

CHAPTER SEVEN: SECTION F

FLOATING POINT ACCELERATOR (FPA)

OVERVIEW

The Sun Floating Point Accelerator (FPA) is a single VME style card, which can be added to the Sun 3 workstation to enhance floating point operations. By configuring the FPA card with the Sun 3 68020/68881 based CPU card, floating point performance is increased allowing greater application to development areas in CAD, modeling, element analysis, simulation and other scientific, engineering or computational graphics endeavors.

For correct operation of the FPA, the CPU board must have an operational floating point co-processor (68881 FPP) running at 16.67 Mhz. Failure of the FPP will cause the FPA to be ignored at boot time.

The FPA card features the standard Sun triple height VME Eurocard, using state-of-the-art computational logic for floating point operations, implemented with Weitek 1164 and 1165 devices (64-bit IEEE floating point multiplier and ALU, respectively). In addition, the FPA card offers a two-level instruction pipeline, up to 32 user contexts, allows single or double precision IEEE 754 formats and operations, and provides on-board logic for high performance interface control.

7F.1 FLOATING POINT OPERATIONS

For executing floating point operations, Sun 3 code (used exclusively on 68020 based Sun workstations) translates the high-level (coded) instructions into assembly code which is used to perform the floating point calculations. A Sun 3 compiler supports these operations and allows the user four options to actually execute the instructions. These four options include software, the 68881, the FPA, or switched code.

The first method, executing floating point in software, is actually the slowest means, but will run on any Sun 3 system.

The second method executes floating point using the on-board 68881 floating point coprocessor. This method is faster than software, but also requires that the optional 68881 chip is resident on the host processor board.

Method three executes floating point by using the FPA board. This is the quickest method the user can select, however, this requires that both the 68881 and the FPA card are available for the operation to transact successfully. This section then, will detail the architecture of the FPA card.

The fourth and final method utilizes a switching code. This method relies on having available libraries (under Sun 3 code) run under any one of the before mentioned methods. This method will run quicker if the FPA is installed.

7F.2 FPA BUS INTERFACE

The FPA is configured to the Sun 3 host processor using a subset of the Sun 3 VME backplane, referred to as the FPA bus. In a typical configuration, the Sun 3 host processor board is set as the VME bus master, while the FPA is set as a slave on the VME bus. Thereby, the FPA will only respond to instructions (or operands) sent by the host processor to its instruction pipeline. The FPA bus is an asynchronous interface between the two boards.

When the FPA responds to instructions from the host processor (via the FPA bus) it causes the Weitek chip set (1164 and 1165) to execute the requested floating point operation. When the results are written to RAM, the host processor can read them over the FPA bus. The FPA then, will never arbitrate to become a bus master, and will therefore not initiate any transactions or operations with any other device.

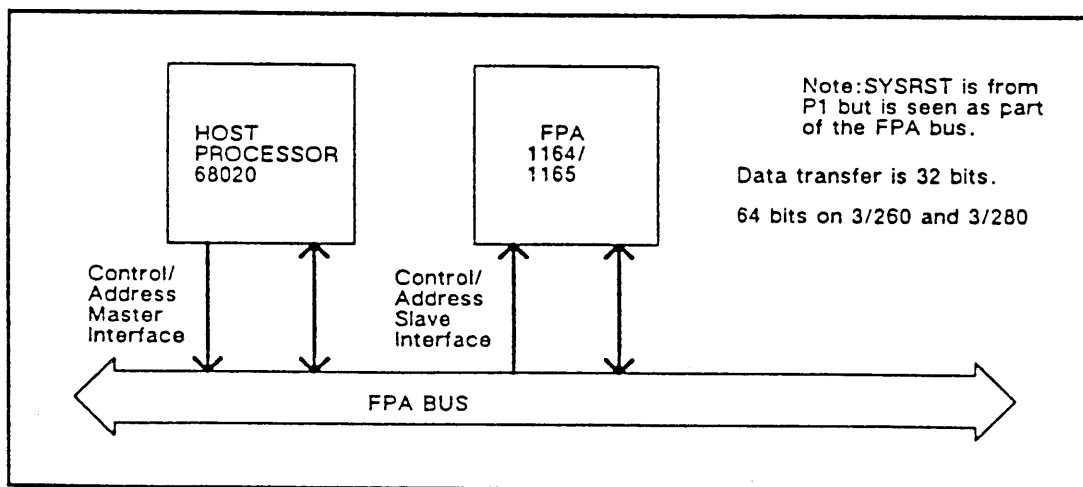


FIGURE 7F-1: FPA BUS INTERFACE TO THE HOST PROCESSOR

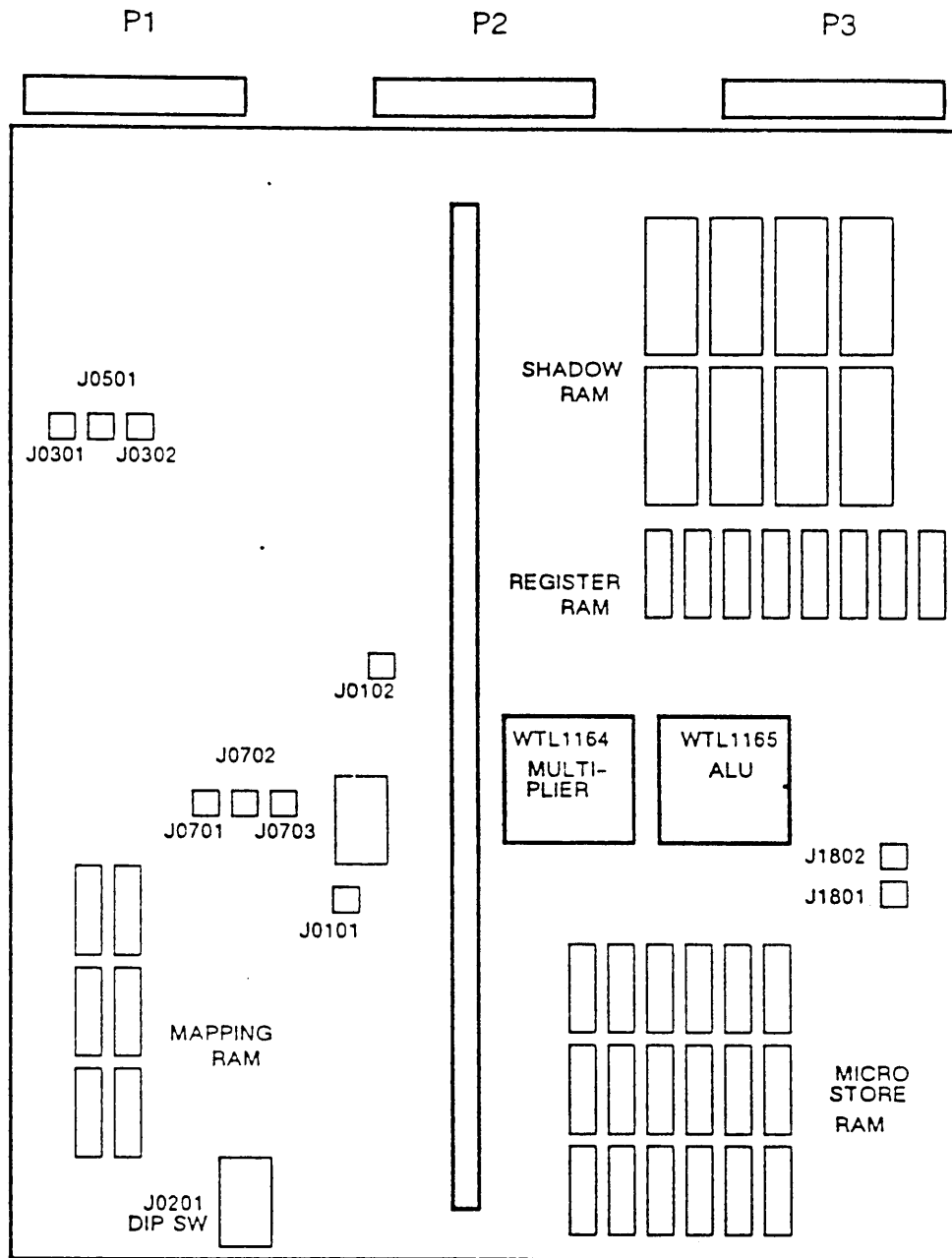


FIGURE 7F-3: FPA BOARD LAYOUT

All data paths to the FPA will be 32-bits wide, and all registers will be aligned on word or long word boundaries. Sizing will be determined by the dynamic sizing mechanism of the 68020, however, 32-bit accesses will be the only type accepted by the FPA. Figure 7F-2 illustrates the address and data path to the FPA.

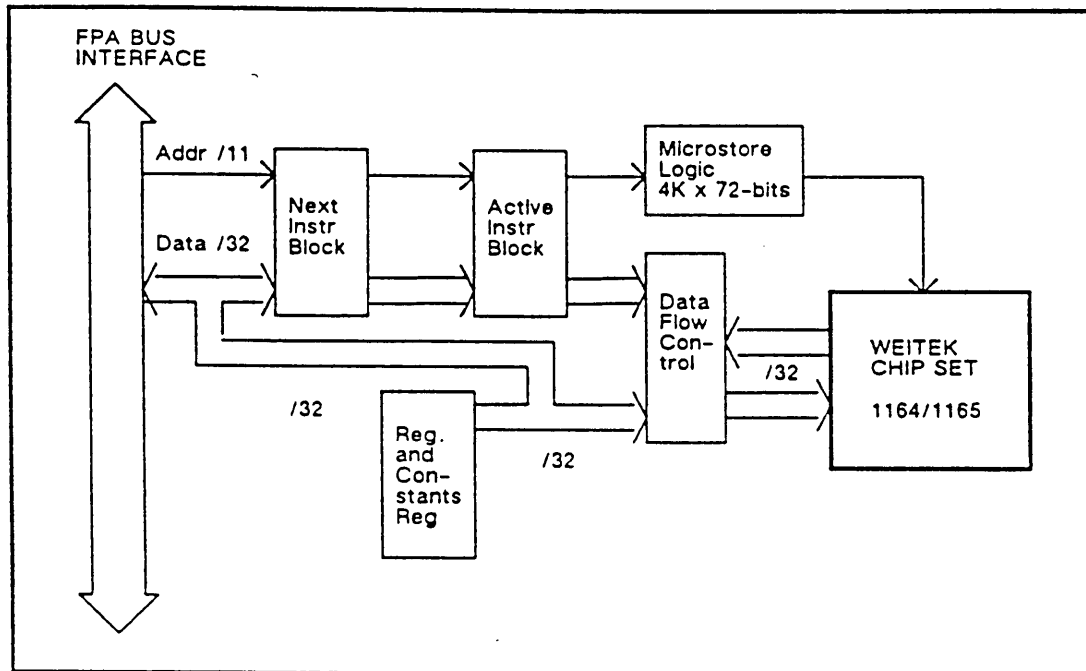


FIGURE 7F-2: FPA BLOCK DIAGRAM

The following Figures illustrate the layout, configuration and installation of the FPA board into an existing Sun 3 workstation:

7F.3 FPA ADDRESSING AND DATA PATH

The FPA is configured within the private memory bus (slots 1 through 6) and is located in virtual address space. The base address of the FPA is found at 0xE0000000.

When the most significant address bits (A31-28) are set to 0xE, and the system enable register (on-board the CPU card) indicate that the enable FPA bit has been set, transactions will bypass the MMU and go directly to the FPA card. The FPA will decode lower address lines to obtain details about the transaction to be performed. These transactions will be WRITE or READ operations.

Resulting transfers between the CPU and the FPA are determined by the CPU cycles. Under WRITE accesses to the FPA (an instruction or operand) the address and the operand are latched. An acknowledge will be generated by the FPA and sent immediately to the host processor.

During a READ access, the requested data is fetched over the FPA bus to the host processor. READ accesses can be simple flow-through operations (status requests) where no microcode is executed, or microcode is executed to route the data (results) to the FPA bus.

Since the FPA is operating outside of the MMU (in virtual address space), the MMU segment maps and page maps will provide no protection during the READ/WRITE accesses. To compensate for this, protected accesses are provided at two levels.

The first level of protection comes from the enable FPA bit, which is set in the system enable register during accesses. This setting will prevent unauthorized use of the device by users which have not been granted one of 32 FPA user contexts.

The next level will be generated by signals on the FPA bus which are set to differentiate between supervisor and user accesses. These context bits will prevent users from accessing each others result registers (resulting data from the operation is stored in these registers), and will prevent users from overwriting the instruction register (microstore), the constants to be used for the operation, or the mapping RAM.

JUMPER	PINS	GRID LOCAL	FUNCTION	INSTALLED
J0101	1-2	F10	50 MHz clk	1-2 IN
J0301	1-2 3-4	P2	Shadow rd ack/nack	1-2 IN(3/160) 3-4 IN(3/260)
J0302	1-2 3-4	P3	FPA access pending	1-2 IN(3/160) 3-4 IN(3/260)
J0501	1-2 3-4	P2	Asynch cntrl for 1st pipe stage	1-2 IN(3/160) 3-4 IN(3/260)
J0701	1-2 3-4	H7	Current version IMASK	1-2 IN
J0702	1-2 3-4	H8	Current version IMASK	1-2 IN
J0703	1-2 3-4	H8	Current version IMASK	1-2 IN
J1801	1-2 3-4	F32	4 VDC for WTL1164 (Multiplier) 5 VDC for WTL1164 (Multiplier)	1-2 OUT 3-4 IN
J1802	1-2 3-4	H32	4 VDC for WTL1165 (ALU) 5 VDC for WTL1165 (ALU)	1-2 OUT 3-4 IN

TABLE 7F-1: FPA BOARD JUMPERS

SECTION							
1	2	3	4	5	6	7	8
OFF	ON	OFF	ON	OFF	ON	ON	ON
NOTE: For timeout for retry, and, count of consecutive retries							

TABLE 7F-2: DEFAULT DIP SWITCH SETTING

7F.4 ADDING THE FPA TO AN EXISTING SYSTEM

Please refer to Appendix B of this manual for complete instructions on adding an FPA to an existing system. Please note that you must be running SunOs revision 3.1 or above.

7F.5 FPA TESTS

```

/usr/etc/fpa/fparel -v
/usr/etc/mc68881version (for the coprocessor)

```

ECO # 3634

PG 9 OF



Sun Floating Point Accelerator Board Configuration Procedures


Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: Part Number 813-2012-04
Revision: A of 3 March 1988

VMEbus is a trademark of Motorola, Inc.

Sun Microsystems and Sun Workstation are registered trademarks of Sun Microsystems, Incorporated.

Sun is a trademark of Sun Microsystems, Incorporated.

The Sun logo  is a registered trademark of Sun Microsystems, Inc.

Multibus is a trademark of Intel Corporation.

CAUTION 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.

Copyright © 1987, 1988 by Sun Microsystems, Inc.

This publication is protected by Federal Copyright Law, with all rights reserved. No part of this publication may be reproduced, stored in a retrieval system, translated, transcribed, or transmitted, in any form, or by any means manual, electric, electronic, electro-magnetic, mechanical, chemical, optical, or otherwise, without prior explicit written permission from Sun Microsystems.

Springfingers are metal strips that are installed between the edge of the PC board and the outer panel to reduce RFI emissions. Serrated metal "fingers" protrude from either side of the strip.

Installation of a board WITHOUT springfingers may affect RFI emissions and may therefore affect FCC compliance. Sun will no longer be responsible for FCC compliance if non-springfingere boards are added to a system originally shipped WITH springfingers and FCC approval.

If a board WITH springfingers is installed next to a board WITHOUT springfingers, the insulator shield outside of the fingers MUST be present to prevent possible shorting of component leads to the springfingers.

If a logic enclosure contains boards WITH and WITHOUT springfingers, use the following guidelines:

- Before removing a board WITHOUT springfingers, remove the board to the left of it (or below it for top models) if that board is equipped WITH springfingers and an outer insulator shield.
- To replace any filler panel equipped WITH springfingers, pull out the air restrictor panel far enough to allow the springfingers to lay against the panel. Push both units into place simultaneously and fasten with the appropriate fasteners. This procedure makes replacement of the filler panels easier and reduces the chance of damage to the springfingers.
- Always install a board WITHOUT springfingers first, and then replace the board WITH springfingers and insulator shield in the slot to the left of it (or below it).

If a board WITH springfingers is installed next to a board or filler panel also equipped WITH springfingers, the outside insulator shields should be removed.

Ensure that the insulator strip between the inner side of the springfingers and the PC board is intact at all times.

When removing and replacing boards with springfingers, check the condition of the insulator strip/shield and replace if damaged.

Call 800-USA-4SUN with questions or for information on how to obtain additional insulator strips or shields.

Printed circuit boards contain components sensitive to damage from electrostatic discharge that may occur, for example, when you walk across a carpet and then touch the board. Before handling a board, place your hand on a conductive surface that is grounded to a common earth ground, (such as the metal screw on the AC wall receptacle) to discharge any static electricity from your body.

Sun Floating Point Accelerator Board General Description

The Sun Floating Point Accelerator (FPA) Board is a single-board option for Sun-3/1XX and -3/2XX workstations. The FPA board improves floating point performance, by an average factor of 3, over that provided by the standard Sun floating point logic. The illustration below provides the physical locations of the various FPA board configuration jumpers.

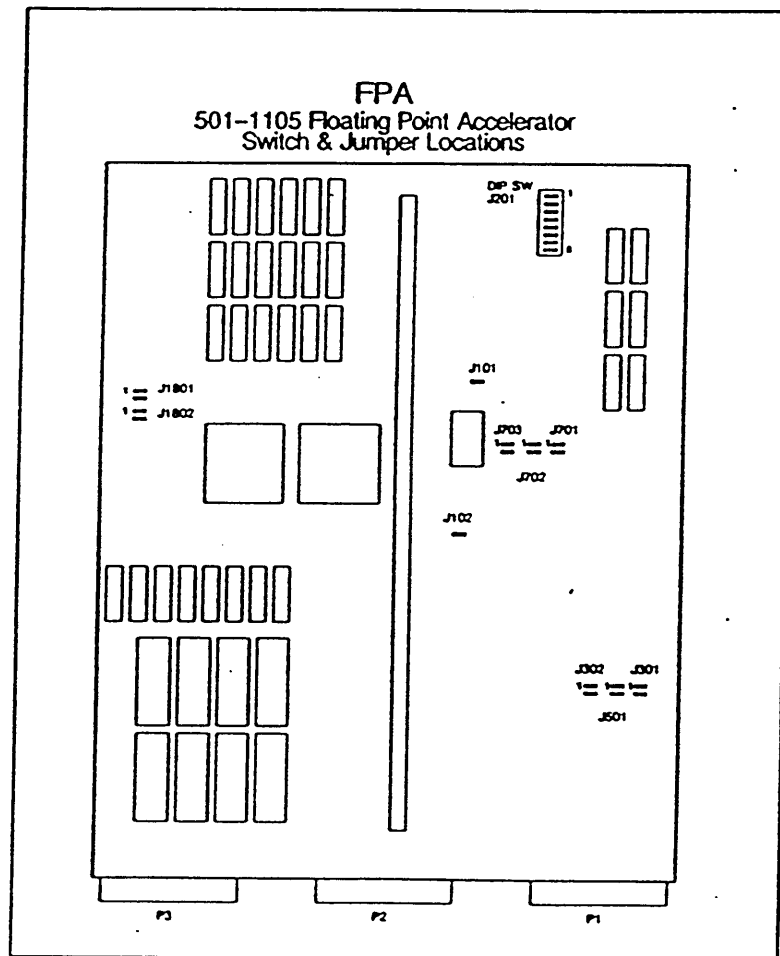


Figure 1 FPA Board Jumper Locations

Label	Pins	In/Out	Description
J0301	1-2 3-4	In	Shadow read ack/nack Out for Sun-3/100's, In for Sun-3/200's*
J0501	1-2 3-4	In	Asynch cntrl for first pipe stage Out for Sun-3/100's, In for Sun-3/200's*
J0302	1-2 3-4	In	FPA access pending Out for Sun-3/100's, In for Sun-3/200's*
J0701	1-2 3-4	In Out	Revision level
J0702	1-2 3-4	In Out	Revision level
J0703	1-2 3-4	In Out	Revision level
J1801	1-2 3-4	Out In	4VDC for WTL1164 (Multiplier) 5VDC for WTL1164 (Multiplier)
J1802	1-2 3-4	Out In	4VDC for WTL1165 (ALU) 5VDC for WTL1165 (ALU)
J0101	1-2	In	50 MHz clock enable

* For 501-1100 and 501-1206, pins 1-2 are out and pins 3-4 are in.

Label	Sw	On/Off	Description
J201	1	On	Bus timeout interval
	2	On	
	3	Off	
	4	On	
	5	Off	
	6	On	
	7	On	
	8	On	

ECO # 3634

PG 13 OF

ECO # 3634

PG 14 OF

Revision His

Revision	Date	Comments
01-1	31 Jan 1986	First release of this configuration procedure.
02-A	6 June 1986	Second release of this configuration procedure.
03-A	11 Nov 1987	Revision to include Sun-3/2xx configuration data.
04-A	2 March 1988	Revision to include J201 DIP switch settings.

Sun™ Floating Point Accelerator User's Manual

Part Number: 800-1378-03
Rev: A of June 2, 1986

Credits and Trademarks

Sun Microsystems, SunStation and Sun Workstation are registered trademarks of Sun Microsystems, Incorporated. **Sun3** is a trademark of Sun Microsystems, Incorporated.

VMEbus is a trademark of Motorola Corporation.

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.

Copyright © 1986 by Sun Microsystems, Inc.

This publication is protected by Federal Copyright Law, with all rights reserved. No part of this publication may be reproduced, stored in a retrieval system, translated, transcribed, or transmitted, in any form, or by any means manual, electric, electronic, electro-magnetic, mechanical, chemical, optical, or otherwise, without prior explicit written permission from Sun Microsystems.

Contents

Chapter 1 Introduction	3
1.1. Purpose	3
1.2. Using this Manual	3
Use of Fonts	3
1.3. Glossary	4
1.4. References	4
Chapter 2 Architecture	7
2.1. Code Generator	7
2.2. Functional Groupings	7
2.3. Operation Overview	8
2.4. System Interface	8
Addressing	8
Protection	9
Weitek Errors	9
FPA Errors	9
Timing and Handling	10
Chapter 3 Machine Level Code	17
3.1. Register Accesses	17
Register RAM Accesses	17
Shadow RAM Accesses	18
Load Pointer Accesses	18
Control Register Accesses	19

Diagnostic Accesses	19
3.2. Instructions	19
Single Precision Short	19
Double Precision Short	20
Extended Instructions	22
Command Register	23
Chapter 4 Assembly Language Programming	29
4.1. Definitions	29
Instruction Notation	29
FPA Registers	30
Operand Types	30
4.2. Instructions	30
Two Operand Instructions	31
Three Operand Instructions	31
Four Operand Instructions	32
Multiply-Add	32
Multiply-Subtract	33
Multiply-Reverse Subtract	34
Add-Multiply	34
Subtract-Multiply	35
Other Operations	35
Floating Point Compares	36
FPA Errors Messages	36
4.3. Instruction Set Summary	37
Chapter 5 Command Translations	45
5.1. Register RAM Accesses	45
5.2. Single Precision Short	46
5.3. Double Precision Short	47
5.4. Extended	47
5.5. Shadow RAM Read	48
5.6. Command Register	49

Chapter 6 Control Registers	53
IERR	53
IMASK	54
STATE Register	54
LOAD_PTR	54
LD_RAM	55
Pipe Access Registers	55
MODE Register	55
WSTATUS	56
READ_REG	57
Register and Microcode Address	57
PIPE_STATUS	57
HARD_CLEAR PIPE	58
CLEAR_PIPE	58
Register List	58
 Appendix A Addresses in Constant RAM	 63

Tables

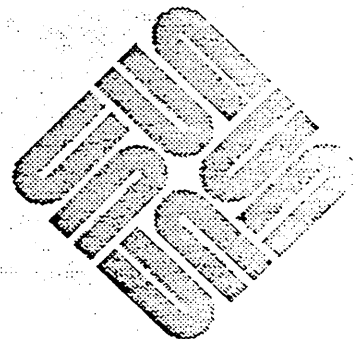
Table 4-1 Other Operations	36
Table 4-2 Instruction Set Summary	37
Table 6-1 Control Registers	59
Table A-1 Addresses in Constant RAM	63

Figures

Figure 2-1 System Block Diagram	13
Figure 2-2 FPA Addressing	14

Introduction

Introduction	3
1.1. Purpose	3
1.2. Using this Manual	3
Use of Fonts	3
1.3. Glossary	4
1.4. References	4



Introduction

This manual describes how to use the Sun-3 Floating Point Accelerator (FPA) board, a tightly coupled optional peripheral which may be installed in some Sun-3 systems.

The floating point accelerator accepts instructions from the FPA bus and uses a Weitek floating point chip set to execute floating point operations. It routes operands to the Weitek chips either from the FPA bus or from on-board registers and it places results in these registers for access by the host processor. The host processor sees the floating point accelerator as a location where it can leave instructions to perform floating point calculations, and come back later to obtain the results.

1.1. Purpose

This manual provides information to help you use the FPA board. It describes the basic functionality of the board, describes how to use the assembly language instructions provided by Sun, and provides information about the board's internal architecture, to help you create your own assembly language instructions.

1.2. Using this Manual

This manual is mostly for assembly language programmers, or persons who are going to create assembly language for the FPA.

Use of Fonts

In this manual, we use fonts to make things a little clearer. The most common fonts are Roman, typewriter, *italic*, and bold. We use them as follows:

Roman

Roman font is the standard for normal text, just as it appears here.

Typewriter

Typewriter font is mostly used for information in displays.

Typewriter Bold

Typewriter bold font represents something that you must type verbatim into the computer. This usually appears together with typewriter font; the computer output appears in typewriter, and what you must type appears in typewriter bold.

Italic

Italic font is either used for notes, or it represents a variable for which you or the computer must substitute some real value. For example:

This field contains a pointer to register *nn*

Bold

Bold font indicates that something deserves more attention than the surrounding text.

1.3. Glossary

In order to avoid confusion, this manual uses standard definitions for some common words. These are:

Access

The word **access** describes one read or write of 32-bits of data by the host processor to the FPA board over the FPA bus.

Host Processor

The host processor is the system CPU .

Instructions

In most of this manual, the word "instruction" describes the requests the host processor sends over the FPA bus to the FPA. One instruction may consist of one or two accesses.

In discussions of the assembly language, the word "instruction" describes a line of assembly language code.

Operation

In the chapter on assembly language, the word **operation** describes the action initiated by a single assembly language statement.

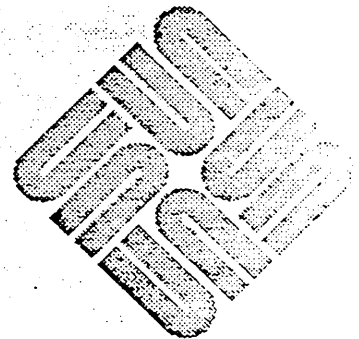
1.4. References

See the following for additional information:

- Weitek 1164/1165 Data Sheets
- Assembly Language Reference Manual for the Sun Workstation (800-1372)
- Motorola MC68020 User's Manual

Architecture

Architecture	7
2.1. Code Generator	7
2.2. Functional Groupings	7
2.3. Operation Overview	8
2.4. System Interface	8
Addressing	8
Protection	9
Weitek Errors	9
FPA Errors	9
Timing and Handling	10



Architecture

This section describes the architecture of the FPA board, and provides a block diagram (Figure 2-1).

2.1. Code Generator

The Sun-3 code generator translates high-level code into assembly code which performs floating point calculations. The Sun-3 compilers support options which provide the user a choice of 4 ways to execute floating point instructions:

Execute Floating Point in Software

This is the slowest method, but the compiled code will run on any Sun-3 machine.

Execute Floating Point using the 68881

This option runs faster than the software option, but it requires the machine to have a 68881 coprocessor installed. Programs compiled using this method do not run on machines without a 68881.

Execute Floating Point using the FPA

This is the fastest method, but it only runs on machines having an FPA and a 68881 installed.

Switched Code

This option generates code that, at run time, selects and uses the fastest hardware installed. Code generated using this option runs on all Sun-3 machines, and it runs fast if the FPA is installed. However, its object files are larger and substantially slower than those generated for the explicit hardware options.

