Part Number 800-1190-01
Revision: 50 of 1 July 1985

Hardware Reference Manual

for the

Sun Graphics Processor

Sun Microsystems, Inc.,
2550 Garcia Avenue,
Mountain View,
California 94043
(415) 960-1300

Credits and Trademarks

Multibus is a trademark of Intel Corporation.

Sun Microsystems and **Sun Workstation** are registered trademarks of Sun Microsystems, Incorporated. **Sun-2, Sun-2/xxx, Deskside, SunStation, SunCore, SunWindows, and DVMA** are trademarks of Sun Microsystems, Incorporated.

UNIX is a trademark of AT&T Bell Laboratories.

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

Contents

Contents

Tables

Figures

Preface

Welcome to the Sun-2 Graphics Processor. This manual presents a description of the Graphics Processor hardware *from a programmer's point of view:* that is, sufficient for a programmer to be able to understand the workings of the board.

Summary of Contents

This manual has seven chapters and an appendix:

Chapter 1 — *Introduction* — contains a basic overview of the the Graphics Processor, its position in the Sun-2 architecture, and associated Graphics Buffer board.

Chapter 2 — *System Configuration* — presents a simplified block diagram of the Graphics Processor and Graphics Buffer boards.

Chapter 3 — *Functional Description* — gives a functional description of the Viewing Processor, Painting Processor, and Graphics Buffer board.

Chapter 4 — *Detailed Description* — gives a fairly detailed description of the circuitry introduced in Chapter 3.

Chapter 5 — *VME Interface* — describes the interface between the Graphics Processor and the rest of the Sun-2 Model 160.

Chapter 6 — *Internal Registers* — describes the format of the registers associated with the Viewing Processor and the Painting Processor.

Chapter 7 — *Microcode Format* — describes the microcode instruction format for both the Viewing Processor and the Painting Processor. It also describes hardware implementations, and limitations of which the microcoder must be aware.

Appendix A — *Specifications* — gives VME and Performance specifications for the GP and GB boards.

Finally, to help us maintain the currency and accuracy of this material we have supplied a reader comment sheet at the end of this guide. Please use the comment sheet to list errors and omissions. Your responses will help a great deal in our efforts to keep our documentation up to date.

Glossary

A few terms are used throughout this document which, without explanation, may seem confusing.

- Positive Logic — positive logic means that the asserted level (see below) of a signal is a logic 1 (see below also).
- Asserted — when we say that a signal is "asserted," we mean that it is in its active, or true, state. In positive logic this means that a signal like READ, when asserted, is equal to its most positive state. When a signal like WRITE*, WRITE-, or WRITE\ (the three are synonymous) is asserted it is equal to its most negative state.
- Logic 1 — in positive logic, a logic 1 stands for the more positive of the two voltage levels. A logic 1 in negative logic stands for the more negative of the two voltage levels.
- Logic 0 — in positive logic, a logic 0 stands for the more negative of the two voltage levels. A logic 0 in negative logic stands for the more positive of the two voltage levels.
- Set — means the same as logical 1.
- Clear — means the same as a logical 0.

Applicable Documents

We emphasize that this manual outlines rather than exhausts many of the topics contained within. References to applicable documents supplied with your system are given throughout; however, and we urge you to read these documents should you need further information.

Table 1 *Sun Documentation*

Sun Part Number	*Description*
800-1191	Graphics Processor Engineering Manual

Table 2 *Vendor Documentation*

Description
AMD Bipolar Microprocessor Logic and Interface Data Book
AMD Bipolar/MOS Memory Data Book
Fairchild Advanced Schottky TTL (FAST) Data Book
Texas Instruments ALS/AS Logic Circuits Data Book
Programmable Array Logic (PAL) Data Book
Weitek 1032/1033 floating-point processor data sheet
4501 FIFO data sheet
VMEbus Specification

1

Introduction

Introduction

1.1. Overview

This manual describes the Graphics Processor (GP) board and the Graphics Buffer (GB) boards. The Graphics Processor is an attached processor to the host processor and can be used to perform many image display tasks. Since the Graphics Processor is faster than and runs in parallel with the host processor, there is a significant increase in system performance.

Floating point performance is a significant limiting factor to graphics performance. Since floating point performance is not suitable for interactive graphics, the intent of the GP is to provide a unit—separate from but controllable by the host processor—which has the necessary performance. It is a microprogrammable unit which, when invoked by the host processor, assists in the execution of a pre-defined task such as transforming, clipping, scaling, and rendering a two- or three-dimensional object.

2

System Configuration

System Configuration

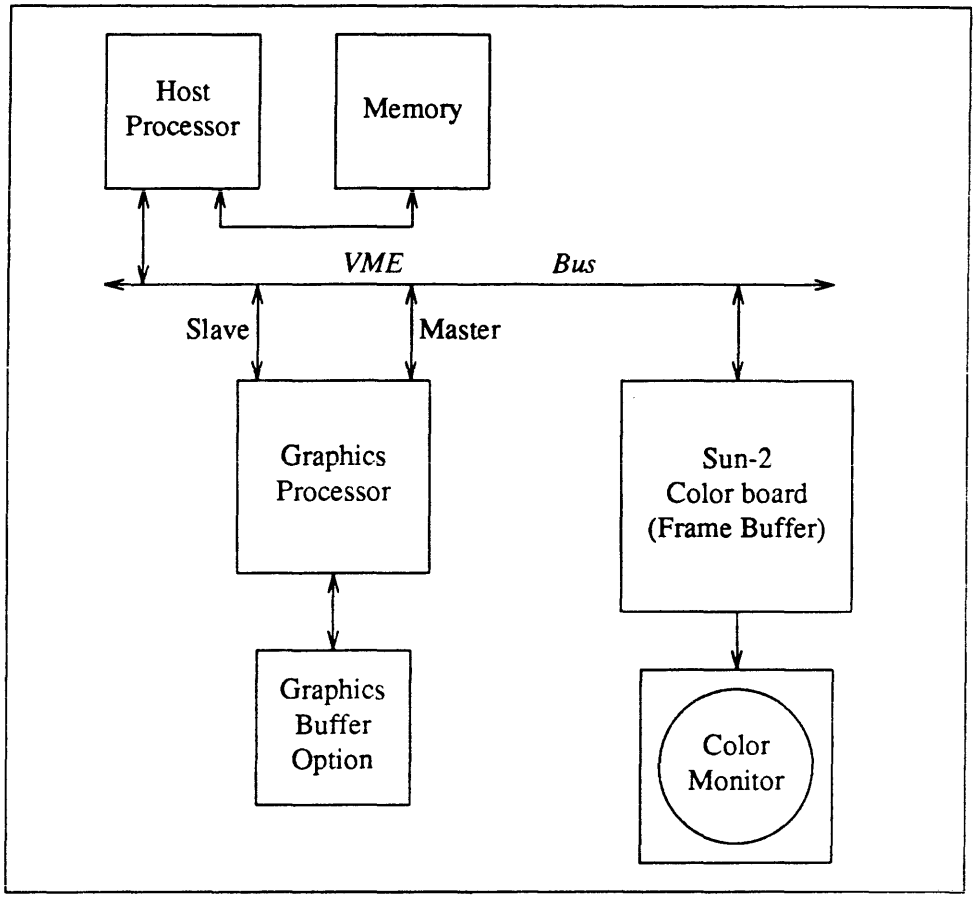
2.1. Overview

The following figure shows a simplified block diagram of a Sun color workstation with a Graphics Processor attached. The GP is either a one or two board set:

- the basic GP board, and
- an optional Graphics Buffer board.

Each board uses the triple high, quad depth (366.67mm by 400mm) Eurocard form factor.

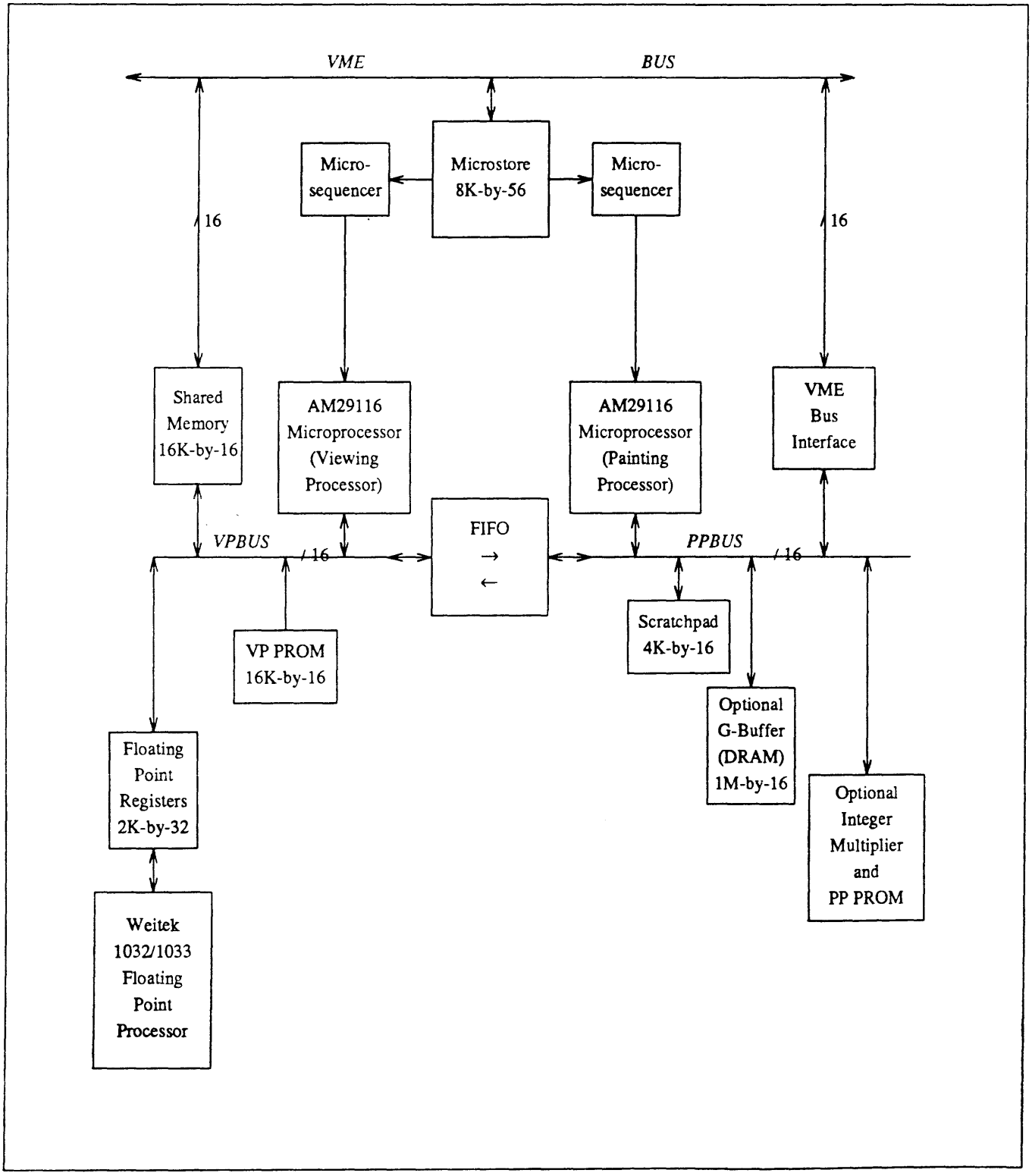
Figure 2-1 *Sun-2/160 Color Workstation: System-Level View*



The VME bus is used as both the system bus and the graphics bus. The host processor passes commands and parameters to the GP which processes them and writes pixels into the Color board. Even though the GP is logically between the host processor and the Color board, the GP and Color board can be installed in any VME slots. The GP and GB (Graphics Buffer) boards, however, must be installed in adjacent VME slots with a private bus between them.

A more detailed block diagram of the GP board is shown in the next figure. As can be seen, the GP contains three interfaces to the VME bus. The shared memory and the microstore interface are bus slaves, used primarily to load commands/parameters and to load microcode, respectively. The third interface ("VME Bus Interface") provides a general-purpose bus master capability, used primarily to access the Color board.

Figure 2-2 *Block Diagram of the Graphics Processor*



The shared memory is a dual-ported, high-speed, static RAM (random access memory), accessible from the VME bus by the processors on the GP board. (These are described in more detail in the next chapter.) Using the VME bus as the system bus, the host processor can load commands and data into the shared memory and receive requested status and data from the same memory.

The VME bus can also be viewed as the graphics bus, since the GP uses this bus to access pixels in the Color board's frame buffer. (The GP can access any VME location in standard or I/O (input/output) address space.) Providing a general-purpose VME connection on the Color board rather than a dedicated GP-to-Color board connection allows you to have a color workstation without a GP. It also allows direct access to the Color board by the host processor or other devices even with the GP installed.

3

Functional Description

Functional Description

3.1. Overview

Referring again to the block diagram of the Graphics Processor (shown previously), the GP consists of three processors:

- two Advanced Micro Devices (AMD) 29116 microprocessors, and
- a Weitek 32-bit floating point processor chip set, the 1032/1033.

The AM29116 is a 10MHz, 16-bit Arithmetic and Logic Unit (ALU) which evolved from bit-slice technology. The Weitek chips consist of an ALU (add, subtract, data format conversions) and multiplier, each capable of 1.1 Mflops in flowthrough mode and 5 Mflops in pipeline mode. Further details on these processors are available in the vendors' documentation.

The programs for the GP are contained within the microstore. This memory is three-ported, readable by each AM29116 section and readable/writable via the VME microstore interface, and is configured as 8K of 56-bit words.

As shown in the block diagram, the GP consists of two sections: The Viewing Processor (VP) and the Painting Processor (PP). These two processors form a pipeline for the execution of graphic commands. A brief discussion of each section's components follows.

3.2. Viewing Processor

The first AM29116, with the attached floating point processor and registers, is called the Viewing Processor. Its function is to receive commands and parameters from the host processor and perform the floating point operations needed to transform the image from world coordinates into screen coordinates.

Shared memory was mentioned in the previous chapter. It is implemented with sixteen 16K-by-1 static RAM chips providing a 16K-by-16 memory. This memory is time-multiplexed allowing independent access from both the VME bus and the Viewing Processor (VPBUS).

Graphic operations are floating point intensive; hence the floating point processor and associated registers. The floating point processor can be configured in a pipeline mode which allows a floating point operation to be initiated every two cycles. This unit is used to transform, clip, and scale the graphic data from world coordinates to screen coordinates. Maximum floating point performance is 4.16 Mflops.

The VP PROM (programmable read only memory) is used for the storage of reciprocal estimates and other constants needed for numerical computations. For

example when calculating a reciprocal, the PROM is used as a look-up table containing the first estimate for the iterative reciprocal algorithm.

The FIFO (first in first out) buffer is used to transfer commands and data from the Viewing Processor to the Painting Processor. Under control of the Viewing Processor, the FIFO can be reversed allowing the Painting Processor to send data to the Viewing Processor. The FIFO size is 512 16-bit words. It is the FIFO which logically and physically separates the GP into the Viewing Processor and the Painting Processor.

3.3. Painting Processor

The second AM29116 section is termed the Painting Processor. Its function is to render the graphic data (pixels) into the frame buffer on the Color board. The components discussed below are shown in the block diagram of the Graphics Processor board.

The scratchpad memory is a fast access, static RAM. Sequential accesses to this memory can be done in single cycles. The memory size is 4K-by-16. The scratchpad is a general-purpose memory useful for various algorithms.

The VME interface logic provides the capability to access VME devices from the Painting Processor, primarily the frame buffer on the Color board. However, the logic is general-purpose and any VME location can be accessed, including host memory and the GP's shared memory. In addition, this logic allows the GP to generate an interrupt to the host processor. This interrupt is under direct microcode control allowing its use to be defined by the particular application.

3.4. Graphics Buffer Board

The optional GB board contains a large memory implemented with dynamic RAM (DRAM) chips. The size of the memory is 1M 16-bit words (1M = 1,048,576). Dynamic RAMs provide large amounts of memory at slower access times. Random accesses to the DRAM take five cycles (two cycles to load the 21-bit address and three cycles to do the read or write) but sequential reads or sequential writes can be done in three cycles per read or write.

- A *fill* mode allows a data word to be written into four consecutive locations in the time required to do one write.
- A *read/modify/write* mode allows a write followed by a read of the next consecutive address to be done in five cycles.

