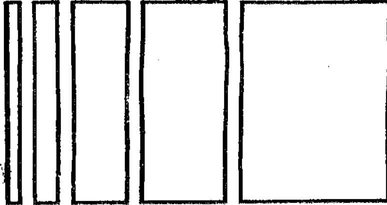


MC68000 CPU WITH SEGMENTED MEMORY MANAGEMENT Model CPU-68000M

FOR IEEE-696/S-100 COMPUTER SYSTEMS

COPYRIGHT©1982
DUAL SYSTEMS CORPORATION
FORM 880005
ALL RIGHTS RESERVED

USER'S MANUAL

DUAL SYSTEMS CORPORATION DUAL system reliability / system integrity

--TABLE OF CONTENTS--

Specifications...4

HARDWARE THEORY OF OPERATION:

I ...Address Space and Data Transfer

1.1 Data Bus..5

1.2 Address Bus..5
Address Bus Block Diagram............fig. 1

1.3 All, Top , Bot, None and Prot Jumpers.........6
Diagram of Mapped/Unmapped Regions...fig. 2
Jumper Locations.....................fig. 3

1.4 I/O-MMU Register Page..........................8
HI LO Jumper Location................fig. 4
Memory Map...........................fig. 4a

II ...Initialization and Bus Signals

2.1 RESET* , SLV_CLR*, POC*.......................11

2.2 Jump Vector...................................11
Switch Location......................fig. 4

2.3 Bus Errors and ERROR* Line....................12

2.4 Temporary Master Access
and TAS Instruction..................12

2.5 RUN, HALT, HOLD, FAST Indicators.............14

III ...Interrupts

3.1 68000 Priority Levels vs. S-100 and
Priority Levels Equivalence..........15

3.2 Device Supplied
Table of 68000 Vector Numbers...Table 1
Vector Numbers vs. AutoVectors.......15

3.3 Protocol for Device Supplied Interrupts......17

3.4 Daisy Chaining Interrupting Devices..........18
Example Circuit to Supply
Vector Numbers to CPU............Fig. 5
Location of XVC Jumper...........Fig. 6

SOFTWARE THEORY OF OPERATION:

IV ...Programming The MMU

4.1 Table of MMU Registers.......................19

APPENDICES:

A) Motorola Documentation of MC 68451

B) Binary Buddy System

Processor: Motorola 68000 L-8 or 68010 L-8 (L-10 for 10 Mhz)

Memory Management:
Motorola 68451-L8 (L-10 for 10 Mhz)

CPU Clock: 8 MHz or 10 MHz

Address Bus: 24 bit physical and logical address bus.
Conforms to IEEE 696/S-100 extended addressing.

Address Space Allocation:
Segmented memory management.
32 dynamically sized segments, 256b to 16Mb.
System level programs may run unmapped.
Relocatable I/O-MMU Page.

Data Bus: 16 bit bidirectional data transfers, 8 bit data transfers. Programs must reside in 16 bit memory.

Interrupts: IEEE 696 Interrupt lines NMI and VIO through VI5 supported.
Motorola device supplied interrupts fully supported by means of S-100 INT* line and daisy chain.
ERROR* line fully supported.

Control: Configured as permanent bus master. Provides TMA protocol as per IEEE 696.

Bus Cycle Time: IEEE 696 S-100 (8 MHZ CPU) Bus Cycle:
Unmapped: 750 nS
Mapped: 1000 nS
IEEE 696 S-100 (10 MHZ CPU) Bus Cycle:
Unmapped: 600 nS
Mapped: 800 nS

P.C. Board: High quality epoxy, solder masked both sides, screened component legend, plated through holes, gold plated edge connector fingers. Sockets provided for all I.C..s.

Power: Consumes 1650 mA nominal from 8 volt line.
7.5V Min, 10.5V Max.

User-Selectable Options: A0 low on even or odd byte access.
Start address switch selectable of 64K boundries.

1.1--Data Bus:

All modes of data transfer of the 68000 are supported by the CPU/68000/M. The only restrictions on the type of memories which may be used in the system are the following:

1) Programs must be executed out of memory with a sixteen bit data path.

2) Memories with only an eight bit data path may be used with the CPU/68000/M, however, only instructions which use a byte as their data type may be used to effect transfers to the memory.

3) Memories which do not support extended addressing may be used with the CPU/68000/M, but will limit the total useable memory to 64Kb.

1.2--Address Bus:

The MC68000 supports a 24 bit address bus, providing to the user a 16 Mb address space. The MC68451 memory management unit (MMU) is physically located between the 16 most significant address lines of the 68000 and the corresponding 16 lines of the S-100 address bus (see block diagram, next page).

The address lines on the 68000 are referred to as Logical Addresses. These are the untranslated addresses which the programmer sees when programming the 68000. Physical Addresses are the addresses on the S-100 bus. These are the addresses which memory, I/O boards and the I/O-MMU page address decoding responds to.

The MMU can manage up to 32 segments in the address space. Since the MMU translates only the most significant 16 address lines, the smallest allowable Segment size is 256 bytes. The size of the Segments must be a binary order of magnitude, 256 bytes, 512, 1024,..., 16Mb. The segments also must be defined such that their physical and logical base addresses are a multiple of their size.

The MMU can write protect segments, protect user or system processes from colliding, and translate logical addresses to physical addresses with an offset. For a complete description of the MMU capabilities, see the Motorola documentation in section 5.

The option is provided on the CPU/68000/M to allow bypassing the MMU on accesses to specified regions of the logical address space. See section 1.3 for a complete description of this feature.

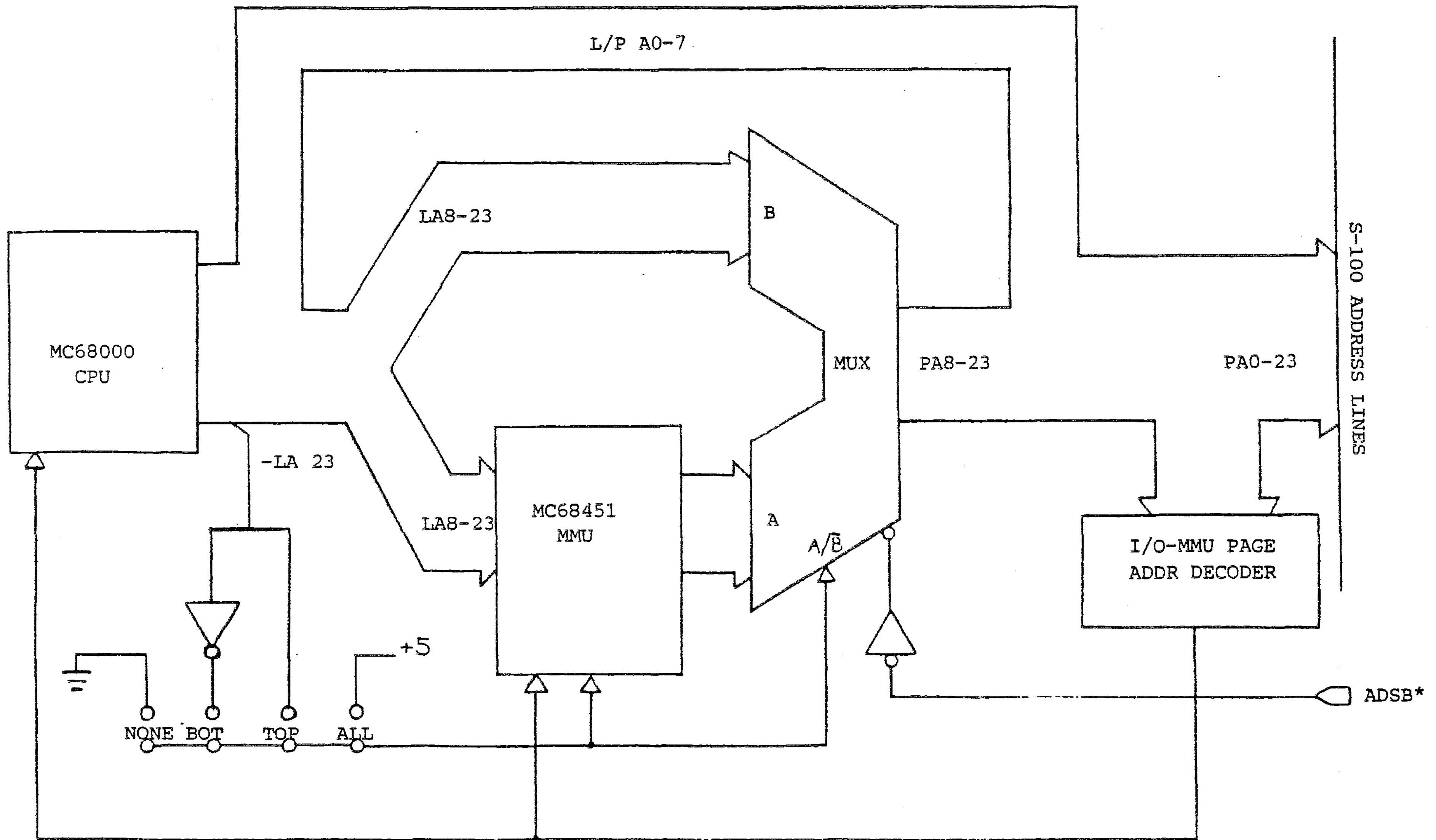


Fig. 1 CPU/68000/M Address Bus Block Diagram

1.3-- ALL, TOP, BOT, NONE and PROT Jumper:

As was mentioned in the last section, it is possible to bypass the MMU. The array of jumpers ALL, TOP, BOT and NONE control which addresses pass through the MMU, and which proceed directly to the physical (S-100) address bus. Make certain that only one of these jumpers is connected at one time.

If the ALL jumper is set, all logical addresses pass through the MMU. If the NONE jumper is selected, all logical addresses pass directly to the physical address bus.

The CPU/68000/M can also partition the logical address space into two equal sections, one mapped and the other unmapped. If the TOP jumper is selected, all logical addresses from 800000 to FFFFFF will pass through the MMU. When the addresses from the logical bus pass through the MMU, they participate in what Motorola refers to as the "matching and translation process" which makes up memory management. This is described in great detail in section 5. Any access to the bottom of the logical address space, 0 to 7FFFFF, will be passed directly onto the physical address bus.

The BOT jumper works the same way, only the bottom half of memory from 0 to 7FFFFF is mapped, while the top of the address space from 800000 to FFFFFF is unmapped.

The reasons for using the TOP or BOT settings are twofold:

> First, it is possible to increase the amount of available segments. If all or a portion of the kernel programs can run in protected unmapped space, more segments are free for user processes.
>
> Second, a bus cycle in an unmapped portion of the address space will run faster than a bus cycle in a mapped portion of the address space. Configuring the board so system tasks run in the faster unmapped space can result in a performance gain.

The significance of each of the settings is summarized below:

ALL: The entire physical address space is mapped. All logical addresses pass through the 68451.

NONE: The entire physical address space is unmapped. All logical addresses on the 68000 address bus pass directly to the physical (S-100) address bus. The memory management is inactive, although the MMU registers may still be accessed.

TOP: The TOP portion of the logical address space from 800000 to FFFFFF passes through the MMU. The logical address space from 0 to 7FFFFF passes directly the physical address bus. Protection of the unmapped space is available via the PROT jumper.

BOT: The BOTtom portion of the logical address space from 0 to 7FFFFF passes through the MMU. The logical address space from 800000 to FFFFFF pass directly onto the physical address bus. Protection of the unmapped space is available via the PROT jumper.

PROT Protection for unmapped space is implemented by means of the jumper labeled PROT. If this jumper is installed, the CPU/68000/M will not be allowed to access the unmapped address space unless the S bit (supervisor mode) is set in the status register. An attempted access to unmapped space by the CPU while in User mode will result in a bus error exception.

The PROT feature was designed to be used in conjunction with the TOP or BOT jumper settings to protect system software running in unmapped address space.

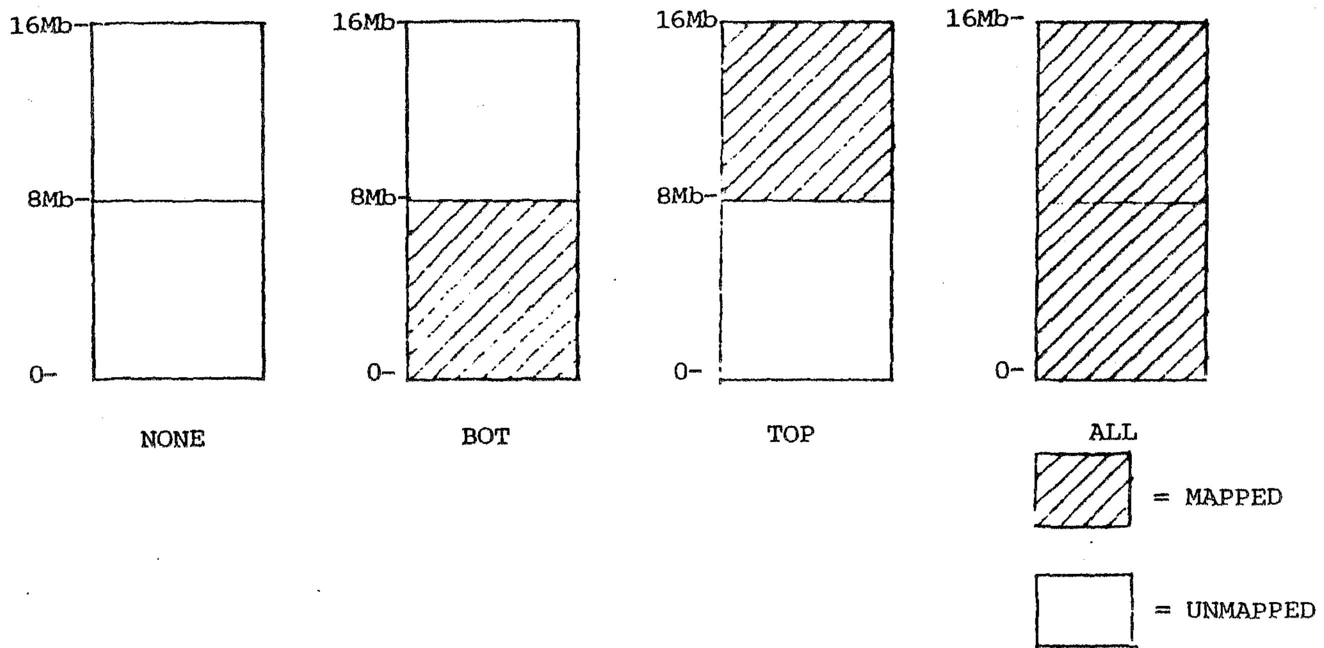
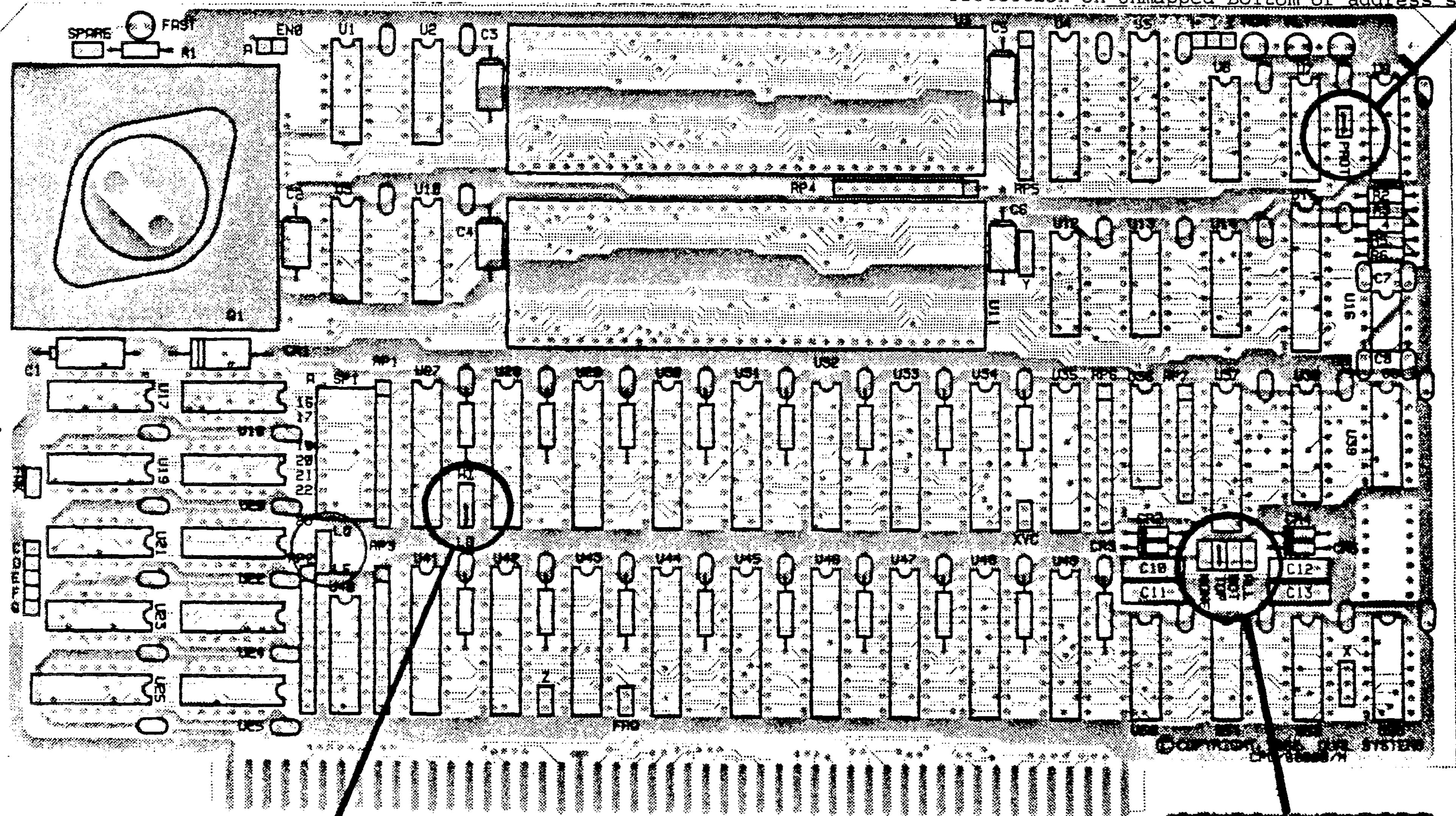