2.2. Functional Groupings

Figure 2-1 shows the major FPA components. These are:

- 1 Instruction and data pipeline — Receives addresses and data from the FPA bus and routes them onto the board. Holds the current instruction and one pending instruction.
- 2 Register RAM and supporting circuits — Holds operands, results from the Weitek chips, and mathematical constants.
- 3 Data Multiplexer — Acts as a switch to move data between the register RAM and the Weitek chips, and in from the data portion of the instruction pipe.

- 4 The Weitek chip set — Consists of a Weitek 1164 (multiplier) and a Weitek 1165 (ALU); these actually perform the floating point operations.
- 5 Micromachine — Provides control signals for the other parts of the board and stores status signals from other parts of the board.

2.3. Operation Overview

The FPA board accepts instructions from the FPA bus, and causes the Weitek chips to perform the required floating point operations. Then it places the results in register RAM where the host processor can read them.

From the FPA bus, instructions enter the pipeline. It has storage capability for an active instruction and a next instruction, so it can receive a second instruction before it finishes the first.

An on-board micromachine controls the sequence of events required to execute an FPA instruction. Instructions enter the pipeline and are executed by the micromachine.

Register RAM is the FPA's central RAM space. It provides temporary storage for operands and results, and provides semi-permanent storage for mathematical constants such as π . The semi-permanent portion of register RAM is sometimes referred to as constant RAM.

The register portion of the register RAM is divided into 32 contexts, each with 32 registers of 64 bits.

The constants are universal; new constants may be added in future releases, but existing constants will never move and will never be deleted. Thus the value for π will always remain at the same address.

Shadow RAM always contains an image of the lower 8 registers in the current context (FPR0-FPR7); it is designed to be read quickly by the host processor. On context changes, an instruction must be sent to the FPA to update these registers. Whenever possible, the host processor should read its results from the shadow RAM.

Once the Weitek chips receive operands and instructions, they perform the operation as requested and output the result to the data flow control block, which places them in the register RAM. The host processor specifies where the results are to be placed; it must then read this address or its shadow RAM equivalent to obtain the result.

2.4. System Interface

The FPA is configured as a tightly coupled peripheral to the host processor. It sits on the system backplane, and uses a common subset of all the Sun-3 bus types. This common subset is called the FPA bus.

Addressing

The FPA addressing appears in Figure 2-2.

The FPA decodes address bits A02 - A12 to obtain details about the command. These identify the operation to be performed and provide other details.

If the access is a read, the requested value is fetched and returned to the processor. If the access is a write and the instruction pipe is not full, the FPA latches the address and operand and returns an acknowledge to the host processor

immediately.

Protection

Because the FPA resides in virtual address space, and does not use the MMU, it requires other means to keep non-FPA users from accessing the FPA, and to keep FPA users from accessing each other's registers. The EN.FPA bit in the host processor's system enable register keeps unauthorized users from accessing the FPA. The FPA state register keeps track of which user is running and what each user is allowed to do. It is described in the chapter "Control Registers" later in this manual.

Weitek Errors

FPA errors fall into two broad categories: Weitek generated errors and FPA generated errors.

The Weitek generated errors include all IEEE 754 errors, and any errors specific to the Weitek chips. These generate a bus error, set the WERR indicator bit, and get their status reported in the Weitek status register (WSTATUS) at the end of each pipeline instruction.

Only instructions which execute through the microcode produce Weitek errors. A bit in the interrupt mask register (IMASK) can be used to mask inexact errors. When this bit is ON, Weitek inexact errors generate a bus error; when it is OFF, the error is masked. The fact that the result was inexact is still reported in WSTATUS, but the indicator bit (WERR) is not set and no bus errors are generated.

For more on the WSTATUS and IMASK registers, and the WERR bit, see the chapter "Control Registers".

FPA Errors

Other errors, caused by conditions on the FPA board, are generated immediately. The access which generated the error receives a negative acknowledge, and the cause is recorded in the IERR register.

The types of errors recorded in the IERR register are:

- Non 32-bit access.
- Protection violation — a) user writes to supervisor space, b) attempt to write to register RAM without register RAM access enable bit set, c) attempt to access map RAM or microstore RAM without load enable bit set, or d) attempt to access illegal address.
- Illegal access — Trying to write a read-only address or read a write-only address.
- Illegal microcode execution — Attempt to execute using microcode when load enable bit is set.
- Hung pipe — Pipe access, shadow access or control access. Indicates Weitek error.
- 256th retry
- Access to illegal control register address.

The IERR bits that identify these errors are described in the chapter "Control Registers".

Timing and Handling

The handling and timing of Weitek errors depends on the type of error and the type of access. The access is transparent if the FPA does not need to advance the pipe before handling the error. Errors occurring during transparent accesses receive group 1 handling. The access is non-transparent if the FPA does need to advance the pipe before handling the error; errors occurring during non-transparent accesses receive group 2 handling.

The following sequence of operations illustrates this difference (assume the pipe is empty when it starts):

```
reg1 <-- reg2 / reg3
```

Because the pipe is cleared there is room for this instruction. (Group 1 — transparent access)

```
reg2 <-- reg1 + op
```

Even though this instruction requires the updated *reg1* from the previous instruction, there is room in the pipe, so this access can be accepted. This is true of both accesses if the instruction is composed of two accesses. (Group 1 — transparent access)

```
read reg4
```

Reg4 can be read from the shadow rams without waiting for any pipeline instruction which could be in the pipe to complete. (Group 1 — transparent access)

```
reg9 <-- reg10 - reg11
```

Because the pipe has received two instructions since it was last cleared, one of these must complete before the first access of this instruction can be acknowledged. (Group 2 — non-transparent access)

Read WSTATUS from "clear pipe" address

The WSTATUS register can be read from two addresses - a "clear" address and a "stable" address. If read from the "clear" address, the data is not returned until all instructions in the pipe complete (pipe is clear), or until the pipe is hung. If read from the "stable" address the pipe stabilizes before the status is returned. Bus errors are generated only if this register is read from the "clear" address. (Group 2 — non-transparent access)

A pipeline instruction which generates an error will not fall out of the pipe. Instead, it hangs the pipe so the error handling routine can identify the source of the error. The following algorithm for handling errors is implemented in hardware:

```
IF (access produces an immediate error) THEN
    return negative acknowledge      : any pending Weitek errors are ignored
ELSE
    CASE:
        (access == transparent)    : return positive acknowledge
        (access == non-transparent):
            IF (there is a WERR pending) THEN
                return negative acknowledge
            ELSE
                wait (for pipe to progress (this
                    will convert access to a
                    transparent access) OR
                    for a WERR to occur (pipe
                    will hang))

                IF (there is a WERR pending) THEN
                    return negative acknowledge
                ELSE
                    return positive acknowledge
                ENDIF
            ENDIF
    ENDCASE
ENDIF
```

The software uses the following algorithm for handling errors:

```
Read the IERR status register

IF (there is an immediate error) THEN
  abort the user
  note: If the same access is again placed to the FPA
        another bus error will be issued by the FPA.
        The only state associated with an immediate
        error is the IERR register which is not used
        by the hardware. It exists only to indicate
        the source of the error to the processor.
ELSE
  read WSTATUS register from "stable pipe" address
                                ;FPA waits for pipe to stabilize before
                                ;acknowledging.
                                ;Status is read from "clear pipe"
                                ;address rather ;than "stable pipe"
                                ;address to guarantee that the FPA
                                ;will issue a positive acknowledge.

  IF (unimplemented instruction error) THEN
    abort the user
  ELSE
    read contents of the pipe
    write any value to CLEAR_PIPE register to clear the pipe
                                ;also clears Werr indicator in the
                                ;WSTATUS register

    re-execute faulting instruction using
      the 68020 and 68881 or some
      other method
    update appropriate FPA registers with results
      from re-execution

    retransmit unexecuted instructions or partial
      instructions which were read from the pipe

    return control to the user
  ENDIF
ENDIF
```

Figure 2-1 System Block Diagram

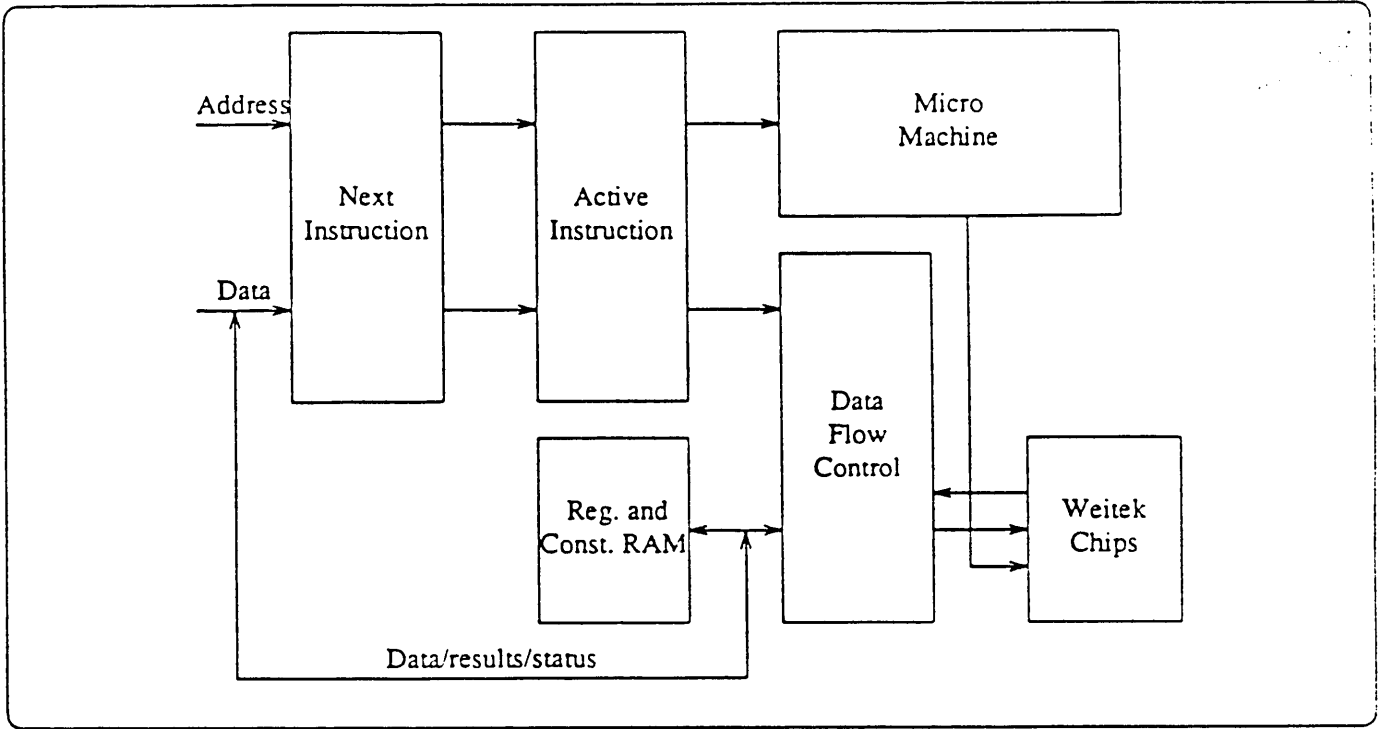


Figure 2-2 FPA Addressing

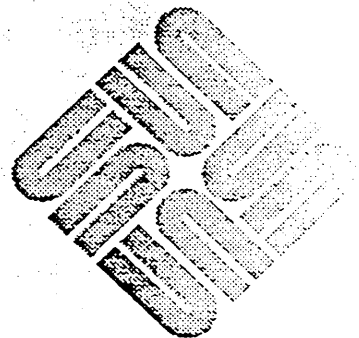
7 YTES XXXX XXX0	6 YTES XXXX XXX0	5 YTES XXXX XXX0	4 YTES XXXX XXX0	3 YTES XXXX XXX0	2 YTES XXXX XXX0	1 YTES XXXX XXX0	WRITE CMD	READ CMD
SHORT INSTRUCTIONS	0000	0000	0000	0000	0000	0000	IMM CLR STBL	IMM CLR STBL
COMMAND REGISTER INSTRUCTIONS	0001	0001	0001	0001	0001	0001	CLB	IMM
USER REGISTERS	0100	0100	0100	0100	0100	0100	IMM	IMM
OTHER	0101	0101	0101	0101	0101	0101		
EXTENDED INSTRUCTIONS	1000	1000	1000	1000	1000	1000		
	1001	1001	1001	1001	1001	1001		
	1010	1010	1010	1010	1010	1010		
	1011	1011	1011	1011	1011	1011		
	1100	1100	1100	1100	1100	1100	CLB	CLB
	1101	1101	1101	1101	1101	1101	CLB	CLB
	1110	1110	1110	1110	1110	1110	CLB	CLB
	1111	1111	1111	1111	1111	1111		

NOTE: THE STRING OF CHARACTERS AT THE TOP OF EACH COLUMN CORRESPOND TO THE 11 LOWER EMPHASIZED ADDRESS BITS OF THE FPA ADDRESS SPACE. NOTE: THE NUMBERS ALONGSIDE EACH COLUMN CORRESPOND TO THE YYY IN THE ADDRESS GIVEN ABOVE. THE TOP OF THE COLUMN



Machine Level Code

Machine Level Code	17
3.1. Register Accesses	17
Register RAM Accesses	17
Shadow RAM Accesses	18
Load Pointer Accesses	18
Control Register Accesses	19
Diagnostic Accesses	19
3.2. Instructions	19
Single Precision Short	19
Double Precision Short	20
Extended Instructions	22
Command Register	23



The register select specifies the address of one of the register RAM registers in the current context (FPR0 through FPR31).

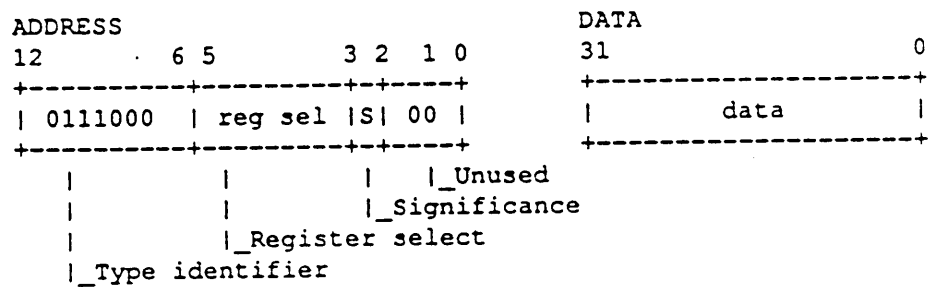
The significance bit specifies the significance of the operand; if bit 2 = 0, the data field contains the most significant portion of the operand; if bit 2 = 1, the data field contains the least significant portion of the operand. For single precision operands, bit 2 must be 0.

Shadow RAM Accesses

Shadow RAM provides fast read access to the lower 8 registers in the current context. During a shadow RAM access, the hardware first determines that the requested datum will not be updated as a result of instructions currently in the pipe. If the value will be updated, the hardware waits for the instruction to complete before performing the shadow RAM read.

Instructions which will update more than one register cause all the shadow registers to be interlocked until the instruction completes. These instructions include matrix moves, sincos, and matrix transposes.

Shadow RAM accesses use the following format:



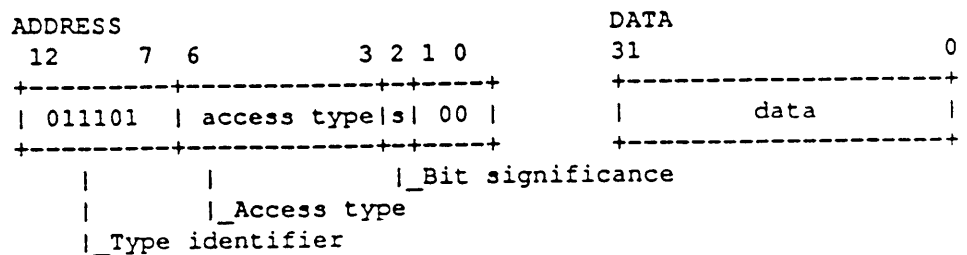
The type identifier identifies this as a shadow RAM access.

The register select specifies the address of one of 8 shadow RAM addresses (FPR0 through FPR7).

Load Pointer Accesses

Load pointer accesses provide a way to access any of the 2k addresses in the register/constants RAM. Load pointer bits 2 through 12 specify the register RAM address to be accessed, and the read/write line identifies the access as a read or a write. Note that if the microcode access bit in the STATE register is not set, it can still be read, but writes return an immediate negative acknowledge.

These accesses use the following format:



Address bits 7 to 12 identify this as a load pointer access.

Bits 6 to 3 further identify the type of access, as follows:

0000 = Read or write the RAM location specified by the load pointer. This is the normal usage.

0001 = Read or write the RAM location specified by the load pointer, and do the read using the recovery register. This command is used by diagnostics to test the recovery register (the recovery register is described in the chapter "Control Registers").

If bit 2 = 0, the operation accesses the most significant 32 bits; if bit 2 = 1, then the operation accesses the least significant 32 bits.

Control Register Accesses

This section describes how to access the FPA control registers. For a complete list of the control registers, including their functions and addresses, see the chapter "Control Registers".

Control registers accesses use the following format:

ADDRESS			DATA	
12	8 7	2 1 0	31	0
+-----+-----+-----+			+-----+-----+	
01111	reg. select	00		data
+-----+-----+-----+			+-----+-----+	
			_Register select (see chapter "Control Registers")	
_Type identifier				

The type identifier identifies this as a control register access. Note that this field provides the 0xF (in Table 6-1) for the register address. The `reg. select` field provides the middle 6 bits of the register address, and the last 2 bits of the register address are always zeros.

The register select specifies which register is being accessed.

Diagnostic Accesses

Diagnostic accesses are used for diagnostics only; they should never be used during normal operation.

3.2. Instructions

The FPA provides 4 different types of instructions; 1) single precision short, 2) double precision short, 3) extended, and 4) command register. Like the register accesses described above, address bits 31 through 28 access the FPA board, and bits 12 through 2 identify the command type, the operation, and other details about the command.

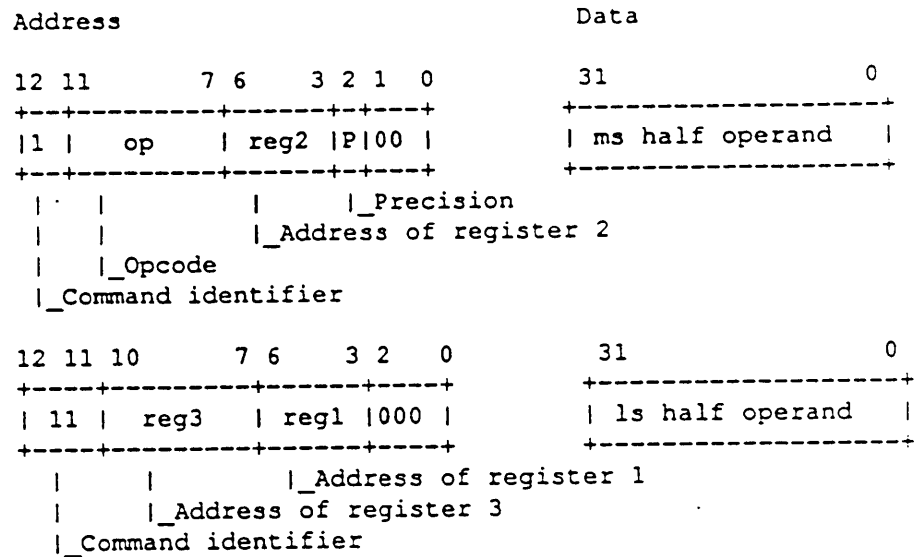
The following sections describe the commands and their variations, and identify the different operations available:

Single Precision Short

A single-precision short command processes a 32-bit operand in single-precision format. It uses the following format:

Extended Instructions

Extended instructions require two transfers, but they provide 22 bits of specification information. The first transfer contains enough information to get the operation started:



The two command identifiers identify this as (respectively) the first and second halves of an extended instruction.

The opcode specifies one of the extended commands. These are listed later.

The register addresses identify the registers for the operation (FPR0 through FPR15).

The precision bit specifies whether this is a single or double precision command. P = 0 specifies single precision; P = 1 specifies double precision. For single precision operations, the ms half holds the operand and the ls half is not used. For double precision operations, the ms half holds the ms half of the operand and the ls half holds the ls half of the operand.

The extended instructions support the following operations:

Opcode	Name	Operation
00110	add	reg1 ← reg2 + operand
00111	subtract	reg1 ← reg2 - operand
01000	multiply	reg1 ← reg2 * operand
01001	divide	reg1 ← reg2 / operand
01010	reverse sub	reg1 ← operand - reg2
01011	reverse div	reg1 ← operand / reg2
10000		reg1 ← reg3 + (reg2 * operand)
10001		reg1 ← reg3 - (reg2 * operand)
10010		reg1 ← (-reg3) + (reg2 * operand)
10011		reg1 ← reg3 * (reg2 + operand)
10100		reg1 ← reg3 * (reg2 - operand)
10101		reg1 ← reg3 * (-reg2 + operand)
10110		reg1 ← operand + (reg3 * reg2)
10111		reg1 ← operand - (reg3 * reg2)
11000		reg1 ← (-operand) + (reg3 * reg2)
11001		reg1 ← operand * (reg3 + reg2)
11010		reg1 ← operand * (reg3 - reg2)

NOTE *The following operations use the most significant half of the operand as a single precision "operand1", and the least significant half as a single precision "operand2". These operations are defined only for single precision:*

00000	reg1 ← operand2 + (reg2 * operand1)
00001	reg1 ← operand2 - (reg2 * operand1)
00010	reg1 ← -operand2 + (reg2 * operand1)
00011	reg1 ← operand2 * (reg2 + operand1)
00100	reg1 ← operand2 * (reg2 - operand1)
00101	reg1 ← operand2 * (-reg2 + operand1)

Command Register

Command register instructions allow the FPA to perform complicated commands with only 1 access. Because command information is encoded into the data field as well as the address field, the operands used by the operation must already be in the required registers.

Note that because the reg1 and reg4 fields are 5-bits wide, this format can specify from FPR0 to FPR31. The reg2 and reg3 fields are 9-bits wide. The C fields associated with reg2 and reg3 control which register that field specifies: if C = 0, the field specifies a register from FPR0 to FPR31 and the high 4 bits are ignored. If C = 1, the field specifies an offset into constant RAM (see Appendix A).

The P bit specifies the precision: 0 = single precision and 1 = double precision.

The format of command register instructions is:


```

010000      reg1 <-- reg2
010001      reg1 <-- reg3 + (reg2 * reg4)
010010      reg1 <-- reg3 - (reg2 * reg4)
010011      reg1 <-- (-reg3) + (reg2 * reg4)
010100      reg1 <-- reg3 * (reg2 + reg4)
010101      reg1 <-- reg3 * (reg2 - reg4)
010110      reg1 <-- reg3 * (-reg2 + reg4)

```

The following matrix operations are also provided:

NOTE *For all matrix move instructions in command register format, the result is undefined if the matrices overlap.*

010111 = reg1 <- (reg2)*(reg3) + (reg2+1)*(reg3+1)

011000 = reg1 <- (reg2)*(reg3) + (reg2+1)*(reg3+1) +
(reg2+2)*(reg3+2)

011001 = reg1 <- (reg2)*(reg3) + (reg2+1)*(reg3+1) +
(reg2+2)*(reg3+2) + (reg2+3)*(reg3+3)

110001 = 2x2 matrix move - 4 consecutive values starting
at reg2 are moved to locations starting at reg1

110010 = 3x3 matrix move - 9 consecutive values starting
at reg2 are moved to locations starting at reg1

110011 = 4x4 matrix move - 16 consecutive values starting
at reg2 are moved to locations starting at reg1

110100 = transpose 2x2 matrix - the matrix with element
1,1 pointed to by reg1 is transposed.

110101 = transpose 3x3 matrix - the matrix with element
1,1 pointed to by reg1 is transposed.

110110 = transpose 4x4 matrix - the matrix with element
1,1 pointed to by reg1 is transposed.

Other miscellaneous instructions:

011010 = load Weitek mode controls bits 3-0

NOTE *See the Weitek literature for definition of modes. The data source for mode bits 3-0 is bits 3-0 of the data operand. The FPA checks these accesses to ensure that bits 04 through 31 are 0's, or that bits 0 through 31 are 0x80000000, and hangs if they are not.*

Operating system commands:

101111 (P=0) - Initialize Weitek mode control bits 15-4 as follows:

- a) load multiplier with value 0x046
- b) load ALU with value 0x006

NOTE *After the Initialize command the Weitek error bits are undefined until the end of the first Weitek instruction.*

111000 (P=1) - Update the shadow registers.
101011 (P=0) - Load Weitek status register (WSTATUS)

NOTE *This instruction restores the Weitek status. Bits 8 through 11 of the data field are written to bits 8 through 11 of the WSTATUS register, then these bits are decoded to generate bits 0 through 4, bits 8 through 11, and bit 15 of the WSTATUS register. Bit 13 (status valid) is set (1), and bit 14 (unimplemented instruction) is reset (0). These decodings are described in the WSTATUS section of the chapter "Control Registers".*

101011 (P=1) - Unimplemented instruction. Returns a bus error with the unimplemented bit set in the bus error register.

100xxx (P=1) - Reserved

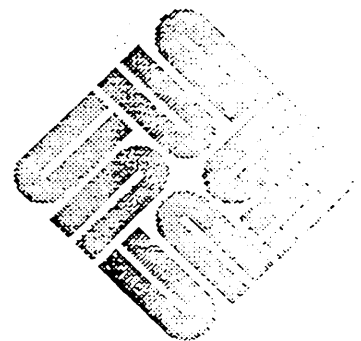
101101 (P=0) - Reserved

111001 (P=0) - Checks the user registers of the current context for any hardware faults. This check is destructive: the contents of all user registers in the current context will be destroyed.

111001 (P=1) - Checks the user registers of the current context for any hardware faults. This check is non-destructive.

Assembly Language Programming

Assembly Language Programming	29
4.1. Definitions	29
Instruction Notation	29
FPA Registers	30
Operand Types	30
4.2. Instructions	30
Two Operand Instructions	31
Three Operand Instructions	31
Four Operand Instructions	32
Multiply-Add	32
Multiply-Subtract	33
Multiply-Reverse Subtract	34
Add-Multiply	34
Subtract-Multiply	35
Other Operations	35
Floating Point Compares	36
FPA Errors Messages	36
4.3. Instruction Set Summary	37



Assembly Language Programming

This chapter describes the syntax of the Sun FPA assembly language instructions.

NOTE *This chapter describes the subset of the assembler which handles FPA instructions. For information about the rest of the assembler, see the "Assembly Language Reference Manual for the Sun Workstation" (800-1372).*

4.1. Definitions

This chapter consists of two parts; the instruction set summary at the end of the chapter, and information to support the instruction set summary. The first sections describe the types of instructions and the different symbols used. The instruction set summary (Table 3-2) provides a complete list of all the FPA assembler instructions, and shows the different operands and their order.

NOTE *Each line of code in the assembly language program is called an instruction. Do not confuse the word 'instruction' used in this chapter with the 'instructions' sent by the CPU to the FPA over the bus.*

The following sections define the elements used in this chapter.

Instruction Notation

This chapter uses the following format to display FPA instructions:

fpopi@A operands

where:

- ***fp*** identifies an FPA instruction
- ***op*** is the opcode name
- ***r*** identifies the precision; ***s*** = single precision, and ***d*** = double precision. For certain instructions, it can be ***l*** for long; these instructions are specially noted.
- ***@A*** is optional; it is used to specify an address register between 0 and 7 which contains the FPA base address. Note that this register must contain the FPA base address (0xE0000000) before this form can be used. If this element is not present, then absolute long addressing, which is more efficient for short routines, is used to refer to the FPA.
- The operand can be one of the following:

X represents either a 68020 effective address, or an FPA data register. Usually, if X is an FPA register, the assembler uses a command register command.

<ea> represents an 68020 effective address.

%x represents a register in constant RAM. x must be between 0 and 511. For locations in constant RAM, see Appendix A.

dn:dn represents a data register pair; an:an represents an address register pair.

FPA opcode names use the letters fp followed by an abbreviation that represents the action the opcode performs. For example, the opcode `fpsubs` specifies a subtraction. Certain instructions specify reversed operations: these begin with the letters `fp`. For example, the opcode `fprsubs` specifies reverse subtraction.

