and hold conditions, is located on the 195117-100 and 195121-100 cards. The PICTURE SYSTEM clock is input on sheet 4 of the 195117-600 drawing (ARBAPSCLK connector 72). The PICTURE SYSTEM clock

is gated with *HOLD from the 195121-600, sheet 5, to generate *MAPCLK. *HOLD is the OR function of the 7 hold conditions which are implemented on sheets 4 and 5.

5.3.5.9 MAP Maintenance Structure

The MAP is equipped with a maintenance structure to enable the troubleshooter to do the following:

1. access the Control Store address
2. access the Control Store DOIT register
3. read the main communication bus of the MAP, the BBUS

Four PSBUS addressable registers are implemented in the maintenance structure. These registers are detailed in the PS2 Reference Manual (pp 2-65 through 2-70). They are:

1. MAP Maintenance Status Register (MMSR)
2. MAP Maintenance Repeat Status Register (MMRSR)
3. MAP Maintenance PROM Address Register (MMPAR)
4. MAP Maintenance B-BUS Register (MMBUS)

The MAPMNT bit in the MMSR is set to put the MAP in maintenance mode. During maintenance mode, the troubleshooter has access of the Control Store address. Also, in maintenance mode the contents of the DOIT register may be examined

or modified. This feature enables the troubleshooter with direct access to control signals within the MAP.

The MAPHLT bit is set to stop the MAP clock, therefore leaving the MAP in a halted condition. With the MAPHLT bit set and the MAPMNT bit clear, the BBUS of the MAP may be examined by reading the MMBUS register. This feature enables the troubleshooter to examine registers, counters, other MAP busses, etc...which are gated to the BBUS during states of the MAP algorithm. Also, with the MAPHLT bit set and the MAPMNT bit clear, the troubleshooter may advance state in the MAP by setting the MAP SSTEP (single step) bit of the MMSR. With this feature, the entire MAP algorithm can be configured in a single step fashion. To sequence states in the MAP algorithm in single step mode, the MAP Input Controller should be programmed to actively fetch data from a PS Memory file and load the input FIFO. This data file in PS Memory must be comprised of RSR commands and associated data to steer the MAP through desired functions of the algorithm. Actually, three methods of controlling the MAP are optional to a troubleshooter. They are:

1. Load the MAP input FIFO RSR commands and associated data, then single step the MAP through functions in the MAP algorithm.

2. Program the Control Store DOIT register while in maintenance mode, then clear maintenance mode and issue a clock pulse to the MAP.

3. Write the Control Store while in maintenance mode with a specific test algorithm, then clear maintenance mode and exercise specific MAP functions at full speed.

Method 1 of the above is suggested unless the troubleshooter has an indepth understanding of the MAP hardware. Following is an example of using method 1 to troubleshoot a MAP failure.

Suppose the MAP diagnostic, QSD017, detects an error and prints out the following error message.

1:	ADDRESS	EXPECT	RECEIVE
	0	144001	144001
		67521	27521

As indicated by the test description found in the PS2 Hardware Diagnostic Manual, the second word (bit 15) is in error.

The troubleshooter can isolate the faulty component by setting the MAP to maintenance mode, writing a LOAD/STORE sequence with the appropriate data into the MAP input FIFO, clearing maintenance mode, then single step the MAP while probing the applicable data lines.

A LOAD/STORE and data sequence need be set up in PS Memory. If properly initialized, the MAP Input Controller will actively fetch the commands and data from PS Memory. The command and

data sequence loaded into PS Memory for this example problems would be as follows:

LOAD	(extended RSR LOAD command)
ADDRESS	(address in MAP where to load data)
DATAWORD1	
DATAWORD2	
STORE	(extended RSR STORE command)
ADDRESS	(address in MAP where to get data)
HALT	(RSR MAP HALT command)

ADDRESS is Ø in this case, and the DATAWORD's are 144001 and 67521. After loading PS Memory, the MAP Input Controller should be programmed to actively fetch data from the PS Memory location pointed to by the MAIA, MAP Active Input Address register. The MAPHLT bit should be set and the MAPMNT bit cleared (both previously set by PS Reset). The trouble-shooter may issue single clock pulses by setting the MAP SSTEP bit in the MMSR; therefore, the MAP will advance state in the algorithm. After 4 clocks, the MAP will be in state 307 (see MAP algorithm) with the MAP's data memory address set up and DATAWORD2 on the memory inputs. At this time, the memory inputs should be checked (by probing) for good data. If good, the logic up to the memory input must be functioning properly and may be ruled out as a possible problem. Another clock pulse will write the memory and put the MAP into the "Get RSR" state (state Ø).

To check the hardware from the memory back to the PSBUS, six more clock pulses should be issued; therefore, the MAP will be in state 314 of the STORE routine with DATAWORD2 output from the data memory. At this time, the data lines from memory to the BBUS and from the BBUS to the output register should be checked. If these data paths are good, another clock will strobe the data into the output register and data lines to the PSBUS should be checked for failure. If everything checks out, the problem must be dynamic rather than static; therefore, the troubleshooter should set up a dynamic test loop and chase the problem with an oscilloscope (loop on error in the diagnostic program).

The command and data file input to PS Memory may be set up with the PICTURE SYSTEM Diagnostic Debugging Technique (QSDDT) program. This program also provides the troubleshooter with access to all PSBUS addressable register; therefore, from the console terminal, the MAP may be initialized and single stepped.

The troubleshooter may generate his own program to load PS Memory, initialize the MAP, and issue clock pulses. The program may be generated from the flow chart of Figure 5.3-24.

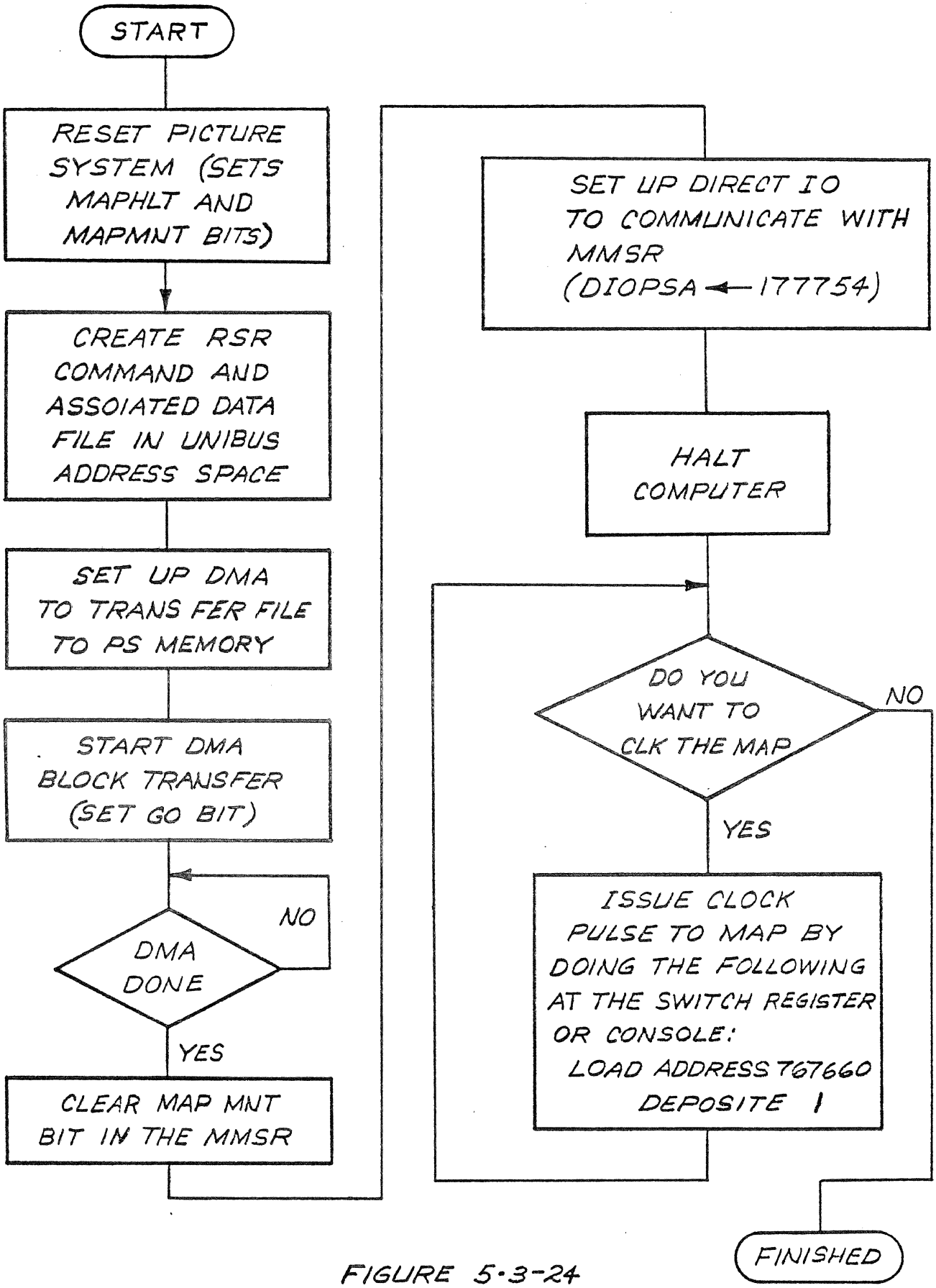


FIGURE 5·3-24

SINGLE STEP MAP FLOW CHART

5.3.6 MAP Output Formatter

The MAP Output Formatter functions as either an active or passive device depending upon the state of the MAO bit in the MSR register. When functioning as an active device the Formatter gains control of the PSBUS and initiates output of data to the passive device addressed by the MAP Active Output Address register (MAOA). When functioning as a passive device the Formatter passively waits to be addressed by an active device, then transfers output data to the PSBUS. In either mode the Formatter's Output Sequencer waits in Idle state until the MAP has output data ready. The MAP then commands the Formatter to output. The Output Controller examines the state of the MODE bits in the MSR along with the current status of the RSR register to determine the mode of output to be performed. The following output modes are possible.

2D UNFORMATTED	- 16 bit X word and 16 bit Y word output.
3D FORMATTED	- 2-16 bit words output with format compatible for Line Generator input.
4D UNFORMATTED	- 16 bit X, 16 bit Y, 16 bit Z, and 16 bit W output.
4D HIGH PRECISION	- 16 bit RSR, 16 bit X, 16 bit Y, and 16 bit Z output.

Upon determination of one of the 4 output modes, the Output Controller dispatches to the appropriate section of the output algorithm. If active, the Formatter gains control of the PSBUS output data to the PSBUS, and the sequencer returns to the Idle state for another command from the MAP. If the Formatter is passive, an active device can not address it until it has received a output command from the MAP and the sequencer is in an output state.

5.3.6.1 Output Control Sequencer

The Output Control Sequencer is implemented as a ROM state sequencer on the 195118-600 card (see sheet 2 of the logic drawings). The sequencer transverses the state diagram depicted in Appendix B of this manual. State 01 is the main dispatch state with four possible branches. All other dispatch states have only two branches. Two dispatch code bits from the current state DOIT register select the appropriate dispatch controls input to IC location 30 (see 195118-600, sheet 2) which passes the selected dispatch control to the two LSB's of the next state address of the ROM.

Initially, a reset clears the DOIT register which generates a next state address of Ø. When the reset signal is cleared the next clock pulse generates *CLKDOIT pulse which puts the sequencer in state Ø, the IDLE state. The IDLE is maintained

until the MAP signals "data ready for output" by asserting *MOUTPUTSET which latches the current RSR status (see IC location 25 on sheet 5) and the sequencer enters the main DISPATCH state. In the DISPATCH state, DISPA and DISPB are gated to the two LSB's of the next state address of the ROM. On the next clock, the sequencer dispatches to one of the four output mode sections of the state diagram. If active, the PSBUS is requested and data output according to the output mode. Upon output completion, the sequencer returns to the IDLE state. If passive the sequencer waits in an output state until addressed by an active device, then passes output data to the PSBUS. Whether active or passive, the sequencer tranerses the same states in the algorithm; however, if passive it never requests the PSBUS. It merely waits in the Request states until it is addressed, then gates output to the PSBUS.

5.4 PICTURE SYSTEM Memory (PS MEMORY)

The potential address space in the PICTURE SYSTEM ranges from Ø to 64K. Of this address space 256 words are reserved as the System Control Block (SCB). This block addresses control registers, status registers, input ports, etc...which in general control the system. This SCB is located in address 177400 through 177777 of PS address space. The remaining addresses, (0-177377) are available for PS Memory. PS Memory is expandable in blocks of 16K words, 16 bits per word.

PS Memory may be used by the user in a number of ways. for instance, a section of the memory may be used to buffer an RSR command and data file as input to the MAP. Another section of PS Memory may be used to buffer output from the MAP. This buffer may also serves as a refresh buffer being input to the Picture Generator. Another application may not require the MAP; therefore, data from the Picture Controller may be buffered in PS Memory while the Picture Generator actively accesses the memory. In both of the above examples, a device called the Refresh Controller coordinates the memory update process with the Picture Generator's refresh process. Section 5.5 of this manual details the operation of the Refresh Controller.

PS Memory is a passive device interfaced to the PSBUS. Only active devices can communicate with the memory. The memory is implemented with 4K MOS Memory chips; therefore, access

requires sequencing 6 bit row and column addresses with appropriate write or read signals to the Memory chips. The memory system has two ports which enables overlap of memory accesses. This overlap results in a minimum access time of 450 ns. When requested by an active device to perform a read access, the memory always responds (if not busy) with a grant to free the PSBUS and a simultaneous deferred signal which alerts the active device to wait while the memory port controller accesses the desired locations. Upon completion of the access, the memory issues a second grant to the waiting device. This second grant acknowledges completion of the read access and the active device can now take valid data from the PSBUS.

The memory may be busy when requested by an active device. In this case, the memory saves the request in a FIFO, acknowlegdes the active device with a grant to free the PSBUS, and simultaneously signals the device with a memory busy signal. This alerts the device to wait for another grant (generated from the saved request in the FIFO) before passing the address (and data if doing a write) to the memory. If the access is a read, the memory responds with the deferred signal simultaneous with this second grant. The device waits for a third grant signifying completion of the memory cycle.

The PS Memory System block diagram (195140-900) illustrates four parts of the memory system.

1. FIFO and Port Latch
2. Port Arbitration and Port Controllers
3. Memory Timing and Sequence Controllers
4. Memory

5.4.1 FIFO and Port Latch

The PS Memory's FIFO and Port Latch are used to buffer requests from an active device. The FIFO is used to que up requests from active devices desiring access of the PS Memory. A Request from an active device is saved in the FIFO if one of the two following conditions exist.

1. The FIFO is not empty. In this case, the request must be qued since other request in the FIFO need service first.

2. If both port controllers are currently busy. In this case the request must be qued until an idle port control can service it.

When a port controller begins servicing a read request, the request (whether from the FIFO or from the PSBUS) is latched in the Port Latch. Upon completion of the read, the saved request in the Port Latch is output as a grant to the active device signifying valid data is on the PSBUS.

Figure 5.2-2 in the PSBUS section of this manual illustrates the request/grant timing relationships for various types of memory accesses. The FIFO and Port Latch implementation is illustrated on sheets 4 and 5 of the 195107-600 logic

drawing.

5.4.2 Port Arbitration and Port Controllers

The dual port structure of the memory system requires an arbitration structure to pass a request from active devices to an available Port Controller. The port controllers manage the loading of data in and out of a memory port. The following rules are applied by the arbitration scheme to determine which of the two port controllers manage a current memory request.

1. If both port controlers are idle, controller A gets the current request.

2. If one controller is idle and the other one busy, the idle controller takes the request.

3. If both controllers are busy upon a request, the request is saved in the FIFO and the next idle port controller takes the qued request from the FIFO.

Both port controllers sequence through the state diagram illustrated in Figure 5.4-1. The idle state previously mentioned is state Ø. States 3 and 2 are executed to perform a read operation, and states 4 and 1 are executed to perform a write operation. To perform a read - modify - write operation, states 7,6,4, and 1 are executed. States

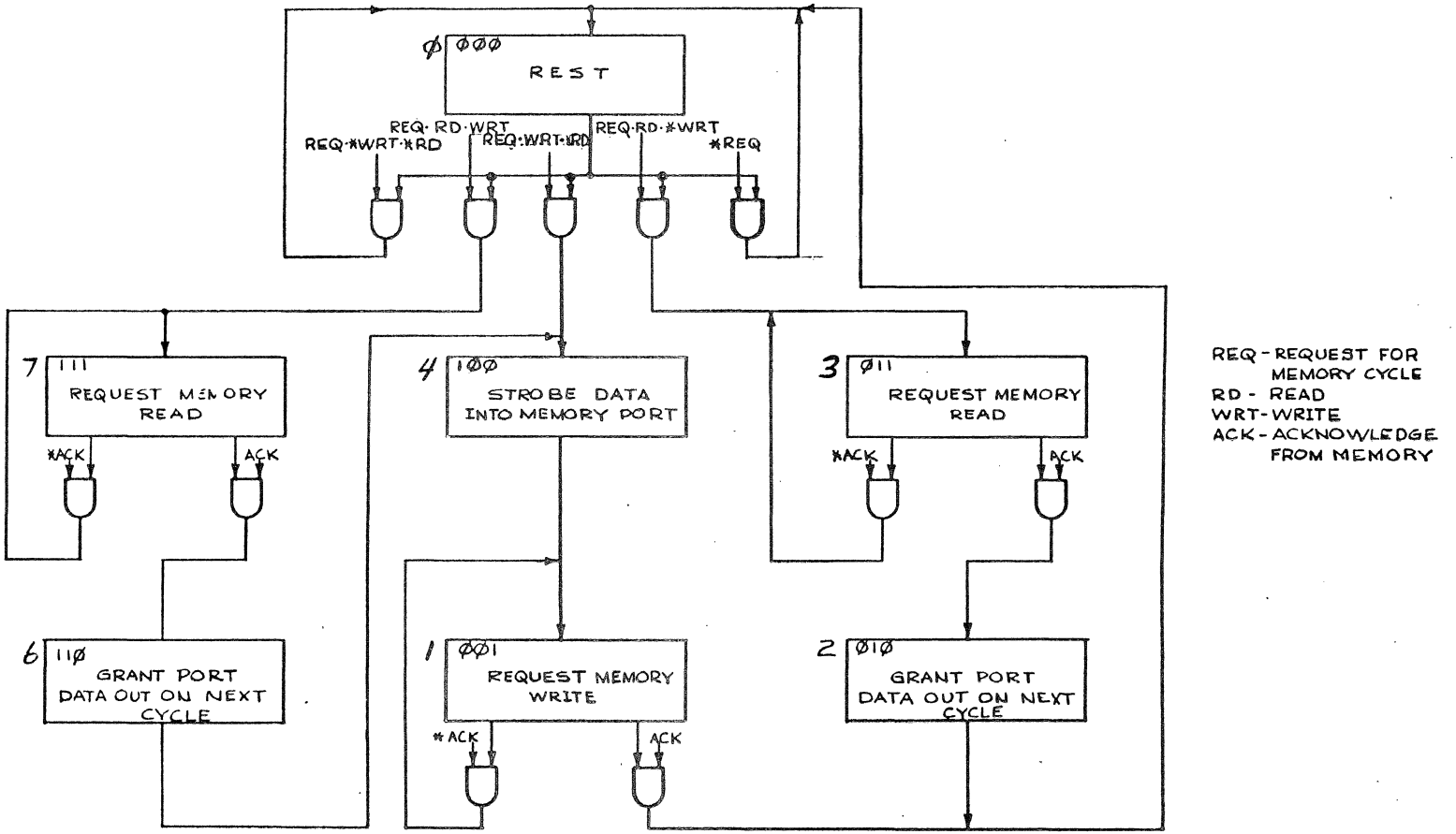


Figure 5.4-1

Port Controller State Diagram

7 and 3 request a memory read cycle and wait for the memory sequencer to acknowledge. Similarly, state 1 requests a memory write cycle and waits to be acknowledged.

Each port controllers has a unique address latch. The appropriate latch is loaded upon a port controller's exit from the idle state. Likewise, each controller has an input and output data latch. The appropriate input data latch is loaded upon exit from state 4, and the appropriate output data latch is loaded upon exit from states 2 or 6. The address latches are located on sheet 3 of the 195141-600 drawing. The data latches are located on the memory card (see 195143-600 drawing). Actually, the memory card implements three ports of which only two are used in the PICTURE SYSTEM 2.

5.4.3 Sequence Controller

The Sequence Controller provides the necessary timing to initiate and execute memory cycles. The principle tasks of the sequencer are:

1. Sense memory cycle requests from the port controllers and initiate a memory cycle.

2. Acknowledge the appropriate port controller upon completion of a memory cycle.

3. Sequence the Row and Column address and appropriate

command signals to the memory during a memory cycle.

4. At necessary time intervals, refresh section of the dynamic MOS memory.

The Sequence Controller is implemented on the 195142-101 card. Sheet 2 of the logic drawing illustrates the implementation of task 1. The Schmitt Trigger, latch, and RC network combinations sense the memory cycle request from port controllers A and B. SAMPLE clock runs at 20MHz (50 ns square wave). T20 clock is 20 ns out of phase with SAMPLE. The sense latches sense for 20 ns. On T20 clock, their results are loaded into the register in bug location 64. If a sensed request is loaded into the register, the output of bug 54 (*BEGIN CYCLE) disables the clock to the register until the completion of the cycle (END CYCLE). The 74S00, 74S10, and 74S20 in locations 53,73, and 44 implement a priority scheme in case more than one sensed request is loaded into the register. Refresh has highest priority, then A, B, and C, respectively. Note in this system C is not used. Upon a request loaded into the register, *BEGIN CYCLE starts the sequencer on sheet 3. BEGIN CYCLE is shifted through the shift register in locations 43, 42, and 32 which generates the ROW Address Select (RAS), Column Address Select (CAS), etc...timing signals to the memory. At the time of CAS, the appropriate F/F on sheet 2 is set which acknowledges the cycle request from the appropriate port controller. The port controller clears its request which

clears the acknowledge F/F and the sense latch is enabled to sense the next request. The dynamic memory refresh sense latch is on sheet 4. The one shot in location 61 determines the refresh interval. The counter in locations 70 and 71 determines which section of memory is refreshed during a particular refresh interval.

5.5 Real Time Clock

The Real Time Clock coordinates the update and refresh processes in the Picture System. The two processes are coordinated by the generation of the following two Control signals:

1. Clock Interrupt to the Picture Controller
2. Sync pulse to the refresh devices

The clock interrupt signal enables the Picture Controller program to periodically check the status of the MAP and Refresh Controller to determine if the update process may be initiated. Upon receiving a clock interrupt, the Picture Controller executes a clock service routine which checks the following three conditions:

1. is the DMA Controller Idle
2. is the Picture Processor Idle
3. is the Refresh Controller Idle

If all three conditions are true, the Picture Controller may begin a new frame update process and a new frame refresh process. If only condition 3 is true, the P.C. may only start the refresh process. The Real Time Clock implements the clock interrupt and sync pulse by counting down two counters. The two counters (Count 1 and Count 2) are loaded

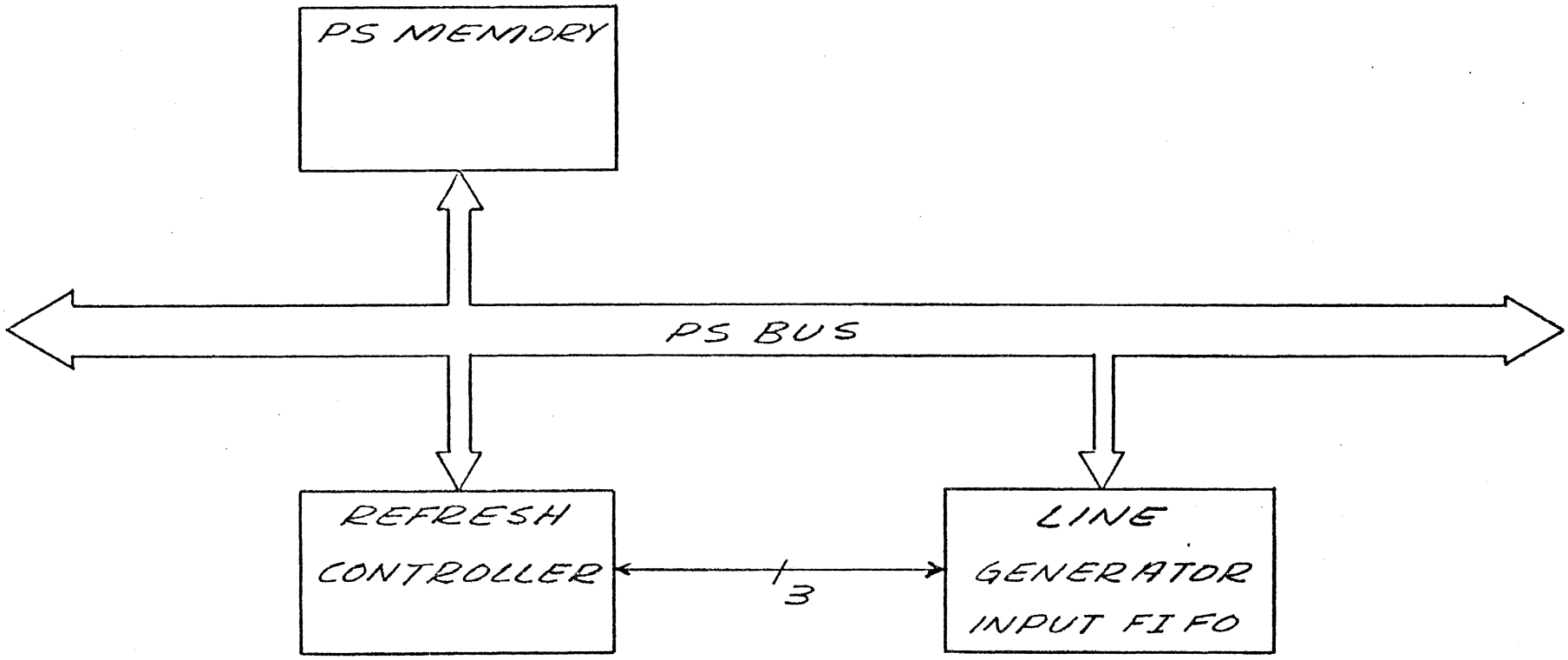
with the contents of two PSBUS writable registers (see drawing 195161-600, sheet 2). The clock signal to the counters is generated by ORing two 60 Hz, input signals which are out of phase by 180°. Therefore, the counters are clocked with a 120 Hz, signal. Upon overflow, the counters are reloaded. The overflow outputs are CARRY1 and CARRY2. If the SYNC bit of the status register RTCSR is clear, CARRY2 generates PSSYNC pulse to the refresh devices. CARRY1 generates an interrupt REQUEST (sheet 5). If the SYNC bit is set, the PSSYNC pulse and interrupt REQUEST occur coincident. This is very useful when coordinating the update and refresh processes while operating the Refresh Controller in automatic refresh mode.

5.6 Refresh Controller

The Refresh Controller is an active device which controls flow of data from PS Memory to the Line Generator Input FIFO. Figure 5.6-1 is a block diagram of the Refresh Controller/ PS MEM/and Line Generator interface. The Refresh Controller contains a start and a limit register. These two registers define a block of PS Memory to be dedicated as refresh buffer. Upon command the controller begins initiating read requests to PS Memory. When data is valid on the PSBUS, the Refresh Controller commands the Line Generator Input FIFO to take the data. When the Refresh Controller has completed the refresh process a stopped bit is set which is monitored by the program.

The Refresh Controller is equipped with eight, PSBUS addressable, control registers which implement refresh buffer segmentation, automatic refresh, write back to memory, and software control. The following eight registers are defined in detail in the PS2 Reference Manual, pp 2-75 through 2-82.

1. RFCSN - Refresh Current Segment Name
2. RFSN - Refresh Segment Name
3. RFAWA - Refresh Active Write Address
4. RFAWL - Refresh Active Write Limit
5. RFAIA - Refresh Active Input Address
6. RFAIL - Refresh Active Input Limit
7. RFASA - Refresh Active Start Address
8. RFSR - Refresh Status Register



REFRESH CONTROLLER/PSMEMORY/
LINE GENERATOR INTERFACE
BLOCK DIAGRAM

FIGURE 5.6-1

5.6.1 Refresh Control Command Words

At appropriate locations in the refresh buffer the user inserts Refresh Control commands to initiate action by the Refresh Controller. The Refresh Control commands are:

1. HALT
2. SEGMENT
3. LIGHT PEN

When executed, the HALT command stops the refresh process and sets the RFSTOPPED bit in the RFSR. The SEGMENT command specifies a segment name and directs how the Refresh Controller is to treat the segment. The LIGHT PEN command is used to direct the LIGHT PEN Controller in the LIGHT PEN Interface. These R.F. Control commands are detailed in the PS2 Reference Manual, pp 2-91 through 2-93.

5.6.2 Frame Synchronization

The Refresh Controller is synchronized to begin accessing the refresh buffer at the beginning of a frame period. When in automatic refresh mode, the controller looks for a pulse on the PSSYNC line. The pulse is generated at the beginning of a frame period determined by the frame time counter of the Real Time Clock card. If in program control mode, the controller waits for a start command from the program. To be

more precise, the Real Time Clock generates an interrupt for service at programmable frame time periods. This interrupt causes the program to check the condition of the Refresh Controller. If the controller is stopped (RFSTOPPED bit of RFSR set) and the Picture Processor has completed updating the refresh buffer, the program starts a new frame refresh (sets the RFSTART bit in the RFSR).

5.6.2.1 Arbitration of Refresh Devices

Some system configurations may include more than one Refresh Controller or other refresh devices. As an example, a system may configure a Refresh Controller and two Remote Terminal Interfaces. In this case a hardwired priority scheme gives the Refresh Controller highest priority. At the beginning of a frame refresh period, the Refresh Controller accesses the refresh buffer. When the Refresh Controller finishes, one of the Remote Terminal Interfaces begins refreshing and when finished the other begins. A new frame refresh will not be initiated until all refresh devices are finished refreshing the current frame.

5.6.2.2 Frame Sync State Machine

Refresh Control Frame synchronization is implemented with a four state machine. Each refresh device configured in the system contains this state machine. Figure 5.6-2 illustrates

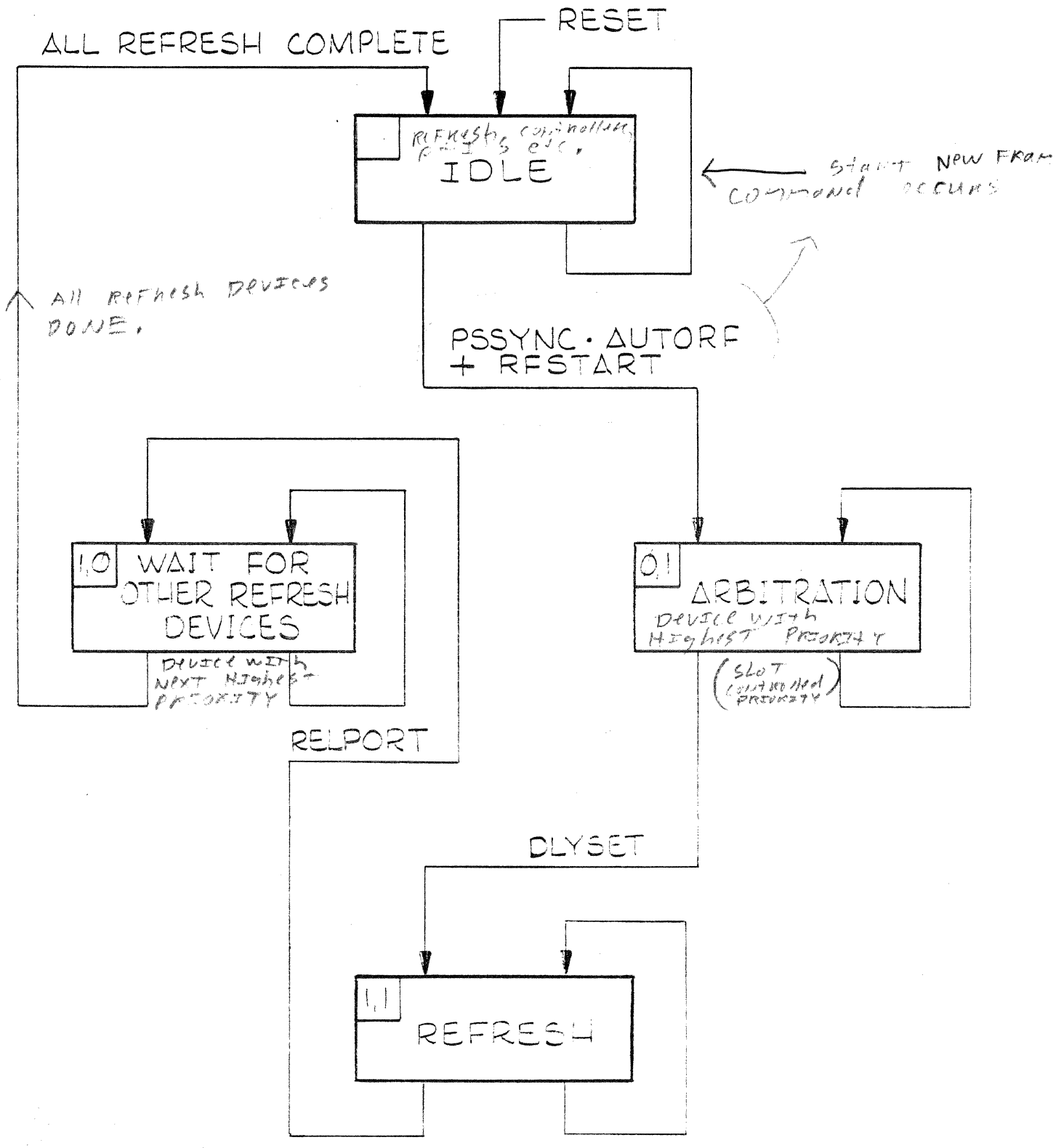


FIGURE 5.6-2 REFRESH CONTROLLER FRAME SYNC STATE DIAGRAM

the state diagram of this machine.