Fig. 2 Mapped and Unmapped Regions
of Logical Address Space

"PROT" Jumper installed.
Protection on Unmapped BOTtom of address space



I/O Page Jumper
set to "LO".

ALL, TOP, BOT, NONE Jumper
set to "TOP". Install only
one at a time.

Fig 3. Location of I/O-MMU Page Jumper
and ALL, TOP, BOT, NONE Jumper.
(Multiuser UNIX configuration)

60015

1.4-- I/O-MMU Register Page

The 68000 instruction set does not have an explicit Input/Output instruction. Motorola architects intended for all 68000 I/O to be memory mapped. Memory mapped I/O takes advantage of the many powerful addressing modes for fast, efficient I/O routines.

To support S-100 I/O mapped peripherals and to allow communication between the 68000 and the 68451, the processor board dedicates a 64 kilobyte page of the address space for I/O and the MMU registers. This page can based in the physical memory space at one of two following addresses:

```
I/O-MMU page:
   1)  7F0000 to 7FFFFF  (LO)
   2)  FF0000 to FFFFFF  (HI)
```

The jumper which selects the location of the physical base address for the I/O-MMU page has two settings: HI and LO. See fig. 3 for the location of the I/O-MMU page jumper. The .HI. and .LO. refer to the position of the I/O-MMU page at either a HI (FF0000) or LO (7F0000) base address. The CPU/68000/M comes configured with the I/O-MMU page jumper set to the .LO. position. Any access by the CPU/68000/M to the I/O page will cause the appropriate I/O status signals to be asserted on the S-100 bus.

The MMU registers reside in top 64 bytes of the I/O-MMU page, from 7FFFC0 to 7FFFFF, or FFFFC0 to FFFFFF in physical memory, depending on where the I/O-MMU page is located. On any access to the MMU registers, all S-100 strobes are rescinded until the data transfer between the CPU and the MMU is complete. Transfers between the 68000 and the MMU occur at 8 MHz. For a list of MMU register locations and a description of MMU register functions, see section 5.

Some Examples:

If the I/O page jumper is set to the HI position, the I/O page base address is FF0000. Hex address FF0002 corresponds to I/O port address 2. So the 68000 instruction:

```
MOVE.B  0FF0002H, D0
```

is functionally equivalent to the 8080 instruction

```
IN 02H
```

If the I/O page jumper were set to the LO position, the base address of the I/O page would be 7F0000. The 68000 instruction to accomplish a read from port address 2 would then be:

```
MOVE.B  07F0002H, D0
```

Note that almost 64 K bytes are dedicated to I/O devices. This allows almost 64 thousand input and output ports. To support this many ports requires that I/O devices decode the least significant 16 address lines. The IEEE specification allows the extended I/O addressing, but does not require it.

The majority of current I/O boards decode only the least significant 8 address bits. This results in only 256 input and output ports. Since the I/O board does not decode the full 16 bit address the 256 port addresses are replicated through the 64 K byte I/O space at 256 byte intervals.

Eight bit (non-extended) I/O addressing example:

When the I/O page is in the LO setting, the instruction:

```
MOVE.B 7F0002 , D0
```

will read from port 2 from a typical non-extended addressing I/O board.

The following will also read from port 2:

```
MOVE.B 7F0102 , D0
```

And so will this:

```
MOVE.B 7F0202 , D0
```

And so on through the I/O space.

See section 5.1 for a sample MMU register access.

Some Suggestions:

When programming I/O, it is a good idea to use the bottom most 256 byte I/O block exclusively so as to avoid accessing the MMU registers accidently. Stay clear of the top 256

block, 7FFF00 to 7FFFFF (LO) or FFFF00 to FFFFFF (HI) to avoid unintentionally accessing the MMU registers.

Since the I/O page location is decoded in the physical address space, it may be assigned a segment and relocated or write protected just as with normal physical memory. Take care that the operating system always has access to the MMU registers, as it is possible to map them out of the reach of the operating system entirely.

If the decision has been made to map only half of the physical memory, we recommend that the the TOP setting be used. The lower 8Mb could be reserved for the unmapped System tasks, as the exception vectors at hex addresses 0 to 400 and the I/O/MMU page at 7FFFBF could all reside in the same half of memory. This would obviate the need to define a separate System segment for the I/O-MMU page and exception vectors, as would be neccessary if everything were located in mapped space.

2.1 RESET*, SLAVE CLR*, POC*:

When power is first applied, RESET*, SLAVE CLR*, and POC* (Power On Clear) are all asserted low for 100 ms. During this time, the CPU clears its registers and resets itself, and the MMU initializes its registers so that it comes up with segment 0 mapping the logical addresses to the physical address space with no offset. The size of the segment is 16Mb, the entire address space. Even if the MMU is not engaged it is still initialized. For more information on the initial state of the MMU and its registers, see SOFTWARE THEORY OF OPERATION section 5.

The 68000 .reset. instruction will cause the S-100 SLAVE CLR* line to be asserted for 62 clock cycles (15.5 us). This is useful for initializing bus slaves to a known state. The .reset. instruction will not reset the 68000 or the MMU.

2.2--Jump Vector Switch:

A normal 68000 system reset operation consists of a 4 byte fetch from vector number 0 (hex address 0) to initialize the stack pointer, and a 4 byte fetch from vector number 1 (hex address 4) to initialize the program counter.

On a hard reset (power up, or RESET* asserted) the CPU/68000/M will fetch the stack pointer and the program counter starting from the location indicated on the dip-switch shown in fig. 4. The jump location must be at a 64K boundary, since only the high order address byte is used to form the jump address.

For Example:

If your boot program resides at hex address 20008, and the Stack Pointer is 7FFF. Locations 20000 through 20007 would have to have the values shown below, and S1 would have to be set as shown on the following page.

```
          +-----STACK POINTER-----+----PROGRAM COUNTER----+
          +                       +                       +
          *************************************************
          *     *     *     *     *     *     *     *     *
          * 00  * 00  * 7F  * FF  * 00  * 02  * 00  * 08  *
          *     *     *     *     *     *     *     *     *
          *************************************************
 Hex        +     +     +     +     +     +     +     +
 Address: 20000 20001 20002 20003 20004 20005 20006 20007
```

On reset, the CPU will fetch the stack pointer and the program counter. It will then begin executing code at the

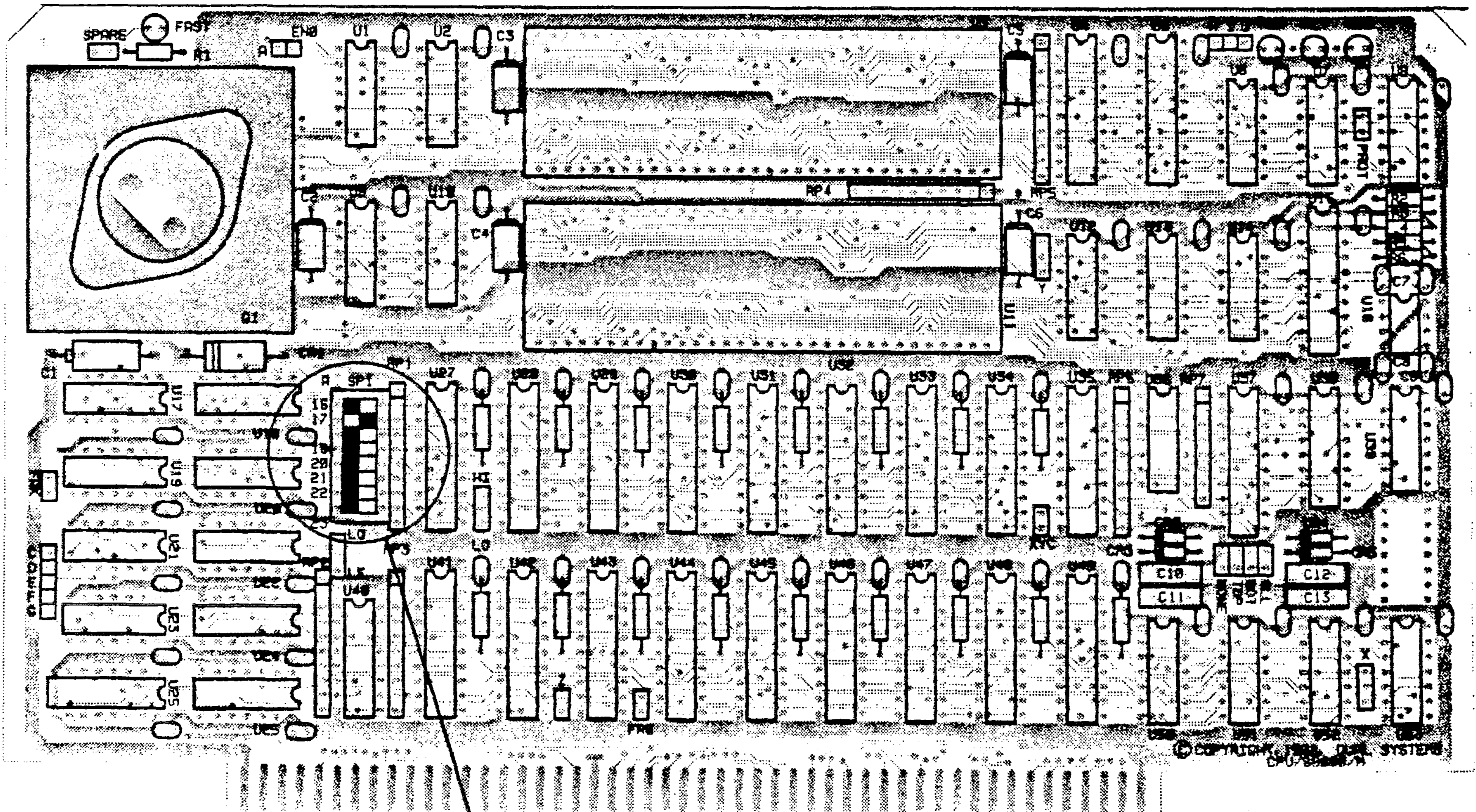


Fig. 4 Location and Setting of Address Switch

address of the PC, at 20008.

2.3--Bus Errors and the ERROR* Line:

Bus errors derive from three sources:

1) The MMU

2) Attempted access of PROTected unmapped System space while in User mode.

3) The S-100 ERROR* line

There are a number of conditions which will cause the MMU to initiate a bus error exception. Two of the most common are:

1) An attempted write to a write-protected segment,

2) An Undefined Segment Access

Refer to section 5 for more details on MMU generated bus errors.

The ERROR* line on the S-100 bus is a generally defined signal used to indicate catastrophic errors. We define its use in the following fashion:

1) The ERROR* line is sampled by the CPU/68000/M at two points in the S-100 bus cycle:

   a) the falling edge of phi during pSYNC.

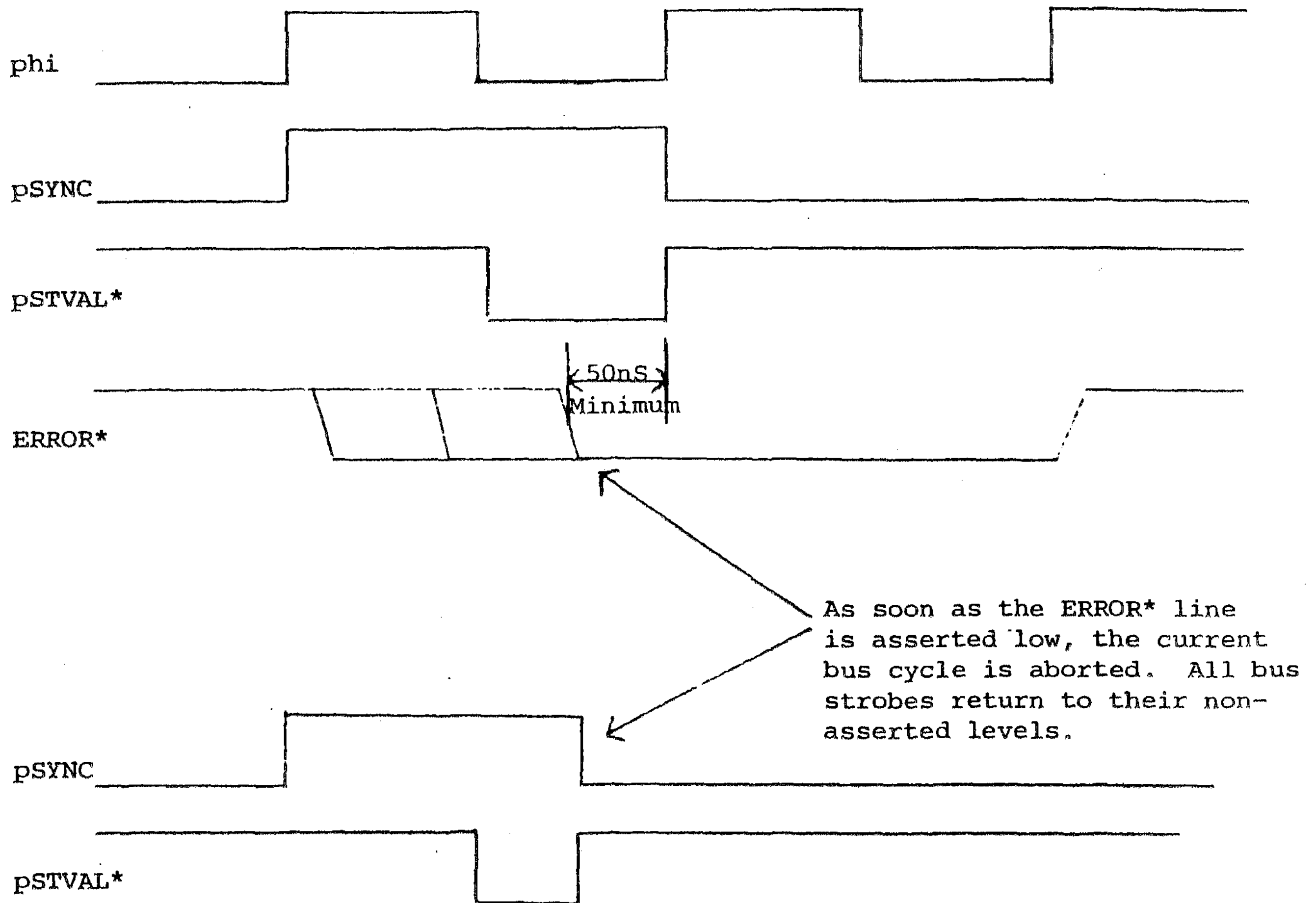
   b) the rising edge of phi immediately preceding the data strobe (pDBIN or pWR*).