This memory is designed to be general-purpose (font storage, anti-aliasing use, etc.) but is especially suitable for hidden surface elimination algorithms.

Also on the optional GB board are an integer multiplier and associated PROM for numerical constant storage. An AMD 29L517 chip is the multiplier chip used; a multiply of two 16-bit operands producing a 32-bit result takes six cycles (including data transfer cycles). This multiply capability greatly speeds up the performance of advanced shading algorithms.

3.5. Summary

The pipeline architecture of the GP is well suited for graphics applications. Graphic data are manipulated serially and independently; therefore, partitioning the tasks for pipelining is straightforward. The intent is to divide the task so that all three stages of the pipeline—the host processor, the Viewing Processor, and the Painting Processor—are active as much as possible. The parallelism thus achieved will permit high performance graphics.

4

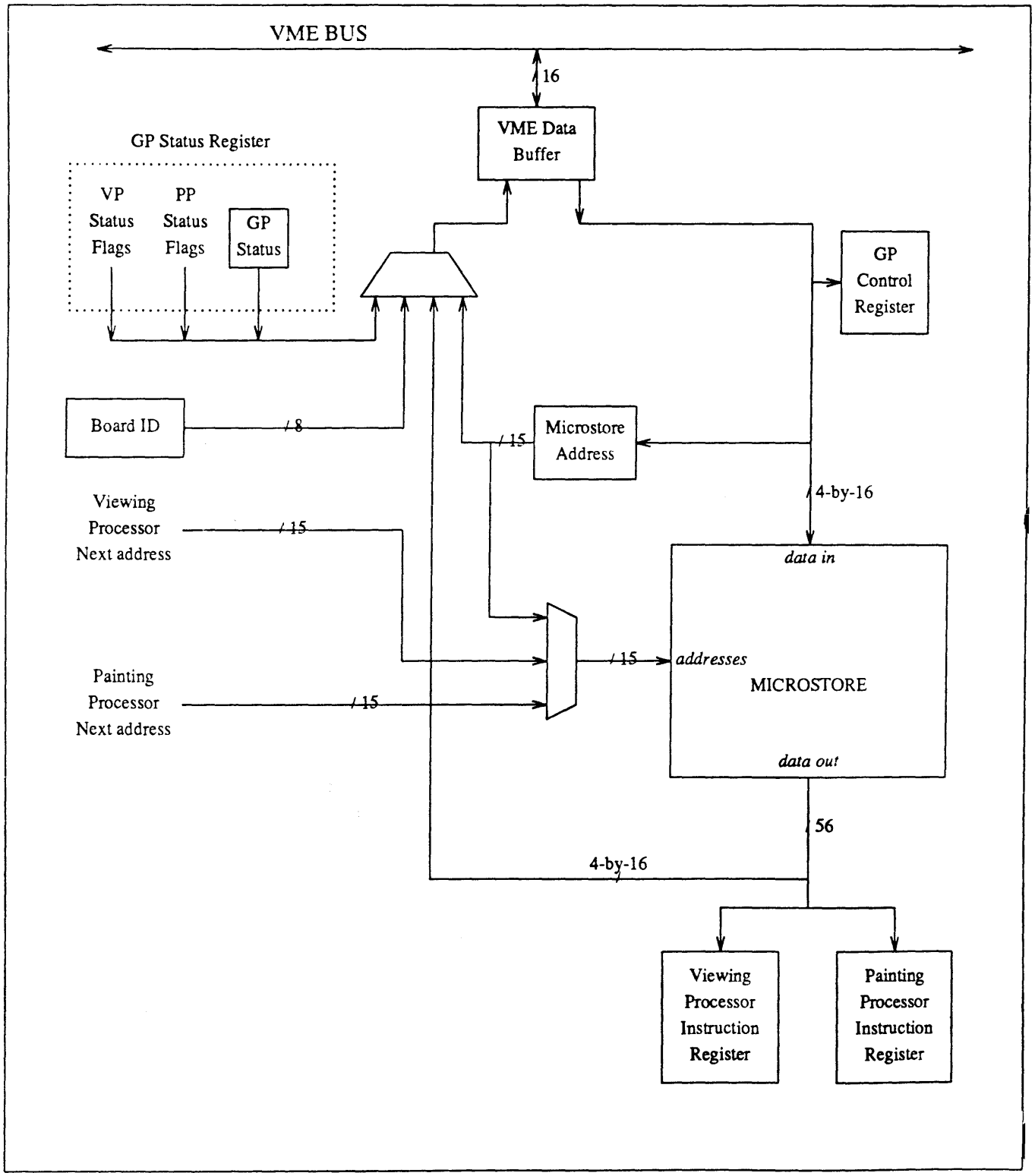
Detailed Description

Detailed Description

4.1. Overview

The Graphics Processor is actually two separate units, the Viewing Processor and the Painting Processor, with both processors sharing the microstore. The figure below details the common microstore. Subsequent figures contain detailed block diagrams of the two processors. The discussion below describes these block diagrams and the operational details of the processors.

Figure 4-1 *The Microstore*



Both processors can be viewed as programmable state machines running with a 120 nanosecond cycle time. During each cycle several parallel operations are possible. On the Viewing Processor for example,

- an AM29116 instruction can be executed;
- a floating point operation performed;
- a branch executed;
- and a data word moved from a VPBUS source to a VPBUS destination.

Similarly, on the Painting Processor,

- an AM29116 instruction can be executed;
- a VME operation initiated;
- a branch executed; and
- a data word moved from a PPBUS source to a PPBUS destination.

The current operation of each processor is controlled by the current contents of each processor's instruction register, containing a microinstruction. The microinstruction format has been chosen to provide as much parallelism as possible within physical constraints (board area, power, etc.)

4.2. Microstore

The microstore contains the microcode for both GP processors. The two units run 180 degrees out-of-phase and require half of the memory bandwidth; thus the microstore can be shared. A third port into memory is available for VME accesses into the microstore (initial program load and verification) but can be active only when the two processors are halted.

The microstore is built with fast, static RAMs. Twenty-eight physical RAM locations are provided and 8K of microstore (using 4K-by-4 chips) is available.

Contained in the block diagram of the microstore is:

- the microstore,
- each processor's next address source and instruction register,
- the microstore registers accessible from the VME.

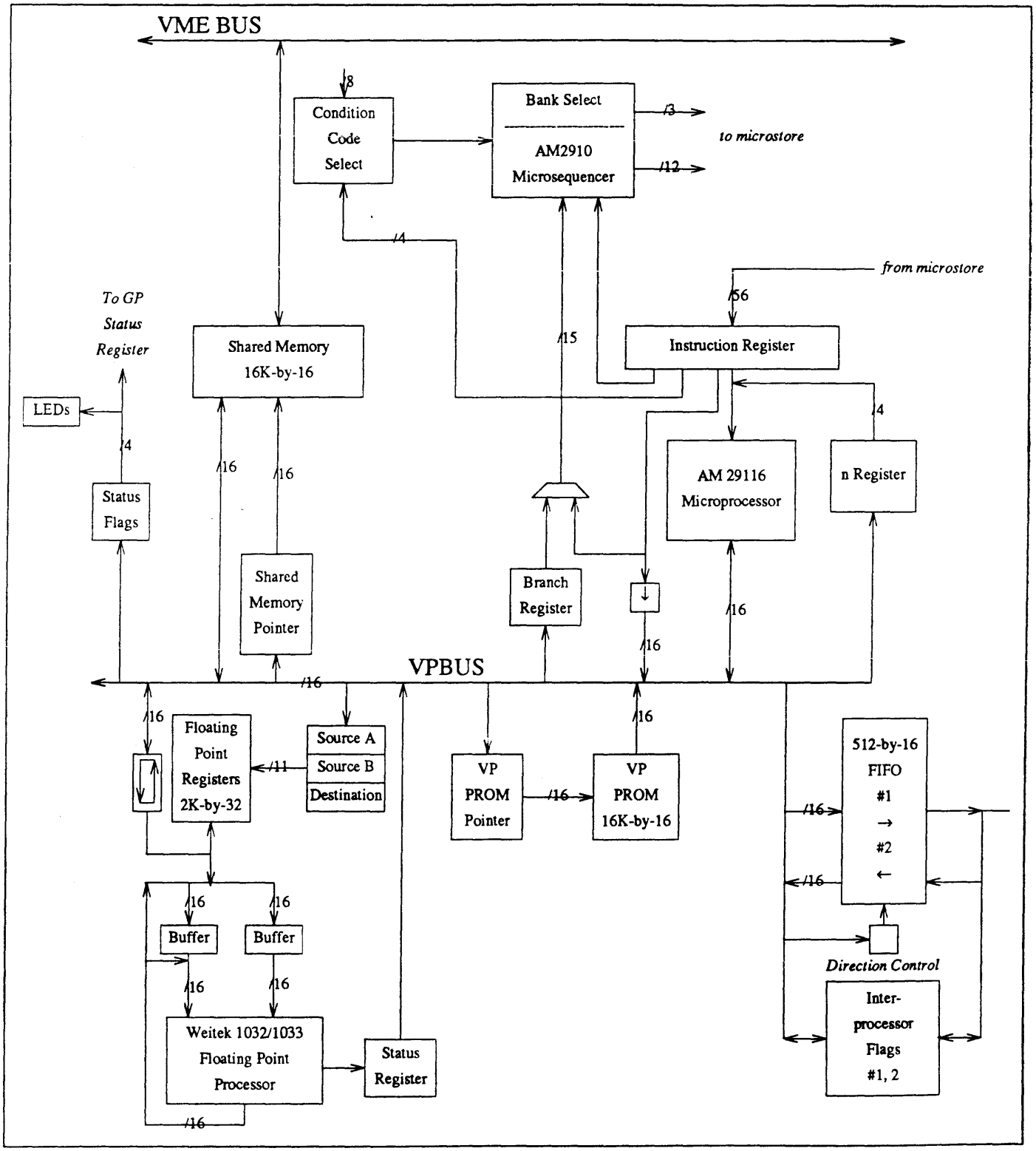
The microstore registers reside in the first 32 Kbytes of VME address space allocated to the GP. (This allocation is determined by a hardware switch on the GP.) Shared memory resides in the next 32 Kbytes.

For each processor, the first half of each cycle is used to determine the address of the next microinstruction. In the second half-cycle, this next address is routed to the microstore, an access is made, and the instruction register is loaded at the beginning of the next cycle defining the new "current" instruction. The sequence then repeats. Since the two processors are running 180 degrees out-of-phase, the first half of a cycle on one processor is the second half for the other processor.

4.3. Viewing Processor

The following figure is a block diagram of the Viewing Processor. The components of the block diagram of the GP Board are recognizable but more detail, especially around the AM29116 and the Weitek floating point units, is shown. Various references are made to the microinstruction in the discussions that follow. The format of this 56-bit word is defined in the chapter describing the microcode format.

Figure 4-2 *Block Diagram of the Viewing Processor*



Program Sequencer and Program Memory

To make the AM29116 into a useful computer, two major components are required: a *program sequencer* and *program memory*. The microstore is the program memory. The sequencer used is an AM2910 which contains a program counter, a stack for subroutine linkage, and branch control logic. A branch is conditional on the state of the condition code input to the AM2910. The condition code select logic multiplexes 16 options (either polarity of eight status flags) into this one input. A limitation of the AM2910 is its 4K address space. The bank select logic is used to expand this space to 32K maximum by selecting one of eight banks. The AM2910 and the bank select provide the address to the microstore.

A bank switch can only be done by executing the AM2910 JMAP instruction. This is an unconditional jump for the AM2910 and is used to flag the bank select logic that a (potential) bank switch is to be done. During the JMAP instruction, the bank select state is updated. When doing a JMAP instruction, the D input to the AM2910 can be either the branch register or the general field of the microinstruction (see below).

A JZ (jump zero) command to the AM2910 forces the microprogram counter to 0. A JZ instruction jumps to location 0 of bank 0. This command thus effects the bank select bits, potentially executing a bank switch.

NOTE *The AM2910 contains an internal R register and stack. These registers are 12-bit only so that branches to these addresses DO NOT perform bank switches. Care must be taken to ensure that, for example, a call is not done in one bank and the corresponding return done in another. Also, the sequencer does not automatically flow from one bank to the next. That is, a continue from location FFFF (hex) in bank 0 (the last location in this bank) will go to location 0000 in bank 0, not location 0000 in bank 1.*

Branch Register

The *branch register* is a 15-bit register that contains a possible source of the next address selected by that AM2910. A branch to the location preloaded into the branch register is executed if two conditions are met. The branch register must be chosen as the D input of the AM2910 (controlled by the DS microinstruction bit), and the branch condition must be successful (determined by the AM2910 instruction and the state of the branch status flag chosen in the microinstruction). The branch register is loaded when it is chosen as the VPBUS destination (controlled by the source/destination field of the microinstruction).

Instruction Register Buffer

Shown in the block diagram of the Viewing Processor are *buffers enabling the general field of the instruction register* onto the VPBUS. These buffers are turned on when the general field is chosen as the VPBUS source (controlled by the microinstruction source/destination field). With this capability, the microcoder can route an assembly-time constant onto the VPBUS and route it to one of several possible destinations, including for example, the AM29116, the shared-memory pointer, or a floating point register pointer.

n Register

Several AM29116 instructions include a 4-bit field (n) to determine, for example, the bit position to test or the number of bits to rotate. The usefulness of these instructions is diminished if the n value is hardcoded into the microinstruction. Therefore, a four-bit *n register* is provided. Under control of the microinstruction, the n register value can be substituted for the n field in the AM29116 instruction, thereby providing a way to calculate n at runtime, not at assembly time. The n register is loaded by chosing it as the VPBUS destination.

Interprocessor Flags

An interprocessor flag mechanism is provided to pass status flags between the two processors. Two 8-bit registers are implemented, one in each direction; FIFO status can be read from them. *Interprocessor flag #1 register* can be written by the Viewing Processor and read by the Painting Processor. It is used by the Viewing Processor to control FIFO direction. *Interprocessor flag #2 register* can be written by the Painting Processor and read by the Viewing Processor. These registers are under total firmware control; they have no special hardware significance.

An ninth bit is also read when the interprocessor flag is chosen as the bus source. On the Viewing Processor this bit is read as a logic 0, and on the Painting Processor this bit is read as a logic 1. This provides a mechanism to distinguish the two processors at reset. After a reset, both processors begin executing at microstore location 0; both read their interprocessor flags; and then each processor branches to its respective initialization routine.

There is a hardware constraint to the interprocessor flag register—it is not possible to read this register into the AM29116 and manipulate it in a single cycle. It must first be read into the AM29116 D-latch and then manipulated by subsequent instructions.

Status Flags Register

The *status flags register* is a four-bit VPBUS destination that provides a mechanism to return general-purpose status bits to the host processor via the VME bus. These bits can be written (chosen as a VPBUS destination) at any time by the Viewing Processor and read at any time from the VME bus. Four LEDs (light emitting diodes) are also driven by this register. This register also contains the fpsel bit, which allows read-back of either set of floating point registers. For further information, see the chapter which describes the internal registers.

FIFO

The *FIFO* is the connection to the other processor. One "reversible" FIFO is implemented but for easier understanding the following nomenclature is used: FIFO #1 is for VP-to-PP transfers and FIFO #2 is for PP-to-VP transfers. The direction is controlled by two bits that are written whenever the interprocessor flags #1 register is written. The FIFO is 512 16-bit words deep.

If FIFO #1 is chosen as the destination, the data word on VPBUS is loaded into the FIFO. A testable status flag (a condition code select option) is used to determine if the FIFO is full or not. Hardware protection is provided to ensure that a data word is not written to a full FIFO, but the microcode must test the status to determine the success or failure of a FIFO load and thus determine if the FIFO load should be re-executed.

If FIFO #2 is chosen as the source, the data word on the top of FIFO #2 is routed onto VPBUS. A testable status flag is used to determine if a valid data word was routed onto the bus. Hardware protection is provided to ensure that an empty FIFO is not read (and thereby prevent a timing glitch which could cause the loss of a data word), but the microcode must check the status flag to determine whether or not another read of the FIFO is necessary to receive valid data.

Because of the asynchronous nature of the FIFO, there is a "recovery time" after each FIFO access and it is therefore not possible for a FIFO to accept or supply data every processor cycle. On FIFO writes, there is a 1 cycle recovery time, so that writes can be done at most every other cycle. On FIFO reads, the recovery time is 2 cycles, so that reads can be done at most every third cycle. However, the FIFO status flags will be valid at all times.