All refresh devices (Refresh Controllers and Remote Terminal Interfaces) wait in the IDLE state until a "start new frame" command is sensed. Upon receipt of the command all refresh devices enter the ARBITRATION state. Arbitration takes place and the device set up with the highest priority enters the REFRESH state and begins refreshing the appropriate (if more than one) Picture Generator. The device enters the WAIT FOR OTHER REFRESH DEVICES state when it is finished. The device with the next highest priority then enters the REFRESH state. When all devices are finished, they enter the IDLE state and wait for the next "start new frame" command.

The four state machine for the refresh controller is implemented on the 195151 card (sheet 4 of the logic drawing). The two F/F's called STA and STB determine the state, If in Auto Refresh mode, PSSYNC from the Real Time Clock generates a new frame pulse. The pulse inputs a Ø into the shift register. When the Ø is clocked to the Q_C output, SET is inserted, and the next clock sets STA and saves SET in the DLYSET F/F on sheet 1. If not in Auto Refresh mode, STA is set by the program. The machine is now in the ARBITRATION state. DLYSET inputs Ø into the shift register (bug 31) and DLYSET is cleared. On the next clock, unless another refresh device has higher priority, STB is set. Now the machine is in the REFRESH state which inserts *SEGBSY to the Refresh

Sequencer. *SEGBSY is equivalent to *GO in the Refresh Control algorithm (See Appendix C). During the Refresh state, Ø's are being shifted through the shift register. The Refresh Sequencer is busy accessing PS Memory and directing the data to the Picture Generator. Upon completion (running into the RFAIL register or executing a HALT command), the Refresh Sequencer asserts RELPORT (Release L.G. Port) which clears the STA F/F and puts the machine into the WAIT FOR OTHER REFRESH DEVICES state. Other devices now have a chance to refresh. If no other devices exist or when all other devices are finished the shift register fills up with 1's which clears the STB F/F and puts the machine back in IDLE.

5.6.3 Refresh Sequencer

The Refresh Sequencer performs the following tasks:

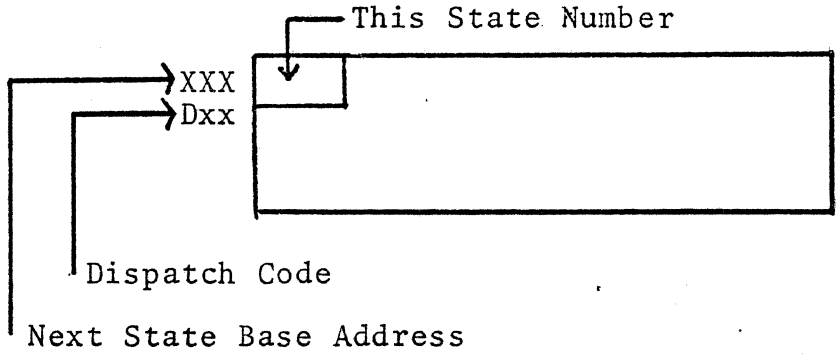
1. Waits for a GO command from the Frame Sync state machine.
2. Initiates PS Memory accesses.
3. Monitors the PSBUS for Refresh Control commands from the Refresh Buffer.
4. Executes Refresh Control commands. (Search for Segment Name, Skip Segments, HALT, ect.)
5. Commands the Line Generator Input FIFO to take data from the PSBUS.

The sequencer is implemented on the 195152 card as a ROM

state machine with 26 states (See Appendix B). The ROM is on sheet 2 of the logic drawing. Output from the ROM is buffered in a command register. The next state is determined by four next state bits in the command register and a dispatch code. The dispatch code selects one of eight conditions input to the 8 to 1 selector in location 51. The selected dispatch condition becomes the LSB of the next state address to the ROM.

APPENDIX A

MAP ALGORITHM STATE DIAGRAM



Next State ← (Next State Base Address) ⊻ Dispatch (low bit)

Example:
DISPATCH CODE = 14
NEXT BASE = 66
MDA<0 = 1
NEXT STATE = 67

*INPUT WAIT DISPATCH

Example:
Go to Input Wait if Dispatch condition is true.

DISPATCH CODES

	(XOR)	(INPUT WAIT)
0 =	ZERO	
1 =	ONE	
2 =	AC2=-1	
3 =	AC3=-1	
4 =	OUTPUT BUSY	
5 =	MRA<0	
6 =	OUT MULT	
7 =	OUT NORM MULT	
10 =	$\overline{\text{ZERO}}$	*MDA<0
11 =	$\overline{(\text{MDA}<0)\wedge\text{CHKNC}}$	*(MDA<0)∧CHKNC
12 =	RSR(0)	
13 =	DIVIDE ERROR	
14 =	MDA<0	
15 =	SV	
16 =	NC	
17 =	FIFO FULL	

ALL NUMBERS ARE OCTAL.

[CHKNC] FLAGS (3) ← CBUS(23)

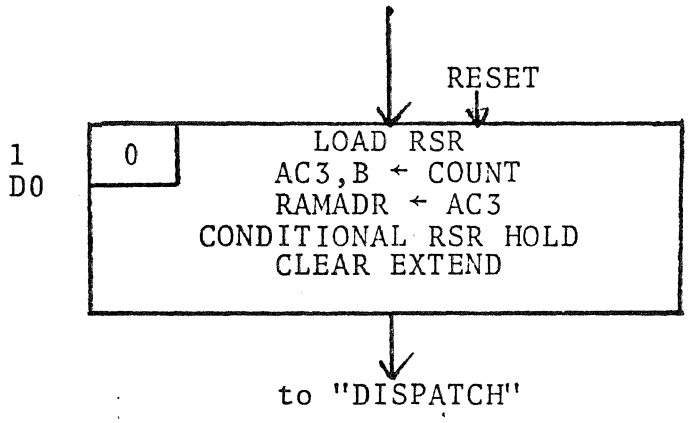
[NC] FLAGS (2) ← FLAGS (3)

[NORMLT] FLAGS (1) ← BBUSS (NORM) < SENSE REGISTER

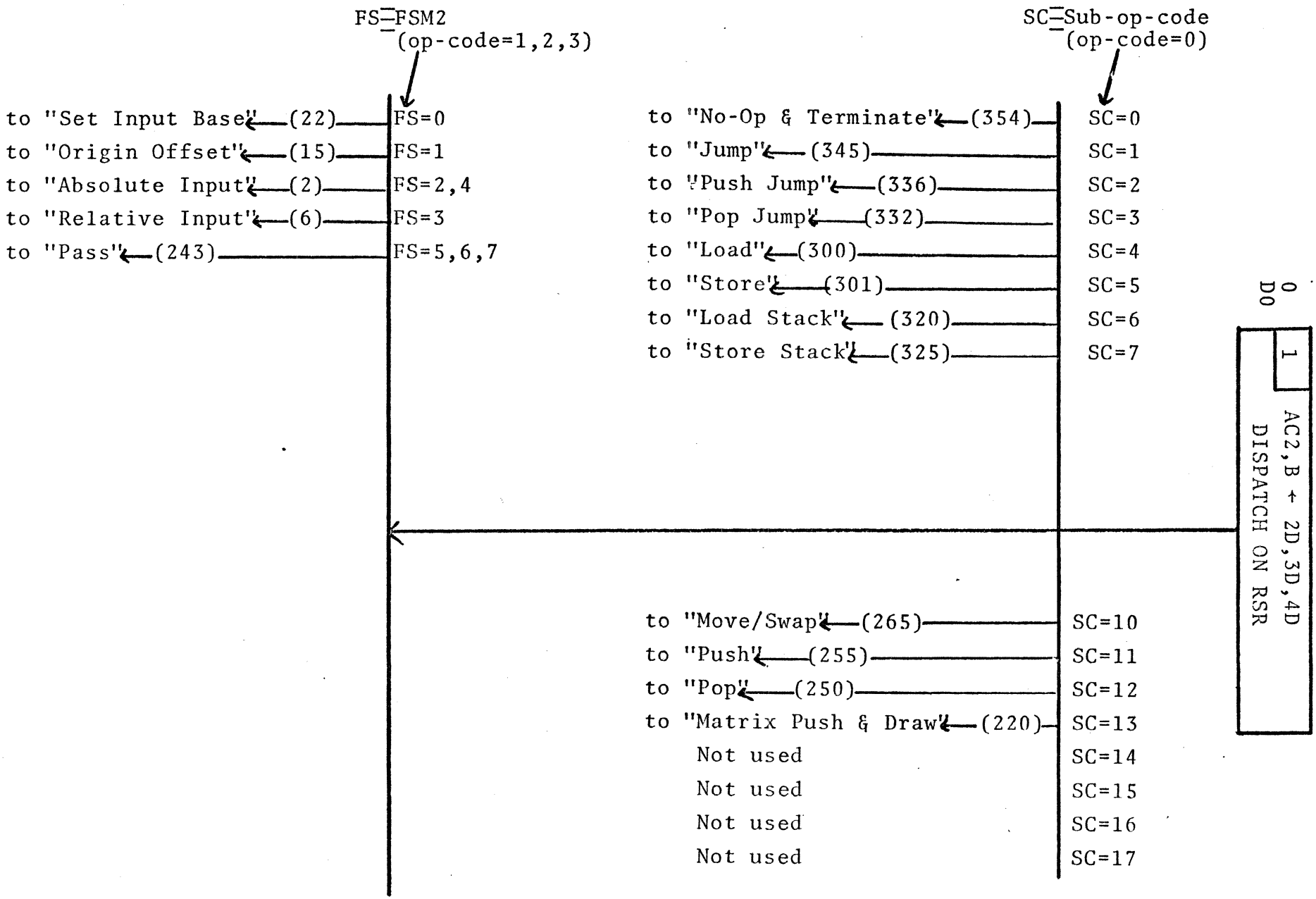
NV ← NC (new point is clipped)

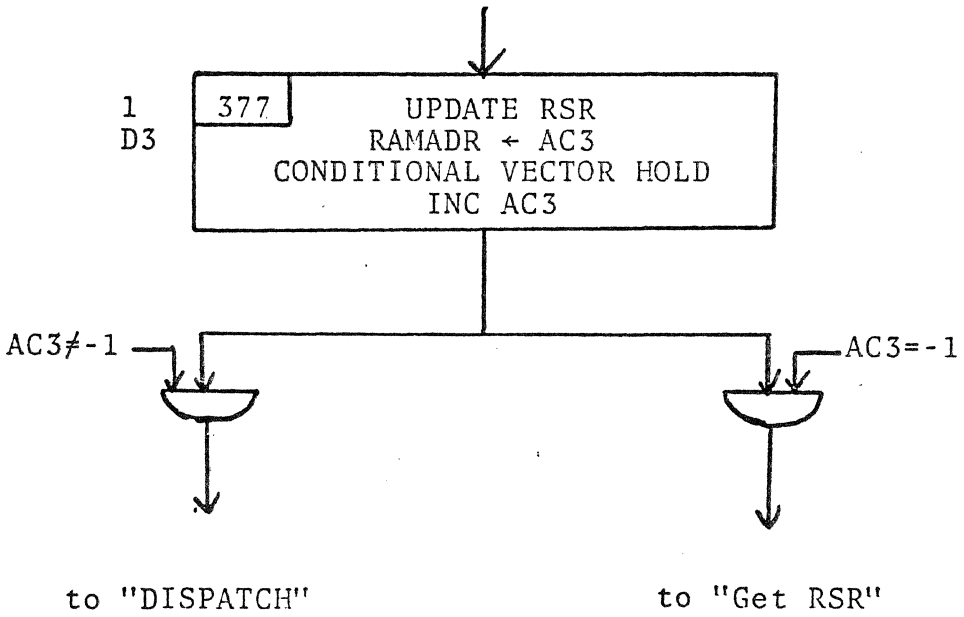
SV ← $\overline{\text{NC}}$ (old point is clipped)

DIVIDE ERROR ← NORMLT V (AC2=-1 Λ MDA < 0) V (MDA(23) ⊻ MDA (21))

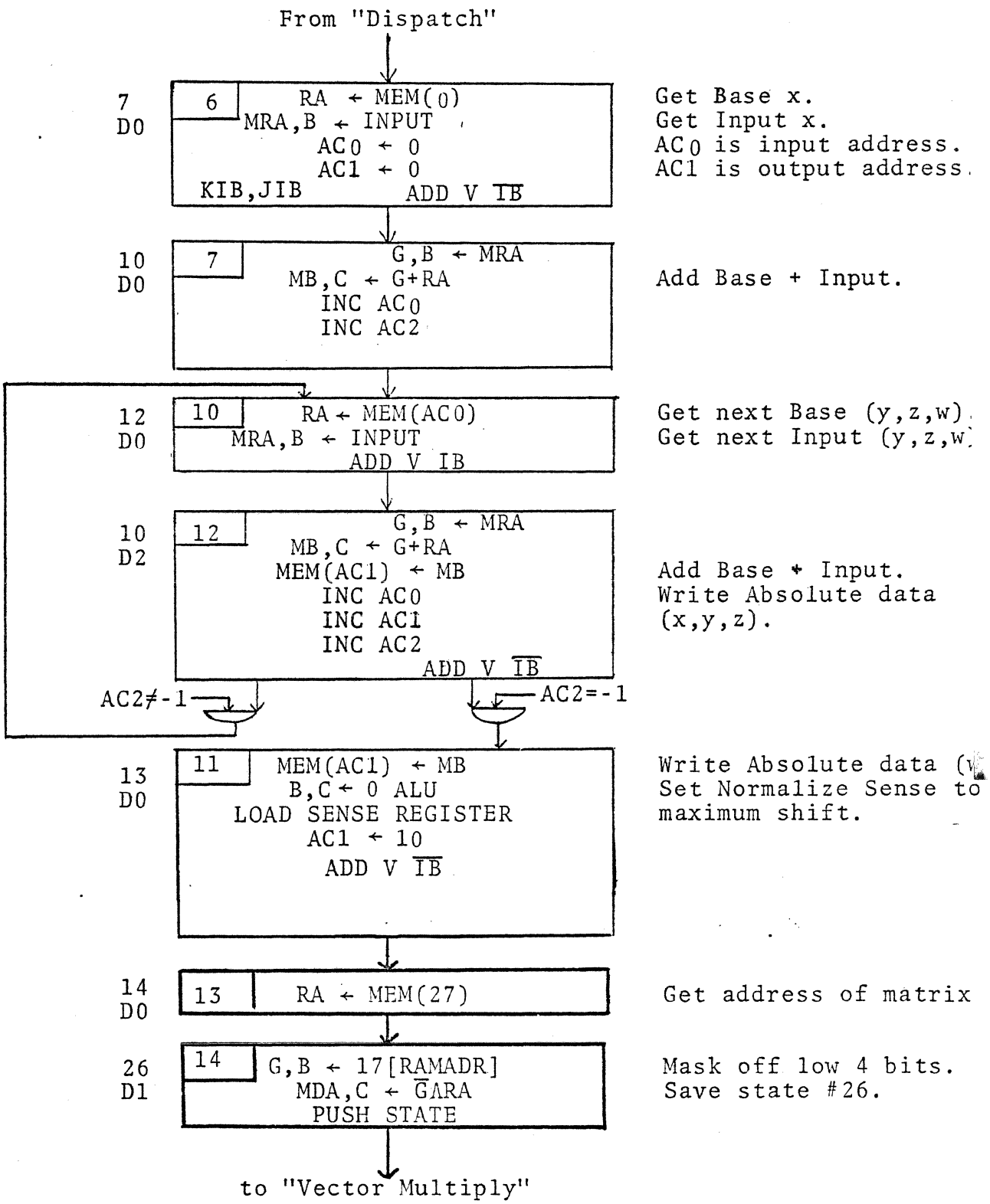

RESET
1
D0
0
LOAD RSR
AC3,B ← COUNT
RAMADR ← AC3
CONDITIONAL RSR HOLD
CLEAR EXTEND
to "DISPATCH"

Get RSR

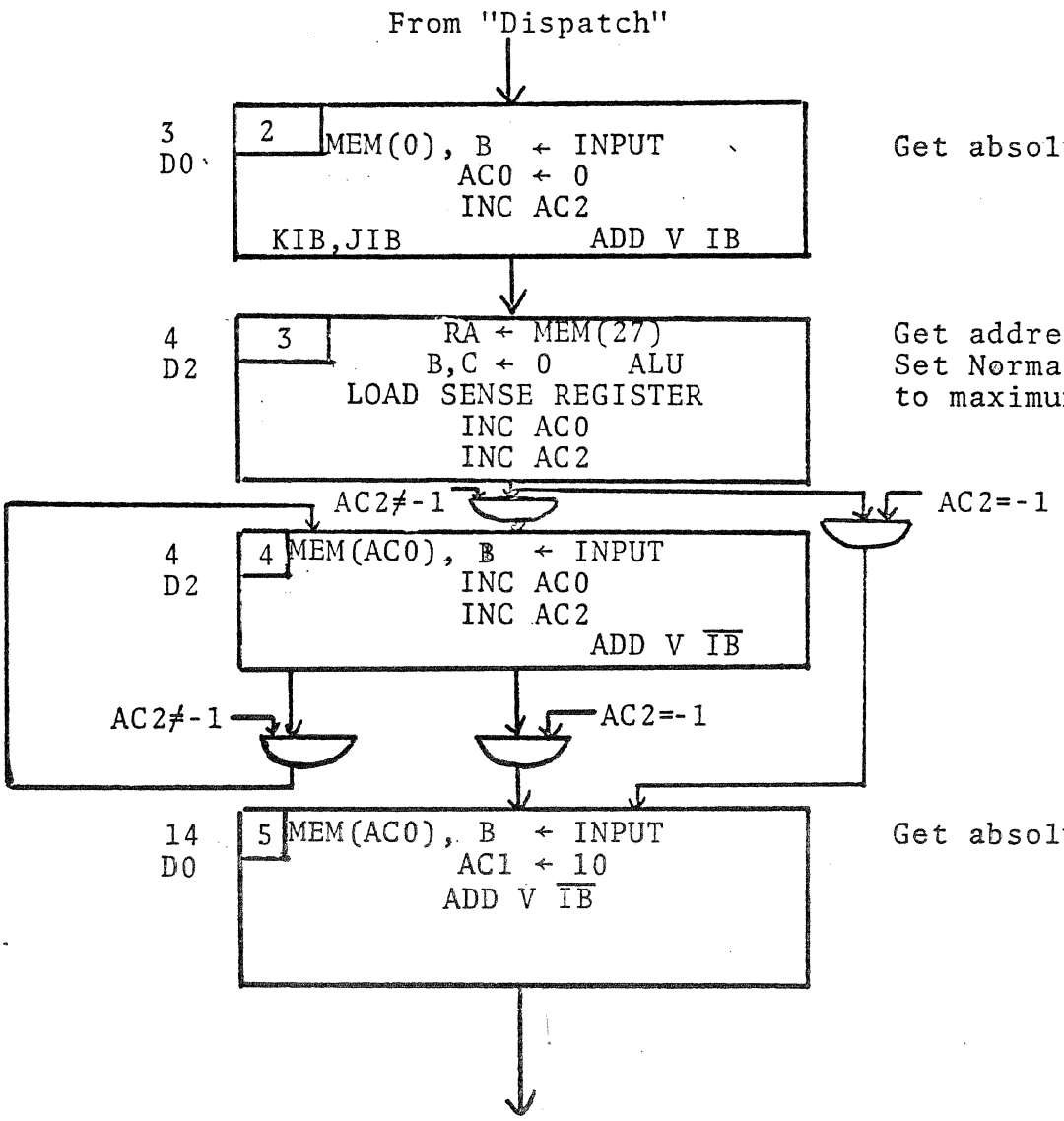

FS=FSM2
(op-code=1,2,3)
SC=Sub-op-code
(op-code=0)
to "Set Input Base" (22)
FS=0
to "Origin Offset" (15)
FS=1
to "Absolute Input" (2)
FS=2,4
to "Relative Input" (6)
FS=3
to "Pass" (243)
FS=5,6,7
to "No-Op & Terminate" (354)
SC=0
to "Jump" (345)
SC=1
to "Push Jump" (336)
SC=2
to "Pop Jump" (332)
SC=3
to "Load" (300)
SC=4
to "Store" (301)
SC=5
to "Load Stack" (320)
SC=6
to "Store Stack" (325)
SC=7
0
D0
1
AC2,B ← 2D,3D,4D
DISPATCH ON RSR
to "Move/Swap" (265)
SC=10
to "Push" (255)
SC=11
to "Pop" (250)
SC=12
to "Matrix Push & Draw" (220)
SC=13
Not used
SC=14
Not used
SC=15
Not used
SC=16
Not used
SC=17


1
D3
377
UPDATE RSR
RAMADR ← AC3
CONDITIONAL VECTOR HOLD
INC AC3
AC3≠-1
AC3=-1
to "DISPATCH"
to "Get RSR"

Next Vector

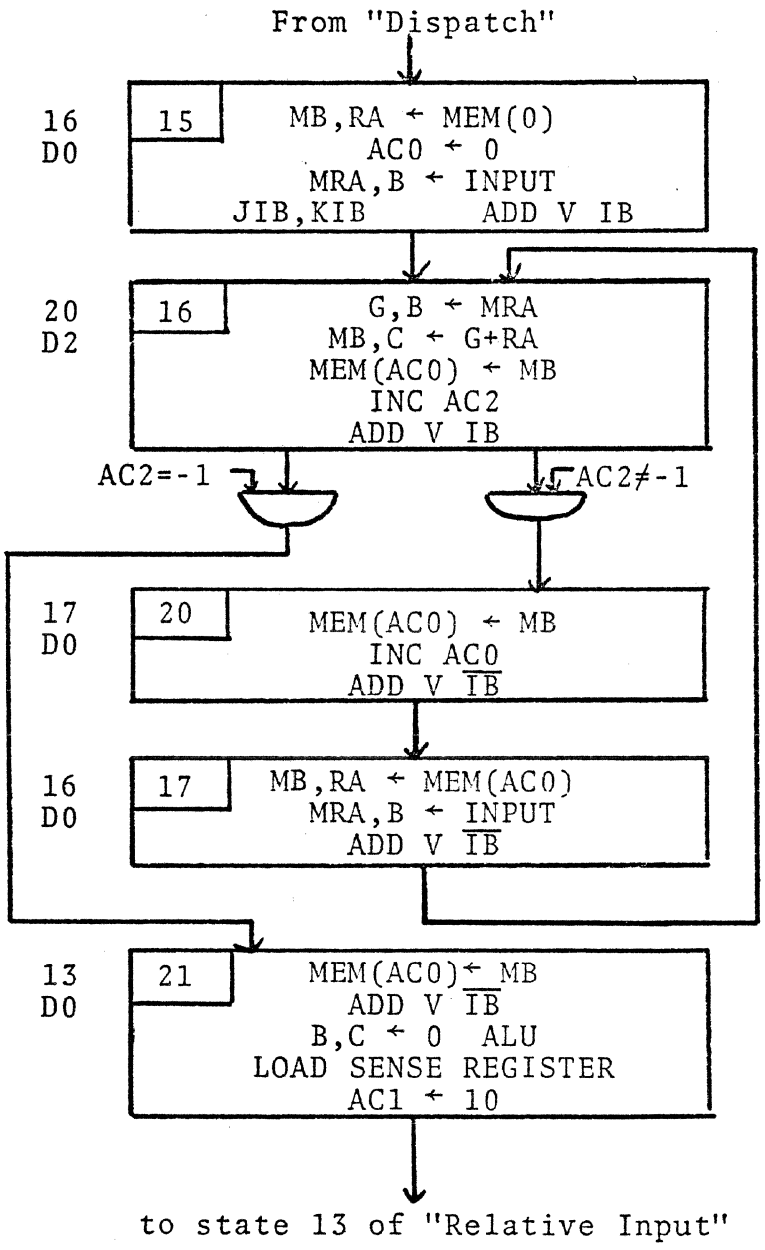


Relative Input

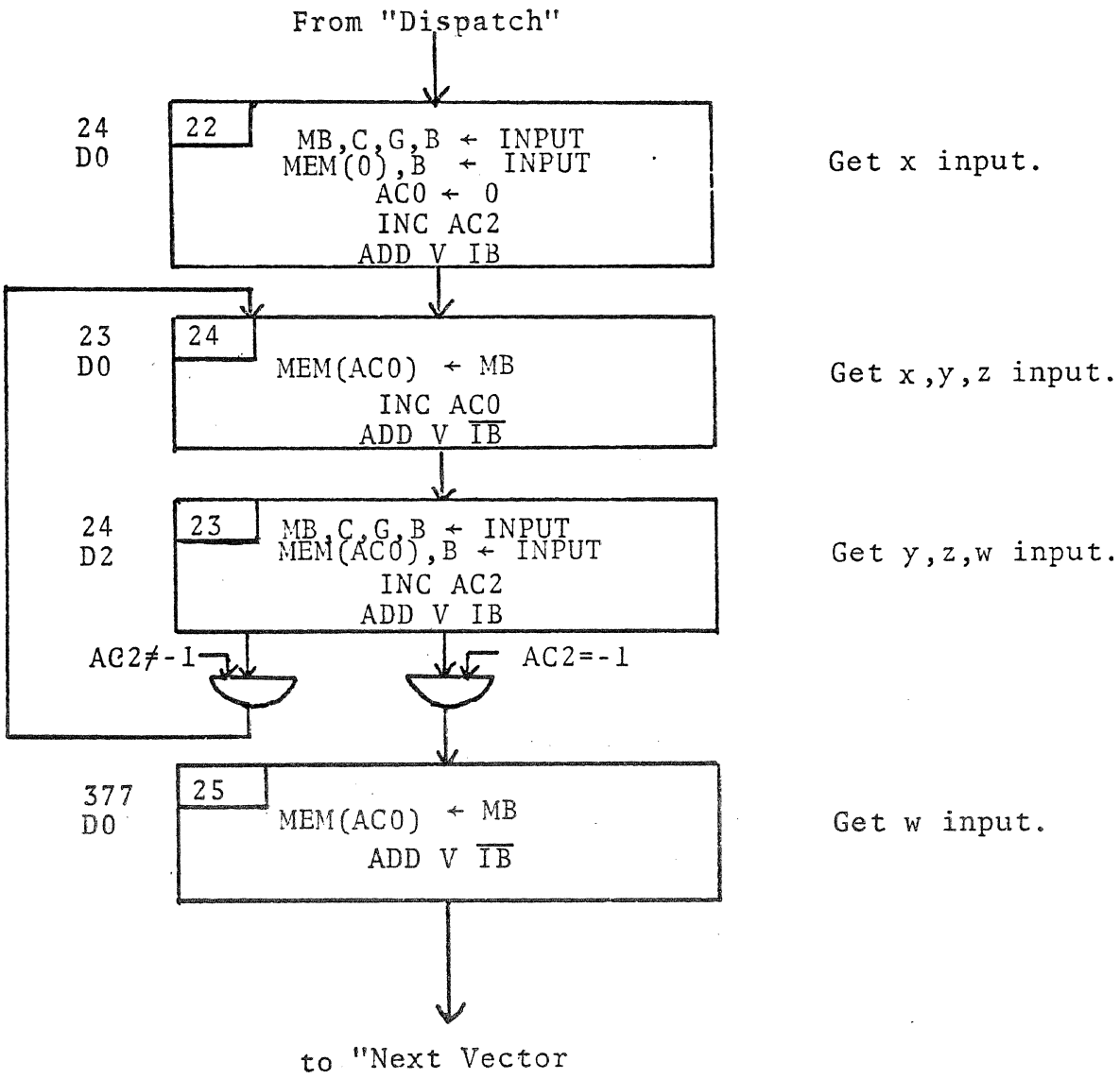


to state 14 of "Relative Input"

Absolute Input

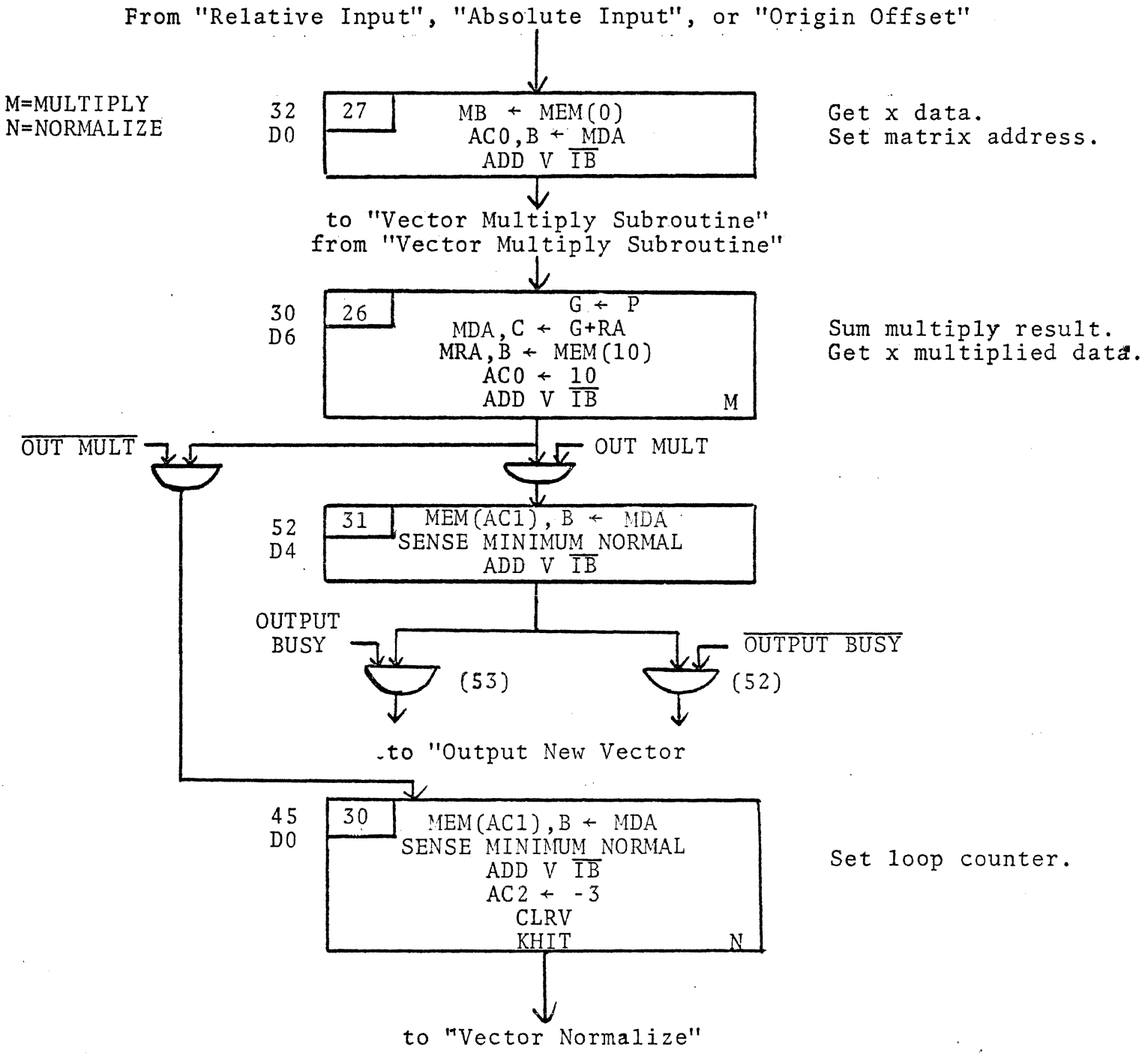


Origin Offset



(Writes input data to both halves of INPUT/BASE)