2) After the ERROR* signal is latched , the current bus cycle is aborted, and 68000 bus error exception processing is held pending. Bus Error exception pro- cessing will proceed only after the ERROR* signal has been rescinded. If another bus error occurs during the exception processing, the 68000 will register a double bus fault condition and the processor will be halted.

Refer to the timing diagrams in the appendix for more detail.

2.4--Temporary Master Access and TAS Instruction

The CPU/68000/M functions as a permanent bus master as specified in the IEEE proposed S-100 standard. Temporary



Bus error exception processing will not proceed until the ERROR* has been brought high. If another bus error occurs during the bus error exception processing, the processor will register a double-bus-fault and halt.

ERROR* Line Timing

Figure 4a

bus masters (DMA devices) request the bus by asserting control input HOLD*. They receive control of the bus when the bus master (CPU/68000/M) asserts control output acknowledge pHLDA.

Upon receipt of HOLD the 68000 completes the current bus cycle and then asserts pHLDA. The 68000 suspends all processing until HOLD* is released. A temporary master may now disable the four disable lines ADSB*, SDSB*, CDSB* and DODSB*. The temporary master now has complete control of the bus for as long as it wishes. When the bus is no longer needed control is returned to the permanent master by releasing the bus disable signals, and finally, HOLD*.

The method of transferring the bus from the permanent bus master to a temporary master is explicitly specified in the IEEE bus standard section 2.8. Of significance is the method used to transfer ownership of the control output bus. To ensure glitch free transfer, both the permanent and temporary master drive the control output bus during the transfer period. Except for pHLDA, all lines are driven at their non-asserted levels. After a specified time (125 nano-seconds) the temporary master asserts CDSB*, disabling the permanent master.s control output bus drivers and acquiring control of the bus.

Up to 16 temporary masters may coexist in a system. A distributed arbitration scheme determines the highest priority device which then takes control of the bus upon assertion of pHLDA.

In general, the CPU/68000/M will relinquish control of the bus after the current bus cycle. However, if HOLD* is received just before the start of a bus cycle, the 68000 will go ahead with the bus cycle, relinquishing control after its completion.

As TMA operations occur on the physical side of the address bus, care must be taken to insure that memory addresses passed to TMA devices are physical addresses, not logical addresses. The corresponding physical address for a given logical address may be determined by a MMU Direct Translation Operation.

TAS Instruction:

The 68000 TAS (Test And Set) results in different CPU timing than other instructions. Motorola defines it as a read-modify-write cycle. The instruction results in sequential read and write cycles on the S-100 bus. The two cycles are indivisble, that is, the write cycle must follow the read cycle. Two distinct S-100 cycles are completed, but interrupts and bus requests will not be accepted until after

the instruction has completed.

2.5--RUN, HALT, HOLD and FAST Indicators:

The four L.E.D.s on the CPU/68000/M board indicate the following conditions:

RUN--(green): Valid addresses on the logical addresses bus, i.e. the AS* pin of the CPU is active. This light is active when the CPU is the current bus master, and instruction execution is proceeding without error.

HALT--(red): CPU is halted. This occurs either on reset (see section 2.1), or if a double bus fault condition is present.

HOLD--(yellow): A temporary master has been granted control of the bus. For example, this light will come on when a disk controller executes a TMA cycle.

3.1--68000 Priority Levels vs. S-100 Priority Levels:

The first item to be aware of about interrupts is the relation of the S-100 priority levels to the 68000 levels. According to the S-100 bus standard, NMI is the highest priority interrupt followed by VI0, VI1...VI7. Of these 9 levels of priority, the 68000 supports 7. The two lowest levels, VI6 and VI7, are not supported by the 68000. Motorola defines 7 as the highest level and 1 as the lowest level. This can be a source of confusion. Below is a chart which relates the two:

IEEE/S-100	68000
NMI	7 ...highest priority
VI0	6
VI1	5
VI2	4
VI3	3
VI4	2
VI5	1 ...lowest priority
VI6	--not supported--
VI7	--not supported--

Asserting one of these lines low generates a 68000 "auto-vector", and the 68000 will fetch a program counter (4 bytes) as described in section 2.2.

3.2--Device-Suplied Vector Numbers vs. AutoVectors:

The MC68000 chip provides a sophisticated interrupt structure which is completely supported by the CPU/68000/M. By supplying vector numbers to the CPU, 192 interrupting devices may be supported per priority level.

The two groups of interrupts to which the the CPU can respond are "device-supplied" and "auto-vector". Interrupt requests are placed on the S-100 lines NMI and VI0,...,VI5. Motorola 68000 device-supplied interrupt vectors are supported through the use of the S-100 INT* line and the S-100 interrupt lines NMI, VI0,...,VI5.

See the next page for a chart illustrating the device supplied interrupt protocol. The CPU/68000/M may be enabled to fetch vector numbers off the data bus by installing the jumper labeled .XVC..

The CPU will only fetch a vector number from the data bus if INT* is asserted. If one of the lines NMI, VI0, ..., VI5 is asserted without INT*, a 68000 autovector operation will result.

S-100 Interrupt Priority	PC Fetch Location (hex)
NMI	7C
VI0	78
VI1	74
VI2	70
VI3	6C
VI4	68
VI5	64

Notes:

INT* never asserted.

CPU will fetch 4 bytes of PC--address of interrupt service routine.

MMU Interrupt and VI0 at same priority level, IVR is the MMU Interrupt Vector Register.

Table 5-2. Exception Vector Assignment

Vector Number(s)	Dec	Address Hex	Space	Assignment
0	0	000	SP	Reset Initial SSP[2]
	4	004	SP	Reset Initial PC[2]
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12[1]	48	030	SD	(Unassigned, Reserved)
13[1]	52	034	SD	(Unassigned, Reserved)
14[1]	56	038	SD	(Unassigned, Reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16-23[1]	64	040	SD	(Unassigned, Reserved)
	95	05F		—
24	96	060	SD	Spurious Interrupt[3]
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors[4]
	191	0BF		
48-63[1]	192	0C0	SD	(Unassigned, Reserved)
	255	0FF		—
64-255	256	100	SD	User Interrupt Vectors
	1023	3FF		—

NOTES:

1. Vector numbers 12, 13, 14, 16 through 23, and 48 through 63 are reserved for future enhancements by Motorola. No user peripheral devices should be assigned these numbers.
2. Reset vector (0) requires four words, unlike the other vectors which only require two words, and is located in the supervisor program space.
3. The spurious interrupt vector is taken when there is a bus error indication during interrupt processing. Refer to Paragraph 5.5.2
4. TRAP #n uses vector number 32 + n.

Table 1. 68000 Vector Number Table

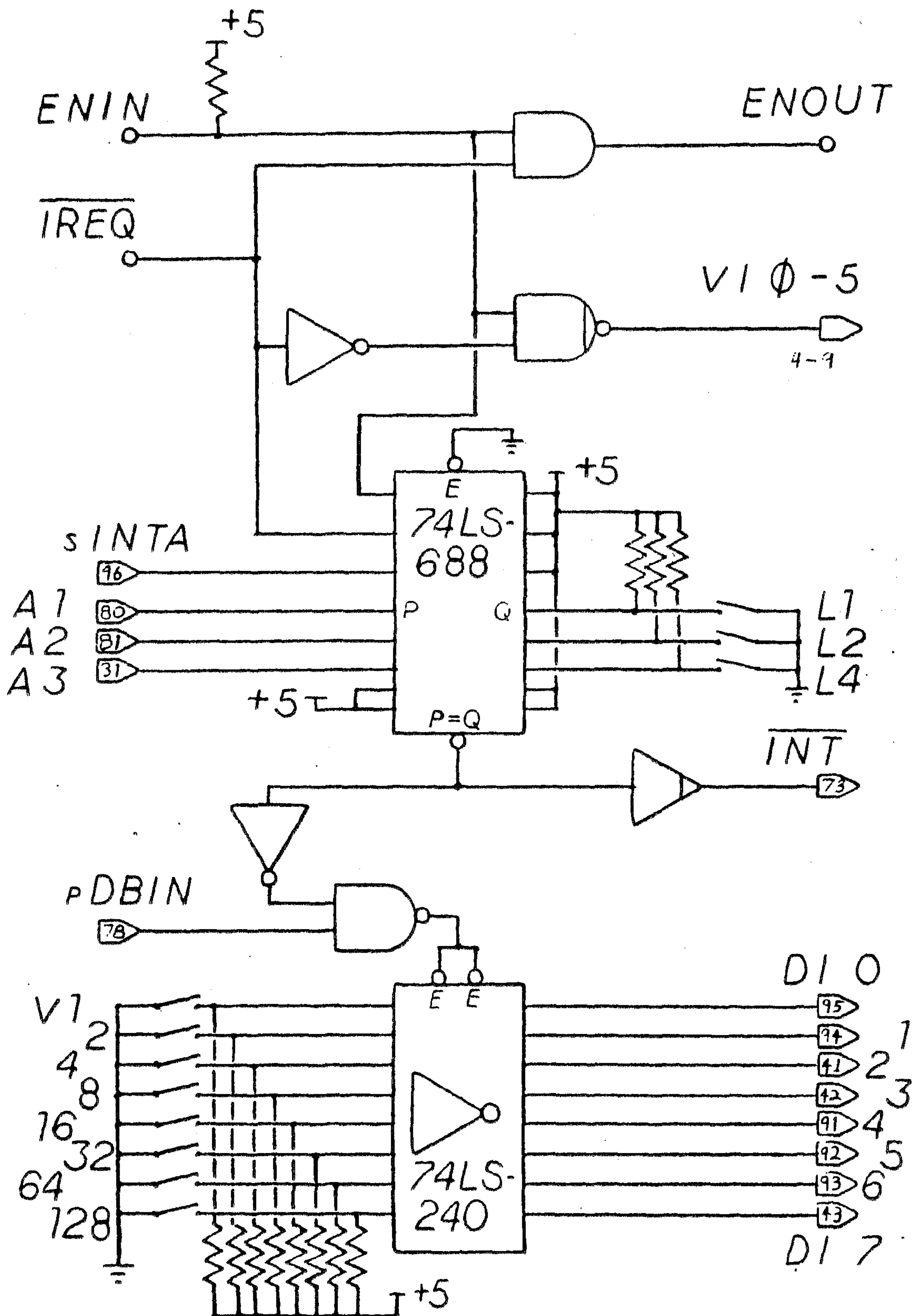


Fig. 5 Example Circuit to
Supply Vector Numbers to CPU.

3.3--Protocol for Device Supplied Interrupts:

```
Interrupting device                     68000 CPU
-------------------                     ---------

Asserts one of NMI, VI0,...,VI5
concurrent with disabling all devices
lower in the chain.
              +
              +
              +-------------------+
                                  +
                                  +
                     Compares interrupt level in status register.
                     Waits for current instruction to complete.
                     Asserts sINTA.
                     68000 interrupt level is placed on
                     address lines A1,A2,A3.
                                  +
                                  +
              +-------------------+
              +
              +
Compares A1,A2,A3 with the interrupt
level which it asserted.  If equal,
deposit vector number on next read,
and assert INT*
              +
              +
              +-------------------+
                                  +
                                  +
                     If INT* asserted:
                     Starts read cycle.
                     Latches vector number on pDBIN.
                     Fetch PC at vector number * 4.
                     Begins exception service routine
                     at new program counter.
                     If INT* not asserted, then
                     autovector
                                  +
                                  +
              +-------------------+
              +
              +
Interrupting device rescinds IRQ*
active low). Enables devices
lower in chain to generate interrupts.
```

3.4--Daisy-Chaining Interrupting Devices:

In order to use two or more devices to supply vector numbers within a priority level (NMI, VI0,..., VI5), it is necessary to daisy-chain devices which supply vector numbers within each priority level.

"Daisy-chaining" means the following:

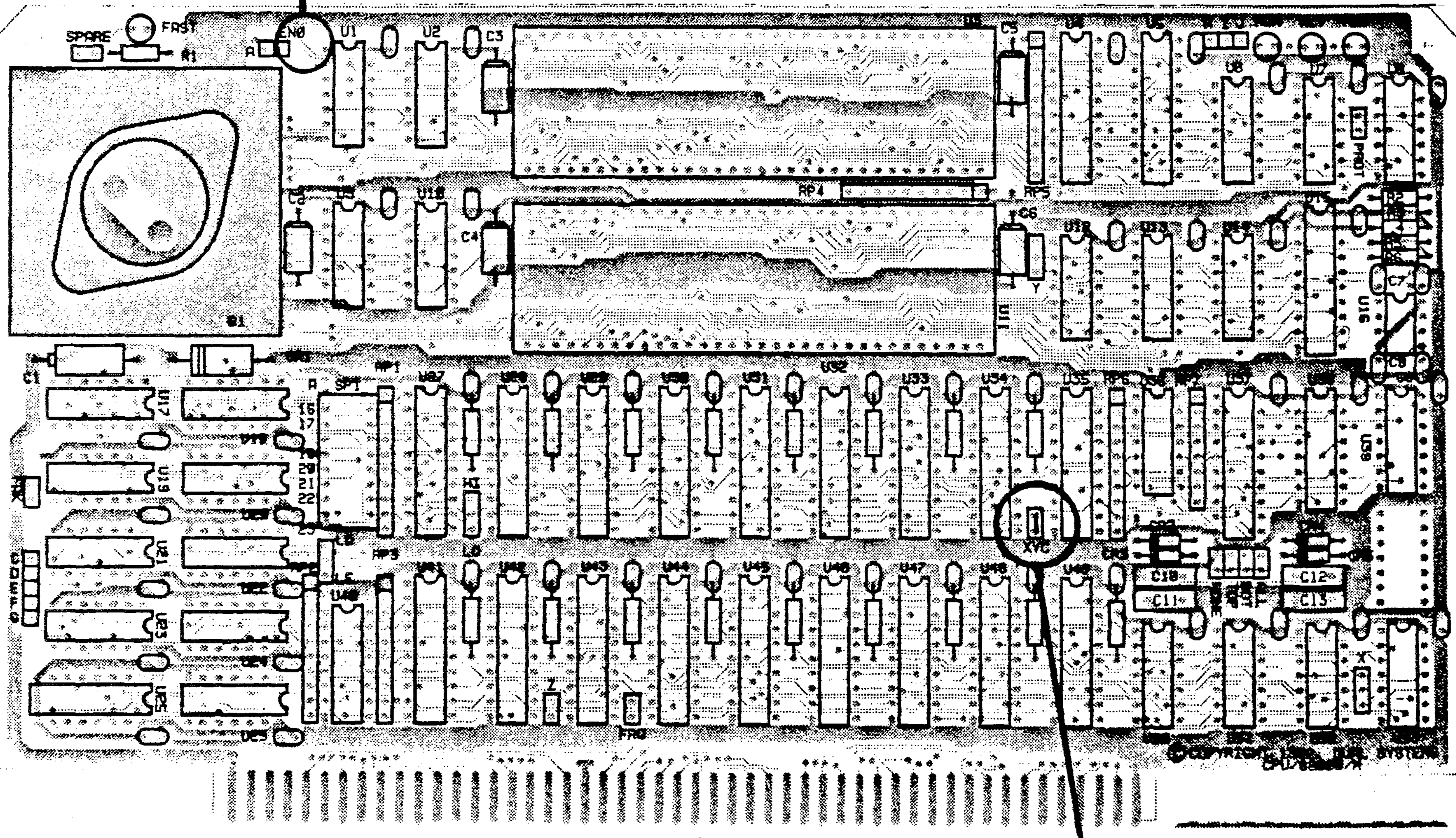
If two or more devices in the chain make simultaneous interrupt requests, the requesting device highest in the chain disables the lower ones, and supplies the vector number.