FPA Registers

The 32 FPA registers associated with each context are referred to as fpa0 through fpa31. In examples, they are referred to as fpa_x, fpa_y, fpa_z, and fpa_n, where fpa_n is the destination register.

Some instructions only allow access to fpa0 through fpa15. When this is the case, it is specified in the description of the instructions. Other operations allow access to constant RAM; these are identified by a percent sign. For example:

`%x`

specifies address x in constant RAM.

Writes to fpa0 through fpa7 update shadow RAM automatically. To optimize FPA speed, place results to be read from the FPA to the CPU in these registers whenever possible.

Operand Types

The assembler supports the following types of operands:

- A 68020 effective address. However, a) absolute short addresses are not allowed for double precision values, and b) if either the data register or the address register is used to hold a double precision value, then this value must be in a register pair, and both registers must appear separated by a colon (for example d0:d1). The instruction `fpltod` (convert integer to double precision) is an exception to this rule.
- Any of the 32 floating point data registers.
- Fpamode or fpastatus registers
- An address in constant RAM.

4.2. Instructions

The FPA instructions are divided into classes based on the numbers of operands they require. The following sections describe each.

Two Operand Instructions

Two operand instructions include things like add, multiply, convert from integer to floating point, and square root. They are represented as:

```
fpopl      X,fpan
```

X can be any valid 68020 effective address for an operand, or it can be an FPA register. If it is an FPA register, it can be either a register in constant RAM (0 to 511), or an FPA data register (0 to 31).

If X is an FPA register, then *fpan* must be an FPA data register in the range 0 to 31. If X is an effective address, then *fpan* must be an FPA data register in the range 0 to 15.

The following examples show some 2-operand instructions:

```
fpnegs      <ea>, fpa1
fpsqrd      <ea>, fpa2
fpsubs      fpa1, fpa2          → fpa2 ← fpa2 - fpa1
fprsubs     fpa1, fpa2          → fpa2 ← fpa1 - fpa2
fpdivs      d0, fpa2           → fpa2 ← fpa2 / d0
fprdivs     d0, fpa2           → fpa2 ← d0 / fpa2
```

The opcodes for sine, cosine, atan, e^x , e^x-1 , $\ln(x)$, $\ln(1+x)$, $\sqrt{\pi}(x)$, and $\text{sincos}(x)$ are all supported as register instructions. They use the following form:

```
fpopl      fpax, fpan
```

fpax is either an FPA register or a register in constant RAM. For sincos instructions, the destination operand is actually a register pair. For example:

```
fpsincos    fpan, fpac:fpas
```

where *fpac* is the cosine destination, and *fpas* is the sine destination.

Three Operand Instructions

The instructions add, subtract, multiply, and divide are supported in extended and command register form as follows:

```
fpop3l     X, fpax, fpan
```

If X is an effective address (<ea>), the operation is an extended command; if X is an FPA register (*fpax* or %*x*), then the operation is a command register command.

For extended instructions, *fpax* and *fpay* must be in the range 0 to 15.

For additions and multiplications, the first two operands can be exchanged without changing the result. For example:

```
fpadd3s    <ea>, fpa1, fpa2
```

is equivalent to:

```
fpadd3s    fpa1, <ea>, fpa2
```

Division and subtraction operations are not commutative; while the instructions will work with the operands exchanged, the result will be different. For example:

```
fpa2 ← <ea> - fpa1
```

must be coded as:

```
fsub3s     fpa1, <ea>, fpa2
```

Four Operand Instructions

Four-operand instructions take the form:

```
fpopt     X, fpax, fpay, fpaz
```

Fpay and X can be exchanged in both single or double precision instructions. In single precision form, two of the four operands may be effective addresses (<ea>); these are either the first and third, or the second and third operands.

NOTE *With commutative operators, the position of X and fpax can be exchanged.*

Just like in 3-operand instructions, if X is an FPA register, the instruction is a command register command, and if X is an effective address, the instruction is an extended instruction.

In the command register form, fpax and fpay can specify constant RAM registers by using the form:

```
%x and %y
```

When X is an effective address, fpax, fpay, and fpaz must be in the range 0 to 15. If X is an FPA register, fpax and fpaz must be from 0 to 31, and fpax and fpay can be either 0 to 31 (FPA register), or 0 to 511 (constant RAM).

Multiply-Add

The fpma instruction does a multiply, then an add. It can be generalized as:

```
fpmat     X, fpax, fpay, fpaz
```

r can be either s or d, and X is either an <ea> or fpa0 through fpa31. In the extended form, X and fpay can be exchanged. In single precision form, either X and fpay or fpax and fpay can be <ea>.

Note that the following example:

```
fpmas     d0, fpa1, fpa2, fpa3
```

is equivalent to the following sequence of instructions:

```

fpmul3s      d0, fpa1, temp
fpadd3s      temp, fpa2, temp
fpmoves      temp, fpa3

```

where `temp` is a temporary register.

The following examples show the different forms of operations, and an assembly code equivalent for each form:

```

reg1 ← reg3 + (reg2 * operand)    → fpma(s,d)  <ea>, reg2, reg3, reg1
reg1 ← operand + (reg3 * reg2)    → fpma(s,d)  reg2, reg3, <ea>, reg1
reg1 ← reg3 + (reg2 * reg4)       → fpma(s,d)  reg4, reg2, reg3, reg1
reg1 ← operand2 + (reg2 * operand1) → fpmas      <ea1>, reg2, <ea2>, reg1

```

Multiply-Subtract

The multiply-subtract instruction takes the form:

```
fpmst      X, fpax, fpay, fpaZ
```

`r` can be either `s` or `d`, and `X` is either an `<ea>` or `fpa0` through `fpa31`. In the extended form, `X` and `fpay` can be exchanged. In single precision form, either `X` and `fpay` or `fpax` and `fpay` can be `<ea>`.

Note that the following example:

```
fpmss fpa1, fpa2, d0, fpa3
```

is equivalent to the following sequence of instructions

```

fpmul3s      fpa1, fpa2, temp
fpsub3s      temp, d0, temp
fpmoves      temp, fpa3

```

The following examples show the different forms of operations, and an assembly code equivalent for each form:

```

reg1 ← reg3 - (reg2 * operand)    → fpms(s,d)  <ea>, reg2, reg3, reg1
reg1 ← operand - (reg3 * reg2)    → fpms(s,d)  reg2, reg3, <ea>, reg1
reg1 ← reg3 - (reg2 * reg4)       → fpms(s,d)  reg4, reg2, reg3, reg1
reg1 ← operand2 - (reg2 * operand1) → fpms      <ea1>, reg2, <ea2>, reg1

```


Multiply-Reverse Subtract

The multiply-reverse subtract instruction takes the form:

```
fpmrt    X, fpax, fpay, fpan
```

t can be either s or d, and X is either an <ea> or fpa0 through fpa31. In the extended form, X and fpay can be exchanged. In single precision form, either X and fpay or fpax and fpay can be <ea>.

Note that the following example:

```
fpmrs    d0, fpa1, fpa2, fpa3
```

is equivalent to the following sequence of instructions:

```
fpmul3s    d0, fpa1, temp
fpsub3s    fpa2, temp, temp
fpmoves    temp, fpa3
```

The following examples show the different forms of operations, and an assembly code equivalent for each form:

```
reg1 ← (-reg3) + (reg2 * operand) → fpmr(s,d) <ea>, reg2, reg3, reg1
reg1 ← (-operand) + (reg3 * reg2) → fpmr(s,d) reg2, reg3, <ea>, reg1
reg1 ← (-reg3) + (reg2 * reg4) → fpmr(s,d) reg4, reg2, reg3, reg1
reg1 ← (-operand2) + (reg2 * operand1) → fpmrs <ea1>, reg2, <ea2>, reg1
```

Add-Multiply

The add-multiply instruction takes the form:

```
fpamt    X, fpax, fpay, fpan
```

t can be either s or d, and X is either an <ea> or fpa0 through fpa31. In the extended form, X and fpay can be exchanged. In single precision form, either X and fpax or fpax and fpay can be <ea>.

Note that the following example:

```
fpams    fpa1, fpa2, fpa3, fpa4
```

is equivalent to the following sequence of instructions:

```
fpadd3s    fpa1, fpa2, temp
fpmul3s    temp, fpa3, temp
fpmoves    temp, fpa4
```

The following examples show the different forms of operations, and an assembly code equivalent for each form:

```

reg1 ← reg3 * (reg2 + operand)      → fpam(s,d) <ea>, reg2, reg3, reg1
reg1 ← operand * (reg3 + reg2)      → fpam(s,d) reg2, reg3, <ea>, reg1
reg1 ← reg3 * (reg2 + reg4)         → fpam(s,d) reg4, reg2, reg3, reg1
reg1 ← operand2 * (reg2 + operand1) → fpsms <ea1>, reg2, <ea2>, reg1

```

Subtract-Multiply

The subtract-multiply instruction takes the following form:

```
fpsmr X, fpax, fpay, fpan
```

r can be either *s* or *d*, and *X* is either an *<ea>* or fpa0 through fpa31. In the extended form, *X* and *fpay* can be exchanged. In single precision form, either *X* and *fpay* or *fpax* and *fpay* can be *<ea>*.

Note that the following example:

```
fpsms d0, fpa1, fpa2, fpa3
```

is equivalent to the following sequence of instructions:

```

fsub3s d0, fpa1, temp
fmul3s temp, fpa2, temp
fpmoves temp, fpa3

```

The following examples show the different forms of operations, and an assembly code equivalent for each form:

```

reg1 ← reg3 * (reg2 - operand)      → fpsm(s,d) <ea>, reg2, reg3, reg1
reg1 ← operand * (reg3 - reg2)      → fpsm(s,d) reg2, reg3, <ea>, reg1
reg1 ← reg3 * (reg2 - reg4)         → fpsm(s,d) reg4, reg2, reg3, reg1
reg1 ← reg3 * (-reg2 + operand)     → fpsm(s,d) reg2, <ea>, reg3, reg1
reg1 ← reg3 * (-reg2 + reg4)        → fpsm(s,d) reg2, reg4, reg3, reg1
reg1 ← operand2 * (reg2 - operand1) → fpsms <ea1>, reg2, <ea2>, reg1
reg1 ← operand2 * (-reg2 + operand1) → fpsms reg2, <ea1>, <ea2>, reg1

```

Other Operations

The following list shows operations not covered in previous sections. In each of these, the last operand is the destination, except for *tst*, *cmp*, and *mcmp*, where *fpstatus* is the implied destination. Note that *X* is either an *<ea>* or fpa0 through fpa31. Also, *r* is either *s* or *d* for all operations except *fpmover*, where *r* is either *s*, *d*, or *l* (for long).

Table 4-1 Other Operations

Opcode	Operands	Operation
fnop		nop
fpstz	X	operand compare with zero
fpcmpr	X, fpam	register m compare with operand
fpmcmpr	X, fpam	register m compare magnitude with operand
fpmover	fpam, fpan	move floating-point registers
fpmove2t	fpam, fpan	2x2 matrix move
fpmove3t	fpam, fpan	3x3 matrix move
fpmove4t	fpam, fpan	4x4 matrix move
fpdot2t	fpax, fpay, fpan	$fpan \leftarrow fpax * fpay + (fpax+1) * (fpay+1)$
fpdot3t	fpax, fpay, fpan	$fpan \leftarrow fpax * fpay + (fpax+1) * (fpay+1) + (fpax+2) * (fpay+2)$
fpdot4t	fpax, fpay, fpan	$fpan \leftarrow fpax * fpay + (fpax+1) * (fpay+1) + (fpax+2) * (fpay+2) + (fpax+3) * (fpay+3)$
fptran2t	fpam, fpan	transpose 2x2 matrix
fptran3t	fpam, fpan	transpose 3x3 matrix
fptran4t	fpam, fpan	transpose 4x4 matrix
fpmove	fpamode, <ea>	read mode register
fpmove	<ea>, fpamode	write to mode register
fpmove	fpastatus, <ea>	read status register
fpmove	<ea>, fpastatus	write to status register
fpmover	fpam, <ea>	read a floating-point data register
fpmover	<ea>, fpan	write to a floating-point data register

Floating Point Compares

The 68020 convention defines how bits are set up in the CC register for floating point compares. However, Sun provides a number of pseudo-ops to provide conditional branching after a floating point condition has been loaded into the CC register. Use the following form:

```
fpcmpt X, fpam
fpmove fpstatus, dn
jfcc
```

where cc =

```
eg equal
ne not equal
lt less than
ll less than or equal
gt greater than
ge greater than or equal
```

FPA Errors Messages

In addition to its normal error reports, the assembler reports the following FPA-specific errors:

- It reports invalid operand in double precision operations, when absolute short addressing, single data, or address register is used.

- For most instructions where one operand is an <ea>, the register range is 0 to 15. If all operands are FPA registers, then the register range is 0 to 31. For constant RAM registers, the range is 0 to 511. The assembler reports an invalid operand, and register out of range when any instruction specifies one of these registers outside of its range.

4.3. Instruction Set Summary

The following table shows the entire set of FPA assembler instructions. Note that in some three and four-operand instructions, the positions of X and the FPA register can be exchanged. This is shown in the fourth column.

Table 4-2 *Instruction Set Summary*

OPCODE	OPERANDS	OPERATION	ALTERNATIVE
fpnegs	X, fpan	negate single	
fpnegd	X, fpan	negate double	
fpabss	X, fpan	absolute value single	
fpabsd	X, fpan	absolute value double	
fpitos	X, fpan	convert integer to single	
fpitod	X, fpan	convert integer to double	
fpstol	X, fpan	convert single to integer	
fpdtol	X, fpan	convert double to integer	
fpstod	X, fpan	convert single to double	
fpdtos	X, fpan	convert double to single	
fpsqrs	X, fpan	square single	
fpsqrd	X, fpan	square double	
fpadds	X, fpan	add single	
fpadd3s	X, fpam, fpan	add single	fpam, X, fpan
fpaddd	X, fpan	add double	
fpadd3d	X, fpam, fpan	add double	fpam, X, fpan
fpsubs	X, fpan	subtract single	
fpsub3s	X, fpam, fpan	subtract single	fpam, X, fpan
fprsubs	<ea>, fpan	reverse subtract single	

Table 4-2 *Instruction Set Summary—Continued*

OPCODE	OPERANDS	OPERATION	ALTERNATIVE
fpsubd	X, <i>fpan</i>	subtract double	
fpsub3d	X, <i>fpam</i> , <i>fpan</i>	subtract double	<i>fpam</i> , X, <i>fpan</i>
fprsubd	<ea>, <i>fpan</i>	reverse subtract double	
fpmuls	X, <i>fpan</i>	multiply single	
fpmul3s	X, <i>fpam</i> , <i>fpan</i>	multiply single	<i>fpam</i> , X, <i>fpan</i>
fpmuld	X, <i>fpan</i>	multiply double	
fpmul3d	X, <i>fpam</i> , <i>fpan</i>	multiply double	<i>fpam</i> , X, <i>fpan</i>
fpdivs	X, <i>fpan</i>	divide single	
fpdiv3s	X, <i>fpam</i> , <i>fpan</i>	divide single	<i>fpam</i> , X, <i>fpan</i>
fprdivs	<ea>, <i>fpan</i>	reverse divide single	
fpdivd	X, <i>fpan</i>	divide double	
fpdiv3d	X, <i>fpam</i> , <i>fpan</i>	divide double	<i>fpam</i> , X, <i>fpan</i>
fprdivd	<ea>, <i>fpan</i>	reverse divide double	
fpnop		nop	
fptsts	X	single compare with 0	
fptstd	X	double compare with 0	
fpcmps	X, <i>fpam</i>	single compare	
fpcmpd	X, <i>fpam</i>	double compare	
fpmcmps	X, <i>fpam</i>	single magnitude compare	
fpmcmpd	X, <i>fpam</i>	double magnitude compare	
fpsins	<i>fpax</i> , <i>fpan</i>	sine single	
fpsind	<i>fpax</i> , <i>fpan</i>	sine double	
fpcooss	<i>fpax</i> , <i>fpan</i>	cosine single	
fpcosd	<i>fpax</i> , <i>fpan</i>	cosine double	
fpatans	<i>fpax</i> , <i>fpan</i>	atan single	
fpatand	<i>fpax</i> , <i>fpan</i>	atan double	
fpetoxs	<i>fpax</i> , <i>fpan</i>	e^x single	

Table 4-2 Instruction Set Summary—Continued

OPCODE	OPERANDS	OPERATION	ALTERNATIVE
fpetoxd	fpax, fpan	e^x double	
fpetoxmls	fpax, fpan	e^x-1 single	
fpetoxmld	fpax, fpan	e^x-1 double	
fplogns	fpax, fpan	$\ln(x)$ single	
fplognd	fpax, fpan	$\ln(x)$ double	
fplognpls	fpax, fpan	$\ln(1+x)$ single	
fplognpld	fpax, fpan	$\ln(1+x)$ double	
fpsincoss	fpax, fpac:fpas	fpac \leftarrow cosine(x), fpas \leftarrow sine(x)	
fpsincosd	fpax, fpac:fpas	fpac \leftarrow cosine(x), fpas \leftarrow sine(x)	
fpmas	X, fpax, fpay, fpan	$fpan \leftarrow (fpax * X) + fpay$	fpax, X, fpay, fpan fpay, fpax, X, fpan X, fpax, X, fpan fpax, X, X, fpan
fpmad	X, fpax, fpay, fpan	$fpan \leftarrow (fpax * X) + fpay$	fpax, X, fpay, fpan fpay, fpax, X, fpan
fpmss	X, fpax, fpay, fpan	$fpan \leftarrow fpay - (fpax * X)$	fpax, X, fpay, fpan fpay, fpax, X, fpan X, fpax, X, fpan fpax, X, X, fpan
fpmstd	X, fpax, fpay, fpan	$fpan \leftarrow fpay - (fpax * X)$	fpax, X, fpay, fpan fpay, fpax, X, fpan
fpmrs	X, fpax, fpay, fpan	$fpan \leftarrow (fpax * X) - fpay$	fpax, X, fpay, fpan fpay, fpax, X, fpan X, fpax, X, fpan fpax, X, X, fpan
fpmrd	X, fpax, fpay, fpan	$fpan \leftarrow (fpax * X) - fpay$	fpax, X, fpay, fpan fpay, fpax, X, fpan

Table 4-2 Instruction Set Summary—Continued

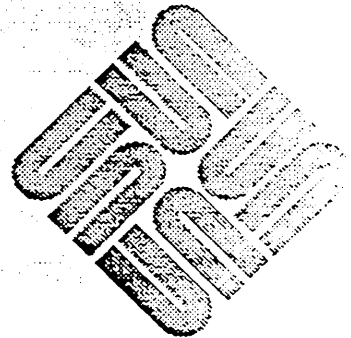
OPCODE	OPERANDS	OPERATION	ALTERNATIVE
fparams	X, fpax, fpay, fpan	$fpan \leftarrow (fpax + X) * fpay$	fpax, X, fpay, fpan fpay, fpax, X, fpan X, fpax, X, fpan fpax, X, X, fpan
fparamd	X, fpax, fpay, fpan	$fpan \leftarrow (fpax + X) * fpay$	fpax, X, fpay, fpan fpay, fpax, X, fpan
fpsms	X, fpax, fpay, fpan	$fpan \leftarrow (fpax - X) * fpay$	fpax, X, fpay, fpan fpay, fpax, X, fpan X, fpax, X, fpan fpax, X, X, fpan
fpsmd	X, fpax, fpay, fpan	$fpan \leftarrow (fpax - X) * fpay$	fpax, X, fpay, fpan fpay, fpax, X, fpan
fpmoves	<ea>, fpan	write to a register, single	
fpmoved	<ea>, fpan	write to a register, double	
fpmovel	<ea>, fpan	write to a register, integer	
fpmoves	fpan, <ea>	read a register, single	
fpmoved	fpan, <ea>	read a register, double	
fpmove2s	fpan, fpan	2x2 matrix move, single	
fpmove2d	fpan, fpan	2x2 matrix move, double	
fpmove3s	fpan, fpan	3x3 matrix move, single	
fpmove3d	fpan, fpan	3x3 matrix move, double	
fpmove4s	fpan, fpan	4x4 matrix move, single	
fpmove4d	fpan, fpan	4x4 matrix move, double	
fpdot2s	fpax, fpay, fpan	$fpan \leftarrow fpax * fpay + (fpax + 1) * (fpay + 1)$	
fpdot2d	fpax, fpay, fpan	$fpan \leftarrow fpax * fpay + (fpax + 1) * (fpay + 1)$	

Table 4-2 Instruction Set Summary—Continued

OPCODE	OPERANDS	OPERATION	ALTERNATIVE
fpdot3s	fpax, fpay, fpan	$fpan \leftarrow fpax * fpay + (fpax+1) * (fpay+1) + (fpax+2) * (fpay+2)$	
fpdot3d	fpax, fpay, fpan	$fpan \leftarrow fpax * fpay + (fpax+1) * (fpay+1) + (fpax+2) * (fpay+2)$	
fpdot4s	fpax, fpay, fpan	$fpan \leftarrow fpax * fpay + (fpax+1) * (fpay+1) + (fpax+2) * (fpay+2) + (fpax+3) * (fpay+3)$	
fpdot4d	fpax, fpay, fpan	$fpan \leftarrow fpax * fpay + (fpax+1) * (fpay+1) + (fpax+2) * (fpay+2) + (fpax+3) * (fpay+3)$	
fptran2s	fpam, fpan	transpose 2x2 matrix, single	
fptran2d	fpam, fpan	transpose 2x2 matrix, double	
fptran3s	fpam, fpan	transpose 3x3 matrix, single	
fptran3d	fpam, fpan	transpose 3x3 matrix, double	
fptran4s	fpam, fpan	transpose 4x4 matrix, single	
fptran4d	fpam, fpan	transpose 4x4 matrix, double	
fpmove	fpamode, <ea>	read the mode register	
fpmove	<ea>, fpamode	write on mode register	
fpmove	fpastatus, <ea>	read the status register	
fpmove	<ea>, fpastatus	write to status register	

Command Translations

Command Translations	45
5.1. Register RAM Accesses	45
5.2. Single Precision Short	46
5.3. Double Precision Short	47
5.4. Extended	47
5.5. Shadow RAM Read	48
5.6. Command Register	49



Command Translations

This section provides a general description of how to reference the FPA board. It provides examples of commands that might be used to do this.

NOTE *Normally, users have no need to deal with the issues in this chapter; the assembler does it for them.*

The 68020 uses move instructions to manipulate the FPA. The examples in this chapter use the *move* instruction. The address portion of this instruction provides information about the move; bits 28 through 31 must be 0xE to identify an FPA access, and bits 02 through 12 contain fields which describe the operation to the FPA. The purpose of this section is to illustrate the meaning of address bits 02 through 12.

NOTE *The FPA only uses bits 2 through 12; bits 0 and 1 are always 0's and appear as such in these examples.*

The read/write line determines whether the access is a read or a write.

The address takes the form:

E000XXXX

where XXXX represents address bits 0 through 12.

5.1. Register RAM Accesses

This example shows a typical register RAM access that loads an operand to FPR0. The general form of the command is:

```
move operand, FPA_address (0x0C04)
```

The address field contains the value 0xE0000C04; the leading E identifies this as an FPA address, the 000 are not used, and address bits 0 through 12 contain:

```

000 01100 0000 01 00
|   |   |   |   |   |_Unused (must be 0's)
|   |   |   |   |   |_Identifies this as the lsw
|   |   |   |   |   |_Selects FPA register 0 (FPR0)
|   |   |   |   |   |_Identifies this as register RAM access
|_Unused (must be 0's)

```

compressed into 4-bit fields:

```
0000 1100 0000 0100
```

or, in hex:

```
0x0C04
```

5.2. Single Precision Short

A single precision short command actually causes the FPA to process a number. Note that the command in this example places the result in FPR5, and that to obtain the result, the 68020 must read the contents of FPR5 from shadow RAM or register RAM as shown later.

The following example shows an add, where:

```
FPR5 <-- operand + FPR5
```

It can be expressed as:

```
move1 operand, FPA_address (0x03A8)
```

Again, the address field holds the value E00003A8. According to the single precision short command description in the chapter "Machine Level Code", address bits 0 through 12 should be:

```

000 00 0111 0101 0 00
|   |   |   |   |   |_Unused (must be 0's)
|   |   |   |   |   |_Specifies single precision
|   |   |   |   |   |_Identifies FPR5
|   |   |   |   |   |_Identifies operation as ``add``
|   |   |   |   |   |_Specifies ``short``
|_Unused (must be 0's)

```

compressed into 4-bit fields:

```
0000 0011 1010 1000
```

translated into hex:

```
0x03A8
```



```
0 00000 0 00000000 0 00000011 0 00010
|-----| | | | |
Unused | | | |
| | | |
| | | |
|_Identifies operand source as register R2M
```

compressed into 4-bit fields:

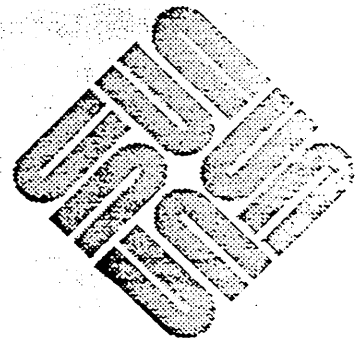
```
0000 0000 0000 0000 0000 0000 1100 0100
```

and in hex:

```
0x000000C4
```

Control Registers

Control Registers	53
IERR	53
IMASK	54
STATE Register	54
LOAD_PTR	54
LD_RAM	55
Pipe Access Registers	55
MODE Register	55
WSTATUS	56
READ_REG	57
Register and Microcode Address	57
PIPE_STATUS	57
HARD_CLEAR PIPE	58
CLEAR_PIPE	58
Register List	58



Control Registers

This chapter lists and describes all the system control registers. It lists the conditions under which the register can be written or read and describes the contents of each register.

Accesses to these registers are described in the chapter "Machine Level Code".

Note that unspecified bits return an unknown value when read.

IERR

The bits in this register identify the cause of any errors. If a bit = 1, the corresponding error occurred.

It can be written and read.

BIT	ERROR
16	non-32 bit access
17	protection violation: user write to supervisor space; attempt to write regRAM without regRAM-access enable bit set; attempt to access ustore/map RAM without load enable bit set; or attempt to access illegal address
18	illegal access: reading a write-only address, or writing a read-only address
19	attempt to execute via microcode when the load enable bit is set
20	illegal access sequence
21	hung pipe - pipe access shadow access control access
22	256th retry
23	illegal control register address

IMASK

Writes to this register turn the Weitek inexact error bit ON and OFF; reads tell you whether it is ON or OFF.

It can be written and read.

BIT	CONTENTS
0	inexact error mask: 0 - errors are disabled 1 - errors are enabled

STATE Register

The state register contains assorted information about the FPA. It can be read anytime, but written from the supervisor state only:

BIT	CONTENTS
4:0	current context
6	register RAM access enable bit 0 - Disables access to reg RAM via pointer. 1 - Enables access
5,7	load enable bit 0 - Disables access to microstore. 1 - Enables access to microstore. Disables execution of instructions which use the microstore. (The data from bit 7 is written to bits 5 and 7 of the STATE register. The contents may be read from either bit.)

LOAD_PTR

The load pointer is used to directly access register RAM. These accesses are described in the "Load Pointer Access" section of the chapter "Machine Level Code".

While this register also works with the LD_RAM (load RAM) register access, these accesses are not described or supported in this manual.

bits	contents
13:2	register RAM address
1:0	ignored

LD_RAM

This register is used to directly access the micromachine. These accesses are not described or supported in this manual.