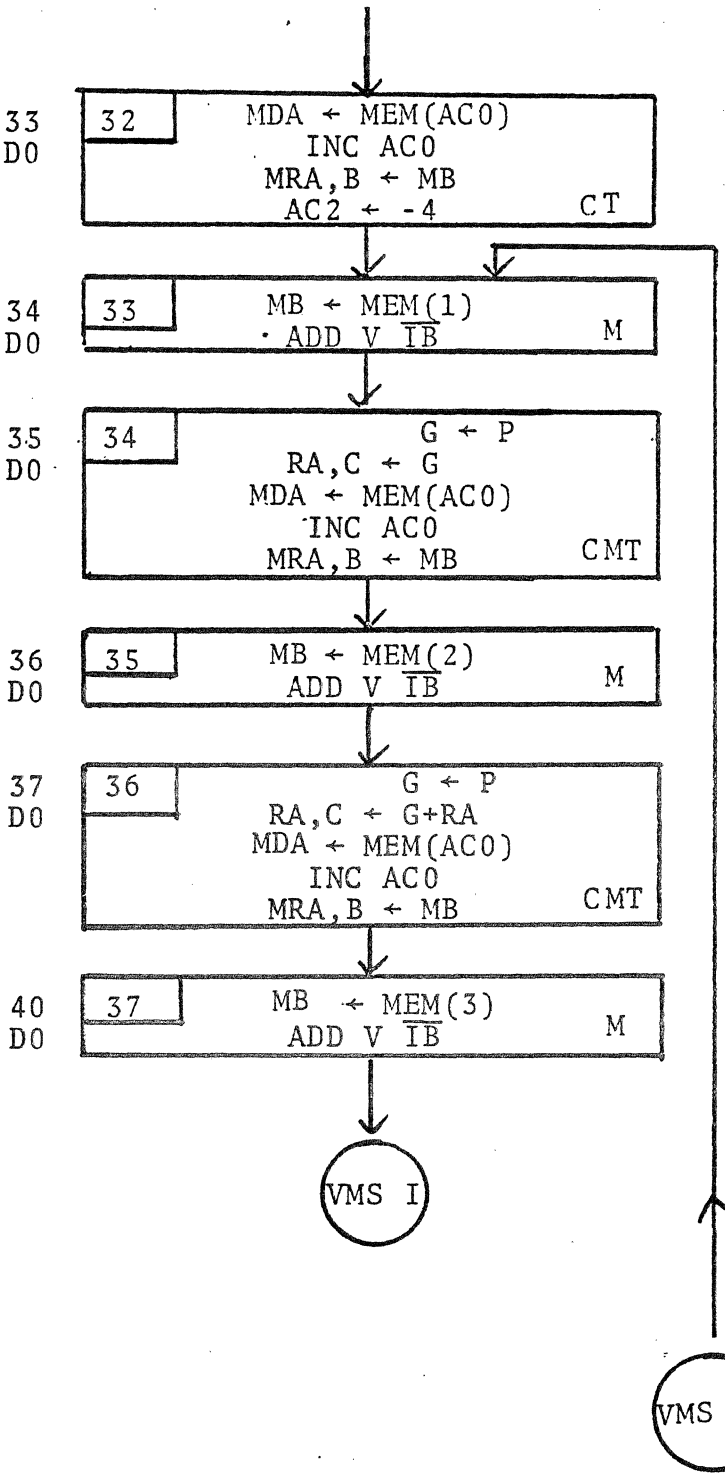
Set Input Base



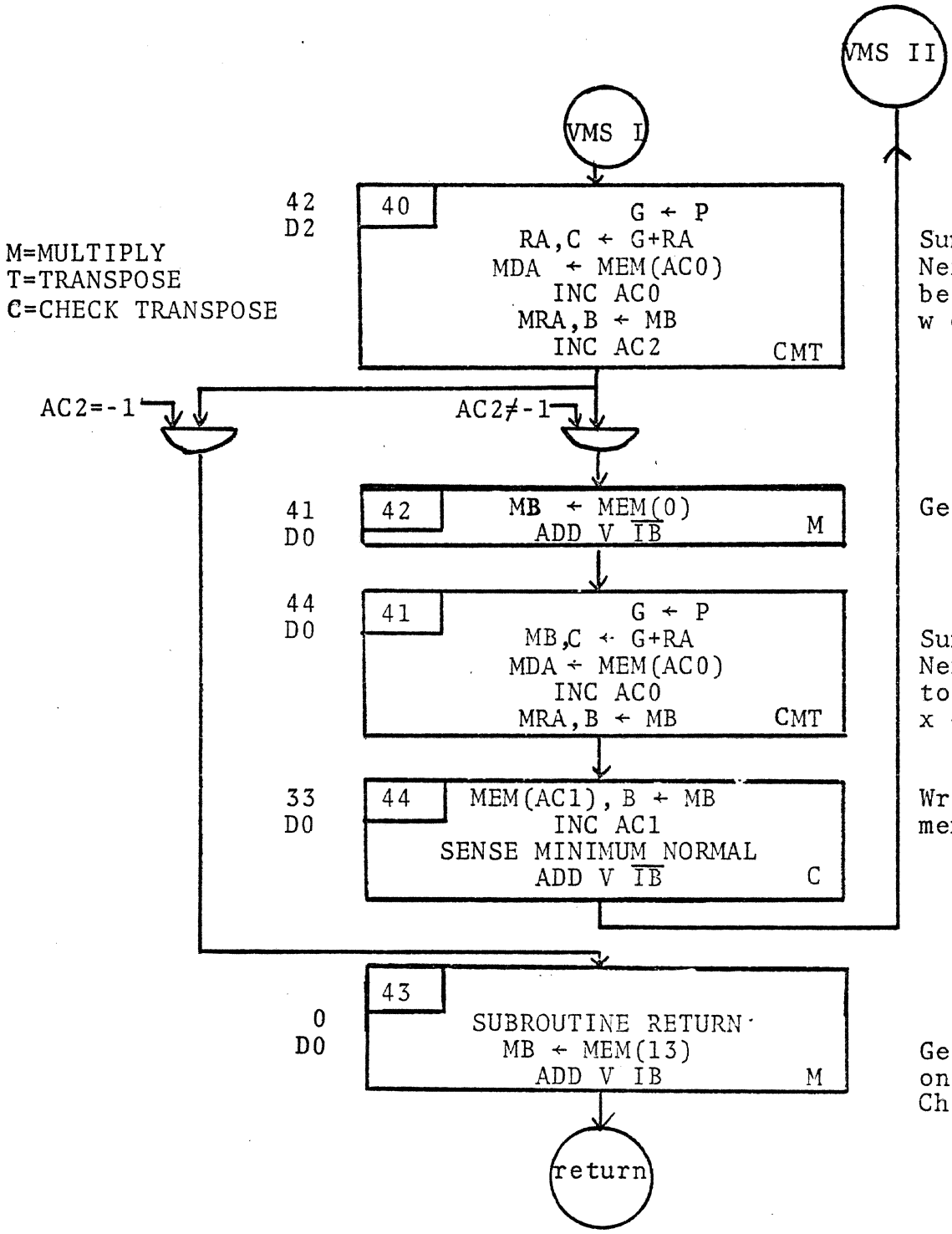
Vector Multiply

From "Vector Multiply" or "Matrix Continue"

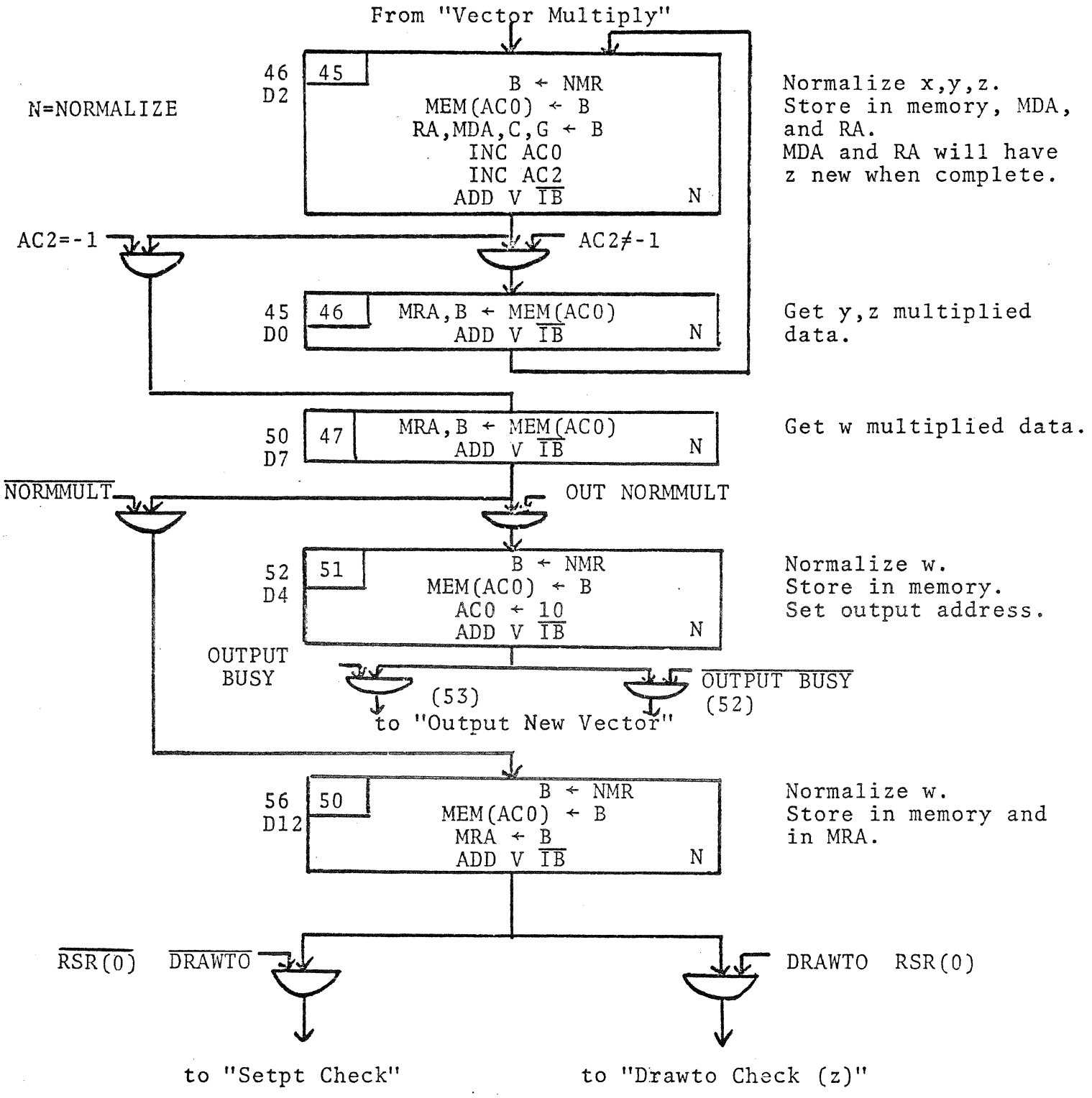
=MULTIPLY
=TRANSPOSE
=CHECK TRANSPOSE



Vector Multiply Subroutine

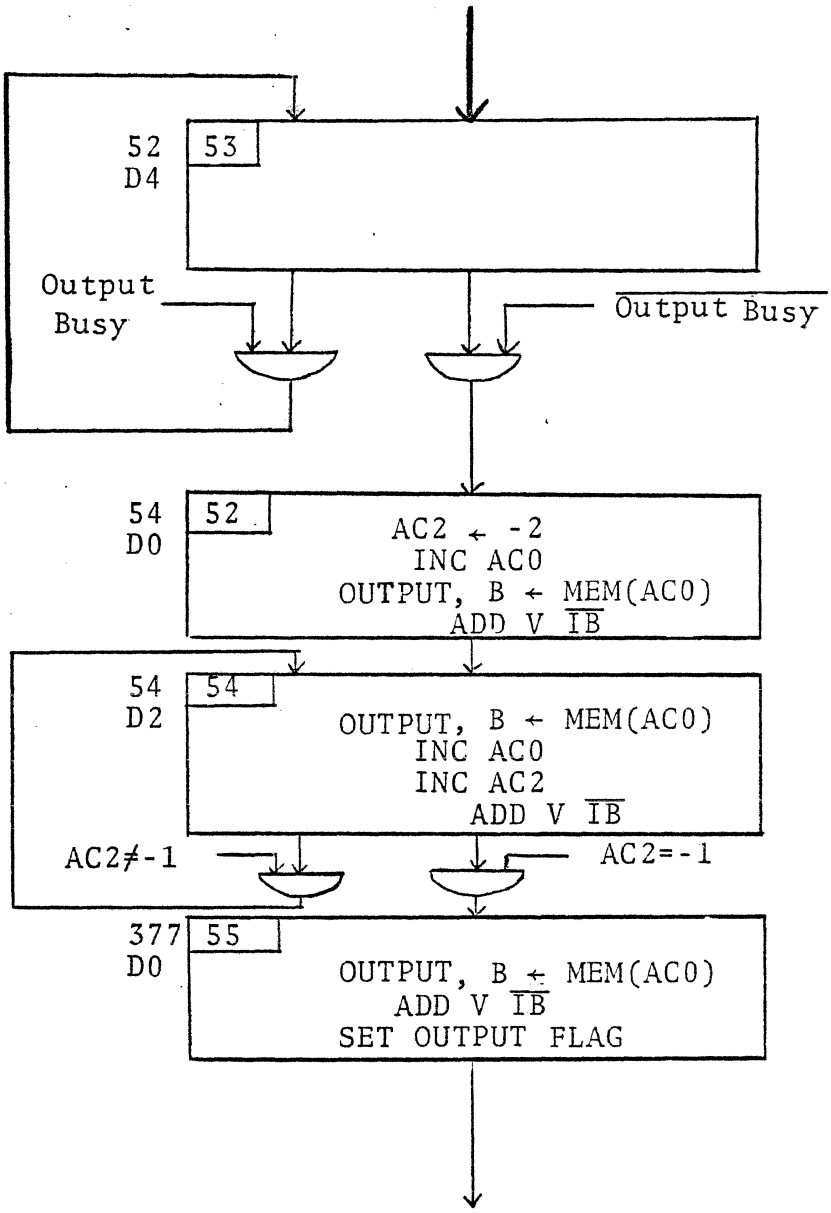


Vector Multiply Subroutine Continued



Vector Normalize

From "Vector Multiply", "Vector Normalize"



Wait for output ready.

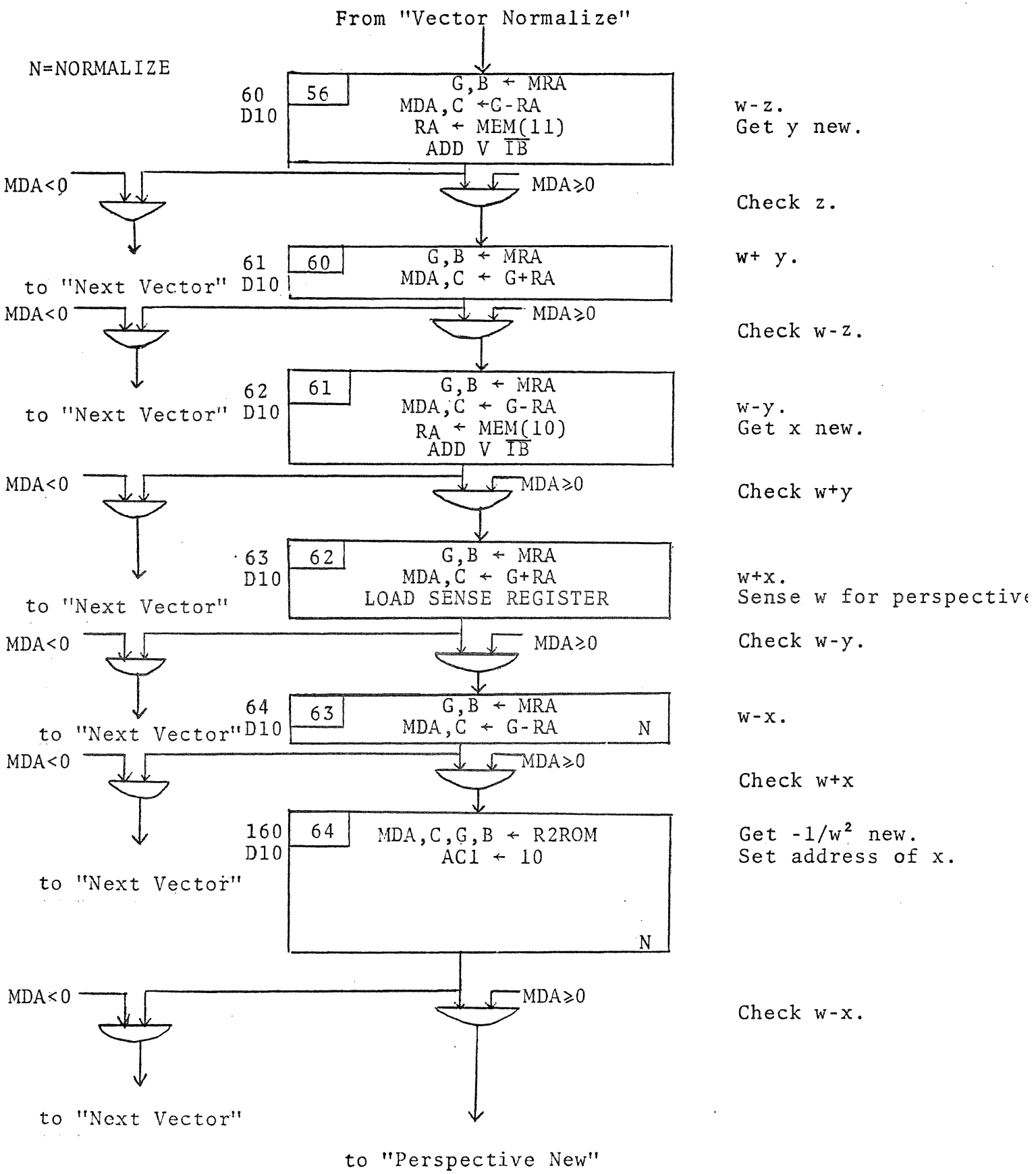
Set loop count.
Output x data.

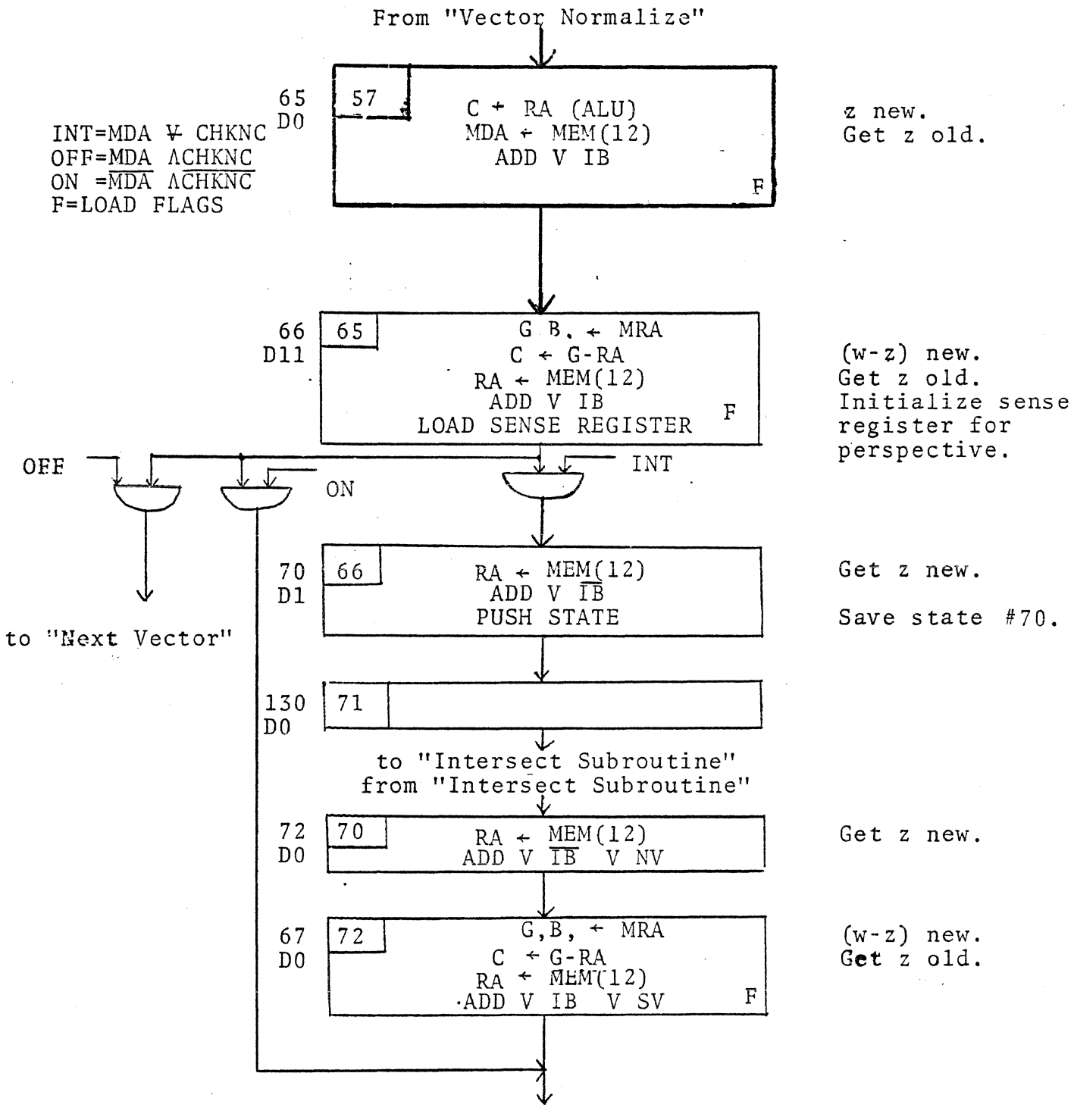
Output y,z, data.

Output w data.
Set output flag.

to "Next Vector"

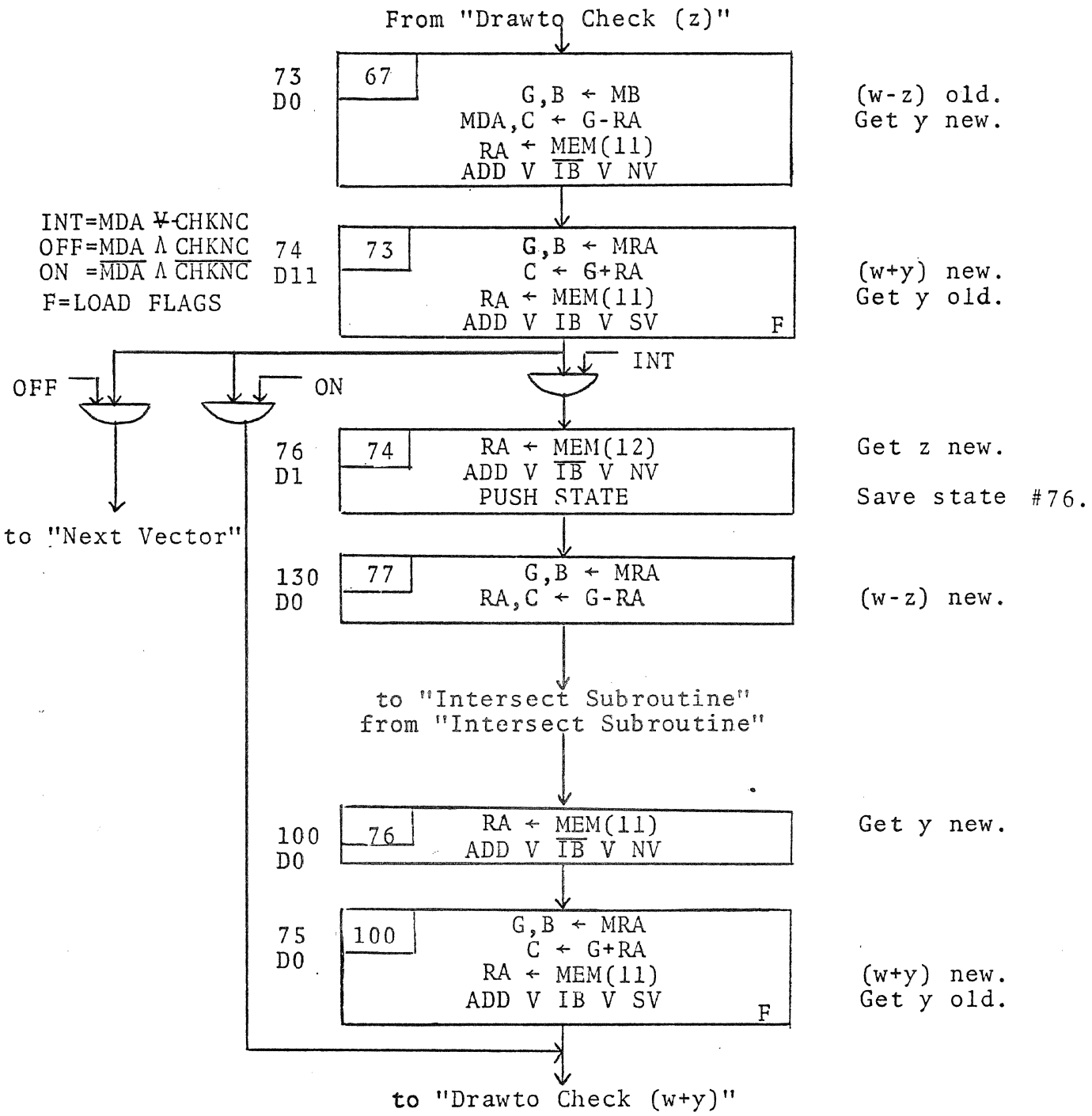
Output New Vector

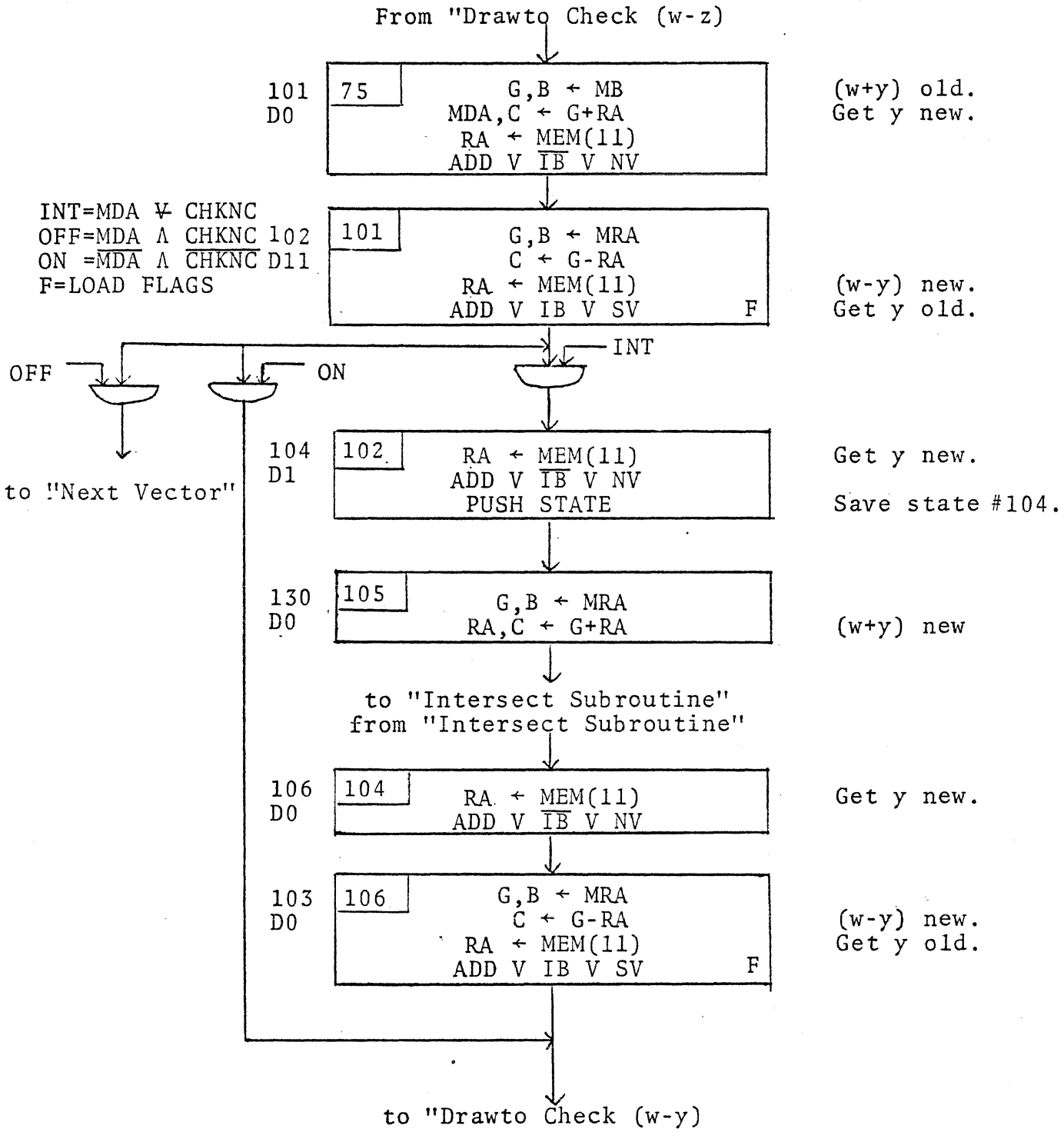

From "Vector Normalize"
N=NORMALIZE
60
D10
56
G,B ← MRA
MDA,C ← G-RA
RA ← MEM(11)
ADD V IB
w-z.
Get y new.
MDA<0
MDA≥0
Check z.
61
D10
60
G,B ← MRA
MDA,C ← G+RA
w+ y.
to "Next Vector"
MDA<0
MDA≥0
Check w-z.
62
D10
61
G,B ← MRA
MDA,C ← G-RA
RA ← MEM(10)
ADD V IB
w-y.
Get x new.
to "Next Vector"
MDA<0
MDA≥0
Check w+y
63
D10
62
G,B ← MRA
MDA,C ← G+RA
LOAD SENSE REGISTER
w+x.
Sense w for perspective
to "Next Vector"
MDA<0
MDA≥0
Check w-y.
64
D10
63
G,B ← MRA
MDA,C ← G-RA
N
w-x.
to "Next Vector"
MDA<0
MDA≥0
Check w+x
160
D10
64
MDA,C,G,B ← R2ROM
AC1 ← 10
N
Get -1/w² new.
Set address of x.
to "Next Vector"
MDA<0
MDA≥0
Check w-x.
to "Next Vector"
to "Perspective New"

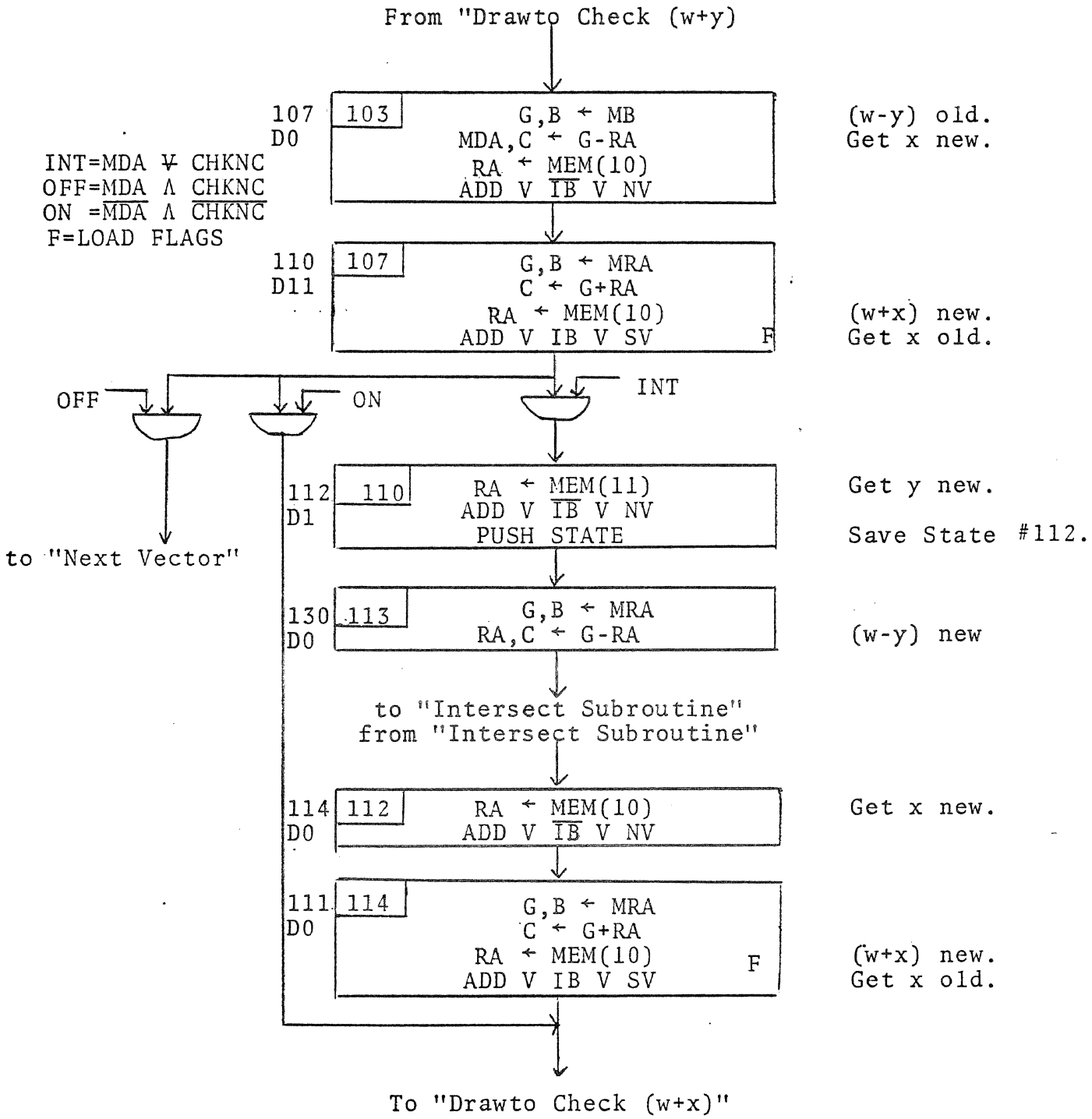


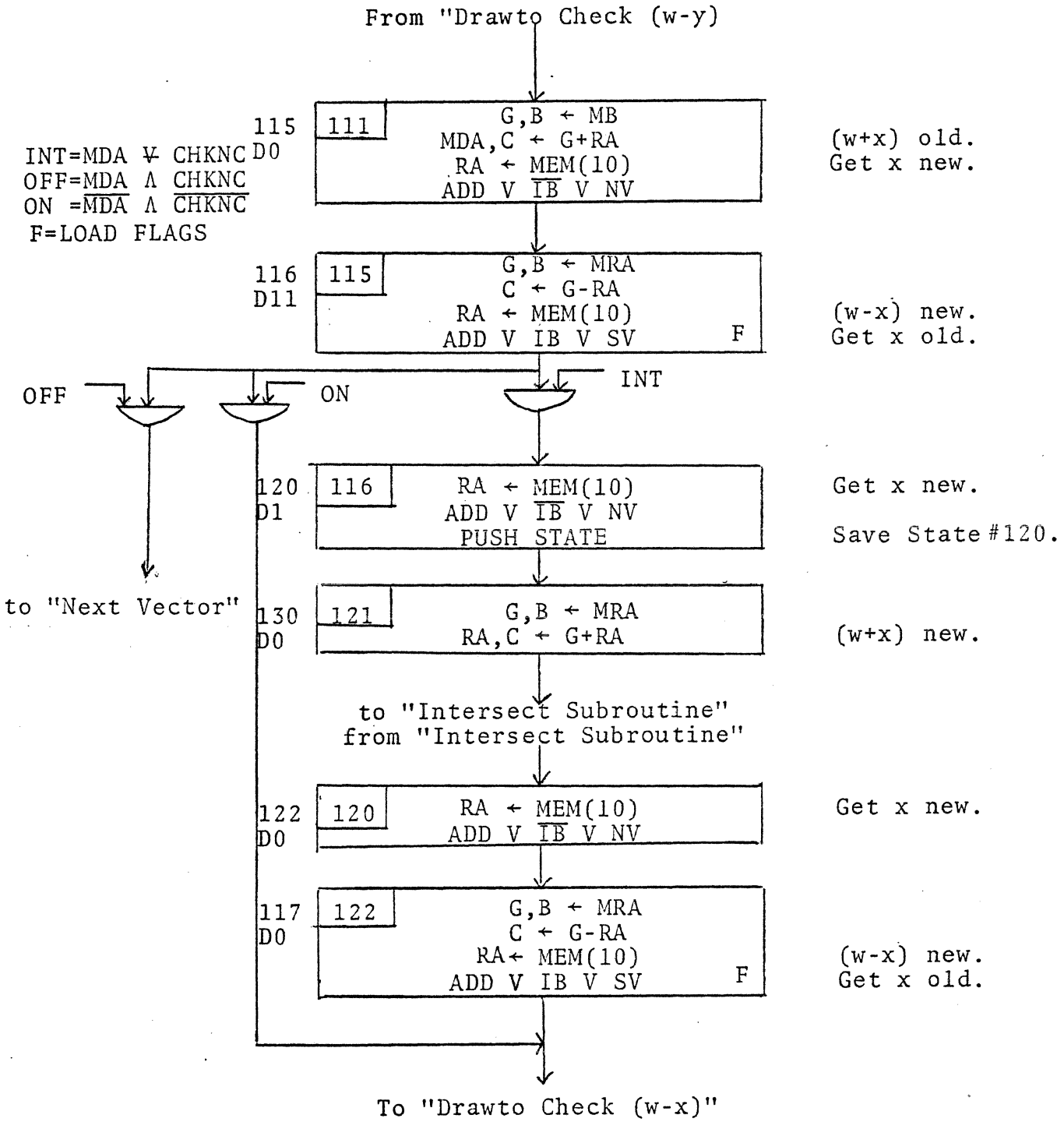
NOTE: MRA has been initialized with w new in "Vector Normalize". MB has been initialized with w old in "Vector Multiply Subroutine.".