A maximum of two interrupting devices may be used within the same 68000 priority level without daisy- chaining. The constraint is that one must supply a vector number to the CPU/68000/M while the other must use an autovector.

Devices are chained only within priority levels because the CPU displays the priority level it is servicing on address lines A1 through A3. Devices must only supply vector numbers to the CPU if lines A1 through A3 on the address lines are equal to the 68000 priority level of the chained devices (see section 3.1 for an explanation of .priority level.). See fig. 5 for a sample circuit using daisy-chained interrupts.

Remember that it is only necessary to daisy-chain interrupting devices if it is desired that two or more devices supply vector numbers within a priority level to the CPU/68000/M. If autovectors only are requested on VI0 (a normal S-100 interrupt), the ENOUT jumper is not required.

ENOut Daisy Chain Point. (The post to the left is a test point.)



XVC Jumper set. Board will respond to both normal S-100 interrupts and devices supplying vector numbers.

Fig. 6 Location of XVC Jumper and ENOut Daisy Chain Post.

60018

4.1--MMU Registers:

MMU Register Page
Physical Base address = 7FFFC0 (LO) or FFFFC0 (HI)
(depending on setting of I/O-MMU page jumper)

Offset (bytes)	Acronym	Function
		Address Space Table:
0	AST 0	reserved
2	AST 1	user data
4	AST 2	user program
6	AST 3	reserved
8	AST 4	reserved
A	AST 5	supervisor data
C	AST 6	supervisor program
E	AST 7	interrupt acknowledge
		Accumulator:
20	AC0	Logical Base Addr/ translation ADDR (MSB)
21	AC1	Logical Base Addr/ translation ADDR (LSB)
22	AC2	Logical Address Mask (MSB)
23	AC3	Logical Address Mask (LSB)
24	AC4	Physical Base Addr/ translated ADDR (MSB)
25	AC5	Physical Base Addr/ translated ADDR (LSB)
26	AC6	Address Space Number
27	AC7	Status Number
28	AC8	Address Space Mask
		Control and Status:
29	DP	Descriptor Pointer
2B	IVR	Not implemented
*2C	GSR	Global Status Register
2F	LSR	Local Status Register
31	SSR	Segment Status Register /transfer descriptor
39	IDP	Not implemented
3B	RDP	Result Descriptor Pointer
3D	DTO	Direct Translation Operation
3F	LDO	Load Descriptor Operation

*WARNING! due to a characteristic of the current mask of the 68451 MMU, writing to this read only register will HALT the processor board.

Calculating an MMU register address;
an example:

To write the contents of D0 to AST7 with I/O-MMU page in L0 (7F0000) physical memory:

First calculate register address;

```
AST7 address = (mmu base addr) + (AST7 offset)
7FFFCE = 7FFFC0 + E
```

Then write the byte in D0 (for example) to that address;

```
MOVE.B D0, (7FFFCE)
```

4.2--Motorola Documentation of 68451:

The following is an edited version of the Motorola documentation for the MC 68451 Memory Management Unit. Note that as there is only one MMU in the system, so it is not necessary to distinguish between the terms global and local in the documentation, i.e. local operations are always global. In addition, the facility of the MMU to generate device supplied interrupts to the CPU is not supported.

4.3--Binary Buddy System

The following paper outlines a memory management scheme suggested by Motorola for use with the 68451 Memory Management Unit.

MEMORY MANAGEMENT
FROM THE SOFTWARE ENGINEER'S VIEWPOINT

Russell E. Schwausch
Staff Engineer

Motorola, Inc.
MOS Integrated Circuits Division
3501 Ed Bluestein Blvd
Austin, TX 78721

SUMMARY

Implementation of memory management in a computer system takes a combination of hardware and software. Since the software issues involved are both complex and system dependent the decisions regarding them are not easy and require compromises based on system priorities. I am going to discuss some of the tradeoffs involving hardware, software, time and memory.

For an introduction to memory management I will use some background information on two current memory management techniques, segmented memory management and paged memory management. Then I am going to talk about descriptor management and its relationship to memory management and I will also discuss memory fragmentation, both internal and external. Since system overhead is a key issue I will present some ideas on how to reduce overhead with various memory management techniques.

I do not intend to do an exhaustive study of memory management philosophies but rather to point out some of the pitfalls you might encounter (hopefully before you encounter them) and to present an approach to memory management that is well-suited to the Motorola 68451 Memory Management Unit (MMU).

MEMORY MANAGEMENT TECHNIQUES

Segmented versus Paged Memory Management

When memory is divided into blocks of more than one size, these blocks are generally referred to as segments. Segment sizes vary depending on the environment. These sizes may be powers of two or they may change in size in steps that are powers of two or they may fall on any byte boundary.

Each task can be assigned one or more segments. In a segmented memory system, a task usually has fewer segments than it would have in a paged system. This means that less memory is required for the allocation tables that must be maintained in memory. The size of a segment assigned to a task is usually determined dynamically at the time of the request. Fewer segments are needed to represent the resident task and as a result less overhead is needed to manage the memory.

In a paged system memory pages are fixed in length and usually small in size with multiple pages assigned to each active task in the system. There is generally a table in memory to keep track of pages available for allocation and another table for pages currently allocated to active tasks. The page size is system dependent.

The most frequently used page management scheme is demand paging. This technique refers to an approach in which pages are allocated to a task as needed. As a result the number of pages of memory that are allocated to a task at

any given time is small. And, since only a few pages are allocated, the page allocation table which is generally resident in memory can be smaller. However, this may result in more frequent system intervention to satisfy requests for additional memory. As a result system overhead increases.

Both of these approaches represent forms of dynamic storage allocation. Dynamic storage allocation simply implies that memory is allocated at run time and is dependent on the current state of the system. With either approach a task may reside in different areas of memory each time it is executed.

MEMORY FRAGMENTATION

Internal Memory Fragmentation

Internal memory fragmentation [4] occurs because memory segments, although variable in size, are only available in discrete sizes. The unused portion of a requested segment of memory represents an instance of internal fragmentation.

Lets look at a memory request that results in internal memory fragmentation. We'll assume that the smallest block of memory that can be allocated is 2K. If a task requests a 7K block of memory we must fill that request with the next larger size segment, an 8K segment. As a result we have an unused 1K fragment that is not available for use by the system. This is referred to as internal fragmentation since the fragment is internal to a memory segment. Figure 1 illustrates this situation.

Internal memory fragmentation can also be thought of as allocated memory that is not needed by the task it is allocated to.

External Memory Fragmentation

External memory fragmentation [4] leads to the situation in which a task cannot get a segment of memory of the requested size even though there is enough memory available in the system. This happens because the memory management mechanism cannot combine the available memory into a contiguous block at least as large as the segment requested.

For an example of external memory

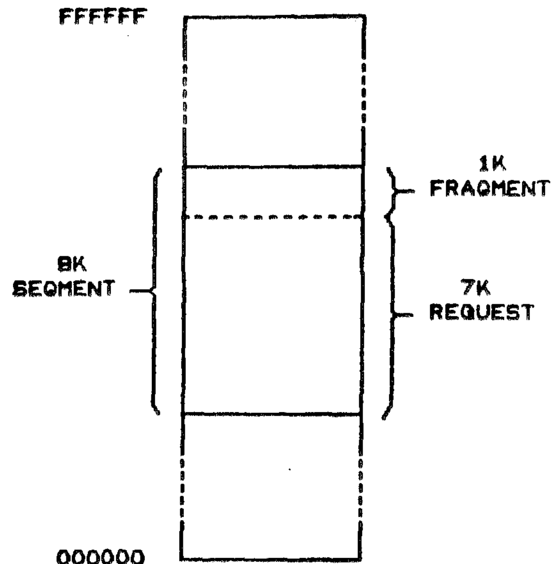


Figure 1. Example of Internal Memory Fragmentation.

fragmentation let's look at memory after a number of requests have been made and some segments deallocated. We'll assume we have a free 16K block at address 4000 and another free 16K block at address 18000. Now a request comes in for a 32K block of memory. A standard memory management algorithm such as the binary buddy algorithm looks for a contiguous 32K block composed of two 16K buddies. Since these two 16K blocks are not contiguous the binary buddy method would ignore them. If a contiguous 32K block could not be found the request would be queued up until a contiguous 32K block was available. According to Knuth [1] this is not as inefficient as it first appears. In fact it doesn't become a problem until 95% of memory has been reserved. Figure 2 illustrates an example of external memory fragmentation.

External memory fragmentation can be thought of as unallocated memory that is currently unusable. This distinguishes it from internal fragmentation which also represents unusable memory but memory that has been allocated.

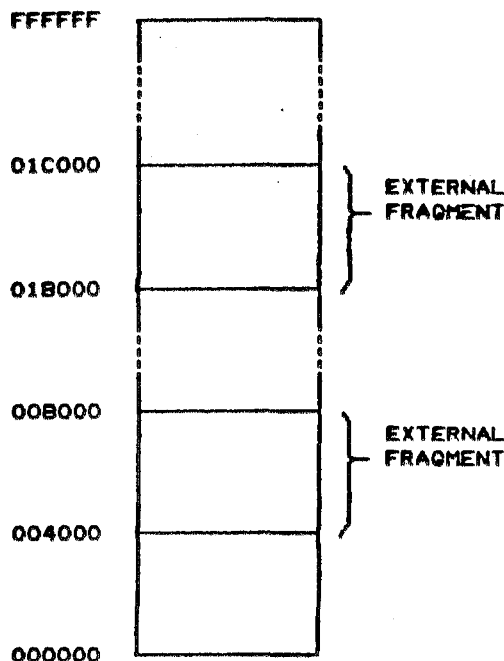


Figure 2. Example of External Memory Fragmentation.

THE 68451 APPROACH

Descriptor Management

Any memory management mechanism involving hardware is capable of representing a fixed number of memory blocks at any one time. In the Motorola 68451 these memory blocks are represented by descriptors. Each descriptor contains information that specifies the first word address of a memory block, its size or length, the physical address it will be mapped to, the task it is assigned to (its address space number), an address space mask that determines which tasks can share this memory block and a segment status byte that contains access control and status information.

The goal of descriptor management is to minimize the number of times that MMU-resident descriptors must be replaced. This happens if descriptors required by another task, being activated, are not resident. When we look at what it takes to do a task switch we find out why we don't want to replace MMU-resident descriptors. If the descriptors are resident we can simply change two entries in the 68451's address space table to select the new user address space number. This is illustrated in figure 6 by the 'TASKSWTCH' routine. However, if the descriptors are not resident, the operating system must first determine which resident descriptors to replace. Then it must load each new descriptor into the MMU as shown by the 'LDDESC' routine in figure 6. And finally it must change the two address space table entries once the descriptors are resident. In some cases the descriptors must be saved before being replaced. This is necessary if they contain status information that is significant to the operating system (see 'SAVEDESC' and 'SVDSCDIS' in figure 6). So we can see that descriptor management by itself is an important task. It must be handled properly to minimize system overhead.

We must deal with two separate tasks in managing MMU descriptors. The first task involves deciding on the number of descriptors to use to describe a given segment of memory. On one hand we would like enough descriptors to minimize the occurrence of internal fragmentation. On the other hand we would like to use descriptors sparingly to reduce the need to replace resident descriptors when a new task is activated. If too many descriptors are used we may need more MMUs or we may increase system overhead required to transfer descriptors in and out of the MMU from memory.

At one extreme we can always allocate memory using only one MMU descriptor. The average memory block size requested will fall half way between a given size 2**k and the next smaller size 2**k-1. Internal fragmentation with this approach will be approximately 25% of the segment size required to fill the request. This is shown in figure 3. The amount of internal fragmentation for a particular block size, represented by one descriptor, will vary between 0% and 50%. The average amount of internal fragmentation will then be 25%. If the fragmentation reaches 50% we would use the next smaller segment size and reduce fragmentation to zero. If we use 2 descriptors per segment we can cut the average waste in half from 25% to 12.5%. As we continue this approach the amount

of internal fragmentation approaches zero. But the number of MMU descriptors left to describe other memory segments also decreases.

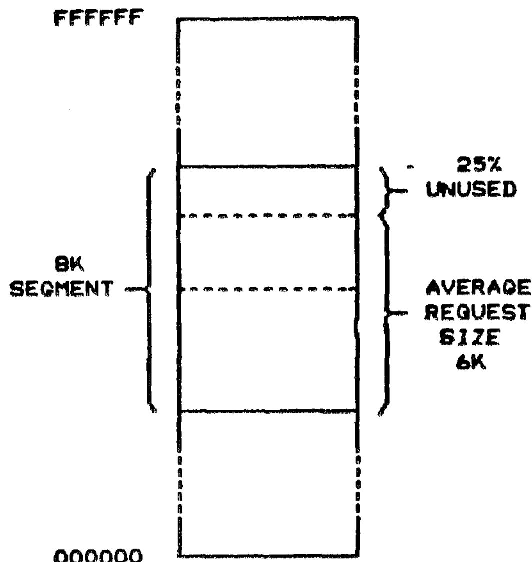


Figure 3. Relationship Between Descriptors and Internal Memory Fragmentation.

Keeping all descriptors in the MMU all the time is one approach. This method works fine if the total number of descriptors is small. There is a tradeoff between the number of descriptors needed by the system and the number of MMUs necessary to hold those descriptors.

The other approach is to keep descriptors in memory when there are not enough descriptors available in the MMU(s). This necessitates swapping descriptors in and out of the MMU(s) on a dynamic basis. Since descriptor swapping overhead is not negligible, we need to minimize the number of swaps required as a function of time. This creates the need for Descriptor Management.

Now we have a better understanding of some of the tradeoffs that must be considered. They involve the three issues of 1) how many MMUs to put in the system, 2) how much memory to put in the system, and 3) how much system overhead to allow for swapping descriptors between the MMU(s) and memory. There is no easy answer to these three issues. Each system has different characteristics and must be tuned for optimum performance. One way to get a grip on these numbers would be to use simulations based on the makeup of a given system. Or we could build a prototype and experiment with these system variables while collecting data on system performance. Keep in mind, though, that performance can change with time as the type of programs run on the system change. Something that is optimal now might not be a few months from now.

Memory Management

Our intent throughout this discussion has been to look at ways to effectively manage memory. In order to manage memory we must break it up into parts that are easy to handle. The Motorola 68451 is designed to divide memory into parts whose sizes are powers of two. There have been several papers written on managing memory in this fashion *. The most common algorithm for use with segments of these sizes is referred to as the binary buddy system.