If the FIFO direction is VP-to-PP, then VP readings of the FIFO will always succeed, but the data returned are meaningless. If the FIFO direction is PP-to-VP, then VP writes to the FIFO are also successful, but the data are discarded. Read/write success is determined by testing the FIFO status flags. This is done to prevent the microcode from hanging by accessing the FIFO in the wrong direction.

Shared Memory

Shared memory has been mentioned several times. The dual-port capability is implemented by allocating two accesses to the memory every Viewing Processor cycle. The first half cycle is allocated for a read or write access by the Viewing Processor. The second half cycle is allocated for VME bus reads or writes. Obviously, a meaningful operation is not done every half cycle, but the allocation is fixed.

Before accessing shared memory, the Viewing Processor must load the shared-memory pointer. This register points to the location at which subsequent Viewing Processor accesses will be made. Under microcode control, this pointer can be incremented, decremented, or cleared.

The shared-memory access is executed by chosing the shared memory as the VPBUS source or destination. By also counting the pointer when making the access, sequential reads or writes can be done at the rate of one per cycle.

Because of the shared-memory architecture, writes to the shared memory (and increments or decrements of the shared-memory pointer when done coincident with the write) are done in the cycle immediately following the cycle with the write instruction. This means that a read of shared memory cannot be done in the cycle immediately following a cycle doing a shared memory write. An increment or decrement of the shared-memory pointer executed in the next cycle is thus redundant unless the shared memory is coincidentally selected as the VPBUS destination.

VP PROM and VP PROM POINTER

When selecting the VP PROM as the VPBUS source, the location pointed to by the *VP PROM pointer* is routed to the VPBUS. The functional VP PROM is implemented with two 16K-by-8 erasable PROMs providing for 16K of 16-bit words. After loading the VP PROM pointer with the address of the location to be read, a two cycle delay must be incurred before selecting the VP PROM as the VPBUS source to allow for the slow access time of the PROM.

4.4. Floating Point Circuitry

This section covers the floating point registers, the pointers into these registers, and the Weitek floating point chips. All floating point operations are done through the floating point registers. The Weitek chips receive their input from these registers and all floating point results are loaded back into these registers.

Overview of the Floating Point Circuitry

There are three pointers into the *floating point registers*:

- source A, and
- source B, and
- destination.

Each of these pointers is 11 bits wide (2K addressing capability) and can be loaded as a destination from the VPBUS. Under microcode control these pointers can be incremented at the end of a cycle.

The floating point registers are implemented with 4K-by-4 static RAMs, providing 4K of 16-bit words. But since floating point numbers are 32-bits wide, these registers are treated as 2K 32-bit registers, explaining the 11-bit width of the pointers. The high (most significant) or low (least significant) word of the 32-bit floating point number is selected by the h/l bit in the current microinstruction. If h/l = 0 then the most significant word is selected; if h/l = 1 then the least significant word is selected.

The source A pointer is used to select

1. the register that is routed to VPBUS when a floating point register has been chosen as the source of data onto VPBUS, or
2. the A operand to the Weitek chips.

The source B pointer is used to select the B operand to the Weitek chip.

The destination pointer is used to select

1. the floating point register into which the Weitek result is loaded, or
2. the location into which the VPBUS data word is loaded when a floating point register is chosen as the VPBUS destination.

For diagnostic purposes, it is possible to route the data pointed to by the source B pointer onto the VPBUS. A flag under microcode control (as part of the status/LED register) determines whether the source A or source B pointer is used to determine the data word read.

NOTE *A hardware implementation detail: the floating point registers are implemented as two banks in order to increase the bandwidth of these registers. The source A pointer points to bank A and the source B pointer points to bank B. The destination pointer points to both banks. Whenever a write is done, both banks are written; therefore, the banks are duplicates. Normally, when the floating point register is chosen as the VPBUS source, bank A is read. However for diagnostics, it is possible to read bank B directly by controlling the floating point flag described previously.*

When the floating point registers are chosen as the VPBUS destination, the data word is first loaded into a holding buffer. It is actually written into the registers

in the next cycle. When the floating point register is the VPBUS destination and the destination pointer count is enabled, the count, like the data load, is executed in the next cycle. An increment the destination pointer executed in the next cycle is thus redundant unless a floating point register is coincidentally selected as the VPBUS destination.

Floating Point Registers

Similar to the shared memory, the floating point registers operate at twice the frequency of the AM29116 cycle. The first half cycle is used to read the register file and supply data for either the Weitek chips (A and B operands) or VPBUS. The second half cycle is used to store VPBUS data or Weitek chip results into the registers if so instructed by the microinstruction.

A hardware restriction is that on consecutive cycles, the floating point registers cannot be used as the destination of a VPBUS operation and then as the destination of a floating point result. In addition, the source A pointer is used to select both the A operand to the Weitek chips and the data word routed to the VPBUS when selected as the source. It is unlikely that the source A pointer can be used to select the VPBUS source data and the Weitek chip source A in a single microinstruction.

Weitek Floating Point Chips

The *Weitek Floating Point* chips are controlled by fields within a microinstruction. The following three operations are possible:

1. (Source A pointer) op (Source B pointer),
2. Weitek result op (Source B pointer),
3. Weitek result --> (Destination pointer).

1 and 2 are mutually exclusive. 3 can be done in the same microinstruction as either 1 or 2.

The operations of the Weitek chips take several cycles to complete. The microcoder must be aware of this and not attempt to use a Weitek result before it is ready. In pipeline mode, it takes two cycles to load the 32-bit operands, 6 cycles of execution delay, 2 cycles of unload instructions, and 2 cycles to output the result. But because of the pipeline, a new operation can be started every two cycles.

To minimize hardware, floating point registers are treated as 32-bit numbers. The microcoder must take care to control the h/l (hi/low word) bit and cause the proper data word (most or least significant word) to be utilized at the proper time. This is of concern in the following operations:

- Floating point register as VPBUS source,
- Floating point register as VPBUS destination,
- Initiating a floating point operation to the Weitek chips,
- Enabling an unload of the Weitek chip result,
- Unloading the Weitek chip result.

The h/l bit directly controls the least significant bit of the address to the floating point registers and the U0 bit of the Weitek chips.

The Weitek chips provide status coincident with each result. This status information is made available on an individual operation and on an accrued basis as part of the floating point status register.

4.5. Painting Processor

The following figures are the block diagrams for the Painting Processor. As before, the functional blocks of the block diagram of the Graphics Processor board are expanded now in greater detail.

Figure 4-3 *Painting Processor Block Diagram — Part 1*

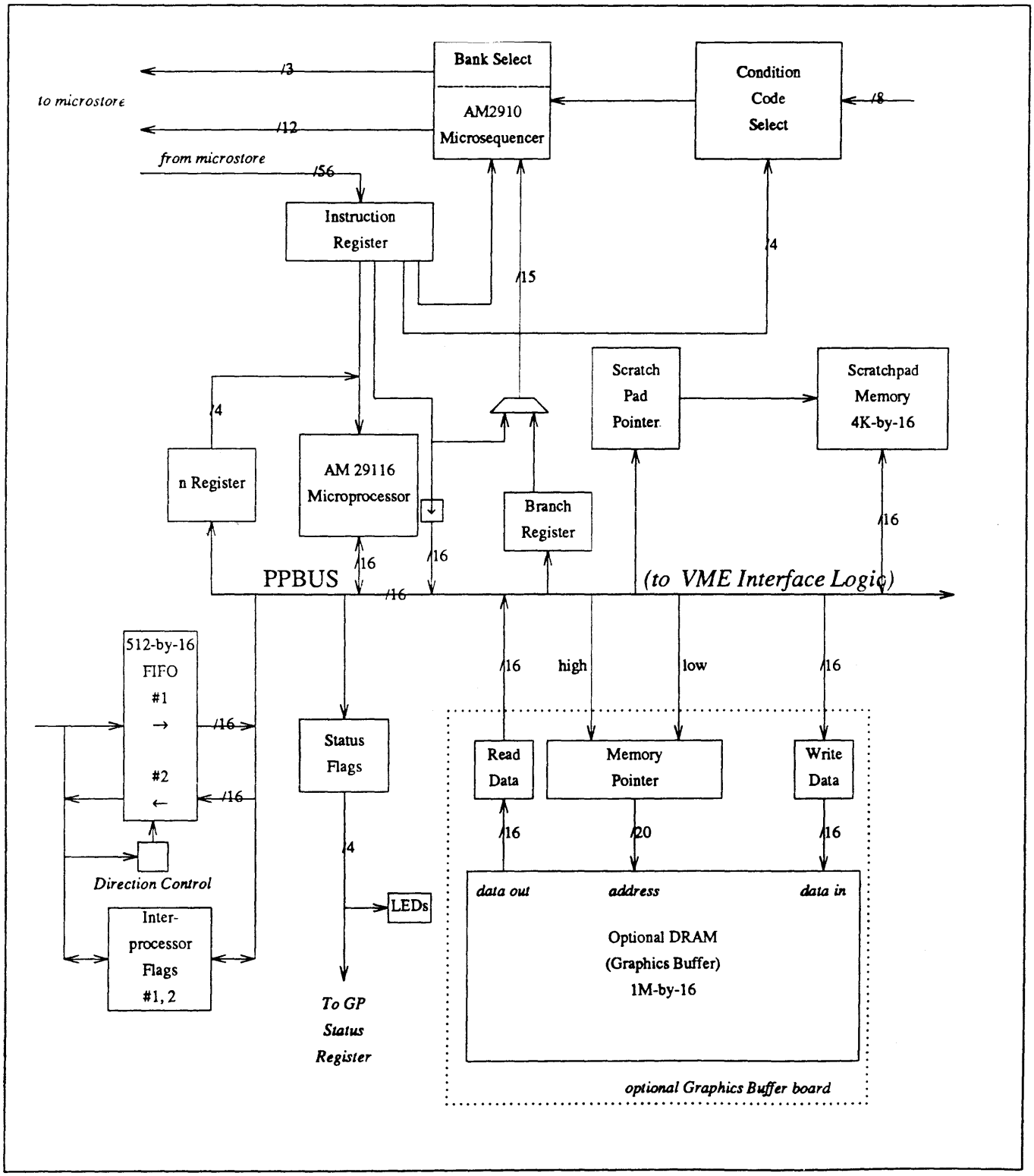
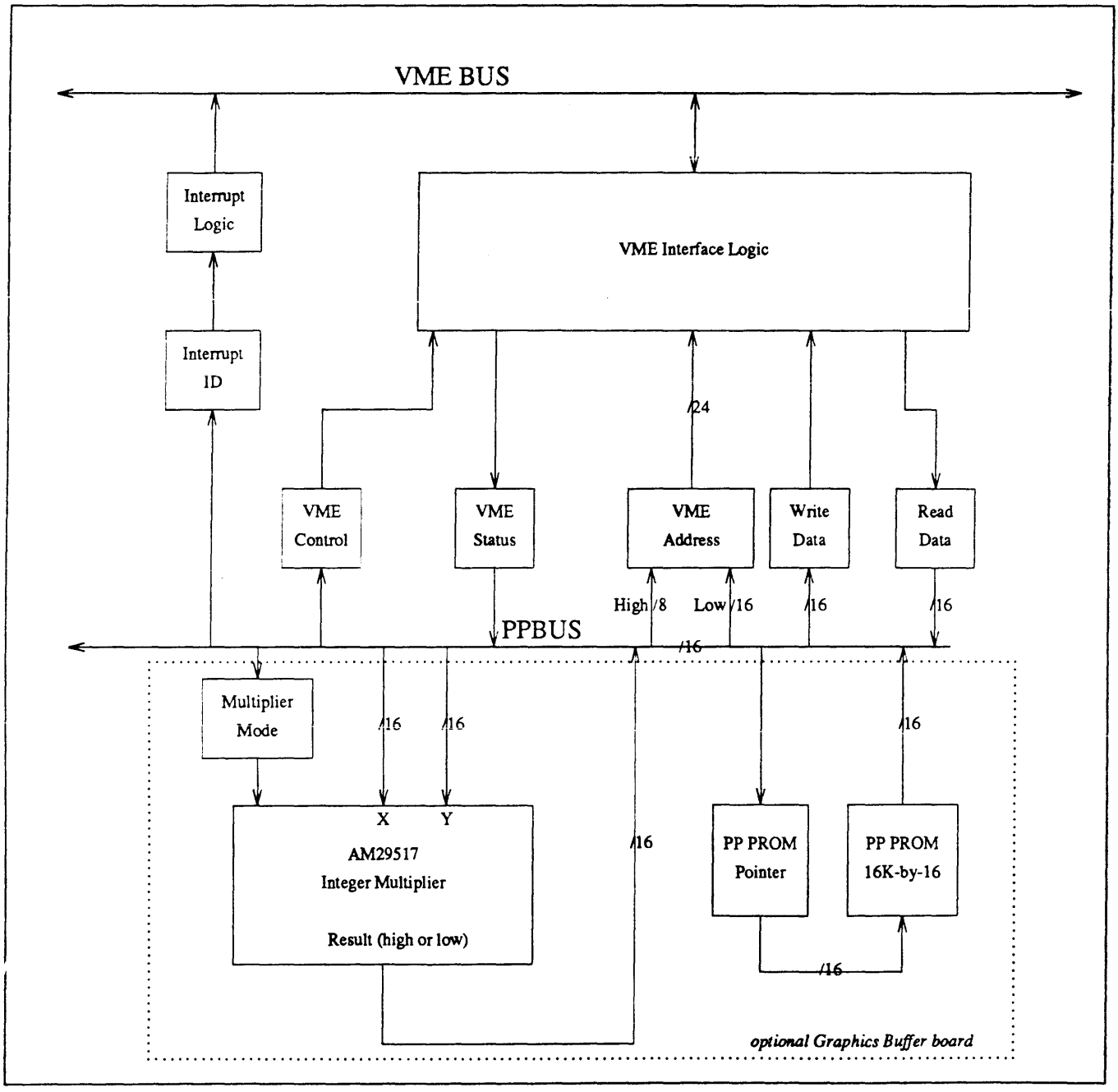


Figure 4-4 *Painting Processor Block Diagram — Part 2*



As the figure suggests, much of this section is the same as the Viewing Processor. These components include

- an AM2910 microsequencer,
- bank select logic,

- branch register,
- general field,
- n register,
- interprocessor flag registers,
- status flag register,
- reversible FIFO.

Branch Register

The *branch register* and the branch restrictions for this section are basically the same as for the Viewing Processor. The only difference is the method to perform on unconditional branch (or call or return). On the Viewing Processor a logic 1 is input to the condition select multiplexer and, if selected, causes a "pass" condition to the AM2910. On the Painting Processor the eight multiplexer inputs are otherwise occupied (see the chapter describing the microcode format) so that the AM2910 condition code enable signal is a microinstruction bit and is used to force a "pass" condition.

Condition Code Select

A difference also exists in the *condition code select.* When initiating a VME bus operation (described below), it is possible to do a 3-way branch. With the proper microinstruction, the following is possible:

1. If the VME busy flag is set, branch to the location specified in the general field.
2. If the VME busy flag is not set, then test another selectable condition (for example, negative) and either
   a) branch to the location specified in the branch register for a pass condition, or
   b) branch to the next instruction for a fail condition.

Scratchpad Memory and Scratchpad Pointer

The *scratchpad memory* is 4 Kwords by 16 bits per word of one-cycle access memory. If chosen as the source, the data word pointed to by the *scratchpad pointer* is loaded onto PPBUS. If chosen as the destination, the data word on the bus is loaded into the location selected by the scratchpad pointer. The pointer can be cleared or incremented under microcode control.

Because of the scratchpad-memory architecture, writes to the scratchpad memory (and increments or decrements to the scratchpad-memory pointer when done coincident with the write) are done in the cycle immediately following the cycle with the write instruction. This means that a read of scratchpad memory cannot be done in the cycle immediately following a cycle doing a scratchpad memory write. An increment the scratchpad pointer executed in the next cycle is redundant unless the scratchpad is coincidentally selected as the PPBUS destination.

Interrupt ID Register

The byte-wide *interrupt ID register* is used to trigger an interrupt on the VME bus. Loading a value into the register (selecting it as the PPBUS destination) causes the following:

1. the interrupt flag in the GP status register is set;
2. if the interrupt enable flag (in the GP control register) is set, a VME interrupt request is generated;
3. in response to the interrupt acknowledge, the value in the interrupt ID register is returned to the host processor as the interrupt vector.