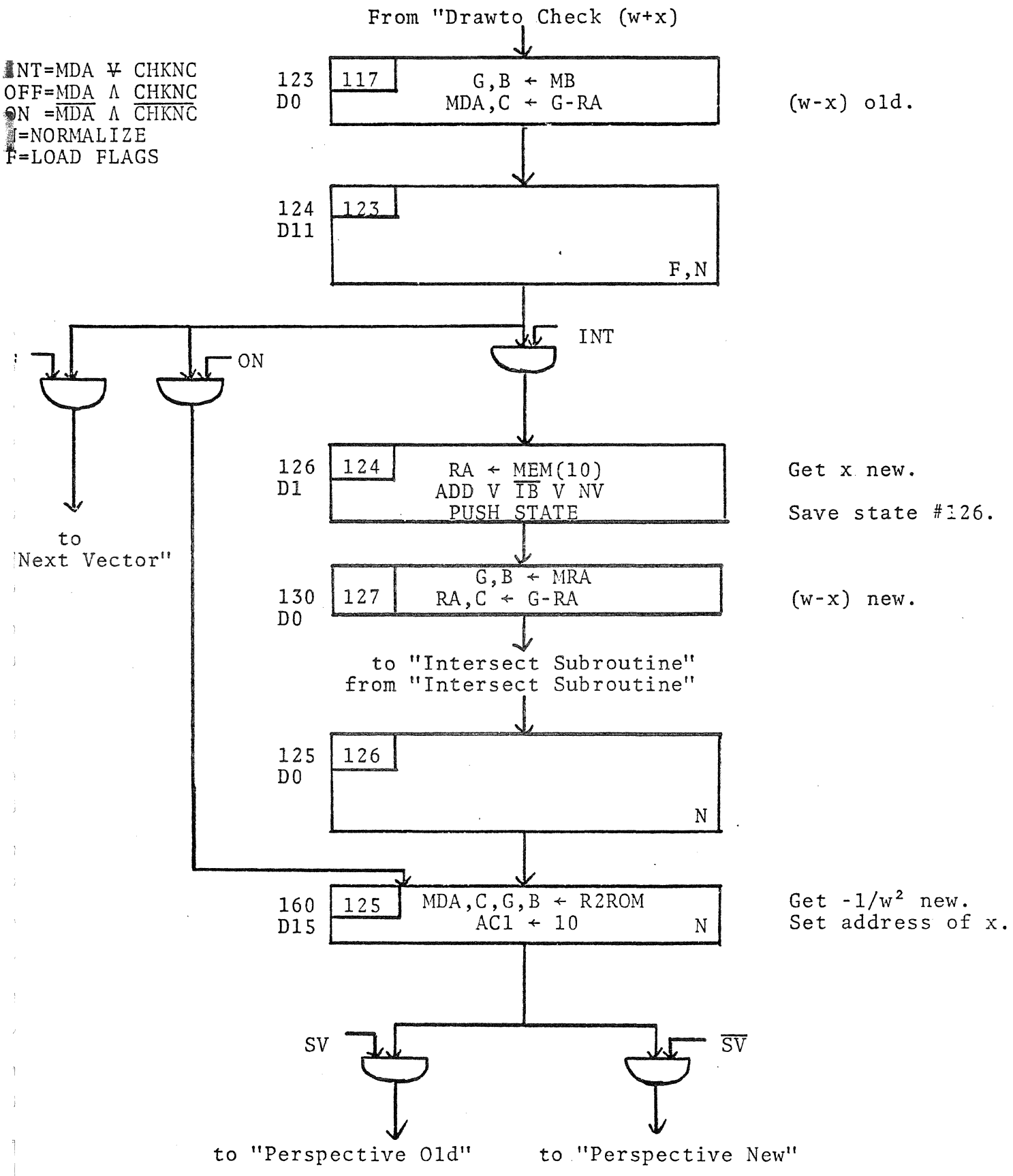
Drawto Check (z)


From "Drawto Check (z)"
73
D0
67
G,B ← MB
MDA,C ← G-RA
RA ← MEM(11)
ADD V IB V NV
(w-z) old.
Get y new.
INT=MDA ⊻ CHKNC
OFF=MDA ∧ CHKNC
ON =MDA ∧ CHKNC
F=LOAD FLAGS
74
D11
73
G,B ← MRA
C ← G+RA
RA ← MEM(11)
ADD V IB V SV
F
(w+y) new.
Get y old.
INT
OFF
ON
76
D1
74
RA ← MEM(12)
ADD V IB V NV
PUSH STATE
Get z new.
Save state #76.
to "Next Vector"
130
D0
77
G,B ← MRA
RA,C ← G-RA
(w-z) new.
to "Intersect Subroutine"
from "Intersect Subroutine"
100
D0
76
RA ← MEM(11)
ADD V IB V NV
Get y new.
75
D0
100
G,B ← MRA
C ← G+RA
RA ← MEM(11)
ADD V IB V SV
F
(w+y) new.
Get y old.
to "Drawto Check (w+y)"

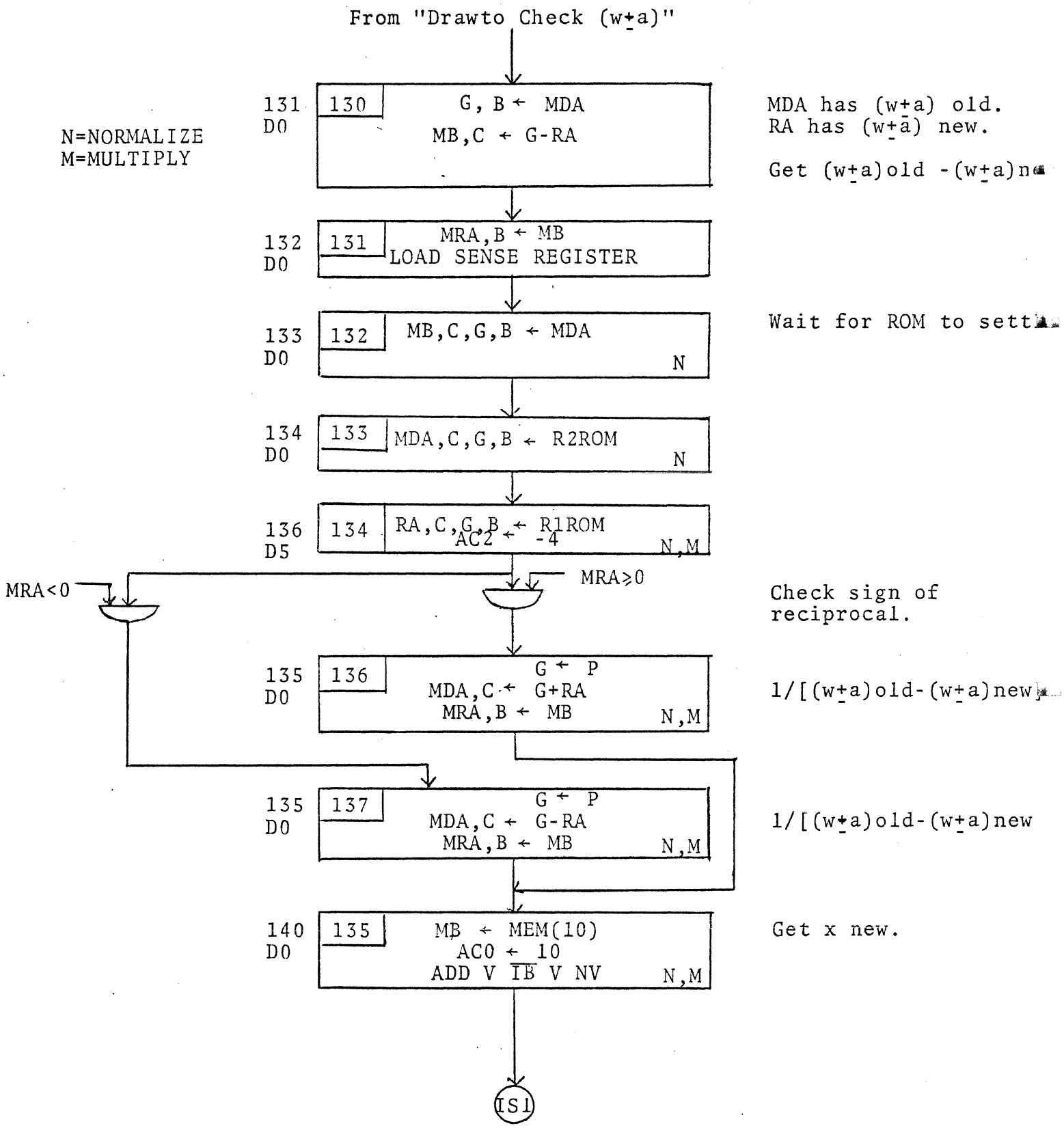

From "Drawto Check (w-z)
101
D0
75
G,B ← MB
MDA,C ← G+RA
RA ← MEM(11)
ADD V IB V NV
(w+y) old.
Get y new.
INT=MDA ⊻ CHKNC
OFF=MDA ∧ CHKNC 102
ON =MDA ∧ CHKNC D11
F=LOAD FLAGS
101
G,B ← MRA
C ← G-RA
RA ← MEM(11)
ADD V IB V SV F
(w-y) new.
Get y old.
INT
OFF
ON
to "Next Vector"
104
D1
102
RA ← MEM(11)
ADD V IB V NV
PUSH STATE
Get y new.
Save state #104.
130
D0
105
G,B ← MRA
RA,C ← G+RA
(w+y) new
to "Intersect Subroutine"
from "Intersect Subroutine"
106
D0
104
RA ← MEM(11)
ADD V IB V NV
Get y new.
103
D0
106
G,B ← MRA
C ← G-RA
RA ← MEM(11)
ADD V IB V SV F
(w-y) new.
Get y old.
to "Drawto Check (w-y)

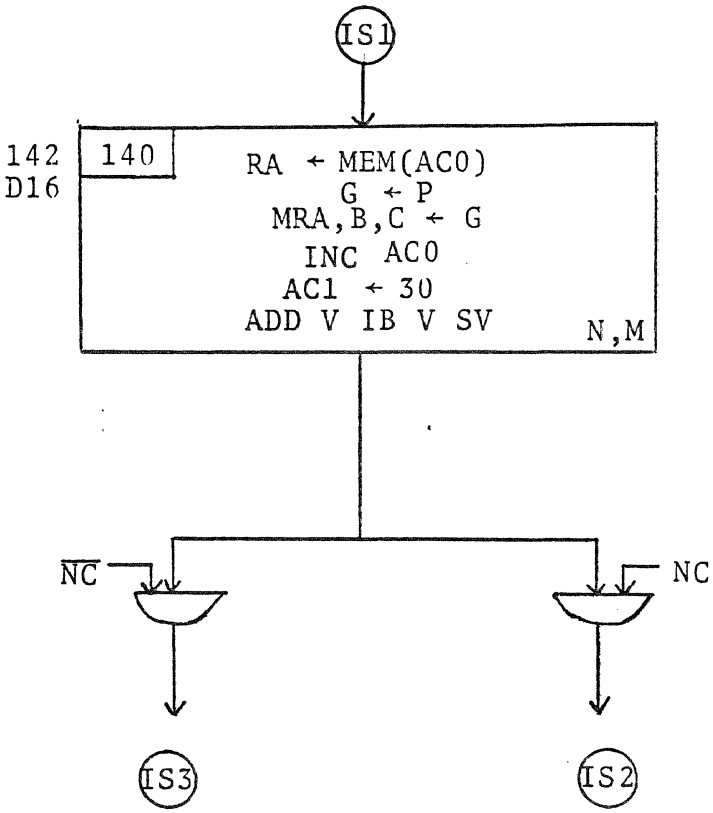

From "Drawto Check (w+y)
107
D0
103
G,B ← MB
MDA,C ← G-RA
RA ← MEM(10)
ADD V IB V NV
(w-y) old.
Get x new.
INT=MDA ∀ CHKNC
OFF=MDA ∧ CHKNC
ON =MDA ∧ CHKNC
F=LOAD FLAGS
110
D11
107
G,B ← MRA
C ← G+RA
RA ← MEM(10)
ADD V IB V SV
F
(w+x) new.
Get x old.
OFF
ON
INT
to "Next Vector"
112
D1
110
RA ← MEM(11)
ADD V IB V NV
PUSH STATE
Get y new.
Save State #112.
130
D0
113
G,B ← MRA
RA,C ← G-RA
(w-y) new
to "Intersect Subroutine"
from "Intersect Subroutine"
114
D0
112
RA ← MEM(10)
ADD V IB V NV
Get x new.
111
D0
114
G,B ← MRA
C ← G+RA
RA ← MEM(10)
ADD V IB V SV
F
(w+x) new.
Get x old.
To "Drawto Check (w+x)"


From "Drawto Check (w-y)
115
D0
111
G,B ← MB
MDA,C ← G+RA
RA ← MEM(10)
ADD V IB V NV
(w+x) old.
Get x new.
INT=MDA ⊻ CHKNC
OFF=MDA ∧ CHKNC
ON =MDA ∧ CHKNC
F=LOAD FLAGS
116
D11
115
G,B ← MRA
C ← G-RA
RA ← MEM(10)
ADD V IB V SV
F
(w-x) new.
Get x old.
OFF
ON
INT
120
D1
116
RA ← MEM(10)
ADD V IB V NV
PUSH STATE
Get x new.
Save State #120.
to "Next Vector"
130
D0
121
G,B ← MRA
RA,C ← G+RA
(w+x) new.
to "Intersect Subroutine"
from "Intersect Subroutine"
122
D0
120
RA ← MEM(10)
ADD V IB V NV
Get x new.
117
D0
122
G,B ← MRA
C ← G-RA
RA← MEM(10)
ADD V IB V SV
F
(w-x) new.
Get x old.
To "Drawto Check (w-x)"



Drawto Check (w-x)


From "Drawto Check (w±a)"
N=NORMALIZE
M=MULTIPLY
131
D0
130
G, B ← MDA
MB,C ← G-RA
MDA has (w±a) old.
RA has (w±ā) new.
Get (w±a)old -(w±a)ne
132
D0
131
MRA,B ← MB
LOAD SENSE REGISTER
133
D0
132
MB,C,G,B ← MDA
N
Wait for ROM to sett
134
D0
133
MDA,C,G,B ← R2ROM
N
136
D5
134
RA,C,G,B ← R1ROM
AC2 ← -4
N,M
MRA<0
MRA≥0
Check sign of
reciprocal.
135
D0
136
G ← P
MDA,C ← G+RA
MRA,B ← MB
N,M
1/[(w±a)old-(w±a)new]
135
D0
137
G ← P
MDA,C ← G-RA
MRA,B ← MB
N,M
1/[(w±a)old-(w±a)new
140
D0
135
MB ← MEM(10)
AC0 ← 10
ADD V IB V NV
N,M
Get x new.
IS1

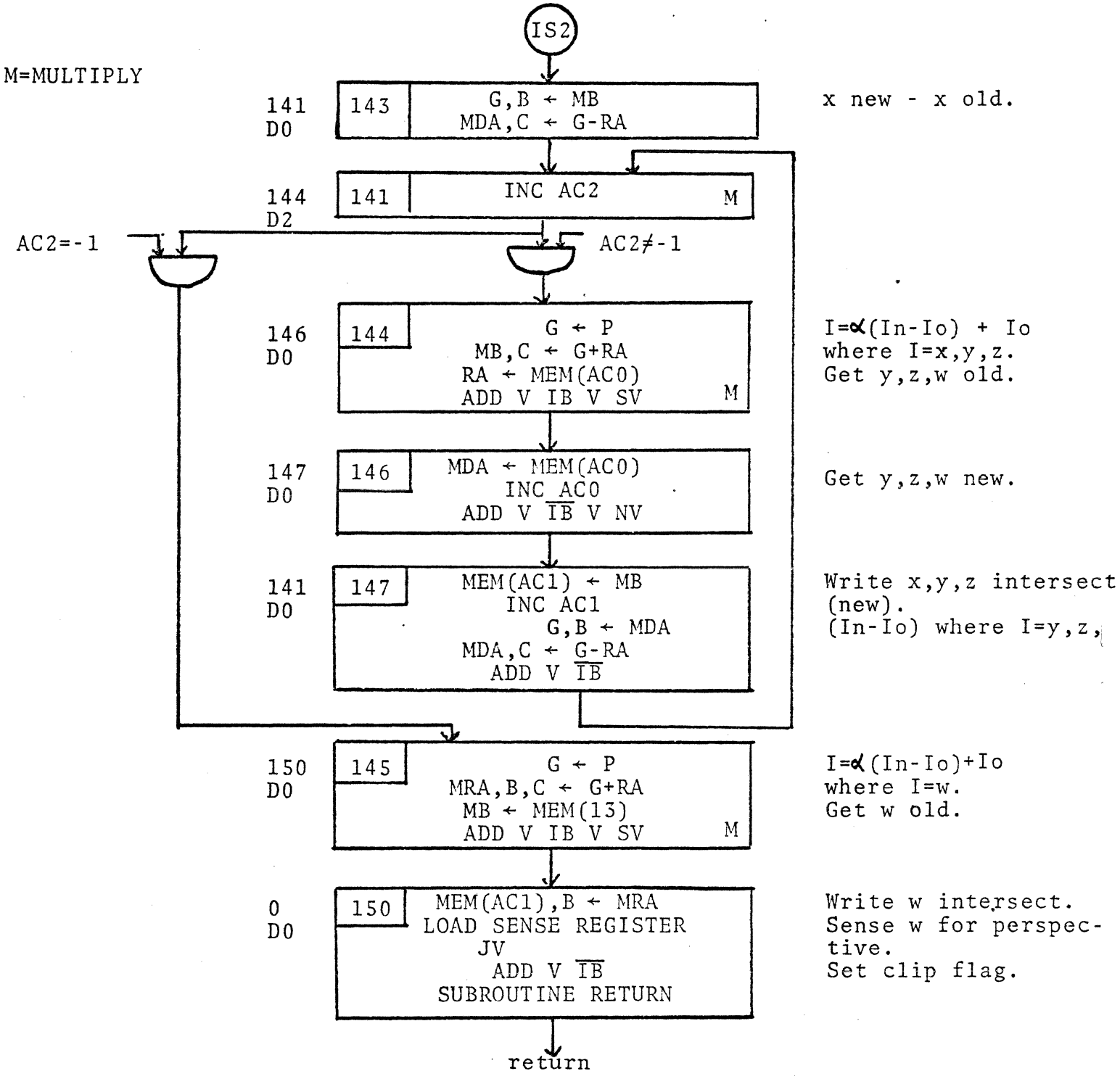


Get x old.
$\propto = (w+a)old/[(w\pm a)old - (w\pm a)new]$

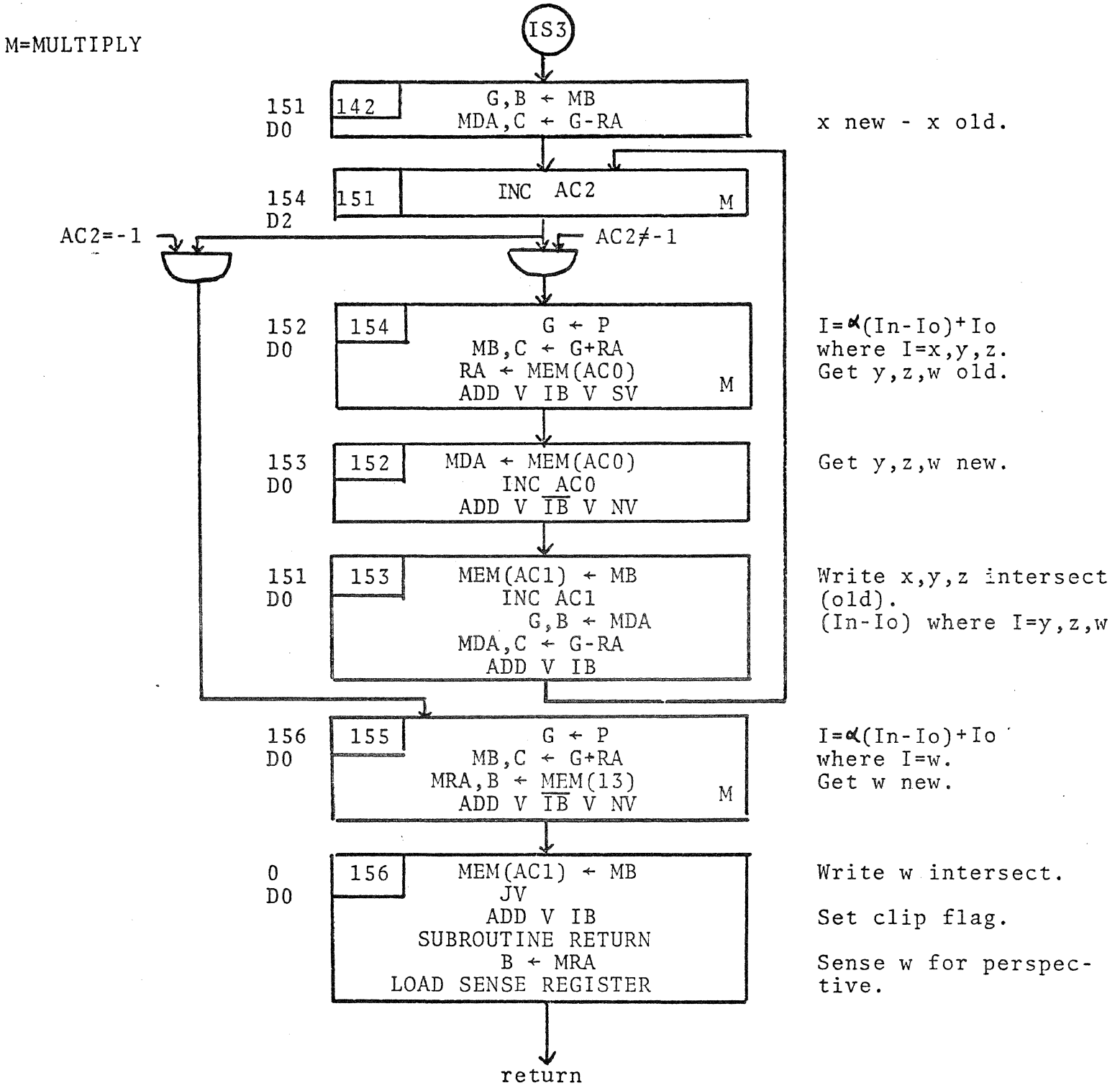
Set output address.

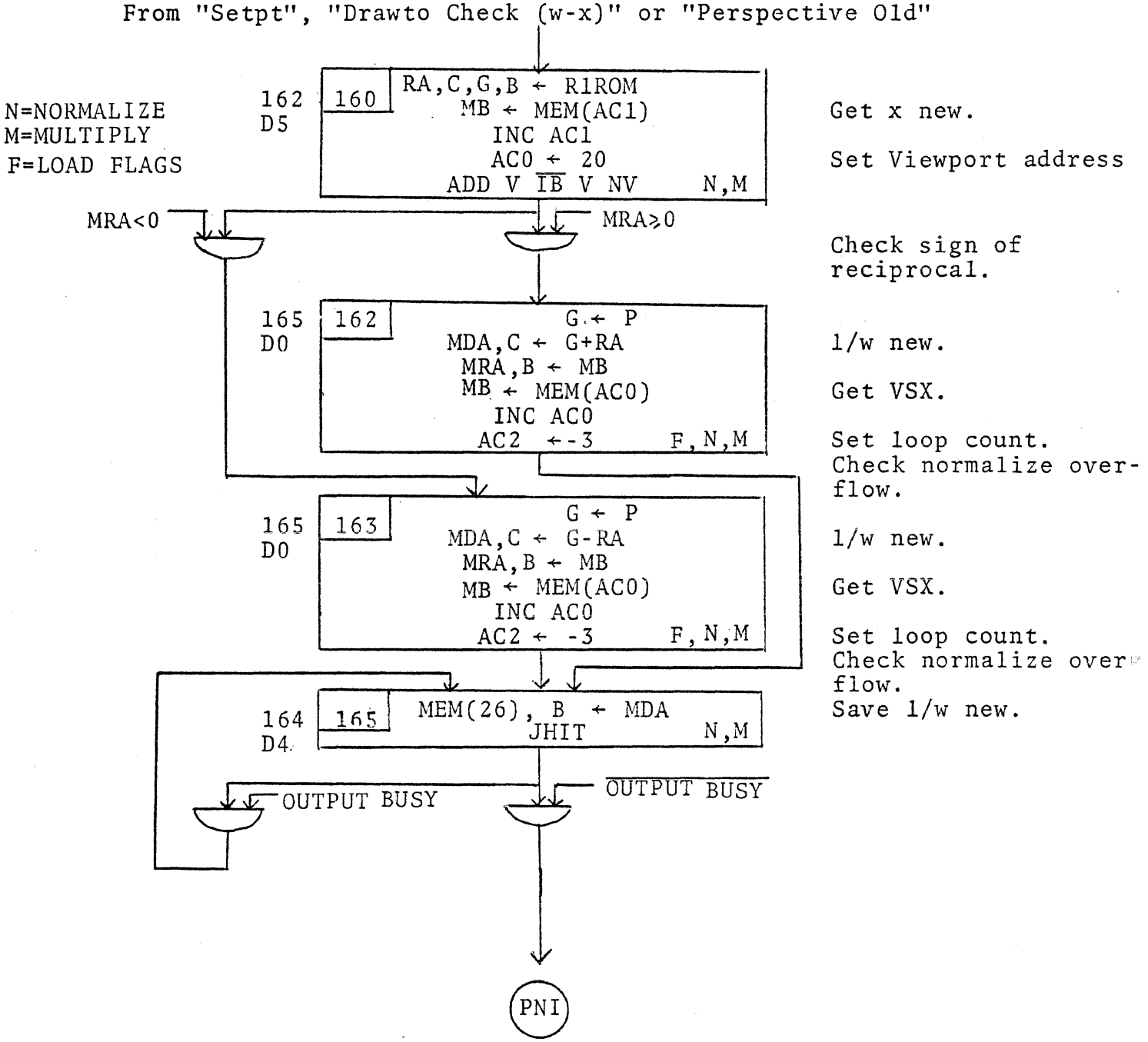
Check to see if new or old point is clippe

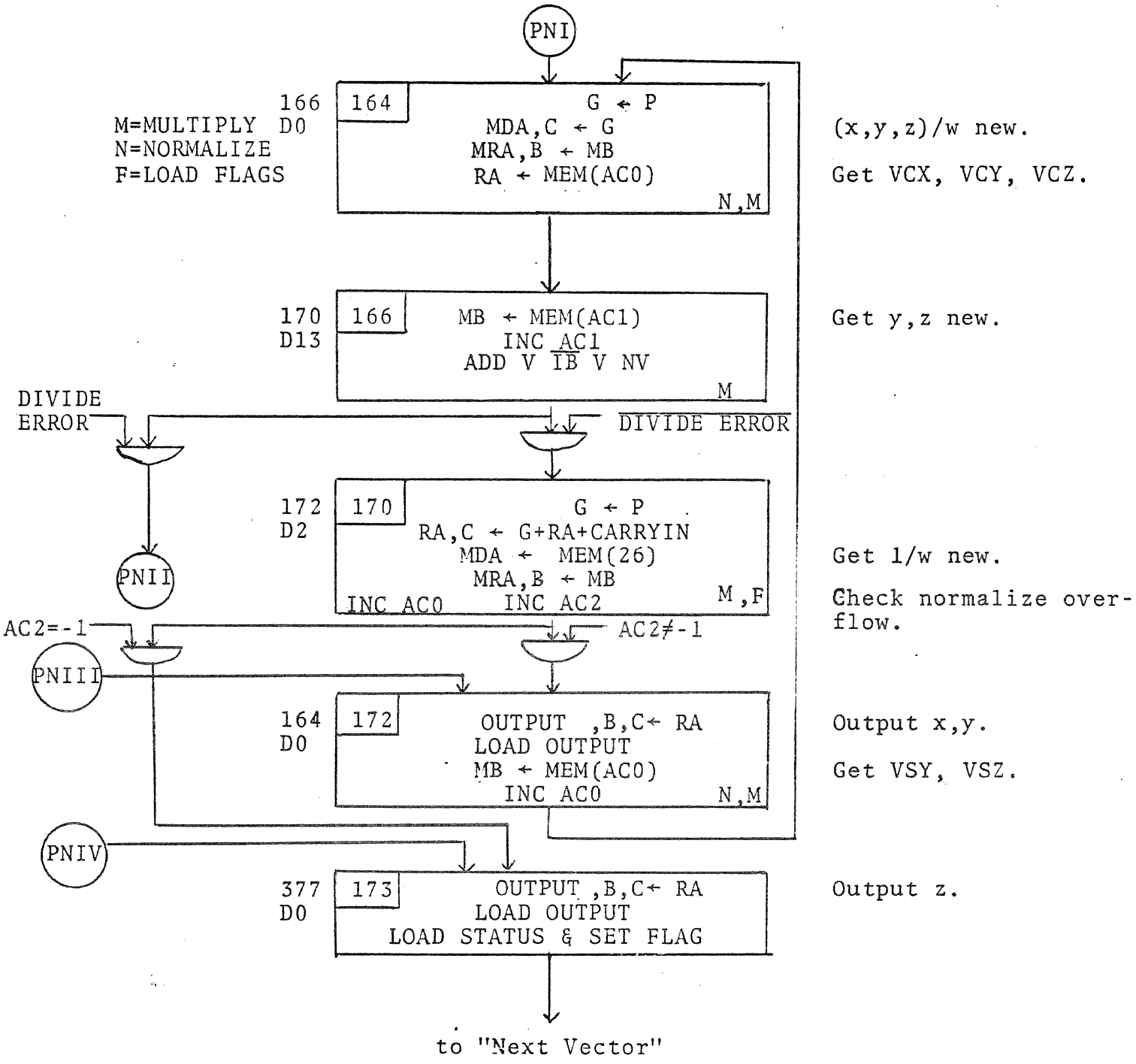
NC=new.
$\overline{NC}$=old.