Pipe Access Registers

The pipe access registers allow you to inspect the contents of the instruction pipe. These are read-only; you cannot write directly to the pipe. The $|v|$ bits determine whether that part of an instruction is valid ($v = 0$ is valid; $v = 1$ indicates invalid):

PIPE_ACT_INS	(r)	+-----+-----+-----+-----+ v instr, 1st half v instr, 2nd half
PIPE_NXT_INS	(r)	+-----+-----+-----+-----+ v instr, 1st half v instr, 2nd half
PIPE_ACT_D1	(r)	+-----+-----+-----+-----+ data, 1st half
PIPE_ACT_D2	(r)	+-----+-----+-----+-----+ data, 2nd half
PIPE_NXT_D1	(r)	+-----+-----+-----+-----+ data, 1st half
PIPE_NXT_D2	(r)	+-----+-----+-----+-----+ data, 2nd half

NOTE Bits 15 and 31 ($|v|$) of the PIPE_ACT_INS and PIPE_NXT_INS registers indicate whether that instruction was a valid instruction previously issued by the processor: If $v = 1$, instruction was not valid and a valid instruction does not exist in the pipe. If $v = 0$, instruction was valid and a valid instruction does exist in the pipe. Each instruction half is the address to which that access was originally written (the $|v|$ bit is appended). To retransmit an instruction a PIPE_XXX_Dy register is written to the address specified by the corresponding half of the corresponding PIPE_XXX_INS register. This address should be offset by 0xE0000000. To retransmit, mask all bits except 2 - 12.

Bits 0, 1, 13, 14, 16, 17, 29 and 30 of PIPE_XXX_INS are undefined. All undefined bits are undefined during reads, and should be masked to 0 for writes.

MODE Register

This register contains the Weitek MODE bits. To write this register, use the form `fpmove <ea>, fpamode`. To read this register, use the form `fpmove fpamode, <ea>`.

+-----+-----+-----+-----+ Mode (0:3) +-----+-----+-----+-----+
--

bits	contents
3:0	Mode bits 3:0 of the Weitek chips. Consult the Weitek spec for a complete description.

NOTE Certain transcendental functions may hang if the mode register contains a value other than 0x2.

WSTATUS

This register contains the Weitek status bits latched out from the Weitek chips when they complete an operation. To read this register, use the form `fpmove1 fpastatus, <ea>`. To write this register, use the form `fpmove1 <ea>, fpastatus`.

Writing to this register does not affect the Weitek chips. However, after a context switch, it may be necessary to restore the status bits from the original context.

On power up and after a clear pipe, the Weitek status is invalid until these bits are set. This occurs every time the WSTATUS register is updated by the microcode.

The WSTATUS register is updated by the microcode on every arithmetic operation or compare. When an instruction that involves multiple arithmetic operations (dot product, multiply-accumulate, transcendentals) is executed, the status will correspond to the last operation, unless there is an exception, in which case the status will correspond to the operation that generated the exception (no indication is given as to which operation is in error). The WSTATUS register is updated by the microcode on every arithmetic operation or compare. When an instruction is executed that involves multiple arithmetic operations (dot product, multiply-accumulate, transcendentals), the status will correspond to the last operation, unless there is an exception, in which case it will correspond to the operation that generated the exception (no indication is given as to which operation is in error).

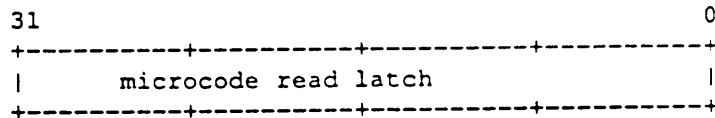
Weitek Status and Error Indicators	
bits	contents
15	Weitek error (WERR - excluding unimpl. instr.) 0 = error 1 = no error
14	unimplemented instruction
13	status valid:
11:8	status taken directly from Weitek chips
4:0	decoded status = Weitek comp cond
	equal (0000) 4 (00100)
	less than (0001) 25 (11001)
	greater than (0010) 0 (00000)
	unordered (1111) 2 (00010)
	all other values 0 (00000)

NOTE *The decoded status fields correspond to Sun's conventional use of the 68020 status register for floating point condition codes. For information on floating point compares, see the chapter "Assembly Language Programming".*

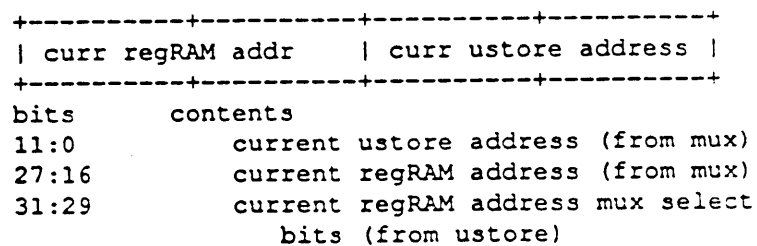
NOTE *If the pipe hangs and bits 11 through 8 contain a 0x4, this indicates either that a transcendental operation had an operand "out of bounds", or that the mode register contains a value other than 0x2 (transcendentals require that the mode register contain 0x2). The Weiteks define status 0x4 as "not used".*

READ_REG

The read register (READ_REG) displays the contents of the read latch (see Figure 1). This is the latch used by the microcode to return data to the processor. Direct access to this register is provided for diagnostic purposes only:

**Register and Microcode Address**

The register and microcode address register (REG_UST_ADDR) allows you to read the current address in microstore, and in register RAM. Note that register RAM addresses are only 13-bits long; bits 29 - 31 provide the current address in the MUX SELECT:

**PIPE_STATUS**

The pipe status register returns bits describing the current state of the instruction pipe. The hardware latches these status bits directly regardless of whether the pipe is moving or stable. After any latched data has stabilized, the data is returned to the processor.

This register is read only. It can be read from an immediate or a stable address, as listed in Table 6-1.

```

+-----+-----+-----+-----+
|                                     | pipe status bits |
+-----+-----+-----+-----+
BIT      CONTENTS
16      stable bit:
        0 - pipe may change state
        1 - pipe is:
            clear (no pending instructions)
            hung ( a WERR has occurred)
            waiting for 2nd half of instr
17      Weitek error has occurred
18      pipeline waiting for 2nd access of
        2-access instruction
19      micromachine waiting for 2nd access of
        2-access instruction
20      |v| bit from PIPE_ACT_INS - 1st half
21      |v| bit from PIPE_ACT_INS - 2nd half
22      |v| bit from PIPE_NXT_INS - 1st half
23      |v| bit from PIPE_NXT_INS - 2nd half
    
```

HARD_CLEAR PIPE

This register can be written from the supervisor state only; it clears the instruction pipe and the command register. All pending instructions are lost and Weitek status is lost. This access requires supervisor privilege, and is write only:

```

+-----+-----+-----+-----+
| clear pipe and cmd reg of current contents |
+-----+-----+-----+-----+
    
```

CLEAR_PIPE

Writing anything to the clear pipe register clears the instruction pipe. All pending instructions are lost and Weitek status is lost. This register is write only:

```

+-----+-----+-----+-----+
|          clear pipe of current contents          |
+-----+-----+-----+-----+
    
```

Register List

The following table lists the registers and the conditions pertinent to each:

Table 6-1 *Control Registers*

Reg Name	Addr	Spec Info	Write Cond	Read Cond	Description
STATE	0xF10	su cx	C	I	Contains enable bits and current context.
IMASK	0xF14	cx	C	I	Masks Weitek inexact error bit
LOAD_PTR	0xF18	cx	C	I	Describes a load operation for LD_RAM
IERR	0xF1C	cx	I	I	Contains errors on FPA board.
LD_RAM	0xFC0	le	C	C	Implements action described in LOAD_PTR.
PIPE_ACT_INS	0xF20	cx		S	Contains the active pipe instruction.
PIPE_NXT_INS	0xF24	cx		S	Contains the next pipe instruction.
PIPE_ACT_D1	0xF28	cx		S	Contains the first half of active pipe data.
PIPE_ACT_D2	0xF2C	cx		S	Contains the second half of active pipe data.
PIPE_NXT_D1	0xF30	cx		S	Contains the first half of next pipe data.
PIPE_NXT_D2	0xF34	cx		S	Contains the second half of next pipe data.
MODE3_0	0xFN8	mc cx		CS->	Clear = 0xFB8; Stable = 0xF38. Contains the Weitek MODE bits.
WSTATUS	0xFNC	mc cx		CS->	Clear = 0xFBC; Stable = 0xF3C. Contains Weitek status and error indicators.
READ_REG	0xF60			S	Reads contents of READ latch.
REG_uST_ADDR	0xF64			S	Contains current micro-store address, register RAM address, and MUX SELECT address.
WLWF_REG	0xF6C				Displays current L+ and F+ bits being sent to Weiteks.
PIPE_STATUS	0xFN8			I S	Immediate = 0x48F; Stable = 0x46F. Displays pipe status.

Table 6-1 Control Registers—Continued

Reg Name	Addr	Spec Info	Write Cond	Read Cond	Description
HARD CLEAR_PIPE	0xF80	su	S		Clears all pipe registers and the command register (where microstore outputs its commands) Pending instructions are lost and the Weitek status is cleared.
CLEAR PIPE	0xF84		S		Clears all pipe registers. Pending instructions are lost and the Weitek status is cleared.

The write and read conditions are:

- I Immediate — Data is always written or returned immediately. Either these registers cannot be altered by an advance of the instruction pipe, or they describe the state of the instruction pipe.
- C Clear — To assure that data is not changed after it is read, the FPA waits for the pipe to clear before writing or returning the data. If the pipe hangs on an error, the FPA returns a negative acknowledge.
- S Stable — The FPA waits for the pipe to stabilize on one of the following conditions before writing or returning the data:
 - clear
 - hung on error
 - waiting for second half of an instruction
 - combination of hung and waiting

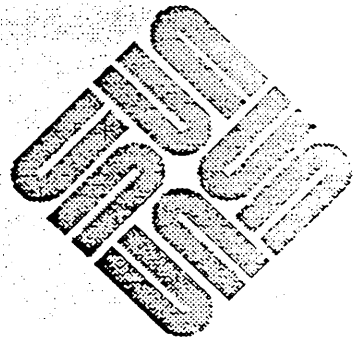
The special conditions are:

- su Supervisor only — Writes are only allowed from supervisor space.
- le LOAD_EN bit in STATE register — The LOAD_EN (load enable) bit in the STATE register must be ON.
- cx Context switch — Contents must be saved on context switch.
- mc Microcode — These are written via microcode. The procedure to read and write them is described in the sections on these registers earlier in this chapter.

A

Addresses in Constant RAM

Addresses in Constant RAM 63



Addresses in Constant RAM

This appendix provides the locations of values in constant RAM. The following table lists the name of the constant, its offset, and the value stored there. Note that the offset is the actual number you place in an FPA register to access the value (see the chapter "Machine Level Code").

The hardware translates these offsets into an actual address in the constant RAM:

Table A-1 *Addresses in Constant RAM*

Name	Offset	Hex Value	Description	Dec Value
szero	0	00000000	single precision zero	0
dzero	0	0000000000000000	double precision zero	
sminsub	1	00000001	minimum subnormal	1.401298464324817071E-45 2 ^{**} -150
dminsub	1	0000000000000001	minimum subnormal	4.940656458412465442E-324 2 ^{**} -1075
smaxsub	2	007FFFFFFF	maximum subnormal	1.175494210692441075E-38 2 ^{**} -127 - 2 ^{**} -150
dmaxsub	2	000FFFFFFFFFFFFFFF	maximum subnormal	2.225073858507200889E-308 2 ^{**} -1023 - 2 ^{**} -1075
sminnorm	3	00800000	minimum normal	1.175494350822287508E-38 2 ^{**} -127
dminnorm	3	0010000000000000	minimum subnormal	2.225073858507201383E-308 2 ^{**} -1023
smaxnorm	4	7F7FFFFFFF	maximum normal	3.402823466385288598E+38 2 ^{**} 127 - 2 ^{**} 103
dmaxnorm	4	7FEFFFFFFFFFFFFFFF	maximum normal	1.797693134862315708E+308 2 ^{**} 1023 - 2 ^{**} 970
sinf	5	7F800000	infinity	
dinf	5	7FF0000000000000	infinity	
ssnan	6	7FBFFFFFFF	signalling NaN (Not a Number)	
dsnan	6	7FF7FFFFFFFFFFFFFFF	signalling NaN	
sqnan	7	7FFFFFFF	silent NaN	
dqnan	7	7FFFFFFFFFFFFFFF	silent NaN	
se	8	402DF854	e	2.718281745910644531
de	8	4005BF0A8B1457692	e	2.718281828459045091

Table A-1 Addresses in Constant RAM—Continued

Name	Offset	Hex Value	Description	Dec Value
s2pi	9	40C90FDB	2*pi	6.283185482025146484
d2pi	9	401921FB54442D18	2*pi	6.283185307179586232
spi	A	40490FDB	pi	3.141592741012573242
dpi	A	400921FB54442D18	pi	3.141592653589793116
spio2	B	3FC90FDB	pi/2	1.570796370506286621
dpio2	B	3FF921FB54442D18	pi/2	1.570796326794896558
ssqrt2	C	3FB504F3	sqrt of 2	1.414213538169860840
dsqrt2	C	3FF6A09E667F3BCD	sqrt of 2	1.414213562373095145
ssqrhalf	D	3F3504F3	sqrt of 1/2	7.071067690849304199E-1
dsqrhalf	D	3FE6A09E667F3BCD	sqrt of 1/2	7.071067811865475727E-1
sone	E	3F800000	one	1
done	E	3FF0000000000000	one	1
shalf	F	3F000000	one half	.5
dhalf	F	3FE0000000000000	one half	.5
smone	10	BF800000	negative one	-1
dmone	10	BFF0000000000000	negative one	-1
stwo	11	40000000	two	2
dtwo	11	4000000000000000	two	2
sthree	B1	40400000	three	3
dthree	B1	4008000000000000	three	3
sfour	12	40800000	four	4
dfour	12	4010000000000000	four	4
seight	13	41000000	eight	8
deight	13	4020000000000000	eight	8
s1o2	14	3f000000	one half	.5
d1o2	14	3fe0000000000000	one half	.5
s1o4	15	3e800000	one quarter	.25
d1o4	15	3fd0000000000000	one quarter	.25
s1o8	16	3e000000	one eighth	.125
d1o8	16	3fc0000000000000	one eighth	.125
s1e1	17	41200000	ten	10
d1e1	17	4024000000000000	ten	10
s1e2	18	42c80000	one hundred	100
d1e2	18	4059000000000000	one hundred	100
s1e3	19	447a0000	one thousand	1000
d1e3	19	408F400000000000	one thousand	1000
s1e4	20	461C4000	ten thousand	10,000
d1e4	20	40C3880000000000	ten thousand	10,000
s1e5	21	47C35000	one hundred thousand	100,000
d1e5	21	40F86A0000000000	one hundred thousand	100,000
s1e6	22	49742400	one million	1,000,000
d1e6	22	412E848000000000	one million	1,000,000
s1e7	23	4B189680	ten million	10,000,000
d1e7	23	416312D000000000	ten million	10,000,000
s1e8	24	4CBEB20	one hundred million	100,000,000
d1e8	24	4197D78400000000	one hundred million	100,000,000

Table A-1 Addresses in Constant RAM—Continued

Name	Offset	Hex Value	Description	Dec Value
s1e9	25	4E6E6B28	one billion	1,000,000,000
d1e9	25	41CDCD6500000000	one billion	1,000,000,000
s1e10	26	501502F9	ten billion	10,000,000,000
d1e10	26	4202A05F20000000	ten billion	10,000,000,000
smpio2	27	BFC90FDB	pi/2	-1.570796370506286621
dmpio2	27	BFF921FB54442D18	pi/2	-1.570796326794896558
slog2e	28	3FB8AA3B	log ₂ (e)	1.442695021629333496
dlog2e	28	3FF71547652B82FE	log ₂ (e)	1.442695040888963387
slog2ten	29	40549A78	log ₂ (10)	3.321928024291992188
dlog2ten	29	400A934F0979A371	log ₂ (10)	3.321928094887362182
slogetwo	2A	3F317218	log _e (2)	6.931471824645996094E-1
dlogetwo	2A	3FE62E42FEFA39EF	log _e (2)	6.931471805599452862E-1
slogeten	2B	40135D8E	log _e (10)	2.302585124969482422
dlogeten	2B	40026BB1BBB55516	log _e (10)	2.302585092994045901
slog10two	2C	3E9A209B	log ₁₀ (2)	3.010300099849700928E-1
dlog10two	2C	3FD34413509F79FF	log ₁₀ (2)	3.010299956639811980E-1
slog10e	2D	3EDE5BD9	log ₁₀ (e)	4.342944920063018799E-1
dlog10e	2D	3FDBC7B1526E50E	log ₁₀ (e)	4.342944819032518167E-1
smhalf	2E	BF000000	negative one half	-1/2
dmhalf	2E	BFE0000000000000	negative one half	-1/2
s1e16	2F	5A0E1BCA	10**16	1.000000027256422400E+16)
d1e16	2F	4341C37937E08000	10**16	1.000000000000000000E+16
s1e32	30	749DC5AE	10**32	1.000000033181353514E+32
d1e32	30	4693B8B5B5056E17	10**32	1.000000000000000054E+32
s1e64	31	7f800000	10**64	infinity
d1e64	31	4D384F03E93FF9F5	10**64	1.000000000000000021E+64
s1e128	32	7f800000	10**128	infinity
d1e128	32	5A827748F9301D32	10**128	1.000000000000000075E+128
s1e256	33	7f800000	10**256	infinity
d1e256	33	75154FDD7F73BF3C	10**256	1.000000000000000030E+256

Index

A

access
 FPA, 4
 read, 8
 write, 8
access protection, 9

C

chip set
 Weitek, 8
code generators, 7
command register, 59
compilers, 7
constant RAM, 8, 30

E

error
 register out of range, 37
error message
 invalid operand, 36, 37
errors
 cause of, 53

F

fonts
 use of in text, 3
FPA bus, 8

H

host processor, 4

I

instruction pipe, 8
 contents of, 55
 status, 57
interlock
 shadow RAM, 18
invalid operand, 36, 37

L

LD_RAM register, 54

M

micromachine, 8
MODE bits
 Weitek, 55

mode control bits
 Weitek, 25
move instruction, 45

O

operand
 significance, 18

R

read latch
 contents of, 57
register
 LD_RAM, 54
register identification, 17
register out of range, 37
register RAM, 8
 writing to, 8

S

shadow RAM, 8, 30
 interlock, 18
significance
 operand, 18
state register, 9
status
 Weitek, 56
switched code, 7

W

Weitek error bit, 56
Weitek errors, 9
Weitek status, 56
WERR, 9, 56

Revision History

Revision	Date	Comments
01	January 10, 1986	Alpha draft
50	Feb 11, 1986	Beta draft — Incorporated comments from Alpha.
A	June 2, 1986	Incorporated comments from Beta

F P A

502-1105-01

Rev A

ECO	Description	Date	Approvals

ECO HISTORY				
ZONE	REV	DESCRIPTION	DATE	APPROVALS
-01	A	PROD. REL. PER ECO #3822	3-15-88	T.H.

SHEET REVISION

SHEET	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
REV	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

SHEET	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
REV	A	A	A	A	A	A	A	A	A	A	A	/	/	/	/	/	/	/	/	/



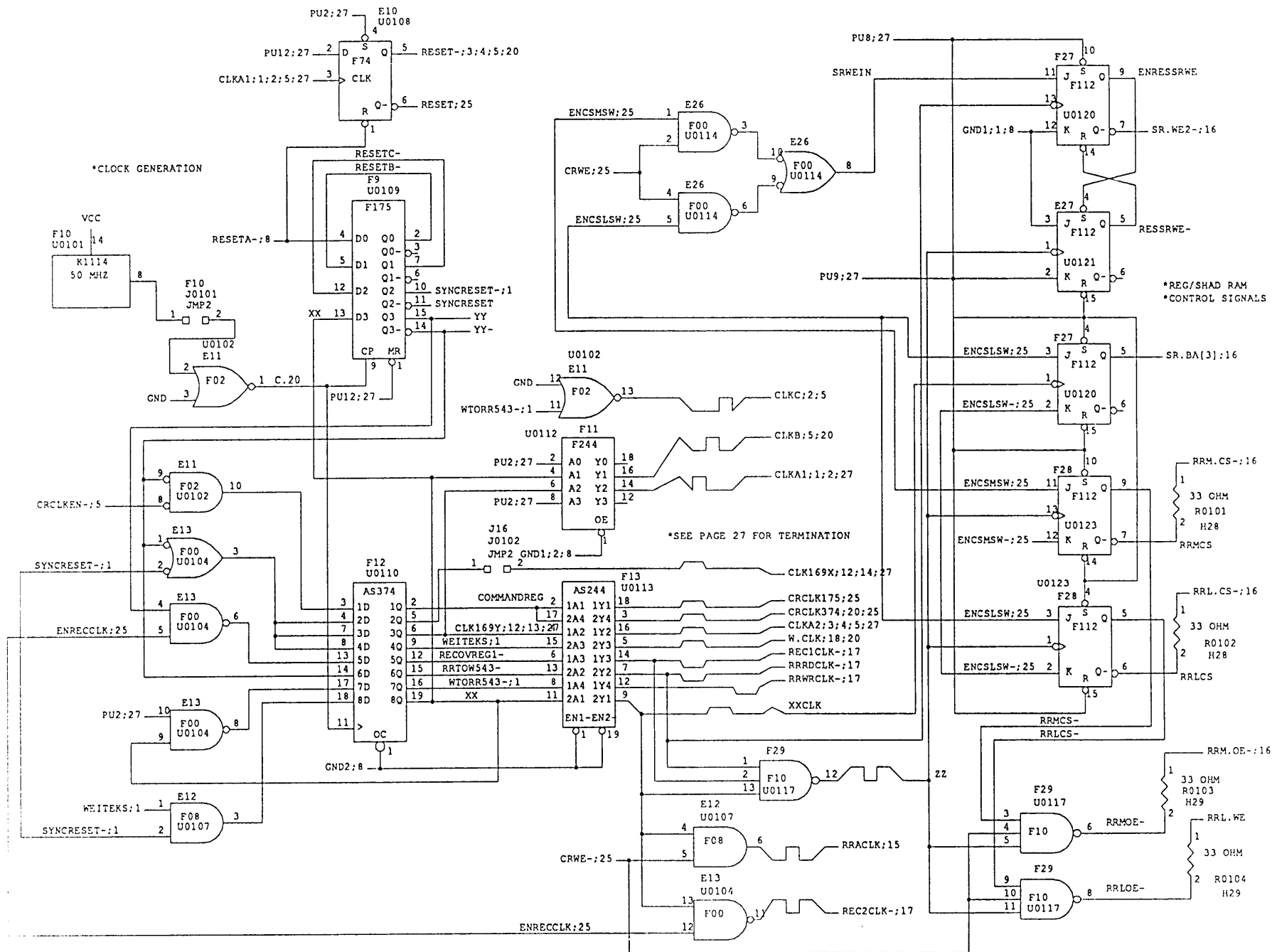
Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystem Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: COVER
 Engineer: XX

Drawing: 502-1105-01
 File: cover.d
 Date: Mon Aug 29 10:58:37 1988

Rev: A

ECO	Description	Date	Approvals
-----	-------------	------	-----------



Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 1 OF 31
 Engineer: S. CARRIE

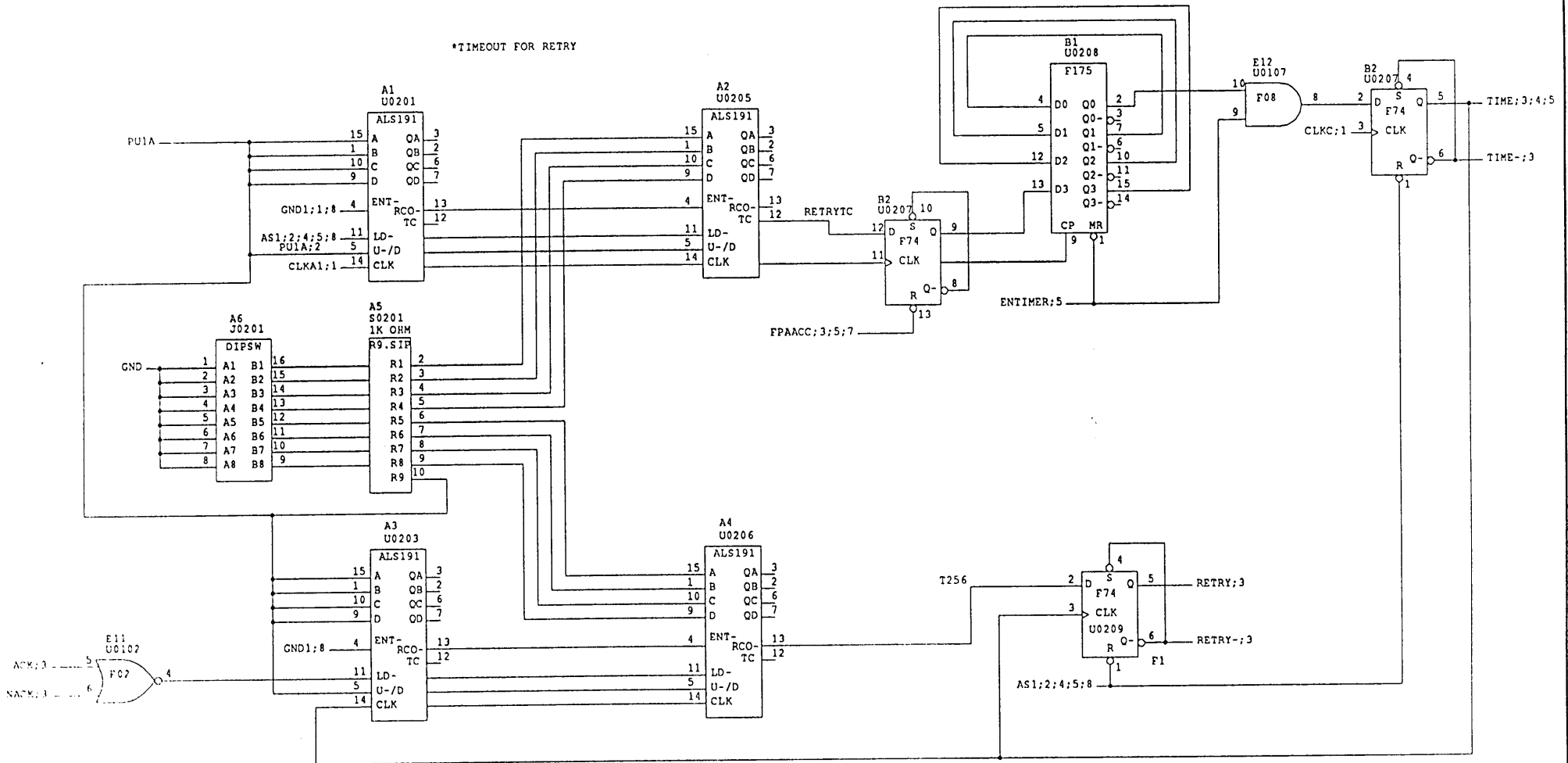
Drawing: 502-1105-01
 File: fpal.d
 Date: Mon Aug 15 15:24:01 1986
 Rev: A