If the interrupt enable is not set, the interrupt flag can be polled. If the interrupt flag is set when loading the interrupt ID register (signifying that the last interrupt has not been acknowledged by the host processor), the interrupt ID register contents are overwritten and the indication of the previous interrupt is lost. By reading the VME control register (see below), the microcoder can determine if there is a pending interrupt since the interrupt flag is replicated in this register as the interrupt pending bit.

VME Bus Interface Logic

A major component of the Painting Processor is the *VME bus interface*. Six registers interface this logic to PPBUS. The internal register format section of this document contains details of these registers.

The VME control register controls the data transfer width (byte or word) and the VME bus address modifier bits. The VME status register contains information on the results of the last VME bus operation, accrued results, and the interrupt status (pending or not pending).

Two registers are used to generate the 24-bit VME bus address: the high and low VME address registers. These registers are implemented as a 24-bit counter:

- the low address register is the low 16 bits of the counter, and
- the high address register is the high 8 bits.

This counter can be incremented or decremented under microcode control.

In addition, the counter is buffered before being routed to the bus so that these two registers can be updated while a VME data transfer is active. The VME address registers specify a byte in VME address space. If doing word transfers, the least significant address bit should be zero. (If it is not zero, the hardware will execute the VME access as if it were zero but will flag an error.) If using the increment or decrement capability when executing word transfers, the microcoder must count the VME address register twice.

The read- and write-data registers are used to store the data fetched on a VME bus read cycle and the data to be written on a VME write cycle.

The following steps are used to execute a VME bus write:

1. The VME control register is initialized.
2. The VME address registers are loaded.
3. The VME write-data register is loaded.

4. The write is started using the miscellaneous control section of the microinstruction.
5. Step 4 automatically causes the VME busy flag to be set.
6. When the VME write completes, the VME ready flag (the complement of the busy flag) is automatically set.
7. A new write or read can be initiated.

Steps 1-3 can be done in any order. Step 4 could be coincident with step 1 or 3 (but NOT 2) after the other registers are loaded. When the VME busy flag is set, the VME control register and the VME write-data register cannot (hardware protected) be altered. The address registers can be updated when the VME busy flag is set.

The following steps are used to execute a VME bus read:

1. The VME control register is initialized.
2. The VME address registers are loaded.
3. The read is started using the miscellaneous control section of the microinstruction.
4. Step 3 automatically causes the VME busy flag to be set.
5. The requested data word or byte is loaded into the VME read-data register.
6. Step 5 automatically causes the VME ready flag to be set.
7. The VME read-data register is read by the Painting Processor.
8. A new read or write can be initiated.

Steps 1 and 3 can be done in the same microinstruction after step 2. Reading the VME read-data register will return garbage while the VME busy flag is set. Testing the busy flag will determine when a valid data word is read. The VME address registers can be altered when the VME busy flag is set.

If doing byte accesses, only the low half (bits 7 to 0) of the VME read and write-data registers are used. The hardware routes the desired byte to/from the correct VME data lines.

NOTE *The remaining sections of the Painting Processor are located on the optional GB board. That is, the GB board will contain all of the following features:*

- *Graphics Buffer Memory*
- *Integer Multiplier*
- *Mode Register*
- *PP PROM*

If the optional board is not installed, no hardware restrictions exist on the GP board but the firmware should not attempt to use any of the GB board features.

Graphics Buffer Memory

The *GB board memory* is implemented with 64 DRAM chips providing 1M 16-bit words. As shown in the block diagram of the Painting Processor, there are two data registers and one address pointer interfacing the DRAM to the PPBUS. The Graphics Buffer ready flag is a testable branch condition and indicates whether the hardware is ready to accept an address change or to execute a DRAM access.

The DRAM has two operation modes: Normal and Read-Modify-Write (RMW). In normal mode, the DRAM is a linear array with hardware assist for sequential access. Fill mode is a submode of normal mode; a fill-mode write causes data to be written into four consecutive locations in the same time required to write a single location. The RMW mode is similar except that reads followed by a write can be done to a single location. RMW mode is useful for the inner loops of hidden surface elimination algorithms.

Normal mode works as follows: the address pointer is initialized by loading the high and low Graphics Buffer address pointers. If a start-read command, specified in the miscellaneous control field of the microinstruction, is coincident with the load of the low address pointer, then a DRAM read is initiated. When the read is complete three cycles later, the fetched data word is loaded into the Graphics Buffer read-data register and can be read by selecting this register as the PPBUS source.

The next sequential location can be read by executing a start-read command. This command may be done coincident with a read of the Graphics Buffer read-data register—or not, as desired. When the start-read command is received, the Graphics Buffer address pointer is incremented and a memory read initiated. When the read is complete three cycles later, the fetched data word is loaded into the Graphics Buffer read-data register and can be read by selecting this register as the PPBUS source. In this way sequential reads can be performed.

Sequential writes start the same way: the high and low Graphics Buffer address pointers are loaded. But no start-read is done. Instead, a write to the Graphics Buffer write-data register is done. The value just loaded into the write-data register is then written into memory at the address in the pointer. When the memory write completes, the pointer is incremented. It takes three cycles for the memory write to finish.

Reads and writes can be mixed. The rules are:

- a start-read increments the pointer (unless coincident with a write to the low address pointer),
- executes a read, and
- loads the Graphics Buffer read-data register.

A load of the Graphics Buffer write-data register

- executes a write, and then
- increments the address.

Fill mode can be used only in normal mode and only when doing writes. No hardware protection is provided to prevent using fill mode with reads or in RMW

mode (other than to protect the hardware from damage) and results will be indeterminate.

Fill mode is entered by writing the appropriate bits in the Graphics Buffer high address pointer (see the chapter describing microcode format). Then the low address pointer is loaded to define the location of the write. The low order two address bits are ignored so that 4 consecutive locations on a modulo 4 boundary are loaded. The actual memory write is triggered by choosing the Graphics Buffer write-data register as the PPBUS destination. The value written into the write-data register is the value written into the 4 memory locations. After the write completes, the address pointer is incremented by 4 so that the next group of 4 locations can be written with a single load of the Graphics Buffer write-data register.

In summary, fill-mode writes work similar to normal-mode writes except that 4 locations are written per load of the Graphics Buffer write-data register and the address pointer is incremented by 4 following each write.

RMW mode works as follows: as above, the address pointer is initialized by loading the high and low Graphics Buffer address pointers. A start-read command must be coincident with the load of the low address pointer (the microcoder must ensure this because there is no hardware protection), and a DRAM read/modify/write (RMW) cycle is initiated. When the read is completed four cycles later, the fetched data word is loaded into the Graphics Buffer read-data register and can be read by selecting this register as the PPBUS source. But the DRAM remains in the RMW state.

Reading the Graphics Buffer read-data register merely routes the register contents to the PPBUS destination. The DRAM remains in the RMW state.

Loading the data write register causes new data to be written into the DRAM thus completing the RMW cycle. The address pointer is then incremented and a new RMW cycle initiated. Five cycles later, the next data word can be read from the Graphics Buffer read-data register.

After reading the read-data register and while the RMW cycle is still active, it may be determined that the data already in memory should not be modified. In this case, a start-read command is used to abort the active RMW. After the RMW is terminated, the address pointer is incremented and a new RMW cycle is initiated. Four cycles later, the next data word can be read from the Graphics Buffer read-data register.

The DRAM logic can remain in the state waiting for a write, a start-read, or an exit from RMW mode for a maximum of 10 microseconds. This means that the user must execute a write or a start-read (or exit) at least once every 10 microseconds while in this mode. It also means that the Painting Processor clock cannot be halted if in RMW mode. If the clock is halted, the DRAM memory will not be refreshed and the contents will be lost.

In normal, fill, or RMW mode, the Graphics Buffer busy (or ready, the opposite state of the same flag) is used to synchronize the Painting Processor microcode and the DRAM. Whenever a start-read or a load of the Graphics Buffer write-data register is done, the flag is set to busy. Whenever a memory read completes

signifying valid data in the Graphics Buffer read-data register, the flag is set to ready. In normal or fill mode, the flag is also reset to ready when a memory write completes. Thus a memory operation should not be initiated if the flag shows that the Graphics Buffer is busy. Because of the hardware protection described below, this test can be done coincident with the operation thereby saving microcode cycles.

Hardware protect is implemented in the DRAM to prevent the inadvertent initiation of a new operation while the current operation is still busy. Protected operations include

- a start-read command,
- writing the high or low address pointers,
- writing the write-data register, or
- reading the read-data register.

Executing these operations when the Graphics Buffer busy flag is set is as if the operations were never performed. If the microcode is also testing the Graphics Buffer busy flag, it is possible to loop on an instruction until it successfully completes.

In all modes, DRAM refresh is transparent to the user. However, refresh will have a minor effect on the timings given above—another reason why it is necessary to test the Graphics Buffer busy flag.

AM29L517 Integer Multiplier

The AM29L517 is a single-cycle, *16-bit integer multiplier.* The X and Y operand registers are routed to the X and Y inputs of the AM29L517; each operand register can be loaded in a single cycle. A 32-bit result is generated which can be read in two cycles. Each independent multiply takes six cycles: load X, load Y, delay one cycle to transfer the X and Y operands into the multiplier, delay one cycle to execute the multiply, read half of the result, and finally read the other half of the result. Chained operations where X or Y does not change or where the result is fed back into the input take less time.

The multiplier supplies a 32-bit result so that two 16-bit reads are necessary. The high and low half results are made available on alternate reads of the result. A bit in the mode register defines the default state as to which result half is returned first. Each time the X or Y operand register is loaded, this default state is engaged.

Mode Register

The *mode register* mode defines four AM29L517 inputs pertaining to unsigned or two's complement number representations and the default state as to which result half is returned first.

PP PROM

The *PP PROM* is identical to the VP PROM and is used for constant storage for numeric operations (for example, reciprocal lookups) when using the AM29L517.

5

VME Interface

VME Interface

5.1. Overview

This section describes how the Graphics Processor looks to VME bus masters, that is, the GP as a VME slave.

5.2. Microstore Interface

The microstore interface includes the registers to control the GP state, to determine GP status, and to load and verify the microstore. These registers are in a 32 Kbyte area in standard (24-bit) VME address space. Supervisory or non-privileged data accesses only are allowed. The sequential access option is not implemented.

The microstore registers include the following:

1. Board identification (8 bits),
2. GP control register,
3. GP status register,
4. Microstore address register,
5. Microstore data register.

The board identification is a fixed, 8-bit value which is read from the VME bus. The bits are set by jumpers on the GP board, and one of the bits is used to indicate the presence of a GB board. At system configuration time, this value is read to provide a positive indication that the GP is indeed installed in the system.

The GP control register contains the bits to reset and to halt/start the Viewing Processor and Painting Processor. In addition, the interrupt enable flag and the clear interrupt flag are located here.

The GP status register contains Viewing Processor and Painting Processor run/halt states, the interrupt enable state, the interrupt flag, the microstore column state, and the eight status flags, four each under control of the Viewing Processor and the Painting Processor.

The next two (microstore address and microstore data) registers are used to load and read back the microstore.

- First an address is specified—a 15-bit value pointing to one 64-bit (of which 56 are used) microword.
- Then four 16-bit quantities, most significant word of the 56-bit microword first, are loaded into the microstore data register and these four values are

written into the microstore at the selected location. (The most significant eight bits of the first write are the bits thrown away to make the 56-bit word.) The microstore address is then automatically incremented by the hardware to point to the next location and the next microword write can be performed.

Reads are similar. The microstore address register is initialized and a series of reads are performed to the microstore data register. Four 16-bit reads are required to read each microword. After each set of four reads, the microstore address register is automatically incremented by the hardware. If desired, reads and writes can be mixed.

To simplify the hardware, word-only accesses are supported to these microstore registers. Byte accesses are assumed to be word accesses. To simplify address decoding, these registers are repeated 4K times in the 32 Kbyte address space.

5.3. Shared Memory

The shared memory is a 32 Kbyte block of memory in the standard (24-bit) address space immediately after the 32 Kbyte area used by the microstore registers. This memory is accessible via a supervisory or non-privileged data access. Byte or word accesses can be made into the shared memory. The sequential access option is not implemented.

The GP shared memory responds as fast as possible to VME accesses. On writes, the data word and address are strobed into a latch and data acknowledge is immediately returned. The data word is written into the shared memory within 360 nsec of data acknowledge. On reads, the data word is available within 360 nsec of being addressed by the VME and data acknowledge is generated at that time.

5.4. VME Bus Addressing (GP as a VME Slave)

This section describes registers affiliated with VME bus addressing, when the Graphics Processor is acting as a slave to the VME.

Microstore Interface Registers

The microstore interface registers respond to the following VME bus addresses.

```
Address Modifier = standard supervisory data access (3D) or
                   standard non-privileged data access (39)

 23           16 15 14     3 2      1
+---------------+--+--------+--------+
|hardware switch| 0|  XXXX  |register|
+---------------+--+--------+--------+

hardware switch  ⇒  selects a 64 Kbyte block in VME
                                        standard address space
bit 15           ⇒  must be 0 to select interface registers
XXXX             ⇒  don't care
register         ⇒  selects a microstore interface register
                            (see below for encoding)
```

The hardware switch is the same as the shared memory hardware switch. The microstore registers are:

- 00 — Board identification (read only) (write to this address will cause a bus error acknowledgement)
- 01 — GP control register (write only)
- 01 — GP status register (read only)
- 10 — Microstore address register (read/write)
- 11 — Microstore data register (read/write)

The format of these registers is shown below. Because bits 14-3 are don't care, these four word locations are replicated 4K times in the 32 Kbyte page. Word-only accesses are supported; byte accesses are interpreted as word accesses and longword accesses cause the generation of a bus error acknowledge.

Shared Memory

The shared memory responds to the following VME bus addresses.

```
Address Modifier = standard supervisory data access (3D) or
                       standard non-privileged data access (39)

 23           16 15 14              1
+---------------+--+-----------------+
|hardware switch| 1|      word       |
+---------------+--+-----------------+

hardware switch  ⇒  selects a 64 Kbyte block in VME
                                          standard address space
bit 15           ⇒  must be 1 to select shared memory
word             ⇒  selects the word within the shared memory
```

The hardware switch is shared by the microstore interface registers. The word field forms the 14-bit address into the 16K-by-16 shared memory. Byte or word accesses are allowed into this memory. Longword accesses cause the generation of a bus error acknowledge. (User programs may use longword read and writes to the shared memory because the host processor will break these up into two consecutive word accesses.)

5.5. Microstore Interface Register Formats

This section describes the microstore interface register formats, while the Graphics Processor is acting as VME slave.

Board Identification

This read-only register contains a constant pattern, selectable as a hardware option. This register can be read, for example at system configuration time, to provide a positive indication that the GP is indeed installed in the system.

```
15          8 7       1  0
+------------+---------+--+
|  not used  | pattern |GB|
+------------+---------+--+

 not used ⇒  read as 1
 pattern  ⇒  constant 7-bit pattern† defined in the hardware
 GB       ⇒  0 indicates the GB is attached
             1 indicates no GB is attached
```

NOTE *The GB bit is a jumper, not a POSITIVE indication that the GB board is installed. It does, however, enable signals between the boards; if this bit indicates NO Graphics Buffer is present, then one cannot communicate with the GB board even if it really is present.*

GP Control Register

The write-only GP control register is formatted as follows.

```
15  14 10 9   8  7   6  5        3 2        0
+-----+-----+----+---+---+----------+----------+
|clrif| xxx |ienbl|xxx|rst|VP control|PP control|
+-----+-----+----+---+---+----------+----------+

 clrif       ⇒    clear interrupt flag: clears a pending
                          interrupt; must be written as a 1
                          to allow subsequent interrupt.
 xxx         ⇒    ignored
 ienbl       ⇒    interrupt enable
                          00   interrupt enable state unchanged
                          01   interrupt enabled
                          10   interrupt disabled
                          11   interrupt enable state will toggle
 rst         ⇒    reset
 VP control  ⇒    Viewing Processor control bits
 PP control  ⇒    Painting Processor control bits
```

The reset bit must be toggled under software control (set to logic 1 then returned to logic 0) to generate a reset to the GP board. (If left in the 1 state, the GP board will not function.) Reset does the following:

- puts both the VP and PP into halt mode
- disables GP-generated interrupts
- enables the reading of the floating point set A registers (set B can be read as a diagnostic function)
- resets the FIFO and FIFO control logic
- sets the FIFO to the VP-to-PP direction

†By current software convention, the 7-bit pattern is 0x75.