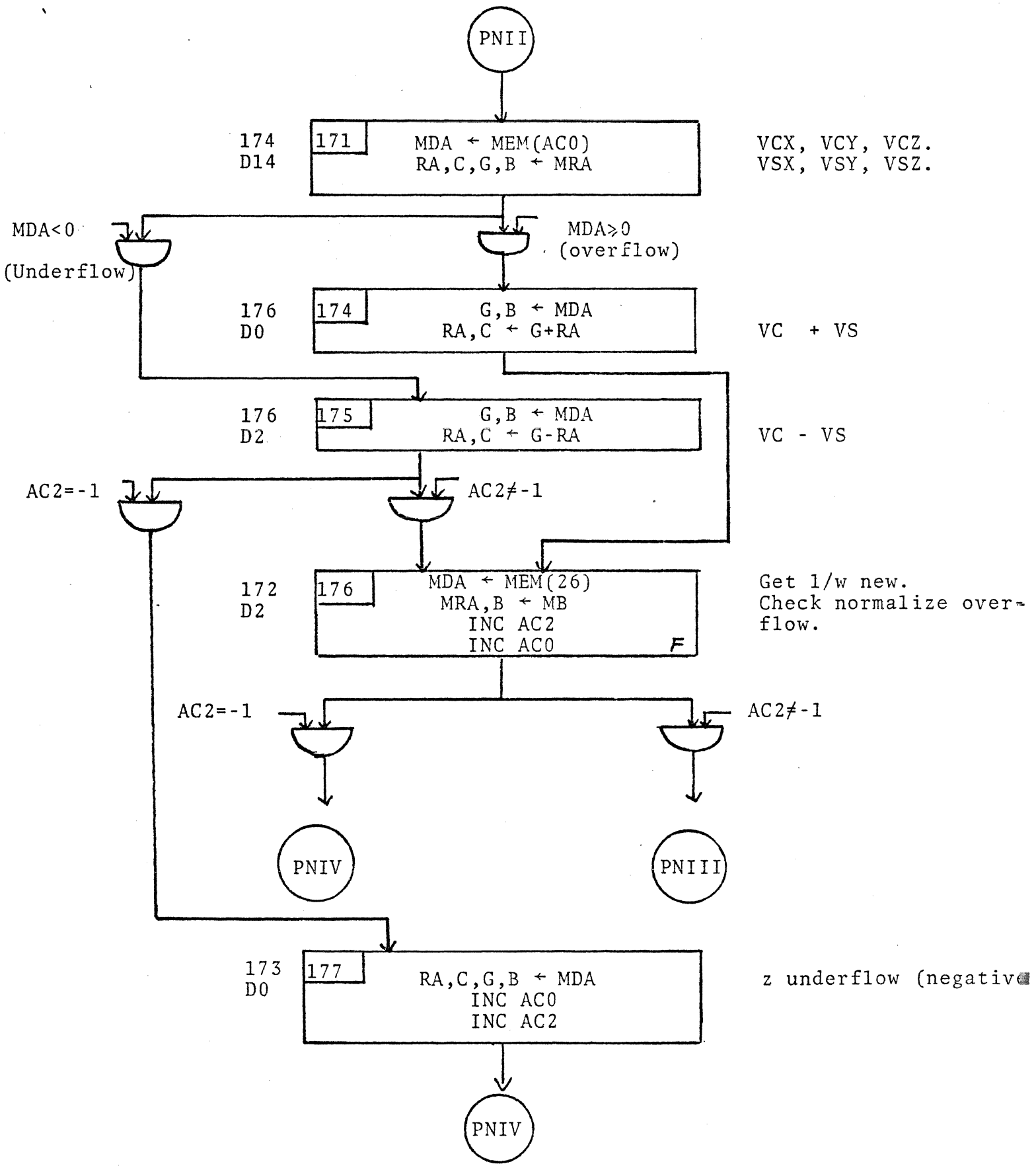


Intersect Subroutine Continued


M=MULTIPLY
IS3
151
D0
142
G,B ← MB
MDA,C ← G-RA
x new - x old.
154
D2
151
INC AC2
M
AC2=-1
AC2≠-1
152
D0
154
G ← P
MB,C ← G+RA
RA ← MEM(AC0)
ADD V IB V SV
M
I=∝(In-Io)+Io
where I=x,y,z.
Get y,z,w old.
153
D0
152
MDA ← MEM(AC0)
INC AC0
ADD V IB V NV
Get y,z,w new.
151
D0
153
MEM(AC1) ← MB
INC AC1
G,B ← MDA
MDA,C ← G-RA
ADD V IB
Write x,y,z intersect (old).
(In-Io) where I=y,z,w
156
D0
155
G ← P
MB,C ← G+RA
MRA,B ← MEM(13)
ADD V IB V NV
M
I=∝(In-Io)+Io
where I=w.
Get w new.
0
D0
156
MEM(AC1) ← MB
JV
ADD V IB
SUBROUTINE RETURN
B ← MRA
LOAD SENSE REGISTER
Write w intersect.
Set clip flag.
Sense w for perspective.
return

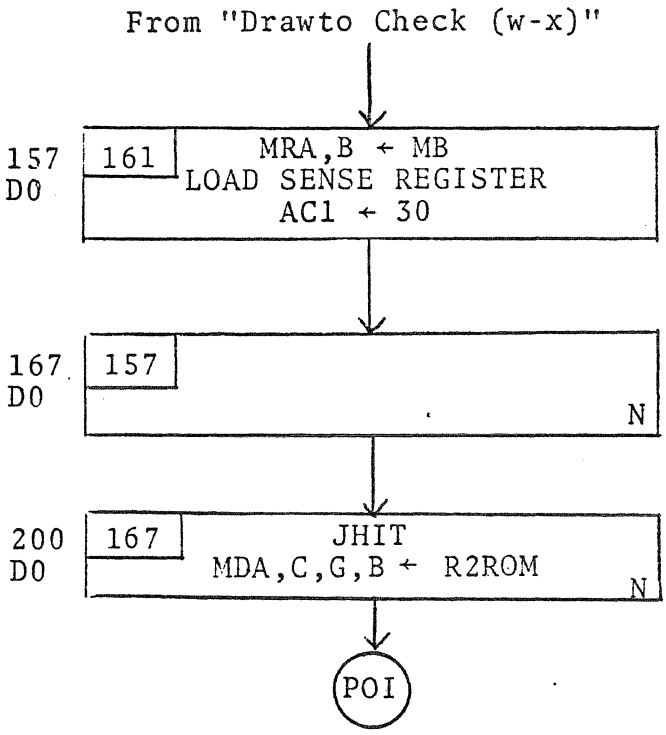

From "Setpt", "Drawto Check (w-x)" or "Perspective Old"
N=NORMALIZE
M=MULTIPLY
F=LOAD FLAGS
162
D5
160
RA,C,G,B ← R1ROM
MB ← MEM(AC1)
INC AC1
AC0 ← 20
ADD V IB V NV N,M
Get x new.
Set Viewport address
MRA<0
MRA≥0
Check sign of reciprocal.
165
D0
162
G ← P
MDA,C ← G+RA
MRA,B ← MB
MB ← MEM(AC0)
INC AC0
AC2 ← -3 F,N,M
1/w new.
Get VSX.
Set loop count.
Check normalize overflow.
165
D0
163
G ← P
MDA,C ← G-RA
MRA,B ← MB
MB ← MEM(AC0)
INC AC0
AC2 ← -3 F,N,M
1/w new.
Get VSX.
Set loop count.
Check normalize overflow.
164
D4
165
MEM(26), B ← MDA
JHIT N,M
Save 1/w new.
OUTPUT BUSY
OUTPUT BUSY
PNI


PNI
166
164
G ← P
MDA,C ← G
MRA,B ← MB
RA ← MEM(AC0)
N,M
M=MULTIPLY
N=NORMALIZE
F=LOAD FLAGS
D0
(x,y,z)/w new.
Get VCX, VCY, VCZ.
170
166
MB ← MEM(AC1)
INC AC1
ADD V IB V NV
M
D13
Get y,z new.
DIVIDE ERROR
DIVIDE ERROR
172
170
G ← P
RA,C ← G+RA+CARRYIN
MDA ← MEM(26)
MRA,B ← MB
INC AC0
INC AC2
M,F
D2
Get 1/w new.
Check normalize overflow.
PNII
AC2=-1
AC2≠-1
PNIII
164
172
OUTPUT ,B,C← RA
LOAD OUTPUT
MB ← MEM(AC0)
INC AC0
N,M
D0
Output x,y.
Get VSY, VSZ.
PNIV
377
173
OUTPUT ,B,C← RA
LOAD OUTPUT
LOAD STATUS & SET FLAG
D0
Output z.
to "Next Vector"



Perspective New Continued

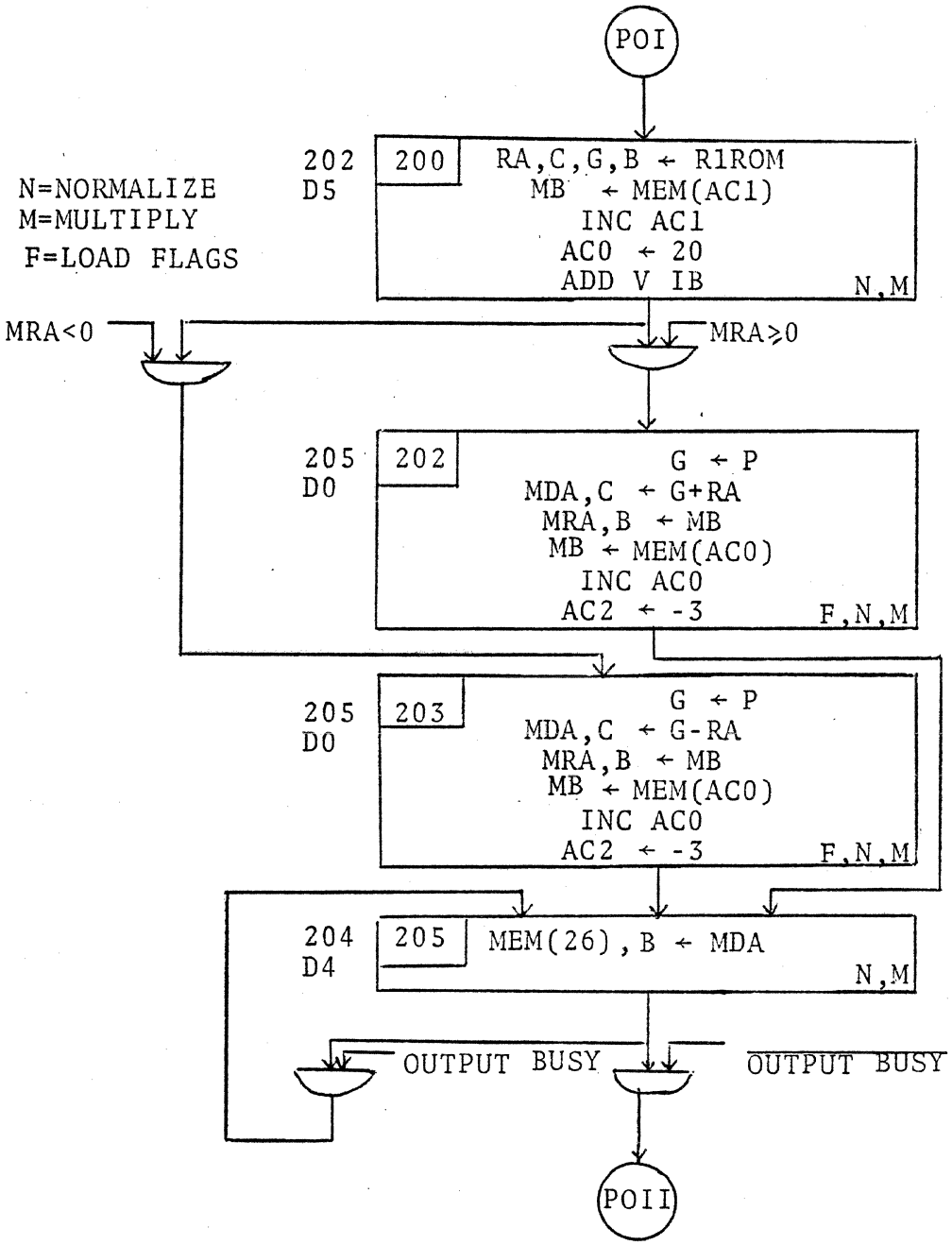
N=NORMALIZE



Get w old.

Address of clipped data.

Wait for ROM.



Get x old.

Set Viewport address.

Check sign of reciprocal.

1/w old.

Get VSX.

Set loop count.

Check normalize overflow.

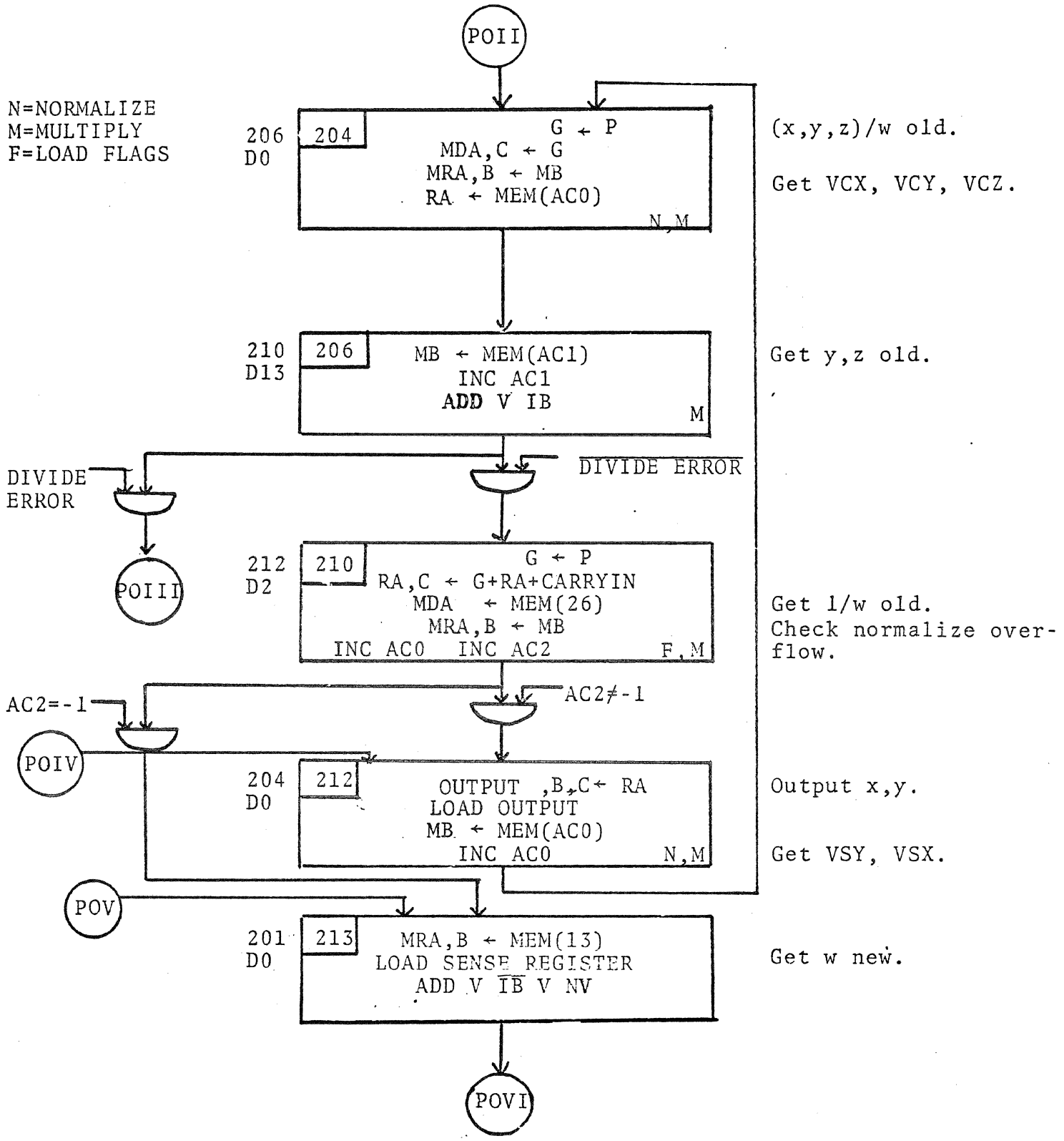
1/w old.

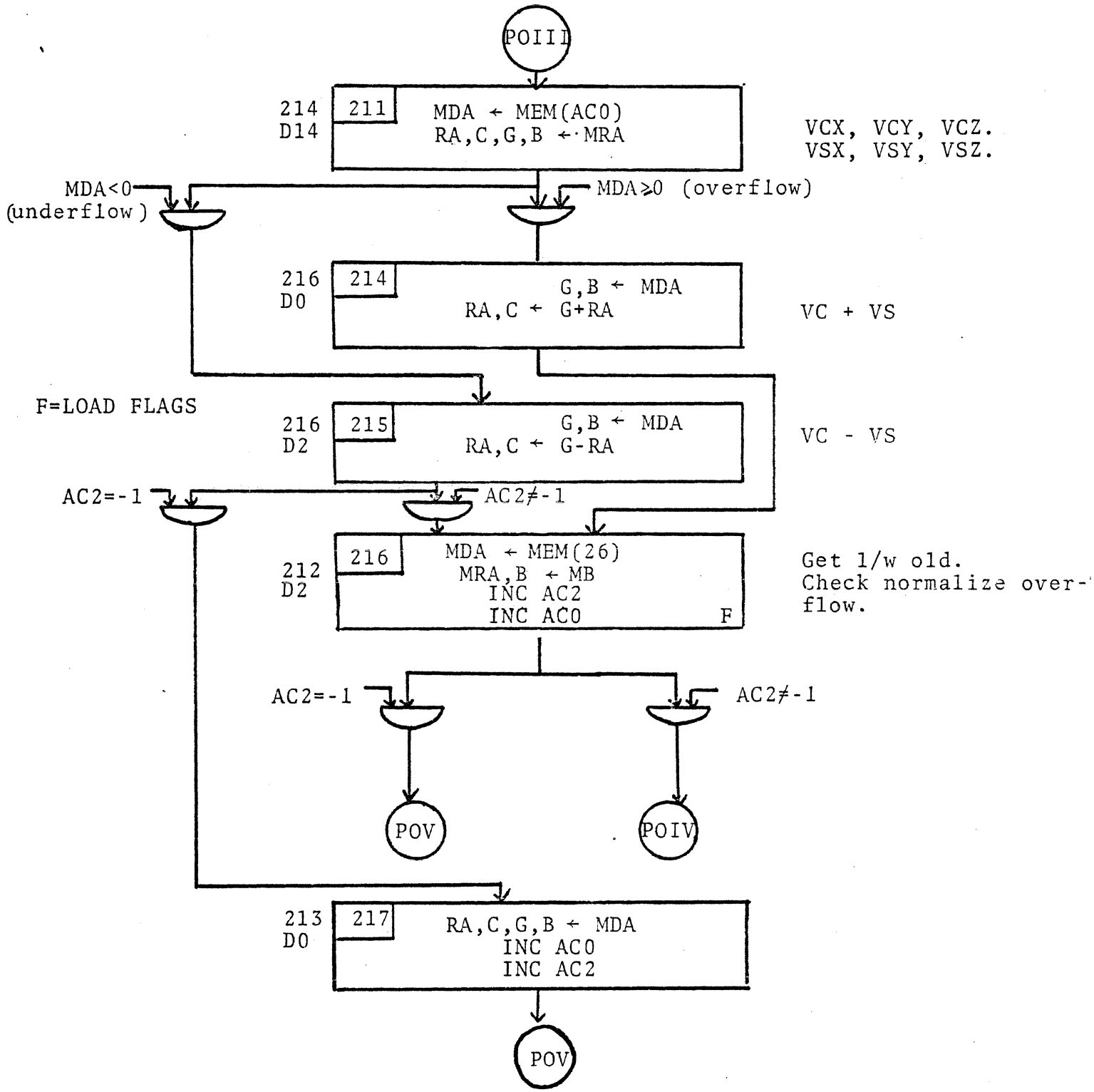
Get VSX.

Set loop count.

Check normalize overflow.

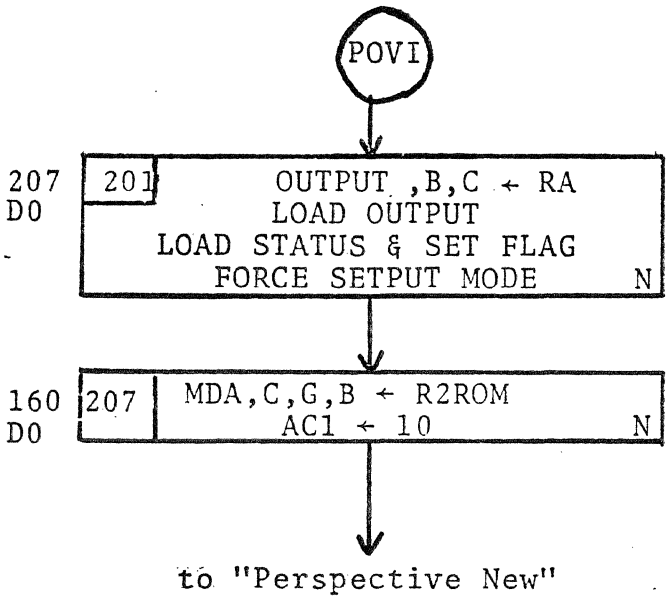
Save 1/w old.


POII
N=NORMALIZE
M=MULTIPLY
F=LOAD FLAGS
206
D0
204
G ← P
MDA,C ← G
MRA,B ← MB
RA ← MEM(AC0)
N,M
(x,y,z)/w old.
Get VCX, VCY, VCZ.
210
D13
206
MB ← MEM(AC1)
INC AC1
ADD V IB
M
Get y,z old.
DIVIDE ERROR
DIVIDE ERROR
POIII
212
D2
210
G ← P
RA,C ← G+RA+CARRYIN
MDA ← MEM(26)
MRA,B ← MB
INC AC0 INC AC2
F,M
Get 1/w old.
Check normalize overflow.
AC2=-1
AC2≠-1
POIV
204
D0
212
OUTPUT ,B,C← RA
LOAD OUTPUT
MB ← MEM(AC0)
INC AC0
N,M
Output x,y.
Get VSY, VSX.
POV
201
D0
213
MRA,B ← MEM(13)
LOAD SENSE REGISTER
ADD V IB V NV
Get w new.
POVI



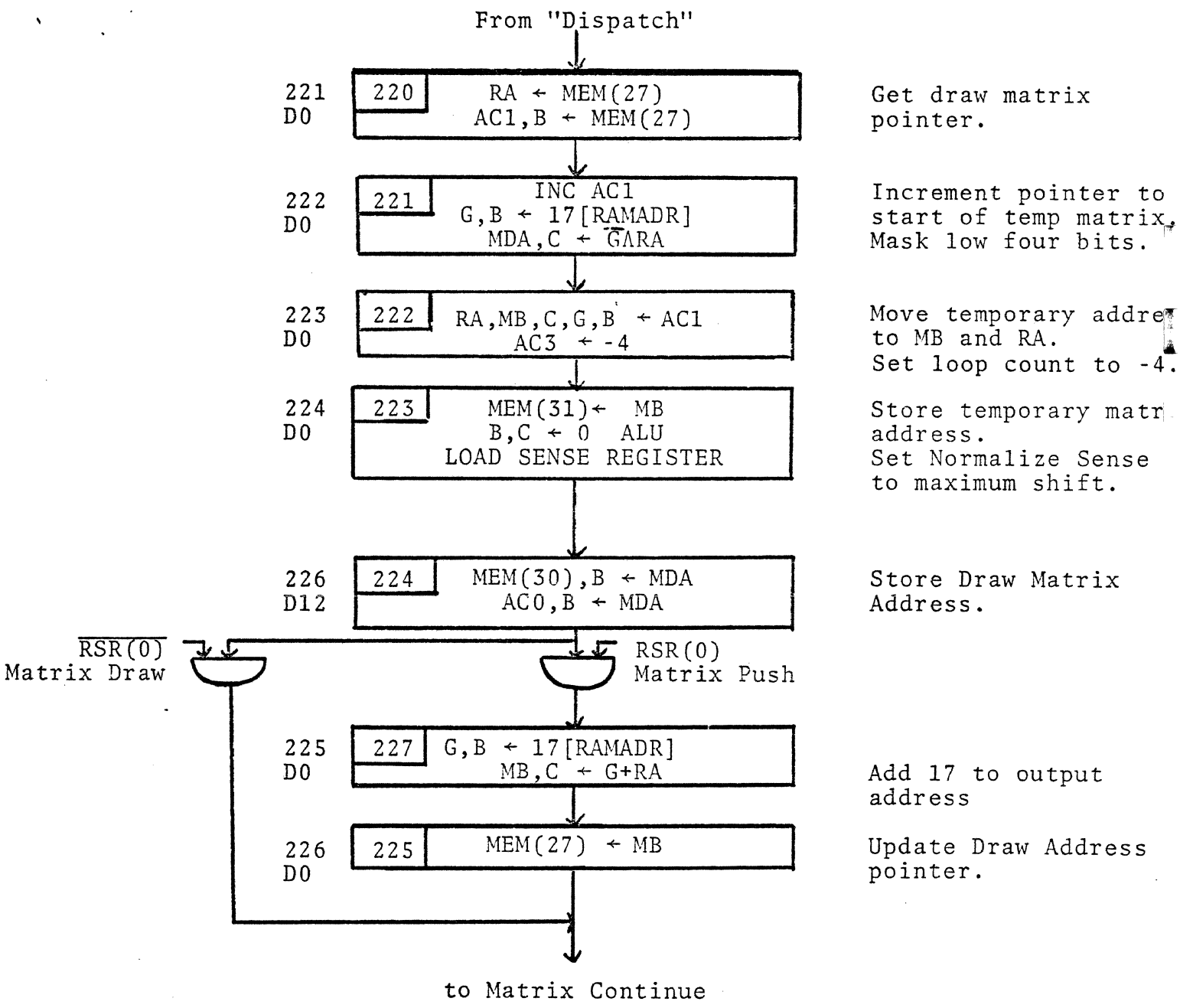
Perspective Old Continued

N=NORMALIZE

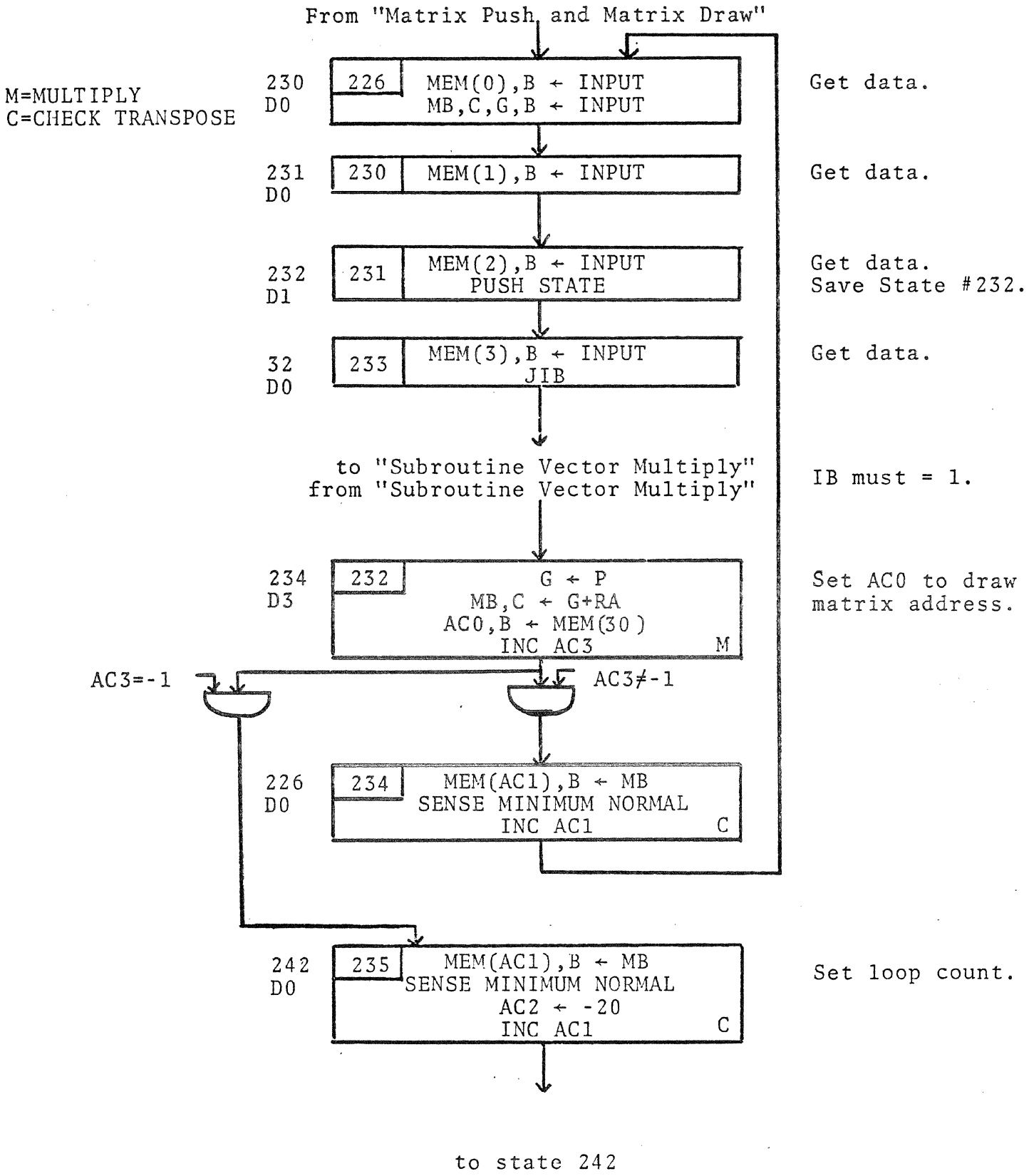


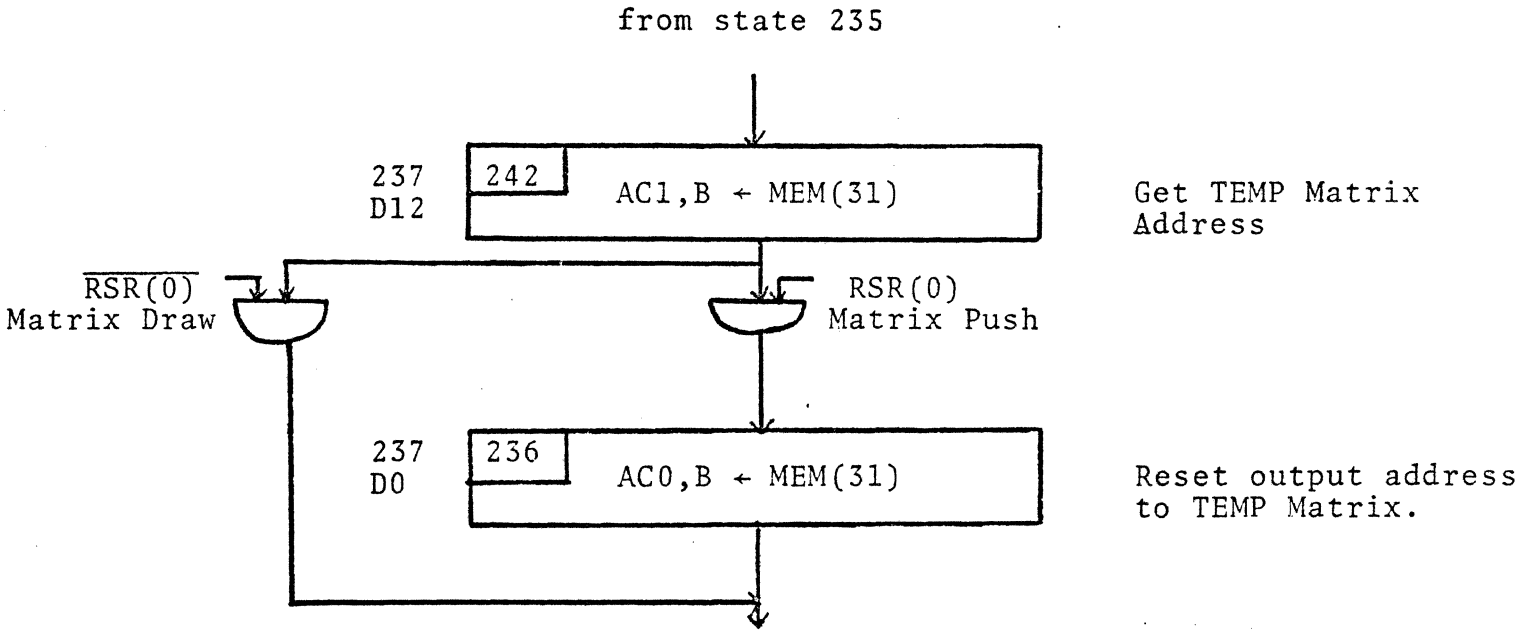
Output z.