ECO	Description	Date	Approvals
-----	-------------	------	-----------

*AVOIDS GLITCHES WITH ACK/NACK

*TIMEOUT FOR RETRY

*COUNT OF CONSECUTIVE RETRIES



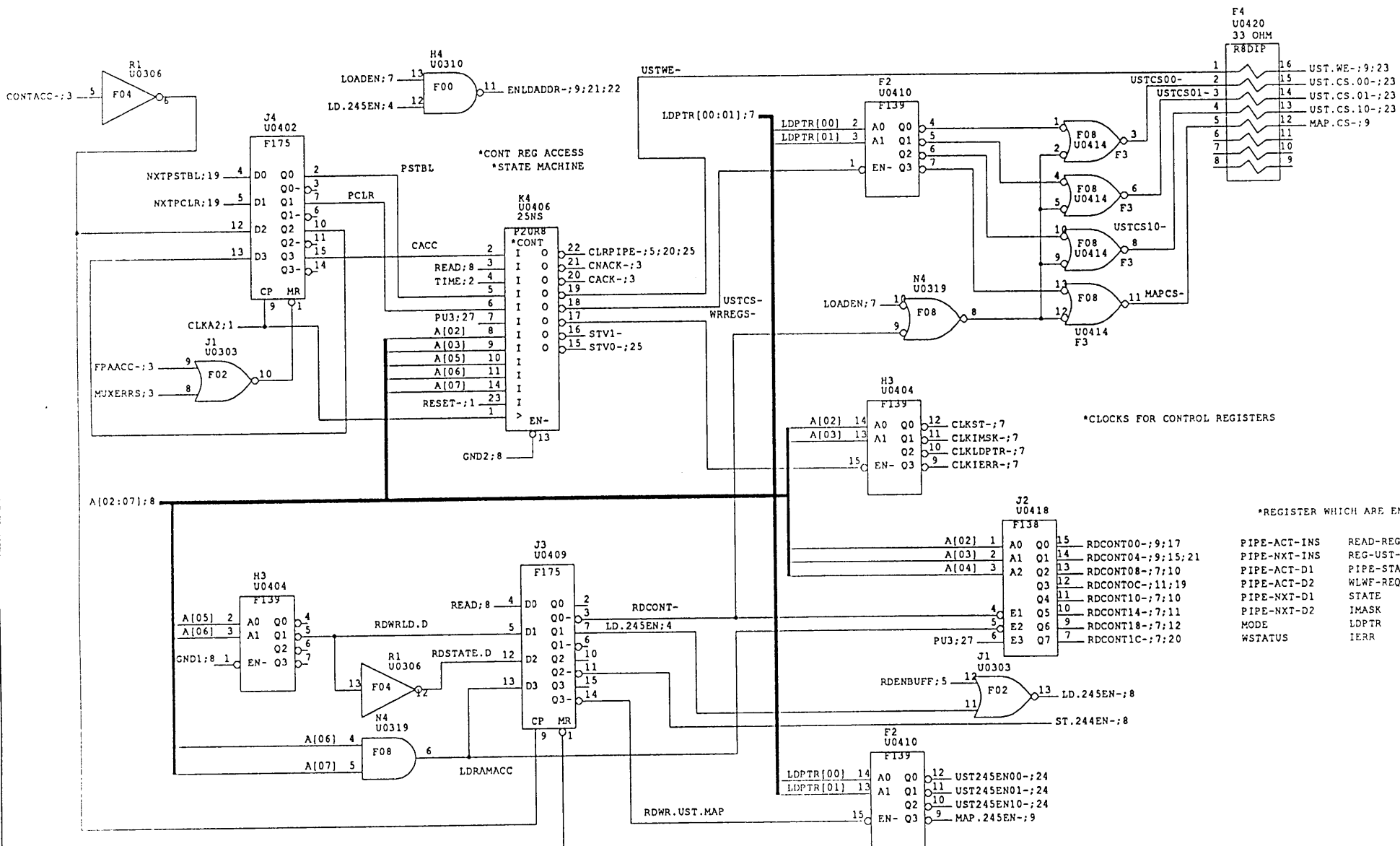
Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 2 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpa2.d
 Date: Mon Aug 15 15:32:49 1988

Rev: A

ECO	Description	Date	Approvals



AS1:8



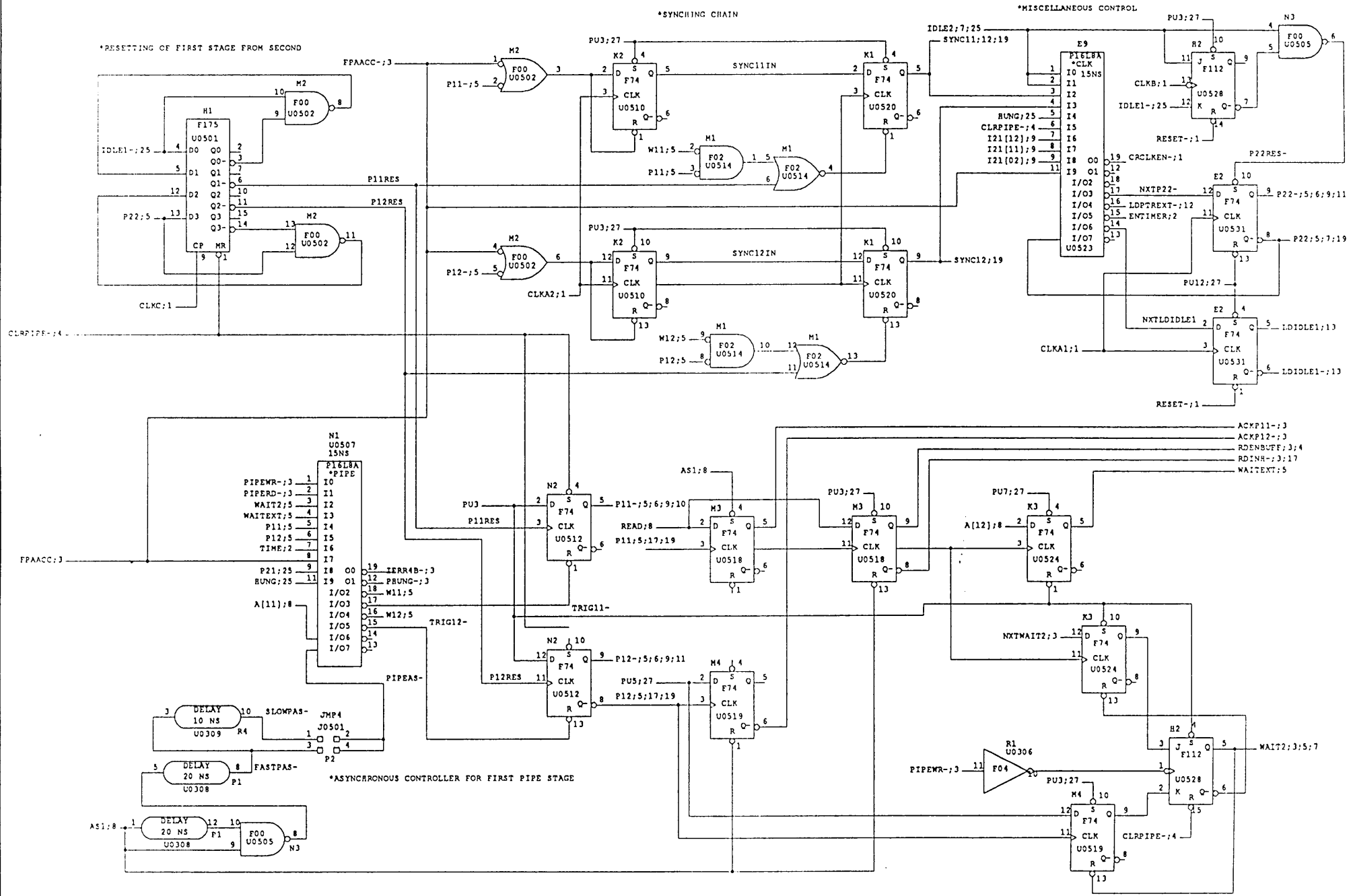
Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 4 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpa4.d
 Date: Mon Aug 15 15:39:47 1988

Rev: A

ECO	Description	Date	Approvals
-----	-------------	------	-----------



Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

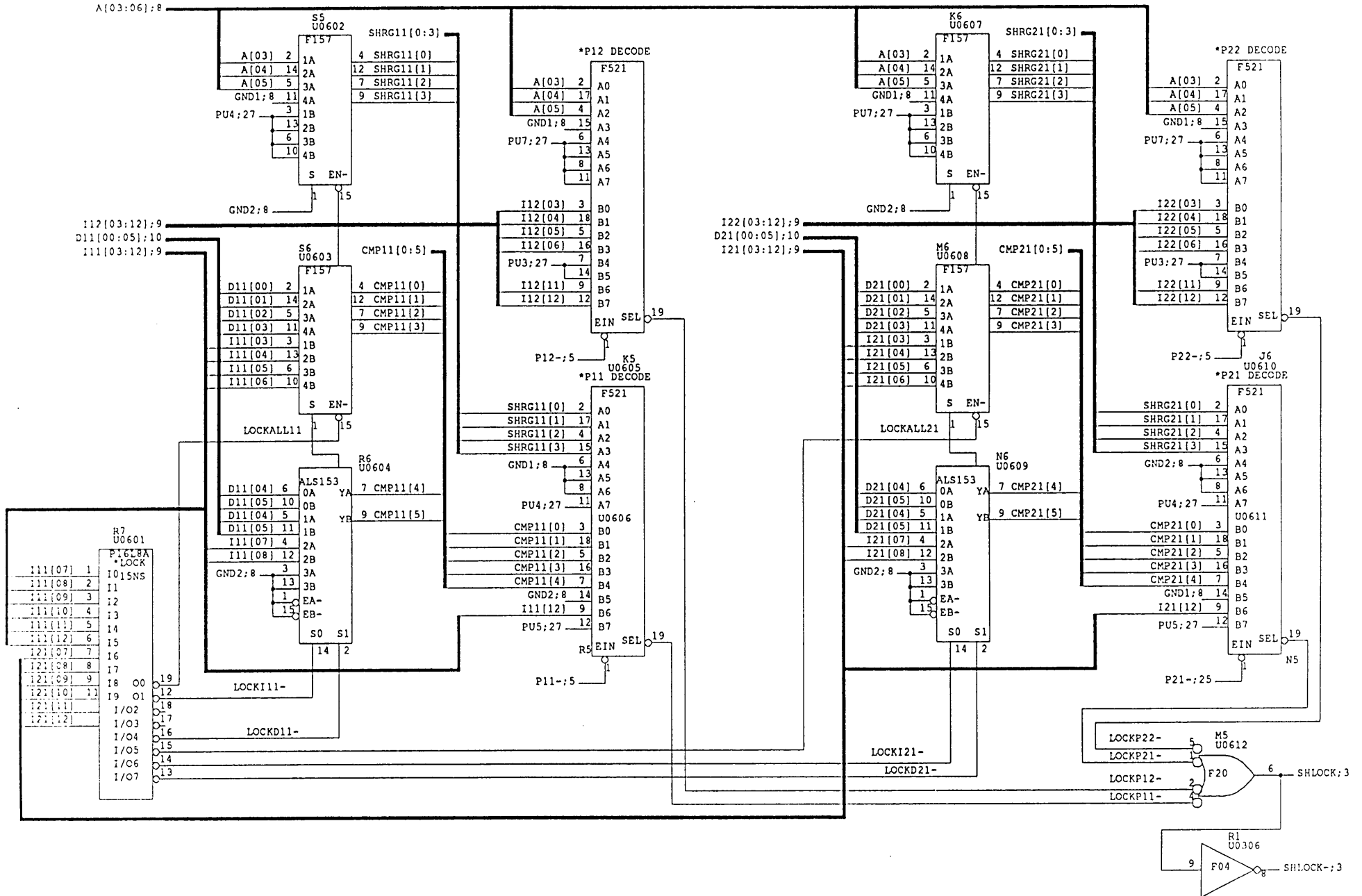
Title: FLOATING POINT ACCELERATOR
 Sheet: 5 OF 31
 Engineer: S CARRIE

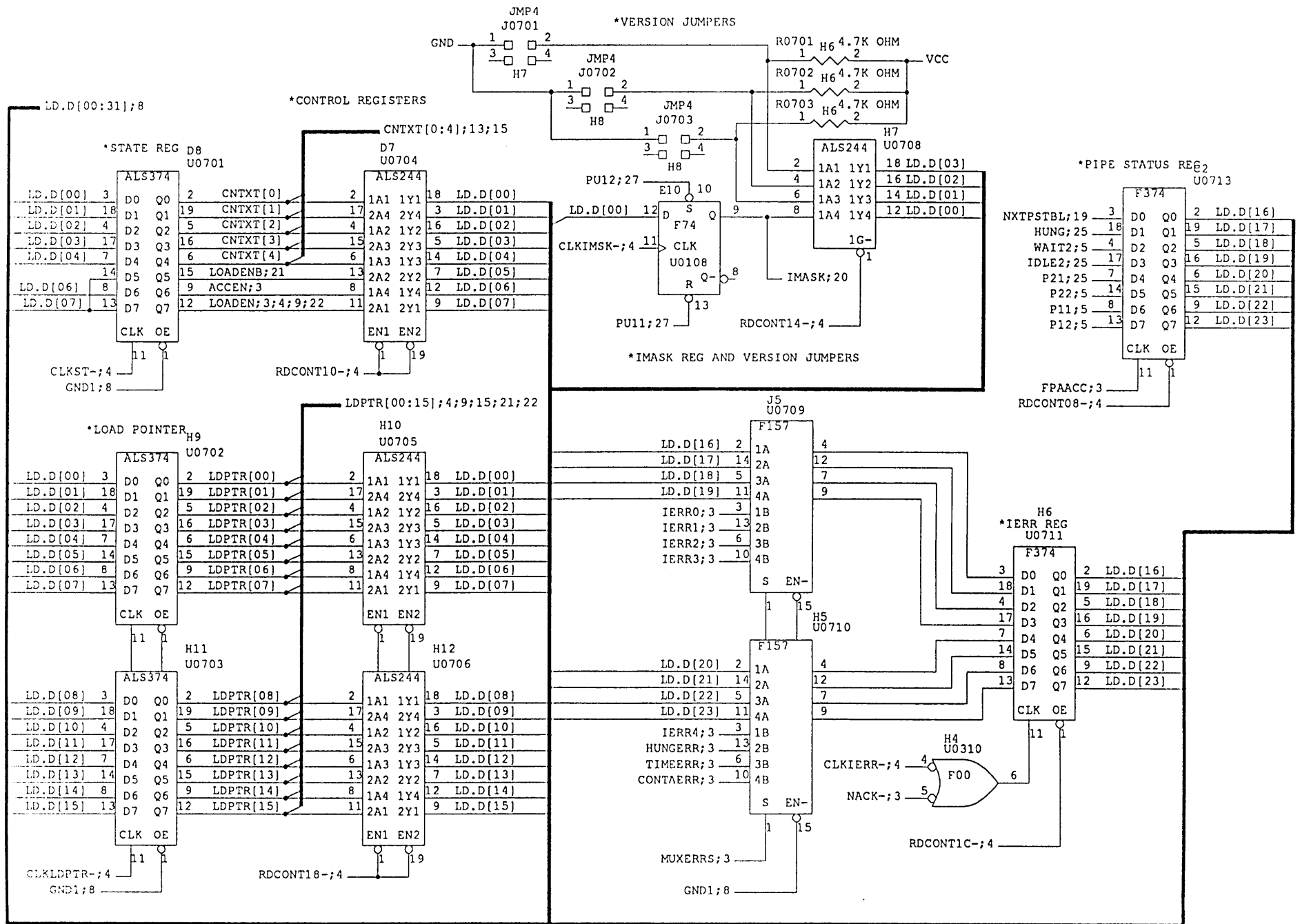
Drawing: 502-1105-01
 File: fpa5.d
 Date: Mon Aug 15 15:40:20 1988

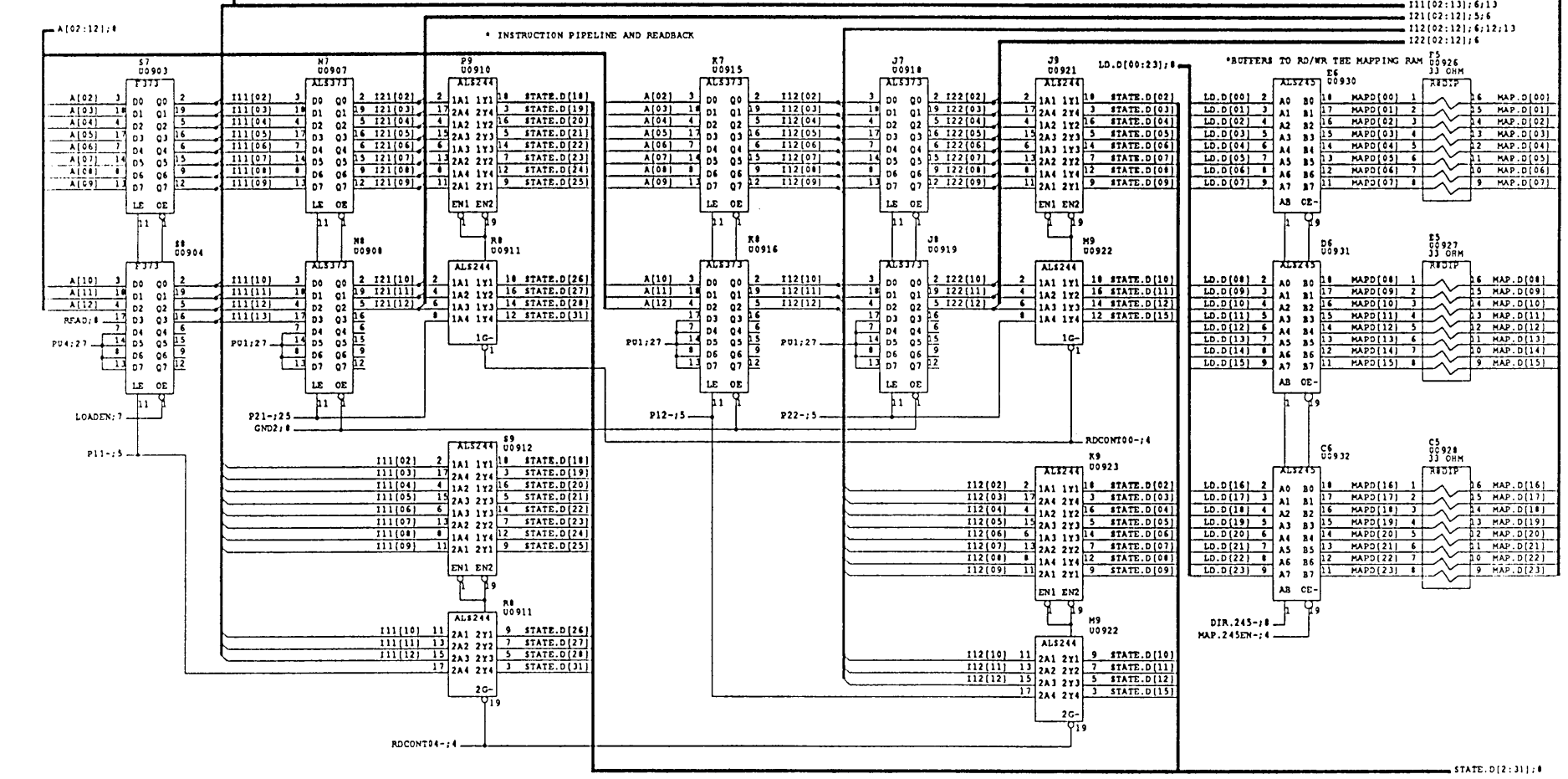
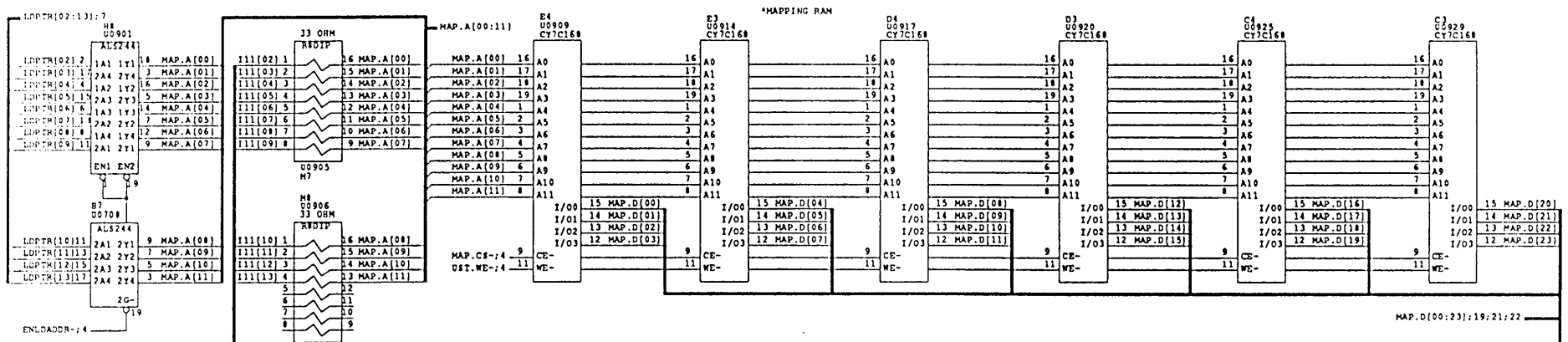
Prv: A

ECO	Description	Date	Approvals
-----	-------------	------	-----------

*SHADOW REGISTER INTERLOCK DECODING



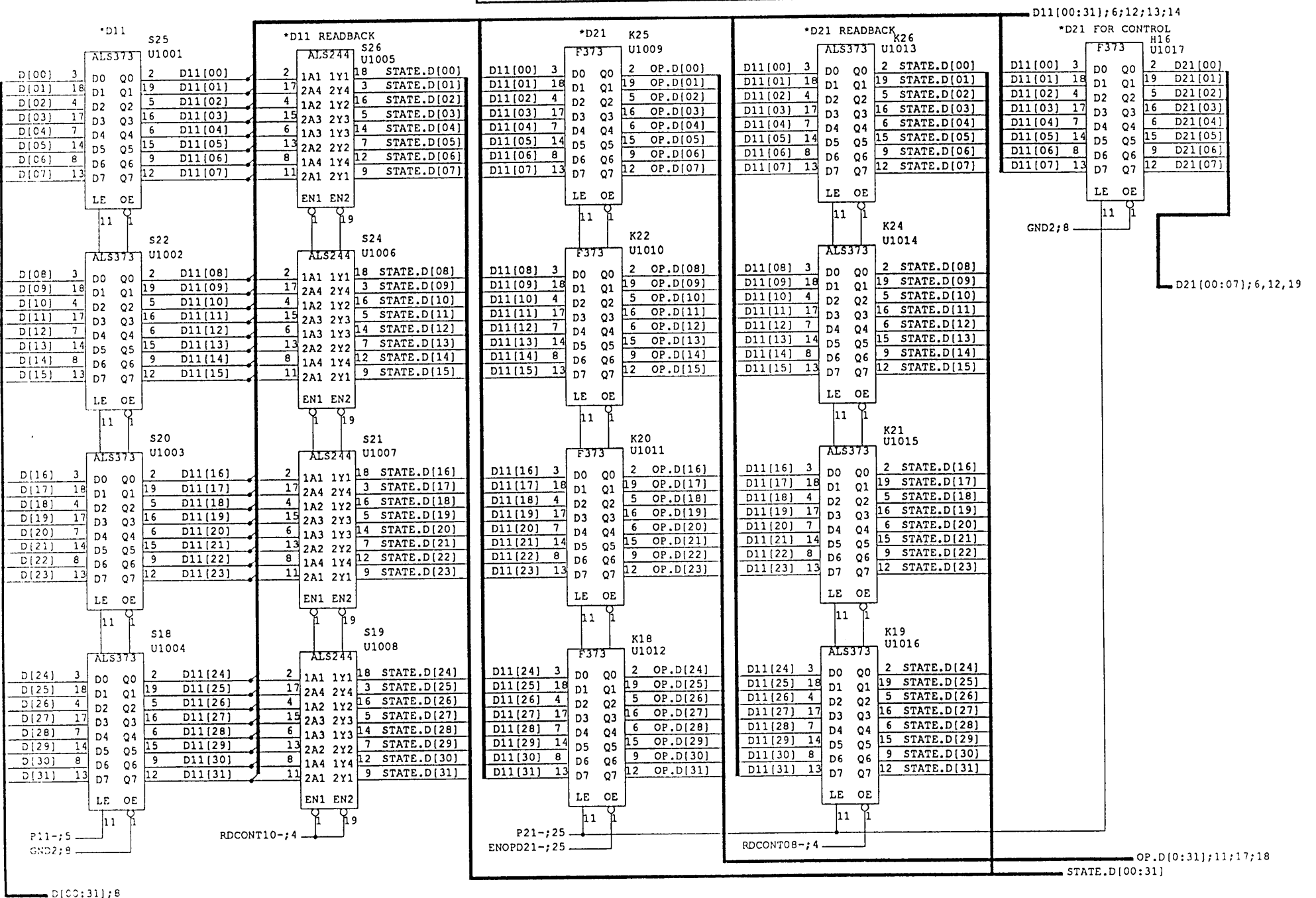




Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 9 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01 Rev: A
 File: fpa9.d
 Date: Mon Aug 15 15:43:47 1988

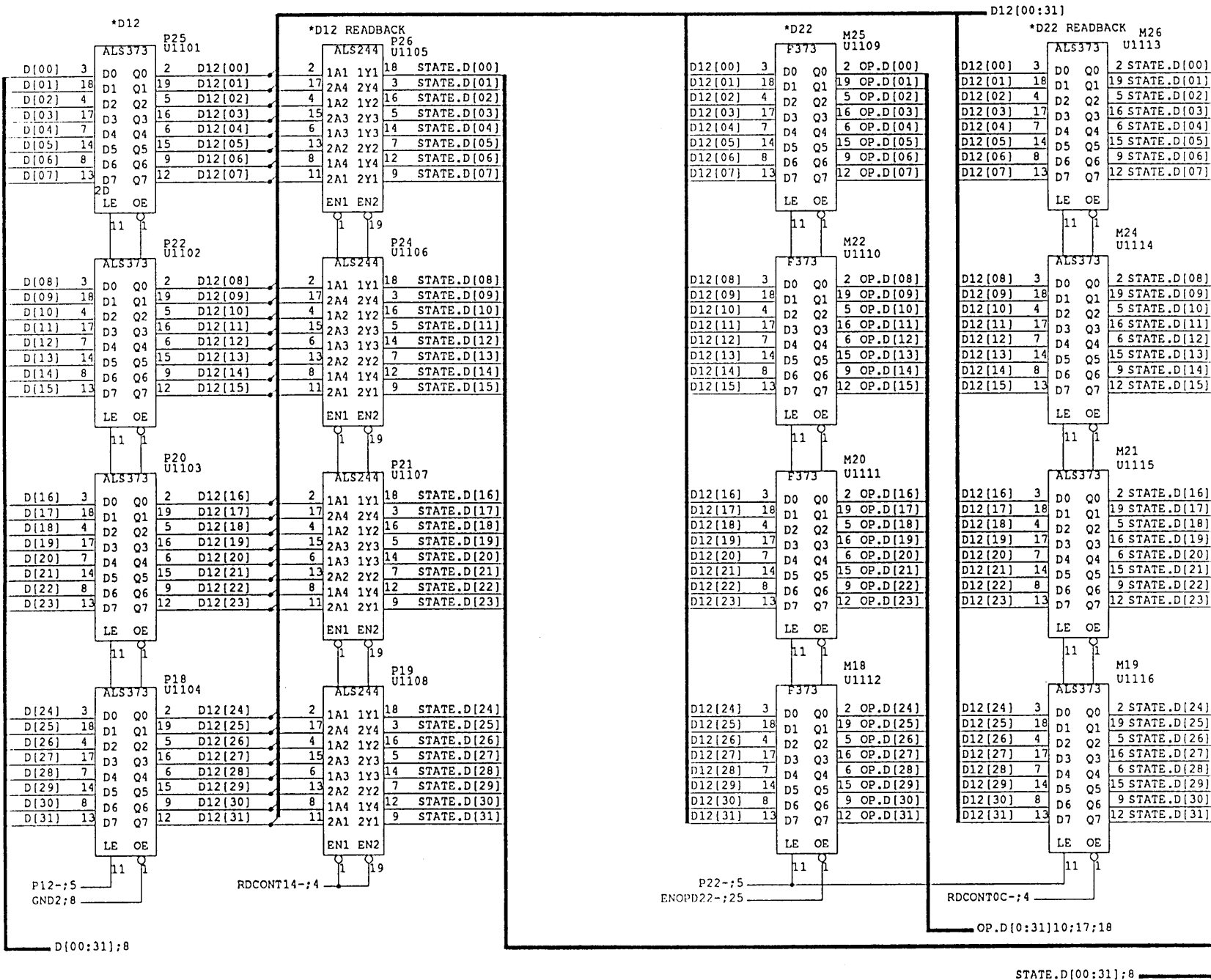


Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

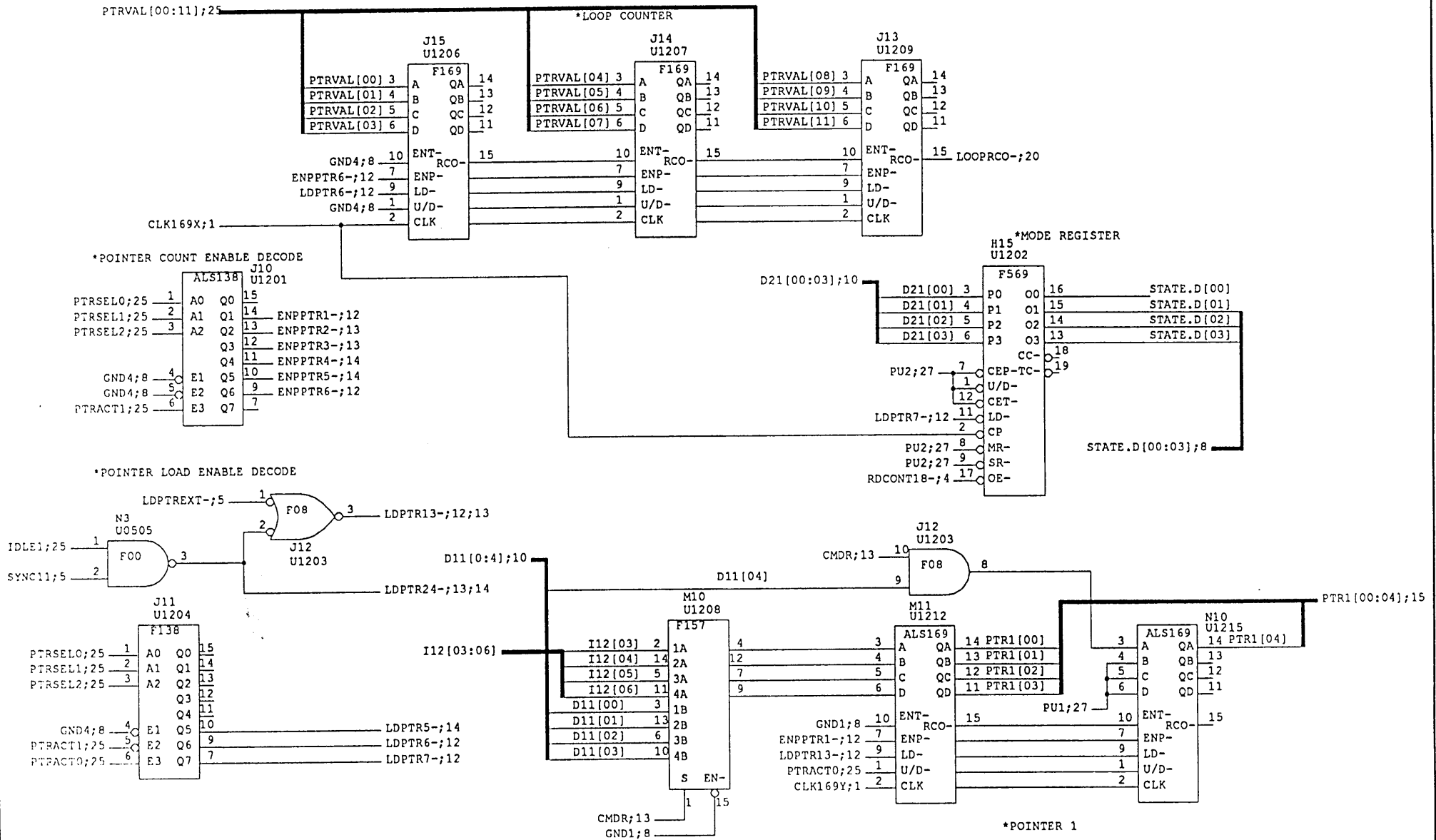
Title: FLOATING POINT ACCELERATOR
 Sheet: 10 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpa10.d
 Date: Mon Aug 15 15:26:06 1988