- sets Graphics Buffer flag to ready.

The VP and PP control fields are formatted as follows.

```
+-----+---+----+
|strt0|hlt|cont|
+-----+---+----+

strt0 ⇒   Start-from-0 is a modifier for the continue.
hlt   ⇒   A halt stops the selected processor, that is,
              disables its clocks.
cont  ⇒   A continue starts the selected processor.
              If strt0 is also set, then the processor
              starts at location 0; otherwise it starts
              at the location where it was last halted.
```

The software must create a rising edge with the cont and hlt signals to initiate the desired function. This is done by first ensuring that the signal is 0 (perhaps by writing a 0 into the bit). Then write a 1 into the desired bit to create the rising edge. The strt0 signal is sampled when the cont signal is loaded with a 1.

Asserting cont and hlt at the same time toggles the run/halt state. If this toggle causes the processor to start, it will start at 0 if strt0 is set or at the location where it was last halted if strt0 is not set.

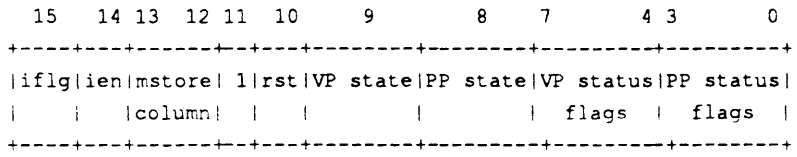
When starting either the PP, VP, or both, the microcoder must be aware of the pipeline nature of the instruction registers. When a continue is executed, the processors will first execute the command *already* in the instruction register then either continue with the instruction at the next address (strt0 not asserted) or with the instruction at location 0 (strt0 asserted). In the latter case, the command already in the instruction register may be bogus. Hardware protection is provided to prevent a VME bus access during this cycle but no protection exists to prevent the possible corruption of internal GP data.

When starting up the GP, the following procedure is recommended:

1. Load all fields of location 0 in the microstore with no-ops—*except* for the AM2910 field, which selects "jump to zero."
2. Execute a continue with strt0 for both the VP and PP. (One bogus instruction will be executed, then the instruction at location 0 will be loaded and continually executed. This primes the pipeline.)
3. Halt the VP and PP.
4. Load the desired microcode.
5. Start the VP and/or PP with a continue and strt0. The "jump to zero" which is in the instruction register will be executed, but there will be no negative side-effects.

GP Status Register

The 16-bit read-only GP status register is formatted as follows:

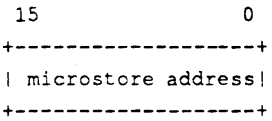



```
iflg    ⇒  interrupt flag
                0 = no pending interrupt
                1 = pending interrupt
ien     ⇒  interrupt enable
                0 = interrupt not enabled
                1 = interrupt enabled
microstore
column  ⇒  indicates which microstore column will
                be read or written on the next access to
                microstore data register, reset to 0 on
                each write to the microstore address
                register (see below for column definition)
rst     ⇒  reset, reflects the state of the GP control register
                reset bit (the GP will not function if this bit is a logic 1)
VP state ⇒ Viewing Processor state
                0 = VP in halt state
                1 = VP is run state

PP state ⇒ Painting Processor state
                0 = PP in halt state
                1 = PP in run state
VP status
  flags  ⇒ four general-purpose flags set by the Viewing Processor
PP status
  flags  ⇒ four general-purpose flags set by the Painting Processor
```

Microstore Address Register

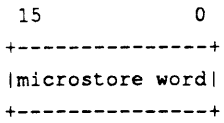
This register is used to point into the microstore for reads and writes. The register is formatted as follows for both reads and writes.



This value points to a 56-bit microword. Bit 15 is not used when making accesses to the microstore (a 32 Kbyte microstore is the maximum), but is otherwise readable and writable from the VME bus.

Microstore Data Register

This register is used to access the microword pointed to by the microstore address register. The register is formatted as follows for both reads and writes.



The following steps are used to load the microstore:

1. Load the microstore address register to point to the first microstore word to be written.
2. Write column 0 (bits 55 to 48) of the 56-bit microword. These eight bits should be in the low byte (7 to 0) of the data word.
3. Write column 1 (bits 47 to 32) of the microword.
4. Write column 2 (bits 31 to 16) of the microword.
5. Write column 3 (bits 15 to 0) of the microword.

NOTE *The column selected at any time is indicated by the microstore column bits which are read back as part of the GP status register.*

6. The hardware automatically increments the microstore address register.
7. Repeat steps 2-6 for the number of microinstruction words to be loaded.

Reads are similar.

NOTE *Word-only accesses are allowed when accessing the microstore.*

5.6. VME Interrupts

The following describes the interrupt operation of the GP.

To initiate an interrupt cycle, the Painting Processor interrupt id register is loaded (chosen as a PPBUS destination). This event will set the interrupt flag in the GP status register if it is not already set. (If it was already set, the previous event was not acknowledged by the host processor.) Note that the interrupt flag in the GP status register and the pending interrupt bit in the PP VME status register are the same bit.

If the interrupt enable bit in the GP control register is *not* set, no VME interrupt cycle is initiated. Even if the interrupt enable is set later while the interrupt flag is set, no VME interrupt cycle will be performed. Instead, the host processor (actually, any VME master) must poll the interrupt flag. When detecting the event (interrupt flag = 1), the host processor must clear the interrupt flag by a write to the GP control register with the clrif bit set.

If the interrupt enable is set when the interrupt flag is set, then a VME interrupt cycle is initiated. As per normal VME interrupt cycles, the GP will place the interrupt id register on the VME bus when so instructed, and the host processor uses this value as the interrupt vector to jump to the GP interrupt service routine. In the interrupt service routine, the programmer must reset the interrupt flag by a write to the GP control register with the clrif bit set. (If this bit remains set, no further interrupts can be generated.)

Note that the state of the interrupt enable does not change and thus does not have to be turned on again under software control. However, if it is possible to service

a GP interrupt before the GP has executed the interrupt cycle (for example, several devices are tied to the same vector and then a poll is performed), the GP will be poised to request an interrupt even though the interrupt has already been serviced. To prevent this, the programmer must toggle the interrupt enable bit—reset it, then set it again. Thus the interrupt service routine needs to access the GP control register twice—once to clear the interrupt flag and the interrupt enable, and again to turn the interrupt enable back on.

If the interrupt flag is set when the interrupt id register is loaded from the Painting Processor, then all that happens is that the new value is written into the interrupt id register potentially defining a new interrupt vector. Even if the interrupt enable is set, no new VME interrupt cycle is performed. It is possible that the host processor is clearing the interrupt flag or the VME interrupt cycle is in-progress at the exact moment that the interrupt id register is being altered and unpredictable things could happen; for example, vectoring to the wrong location in the host processor. (The probability of this occurrence is very small.) To prevent this from happening, the PP microcoder should ensure that the interrupt flag is not set (read the interrupt pending flag in the VME status register) before writing the interrupt id register.

5.7. GP as VME Master

The Painting Processor has the capability of acquiring the VME bus and performing a data transfer.

For VME accesses, the PP must select a slave as the data source or destination. The slave is selected by the 24 address lines and the 6 address modifier bits. As described in this reference manual, two VME address registers are used to compute the 24-bit address. The address modifier bits are specified in the VME control register.

For VME writes, a data byte or word is transferred from the GP to the selected slave. The data value is loaded into the VME write-data register. Byte or word is selected by bits in the VME control register.

The miscellaneous controls field of the microinstruction is used to initiate the write. If the GP is not the current bus master, the GP requests the bus. When the request is honored or if the GP was already the bus master, the data transfer is executed. When the transfer ends, the GP remains master until some other VME device requests mastership.

VME reads are similar. The read is initiated by properly encoding the microcode miscellaneous control field. If the GP is not the current bus master, it requests the bus. When bus mastership is granted or if the GP was already the bus master, the data transfer is executed. The returned data value is loaded into the VME read data register. As with writes, the GP remains bus master until another VME master requests the bus.

6

Internal Registers

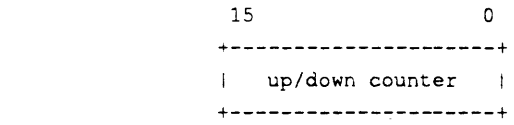
Internal Registers

6.1. Viewing Processor

This section describes those internal registers specific to the Viewing Processor portion of the GP.

Shared Memory Pointer

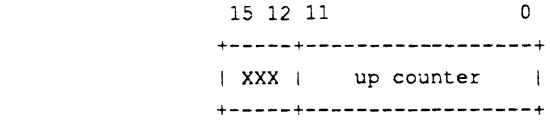
The write-only shared memory pointer is formatted as follows:



Current shared-memory size is 16 Kwords. The up/down count of this register is under microcode control.

Source A, Source B, Destination Pointers

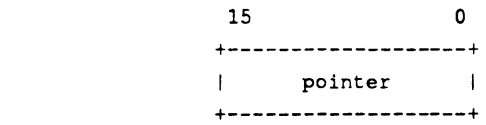
These write-only pointers are formatted as follows:



Each counter is implemented with three 4-bit counters allowing for up to 4K of 32-bit floating point numbers. The current design only implements 2K, thus bit 11 is unused. This up-only counter is under microcode control.

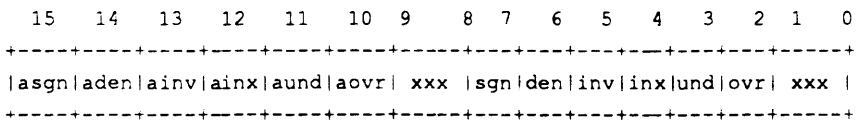
VP PROM Pointer

The write-only VP PROM pointer is formatted as follows:



Floating Point Status Register

This 16-bit read-only status register is formatted as follows. The bits of this register are read back in their complement state.



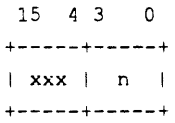
asgn ⇒ accrued sign bit
aden ⇒ accrued denormalized input to multiplier
ainv ⇒ accrued invalid input
ainx ⇒ accrued inexact result
aund ⇒ accrued underflow result
aovr ⇒ accrued overflow result
xxx ⇒ don't care
sgn ⇒ sign of last floating point result
den ⇒ denormalized input to multiplier
inv ⇒ invalid input
inx ⇒ inexact result
und ⇒ underflow result
ovr ⇒ overflow result
xxx ⇒ don't care

These signals are active low; that is, a logic 0 indicates the condition occurred. The "don't cares" are read back as logic ones.

On each floating point result, the status flags are updated. The accrued status is updated by logically ORing the new status with the old accrued status. The accrued status is cleared on each read of this register.

n Register

This write-only register is formatted as follows:



n register values override bits 12-9 in the following four 29116 instructions:

```
Bit Oriented Instructions
Rotate By n Bit Instructions
Rotate and Merge Instruction
Rotate and Compare Instructions
```

Replacing bits 12-9 in these four instructions by the value in the n register allows assignment of runtime variables.

Interprocessor Flag #1 Register

This write-only register is formatted as follows:

```
 15  14 13  8 7          0
+------+-----+-----------+
| fdir | xxx |  flags #1 |
+------+-----+-----------+

 fdir ⇒ FIFO direction control
            power-on default is VP-to-PP direction
               00: does not change direction
               01: VP-to-PP direction
               10: PP-to-VP direction
               11: toggle FIFO direction
```

The flags are read by the PP as the interprocessor flag #1 register.

Interprocessor Flag #2 Register

This read-only register is formatted as follows:

```
 15    11 10         9  8 7         0
+-------+-----------+--+-----------+
|  1's  |FIFO status| 0| flags #2  |
+-------+-----------+--+-----------+

 FIFO status  ⇒
    bit 10   0: FIFO has 0 or 1 word to be read
             1: FIFO has 1 or more words to be read

    bit 9    0: FIFO direction is VP-to-PP
             1: FIFO direction is PP-to-VP

 flags #2    ⇒  flags set by the PP
```

Status Flags/LED Register

This write-only register is formatted as follows:

```
 15  14 13  4 3           0
+------+-----+-------------+
| fpsel| xxx |status flags |
+------+-----+-------------+

 fpsel ⇒ floating point register set select;
            power-on default selects A
             00: no change
             01: selects set A
             10: selects set B
             11: toggles set
status ⇒ bits set by the VP and read via the
             VME bus as part flags of the GP status register
```

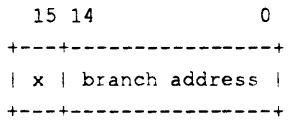
The status flags also drive four LEDs on the GP board. A logic 0 turns on the LEDs.

The fpsel is a diagnostic function allowing read-back of either set of floating point registers. The floating point register sets A and B are duplicates, necessary to increase the bandwidth into the registers. Both are written at the same time with the same data. Reading the same location from either set should return the same value. During normal operation, set A is read when chosen as a VPBUS source. These bits allow set B to be read from the VPBUS for diagnostics.

When reading from set A, floating point source A pointer is used to specify the location fetched. When reading from set B, the source B pointer is used.

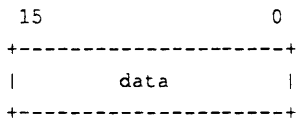
Branch Register

The write-only branch register is formatted as follows:



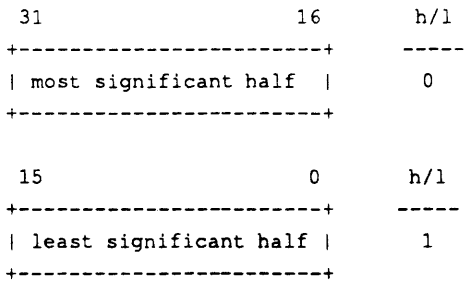
Shared Memory

The shared memory registers are 16-bit data registers which are formatted as follows:



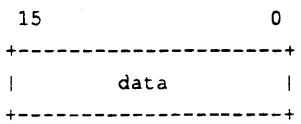
Floating Point Registers

The 32-bit floating point registers are formatted as follows:



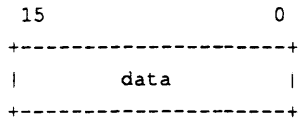
VP PROM Registers

Data read from the VP PROM are formatted as follows:



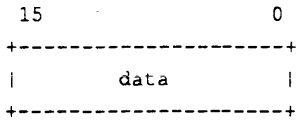
FIFO Registers

Data read from the FIFOs (FIFO 1 and FIFO 2) are formatted as follows:



29116 Registers

Data transferred to or from the 29116 are formatted as follows:

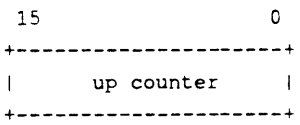


6.2. Painting Processor

This section describes those internal registers specific to the Painting Processor portion of the GP.

Scratchpad Pointer

The write-only scratchpad pointer is formatted as follows:

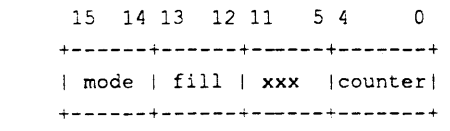


The scratchpad memory is 4 Kwords. The count of this register is under microcode control.

Graphics Buffer Address Pointers

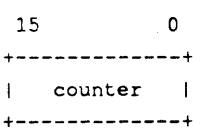
The write-only Graphics Buffer address pointers used to access the dynamic memory located on the Graphics Buffer board are formatted as follows—

high address pointer:




```
mode ⇒ selects normal or read-modify-write mode;
             power-on default is normal mode
              00: no change to mode
              01: select normal mode
              10: select RMW mode
              11: toggle mode
fill ⇒ selects fill mode (a normal submode);
             power-on default is fill mode not enabled
              00: no change
              01: enable fill mode
              10: disable fill mode
              11: toggle state
counter ⇒ high-order continuation of the address
               counter from the low address pointer
```

low address pointer:



The Graphics Buffer pointer is a 21-bit counter. The counter increments on a start-read command or a write command. More details are given in the following chapter, which describes the microcode format.