Perspective Old Continued

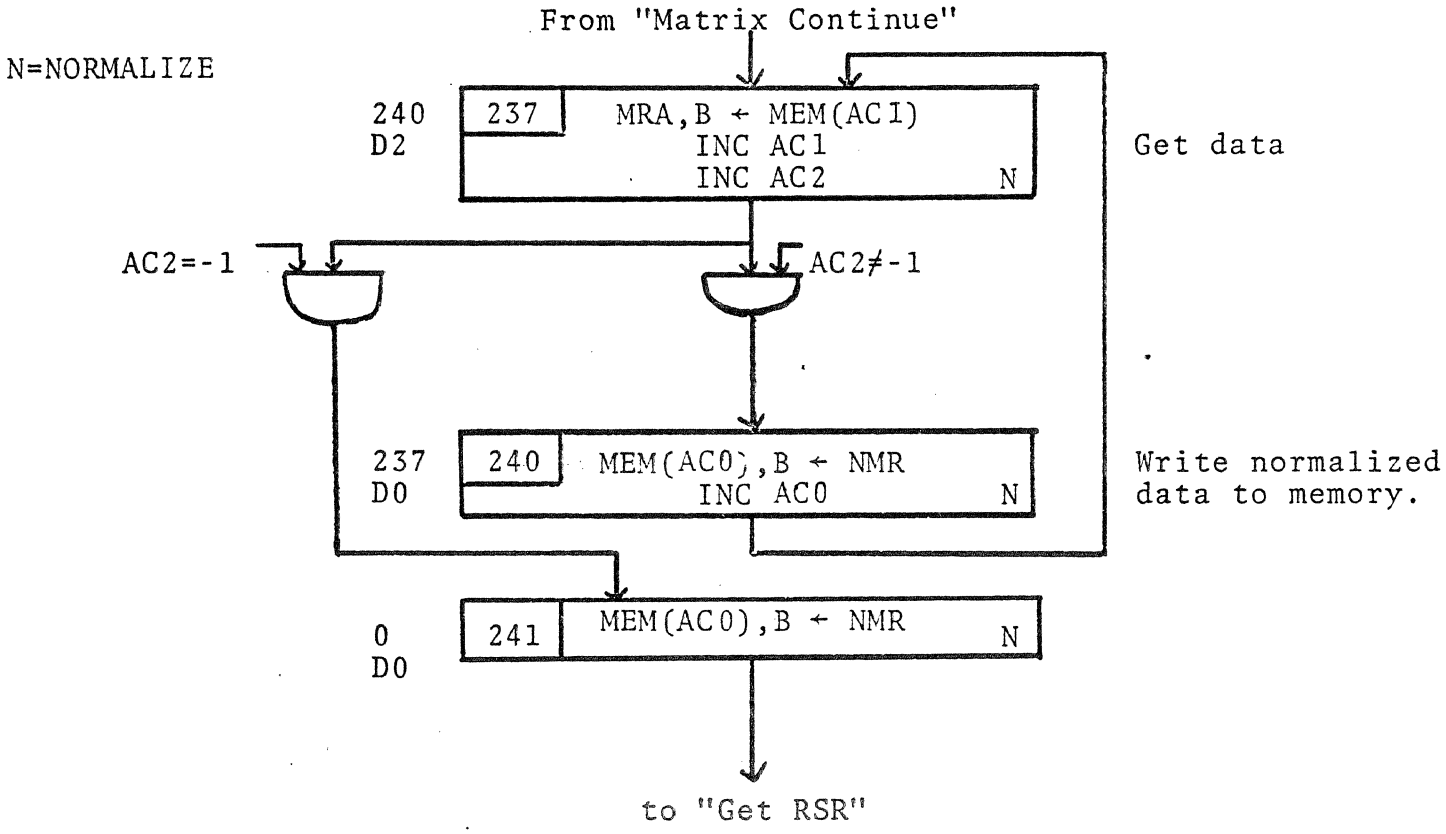


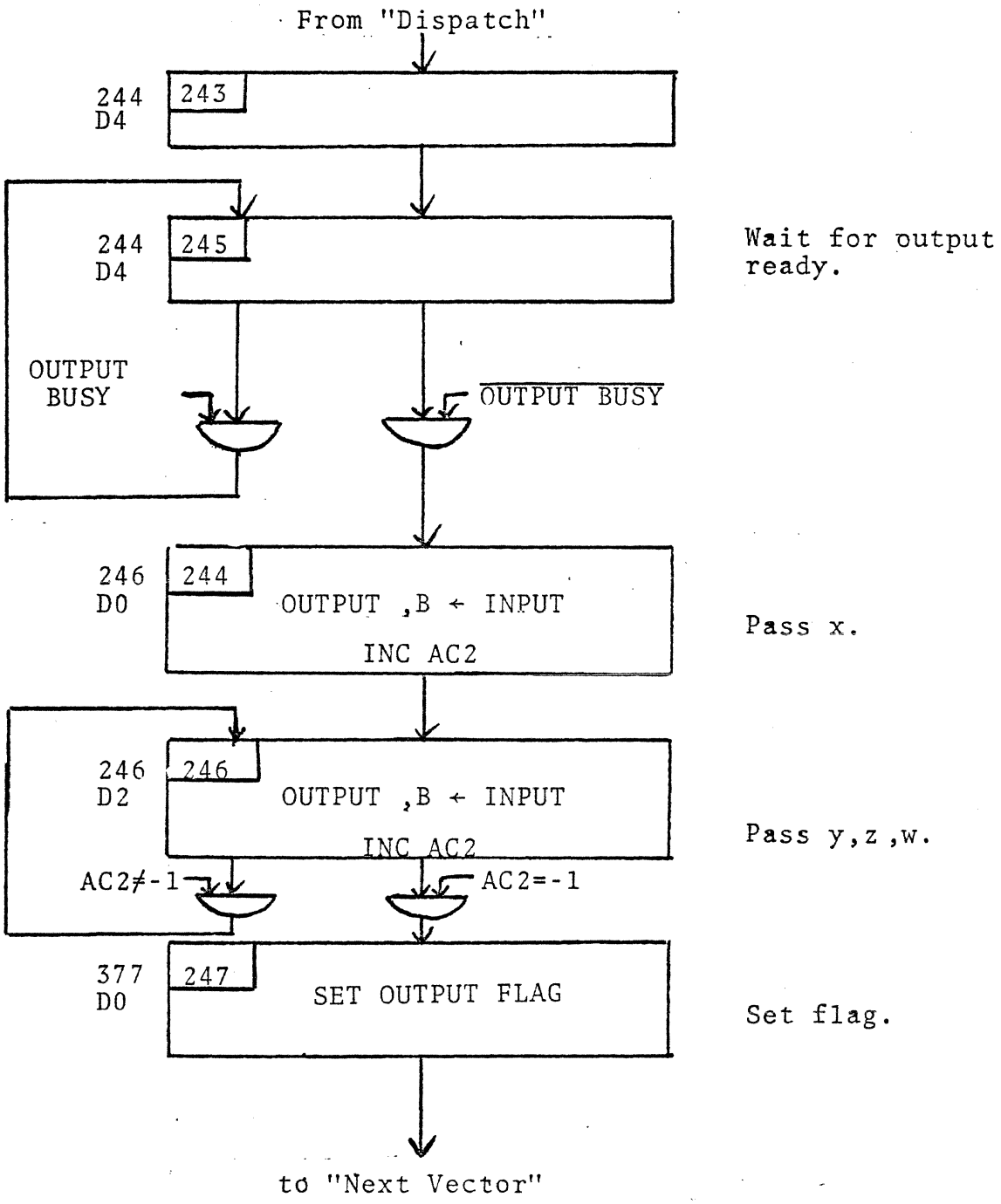
Matrix Push & Matrix Draw


From "Matrix Push and Matrix Draw"
M=MULTIPLY
C=CHECK TRANSPOSE
230
D0
226
MEM(0),B ← INPUT
MB,C,G,B ← INPUT
Get data.
231
D0
230
MEM(1),B ← INPUT
Get data.
232
D1
231
MEM(2),B ← INPUT
PUSH STATE
Get data.
Save State #232.
32
D0
233
MEM(3),B ← INPUT
JIB
Get data.
to "Subroutine Vector Multiply"
from "Subroutine Vector Multiply"
IB must = 1.
234
D3
232
G ← P
MB,C ← G+RA
AC0,B ← MEM(30)
INC AC3
M
Set AC0 to draw
matrix address.
AC3=-1
AC3≠-1
226
D0
234
MEM(AC1),B ← MB
SENSE MINIMUM NORMAL
INC AC1
C
242
D0
235
MEM(AC1),B ← MB
SENSE MINIMUM NORMAL
AC2 ← -20
INC AC1
C
Set loop count.
to state 242

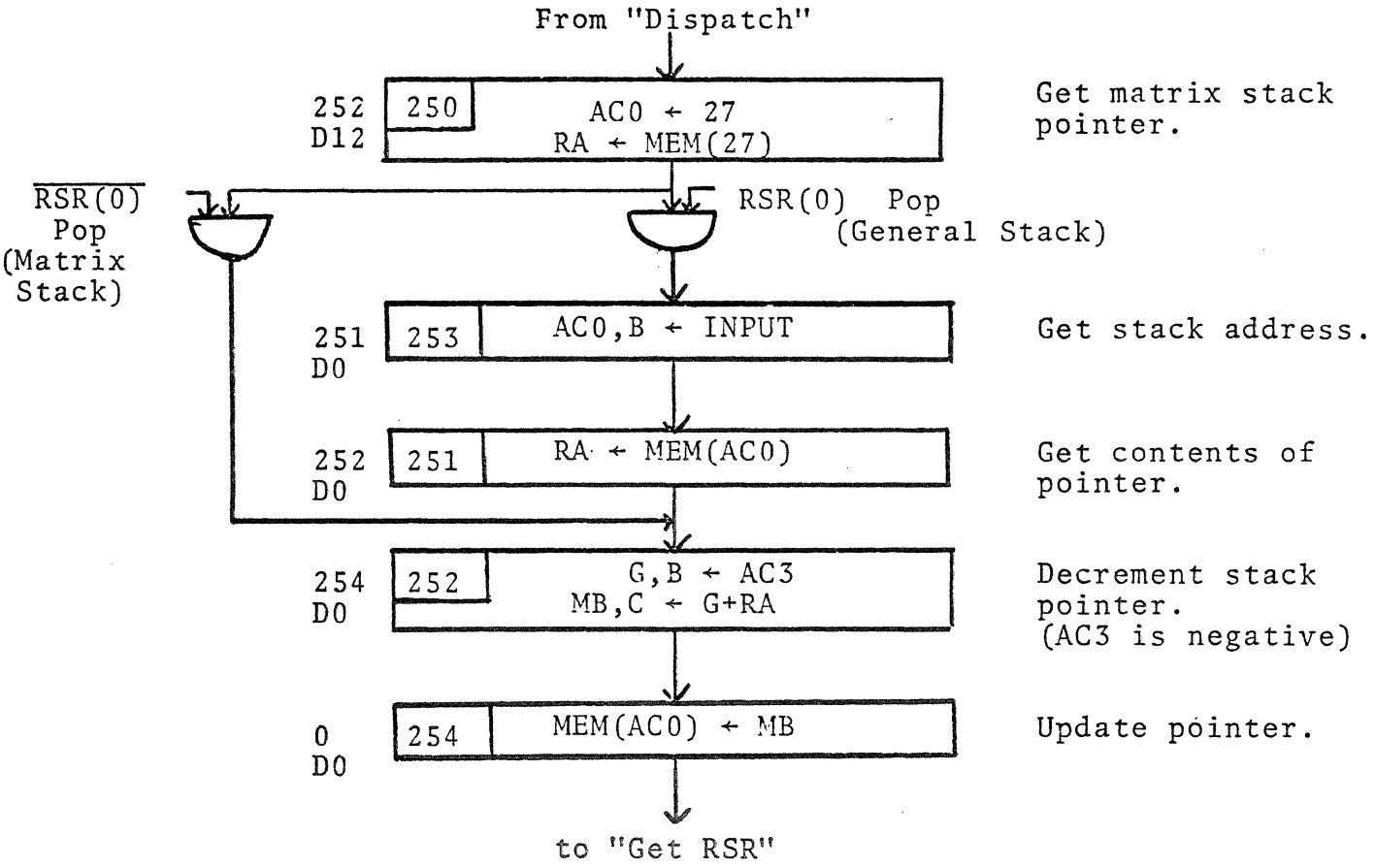


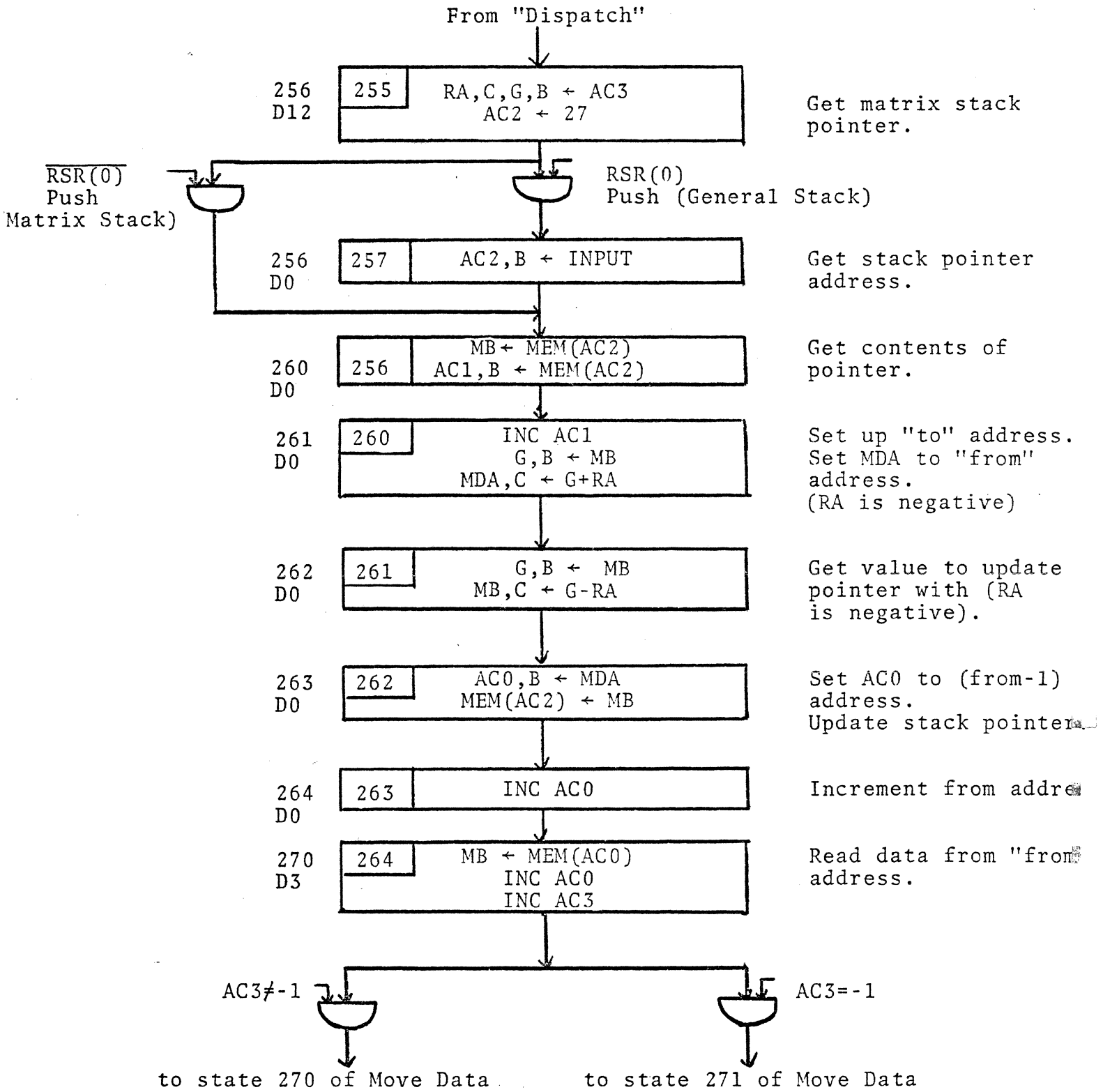
Matrix Continue

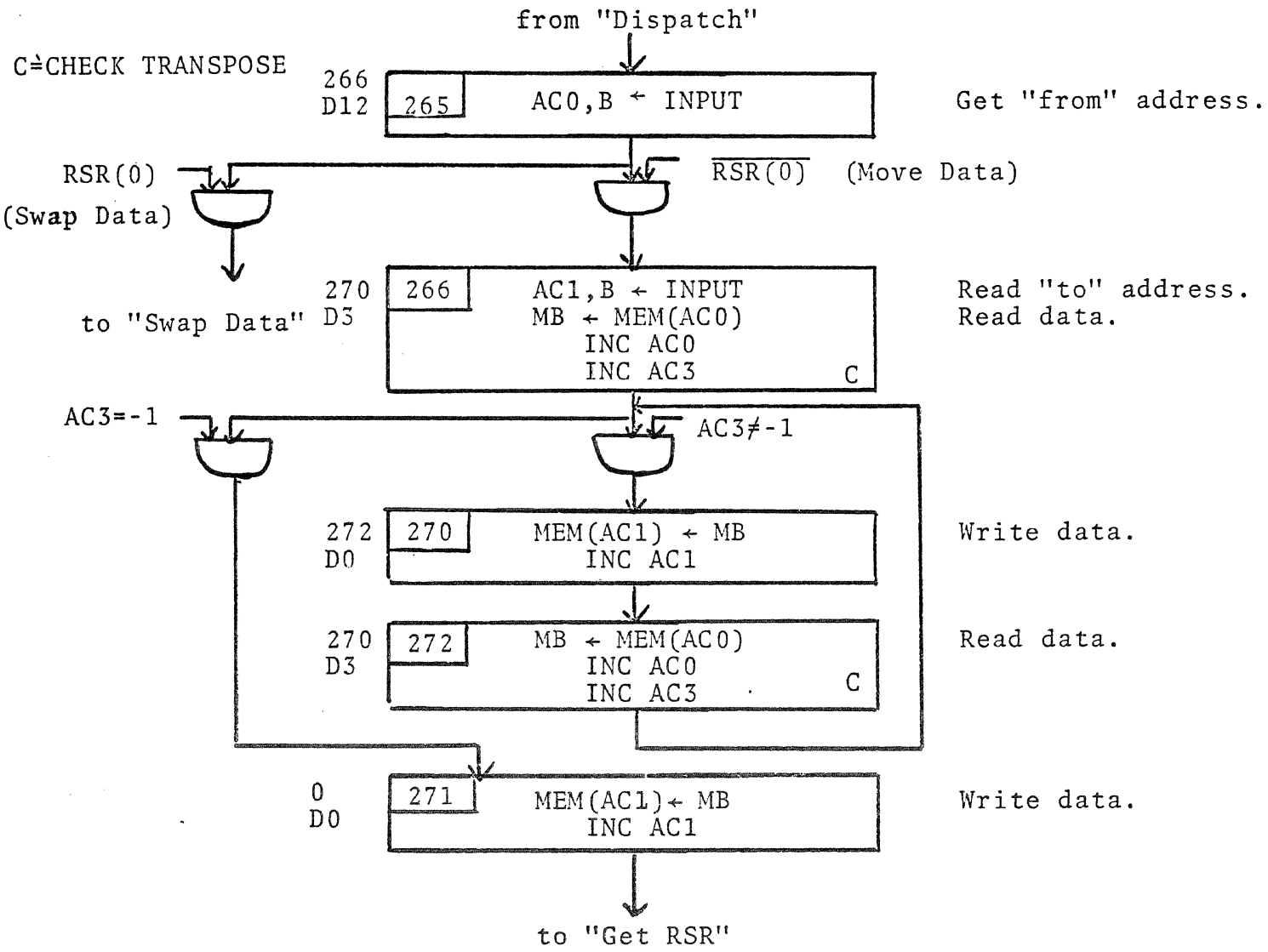

From "Matrix Continue"
N=NORMALIZE
240
D2
237
MRA,B ← MEM(AC1)
INC AC1
INC AC2
N
Get data
AC2=-1
AC2≠-1
237
D0
240
MEM(AC0),B ← NMR
INC AC0
N
Write normalized data to memory.
0
D0
241
MEM(AC0),B ← NMR
N
to "Get RSR"


From "Dispatch"
244
D4
243
244
D4
245
Wait for output ready.
OUTPUT
BUSY
OUTPUT BUSY
246
D0
244
OUTPUT ,B ← INPUT
INC AC2
Pass x.
246
D2
246
OUTPUT ,B ← INPUT
INC AC2
Pass y,z ,w.
AC2≠-1
AC2=-1
377
D0
247
SET OUTPUT FLAG
Set flag.
to "Next Vector"

Pass

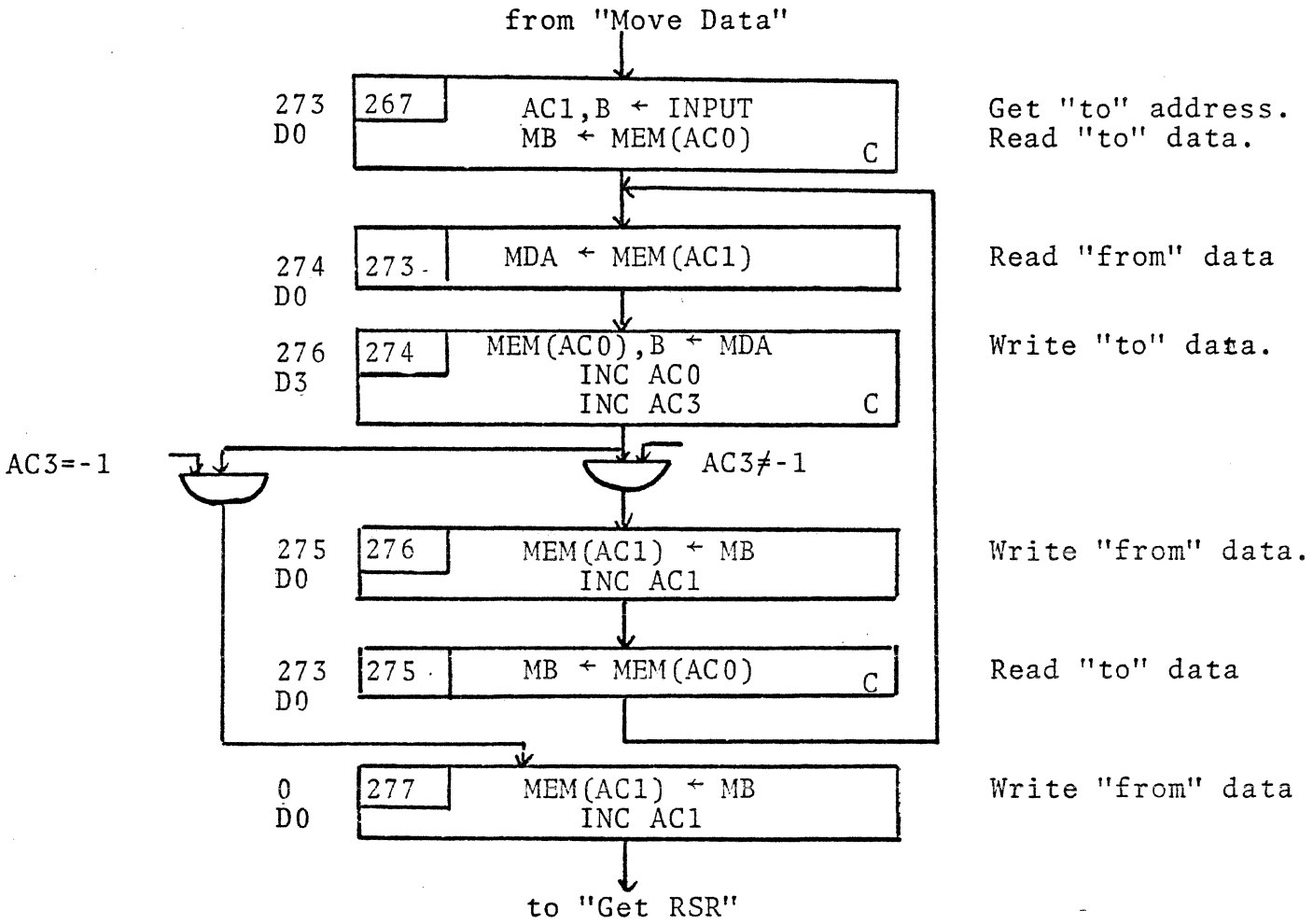

From "Dispatch"
252
D12
250
AC0 ← 27
RA ← MEM(27)
Get matrix stack pointer.
RSR(0) Pop (Matrix Stack)
RSR(0) Pop (General Stack)
251
D0
253
AC0,B ← INPUT
Get stack address.
252
D0
251
RA ← MEM(AC0)
Get contents of pointer.
254
D0
252
G,B ← AC3
MB,C ← G+RA
Decrement stack pointer. (AC3 is negative)
0
D0
254
MEM(AC0) ← MB
Update pointer.
to "Get RSR"


From "Dispatch"
256
D12
255
RA,C,G,B ← AC3
AC2 ← 27
Get matrix stack pointer.
RSR(0)
Push
Matrix Stack)
RSR(0)
Push (General Stack)
256
D0
257
AC2,B ← INPUT
Get stack pointer address.
260
D0
256
MB← MEM(AC2)
AC1,B ← MEM(AC2)
Get contents of pointer.
261
D0
260
INC AC1
G,B ← MB
MDA,C ← G+RA
Set up "to" address.
Set MDA to "from" address.
(RA is negative)
262
D0
261
G,B ← MB
MB,C ← G-RA
Get value to update pointer with (RA is negative).
263
D0
262
AC0,B ← MDA
MEM(AC2) ← MB
Set AC0 to (from-1) address.
Update stack pointer.
264
D0
263
INC AC0
Increment from addre
270
D3
264
MB ← MEM(AC0)
INC AC0
INC AC3
Read data from "from" address.
AC3≠-1
AC3=-1
to state 270 of Move Data
to state 271 of Move Data

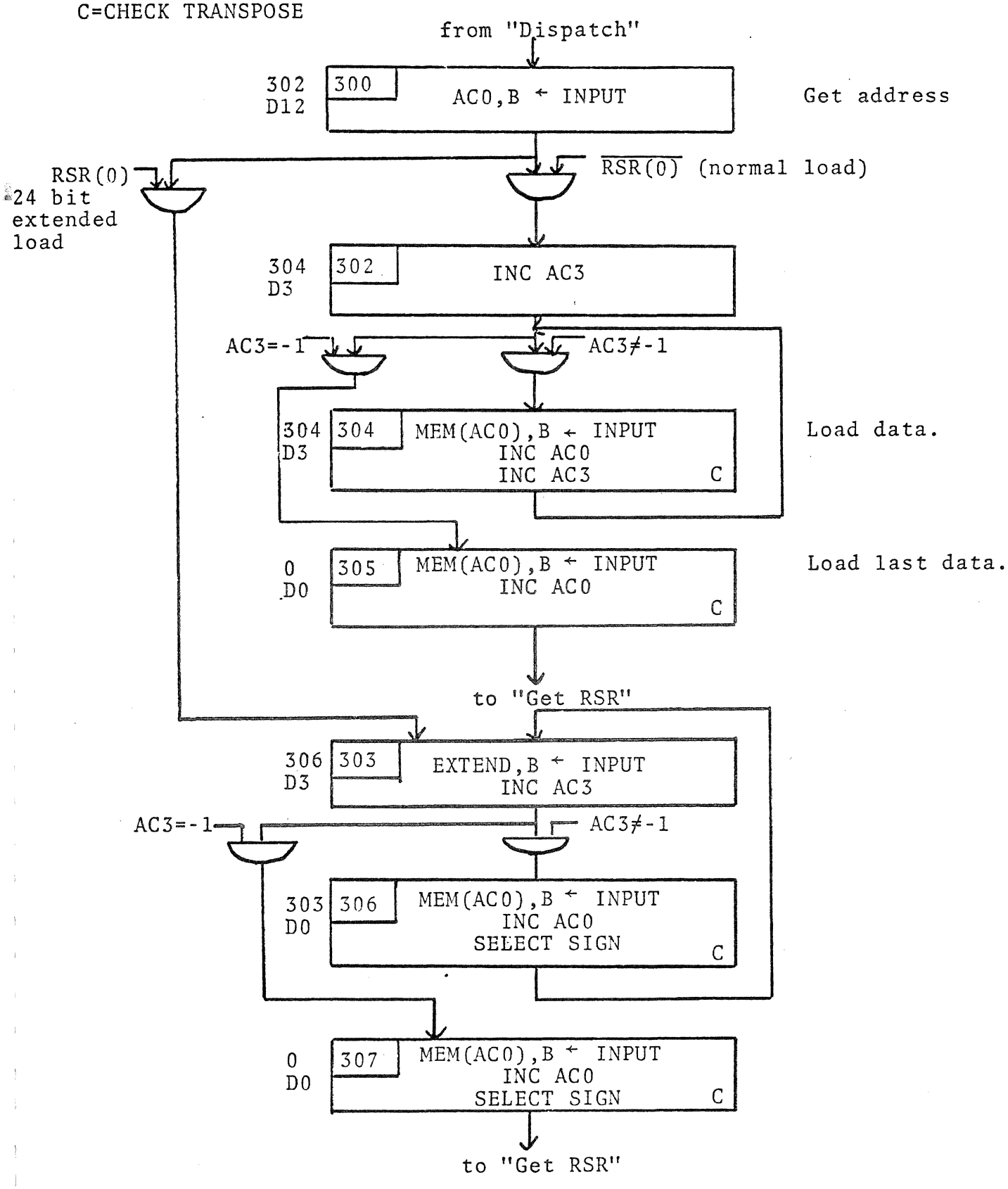

from "Dispatch"
C≐CHECK TRANSPOSE
266
D12
265
AC0,B ← INPUT
Get "from" address.
RSR(0)
(Swap Data)
RSR(0) (Move Data)
to "Swap Data"
270
D3
266
AC1,B ← INPUT
MB ← MEM(AC0)
INC AC0
INC AC3
C
Read "to" address.
Read data.
AC3=-1
AC3≠-1
272
D0
270
MEM(AC1) ← MB
INC AC1
Write data.
270
D3
272
MB ← MEM(AC0)
INC AC0
INC AC3
C
Read data.
0
D0
271
MEM(AC1)← MB
INC AC1
Write data.
to "Get RSR"