Rev: A



ECO	Description	Date	Approvals

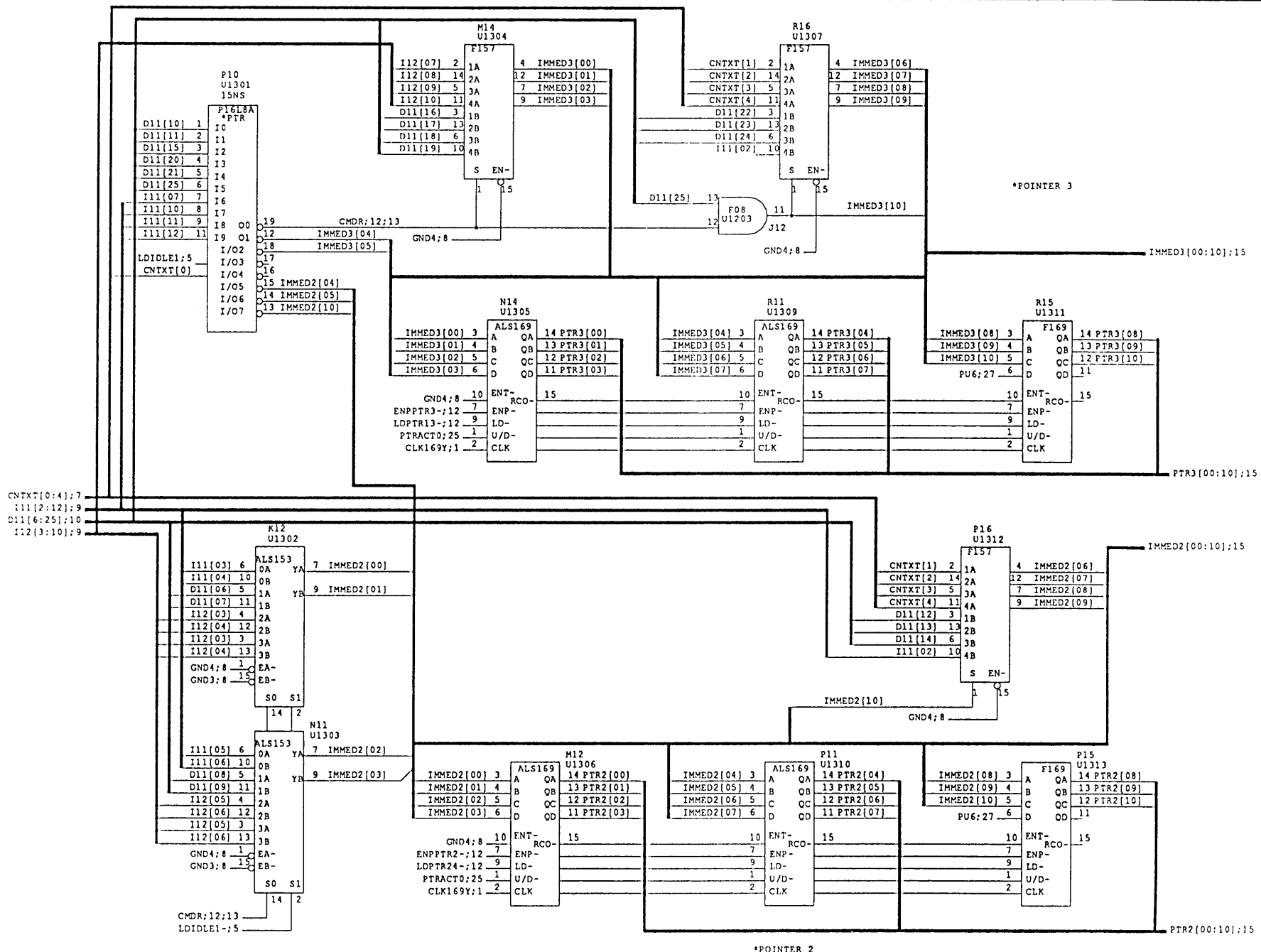


Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 12 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpal2.d
 Date: Mon Aug 15 15:27:05 1988

Rev: A

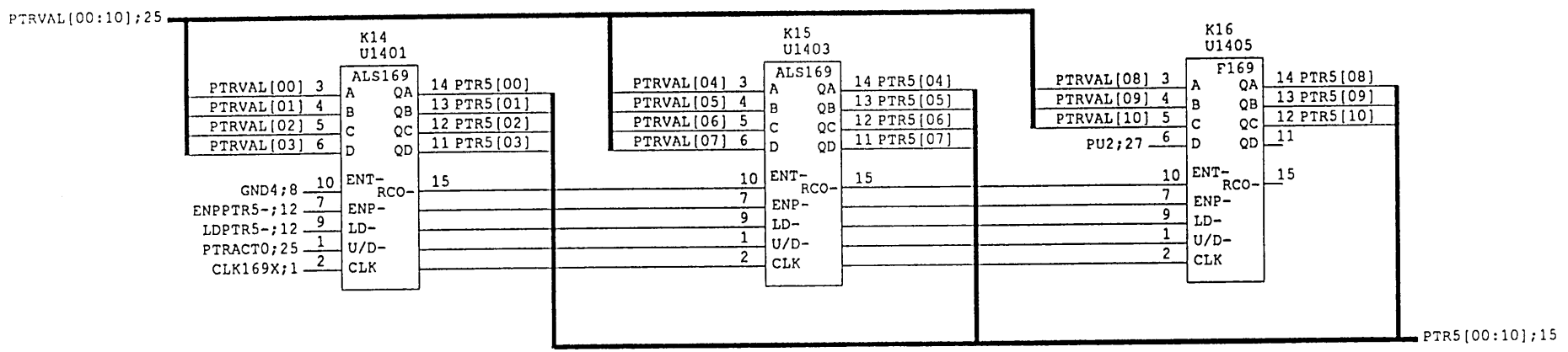


Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

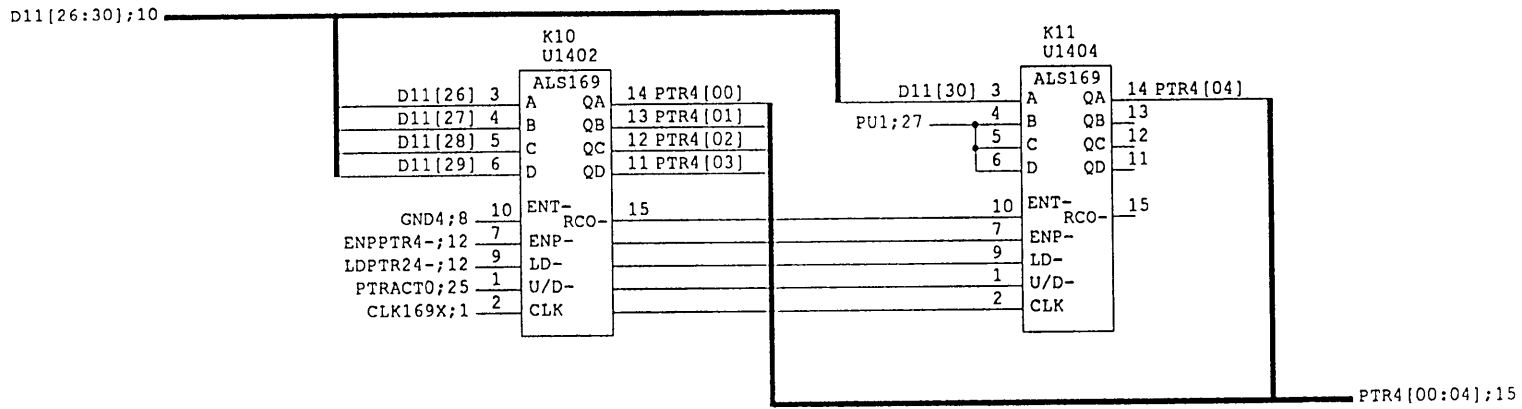
Title: FLOATING POINT ACCELERATOR
 Sheet: 13 OF 31
 Engineer: S CARRIE

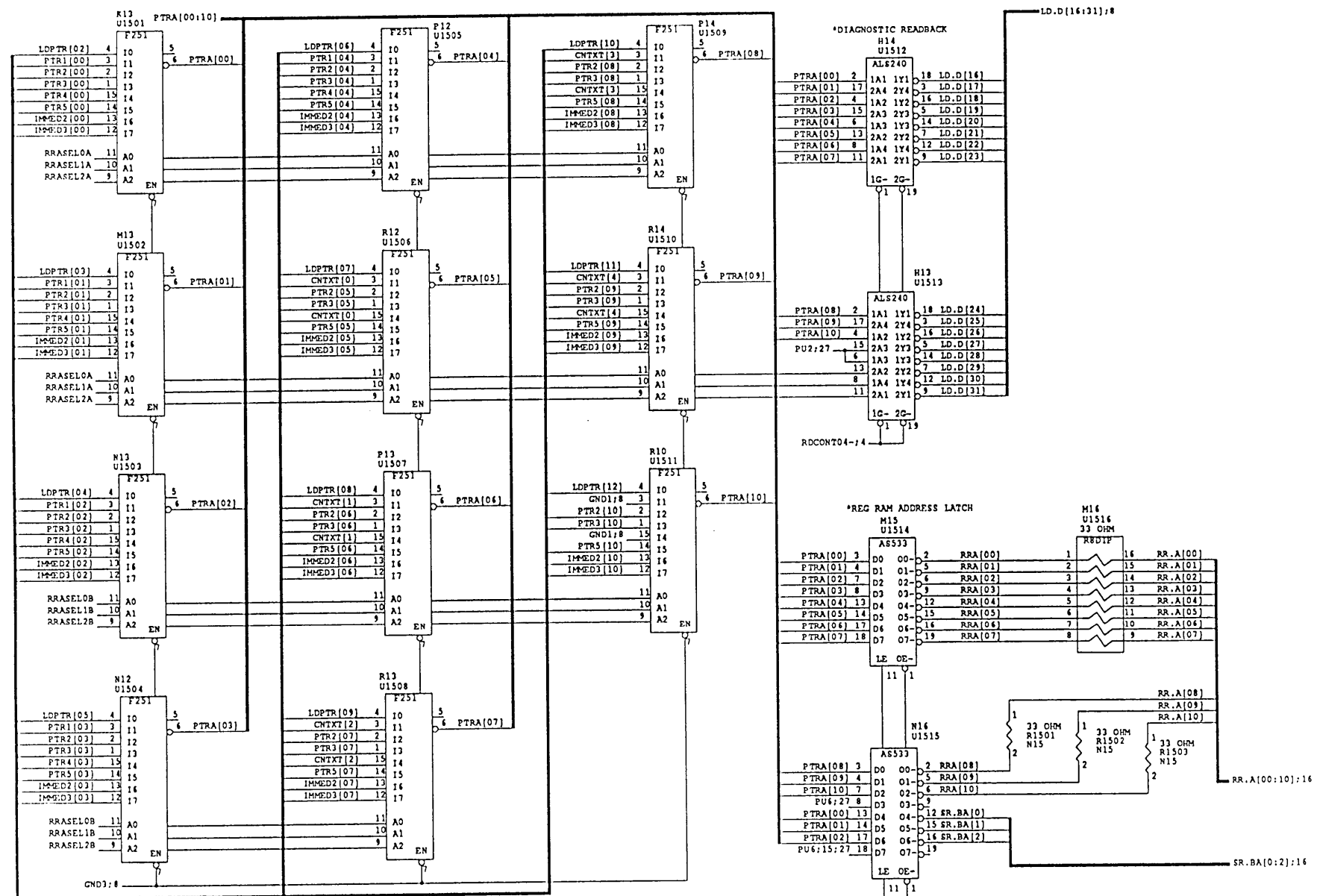
Drawing: 502-1105-01
 File: fpa13.d
 Date: Mon Aug 15 15:27:39 1986
 Rev: A

*POINTER 5



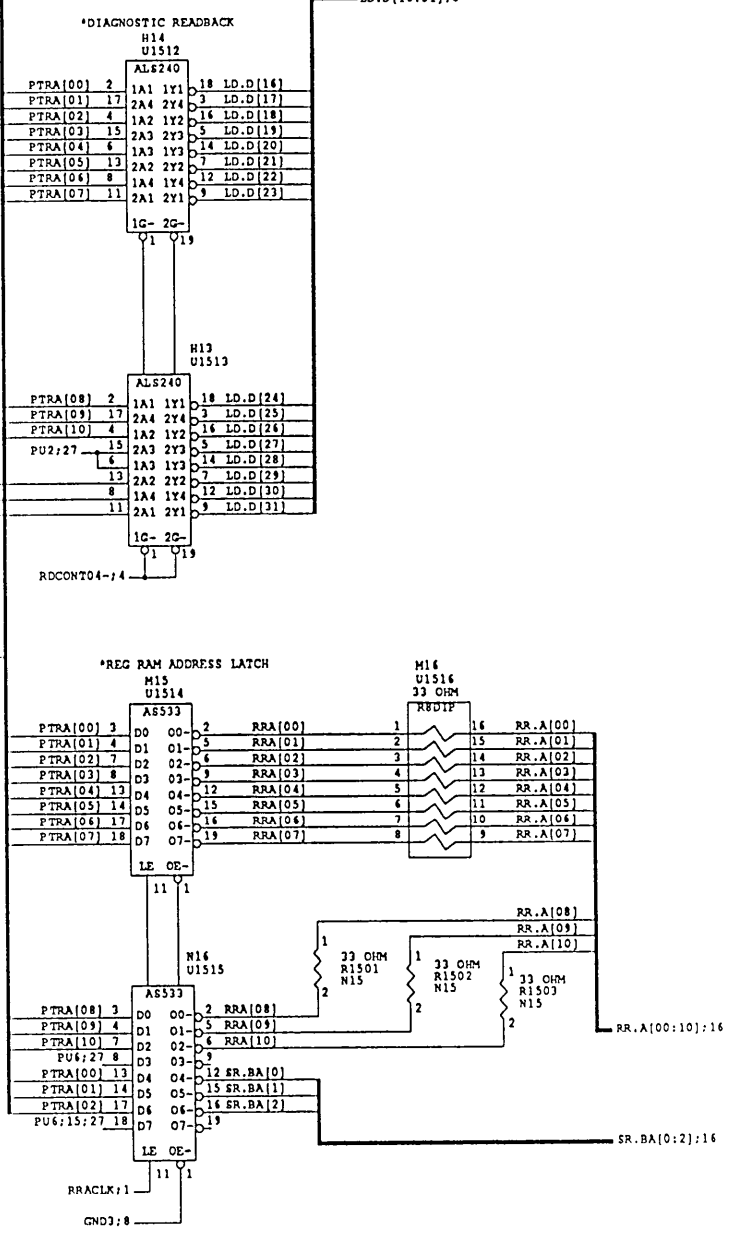
*POINTER 4





- LDPTR[02]:7
- PTR1[00]:10:12
- CNTXT[0:4]:7
- PTR2[00]:10:13
- PTR3[00]:10:13
- PTR4[00]:10:14
- PTR5[00]:10:14
- IMMED2[00]:10:13
- IMMED3[00]:10:13

*REGISTER RAM ADDRESS MULTIPLEXING

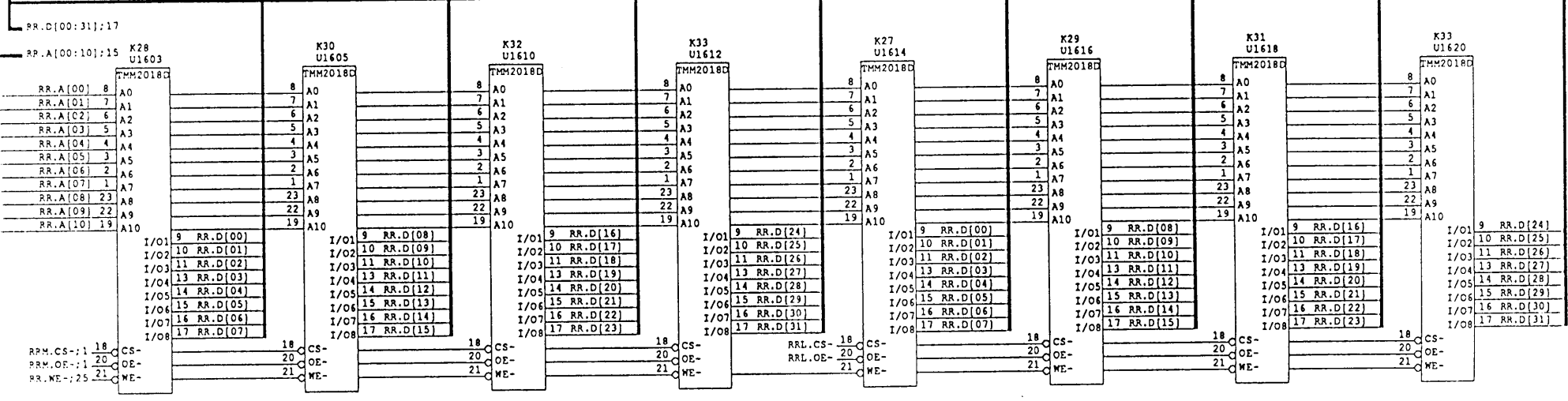
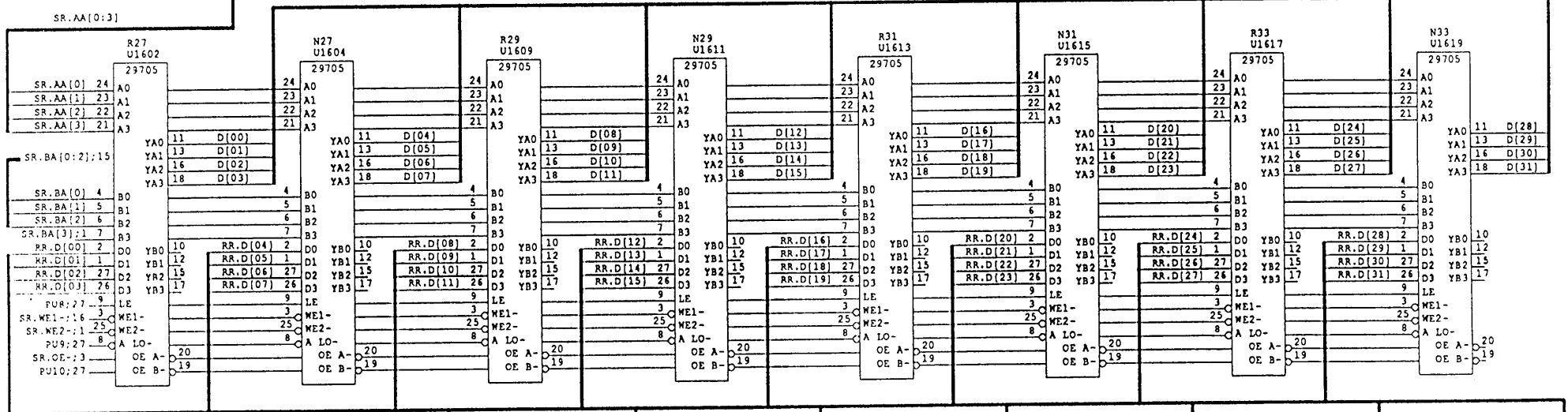
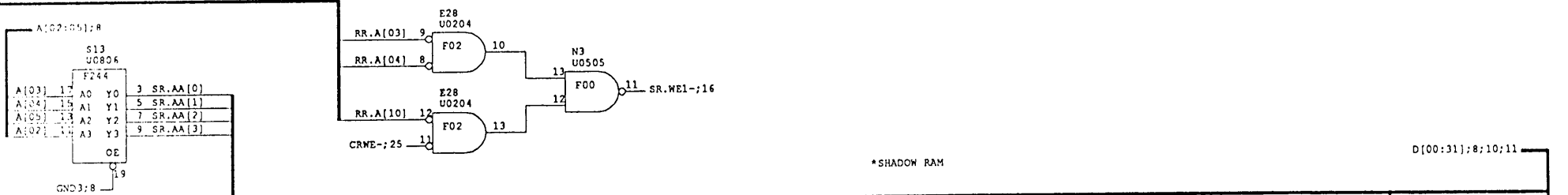


Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 15 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpals.d
 Date: Mon Aug 15 15:28:18

Rev: A



*REGISTER RAM - MOST SIGNIFICANT WORD

*REGISTER RAM - LEAST SIGNIFICANT WORD

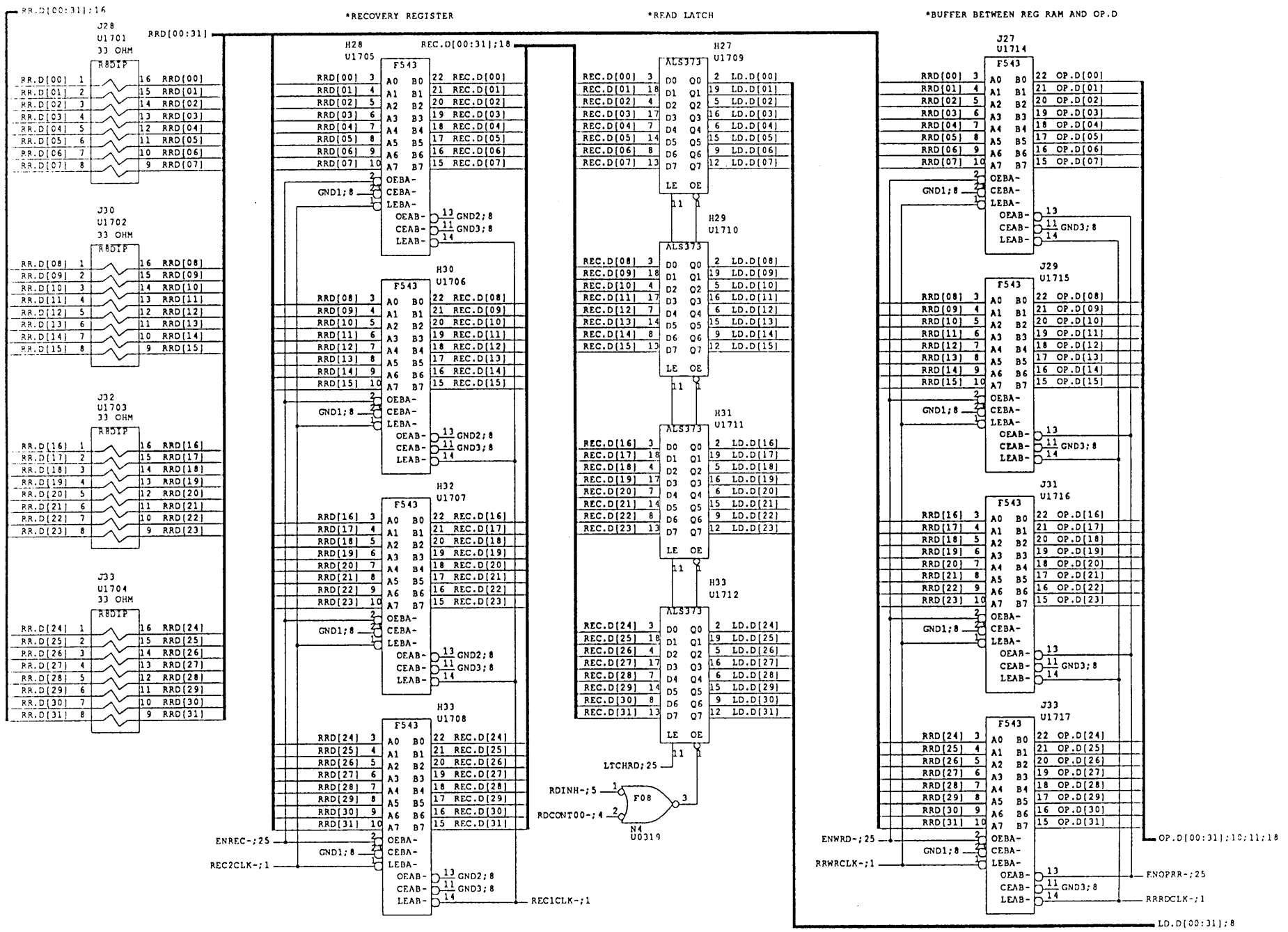


Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 16 OF 31
 Engineer: S CARRIE