VME Control Register

This write-only register is formatted as follows:

```
 15   7 6                1      0
+------+-------------------+-----------+
| xxxx | address modifier |access type|
+------+-------------------+-----------+

address modifier ⇒  the VME bus address modifier bits
access type      ⇒  selects the access type on VME bus data transfers
                        0: byte
                        1: word
```

VME Status Register

This read-only register is formatted as follows:

```
 15  14  5    4     3   2   1   0
+----+-----+------+----+---+---+--+
|ipnd| xxx |illacc|aerr|ato|err|to|
+----+-----+------+----+---+---+--+

ipnd   ⇒   interrupt pending flag (same as the GP control
               register interrupt flag)
xxx    ⇒   don't care
illacc ⇒   accrued illegal access, set if a word transfer
               was executed with address bit 0 set to 1.
aerr   ⇒   accrued bus error
ato    ⇒   accrued time-out error
err    ⇒   bus error, set if a bus error acknowledge rather than
               the normal dtack was returned on the last GP data
               transfer
to     ⇒   time-out error, set if no dtack or bus error was returned
               within 5 µs on the last VME data transfer
```

These signals are active low; that is, a logic 0 indicates the condition did occur. The don't cares are read back as logic ones.

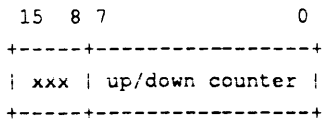
Word accesses to VMEbus byte locations are illegal. Since the least significant address bit (A00) is not output onto the VMEbus, a word access to a byte location behaves like an access to a word location. Thus if a VME word transfer with a byte address (A00 = 1) is specified, the VME transfer is executed as if A00 = 0 but the illacc error flag in the VME status register is set.

The accrued errors are updated by ORing the previous state of the accrued error with the current state of the corresponding error signal. The accrued errors are cleared each time this register is read (chosen as a PPBUS source).

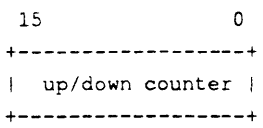
VME Address Registers

These write-only registers are formatted as follows—

high address register:



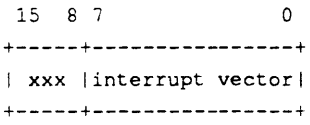
low address register:



These two registers form a 24-bit up/down counter.

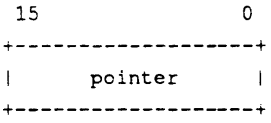
Interrupt ID Register

Writing this register generates an event which sets the interrupt flag in the GP status register and the interrupt pending flag in the VME status register. If enabled, this event generates a VME interrupt cycle.



PP PROM Pointer

The write-only PP PROM pointer is formatted as follows:



Multiplier Mode Register

This write-only register controls the number representation into the AM29L517 multiplier, unsigned or two's complement.

```
 15  5    4    3    2    1    0
+-----+------+----+------+---+
| xxx |mpenbl| mode |format|rnd|
+-----+------+----+------+---+

mpenbl ⇒  selects most significant (0) or least significant (1)
              half of multiplier product to be read first (see below)
mode   ⇒  AM29L517 XM and YM bits
format ⇒  AM29L517 format adjust bit
rnd    ⇒  AM29L517 round control
```

A multiplier operation works as follows. First one or both of the X and Y operand registers are loaded. A load of either the X or Y operand register causes the mpsel flip-flop to be set if mpenbl = 1 or reset if mpenbl = 0; (loading both causes a redundant set or reset of the mpsel flip-flop.) After delaying two cycles for the multiply execution, the multiplier product is read. If the mpsel flip-flop = 0, then the most significant half word is read; if the mpsel flip-flop = 1, then the least significant half word is read. After each read of the multiplier result, the mpsel flip-flop is toggled so that the next read enables the other half of the result. Successive reads thus read alternative halves of the product.

NOTE *It is not necessary to read both halves of a result if, for example, it is known that the result is 16 bits or less.*

If the multiplier result is routed to either the X or Y operand register, then the half selected by the mpenbl bit will be read.

n Register

This write-only register is formatted as follows:

```
 15  4 3   0
+-----+-----+
| xxx |  n  |
+-----+-----+
```

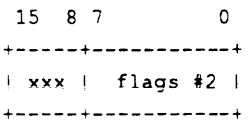
n register values override bits 12-9 in the following four 29116 instructions:

```
Bit Oriented Instructions
Rotate By n Bit Instructions
Rotate and Merge Instruction
Rotate and Compare Instructions
```

Replacing bits 12-9 in these four instructions by the value in the n register allows assignment of runtime variables.

Interprocessor Flag #2 Register

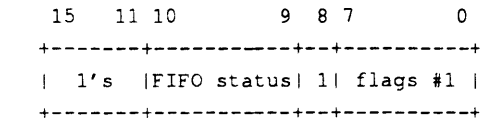
This write-only register is formatted as follows:



The flags are read by the VP as the interprocessor flag #2 register.

Interprocessor Flag #1 Register

This read-only register is formatted as follows:

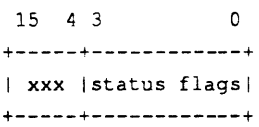



```
FIFO status ⇒
     bit 10     0: FIFO has 0 or 1 word to be read
                1: FIFO has 1 or more words to be read
     bit 9      0: FIFO direction is VP-to-PP
                1: FIFO direction is PP-to-VP

flags #1    ⇒   flags set by the VP
```

Status Flag/LED Register

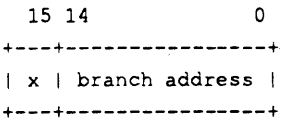
This write-only register is formatted as follows:



These flags are read via the VME bus as part of the GP status register. They also drive four LEDs; a logic 0 turns on the corresponding LED.

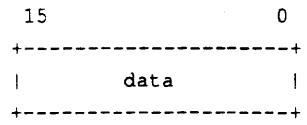
Branch Register

The write-only branch register is formatted as follows:



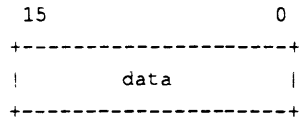
Scratchpad Memory

The scratchpad registers are 16-bit data registers which are formatted as follows:



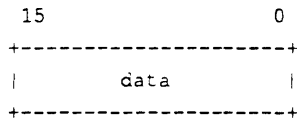
GB Data Registers

The Graphics Buffer read- and write-data registers are formatted as follows:



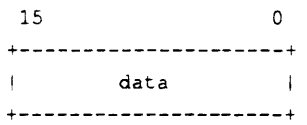
VME Data Registers

The VME read- and write-data registers are formatted as follows:



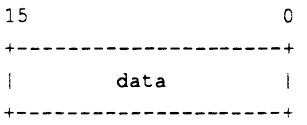
PP PROM Registers

Data read from the PP PROM are formatted as follows:



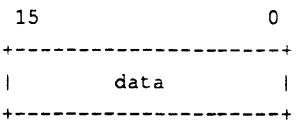
Multiplier X, Y, and Result Registers

The multiplier registers are formatted as follows:



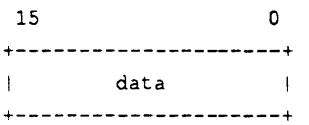
FIFO Registers

Data read from the FIFOs (FIFO 1 and FIFO 2) are formatted as follows:



29116 Registers

Data read from the 29116 are formatted as follows:



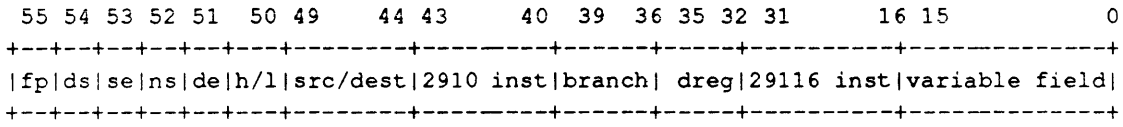
7

Microcode Format

Microcode Format

7.1. Viewing Processor Microcode

The microcode for the Viewing Processor is 56 bits wide and is arranged in twelve fields.




```
fp    ⇒     floating point type instruction, controls variable field
                  0: general field enabled
                  1: floating point operation enabled
ds    ⇒     data source to AM2910 D input
                  0: general field selected
                  1: branch register is selected
se    ⇒     status (zero, negative, carry, overflow) update enable
                  0: enables status update
                  1: disables status update
ns    ⇒     n field select
                  0: instruction register n field selected
                      (part of the 29116 inst field)
                  1: n register selected
de    ⇒     destination enable, controls use of dreg field
                  1: dreg is used for miscellaneous controls
                  0: dreg is used to select AM29116 destination
                       register in a 2-address instruction
h/l   ⇒     high/low specifier for floating point operations (Weitek chips
                        and floating point registers)
                  0: most significant word
                  1: least significant word
src/dest    ⇒ selects VPBUS source and destination (see below for encoding)
2910 inst   ⇒   AM2910 sequencer instruction (see AMD literature)
branch      ⇒   branch condition code select
dreg        ⇒   destination register or miscellaneous controls
29116 inst  ⇒   16-bit AM29116 instruction (see AMD literature)
variable field ⇒ variable format field: general field or floating point
```

AM29116 Instruction

The 29116 inst, de, dreg, ns and se fields apply to the AM29116 microprocessor. The 16-bit 29116 instruction is described in detail in the AMD literature.

The GP supports two address operations with the AM29116. For example, the following is possible:

```
Rm <-- Rn op ACC
```

Rn is specified by the least significant 5 bits of the 29116 inst field. If de is 0, then Rm is selected by the dreg field. (If de is 1, then m equals n; that is, the source and destination are the same.) Since dreg is only four bits, Rm and Rn must be in same group of 16 registers within the AM29116, 0 to 15 or 16 to 31. If de is 1, the dreg field is used as described below in the miscellaneous controls section, and the least significant four bits of the AM29116 instruction are not altered.

NOTE *The AM29116 does not support two address instructions with immediate instructions. For an immediate instruction, de must equal 1.*

For AM29116 bit-oriented instructions (test bit, set bit, etc.), bits 12-9 of the AM29116 instruction select n, the bit to be operated on. Having this bit hardcoded in the microinstruction severely limits its usefulness. Therefore it is possible to substitute a runtime value (from the n register) for the hardcoded value. ns=0 selects the assemble time value, ns=1 selects the n register.

The bit se controls the updating of the status register containing the AM29116 flags: zero, negative, carry, and overflow. se=0 enables updating, se=1 disables updating.

The carry status bit changes its meaning slightly for subtract operations. For these operations, a logic 1 indicates no borrow and a 0 indicates borrow. This has significance when doing double precision operations.

NOTE *It is the responsibility of the microcoder to ensure that it makes sense to use the de or ns capabilities. For example, if the microinstruction is a non-register operation, setting de to 0 and enabling the dreg field into the AM29116 could create havoc.*

Miscellaneous Controls

If de=1, then the dreg field is used for the following functions:

```
0000  no operation
0001  clear shared memory pointer
0010  count up shared memory pointer
0011  count down shared memory pointer
0100  count up floating point source A pointer
0101  count up floating point source B pointer
0110  count up floating point destination pointer
0111  count up floating point source A and B pointers
1000  count up floating point source A and destination pointers
1001  count up floating point source B and destination pointers
1010  count up floating point source A, B and destination pointers
1011  count up shared memory pointer and floating point
                  source A and source B pointers
1100  count up shared memory pointer and floating point
                  destination pointer
1101  count down shared memory pointer and count up
                  floating point source A and source B pointers
1110  count down shared memory pointer and count up
                  floating point destination pointer
1111  reserved
```

Source and Destination

A single VPBUS operation can be done during each VP cycle. The source/destination field is encoded as follows:

```
          VPBUS source --> VPBUS destination

000000  reserved - no source or destination
000001  interprocessor flag #2 register --> AM29116
000010  AM29116 --> status flag/LED register
000011  AM29116 --> n register
000100  reserved
000101  AM29116 --> interprocessor flag #1 register
000110  AM29116 --> FIFO #1
000111  AM29116 --> AM29116
001000  AM29116 --> branch register
001001  AM29116 --> VP PROM pointer
001010  AM29116 --> shared memory
001011  AM29116 --> floating point register
001100  AM29116 --> floating point source A pointer
001101  AM29116 --> floating point source B pointer
001110  AM29116 --> floating point destination pointer
001111  AM29116 --> shared memory pointer
010000  reserved
010001  FIFO #2 --> AM29116
010010  FIFO #2 --> branch register
010011  FIFO #2 --> VP PROM pointer
010100  reserved
```

```
010101  FIFO #2 --> shared memory pointer
010110  shared memory --> FIFO #1
010111  shared memory --> AM29116
011000  shared memory --> branch register
011001  shared memory --> VP PROM pointer
011010  reserved
011011  shared memory --> floating point register
011100  shared memory --> floating point source A pointer
011101  shared memory --> floating point source B pointer
011110  shared memory --> floating point destination pointer
011111  shared memory --> shared memory pointer
100000  VP PROM --> FIFO #1
100001  VP PROM --> AM29116
100010  VP PROM --> shared memory
100011  VP PROM --> floating point register
100100  reserved
100101  general field --> interprocessor flag #1 register
100110  general field --> FIFO #1
100111  general field --> AM29116
101000  general field --> branch register
101001  general field --> VP PROM pointer
101010  general field --> shared memory
101011  general field --> floating point register
101100  general field --> floating point source A pointer
101101  general field --> floating point source B pointer
101110  general field --> floating point destination pointer
101111  general field --> shared memory pointer
110000  reserved
110001  floating point status register --> AM29116
110010  floating point status register --> shared memory
110011  floating point status register --> floating point register
110100  reserved
110101  reserved
110110  floating point register --> FIFO #1
110111  floating point register --> AM29116
111000  floating point register --> branch register
111001  floating point register --> VP PROM pointer
111010  floating point register --> shared memory
111011  floating point register --> floating point register
111100  floating point register --> floating point source A pointer
111101  floating point register --> floating point source B pointer
111110  floating point register --> floating point destination pointer
111111  floating point register --> shared memory pointer
```

FIFO #1 and FIFO #2 are the same FIFO. FIFO #1 is the VP-to-PP direction and FIFO #2 is the PP-to-VP direction.

When the AM29116 is chosen as the bus source, its internal Y bus is routed to the VPBUS. When the AM29116 is chosen as the destination, its internal D register is made transparent until the end of the cycle at which time it is latched for possible use in subsequent instructions.

When the shared memory is chosen as the VPBUS destination, the actual write into the memory is executed in the next cycle. This means that a write followed by a read in consecutive cycles is illegal. There is no hardware protection against this occurrence and results are indeterminate.

Similarly when a floating point register is chosen as the VPBUS destination, the actual write into the memory is executed in the next cycle. However because there are separate source and destination pointers, there is no restriction for reads on subsequent cycles. Also because of the separate pointers, it is possible to move a floating point register from one location to another in one microinstruction.

In both of these cases, after the write the microcoder must wait at least one cycle before reading the data written.

Another hardware constraint involves the interprocessor flag register. It is not possible to read this register into the AM29116 and manipulate it in a single cycle. It must first be read into the AM29116 D-latch and then manipulated by subsequent instructions.

Hardware Protection

Protection is implemented in the hardware to prevent the following operations:

- reading an empty FIFO #2,
- writing a full FIFO #1,
- reading or writing the FIFO when in the wrong direction.

If these operations are attempted, the hardware subsection never receives the command; that is, as far as the hardware is concerned, it is as if the instruction was never attempted. This allows the operation to be looped until successful completion. For example, the following statement will adapt to the FIFO state:

```
HERE:  MOVE Rn --> FIFO #1; BR HERE IF FIFO #1 IS FULL.
```

Note that since it is unknown how many times the instruction will be attempted, no arithmetic operation other than the move to FIFO #1 should be performed. For example, with the following instruction, the final value of Rn is unknown; it will be incremented once for each unsuccessful attempt and once for the successful attempt.

```
HERE:  Rn <-- Rn+1; MOVE Rn --> FIFO #1; BR HERE IF FIFO #1 IS FULL.
```

FIFO #1 and FIFO #2 are actually the same "reversible" FIFO. FIFO #1 is for VP-to-PP transfers and FIFO #2 is for PP-to-VP transfers. It is therefore undefined for the VP to read FIFO #2 while the direction is VP-to-PP, and to write FIFO #1 while the direction is PP-to-VP. The branch condition codes are defined as follows:

Direction	Condition Code
VP-to-PP	FIFO #1 full or not full is valid
VP-to-PP	FIFO #2 is always not empty
PP-to-VP	FIFO #1 is always not full
PP-to-VP	FIFO #2 empty or not empty is valid

No hardware protection is provided for reading the VP PROM before the data word is valid.