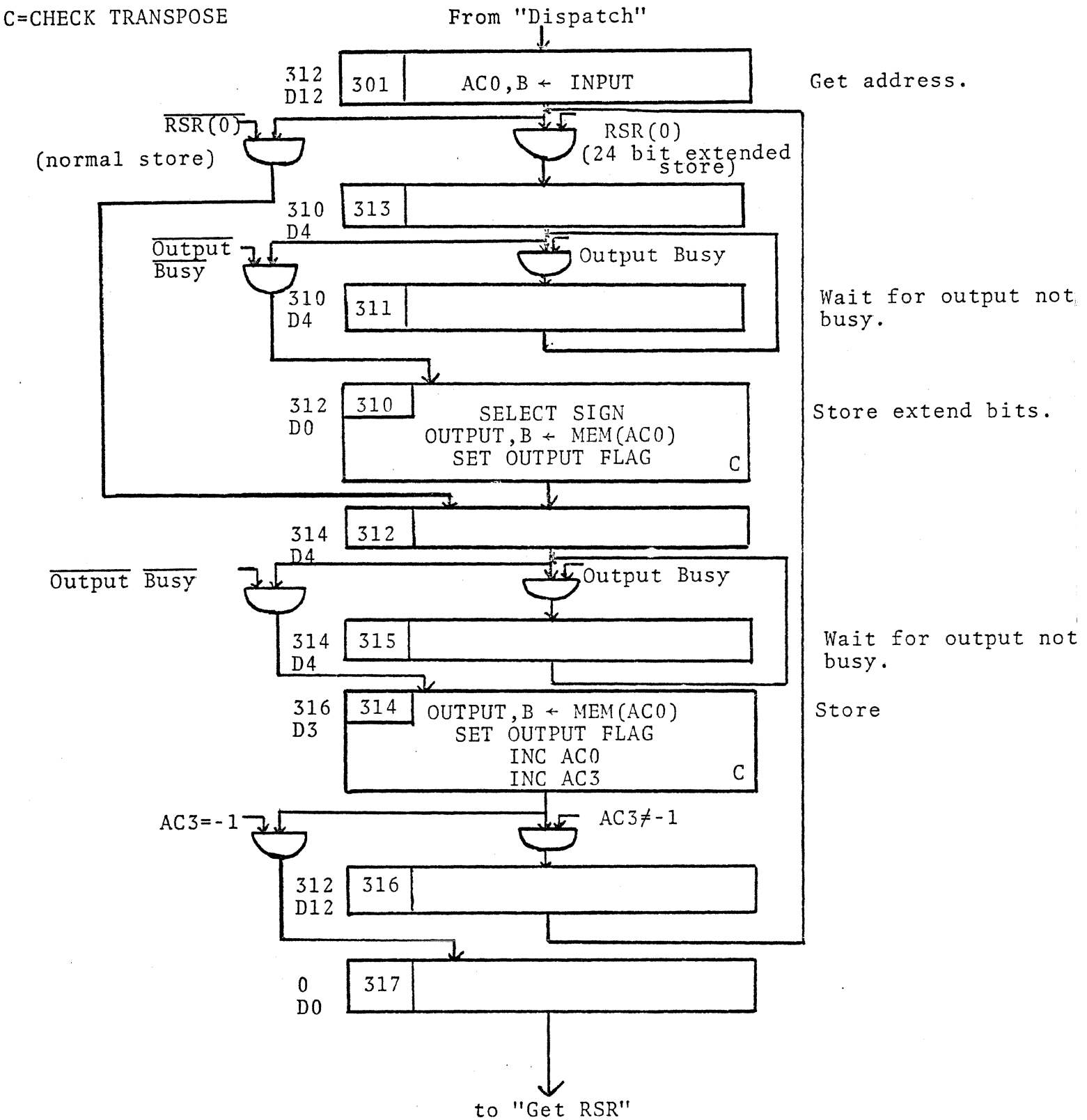
Move Data

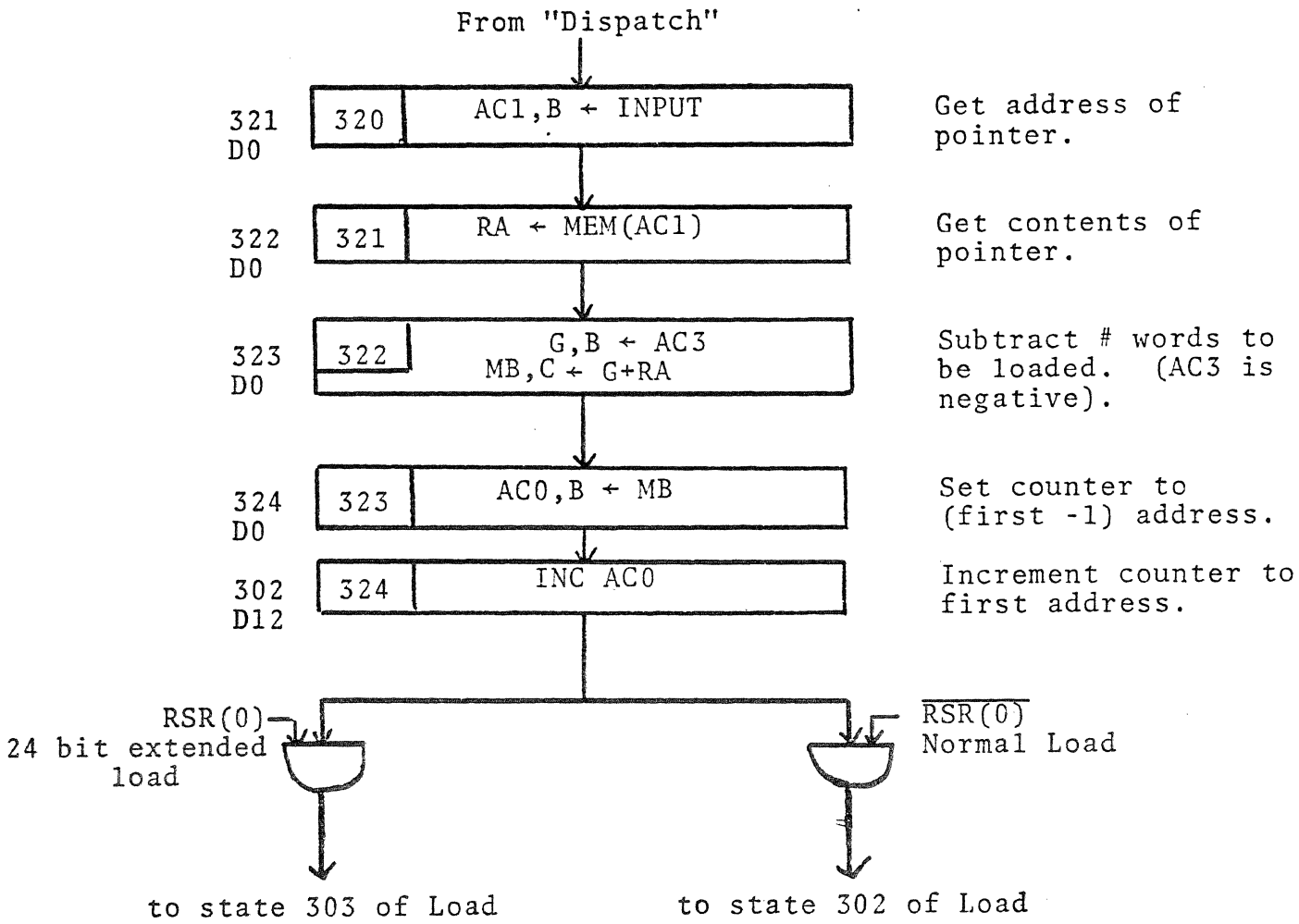
C=CHECK TRANSPOSE



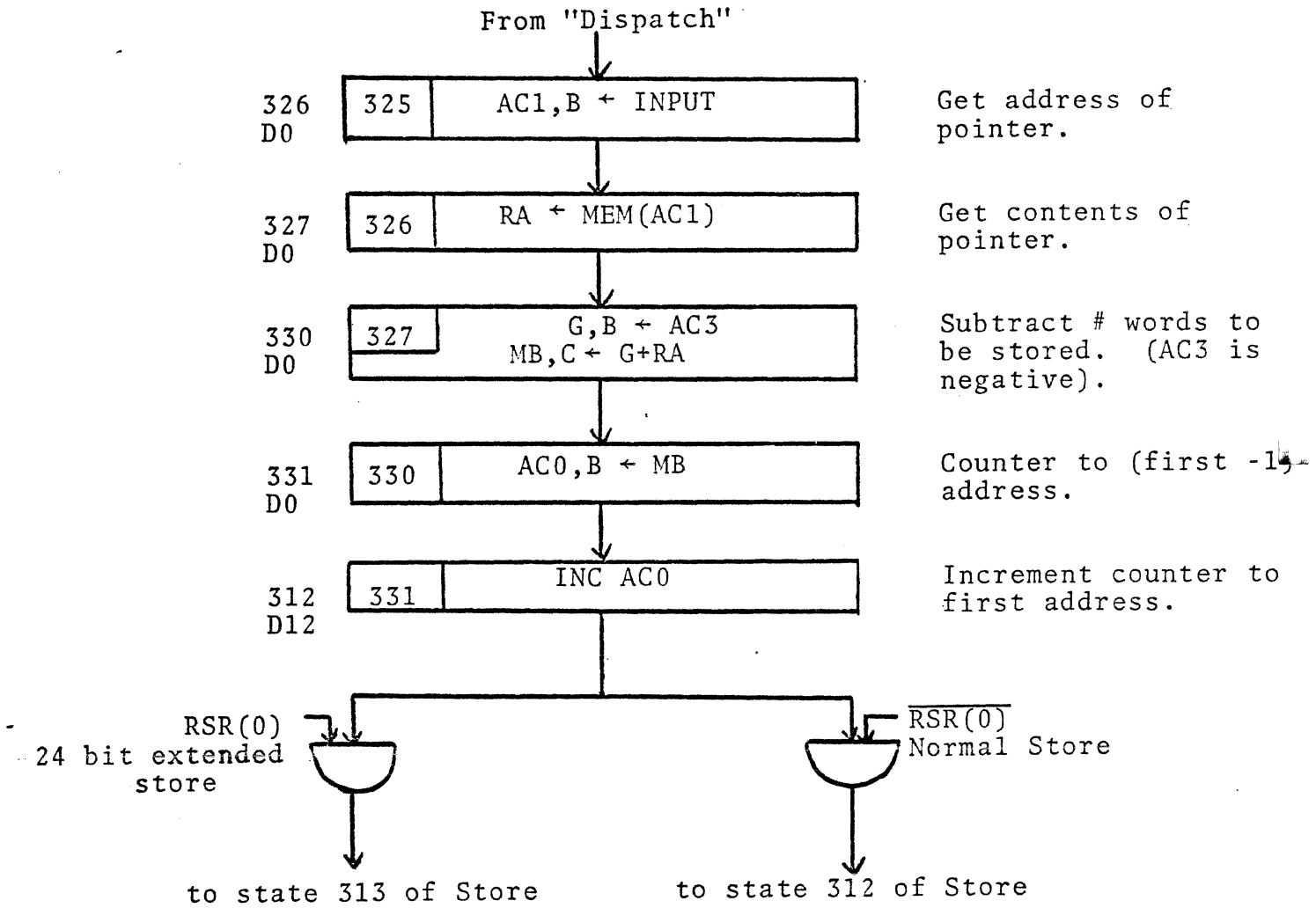
C=CHECK TRANSPOSE



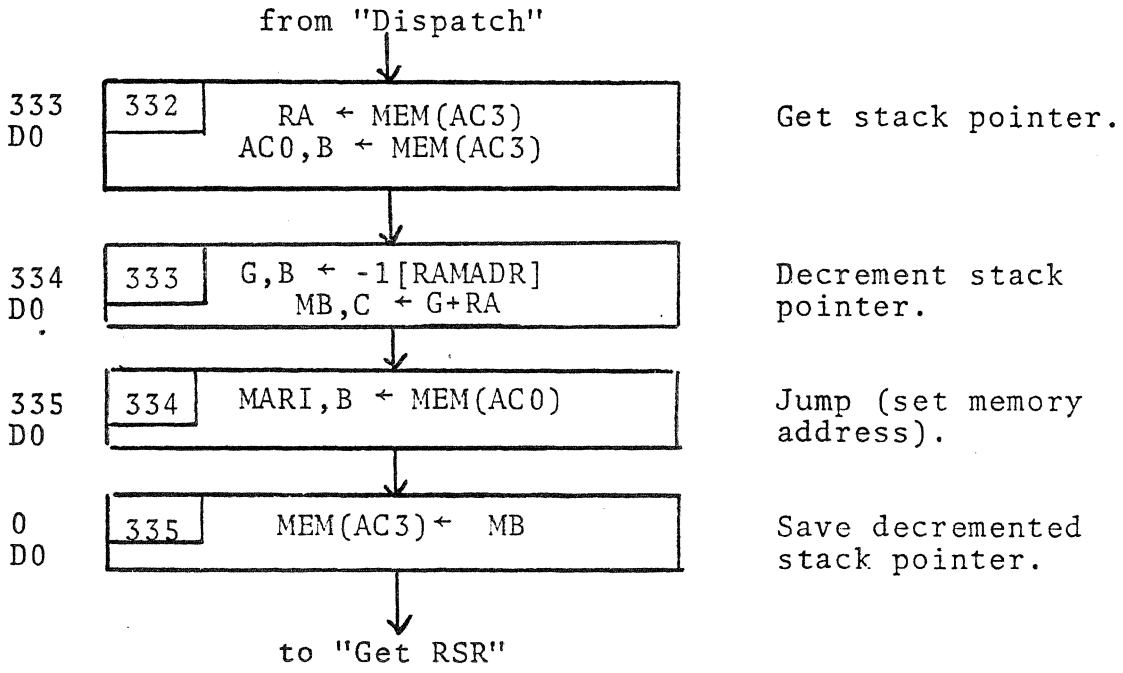

C=CHECK TRANSPOSE
From "Dispatch"
312
D12
301
AC0,B ← INPUT
Get address.
RSR(0)
(normal store)
RSR(0)
(24 bit extended store)
310
D4
313
Output Busy
Output Busy
310
D4
311
Wait for output not busy.
312
D0
310
SELECT SIGN
OUTPUT,B ← MEM(AC0)
SET OUTPUT FLAG
C
Store extend bits.
314
D4
312
Output Busy
Output Busy
314
D4
315
Wait for output not busy.
316
D3
314
OUTPUT,B ← MEM(AC0)
SET OUTPUT FLAG
INC AC0
INC AC3
C
Store
AC3=-1
AC3≠-1
312
D12
316
0
D0
317
to "Get RSR"

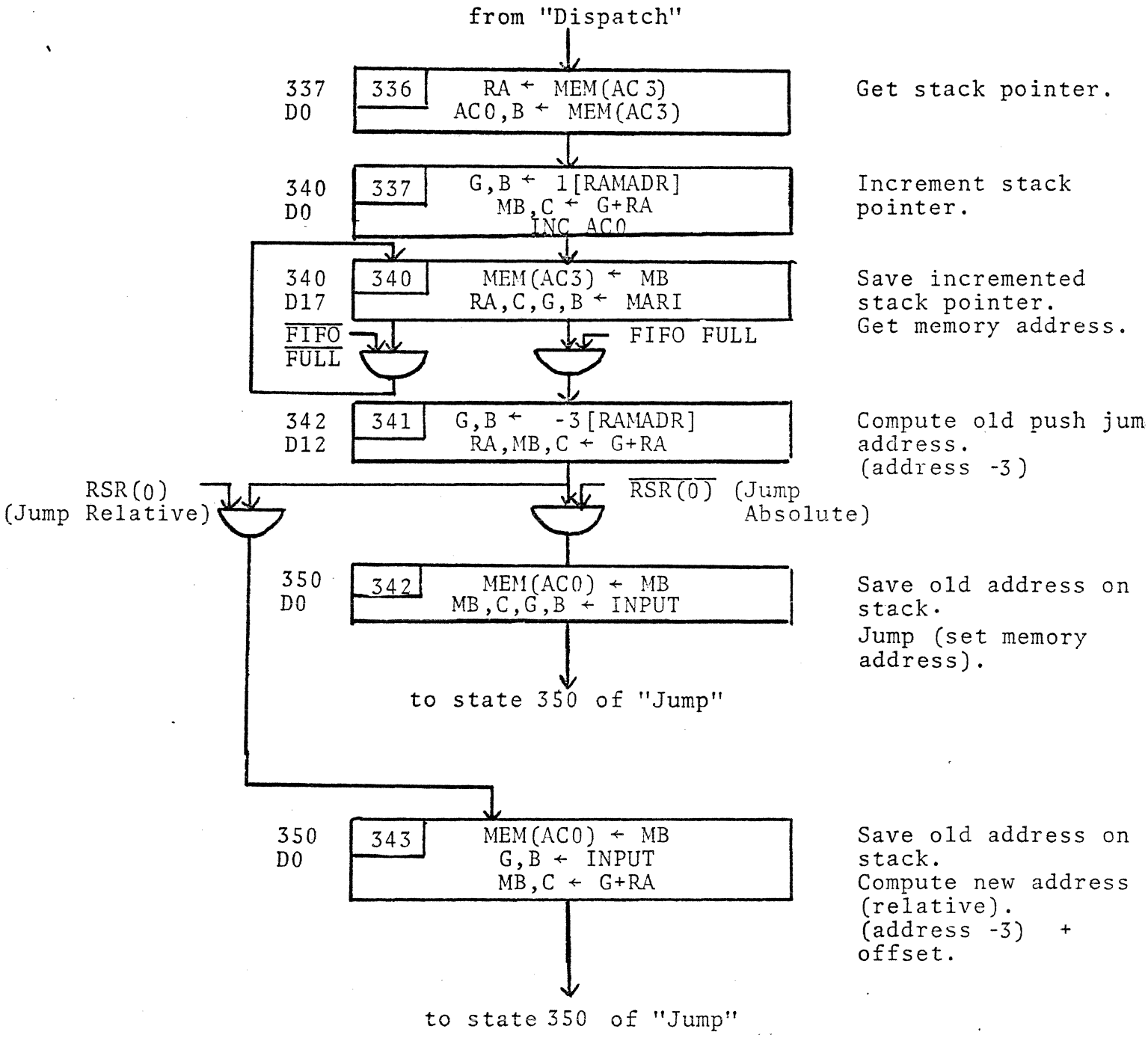

From "Dispatch"
321
D0
320
AC1,B ← INPUT
Get address of pointer.
322
D0
321
RA ← MEM(AC1)
Get contents of pointer.
323
D0
322
G,B ← AC3
MB,C ← G+RA
Subtract # words to be loaded. (AC3 is negative).
324
D0
323
AC0,B ← MB
Set counter to (first -1) address.
302
D12
324
INC AC0
Increment counter to first address.
RSR(0)
24 bit extended load
RSR(0)
Normal Load
to state 303 of Load
to state 302 of Load

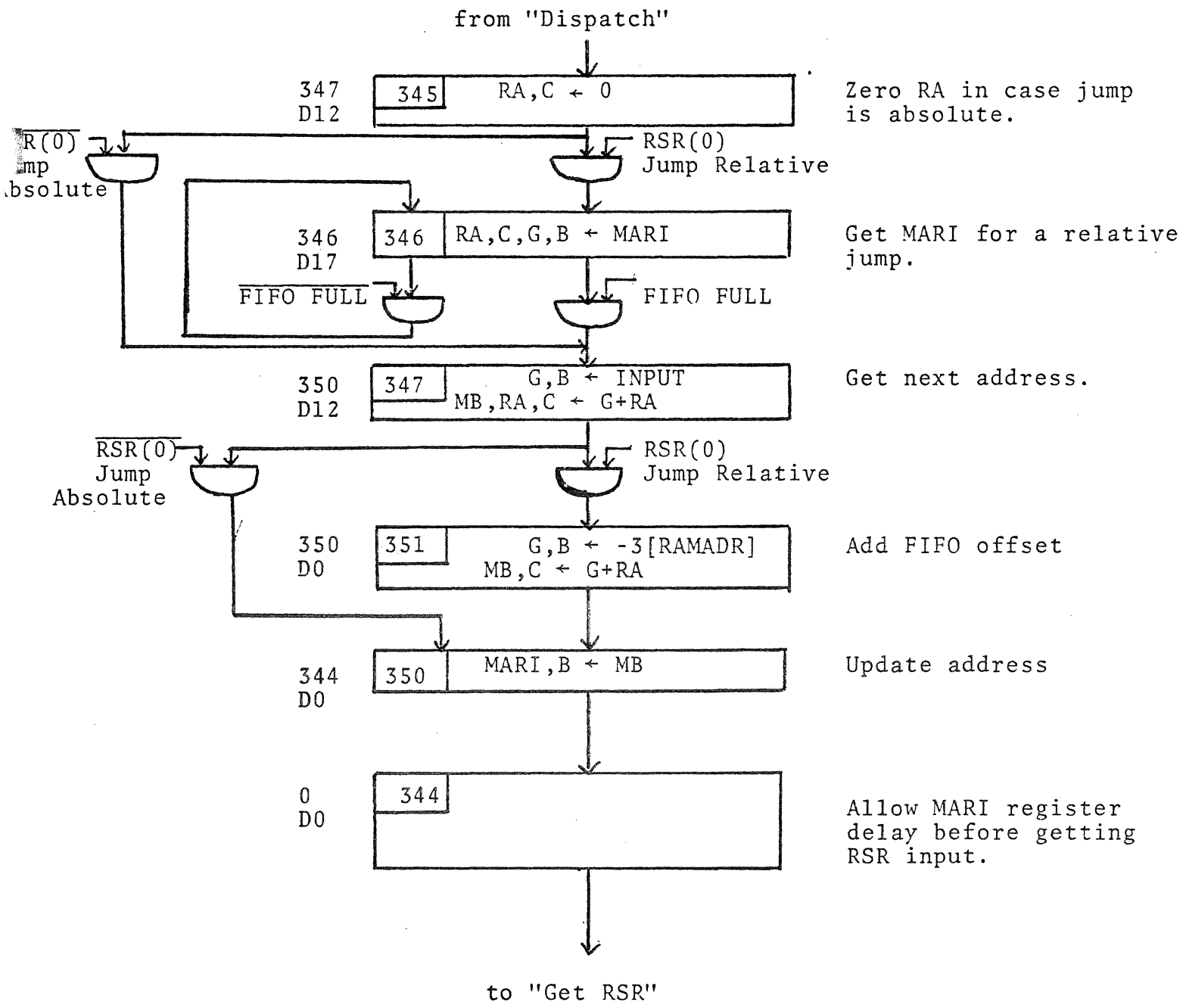
Load Stack

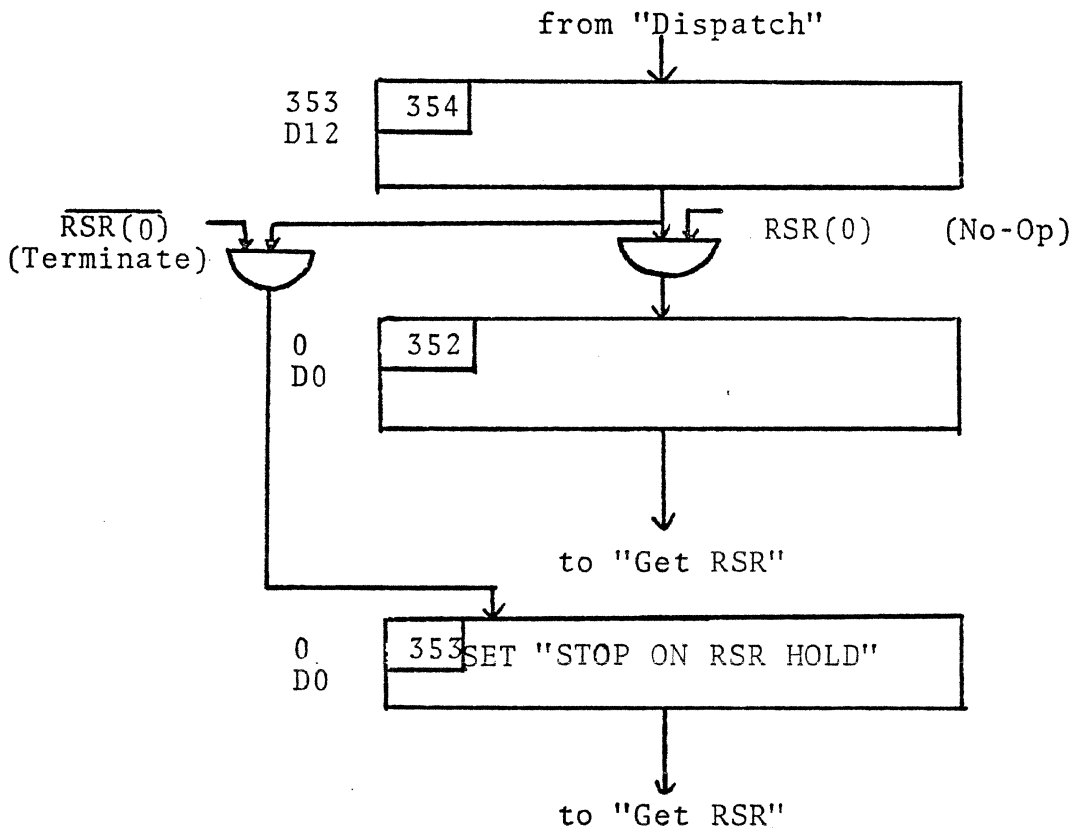

From "Dispatch"
326
D0
325
AC1,B ← INPUT
Get address of pointer.
327
D0
326
RA ← MEM(AC1)
Get contents of pointer.
330
D0
327
G,B ← AC3
MB,C ← G+RA
Subtract # words to be stored. (AC3 is negative).
331
D0
330
AC0,B ← MB
Counter to (first -1) address.
312
D12
331
INC AC0
Increment counter to first address.
RSR(0)
24 bit extended store
RSR(0)
Normal Store
to state 313 of Store
to state 312 of Store

Store Stack


from "Dispatch"
333
D0
332
RA ← MEM(AC3)
AC0,B ← MEM(AC3)
Get stack pointer.
334
D0
333
G,B ← -1[RAMADR]
MB,C ← G+RA
Decrement stack pointer.
335
D0
334
MARI,B ← MEM(AC0)
Jump (set memory address).
0
D0
335
MEM(AC3)← MB
Save decremented stack pointer.
to "Get RSR"


from "Dispatch"
337
D0
336
RA ← MEM(AC3)
AC0,B ← MEM(AC3)
Get stack pointer.
340
D0
337
G,B ← 1[RAMADR]
MB,C ← G+RA
INC AC0
Increment stack pointer.
340
D17
340
MEM(AC3) ← MB
RA,C,G,B ← MARI
Save incremented stack pointer.
Get memory address.
FIFO FULL
FIFO FULL
342
D12
341
G,B ← -3[RAMADR]
RA,MB,C ← G+RA
Compute old push jum address.
(address -3)
RSR(0) (Jump Relative)
RSR(0) (Jump Absolute)
350
D0
342
MEM(AC0) ← MB
MB,C,G,B ← INPUT
Save old address on stack.
Jump (set memory address).
to state 350 of "Jump"
350
D0
343
MEM(AC0) ← MB
G,B ← INPUT
MB,C ← G+RA
Save old address on stack.
Compute new address (relative).
(address -3) + offset.
to state 350 of "Jump"


from "Dispatch"
347
D12
345
RA,C ← 0
Zero RA in case jump is absolute.
R(0)
mp
bsolute
RSR(0)
Jump Relative
346
D17
346
RA,C,G,B ← MARI
Get MARI for a relative jump.
FIFO FULL
FIFO FULL
350
D12
347
G,B ← INPUT
MB,RA,C ← G+RA
Get next address.
RSR(0)
Jump
Absolute
RSR(0)
Jump Relative
350
D0
351
G,B ← -3[RAMADR]
MB,C ← G+RA
Add FIFO offset
344
D0
350
MARI,B ← MB
Update address
0
D0
344
Allow MARI register delay before getting RSR input.
to "Get RSR"



No-op and Terminate

MAP STATE/NAME CROSS-REFERENCE TABLE

0	Get RSR
1	Dispatch
2	Absolute Input
3	" "
4	" "
5	" "
6	Relative Input
7	" "
10	" "
11	" "
12	" "
13	" "
14	" "
15	Origin Offset
16	" "
17	" "
20	" "
21	" "
22	Set Input Base
23	" " "
24	" " "
25	" " "
26	Vector Multiply
27	" "
30	" "
31	" "
32	Vector Multiply Subroutine
33	" " "
34	" " "
35	" " "
36	" " "
37	" " "
40	Vector Multiply Subroutine Continued
41	" " " "
42	" " " "

```
43    Vector Multiply Subroutine Continued
44    "      "        "           "
45    Vector Normalize
46    "      "
47    "      "
50    "      "
51    "      "
52    Output New Vector
53    "      "   "
54    "      "   "
55    "      "   "
56    Setpt Check
57    Drawto Check (z)
60    Setpt Check
61    "      "
62    "      "
63    "      "
64    "      "
65    Drawto Check (z)
66    "      "       "
67    Drawto Check (w-z)
70    Drawto Check (z)
71    "      "       "
72    "      "       "
73    Drawto Check (w-z)
74    "      "       "
75    Drawto Check (w+y)
76    Drawto Check (w-z)
77    "      "       "
100   "      "       "
101   Drawto Check (w+y)
102   "      "       "
103   Drawto Check (w-y)
104   Drawto Check (w+y)
105   "      "       "
106   "      "       "
107   Drawto Check (w-y)
110   "      "       "
```

```
111   Drawto Check (w+x)
112   Drawto Check (w-y)
113   "      "       "
114   "      "       "
115   Drawto Check (w+x)
116   "      "       "
117   Drawto Check (w-x)
120   Drawto Check (w+x)
121   "      "       "
122   "      "       "
123   Drawto Check (w-x)
124   "      "       "
125   "      "       "
126   "      "       "
127   "      "       "
130   Intersect Subroutine
131   "         "
132   "         "
133   "         "
134   "         "
135   "         "
136   "         "
137   "         "
140   Intersect Subroutine Continued
141   "         "          "
142   "         "          "
143   "         "          "
144   "         "          "
145   "         "          "
146   "         "          "
147   "         "          "
150   "         "          "
151   "         "          "
152   "         "          "
153   "         "          "
154   "         "          "
155   "         "          "
156   "         "          "
```

```
157   Perspective Old
160   Perspective New
161   Perspective Old
162   Perspective New
163   "           "
164   Perspective New Continued
165   Perspective New
166   Perspective New Continued
167   Perspective Old
170   Perspective New Continued
171   "           "   "
172   "           "   "
173   "           "   "
174   "           "   "
175   "           "   "
176   "           "   "
177   "           "   "
200   Perspective Old Continued
201   "           "   "
202   "           "   "
203   "           "   "
204   "           "   "
205   "           "   "
206   "           "   "
207   "           "   "
210   "           "   "
211   "           "   "
212   "           "   "
213   "           "   "
214   "           "   "
215   "           "   "
216   "           "   "
217   "           "   "
220   Matrix Push & Matrix Draw
221   "      "      "      "
222   "      "      "      "
223   "      "      "      "
224   "      "      "      "
```

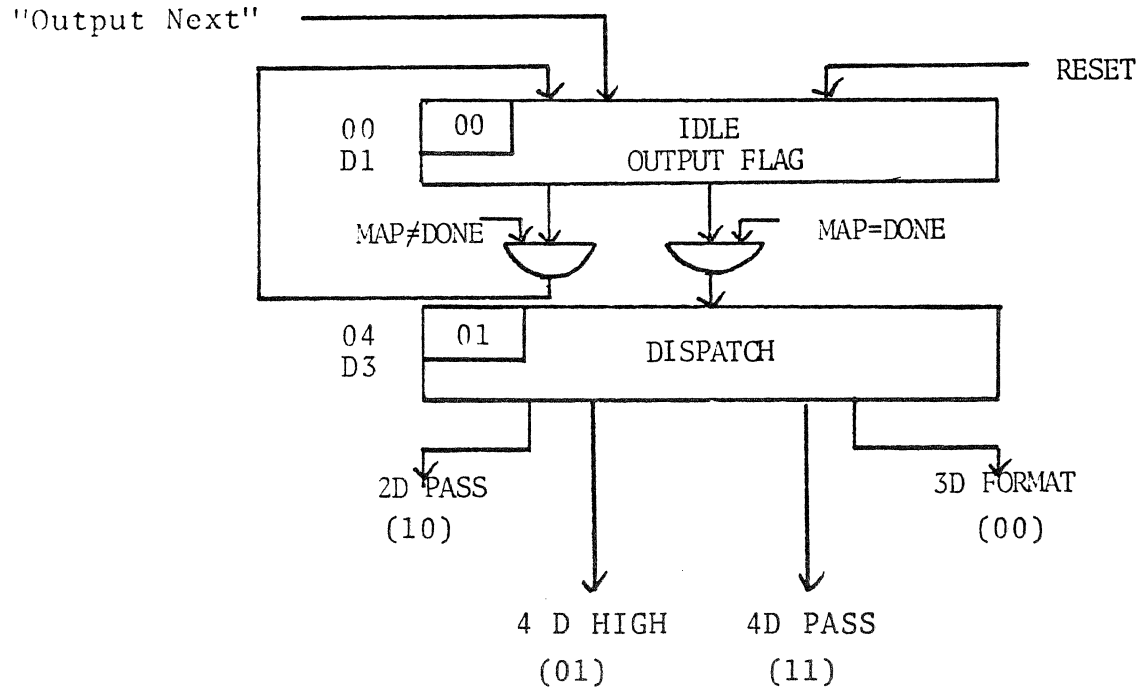
```
225   Matrix Push & Matrix Draw
226   Matrix Continue
227   Matrix Push & Matrix Draw
230   Matrix Continue
231   "      "
232   "      "
233   "      "
234   "      "
235   "      "
236   "      "
237   Matrix Normalize
240   "      "
241   "      "
242   Matrix Continue
243   Pass
244   "
245   "
246   "
247   "
250   Pop
251   "
252   "
253   "
254   "
255   Push
256   "
257   "
260   "
261   "
262   "
263   "
264   "
265   Move Data
266   "
267   Swap Data
270   Move Data
271   "    "
272   "    "
273   Swap Data
```

```
274   Swap Data
275   "    "
276   "    "
277   "    "
300   Load
301   Store
302   Load
303   "
304   "
305   "
306   "
307   "
310   Store
311   "
312   "
313   "
314   "
315   "
316   "
317   "
320   Load Stack
321   "    "
322   "    "
323   "    "
324   "    "
325   Store Stack
326   "     "
327   "     "
330   "     "
331   "     "
332   Pop-Jump (return)
333   "   "     "
334   "   "     "
335   "   "     "
336   Push Jump
337   "    "
340   "    "
341   "    "
```

```
342   Push Jump
343   "    "
344   Jump
345   "
346   "
347   "
350   "
351   "
352   No-op & Terminate
353   "       "
354   "       "
355
356
357
360
361
362
363
364
365
366
367
370
371
372
373
374
375
376
377   Next Vector
```