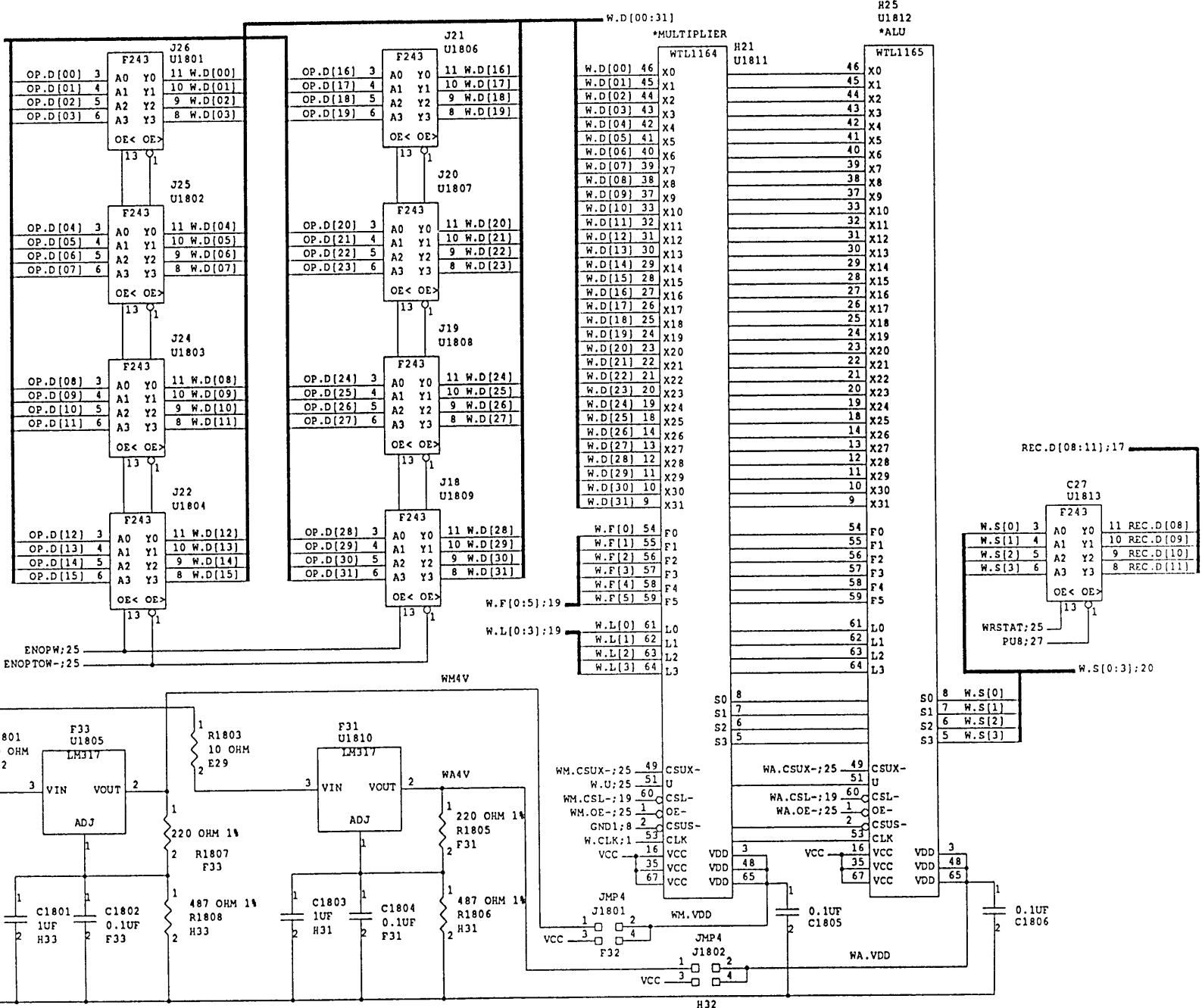
Drawing: 502-1105-01
 File: fpal6.d
 Date: Mon Aug 15 15:28:57 1988

Rev: A



ECO	Description	Date	Approvals

CP.D[00:31]:10:11:17



Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

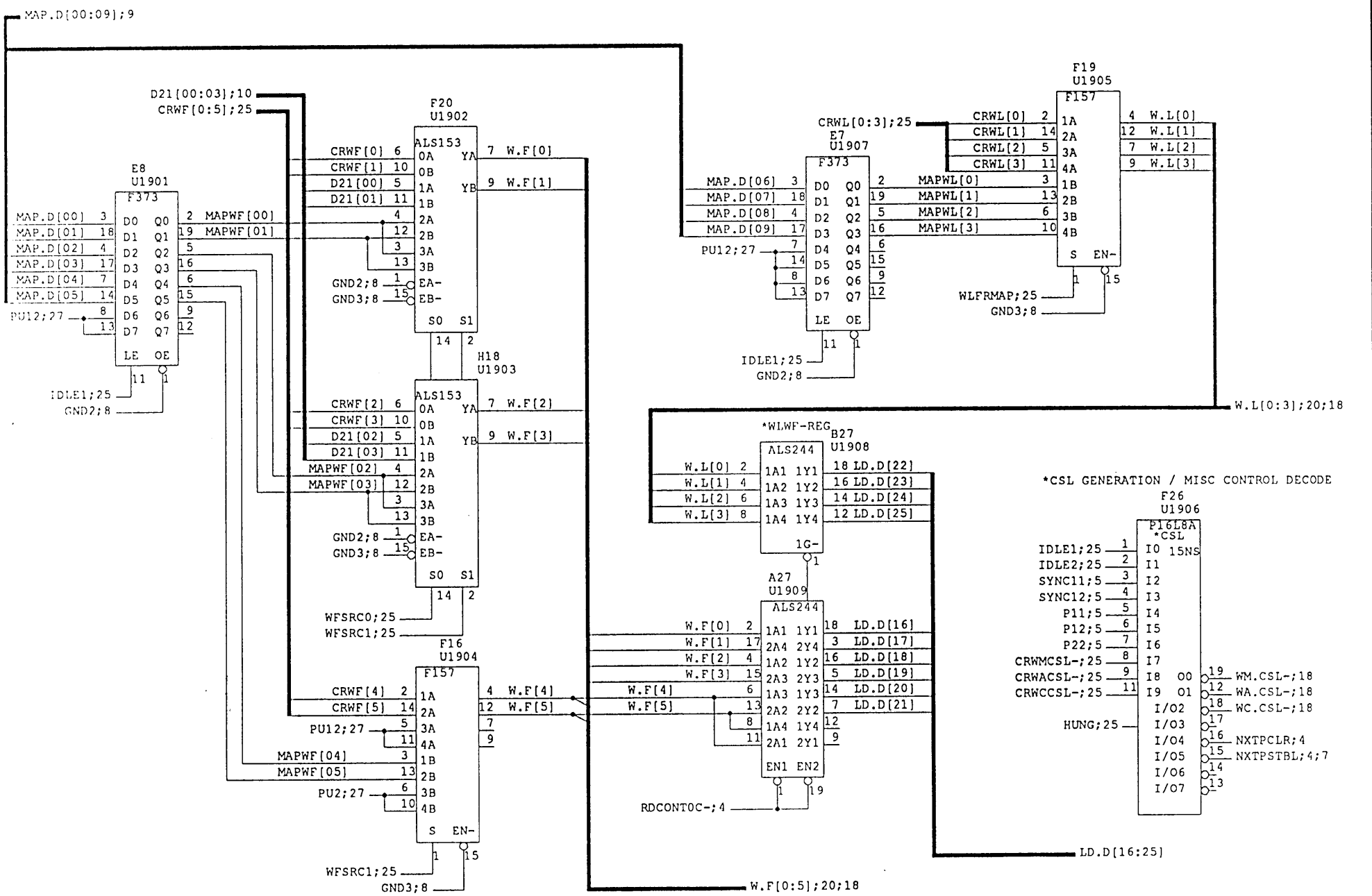
Title: FLOATING POINT ACCELERATOR
 Sheet: 18 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpal8.d
 Date: Mon Aug 15 15:30:02 1988

Rev: A

*F+ MUXING: OPERAND/USTORE/MAPPING RAM --> WEITEK CHIPS

*L+ MUXING: USTORE/MAPPING RAM --> WEITEK CHIPS



Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

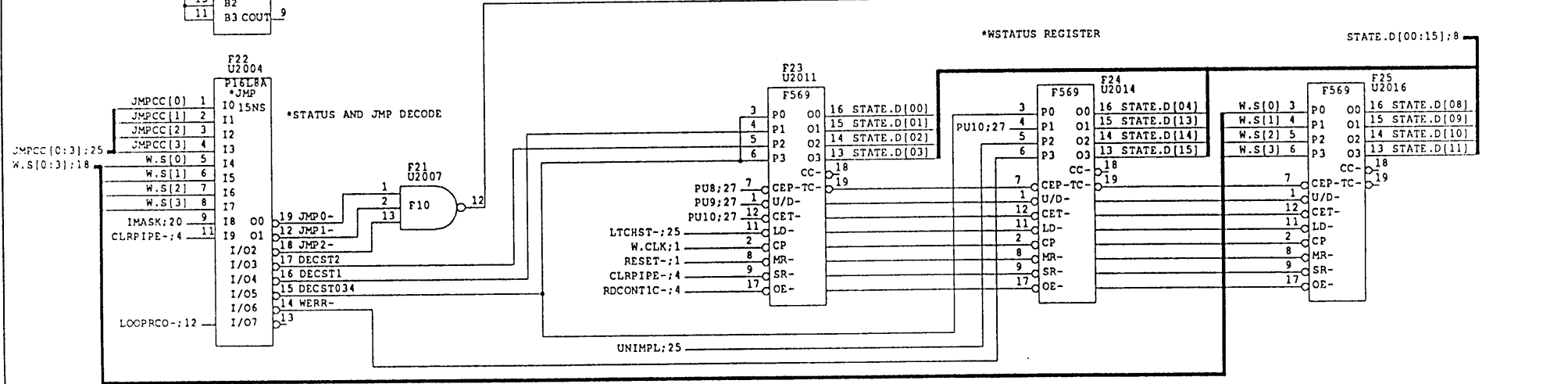
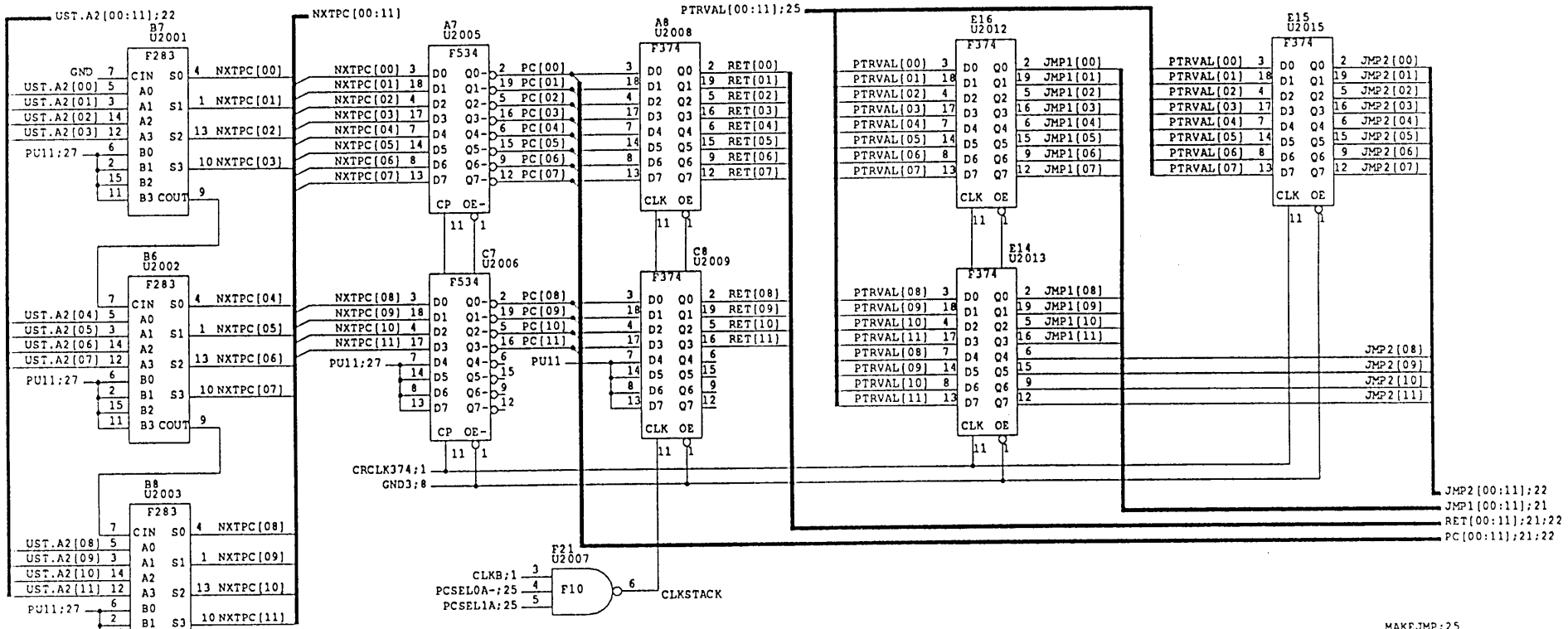
Title: FLOATING POINT ACCELERATOR
 Sheet: 19 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpal9.d
 Date: Mon Aug 15 15:50:32 1988

Rev: A

ECO	Description	Date	Approvals
-----	-------------	------	-----------

*PC DECREMENT *PROGRAM COUNTER *1 DEEP CALL/RET STACK *ADDRESS FOR DELAYED CONDITIONAL BRANCH



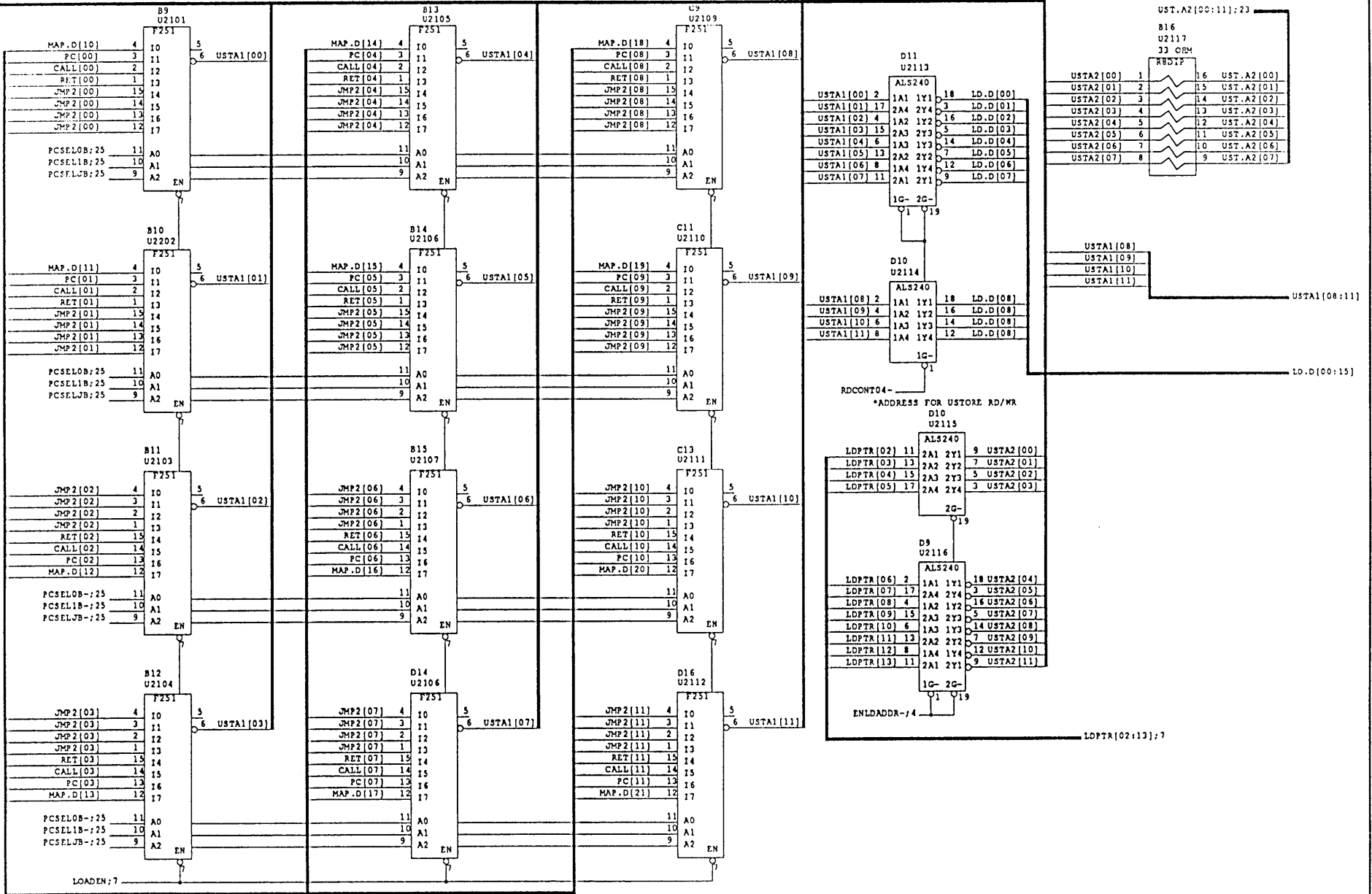
Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 20 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpa20.d
 Date: Mon Aug 15 15:33:14 1988

Rev: A

UST.A2[00:11]



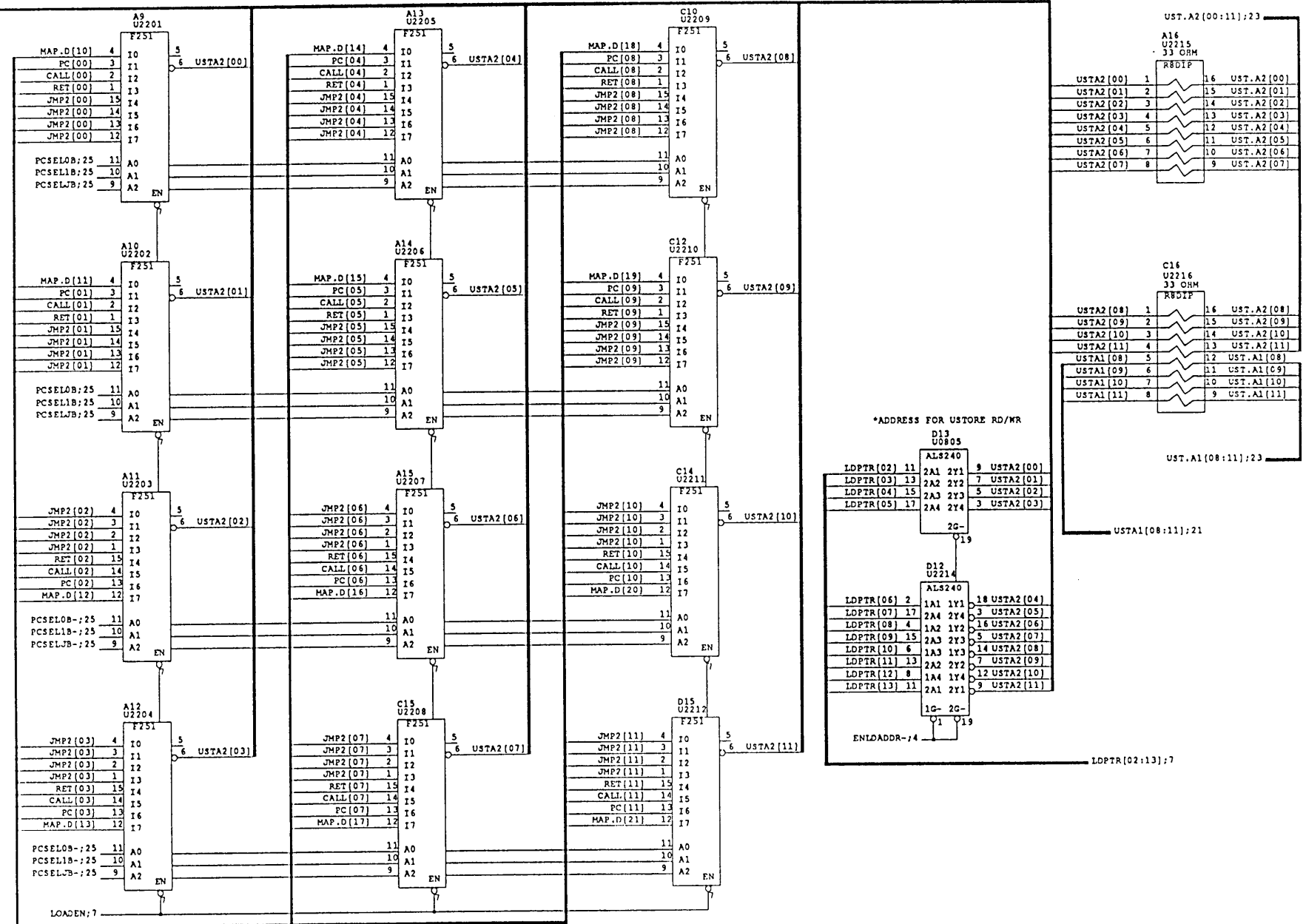
Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 21 OF 31
 Engineer:

Drawing: 502-1105-01
 File: fpa21.d
 Date: Mon Aug 15 15:33:40 1988

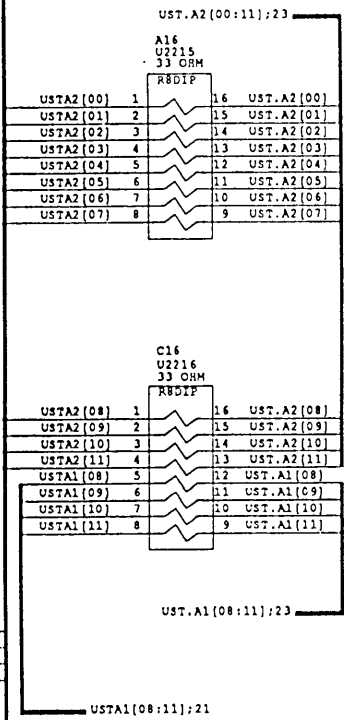
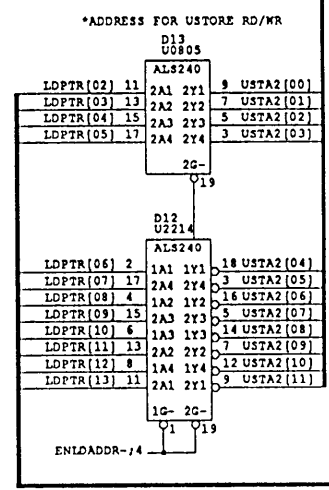
Rev: A

USTA2[00:11]

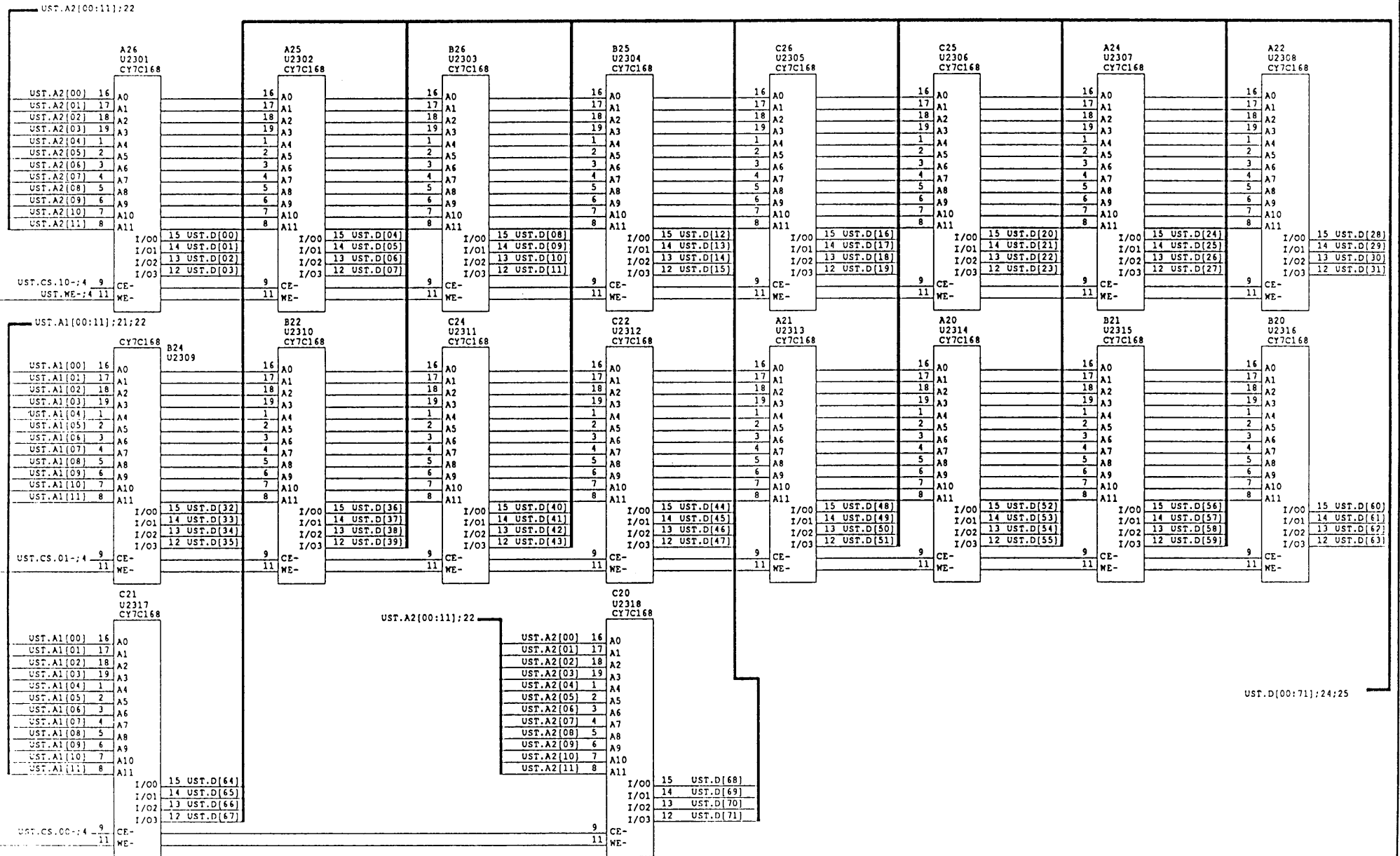


- MAP.D[10:23]
- PC[00:11]
- CALL[00:11]
- RET[00:11]
- JMP2[00:11]

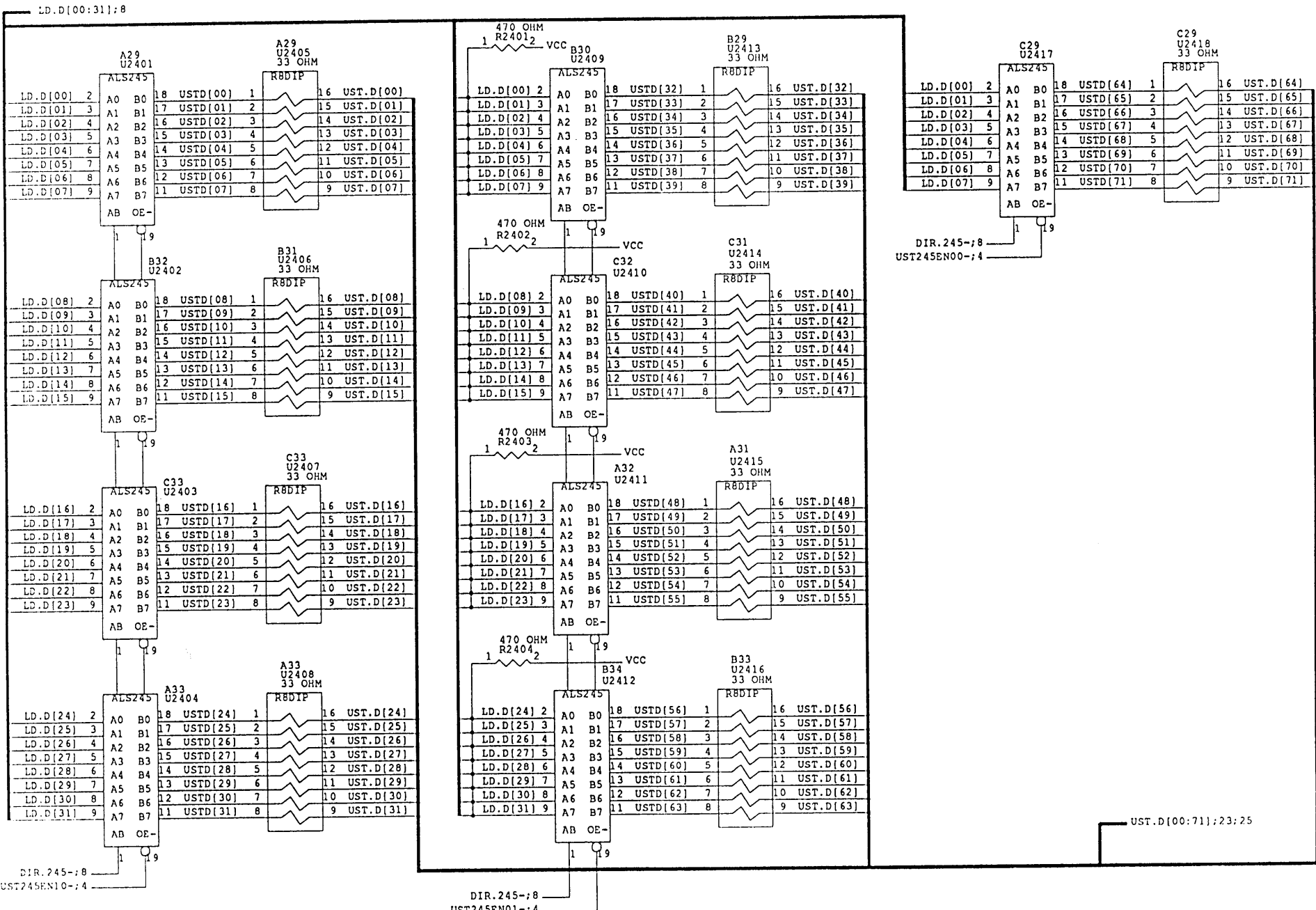
*ADDRESS MULTIPLEXING FOR 1/2 OF USTORE RAM