Because slow PROMs are used to provide the needed size, there is a two cycle delay between loading the VP PROM pointer and having access to valid data. No hardware protect is implemented to ensure valid data; the microcode must delay at least the minimum cycles.

Branch Logic

The branch logic includes the AM2910 instruction, the branch condition select field, and the ds bit. The AM2910 microsequencer is described in the AMD literature.

The branch field selects the branch condition select option as follows:

```
0000  Zero
0001  Negative
0010  Carry
0011  Overflow
0100  FIFO #1 not full
0101  FIFO #2 not empty
0110  Last floating point result negative
0111  Unconditional pass
1000  Not zero
1001  Non-negative
1010  No carry
1011  No overflow
1100  FIFO #1 full
1101  FIFO #2 empty
1110  Last floating point result non-negative
1111  Never pass
```

The ds bit is used to determine whether the contents of the branch register or the general field (see below) are routed to the D input of the AM2910, possibly the address of the next microinstruction. ds=0 selects the general field; ds=1 selects the contents of the branch register.

The status bits are updated at the end of each cycle (conditional update for the zero, negative, carry, and overflow flags), and can be used at the beginning of the next cycle for conditional branches. For example, to determine if the AM29116 Rn and accumulator are equal, the code would be as follows.

```
NO DESTINATION <-- Rn - ACC, SE
NOP; BR TO EQUAL IF ZERO STATUS FLAG IS SET
```

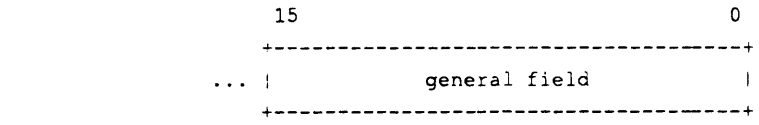
Count Hardware

The miscellaneous controls are used to enable the counting of the shared memory pointer. If counting coincides with a shared memory read cycle or on a cycle which is not accessing the shared memory, the count occurs at the end of the current cycle. For a shared memory write cycle, the count is executed at the end of the next cycle because, as explained above, the write is done in the next cycle. If the current cycle is a shared memory write with shared memory count enabled then the next consecutive cycle should not be a shared memory read cycle or a non-shared-memory-access cycle with shared memory count enabled.

The floating point register pointers can be incremented under microcode control. Similar to the shared memory pointer, a restriction exists on the incrementing of the floating point destination pointer after a write cycle. If the current cycle is a VPBUS write to the floating point registers with the destination pointer increment enabled, then the next cycle should not enable the incrementing of the destination pointer unless it is also a VPBUS write to the floating point registers.

General Field

The general field is a way of specifying a value in the microcode and using it in one of several places. For example, it could be an address pointing to the possible next microinstruction (see ds bit above). It could be a constant loaded into an AM29116 register. It could be an address pointing to a pre-defined constant in the floating point registers. The format of variable field when fp=0 is as follows.

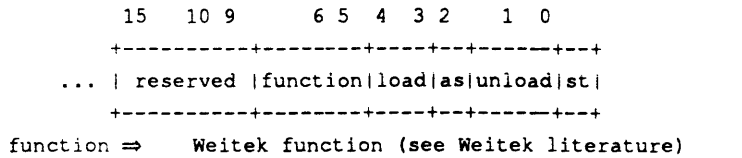


The general field is routed onto the VPBUS bus if selected by the source field and/or routed to the D input of the AM2910 microsequencer and bank select logic if selected by the ds bit.

Floating Point

In the discussion below, it will be helpful to refer the Weitek data sheet describing the floating point processor, the Weitek 1032/1033. 5 MHz Weitek parts are used in the Graphics Processor.

If the fp bit is a 1, then the instruction is a floating point operation, and the format of the variable field is as follows.




```
function ⇒    Weitek function (see Weitek literature)
load     ⇒    Weitek load control (see Weitek literature)

     (The function and load bits are wired in parallel to
     the two Weitek chips.)

as       ⇒    A source (to Weitek chips) select
                  0 - register A set
                  1 - result from Weitek chips (feedback loop
                      for chained operation)
unload   ⇒    select which chip receives the unload enable
                  00 - neither chip
                  01 - ALU
                  10 - Multiplier
                  11 - illegal (indeterminate results)
st       ⇒    enables (st=1) the storing of a floating point
                  result into the floating point register
                  pointed to by the destination pointer
```

If a floating point operation is selected (fp=1), then the microcode should not select the general field as the source to the VPBUS or as the source to the AM2910. However, there is no hardware protection to prevent this.

If not doing a floating point operation (fp=0), the load is forced to nop, as is forced to 0, and the unload field and store enable are disabled.

When running in flowthrough mode, the instruction sequence for a floating point operation is as follows.

```
cycle
 0    initiate floating point operation, most significant word of
          source A (or Weitek result) and source B are loaded
 1    initiate floating point operation, load least significant word
 2    delay - no floating point operation
 3    delay - no floating point operation
 4    delay - no floating point operation
 5    delay - no floating point operation
 6    enable unloading of most significant half
 7    enable unloading of least significant half
 8    unload most significant word of result C into floating point
          registers and/or back into Weitek chips†
 9    unload least significant word†
```

The next floating point instruction can be started at cycle 6, or anytime thereafter. More delays can be inserted if desired as long as another floating point operation is not initiated.

†Unload must follow corresponding enable by two cycles.

When running in pipeline mode, the instruction sequence for a floating point operation is as follows:

```
cycle
 0    floating point operation, most significant word of
       source A (or Weitek result) and source B are loaded
 1    floating point operation, least significant word
 2    PA (pipeline advance)
 3    PA
 4    PA
 5    PA
 6    PA
 7    PA
 8    PA  enable unloading of most significant half
 9    PA  enable unloading of least significant half
10    unload most significant word of result C into floating point
          registers and/or back into Weitek chips†
11    unload least significant word†
```

A pipeline advance is any floating point operation. If a meaningful operation can be done—great. If not, a dummy operation must be initiated. That is, the Weitek chip pipeline only proceeds when new operations are input to the chip. A floating point operation or pipeline advance can be done in parallel with the result unload.

When unloading the result, four actions are possible:

1. nothing (perhaps only the status was used),
2. the result is stored in the floating point registers, in which case the st bit is set,
3. the result is used in a chained operation in which case the as bit is set, or
4. both 2 and 3.

When enabling the unload, the Weitek chips always output data two cycles later. Thus, the unloads and stores are always coupled.

Because of the shared bus lines into the floating point registers, it is illegal to select the floating point registers as the destination of a VPBUS operation in one cycle and to store a Weitek chip result into the floating point registers in the next microinstruction cycle. (The reason for the one cycle offset is because of the one cycle delay between choosing the floating point registers as the VPBUS destination and the actual write into the floating point registers.) Results are indeterminate.

The h/l bit is used to specify the least significant address bit to the floating point registers and the most/least significant designator for the Weitek unload command. It thus defines the most and least significant words of a 32-bit floating point number. The definition is as follows:

†Unload must follow corresponding enable by two cycles.

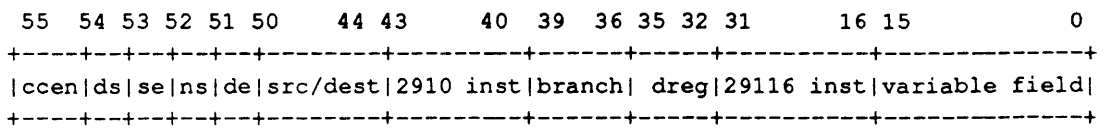
- 0: most significant word
- 1: least signficant word

Thus, 32-bit floating point numbers are aligned on even word boundaries.

When initiating a floating point operation, first the most significant word then the least significant word must be routed from the floating point registers to the Weitek chips; the h/l bit controls this procedure. If a chained operation is being performed, the Weitek result must be unload most significant first; this is also specified by the h/l bit. When storing a Weitek result, this bit controls the unload order from the Weitek chips and the load order into the floating point registers. And finally when accessing the floating point registers from the VPBUS (source or destination) this bit defines the least significant address bit into the registers.

7.2. Painting Processor Microcode

The microcode for the Painting Processor is 56 bits wide and is arranged in eleven fields.




```
ccen    ⇒   condition code enable (AM2910 control input, see AMD
                literature)
                  0: branch field select AM2910 pass condition
                  1: forces pass condition to AM2910
ds      ⇒   data source to AM2910 D input
                  0: general field selected
                  1: branch register is selected
se      ⇒   status (zero, negative, carry, overflow) update enable
                  0: enables status update
                  1: disables status update
ns      ⇒   n field select
                  0: instruction register n field selected (part of the
                     29116 inst field)
                  1: n register selected
de      ⇒   destination enable, controls use of dreg field
                  1: dreg is used for miscellaneous controls
                  0: dreg is used to select AM29116 destination
                     register in a 2-address instruction
src/dest      ⇒   selects PPBUS source and destination
                        (see below for encoding)
2910 inst     ⇒   AM2910 sequencer instruction (see AMD literature)
branch        ⇒   branch condition code select
dreg          ⇒   destination register or miscellaneous controls
29116 inst    ⇒   16-bit AM29116 instruction (see AMD literature)
general field ⇒   general (or variable) field
```

AM29116 Instruction

The 29116 inst, de, dreg, ns and se fields apply to the AM29116 microprocessor. The 16-bit 29116 instruction is described in detail in the AMD literature.

The GP supports two address operations with the AM29116. For example, the following is possible:

```
Rm <-- Rn op ACC
```

Rn is specified by the least significant 5 bits of the 29116 inst field. If de is 0, then Rm is selected by the dreg field. (If de is 1, then m equals n; that is, the source and destination are the same.) Since dreg is only four bits, Rm and Rn must be in same group of 16 registers within the AM29116, 0 to 15 or 16 to 31. If de is 1, the dreg field is used as described below in the miscellaneous controls section, and the least significant four bits of the AM29116 instruction are not altered.

NOTE *The AM29116 does not support two address instructions with immediate instructions. For an immediate instruction, de must equal 1.*

For AM29116 bit-oriented instructions (test bit, set bit, etc.), bits 12-9 of the AM29116 instruction select n, the bit to be operated on. Having this bit hard-coded in the microinstruction severely limits its usefulness. Therefore it is possible to substitute a runtime value (from the n register) for the hard-coded value. ns=0 selects the assemble time value, ns=1 selects the n register.

The bit se controls the updating of the status register containing the AM29116 flags: zero, negative, carry, and overflow. se=0 enables updating, se=1 disables updating.

The carry status bit changes its meaning slightly for subtract operations. For these operations, a logic 1 indicates no borrow and a 0 indicates borrow. This has significance when doing double precision operations.

NOTE *It is the responsibility of the microcoder to ensure that it makes sense to use the de or ns capabilities. For example, if the microinstruction is a non-register operation, setting de and enabling the dreg field into the AM29116 could create havoc.*

Miscellaneous Controls

If de=1, then the dreg field is used for the following functions:

```
0000  no operation
0001  increment VME address registers
0010  decrement VME address registers
0011  clear scratchpad-memory pointer
0100  increment scratchpad-memory pointer
0101  start-read: start Graphics Buffer read
         (see Graphics Buffer operation described below)
0110  initiate VME bus read (if VME is ready)
         (no 3-way branch option)
0111  initiate VME bus write (if VME is ready)
         (no 3-way branch option)
1000  reserved
1001  reserved
1010  reserved
1011  reserved
1100  reserved
1101  reserved
1110  initiate VME bus read (if VME is ready)
         (with 3-way branch option)
1111  initiate VME bus write (if VME is ready)
         (with 3-way branch option)
```

The 3-way branch is discussed below in the branch logic section.

Source and Destination

The source/destination field is encoded as follows. A single PPBUS operation can be done during each PP cycle.

```
          PPBUS source --> PPBUS destination

0000000   reserved - no source or destination
0000001   interprocessor flag #1 register --> AM29116
0000010   AM29116 --> status flag/LED register
0000011   AM29116 --> n register
0000100   AM29116 --> branch register
0000101   AM29116 --> scratchpad pointer
0000110   AM29116 --> interprocessor flag #2
0000111   AM29116 --> PPPROM pointer
0001000   AM29116 --> FIFO #2
0001001   AM29116 --> AM29116
0001010   AM29116 --> scratchpad memory
0001011   AM29116 --> Graphics Buffer write-data register and
                      Graphics Buffer is set busy
0001100   DO NOT USE (AM29116 --> no where)
0001101   AM29116 --> VME write-data register
0001110   AM29116 --> multiplier X operand
0001111   AM29116 --> multiplier Y operand
0010000   AM29116 --> multiplier mode register
0010001   AM29116 --> interrupt id register
0010010   AM29116 --> VME high address register
0010011   AM29116 --> VME low address register
0010100   AM29116 --> VME control register
0010101   DO NOT USE (AM29116 --> no where)
0010110   AM29116 --> Graphics Buffer high address pointer
0010111   AM29116 --> Graphics Buffer low address pointer
0011000   PPPROM --> FIFO #2
0011001   PPPROM --> AM29116
0011010   PPPROM --> scratchpad memory
0011011   PPPROM --> Graphics Buffer write-data register and
                     Graphics Buffer is set busy
0011100   DO NOT USE (PPPROM --> no where)
0011101   PPPROM --> VME write-data register
0011110   PPPROM --> multiplier X operand
0011111   PPPROM --> multiplier Y operand
0100000   DO NOT USE (no source or destination)
0100001   Graphics Buffer read-data register --> AM29116
0100010   Graphics Buffer read-data register --> VME high address register
0100011   Graphics Buffer read-data register --> VME low address register
0100100   Graphics Buffer read-data register --> branch register
0100101   Graphics Buffer read-data register --> scratchpad pointer
0100110   DO NOT USE (no source or destination)
0100111   Graphics Buffer read-data register --> PPPROM pointer
0101000   reserved
0101001   DO NOT USE (no source --> AM29116)
```

```
0101010  reserved
0101011  reserved
0101100  reserved
0101101  DO NOT USE (no source --> VME write-data register)
0101110  DO NOT USE (no source --> multiplier X operand)
0101111  DO NOT USE (no source --> multiplier Y operand)
0110000  reserved
0110001  reserved
0110010  reserved
0110011  reserved
0110100  reserved
0110101  reserved
0110110  reserved
0110111  reserved
0111000  reserved
0111001  VME read-data register --> AM29116
0111010  reserved
0111011  reserved
0111100  DO NOT USE (VME read-data register --> no where)
0111101  VME read-data register --> VME write-data register
0111110  VME read-data register --> multiplier X operand
0111111  VME read-data register --> multiplier Y operand
1000000  VME status register --> FIFO #2
1000001  VME status register --> AM29116
1000010  VME status register --> scratchpad memory
1000011  VME status register --> Graphics Buffer write-data register
                and Graphics Buffer is set busy
1000100  general field --> branch register
1000101  general field --> scratchpad pointer
1000110  general field --> interprocessor flag #2
1000111  general field --> PPPROM pointer
1001000  general field --> FIFO #2
1001001  general field --> AM29116
1001010  general field --> scratchpad memory
1001011  general field --> Graphics Buffer write-data register and
                Graphics Buffer is set busy
1001100  DO NOT USE (general field --> no where)
1001101  general field --> VME write-data register
1001110  general field --> multiplier X operand
1001111  general field --> multiplier Y operand
1010000  general field --> multiplier mode register
1010001  general field --> interrupt id register
1010010  general field --> VME high address register
1010011  general field --> VME low address register
1010100  general field --> VME control register
1010101  DO NOT USE (general field --> no where)
1010110  general field --> Graphics Buffer high address pointer
1010111  general field --> Graphics Buffer low address pointer
1011000  scratchpad memory --> FIFO #2
1011001  scratchpad memory --> AM29116
1011010  reserved
1011011  scratchpad memory --> Graphics Buffer write-data register and
                Graphics Buffer is set busy
```

```
1011100  DO NOT USE (scratchpad memory --> no where)
1011101  scratchpad memory --> VME write-data register
1011110  scratchpad memory --> multiplier X operand
1011111  scratchpad memory --> multiplier Y operand
1100000  reserved
1100001  FIFO #1 --> AM29116
1100010  FIFO #1 --> VME high address register
1100011  FIFO #1 --> VME low address register
1100100  FIFO #1 --> branch register
1100101  FIFO #1 --> scratchpad pointer
1100110  reserved
1100111  FIFO #1 --> PPPROM pointer
1101000  reserved
1101001  reserved
1101010  reserved
1101011  reserved
1101100  reserved
1101101  reserved
1101110  DO NOT USE (no source --> multiplier X operand)
1101111  DO NOT USE (no source --> multiplier Y operand)
1110000  scratchpad memory --> multiplier mode register
1110001  scratchpad memory --> interrupt id register
1110010  scratchpad memory --> VME high address register
1110011  scratchpad memory --> VME low address register
1110100  scratchpad memory --> branch register
1110101  scratchpad memory --> scratchpad pointer
1110110  scratchpad memory --> Graphics Buffer high address pointer
1110111  scratchpad memory --> Graphics Buffer low address pointer
1111000  Multiplier result --> FIFO #2
1111001  Multiplier result --> AM29116
1111010  Multiplier result --> scratchpad memory
1111011  Multiplier result --> Graphics Buffer write-data register and
                Graphics Buffer is set busy
1111100  DO NOT USE (multiplier result --> no where)
1111101  Multiplier result --> VME write-data register
1111110  Multiplier result --> multiplier X operand
1111111  Multiplier result --> multiplier Y operand
```

The result from the integer multiplier (AM29L517) is 32-bits so that the Multiplier result must be read twice to obtain the high and low order 16-bits. Each time the multiplier X and/or Y operand register is loaded, either the high or low order result is pre-enabled; (the user selects which result in the multiplier mode register). Then each time the Multiplier result is read, the enable state is toggled thus allowing access to both halves of the multiply result. An integer multiply operation takes six cycles as follows:

```
LOAD X (high or low result is enabled)
LOAD Y (high or low result is again enabled)
delay for transfer from X and Y operand
        registers into the multiplier
delay for multiply execution
READ RESULT (high or low result)
READ RESULT (low or high result).
```

FIFO #1 and FIFO #2 are the same FIFO. FIFO #1 is the VP-to-PP direction and FIFO #2 is the PP-to-VP direction.