The Binary Buddy System

The binary buddy system refers to a segmentation scheme in which all segments are of sizes that are powers of 2. As a result any segment can be split into two 'buddies' whose sizes are also powers of two. For instance an 8K memory segment at address xxxx can be split into two 4K buddies at addresses xxxx and xxxx+4K. Binary buddies are always equal in size. The first word address of the buddy for a segment is found by performing the exclusive 'OR' of the segment first word address and its size. Figure 4 shows how this is done.

The desirability of this scheme lies in the ease with which buddies can be recombined into larger segments. We would like to recombine free memory segments into the largest possible contiguous blocks in preparation for a large memory request. However this scheme is not without its problems. Overhead is

* See references at end of paper.

Segment Address		18000		1C000
Segment Size	EOR	04000	EOR	04000
Address of Buddy		1C000		18000

Figure 4. Binary Buddy Address Calculation.

consumed in recombining buddies even though the system may have to split these same buddies to satisfy the next request. If a smaller block of memory is needed next this might happen. In addition it is possible to have two adjacent memory segments of the same size that cannot be recombined because they aren't buddies.

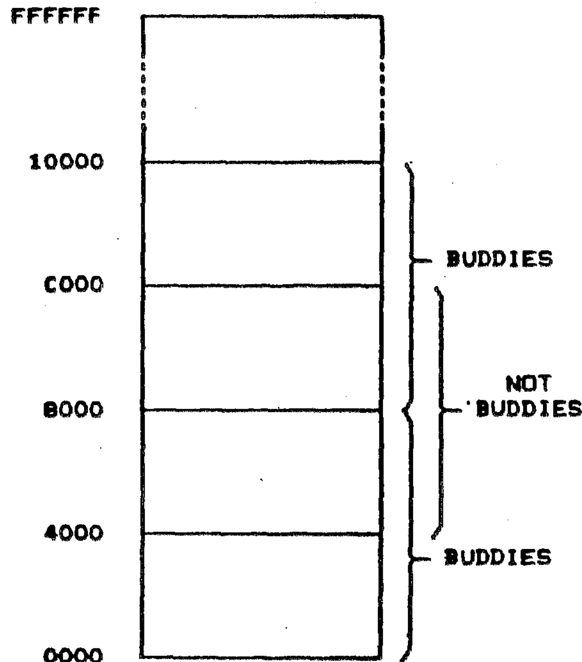


Figure 5. Example of External Memory Fragmentation.

In figure 5 we have two 16k segments that can not be combined because they aren't buddies. Even though they are contiguous, the binary buddy system could not use them to fill a request for 32K of memory. But with the 68451 we could fill the request by using two descriptors. The 68451 would even permit us to fill the request with two 16K segments that aren't contiguous. The 68451 can overcome the problems caused by external memory fragmentation in this way.

<u>Memory Allocation</u>

In order to allocate all of the memory for a task efficiently we should be able to vary the size of the memory blocks we allocate. The 68451 is capable of allocating blocks of memory in sizes from 256 bytes up to 16777216 bytes. These sizes are powers of two.

Mask	Power	Size
FFFF00	2**08	100
FFFE00	2**09	200
FFFC00	2**10	400
FFF800	2**11	800
FFF000	2**12	1000
FFE000	2**13	2000
FFC000	2**14	4000
FF8000	2**15	8000
FF0000	2**16	10000
FE0000	2**17	20000
FC0000	2**18	40000
F80000	2**19	80000
F00000	2**20	100000
E00000	2**21	200000
C00000	2**22	400000
800000	2**23	800000
000000	2**24	1000000

Table 1. 68451 Memory Segment Sizes.

Table 1 shows all possible block sizes for the 68451. These block sizes are selected by loading the Logical Address Mask with the appropriate value as indicated in the table. Basically the Logical Address Mask determines how many bits of the logical address to pass through and how many to get from the Physical Base Address Register in the descriptor. Ones in the mask indicate

```
 1                        * CODE EXAMPLES FOR INTERFACE TO THE MOTOROLA 68451 MEMORY MANAGEMENT UNIT
 2
 3                        * WRITTEN BY: RUSSELL E. SCHMAUSCH
 4                        * DATE: 23 DEC 1981
 5
 6         00000000              RORG $0
 7
 8                        * PROGRAM EQUATES
 9
10         00FF0000       MMU    EQU    $FF0000        MMU BASE ADDRESS
11         00000000       AST    EQU    $00            ADDRESS SPACE TABLE OFFSET
12         00000020       AC0    EQU    $20            ACCUMULATOR 0 OFFSET
13         00000026       AC6    EQU    $26            ACCUMULATOR 6 OFFSET
14         00000028       AC8    EQU    $28            ACCUMULATOR 8 OFFSET
15         00000031       TD     EQU    $31            OFFSET FOR TRANSFER DESCRIPTOR OPERATION (READ)
16         00000031       SS     EQU    $31            OFFSET FOR SEGMENT STATUS WRITE OPERATION (WRITE)
17         0000003D       DT     EQU    $3D            OFFSET FOR DIRECT TRANSLATION OPERATION (READ)
18         0000003F       LD     EQU    $3F            OFFSET FOR LOAD DESCRIPTOR OPERATION (READ)
19
20
21
22                        * THE FOLLOWING ROUTINE ASSUMES THAT DESCRIPTOR INFORMATION EXISTS IN
23                        * SYSTEM RAM AND THAT A DECISION HAS ALREADY BEEN MADE TO LOAD A SPECIFIED
24                        * DESCRIPTOR WITH SELECTED INFORMATION. THE DESCRIPTOR NUMBER HAS ALREADY
25                        * BEEN LOADED INTO THE DESCRIPTOR POINTER REGISTER. THIS ROUTINE IS DESIGNED
26                        * TO MINIMIZE THE NUMBER OF INSTRUCTION CYCLES REQUIRED TO ACCOMPLISH THE
27                        * LOADING OF THE DESCRIPTOR.
28
29                        * REGISTER A0 POINTS TO THE TABLE OF DESCRIPTORS FOR THE INTENDED TASK.
30                        * REGISTER D0 POINTS TO THE SELECTED DESCRIPTOR INFORMATION IN THE TABLE.
31
32                        * EACH TABLE ENTRY HAS THE FOLLOWING FORMAT:
33
34                        *      TABENT         LLLL    (16-BIT LOGICAL BASE ADDRESS)
35                        *      TABENT+2       MMMM    (16-BIT LOGICAL ADDRESS MASK)
36                        *      TABENT+4       PPPP    (16-BIT PHYSICAL BASE ADDRESS)
37                        *      TABENT+6       NN      (8-BIT ADDRESS SPACE NUMBER)
38                        *      TABENT+7       SS      (8-BIT SEGMENT STATUS)
39                        *    - TABENT+8       AA      (8-BIT ADDRESS SPACE MASK)
40
41
42         00000000       LDDESC EQU    *              LOAD DESCRIPTOR
43
44 000000 4CF000300000           MOVEM.L 0(A0,D0),D4-D5  MOVE LBA, LAM, PBA, ASN, & SS TO D4 & D5
45 000006 1C300008               MOVE.B  8(A0,D0),D6     MOVE ASM TO D6
46
47 00000A 48F90030
          00FF0020              MOVEM.L D4-D5,MMU+AC0   MOVE LBA, LAM, PBA, ASN, & SS TO MMU ACCUMULATOR 0-7
48 000012 13C600FF0028           MOVE.B  D6,MMU+AC8      MOVE ASM TO MMU ACCUMULATOR 8
49 000018 103900FF003F           MOVE.B  MMU+LD,D0       INITIATE DESCRIPTOR LOAD BY READING ADDRESS MMU+LD
50
51 00001E 4E75                   RTS
52
53
54
55                        * THE FOLLOWING ROUTINE SAVES THE CONTENTS OF A DESCRIPTOR THAT IS RESIDENT
56                        * IN THE MMU. IT DOES NOT AFFECT THE 'USED' OR 'MODIFIED' BITS. IT DOES NOT
57                        * DISABLE THE DESCRIPTOR. THE DESCRIPTOR POINTER REGISTER, WHICH SELECTS THE
58                        * DESCRIPTOR TO BE READ, HAS BEEN LOADED PREVIOUSLY.
59
60                        * REGISTER A0 POINTS TO THE TABLE OF DESCRIPTORS FOR THE INTENDED TASK.
61                        * REGISTER D0 POINTS TO THE SELECTED DESCRIPTOR INFORMATION IN THE TABLE.
62
63         00000020       SAVEDESC      EQU    *       SAVE DESCRIPTOR
64
65 000020 103900FF0031           MOVE.B  MMU+TD,D0       TRANSFER DESCRIPTOR TO MMU ACCUMULATOR  0-8
66
67 000026 4CF90030
          00FF0020              MOVEM.L MMU+AC0,D4-D5   MOVE CONTENTS OF ACCUMULATOR 0-7 TO DATA REGS
68 00002E 1C3900FF0028           MOVE.B  MMU+AC8,D6      MOVE CONTENTS OF ACCUMULATOR 8 TO DATA REG
69
70 000034 48F000300000           MOVEM.L D4-D5,0(A0,D0)  MOVE LBA, LAM, PBA, ASM & SS FROM DATA REGS. TO RAM
71 00003A 11860008               MOVE.B  D6,8(A0,D0)     MOVE ASM FROM DATA REGISTER TO MEMORY
72
73 00003E 4E75                   RTS
74
75
76
```

Figure 6. 68451 Utility Software Routines.

```
77                       * THE FOLLOWING ROUTINE SAVES THE CONTENTS OF A DESCRIPTOR THAT IS RESIDENT
78                       * IN THE MMU. IT DOES NOT AFFECT THE 'USED' OR 'MODIFIED' BITS. IT DOES
79                       * DISABLE THE DESCRIPTOR. THE DESCRIPTOR POINTER REGISTER, WHICH SELECTS THE
80                       * DESCRIPTOR TO BE READ, HAS BEEN LOADED PREVIOUSLY.
81
82                       * REGISTER A0 POINTS TO THE TABLE OF DESCRIPTORS FOR THE INTENDED TASK.
83                       * REGISTER D0 POINTS TO THE SELECTED DESCRIPTOR INFORMATION IN THE TABLE.
84
85        00000040       SVDSCDIS       EQU     *         SAVE DESCRIPTOR, THEN DISABLE IT
86
87 000040 103900FF0031          MOVE.B  MMU+TD,D0         TRANSFER DESCRIPTOR TO MMU ACCUMULATOR  0-8
88
89 000046 4CF90030
          00FF0020             MOVEM.L MMU+AC0,D4-D5     MOVE CONTENTS OF ACCUMULATOR 0-7 TO DATA REGS
90 00004E 1C3900FF0028          MOVE.B  MMU+AC8,D6        MOVE CONTENTS OF ACCUMULATOR 8 TO DATA REG
91
92 000054 48F000300000          MOVEM.L D4-D5,0(A0,D0)    MOVE LBA, LAM, PBA, ASM & SS FROM DATA REGS. TO RAM
93 00005A 11860008              MOVE.B  D6,8(A0,D0)       MOVE ASM FROM DATA REGISTER TO MEMORY
94
95 00005E 13FC0000
          00FF0031             MOVE.B  #0,MMU+SS         DISABLE DESCRIPTOR BY WRITING 0 TO SEG. STAT. BIT 0
96
97 000066 4E75                  RTS
98
99
100
101                      * THE FOLLOWING ROUTINE DOES A TASK SWITCH BY SIMPLY CHANGING THE ADDRESS
102                      * SPACE NUMBERS FOR USER DATA SPACE AND USER PROGRAM SPACE IN THE ADDRESS
103                      * SPACE TABLE OF THE MMU(S). THIS MUST BE DONE WHILE THE PROCESSOR IS IN
104                      * SUPERVISORY MODE. TO CHANGE THE SUPERVISOR ADDRESS SPACE NUMBER IT IS
105                      * NECESSARY TO SWITCH TO USER MODE.
106
107
108                      * D0 CONTAINS THE ADDRESS SPACE NUMBERS FOR USER DATA SPACE AND USER PROGRAM
109                      * SPACE IN THE FOLLOWING FORMAT:
110                      *
111                      *              D0 = XX DD XX PP
112                      *
113                      * WHERE XX=DON'T CARE, PP=USER PROGRAM SPACE ASN, DD=USER DATA SPACE ASN
114
115                      * A0 CONTAINS THE ADDRESS OF THE AST LOCATION TO BE CHANGED
116                      * IN THIS CASE, MMU+AST+1.
117
118
119       00000068       TASKSWTCH      EQU     *         SWITCH TO A NEW USER TASK
120
121 000068 2080                 MOVE.L  D0,(A0)           MOVE USER DATA ASN TO AST1 & USER PROGRAM ASN TO AST2
122
123 00006A 4E75                 RTS
124
125
126
127                      * THE FOLLOWING ROUTINE DOES A DIRECT TRANSLATION. THIS OPERATION IS USED
128                      * BY THE OPERATING SYSTEM TO DETERMINE THE ABSOLUTE ADDRESS REPRESENTED BY
129                      * A LOGICAL ADDRESS. THIS IS USEFUL FOR RETRIEVING PARAMETERS FROM USER
130                      * ADDRESS SPACE AND FOR HANDLING DATA TRANSFERS REQUESTED BY A SYSTEM CALL.
131                      * THESE ARE ONLY TWO OF MANY POSSIBLE USES FOR THIS MMU OPERATION.
132
133                      * IN ESSENCE THE MMU DOES AN ADDRESS TRANSLATION. HOWEVER, RATHER THAN
134                      * CONTINUING WITH A MEMORY ACCESS, THE MMU CATCHES THE PHYSICAL ADDRESS
135                      * GENERATED AND SAVES IT IN ACCUMULATOR 4 & 5 OF THE MMU THAT DID THE
136                      * TRANSLATION. THIS REPRESENTS BITS 8 THROUGH 23 OF THE ADDRESS. BITS 0
137                      * THROUGH 7 GENERALLY BYPASS THE MMU AND ARE NOT MAPPED.
138
139                      * A0 CONTAINS THE 16-BIT LOGICAL ADDRESS TO BE TRANSLATED
140                      * D0 CONTAINS THE ADDRESS SPACE NUMBER IN WHICH TO DO THE TRANSLATION
141
142                      * ON RETURN, D1 CONTAINS THE STATUS INDICATING IF THE TRANSLATION WAS SUCCESSFUL
143
144
145       0000006C       DTRAN   EQU     *                PERFORM A DIRECT TRANSLATION
146
147 00006C 33C800FF0020         MOVE    A0,MMU+AC0        MOVE 16-BIT LOGICAL ADDRESS TO ACCUMULATOR 0 & 1
148 000072 13C000FF0026         MOVE.B  D0,MMU+AC6        MOVE 8-BIT ADDRESS SPACE NUMBER TO ACCUMULATOR 6
149
150 000078 123900FF003D         MOVE.B  MMU+DT,D1         READ MMU ADDRESS TO PERFORM DIRECT TRANSLATION
151
152 00007E 4E75                 RTS                       STATUS RETURNED IN D1, PHYS. ADDR. IN ACCUM. 4 & 5
153
154                             END
```

Figure 6. 68451 Utility Software Routines (Cont'd).

bits that come from the Physical Base Address Register. Zeros indicate bits of the logical address from the CPU that are passed through the MMU to the physical address bus. As more bits of the logical address are used both the addressing range and the segment size increase.

Memory Liberation

When a memory segment is freed or liberated it is recombined with its buddy if the buddy is also free. This is a recursive type of operation that continues until a free segment doesn't have a free buddy. This process can be time-consuming and becomes another factor that affects overall system performance.

In order to reduce system overhead needed to recombine buddies we could stop the recombination process when the segment has reached a size that is optimum for our particular system. This size will be one that satisfies most of the memory requests received. However, the system will use additional time to fill requests that are larger than this optimum size. This is one of the system variables that will need to be fine-tuned for best performance.