*STORE RAM

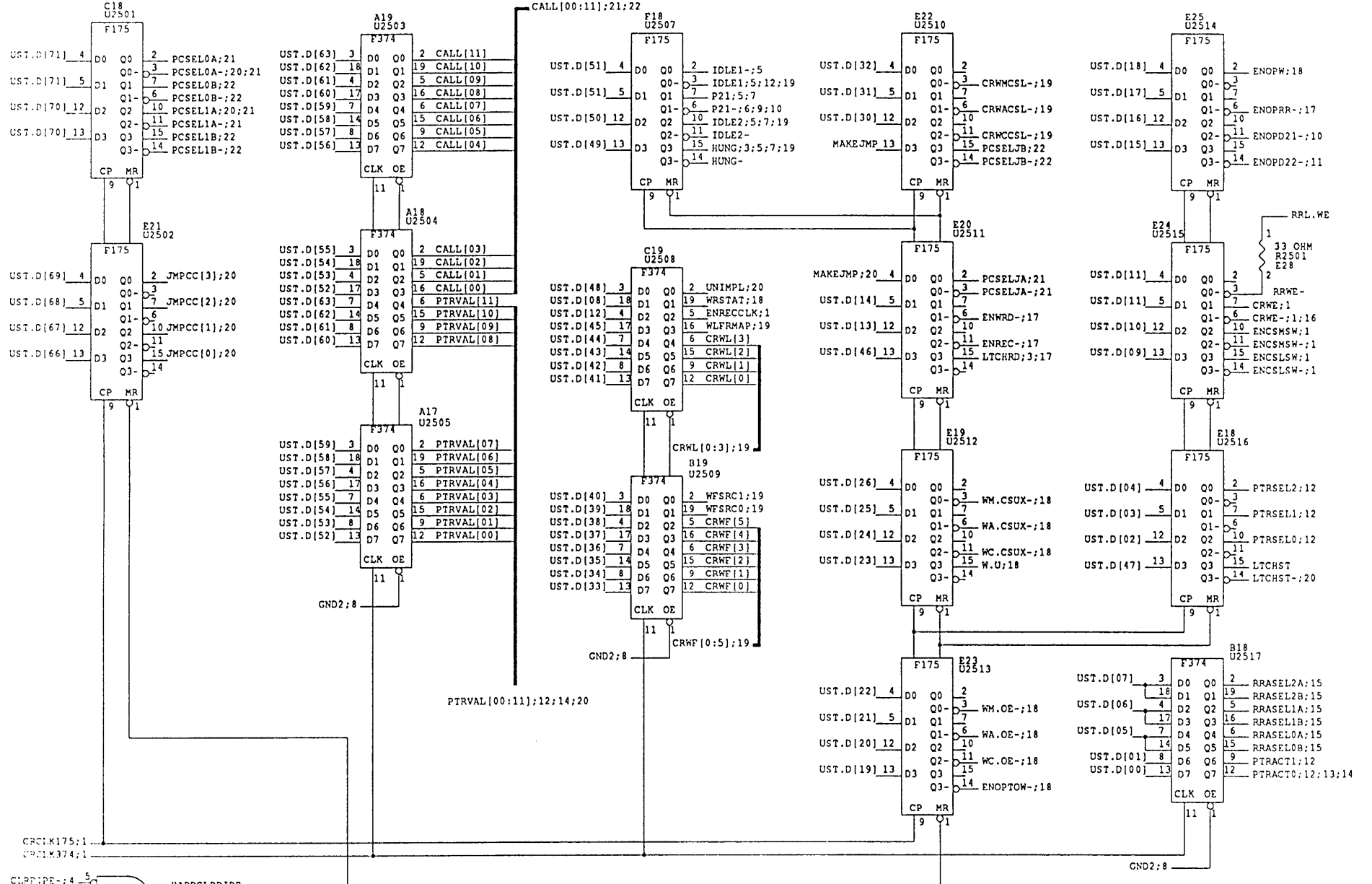


*BUFFERS USED TO RD/WR THE USTORE RAM

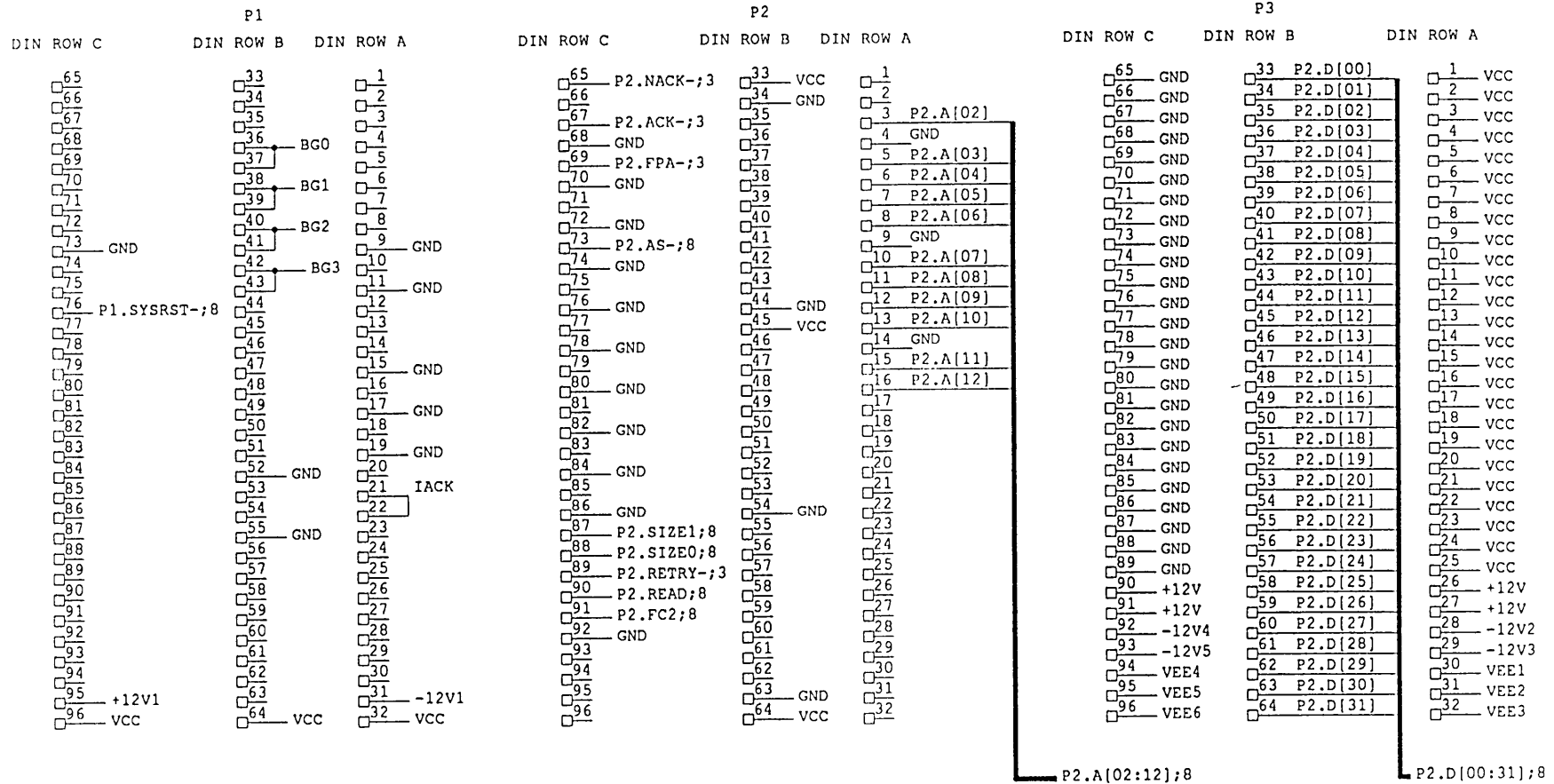


*USTORE COMMAND REGISTER

*USTORE COMMAND REGISTER



ECO	Description	Date	Approvals
-----	-------------	------	-----------



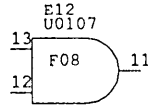
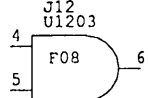
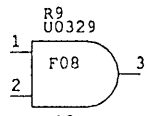
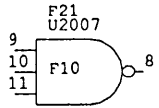
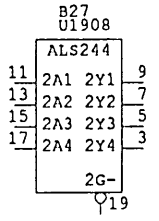
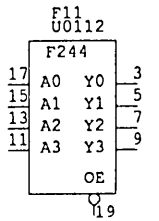
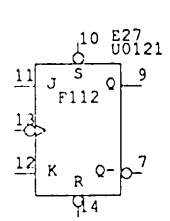
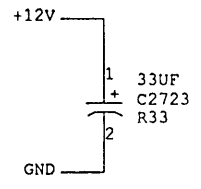
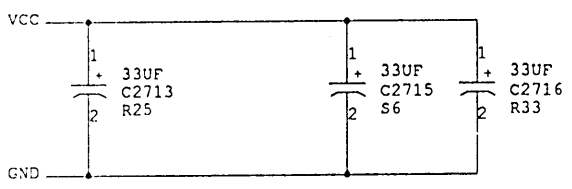
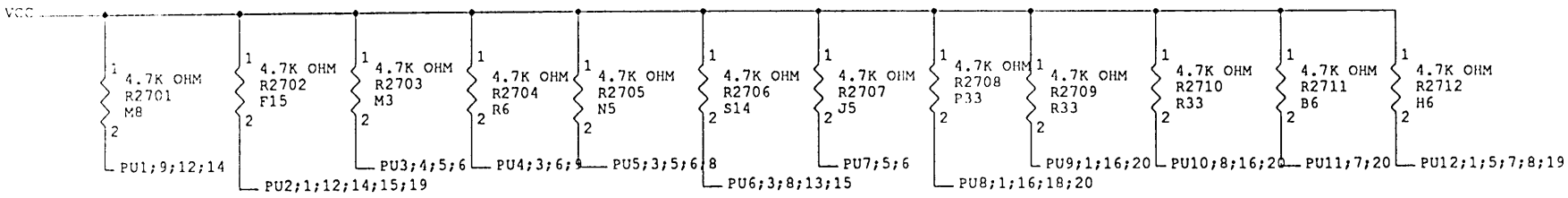
Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystem Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 26 OF 31
 Engineer: S CARRIE

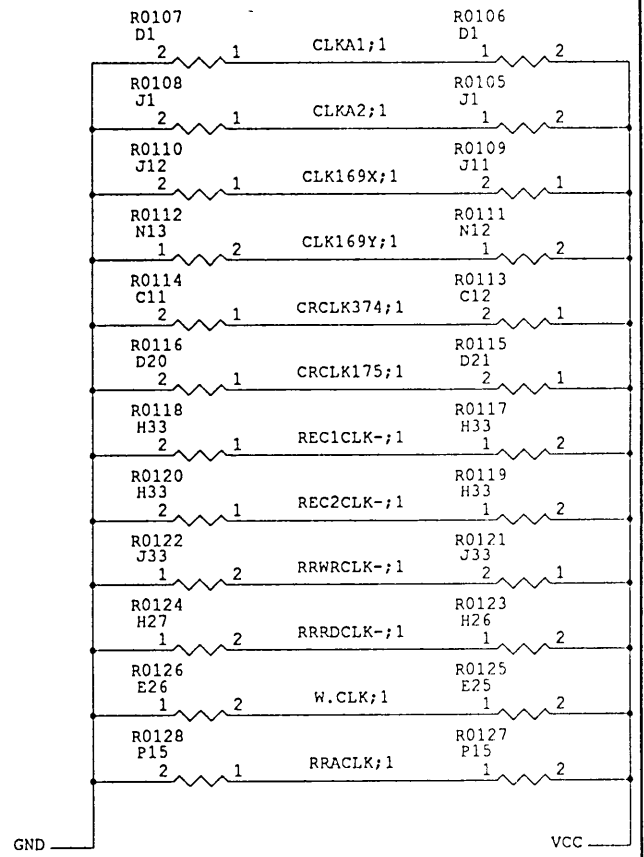
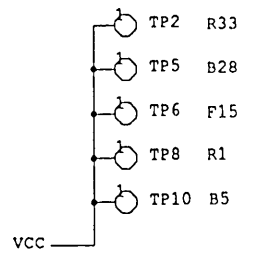
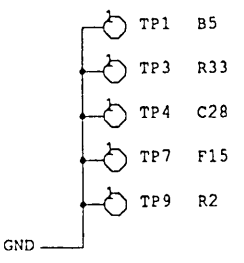
Drawing: 502-1105-01
 File: fpa26.d
 Date: Mon Aug 15 15:36:00 1988

Rev: A

ECO	Description	Date	Approvals
-----	-------------	------	-----------



*SPARES AND TEST POINTS



*SPACE FOR THESE TERMINATION RESISTORS WAS MADE
 *AVAILABLE IN CASE TERMINATION WAS NEEDED
 *BECAUSE TERMINATION WAS NOT NECESSARY THESE
 *RESISTORS SHOULD NOT BE LOADED

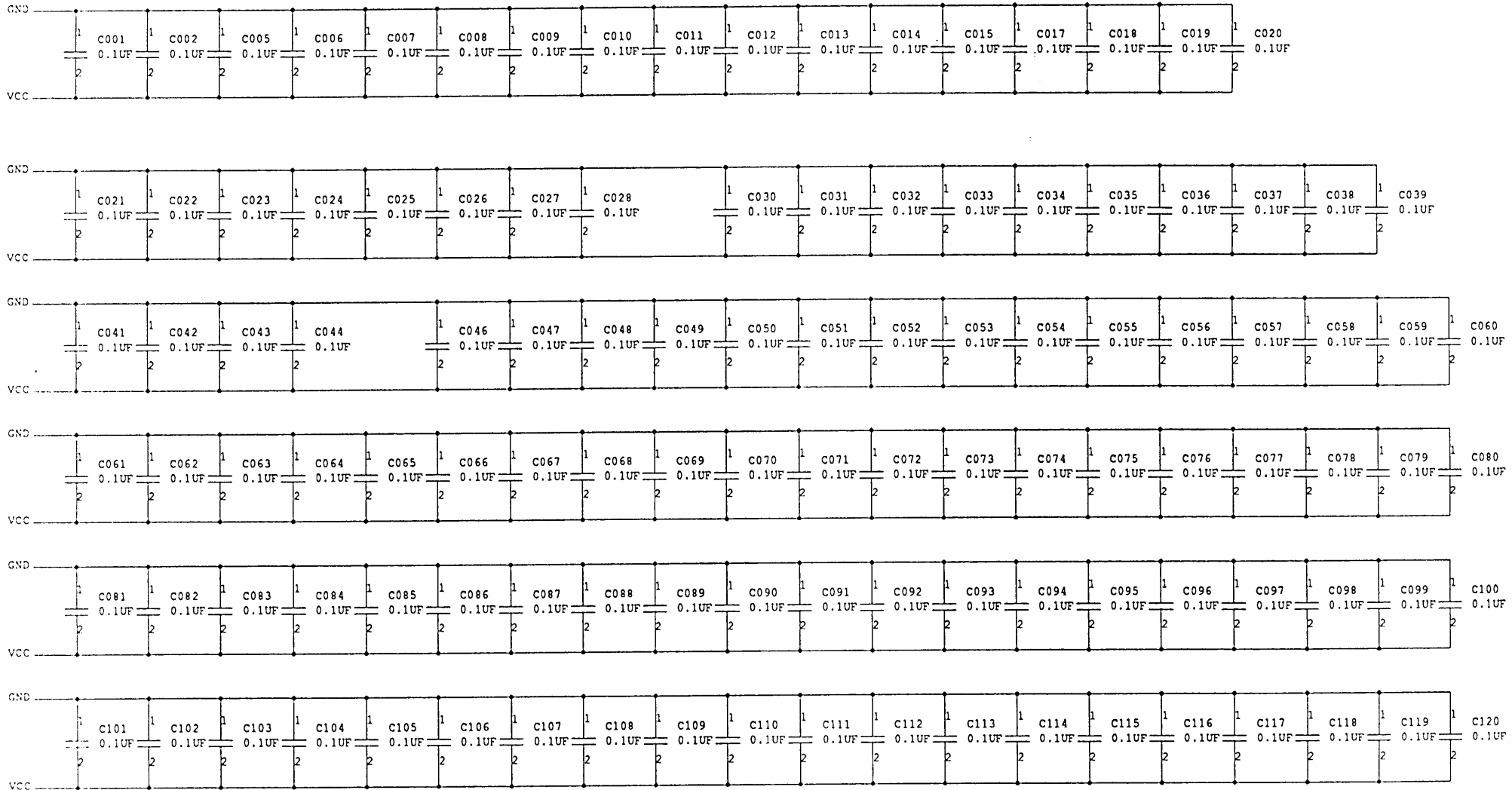


Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 27 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpa27.d
 Date: Mon Aug 15 15:36:22 1
 Rev: A

ECO	Description	Date	Approvals



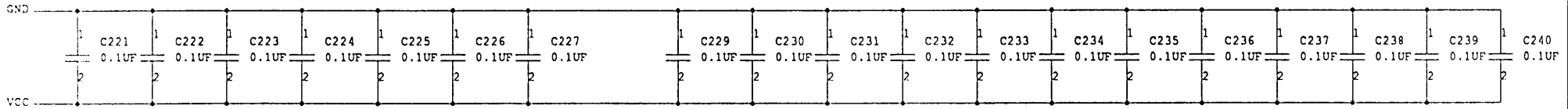
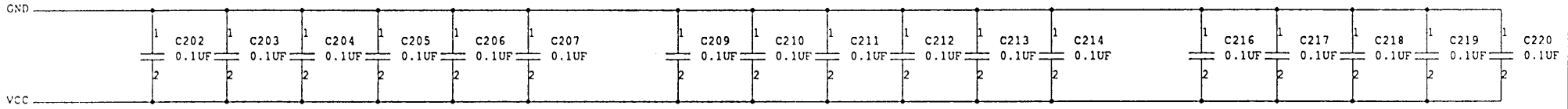
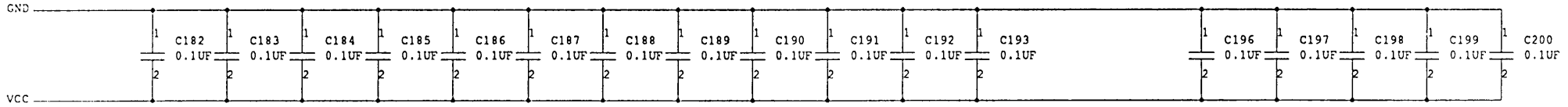
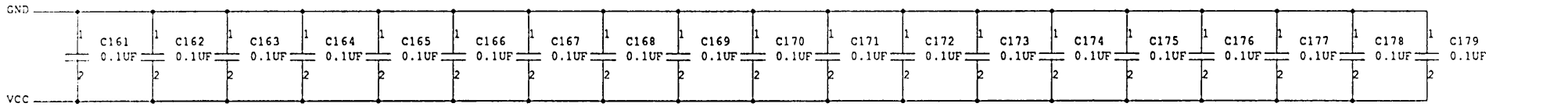
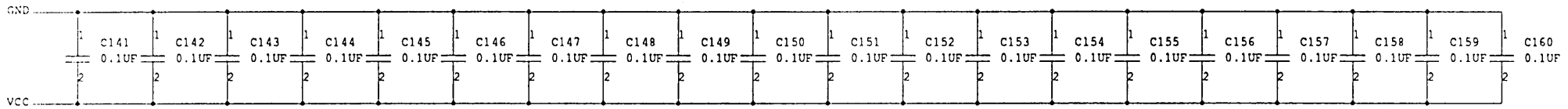
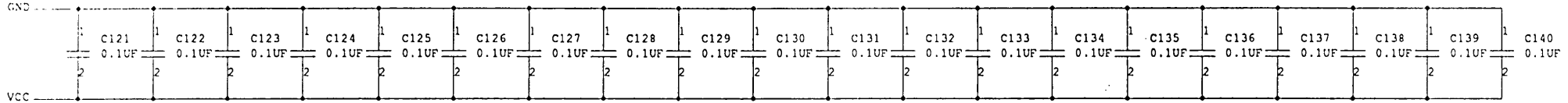
Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved.
 This material contains trade secrets information of Sun Microsystems Inc.
 Use, reproduction or disclosure is prohibited except under written license
 by Sun Microsystems Inc. Use of copyright notice is precautionary only
 and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 28 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpa28.d
 Date: Mon Aug 15 15:36:47 1988

Rev: A

ECO	Description	Date	Approvals
-----	-------------	------	-----------



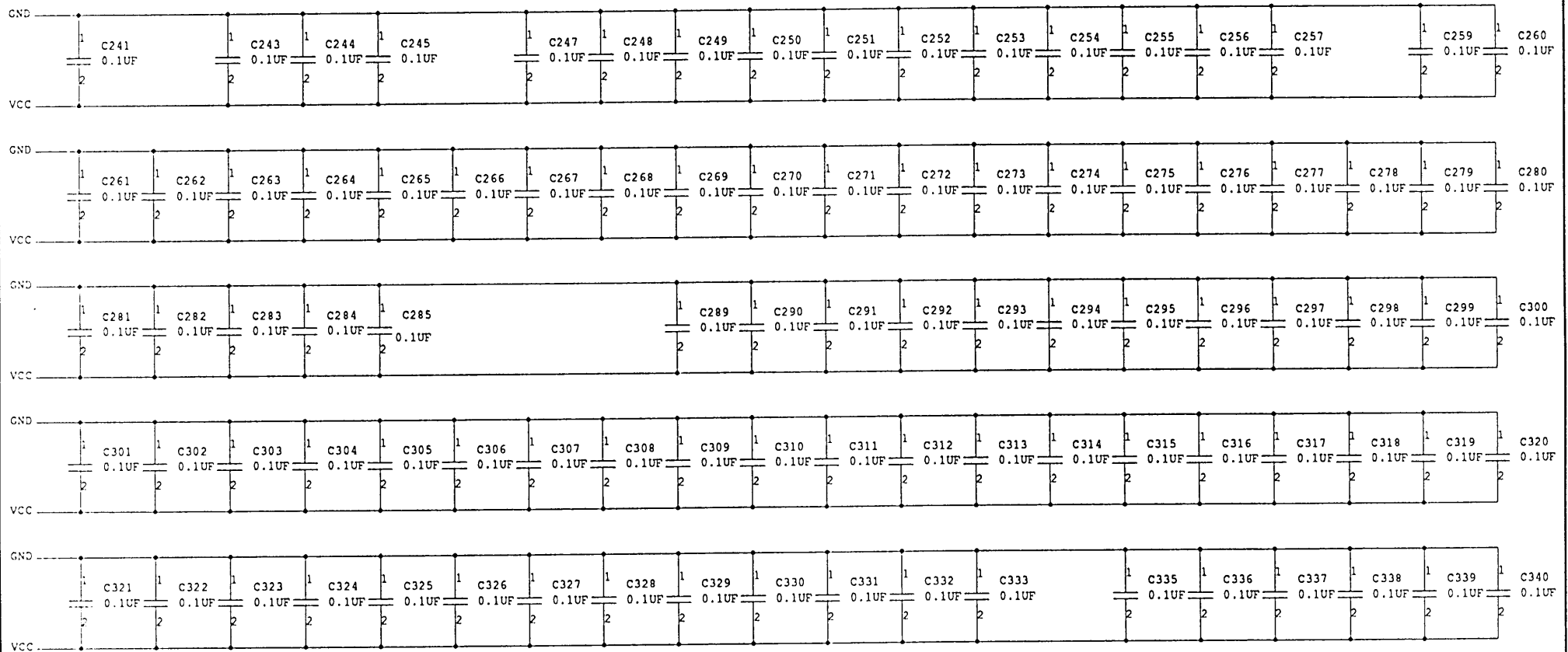
Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystem Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 29 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpa29.d
 Date: Mon Aug 15 15:37:10 1988

Rev: A

ECO	Description	Date	Approvals



Confidential - Copyright (C) 1986 Sun Microsystems. All rights reserved. This material contains trade secrets information of Sun Microsystems Inc. Use, reproduction or disclosure is prohibited except under written license by Sun Microsystems Inc. Use of copyright notice is precautionary only and does not imply publication or disclosure.

Title: FLOATING POINT ACCELERATOR
 Sheet: 30 OF 31
 Engineer: S CARRIE

Drawing: 502-1105-01
 File: fpa30.d
 Date: Mon Aug 15 15:38:49 1988

Rev: A

THIS TABLE INDICATES THE PAGE ON WHICH EACH GATE 1/2 BUFFER ETC. OF A SECTIONED COMPONENT IS LOCATED -- THE SECTION REFERS TO THE SECTIONS AS DEFINED BY THE CADROID LIBRARY

COMP TYPE	DESIG	SECTION					COMP TYPE	DESIG	SECTION		
		0	1	2	3	4			5	0	1
F00	U0104	01	01	01	01		F74	U0108	01	07	
F00	U0114	01	01	01	08		F74	U0207	02	02	
F00	U0310	03	07	03	04		F74	U0209	02	03	
F00	U0502	05	05	05	05		F74	U0307	03	03	
F00	U0505	12	05	05	16		F74	U0320	03	03	
F02	U0102	01	02	01	01		F74	U0322	03	03	
F02	U0204	25	25	16	16		F74	U0510	05	05	
F02	U0303	03	03	04	04		F74	U0512	05	05	
F02	U0514	05	05	05	05		F74	U0518	05	05	
F04	U0306	03	03	04	06	05	04	F74	U0519	05	05
F08	U0107	01	01	02	27		F74	U0520	05	05	
F08	U0319	17	04	04	03		F74	U0524	05	05	
F08	U0329	27	03	03	03		F74	U0531	05	05	
F08	U0414	04	04	04	04		F112	U0120	01	01	
F08	U1203	12	27	12	13		F112	U0121	01	27	
F10	U0117	01	01	01			F112	U0123	01	01	
F10	U2007	20	20	27			F112	U0528	05	05	
F20	U0313	03	03				ALS240	U0805	08	22	
F20	U0315	03	03				ALS240	U2114	21	21	
F20	U0612	06	08				ALS244	U0708	07	09	
F139	U0404	04	04				ALS244	U0911	09	09	
F139	U0410	04	04				ALS244	U0922	09	09	
DLY20	U0308	05	03	05			ALS244	U1908	19	27	
DLY10	U0309	03	05	03			F240	U0326	03	03	
							F240	U0328	03	08	
							F244	U0112	01	27	
							F244	U0806	08	16	

VCC

THIS TABLE PROVIDES MORE INFORMATION ON THE JUMPERS

	J0301	J0302	J0501	J0101	J1801	J1802
CARRERA	1-2	1-2	1-2	1-2		
SIRIUS	3-4	3-4	3-4	1-2		
WEITEK VDD=4V					1-2	1-2
WEITEK VDD=5V					3-4	3-4
J0201						
DEFAULT SETTING	2-15	4-13	6-11	7-10	8-9	(5US RETRY 80US TIMEOUT)
J0701 J0702 J0703						
VERSION 0	1-2	1-2	1-2	CURRENT VERSION		
VERSION 1	1-2	1-2	3-4			
VERSION 2	1-2	3-4	1-2			
VERSION 3	1-2	3-4	3-4			
VERSION 4	3-4	1-2	1-2			
VERSION 5	3-4	1-2	3-4			
VERSION 6	3-4	3-4	1-2			
VERSION 0	3-4	3-4	3-4			

VCC