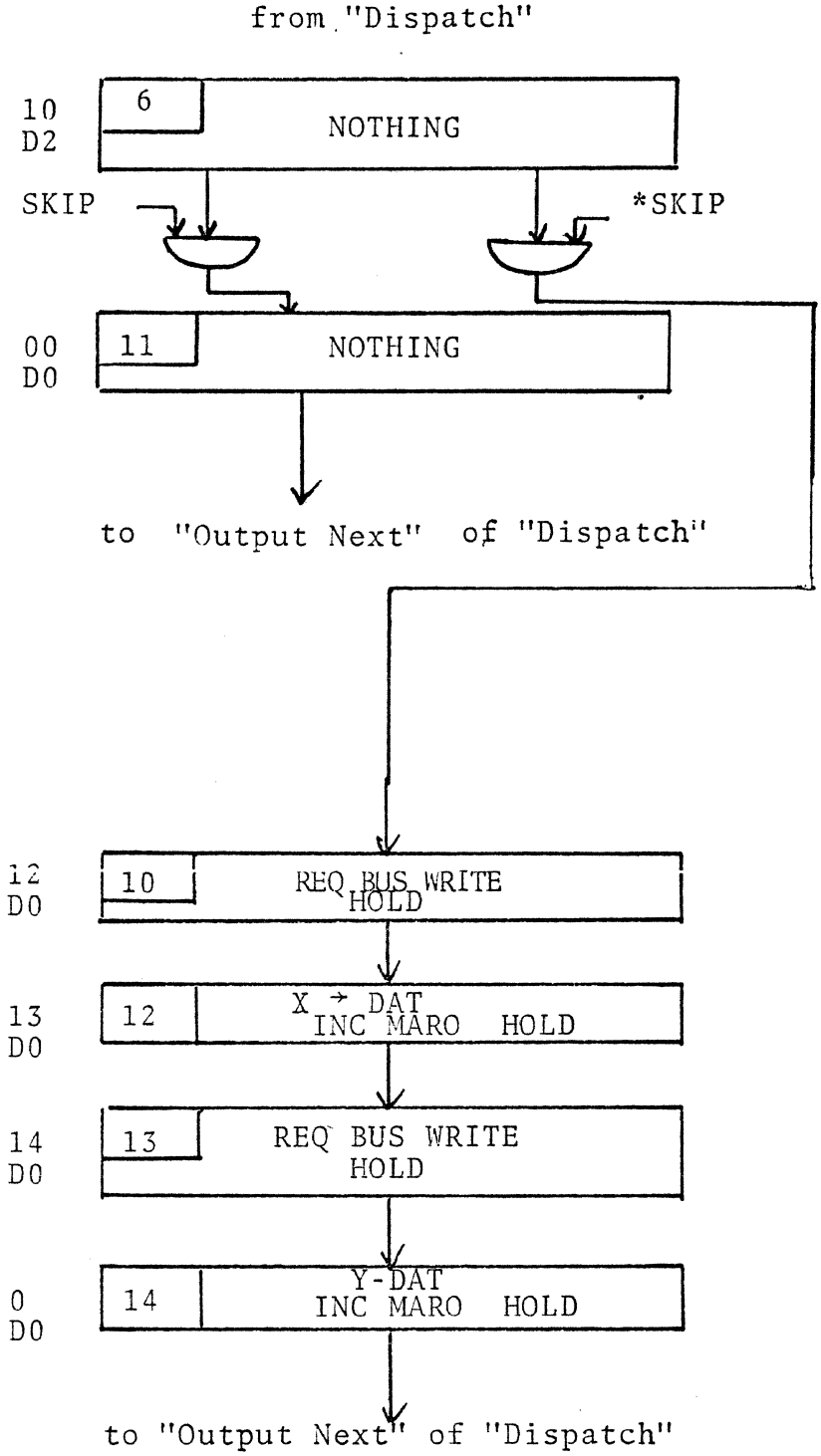
APPENDIX B

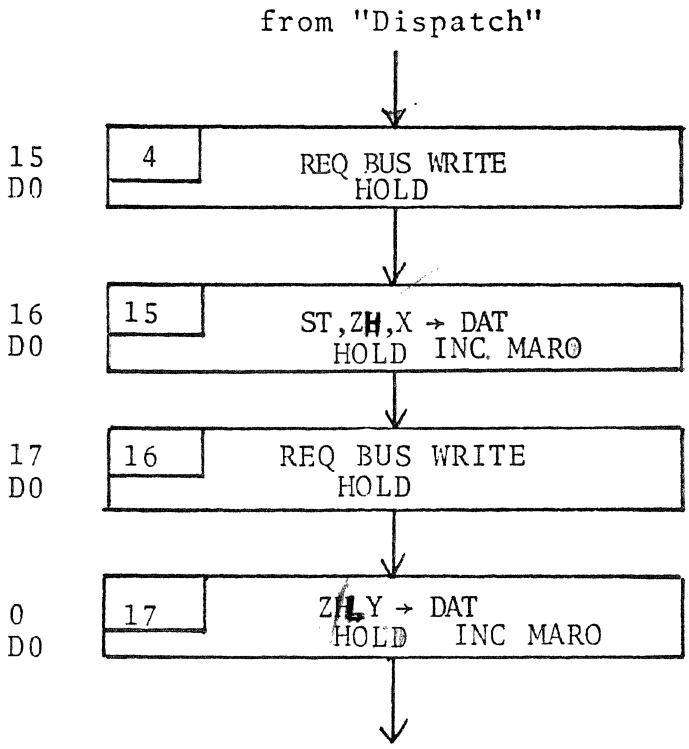
MAP OUTPUT SEQUENCER STATE DIAGRAM



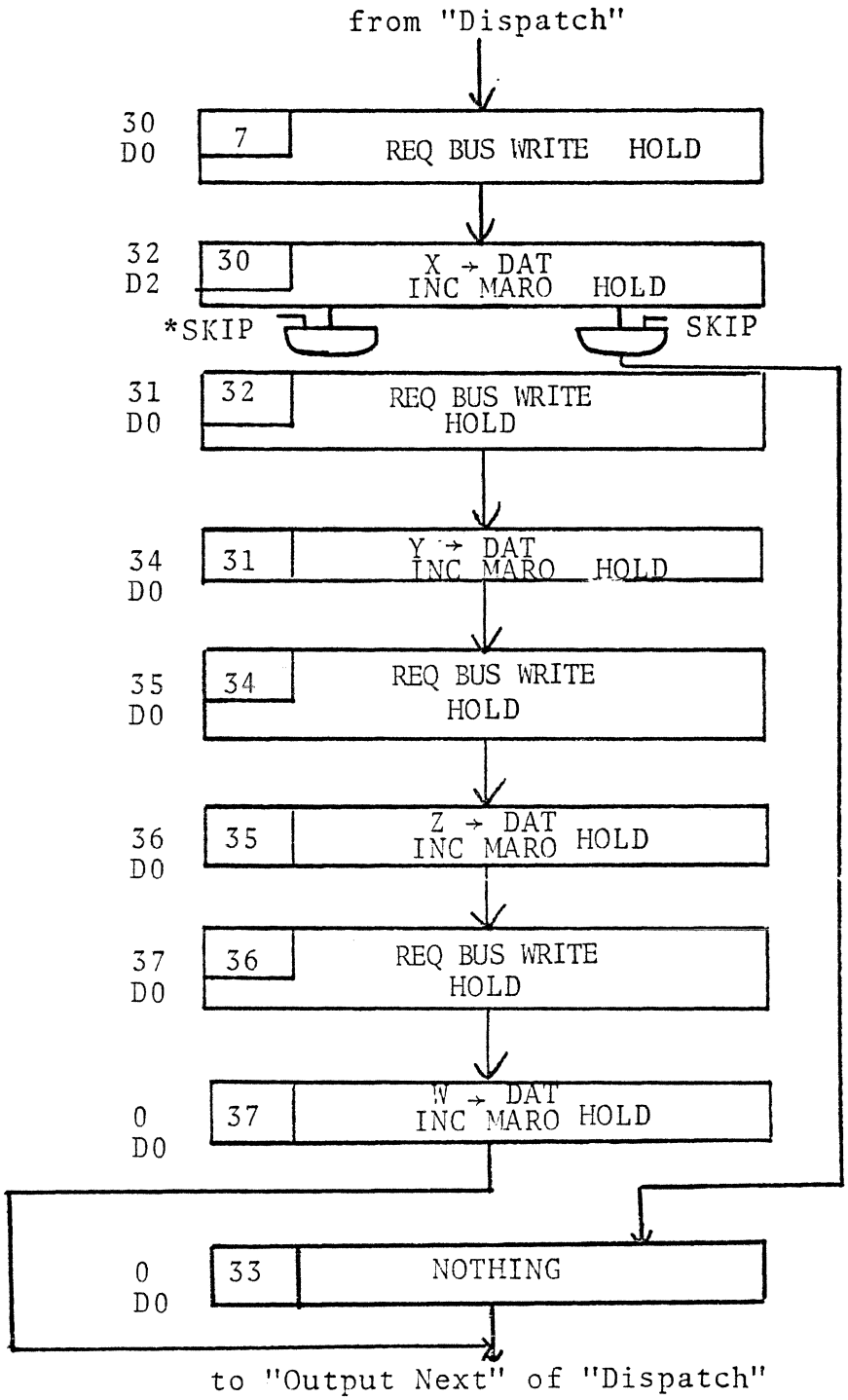
D0 NEXT ADDRESS
D1 LSB = MAPDONE
D2 LSB = SKIP
D3 A1,A0 = DISPATCH CODE

Output Sequencer Dispatch

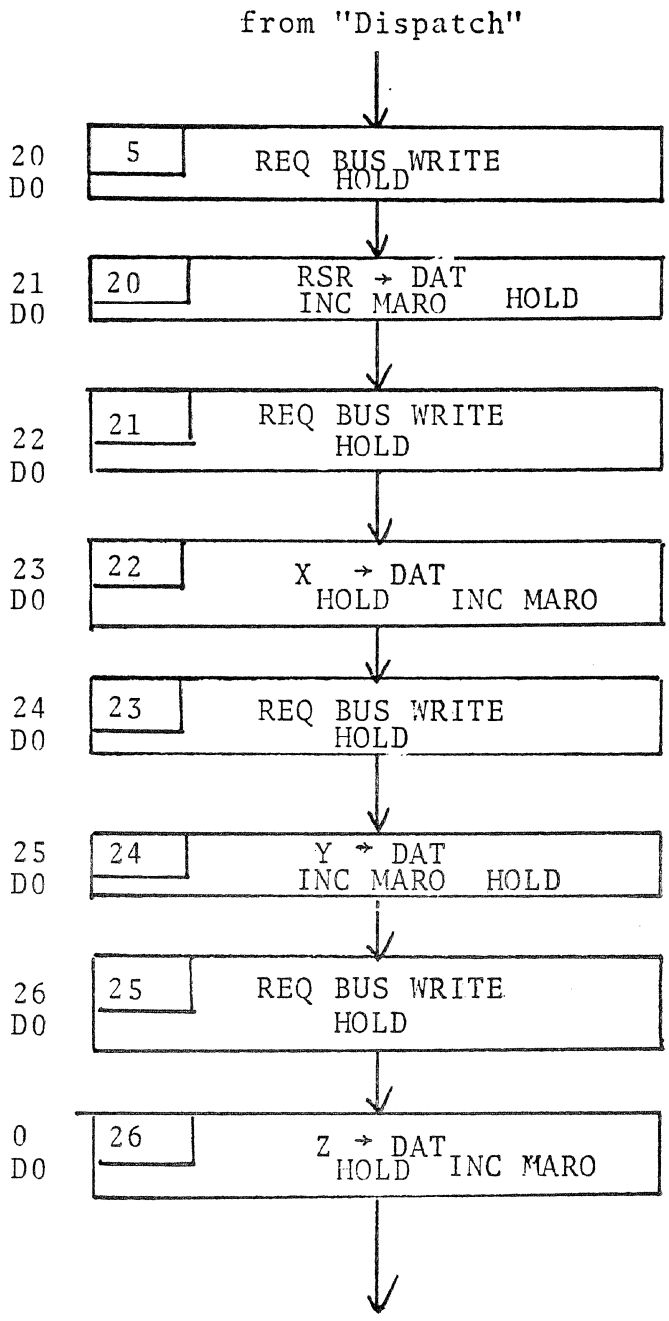

from "Dispatch"
10
D2
6
NOTHING
SKIP
*SKIP
00
D0
11
NOTHING
to "Output Next" of "Dispatch"
12
D0
10
REQ BUS WRITE
HOLD
13
D0
12
X → DAT
INC MARO HOLD
14
D0
13
REQ BUS WRITE
HOLD
0
D0
14
Y-DAT
INC MARO HOLD
to "Output Next" of "Dispatch"


from "Dispatch"
15
D0
4
REQ BUS WRITE
HOLD
16
D0
15
ST,ZH,X → DAT
HOLD INC. MARO
17
D0
16
REQ BUS WRITE
HOLD
0
D0
17
ZL,Y → DAT
HOLD INC MARO
to "Output Next of Dispatch"

3D Format


from "Dispatch"
30
D0
7
REQ BUS WRITE HOLD
32
D2
30
X → DAT
INC MARO HOLD
*SKIP
SKIP
31
D0
32
REQ BUS WRITE
HOLD
34
D0
31
Y → DAT
INC MARO HOLD
35
D0
34
REQ BUS WRITE
HOLD
36
D0
35
Z → DAT
INC MARO HOLD
37
D0
36
REQ BUS WRITE
HOLD
0
D0
37
W → DAT
INC MARO HOLD
0
D0
33
NOTHING
to "Output Next" of "Dispatch"

4D Pass

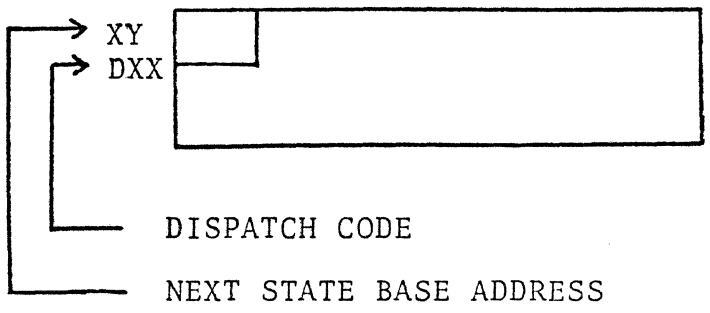

from "Dispatch"
20
D0
5
REQ BUS WRITE
HOLD
21
D0
20
RSR → DAT
INC MARO HOLD
22
D0
21
REQ BUS WRITE
HOLD
23
D0
22
X → DAT
HOLD INC MARO
24
D0
23
REQ BUS WRITE
HOLD
25
D0
24
Y → DAT
INC MARO HOLD
26
D0
25
REQ BUS WRITE
HOLD
0
D0
26
Z → DAT
HOLD INC MARO
to "Output Next of Dispatch"

4D High

APPENDIX C

REFRESH SEQUENCER STATE DIAGRAM

THIS STATE NUMBER

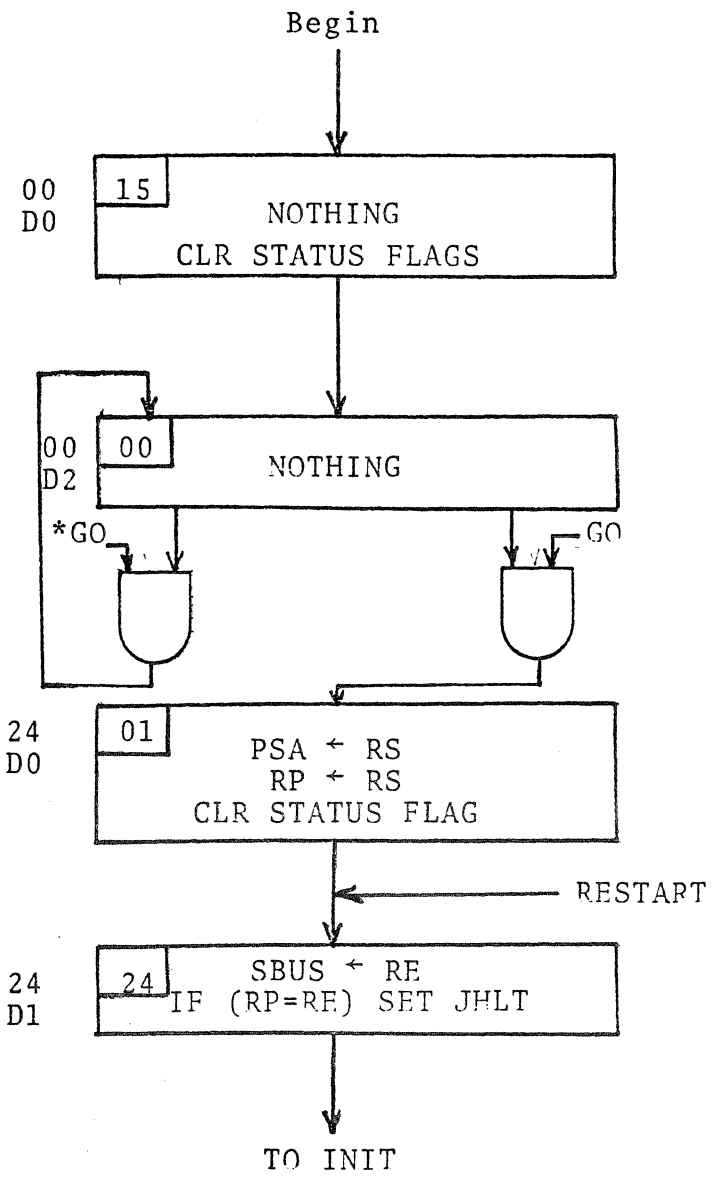


NEXT STATE ← (NEXT STATE BASE) + DISPATCH

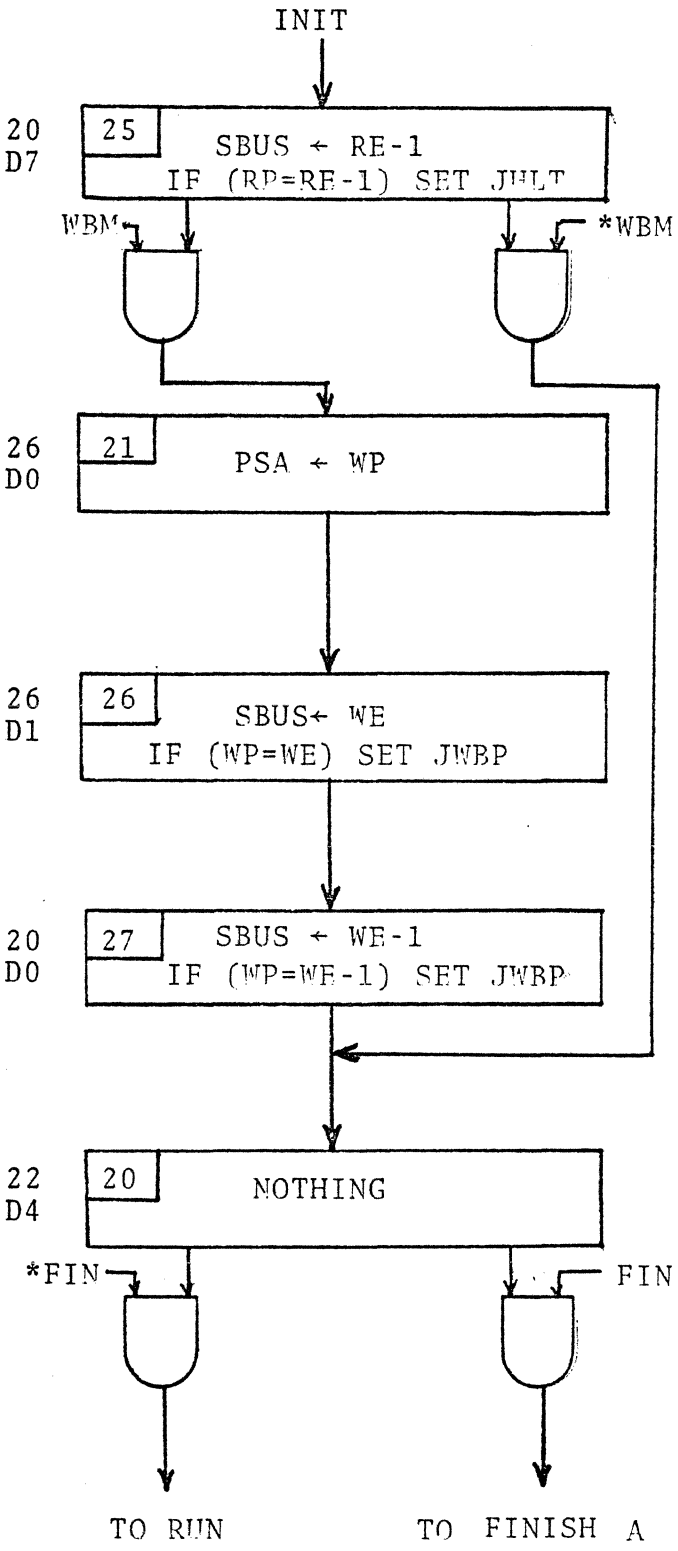
DISPATCH CODES

D0 = ZERO
D1 = ONE
D2 = GO
D3 = NAME
D4 = FINISH
D5 = FIFO BUSY
D6 = HLTREQ
D7 = WRT BACK

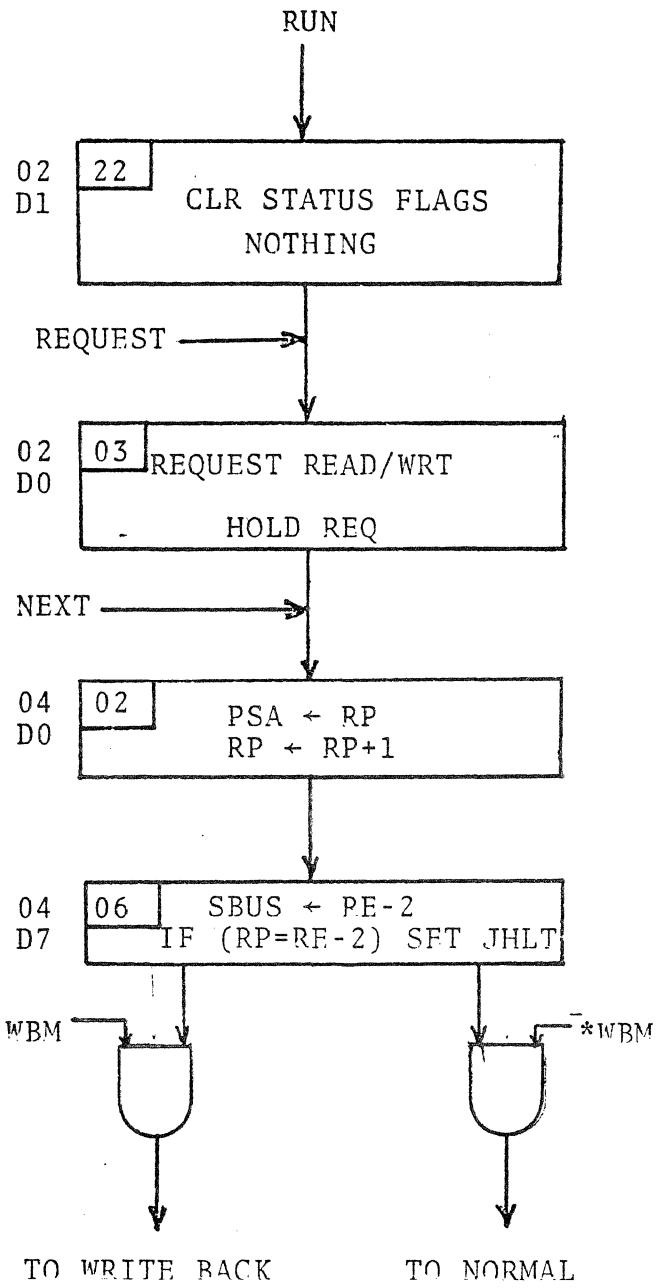
REFRESH CONTROLLER STATE MACHINE


Begin
00
D0
15
NOTHING
CLR STATUS FLAGS
00
D2
00
NOTHING
*GO
GO
24
D0
01
PSA ← RS
RP ← RS
CLR STATUS FLAG
RESTART
24
D1
24
SBUS ← RE
IF (RP=RE) SET JHLT
TO INIT

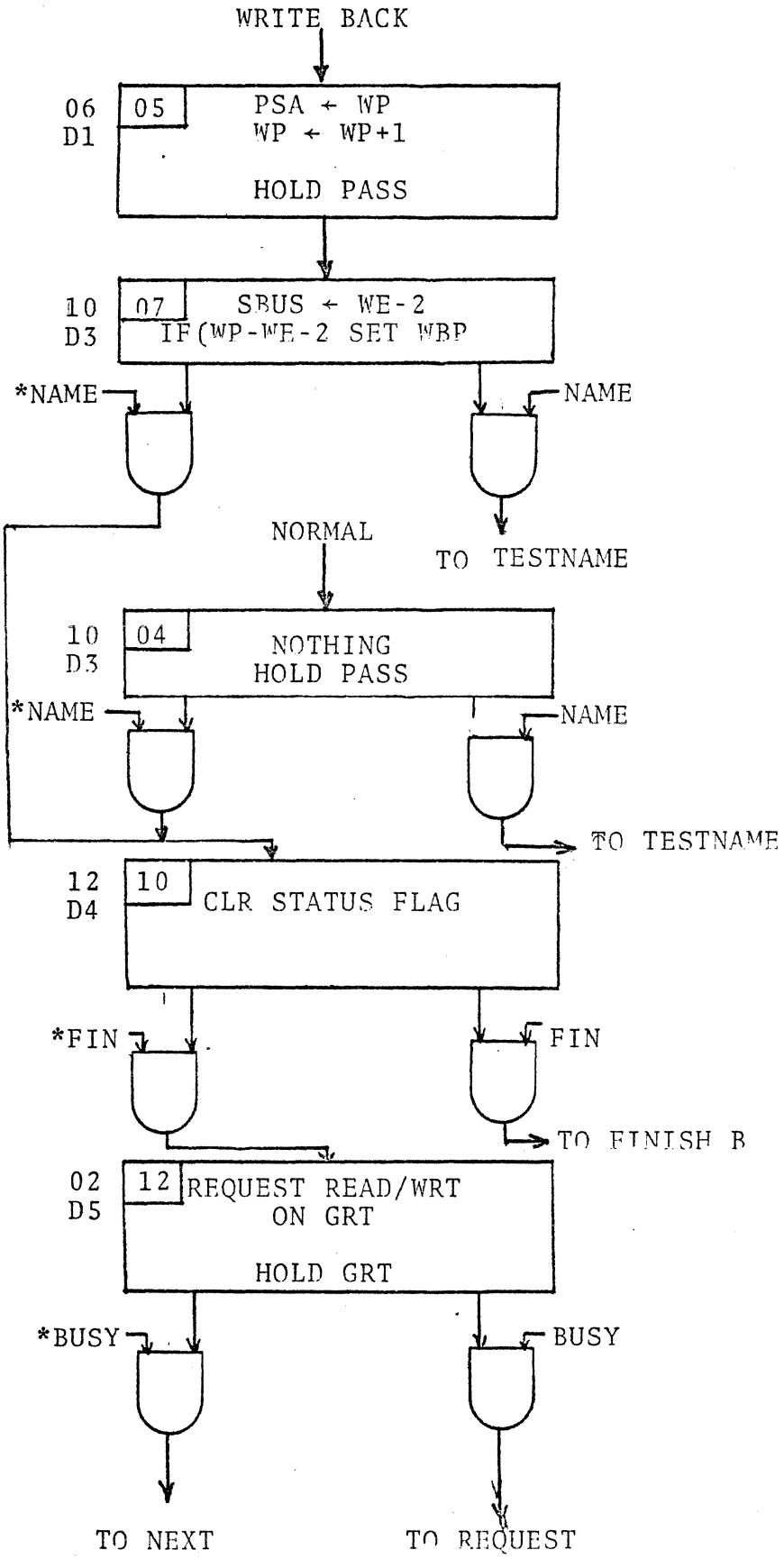
BEGIN



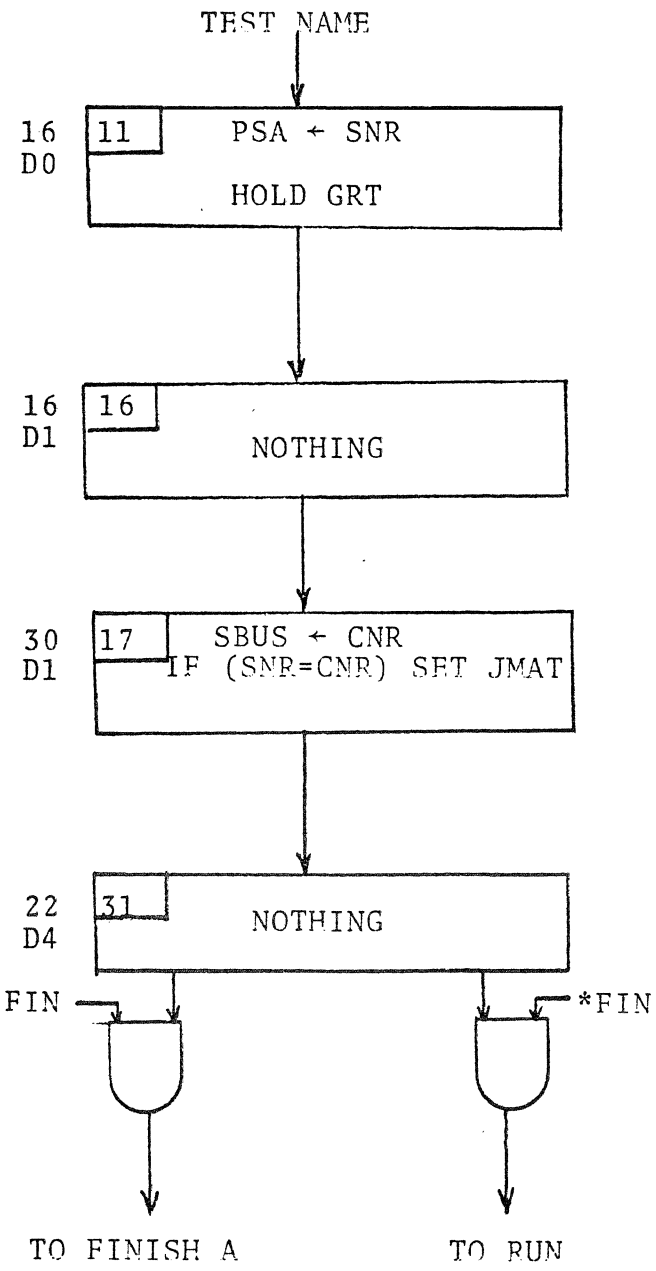
INITIALIZATION



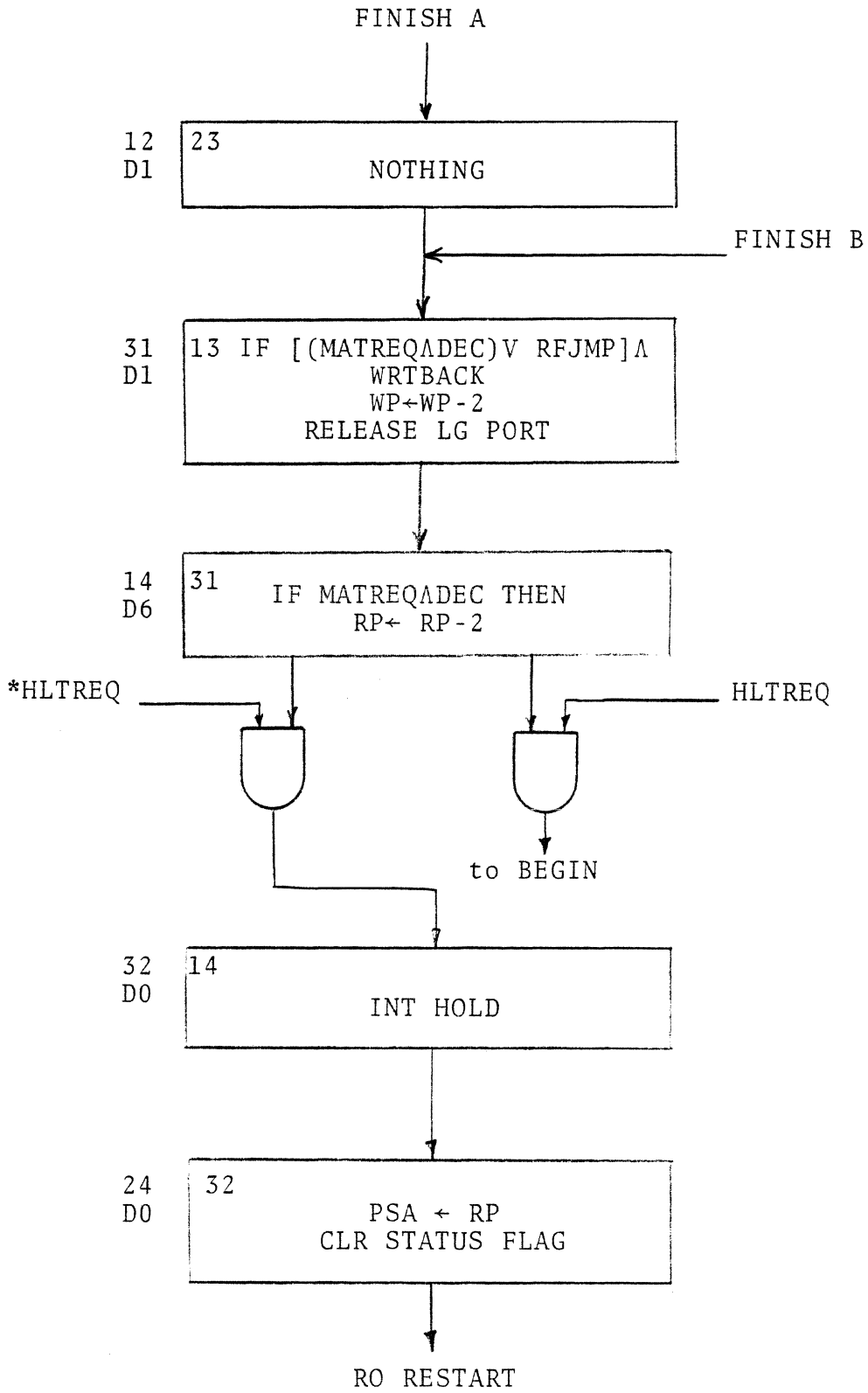
REQUEST CYCLE



COMPLETE CYCLE


TEST NAME
16
D0
11
PSA ← SNR
HOLD GRT
16
D1
16
NOTHING
30
D1
17
SBUS ← CNR
IF (SNR=CNR) SET JMAT
22
D4
31
NOTHING
FIN
*FIN
TO FINISH A
TO RUN

TEST NAME


FINISH A
12
D1
23
NOTHING
FINISH B
31
D1
13 IF [(MATREQ∧DEC)V RFJMP]∧
WRTBACK
WP←WP-2
RELEASE LG PORT
14
D6
31
IF MATREQ∧DEC THEN
RP← RP-2
*HLTREQ
HLTREQ
to BEGIN
32
D0
14
INT HOLD
24
D0
32
PSA ← RP
CLR STATUS FLAG
RO RESTART

FINISH