CONCLUSION

The Motorola 68451 effectively supports memory management. The binary buddy system can be easily implemented using the 68451. Memory management is important for today's microprocessor-based systems with their large address spaces and Motorola has the hardware to support it.

REFERENCES

[1] Knuth, D.E. The Art of Computer Programming, Vol. 1, (2nd printing). Addison-Wesley, Reading, Mass., 1968. pp. 435-455.

[2] Hinds, J.A. "An Algorithm for Locating Adjacent Storage Blocks in the Buddy System," Communications of the ACM, Vol. 18, No. 4, 1975, pp. 221,222.

[3] Hirschberg, D.S. "A Class of Dynamic Memory Allocation Algorithms," Communications of the ACM, Vol.16, No. 10, 1973, pp. 615-618.

[4] Knowlton, K.C. "A Fast Storage Allocator," Communications of the ACM, Vol. 8, No. 10, pp. 623-625.

[5] Peterson, J.L. and Norman, T.A. "Buddy Systems," Communications of the ACM, Vol. 20, No. 6, 1977, pp. 421-431.

[6] Purdom, P.W., and Stigler, S.M. Statistical Properties of the Buddy System. J. ACM 17, 4 (Oct. 1970), pp. 683-697.

[7] Scales III, H.L. "Implementing a Virtual Memory System using the MC68451 Memory Management Unit," Conference Proceedings, Wescon, 1981.

[8] Shen, K.K. and Peterson, J.L. "A Weighted Buddy Method for Dynamic Storage Allocation," Communications of the ACM, Vol. 17, No. 10, 1974, pp. 558-562.

[9] Stockton, J.F. "The MC68451 Memory Management Unit," Computer Design, Publication Pending.

[10] Taylor, M.B. "Efficient Memory Allocation with the Binary Buddy Algorithm," Motorola, November 1981.

[11] Tremblay J.P. and Sorenson, P.G. An Introduction to Data Structures with Applications, McGraw-Hill, New York, N.Y., 1976, pp. 442-463.

[12] "MC68451 Advance Information Data Sheet", Motorola Inc., Austin, Tex., September 1981.

MC68451 MEMORY MANAGEMENT UNIT

F.1 INTRODUCTION

The MC68451 Memory Management Unit (MMU) provides address translation and protection for the 16 megabyte addressing range of the MC68000 processor. Each bus master (or processor) in the M68000 family provides a function code and an address during each bus cycle. The function code specifies an address space and the address specifies a location within that address space. The function codes distinguish between user and supervisor spaces and, within these, between data and program spaces. This separation of address spaces provides the basis for memory management and protection by the operating system. Provision is also made for other bus masters, such as the MC68450 Direct Access Controller (DMAC), to have separate address spaces for efficient direct memory access. A multitasking operating system is simplified and reliability is enhanced through the use of a memory management unit.

The MC68451 is the basic element of a memory management mechanism in an MC68000-based system. The operating system is responsible for ensuring the proper execution of user tasks in the system environment, and memory management is basic to this responsibility. The memory management mechanism provides the operating system with the capability to allocate, control, and protect the system memory. A block diagram of a memory management mechanism using a single MMU is shown in Figure F-1.

A memory management mechanism implemented with one or more MC68451 MMUs can provide address translation, separation, and write protection for the system memory. The memory management mechanism can be programmed to cause an interrupt when a chosen section of memory is accessed, and can directly translate a logical address into a physical address, making it available to the processor for use by the operating system. Using these features, the memory management mechanism can provide separation and security for user programs and allow the operating system to manage the memory in an efficient fashion for multitasking.

F.2 MEMORY SEGMENTS

The memory management mechanism partitions the logical address space into contiguous pieces called segments. Each segment is a section of the logical address space of a task which is mapped via the memory management mechanism into the physical address space. Each task may have any number of segments. Segments may be defined as user or supervisor, data-only or program-only, or program and data. They may be accessed by only one task or shared between two or more tasks. In addition, any segment can be write protected to ensure system integrity. If an undefined segment is accessed, a FAULT is generated by the MMU and applied to the bus error input of the processor.

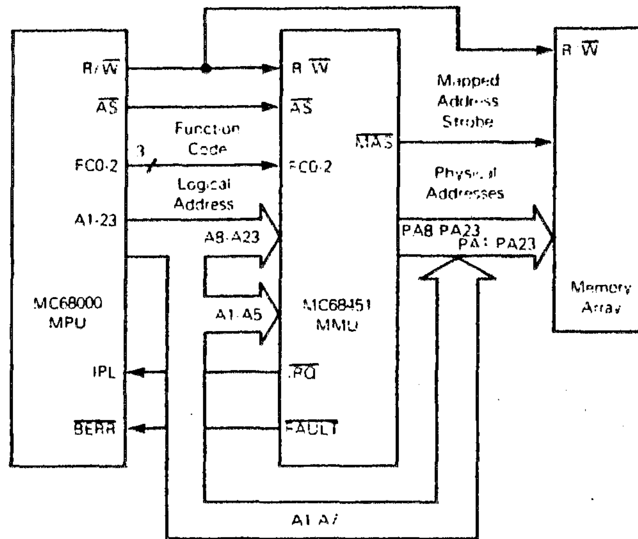


Figure F-1. Memory Management Mechanism Block Diagram

F.3 FUNCTION CODES AND ADDRESS SPACES

Each bus master in the M68000 family provides a function code during each bus cycle to indicate the address space to be used for that cycle. The address bus then specifies a location within this address space for the operation taking place during that bus cycle.

The function codes appear on the FC0-FC2 lines of the MC68000 and divide the memory references into two logical address spaces — the supervisor and the user spaces. Each of these is further divided into program and data spaces. A separate address space is also provided for interrupt acknowledge bus cycles giving a total of five defined function codes.

In addition to the 3-bit function code provided by the MC68000, the MMU also allows a fourth function code line (FC3) which provides for the possibility of another bus master in the system. In this case, FC3 would be tied to bus grant acknowledge input of the MC68000 to enable a second set of eight function codes. This raises the total number of possible function codes to sixteen. If there is only one bus master (the MPU), the FC3 pin of the MMU should be tied low and only eight address spaces used.

F.4 ADDRESS SPACE NUMBERS

To separate the address spaces of different tasks, each address space is given an identifying number. This should not be confused with the address space indicated by the function code. Each function code defines a unique address space and within each of these there can exist a number of different tasks. Each of these tasks needs an address space number (ASN) to distinguish it from the other tasks with which it may share an address space.

The address space numbers are kept in the MMU in a set of registers called the address space table (AST). The AST contains an 8-bit entry for each possible function code (16). Each entry can be assigned an address space number and, during a bus cycle, the function code is used to index into this table to select the cycle address space number. This number is then associatively compared with the address space number in each descriptor to attempt to find a match.

F.5 DESCRIPTORS

Address translation is done using descriptors. A descriptor is a set of six registers (nine bytes) which describes a memory segment and how that segment is to be mapped to the physical addresses. Each descriptor contains base addresses for the logical and physical spaces of each segment. These base addresses are then masked with the logical address masks. The size of the segment is then defined by "don't cares" in the low-order bits of the masks. This method allows segment sizes from a minimum of 256 bytes to a maximum of 16 megabytes in binary increments (i.e., powers of two). This also forces both logical and physical addresses of segment boundaries to lie on a segment size boundary. That is, a segment can only start on an address which is a multiple of 2^k. The segments can be defined so that they are physically shared between tasks. A functional block diagram of an MC68451 MMU is shown in Figure F-2.

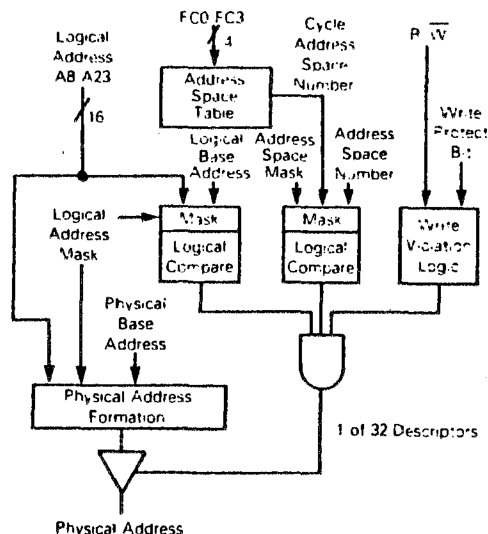


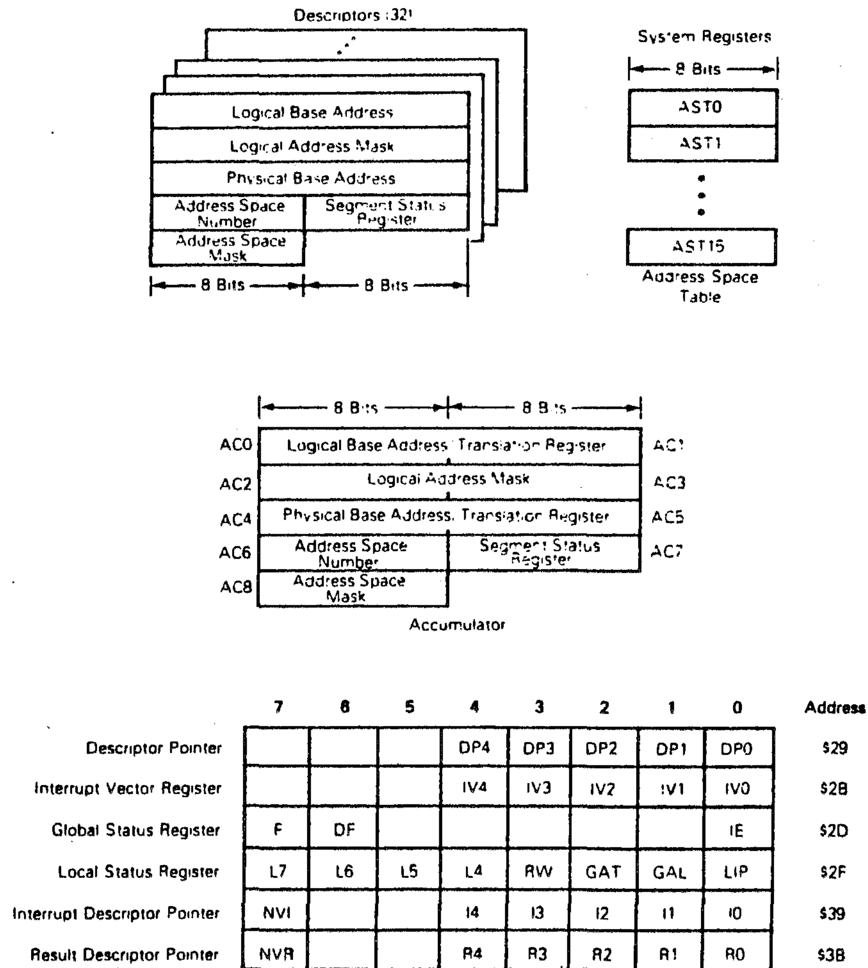
Figure F-2. MC68451 Functional Block Diagram

During normal translation, the MMU translates the logical address provided by the MC68000 to a physical address which is then presented to the memory array. This is accomplished by matching the logical address with the information in the descriptors and then mapping it into the physical address space.

The logical address is composed of address lines A1-A23 as shown in the memory management mechanism block diagram, Figure F-1. The upper 16 bits of this address (A8-A23) are translated by the MMU and mapped into a physical address (PA8-PA23). The lower seven bits of the logical address (A1-A7) bypass the MMU and become the low-order physical address bits (PA1-PA7).

F.6 MMU REGISTER DESCRIPTION

A programmer's model of the MMU is shown in Figure F-3. The MMU register consists of two groups: the descriptors and the system registers. Each of the 32 descriptors is nine bytes long and defines one memory segment.



	7	6	5	4	3	2	1	0	Address
Descriptor Pointer				DP4	DP3	DP2	DP1	DP0	$29
Interrupt Vector Register				IV4	IV3	IV2	IV1	IV0	$2B
Global Status Register	F	DF						IE	$2D
Local Status Register	L7	L6	L5	L4	RW	GAT	GAL	LIP	$2F
Interrupt Descriptor Pointer	NVI			I4	I3	I2	I1	I0	$39
Result Descriptor Pointer	NVR			R4	R3	R2	R1	R0	$3B

Figure F-3. MC68451 Programmers Model

The system registers contain both information local to the MMU and information global to the memory management mechanism. Each bit in the system registers and the segment status registers, except the address space table, is one of four types:

Control	Control bits can be set or cleared by the processor to select MMU options. These are read/write bits.
Status Alterable	Status alterable bits are set or cleared by the MMU to indicate status information. These are also read/write bits.
Status Unalterable	Status unalterable bits are set or cleared by the MMU to reflect status information. These bits cannot be written by the processor.
Reserved	Reserved bits are reserved for future expansion. They cannot be written and are zero when read.

The system registers are all directly addressable from the physical address space. Accessing these registers causes certain operations to be performed. The descriptors are not directly addressable, but are accessed using the descriptor pointer and the accumulator.

In the following discussion, a segment access is defined as a successful match occurring on a segment during normal translation.

F.6.1 DESCRIPTORS. Each MMU contains 32 descriptors (0-31), each of which can define one memory segment. A descriptor is loaded by the processor using the accumulator and descriptor pointer with a load descriptor operation. The segment status register (SSR) can be written to indirectly by the processor using the descriptor pointer. Each descriptor consists of the following registers:

- Logical Base Address (LBA)
- Logical Address Mask (LAM)
- Physical Base Address (PBA)
- Address Space Number (ASN)
- Address Space Mask (ASM)
- Segment Status Register (SSR)

F.6.1.1 Logical Base Address (LBA). The logical base address register is a 16-bit register which, together with the logical address mask, defines the logical addressing range of a segment. This is typically the first address in the segment, although it can be any address within the range defined by the logical address mask.

F.6.1.2 Logical Address Mask (LAM). The logical address mask is a 16-bit mask which defines the bit positions in the logical base address register which are to be used for range matching. Ones, in the mask, mark significant bit positions while zeroes indicate "don't care" positions. A range match occurs if, in each bit position in the logical address mask which is set to one, the logical base address register matches the incoming logical address. The matching function is depicted schematically in Figure F-4.

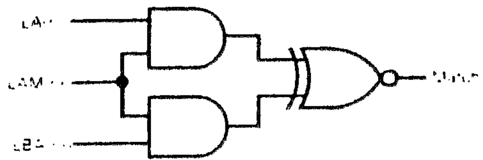


Figure F-4. Schematic Representation for Address Matching

F.6.1.3 Physical Base Address (PBA). The physical base address register is a 16-bit register which, together with the logical address mask and the incoming logical address, is used to form the physical address. The logical address is passed through to the physical address in those bit positions of the logical address mask which contain zeroes (the "don't cares") and the physical base address is gated out in those positions which contain ones. A schematic representation of the physical address generation mechanism is shown in Figure F-5.

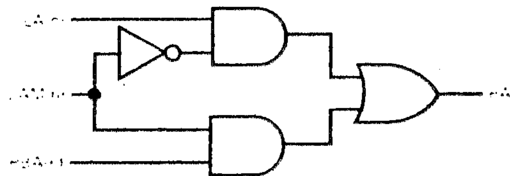


Figure F-5. Schematic Representation of Physical Address Generation

F.6.1.4 Address Space Number (ASN). The address space number is an 8-bit number which, together with the address space mask, is used in detecting a match with the cycle address space number.

F.6.1.5 Address Space Mask (ASM). The address space mask is an 8-bit mask which defines the significant bit positions in the address space number to be used in descriptor matching. As in the logical address mask, the bit positions which are set are used for matching and the bit positions that are clear are "don't cares." A space match occurs if, in the significant bit positions, the cycle address space number matches the address space number. Address space matching is schematically similar to logical address matching as shown in Figure F-4.