When the AM29116 is chosen as the bus source, its internal Y bus is routed to the PPBUS. When the AM29116 is chosen as the destination, its internal D register is made transparent until the end of the cycle at which time it is latched.

When the scratchpad memory is chosen as the PPBUS destination, the actual write into the memory is executed in the next cycle. This means that the PP microcoder cannot use this value for at least one cycle. In addition because there is only one scratchpad pointer, the data read in the cycle immediately after the write is useless.

Another hardware constraint involves the interprocessor flag register. It is not possible to read this register into the AM29116 and manipulate it in a single cycle. It must first be read into the AM29116 D-latch and then manipulated by subsequent instructions.

Hardware Protection

Protection is implemented in the hardware to prevent the following operations:

- reading an empty FIFO #1,
- reading the Graphics Buffer read-data register when the Graphics Buffer is busy,
- reading the VME read-data register when the VME bus interface is busy,
- writing a full FIFO #2,
- writing the Graphics Buffer address pointers when the Graphics Buffer is busy,
- writing the Graphics Buffer write-data register when the Graphics Buffer is busy,
- writing the VME control register when the VME bus interface is busy,
- writing the VME write-data register when the VME bus interface is busy,
- reading or writing the FIFO when in the wrong direction,
- executing a start-read command when the Graphics Buffer is busy,
- initiating a VME operation (read or write) when the VME bus interface is busy.

If these operations are attempted, the hardware subsection never receives the command; that is, as far as the hardware is concerned, it is as if the instruction was never attempted. This allows the operation to be looped until successful completion. For example, the following statement will adapt to the FIFO state:

```
HERE:  MOVE Rn <-- FIFO #1; BR HERE IF FIFO #1 IS EMPTY.
```

Note that since it is unknown how many times the instruction will be attempted, no arithmetic operation other than the move from FIFO #1 should be performed. For example with the following instruction, the final value of Rn is unknown; it will be incremented once for each unsuccessful attempt and once for the successful attempt.

```
HERE:  Rn <-- Rn+1; MOVE Rn <-- FIFO #1; BR HERE IF FIFO #1 IS EMPTY.
```

FIFO #1 and FIFO #2 are actually the same "reversible" FIFO. FIFO #1 is for VP-to-PP transfers and FIFO #2 is for PP-to-VP transfers. It is therefore undefined for the PP to read FIFO #1 while the direction is PP-to-VP, and to write FIFO #2 while the direction is VP-to-PP. The branch condition codes are defined as follows:

Direction	Condition Code
VP-to-PP	FIFO #1 empty or not empty is valid
VP-to-PP	FIFO #2 is always not full
PP-to-VP	FIFO #1 is always not empty
PP-to-VP	FIFO #2 full or not full is valid

No hardware protection is necessary for writes to the VME high and low address registers because these registers are buffered.

No hardware protection is provided for the following:

- reading the PPPROM before the data word is valid,
- reading the Graphics Buffer read-data register when in fill mode,
- executing a Graphics Buffer start-read command when in fill mode,
- placing the Graphics Buffer in read/modify/write and fill modes at the same time.

Because slow PROMs are used to provide the needed size, there is a two cycle delay between loading the PPPROM pointer and having access to valid data. No hardware protect is implemented to ensure valid data; the microcode must delay at least the minimum cycles.

Branch Logic

The branch logic includes the AM2910 instruction, the branch condition select field, the ds bit, and the ccen bit. The AM2910 microsequencer is described in the AMD literature.

The branch field selects the branch condition select option as follows:

```
0000  Zero
0001  Negative
0010  Carry
0011  Overflow
0100  FIFO #2 not full
0101  FIFO #1 not empty
0110  Graphics Buffer ready
0111  VME interface ready
1000  Not zero
1001  Non-negative
1010  No carry
1011  No overflow
1100  FIFO #2 full
1101  FIFO #1 empty
1110  Graphics Buffer busy
1111  VME interface busy
```

The ds bit is used to determine whether the contents of the branch register or the general field are routed to the D input of the AM2910, possibly the address of the next microinstruction. ds=0 selects the general field; ds=1 selects the contents of the branch register.

Since there is not a "1" option for the branch conditions, a method to execute unconditional branches must be provided: the ccen bit. If ccen=1, then a pass condition is forced in the AM2910. If ccen=0, the pass condition is conditional on the cc bit selected from the above 16 options. Unlike the VP, a fail condition cannot be forced in a single cycle. To force a fail condition in the PP, a known status must be created in the condition codes (for example, zero) and a ''fail'' instruction executed (for example, jump on not zero). Further details are available in the AMD 2910 literature.

The status bits are updated at the end of each cycle (conditional update for the zero, negative, carry, and overflow flags), and can be used at the beginning of the next cycle for conditional branches. For example, to determine if the AM29116 Rn and accumulator are equal, the code would be as follows.

```
NO DESTINATION <-- Rn - ACC, SE
NOP; BR TO EQUAL IF ZERO STATUS FLAG IS SET
```

When initiating reads or writes to the VME bus, it is possible to invoke a 3-way branch. The microinstruction word must meet the following format:

```
ccen=0    ds=1    de=1    dreg=111x
```

If the VME interface is busy, then the next program counter is contained in the general field. If the VME interface is not busy, then this instruction works like a "normal" branch: if the branch condition selected (for example, negative) is true, a branch to the address in the branch register is executed; otherwise, the next sequential instruction is executed.

Count Hardware

The miscellaneous controls are used to enable the counting of the scratchpad pointer and the VME address register. Because the VME high and low address registers are buffered, they can be counted at anytime. However restrictions exist on count enables to the scratchpad memory. Counts enabled on scratchpad read cycles or non-scratchpad-access cycles are executed in the current cycle. But because the write into the scratchpad memory is delayed one cycle, the count (if enabled) is also delayed one cycle. This means that a cycle immediately after the write cycle with count enabled should not be a scratchpad read with scratchpad count enabled or a non-scratchpad-access cycle with scratchpad count enabled.

Increment hardware in the Graphics Buffer logic is described in the next section.

General Field

The general field is a way of specifying a value in the microcode and using it in one of several places. For example, it could be an address pointing to the possible next microinstruction (see ds bit above). It could be a constant loaded into an AM29116 register. It could be an address pointing to a predefined constant in the scratchpad memory.

The general field is routed onto the PPBUS bus if selected by the source field and/or routed to the D input of the AM2910 microsequencer and bank select logic if selected by the ds bit (also see branch logic section above).

Graphics Buffer Board Memory (Graphics Buffer)

In order to make the dynamic random access memory (DRAM) accesses as fast as possible, special modes of operation are designed into the GP. The DRAM can be accessed in either of two modes: Normal or Read-Modify-Write (RMW).

In *normal mode*, the DRAM acts like a linear array with hardware assist for sequential accesses. If the low address pointer is loaded coincident with start-read (see miscellaneous control field), a read is performed at the specified location and the fetched data word loaded into the read-data register. (A load of the high address pointer or the low address pointer with no start-read merely loads the respective address pointer; there are no side-effects.) When the read-data register is read (selected as the PPBUS source), its contents are routed to the selected destination. If a start-read is specified coincident with this read or anytime after, the Graphics Buffer address pointer is incremented, another memory read is initiated, and when the fetch is completed, the data word is loaded into the read-data register. If the write-data register is loaded (specified as the PPBUS destination), then a memory write is initiated and value just written into the write-data register is written into the Graphics Buffer. After the completion of this write, the address pointer is incremented.

Fill mode is a normal sub-mode and must only be used with writes to the write data register. While in this mode, writes to the Graphics Buffer write-data register cause 4 memory locations to be written with the value just written into the write-data register. After the DRAM write completes, the Graphics Buffer address pointer is incremented by 4 so that the next 4 memory locations can be written. The 4 locations written are selected by the Graphics Buffer address pointer while ignoring the least significant two bits. Thus a fill mode write starts on a modulo 4 boundary. (The least significant two address bits are don't care in this mode and their final value after a series of fill mode writes is indeterminate.)

When using the DRAM as, for example, a graphics buffer, the *RMW mode* is invoked. When the low address pointer is loaded coincident with a start-read, a RMW cycle is initiated; the fetched data word is loaded into the read-data register, but the RMW cycle remains active. Reading the read-data register (selecting it as the PPBUS source) causes the register to be routed to the selected destination but does not affect the active RMW cycle. Writing the write-data register (selecting it as the PPBUS destination) causes the value just loaded into write-data register to be written into the DRAM thus ending the RMW cycle. When the DRAM write completes, the pointer is auto-incremented, a new RMW cycle is initiated, and the fetched data word is loaded into the read-data register.

NOTE *After entering RMW mode, a start-read must precede the first write. (Unknown calamities could otherwise occur.) This boundary condition occurs because of the special hardware which executes the read/modify/write cycles.*

In RMW mode, a new operation is required. After reading the read-data register, it may be determined that the data word already at that location in the DRAM is not to be changed. In this case the start-read is used. Using the start-read instead of the write-data register causes the active RMW to be terminated with no write. The address pointer is auto-incremented, a new RMW cycle is initiated, and the fetched data word read is loaded into the read-data register. To exit RMW mode, change the mode with a write to the high address pointer, and the active RMW is terminated with no write.

CAUTION **When in RMW mode, the microcoder must be aware of two restrictions:**

1. **When entering RMW mode from normal mode, the microcoder must delay at least four (4) instruction cycles before loading the low address pointer coincident with a start-read. (This allows an active DRAM refresh cycle to complete.)**
2. **Once the low address pointer is initialized, it cannot be loaded again without first exiting RMW mode. Loading a new address while in RMW mode could change the active bank, thus creating a runt RAS pulse to the newly-selected memory bank. The high address pointer may be loaded to change the mode (to normal).**

The Graphics Buffer ready flag is used to determine DRAM status. If the flag is set, then writes to the address pointers, reads of the data read register, and writes to the write-data register are allowed. If the flag is not set, then the DRAM is busy—a read or write is active—and these operations are inhibited (see above in Hardware Protection section). The Graphics Buffer ready flag is reset to not ready by a start-read command or a load of the Graphics Buffer write-data register. The flag is set to ready automatically by the hardware when a memory read finishes (in both normal and RMW modes) or when a memory write finishes (in normal mode only).

Memory refresh is handled by the hardware, and the firmware adapts to the longer access times during a refresh by testing the Graphics Buffer ready flag.

The Graphics Buffer operations are summarized in the following chart.

Operation	Normal Mode		RMW mode	
	no start-read	with start-read	no start-read	with start-read
load high address pointer	loads high address ptr & mode	DO NOT USE	loads high address ptr & mode	DO NOT USE
load low address pointer	loads low address ptr	loads low address ptr & starts read	loads low address ptr DO NOT USE	loads low address ptr & starts RMW
read GBuffer read-data register	PPBUS source	PPBUS source & increments ptr & starts read	PPBUS source	PPBUS source & terminates active RMW & increments ptr & starts RMW
load GBuffer write-data register	PPBUS dest. & does write & increments ptr	PPBUS dest. & does write & increments ptr	PPBUS dest. & does write (ends RMW) & increments ptr & starts RMW	PPBUS dest. & does write (ends RMW) & increments ptr & starts RMW
start-read command		increments ptr & starts read		terminates active RMW & increments ptr & starts RMW

NOTE *Start-read is ignored if executing a load the Graphics Buffer write-data register. Also note that a change mode can be done at anytime. For example, exiting RMW mode will terminate the active RMW cycle.*

A

Graphics Processor/Graphics Buffer Specifications

Graphics Processor/Graphics Buffer Specifications

PERFORMANCE

Shared Memory
- Dual-ported
- 16K-by-16 bits
- 360 nsec max cycle time from VME bus
- 1 cycle access time from AM29116

Floating point
- Separate Multiplier and ALU
- 4.16 Mflop maximum performance
- 32-bit, IEEE standard format

AM29116 (2)
- General-purpose ALU
- 120 nsec cycle time
- 8K (max) by 56-bit writable microstore
- Microstore is software-partitionable

Graphics Buffer memory
- 2 Mbyte (1 Mword)
- Overlapped accesses

VME Interface
- 16-bit data
- 24-bit address
- Interrupt capability

Graphics Performance
- 480 nsec/pixel vector draw rate
  (actual performance limited by Color board)
- 25K 3D-vectors/second (about .5 inch vectors)
- 26M pixels/second area fill rate
- 8.9 usec/point 3D coordinate transform rate
- 960 nsec/pixel (worst case) shading rate with
  Gouraud or flat shading and hidden surface elimination
- 1K triangles/second (about .25 inch per side triangles) with
  Gouraud or flat shading and hidden surface elimination

VME BOARD SPECIFICATION

```
MASTER DATA TRANSFER OPTIONS
        A24:D16
        ROR arbitration, level 3 request
        TOUT = 5 usec
        Sequential access not supported

SLAVE DATA TRANSFER OPTIONS
        A24:D16 for shared memory access
        A24:D16 (word access only) for microstore interface access
        Sequential access not supported

INTERRUPTER OPTIONS
        Level 4 interrupt

RESET OPTIONS
        ACFAIL    not used
        SYSRESET  resets board
        SYSFAIL   not used

ENVIRONMENTAL OPTIONS
        OPERATING TEMPERATURE: 0 to 70 degrees C
        MAXIMUM OPERATING HUMIDITY: 90%

POWER OPTIONS
        Graphics Processor board: 20 A MAX (18 A typ) at + 5 VDC
        Graphics Buffer board: 4 A MAX (3 A typ) at + 5 VDC

PHYSICAL CONFIGURATION OPTIONS
        Triple wide, extended height (400mm by 366.67 mm) VME boards
```

Revision History

Revision	Date	Comments
50	1 July 1985	First release of this Hardware Reference Manual.

Index

READER COMMENT SHEET

Dear Customer,
We who work at Sun Microsystems wish to provide the best possible documentation for our products. To this end, we solicit your comments on this manual. We would appreciate your telling us about errors in the content of the manual, and about any material which you feel should be there but isn't.

Typographical Errors:

Please list typographical errors by page number and actual text of the error.

Technical Errors:

Please list errors of fact by page number and actual text of the error.

Content:

Please list errors of fact by page number and actual text of the error.

Content:

Did this guide meet your needs? If not, please indicate what you think should be added or deleted in order to do so. Please comment on any material which you feel should be present but is not. Is there material which is in other manuals, but would be more convenient if it were in this manual?

Layout and Style:

Did you find the organization of this guide useful? If not, how would you rearrange things? Do you find the style of this manual pleasing or irritating? What would you like to see different?