F.6.1.6 Segment Status Register (SSR). Each descriptor has an 8-bit segment status register. The segment status register can be written to in two ways: using the load descriptor operation or indirectly using the descriptor pointer in a write status register operation. Each bit is labeled as control or status alterable. Bits 5 and 6 are reserved for future use.

U U (Used) is set by the MMU if the segment was accessed since it was defined. This bit is status alterable.

Set: a) by a Segment access (successful translation using the segment)
b) by an MPU write of "1"

Cleared: a) Reset (in segment #0 of master)
b) MPU write of "0"

I If the I (Interrupt) control bit is set, an interrupt is generated upon accessing the segment.

Set: a) MPU writes "1"

Cleared: a) MPU writes "0"
b) Reset (segment #0 of master)

IP IP (Interrupt Pending) is set if the "I" bit is set when the segment is accessed. $\overline{IRQ}$out is asserted if an IP bit, in one or more SSRs, is set and IE in the global status register is set. $\overline{IRQ}$out is negated when all the IP bits in all SSRs are clear or IE is cleared. IP is status alterable and should be cleared by the interrupt service routine.

Set: a) Segment access and "1" is set
b) MPU writes "1"

Cleared: a) MPU writes a "0"
b) Reset (in segment #0 of master)
c) E bit is a "0"

M The M (Modified) bit is set by the MMU if the segment has been written to since it was defined. The M bit is status alterable.

Set: a) Successful write to the segment
b) MPU writes a "1"

Cleared: a) MPU writes a "0"
b) Reset (segment #0 in master)

WP If the WP (Write Protect) control bit is set, the segment is write protected. A write access to the segment with WP set will cause a write violation.

Set: a) MPU writes a "1"

Cleared: a) MPU writes a "0"
b) Reset (segment #0 in master)

E E (Enable) is a control bit which, when set, enables the segment to participate in the matching process. E can be cleared (the segment disabled) by a write to the SSR, but a load descriptor operation must be performed to set it.

Set: a) Load descriptor with AC7, bit #0
b) Reset (segment #0 in master)

Cleared: a) MPU writes a "0"
b) Unsuccessful load descriptor operation on this descriptor
c) Load descriptor operation with AC7, bit #0 clear

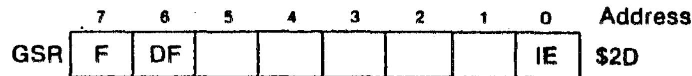
F.6.2.2. Accumulator (AC0-AC8). The accumulator (Figure F-3) is used to access the descriptors, perform direct translation, and latch information during a fault. The accumulator consists of nine 8-bit registers. The register assignments for each operation in which it participates is shown in Table F-1.

The contents of the accumulator can be either local or global depending on the preceding operations. The global accumulator for load and global accumulator for translate bits in the local status register (LSR) indicate whether the information in the accumulator is sufficiently global to perform a load descriptor or direct translation operation.

Table F-1. Accumulator Assignments for Operation

Register Assignment	Load/Read Descriptor	Direct Translation	Normal Translation (Fault)
AC0	Logical Base Address (MSB)	Logical Translation Register (MSB)	Logical Address (MSB)
AC1	Logical Base Address (LSB)	Logical Translation Register (LSB)	Logical Address (LSB)
AC2	Logical Address Mask (MSB)		
AC3	Logical Address Mask (LSB)		
AC4	Physical Base Address (MSB)	Physical Translation Register (MSB)	
AC5	Physical Base Address (LSB)	Physical Translation Register (LSB)	
AC6	Address Space Number	Address Space Number	Cycle Address Space Number
AC7	Segment Status		
AC8	Address Space Mask		

F.6.2.3 Global Status Register (GSR). The global status register is an 8-bit register used to reflect faults and to enable interrupts from an MMU. All MMUs maintain identical information in their global status registers. Bits 1, 2, 3, 4, and 5 are reversed for future use. The organization of the global states register is shown below.



F — F (Fault) is a status alterable bit that is set by the MMU whenever $\overline{\text{FAULT}}$in is detected. Clearing the F bit automatically clears bits L4-L7 in the local status register.

Set:
- a) Write violation detected in this MMU
- b) $\overline{\text{FAULT}}$in detected (write violation in another MMU)
- c) ALLin detected (Undefined Segment Access)
- d) MPU writes a "1"

Cleared:
- a) Reset asserted
- b) MPU writes a "0"

DF — DF (Double Fault) is set if a $\overline{\text{FAULT}}$in signal was detected with F set. DF is a status alterable bit.

Set:
- a) $\overline{\text{FAULT}}$in detected and F was previously set
- b) MPU writes a "1"

Cleared:
- a) Reset
- b) MPU writes a "0"

F.6.2 SYSTEM REGISTERS. The system registers consist of:

Address Space Table (AST)	Descriptor Pointer (DP)
Accumulator (AC0-AC8)	Result Descriptor Pointer (RDP)
Global Status Register (GSR)	Interrupt Descriptor Pointer (IDP)
Local Status Register (LSR)	Interrupt Vector Register (IVR)

F.6.2.1 Address Space Table (AST). Each MMU has a local copy of the address space table. This table is organized as sixteen 8-bit, read/write registers located starting at address $00. Each entry is programmed by the operating system with a unique address space number, each of which is associated with a task. During a memory access, the MMU receives a 4-bit function code (FC0-FC3) which is used to index into the address space table to select the cycle address space number. This number is then used to check for a match with the address space number in each of the 32 segment descriptors.

Only the MC68000 microprocessor and the MC68450 direct memory access controller only provide a 3-bit function code. In a system with more than one bus master, the bus grant acknowledge signal from the processor could be inverted and used as the fourth bit, FC3. This would result in the address space table organization shown in Figure F-6.

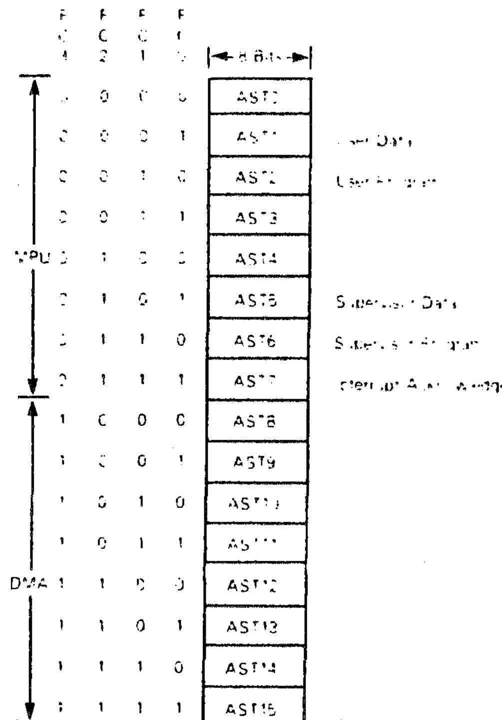


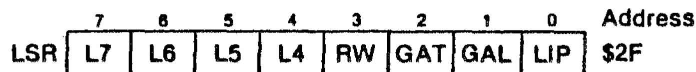
Figure F-6. Address Space Table Organization

IE If IE (Interrupt Enable) is set, the interrupt-request line is enabled. This is a read/write control bit.

Set: a) MPU writes a "1"

Cleared: a) Reset
b) MPU writes a "0"

F.6.2.4 Local Status Register (LSR). The local status register is an 8-bit register which reflects information local to its MMU. The local status register can be globally written but the global accumulator for load, global accumulator for translate, and local interrupt pending bits will not be affected. Bits L4-L7 are cleared if the fault line in the global status register is cleared. All bits in the local status register are cleared on reset. The organization of the local status register is shown below.



RW RW is a status alterable bit which reflects the state of the R/$\overline{W}$ pin at the time $\overline{\text{FAULT}}$in is asserted.

Set: a) MPU writes a "1"
b) Read of segment when F in SSR is set

Cleared: a) Reset
b) MPU writes a "0"
c) Write of segment when F in SSR is set

GAT GAT (Global Accumulator for Translate) is set by the MMU if AC0, AC1, and AC6 are globally consistent.

Set: a) If AC0, AC1, and AC6 are globally consistent (they were last modified as a result of a global write)

Cleared: a) Reset
b) If AC0, AC1, and AC6 are not globally consistent

GAL GAL (Global Accumulator for Load) is set if AC0, AC1, AC2, AC3, AC6, and AC8 are globally consistent.

Set: a) If AC0, AC1, AC2, AC3, AC6, and AC8 are globally consistent

Cleared: a) Reset
b) If AC0, AC1, AC2, AC3, AC6, and AC8 are not globally consistent

LIP LIP (Local Interrupt Pending) is set if one or more descriptors have IP set in their segment status registers.

Set: a) If IP is set in any descriptor

Cleared: a) Reset
b) If all IP bits are clear

L4-L7 The status information encoded in L4-L7 reflects the status of the MMU after the last event (an operation or fault). These bits are encoded and changed as a unit. They are cleared whenever the F bit in the GSR is cleared and are alterable by the MPU.

L7	L6	L5	L4		
0	0	0	0	NO	The MMU was not the source of the last event.
1	0	0	0	DT	A direct translation was locally successful. A match was found in one of the MMUs descriptors.
1	0	0	1	LD	A load descriptor fault occurred. A previously defined descriptor conflicts with the descriptor being loaded.
1	0	1	0	USA	An undefined segment access was attempted. The logical address was not matched in any descriptor in the MMU.
1	1	0	0	WV	A write violation occurred. A segment defined in this MMU was write protected and a write to that memory segment was attempted. The NVR bit in the RDP will show whether the USA or WV occurred in this MMU.

Set:
a) Various bits set if DT, LD, USA, or WV occur
b) MPU writes a "1"

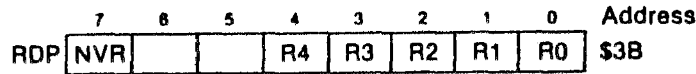
Cleared:
a) Reset
b) MPU writes a "0"
c) When F bit in GSR is cleared
d) If MMU was not the source of the last event (NO)

F.6.2.5 Descriptor Pointer (DP). The descriptor pointer is an 8-bit read/write pointer register located at address $29. The five low-order bits identify the descriptor to be used in the load descriptor, read segment status (transfer descriptor), and write segment status operations. Bits 5, 6, and 7 are reserved.

The descriptor pointer is initialized to $00 on reset. It can be globally written by the processor. The descriptor pointer is loaded by the memory management mechanism with the number of the descriptor matched in a direct translation operation to allow a subsequent transfer descriptor operation to load the matched descriptor into the accumulator.

F.6.2.6 Result Descriptor Pointer (RDP). The result descriptor pointer is an 8-bit, read-only register that identifies a descriptor involved in the following events: a write violation, a load descriptor failure, or a direct translation success. The result descriptor pointer is loaded from a priority encoder which determines the highest priority descriptor involved. For example, in a load descriptor operation, more than one descriptor currently in the MMU may collide with the descriptor being loaded. Only the number of the highest priority descriptor will be loaded into the result descriptor pointer. Descriptor 0 is considered to be the highest priority and 31 is the lowest.

The bit assignments are shown below. Bits 5 and 6 are reserved. The result descriptor pointer is initialized to $80 on reset.



	7	6	5	4	3	2	1	0	Address
RDP	NVR			R4	R3	R2	R1	R0	$3B

NVR If no descriptor is selected by the priority encoder when the RDP is loaded, NVR (No Valid Result) is set, otherwise it is cleared. This bit is status unalterable.

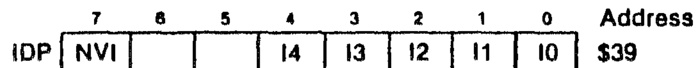
Set: a) Reset
b) No result from WV, LD, or DT

Cleared: a) A WV, LD failure of DT success in this MMU

R0-
R4 R0-R4 encode the number of the descriptor selected by the priority encoder

F.6.2.7 Interrupt Descriptor Pointer (IDP). The interrupt descriptor pointer is an 8-bit read-only register that is read to determine which descriptor caused an interrupt. Each time it is read, the interrupt descriptor pointer is loaded from the priority encoder with the highest-priority descriptor which has the interrupt pending bit in its segment status register set. If no descriptor has an interrupt pending bit set, the no valid interrupt bit is set.

The bit assignment is shown below. Bits 5 and 6 are reserved.



	7	6	5	4	3	2	1	0	Address
IDP	NVI			I4	I3	I2	I1	I0	$39

NVI NVI is set if no descriptor has IP set, otherwise it is cleared.

I0-I4 These bits encode the number of the descriptor selected by the priority encoder.

F.6.2.8 Interrupt Vector Register (IVR). The interrupt vector register is an 8-bit read/write register containing the interrupt vector. Its contents are put on data lines D0-D7 during the interrupt acknowledge operation to provide the processor with a vector number. The interrupt vector register is initialized to $0F (the MC68000 uninitialized-device vector number) on reset.

F.7 MMU OPERATIONS

Table F-2 shows the operations which can be performed. Each operation is initiated by the access of an address given on the register select lines RS1-RS5 and the upper and lower data strobes. The access can be from either the logical or physical address bus. In a multiprocessor system, an external processor could access the memory management mechanism from the physical address bus. If the access is from the logical address bus, an address translation is first performed. If the access is from the physical address bus, the operation state is entered directly from the idle state.

Table F-2. Summary of MMU Functions

Function	Summary
Idle	The MMU backs off the bus to prepare for a new access
Reset	The MMU is pre-emptively initialized.
Normal Translation	The MMU attempts to translate an access from the logical address bus
Operations	The MMU is accessed from the logical or physical bus.
Write System Registers	An operation to globally write system registers
Read System Registers	An operation to read the system registers
Write Segment Status	The SSR of a descriptor can be quickly changed using this operation. The enable bit cannot be set using it, however
Load Descriptor	With this operation, the contents of the accumulator are loaded into the descriptor pointed to by the descriptor pointer.
Transfer Descriptor	This operation transfers the contents of the selected descriptor into the accumulator.
Direct Translation	An operation to globally translate a logical address for the operating system.
Interrupt Acknowledge	An operation that supplies a vector number to the MPU in response to IACK.

The operation phase is always entered with PAD0-PAD15 in the high-impedance state and either (in the case of an operation following a normal translation) one MMU asserting $\overline{\text{HAD}}$ to hold the physical address, or (in the case of an access from the physical bus) the external processor holding the address. If both chip select and either the upper or lower data strobe is asserted or interrupt acknowledge and interrupt request in are asserted, the MMU asserts $\overline{\text{ED}}$ to enable the data transceivers.

If interrupt acknowledge and interrupt request in are asserted, an interrupt acknowledge operation is performed. If chip select and either the upper or lower data strobe is asserted, the memory management mechanism determines which operation to perform by decoding the register select lines and the read/write line. These signals tell which register is associated with the operation, which operation to perform, and whether the operation is local or global.

After each operation, data transfer acknowledge is asserted to indicate to the processor that the operation is finished. When the processor negates the data strobe, data transfer acknowledge and $\overline{\text{ED}}$ are rescinded and PAD0-PAD15 are placed in the high-impedance state. If address strobe is negated, or had been negated since the last normal translation, the MMU enters the idle state.

After the data transfer acknowledge handshake, if address strobe remains asserted and chip select and either the upper or lower data strobe is asserted, another master operation is performed. If address strobe remains asserted and $\overline{\text{GOin}}$ and either the upper or lower data strobe is asserted, another slave operation is performed.

F.7.1 OPERATIONS ADDRESS MAP. Table F-3 shows the operations address map. Each system register has an address at which it can be read or written. In addition, some addresses do not correspond to a register, but rather designate an operation to be performed by reading that location.

The data strobes are logically separate and operations using both are independent. The operation ends when both data strobes are negated.

Some addresses are reserved for future expansion. Any access to an unused location will result in a null operation. If the access is a read, the appropriate byte of the data bus is driven high. If the access is a write, no side-effect occurs.

F.7.2 LOCAL OPERATIONS. Some operations, such as reading the status registers, affect only one MMU. These are called local operations. Local operations include:

- Interrupt Acknowledge
- Read System Register
- Transfer Descriptor
- Write Segment Status Register

F.7.2.1 Interrupt Acknowledge. The interrupt acknowledge operation is performed if interrupt acknowledge and interrupt request in are asserted at the beginning of the operation phase. During interrupt acknowledge, the contents of the interrupt vector register are placed on data lines D0-D7 to provide the processor with a vector number.

F.7.2.2 Read System Register. Each system register has an address at which it can be read. Each MMU should be chip selected at a different location to access the registers in each. During a processor read of the interrupt descriptor pointer, it is first loaded from the priority encoder and then gated onto data lines D0-D7.

F.7.2.3 Transfer Descriptor. In order to read the contents of a descriptor, it must be transferred into the accumulator and read from there. The descriptor pointer is first written by the processor with the number of the descriptor desired. The transfer descriptor operation is then performed by reading from the segment status register address ($31).

The contents of the selected descriptor is then transferred into the accumulator as shown in Table F-1 and the contents of the segment status register are gated onto data lines D0-D7. The descriptor registers may then be read from the accumulator.

F.7.2.4 Write Segment Status Register. The segment status register of any descriptor can be written using the descriptor pointer as a pointer. Any bit may be written except the enable bit. Enable may be cleared using this operation but it may not be set.

F.7.3 GLOBAL OPERATIONS. A global operation is one which is performed in parallel on all MMUs in the system. Global operations include:

- Writes to System Registers
- Load Descriptor Operation
- Direct Translation

In global operation, one MMU must be the master and the rest must be slaves. The operation begins with chip select and either the upper or lower data strobe asserted on one MMU. The MMU with chip select asserted becomes the master for that operation. The master asserts $\overline{\text{GO}}$out and, upon detecting $\overline{\text{GO}}$in as true, the other MMUs become slaves in the operation.

If there is only one MMU present in the system, the $\overline{\text{ANY}}$, ALL, and $\overline{\text{GO}}$ pins must be tied to V_{CC} through pull-up resistors. Global operations then become local only.

F.7.3.1 Write System Register. Each system register that can be written to is written globally. This includes: the accumulator, the address space table, the descriptor pointer, the interrupt vector register, and the local and global status registers. The operation is performed by writing to the desired register's address.

The MMU which has chip select asserted becomes the master by asserting $\overline{\text{GO}}$out. The other MMUs detect $\overline{\text{GO}}$in and become slaves. Each MMU transfers the data on the data bus to the selected register. If the write is to a byte of the accumulator, that register is marked as global. If the fault bit in the global status register is clear, local status register bits L4-L7 are also clear.

When the transfer is completed in each MMU, each will assert ALLout. After all MMUs have asserted ALLout, ALLin will be true and, upon detecting ALLin, the master rescinds $\overline{\text{GO}}$.

F.7.3.2 Load Descriptor Operation. Descriptors are loaded by transferring the contents of the accumulator to the descriptor after performing global checks for collisions. A collision exists when two or more enabled descriptors are programmed to translate the same logical address.

To prepare for descriptor loading, the accumulator must be loaded globally with the logical base address, logical address mask, address space number, and address space mask. To make global collision checks, accumulators AC6 and AC8 must have been globally loaded. If they are, the global accumulator for load bit in the local status register of each MMU is set. To initiate the operation, a read from the address $3F is done. If the load is successful, the data bus will be set to $00. If a collision is found, the load is unsuccessful and the data bus is set to $FF.

During the load descriptor operation, the MMU with chip select asserted becomes the master by asserting $\overline{\text{GO}}$out. The other MMUs detect $\overline{\text{GO}}$in and become slaves. The slave MMUs decode the operation from the register select lines, the read/write line, and the data strobes. The descriptor whose number is in the descriptor pointer is disabled (its enable bit is cleared so that it cannot cause a collision).

If the global accumulator for load bit in the global status register of a slave is clear, bits LA4-LA7 in the local status register are encoded to indicate that a load descriptor fault has occurred and $\overline{\text{ANY}}$out is asserted. If global accumulator for load is set, the slave checks the enabled descriptors against its accumulator for collisions. If a conflict is found, the slave asserts $\overline{\text{ANY}}$out and loads its result descriptor pointer with the number of the descriptor which caused the collision. If no collision is detected, bits L4-L7 in the local status register are cleared. When $\overline{\text{GO}}$in is detected, ALLout and $\overline{\text{ANY}}$out are negated and the operation ends.

The master aborts the transfer if there is a local descriptor conflict, the global accumulator for load bit is clear, or if $\overline{\text{ANY}}$in is asserted. If the failure was not local, bits L4-L7 in the local status register are cleared. Otherwise, bits L4-L7 are encoded for a load

descriptor fault and $\overline{\text{ANY}}$out is asserted by the master. The master then puts $FF on data lines to indicate a failure to the processor, negates ALLout and $\overline{\text{ANY}}$out, and rescinds $\overline{\text{GO}}$out. When $\overline{\text{ANY}}$in is negated, the operation is terminated.

If there were no local collisions, its global accumulator for load bit was set, and ALLin is asserted, the master completes the transfer and enables the loaded descriptor. It then puts $00 on D0-D7 to indicate success, clears L4-L7, negates ALLout, and rescinds $\overline{\text{GO}}$out.

F.7.3.3 Direct Translations. The memory management mechanism can be used to directly translate the logical address into a physical address and make it available to the processor in the accumulator. The logical address to be translated is globally loaded into accumulator AC0-AC1 and the address space number to be used is loaded into accumulator AC6. Translation is initiated with a read from the address $3D.

If the translation is successful, the descriptor pointer and result descriptor pointer point to the descriptor which performed the translation and the physical address is loaded into accumulator AC4-AC5. The processor reads $00 from the data bus.

If the logical address could not be translated because it was globally undefined, the data bus is set to $FF to indicate the failure.

Using accumulator AC6 to supply the cycle address space number, each MMU attempts to match the logical address contained in accumulator AC0-AC1 with one of its enabled descriptors. Each MMU must have the same information in accumulator AC0, AC1, and AC6. The global accumulator for translate bit in the local status register is set if these registers have each been globally loaded.

If a match is found and global accumulator for translate bit is set, the physical address is formed as in normal translation and put into accumulator AC4-AC5. The result descriptor pointer and descriptor pointer are loaded from the priority encoder and bits L4-L7 in the local status register are encoded to indicate direct translation. The master puts $00 on data lines D0-D7 to signal that the translation was successful and rescinds $\overline{\text{GO}}$ to terminate the operation.

If no match is found, or the global accumulator for translate bit is clear, the MMU asserts ALLout and bits L4-L7 in the local status register are cleared. The master monitors the $\overline{\text{ANY}}$in and ALLin inputs.

If $\overline{\text{ANY}}$in becomes asserted, then another MMU performed the translation. The master puts $00 on data lines D0-D7 to indicate success, negates ALLout, and rescinds $\overline{\text{GO}}$out. It waits until $\overline{\text{ANY}}$in is negated before terminating the operation.

If ALLin becomes asserted, then none of the MMUs performed the translation. The master puts $FF on data lines D0-D7 to indicate failure, negates ALLout, and rescinds $\overline{\text{GO}}$out to terminate the operation. Each slave MMU negates $\overline{\text{ANY}}$out and ALLout when the master MMU rescinds $\overline{\text{GO}}$ at the end of the operation.

F.8 MMU FUNCTIONAL DESCRIPTION

The memory management mechanism is comprised of one or more memory management units. Each MMU is capable of describing thirty-two segments. If more than thirty-two segments are required in the system, more MMUs can be added to increase the number in 32-segment increments.

In order to perform its operations, some of the information in the MMU's registers must be global. That is, it must be duplicated in all the MMUs in the system. For example, the address space table must be global to ensure that the address space numbers are common to all MMUs. To allow this, certain operations are defined as global. Any system register that can be written is written globally. This includes the accumulator, the address space table, the descriptor pointer, the interrupt vector register, the global status register, and the local status register. The result descriptor pointer and the interrupt descriptor pointer are read-only and, therefore, are local and not global.

The $\overline{\text{ANY}}$, ALL, and $\overline{\text{GO}}$ signal lines are used to connect multiple MMUs to form the memory management mechanism. The memory management mechanism uses these input/output signals to communicate information between MMUs and maintain functional unity. The global operation ($\overline{\text{GO}}$) pin is used to establish the master-slave relationship between MMUs for a given operation. The $\overline{\text{ANY}}$ signal is detected as true if any MMU asserts it, allowing MMUs to report conditions that are important in even one device. The ALL signal is detected as true only if all MMUs assert it. It is used to verify that all MMUs in the system have performed some operation or are in the same state. A sample circuit diagram of a two-MMU system is shown in Figure F-7.

During each global operation, one MMU is specified as the master; all others are designated as slaves. The MMU which has its chip select asserted becomes the master by asserting the $\overline{\text{GO}}$out signal. This signals the other MMUs that they are slaves for that operation. Note that all MMUs may be accessed and, therefore, any one may be the master for a given operation.

F.8.1 MMU FUNCTIONAL STATES. At any time, an MMU may be in one of five states:

Reset

Idle

Normal Translation

Local Operations

Global Operations

In a global operation, an MMU may be a master (if the chip select signal is asserted) or a slave (if $\overline{\text{GO}}$in is asserted). In addition, two actions can occur regardless of the current state:

1. If $\overline{\text{RESET}}$ is asserted, the Reset operation begins. The memory management mechanism will remain in the Reset state until $\overline{\text{RESET}}$ is negated.
2. $\overline{\text{IRQ}}$out is asserted if local interrupt pending bit in the local status register and interrupt enable bit in the global status register are set, otherwise it is placed in the high-impedance state and should be negated with a pullup resistor.

F.8.1.1 Reset State. Asserting RESET will initiate the reset sequence regardless of the state of the MMU. During reset, GO, data transfer acknowledge, ED, MAS, HAD, and WIN signals are rescinded. The physical address port, FAULT, and ANY lines are placed in the high-impedance state. Pullup resistors on the FAULT and ANY lines keep these signals negated. The ALL pin is driven low to negate it.

The global status register, local status register, descriptor pointer, and the entire address space table are initialized to $00. The result descriptor pointer is initialized to $80 and the interrupt vector register to $0F. All descriptors are disabled by clearing the enable bits in their segment status registers.

In order to allow the address bus to function before the operating system can initialize the memory management mechanism, one MMU is selected to have descriptor #0 initialized so that it maps any logical address unchanged to the physical address bus. The MMU is selected for this by having its chip select line asserted during Reset. This circuit is shown in the diagram in Figure F-7.

Descriptor zero in the selected MMU will have had its logical address mask and address space number cleared to $00, its address space mask set to $FF, and the enable bit set. Because of this, the logical address passes to the physical address bus (via descriptor zero) without alteration. The enable bits of descriptors 1-31 are cleared to zero to disable them and their contents remain uninitialized. If the MMU is not chip selected during reset, the enable bits in all descriptors are cleared and no descriptor is initialized.

F.8.1.2 Idle States. The idle state is used to terminate bus accesses and prepare for new ones. The MMU is "backed-off" the bus; i.e., the data transceivers are placed in the high-impedance state and the address latches are put into the transparent mode. The outputs are driven to the same levels as in reset except that HAD is rescinded one-half clock after MAS to provide address hold time.

While in the idle state, the MMU uses the function code inputs to index into the address space table to provide the cycle address space number. If address strobe is asserted, a normal translation is performed. If address strobe is negated and chip select, interrupt acknowledge, interrupt request in, GO, and the data strobes indicate an access from the physical bus, an operation is performed.

F.8.2 NORMAL ADDRESS TRANSLATION. At the start of a bus cycle, the processor presents the logical address, read/write signal, and the function code to the memory management mechanism. The function code is used to index into the address space table to select the cycle address space number. When address strobe is asserted, the normal translation phase begins by sending the cycle address space number, the logical address, and the read/write signal to each descriptor for matching.

NOTE

The function codes must be valid before address strobe is asserted to allow for the table lookup. Current versions of the MC68000 provide this setup time; however, early mask set (R9M, T6E) do not. With these early mask sets, address strobe must be delayed to the MMU.

F.8.2.1 Matching. Matches can occur in two areas: range and space.

A range match occurs if, in each bit position in the logical address mask which is set, the incoming logical address matches the logical base address.

A space match occurs if, in each bit position in the address space mask which is set, the cycle address space number matches the address space number.

F.8.2.2 Translation. An address match occurs if there is a range match and a space match. A write violation occurs if a write is attempted to a write-protected segment. If there is an address match in a descriptor and no write violation, the physical address is formed from the physical base address of that descriptor and the logical address. The logical address is passed through in those bit positions in the logical address mask which are clear (the "don't cares"). In the other bit positions, the physical base address is gated out to the physical address bus.

The used and, if the cycle was a write, the modified bits in the segment status register are set. If the interrupt bit is set, then the interrupt pending bit is set. WIN is asserted if the write protect bit is set and the cycle was a read or a read-modify-write. If the cycle was a write, MAS is not asserted to prevent the write from modifying data.

After the physical address is stable, MAS is asserted to indicate a valid address is on the bus. HAD is asserted to hold the address stable on the latches and the PAD0-PAD15 lines are then placed in the high-impedance state. If address strobe is then negated, the cycle has terminated and the MMU returns to the idle state. If address strobe is not negated, the cycle can continue in three ways:

1. Chip select or interrupt acknowledge and interrupt request in are asserted, the MMU will begin an operation as a master.
2. If GOin is detected by an MMU it will begin a slave operation.
3. If a high-to-low transition is detected on the read/write line, indicating a write, address strobe remains asserted and the matched segment is write protected, a write violation occurs. This would be the result of a read/modify/write bus cycle on a protected segment.

F.8.2.2.1 WRITE VIOLATION. If an address match occurs but the bus cycle was a write to a write protected segment, a write violation occurs. In this case, the result descriptor pointer is loaded from the priority encoder, the fault bit is set in the global status register, and the double fault bit is set if the fault bit was previously set. The state of the read/write line is latched into the read/write bit of the local status register and bits L4-L7 are encoded to indicate write violation. The FAULTout signal is then asserted for five clock cycles or until address strobe is negated, whichever is greater.

The logical address is latched into AC0 (MSB) and AC1 (LSB) of the accumulator. The cycle address space number is latched into AC6. These registers are marked as non-global with respect to the global accumulator for translate and global accumulator for load bits. If the FAULT pin has been connected to the bus error pin on the MC68000, address strobe will be negated as the processor begins the bus error exception processing. When address strobe is negated, the MMU will enter the idle state.

WARRANTY

Dual Systems Corporation warrants the equipment covered hereby to be free from defects in material and workmanship for twelve (12) months from date of original shipment to purchaser. During this warranty period Dual Systems will repair or replace defective equipment FOB its place of business without charge to purchaser.

This warranty applies to defects arising out of normal use and service of the equipment as specified by Dual Systems. This warranty does not cover abnormal operation of the equipment, accident, alteration, negligence, misuse and repairs or service performed by other than Dual Systems' authorized representatives. Purchaser shall upon request by Dual Systems furnish reasonable evidence that the defect arose from causes placing a liability on Dual Systems.

The obligation of Dual Systems under this warranty is limited to repair or replacement of the defective equipment and is the only warranty applicable to the equipment. Dual Systems shall not be liable for any injury, loss or damage, direct or consequential, arising out of the use or inability to use the product. No changes in the warranty shall be effective without the prior approval in writing of both parties. This warranty and obligations and liabilities thereunder shall replace all warranties or guarantees express or implied including the implied warranty of merchantability.

Dual Systems Corporation
2530 San Pablo Avenue
Berkeley, California 94702

(415) 549-3854