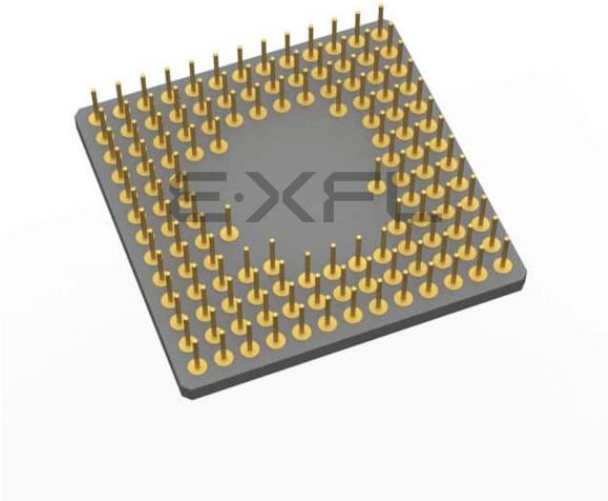


Welcome to E-XFL.COM

Understanding Embedded - Microprocessors

Embedded microprocessors are specialized computing chips designed to perform specific tasks within an embedded system. Unlike general-purpose microprocessors found in personal computers, embedded microprocessors are tailored for dedicated functions within larger systems, offering optimized performance, efficiency, and reliability. These microprocessors are integral to the operation of countless electronic devices, providing the computational power necessary for controlling processes, handling data, and managing communications.

Applications of Embedded - Microprocessors

Embedded microprocessors are utilized across a broad spectrum of applications, making them indispensable in

Details	
Product Status	Obsolete
Core Processor	68030
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	33MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5.0V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	128-BPGA
Supplier Device Package	128-PGA (34.55x34.55)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68030rc33c

Notations for operations that have two operands, written <operand> <op> <operand>, where <op> is one of the following:

- ⇨—The source operand is moved to the destination operand
- ⇔—The two operands are exchanged
- +—The operands are added
- –—The destination operand is subtracted from the source operand
- ×—The operands are multiplied
- ÷—The source operand is divided by the destination operand
- <—Relational test, true if source operand is less than destination operand
- >—Relational test, true if source operand is greater than destination operand
- V—Logical OR
- ⊕—Logical exclusive OR
- Λ—Logical AND

shifted by, rotated by—The source operand is shifted or rotated by the number of positions specified by the second operand

Notation for single-operand operations:

~<operand>—The operand is logically complemented

<operand>sign-extended—The operand is sign extended; all bits of the upper portion are made equal to the high-order bit of the lower portion

<operand>tested—The operand is compared to zero, and the condition codes are set appropriately

Notation for other operations:

TRAP—Equivalent to Format/Offset Word ⇨ (SSP); SSP – 2 ⇨ SSP; PC ⇨ (SSP); SSP – 4 ⇨ SSP; SR ⇨ (SSP); SSP – 2 ⇨ SSP; (vector) ⇨ PC

STOP—Enter the stopped state, waiting for interrupts

If <condition> then <operations> else <operations>—The condition is tested. If true, the operations after "then" are performed. If the condition is false and the optional "else" clause is present, the operations after "else" are performed. If the condition is false and else is omitted, the instruction performs no operation. Refer to the Bcc instruction description as an example.

The CAS and CAS2 instructions together allow safe operations in the manipulation of system linked lists. Controlling a single location, HEAD in the example, manages a last-in-first-out linked list (see Figure 3-2). If the list is empty, HEAD contains the NULL pointer (0); otherwise, HEAD contains the address of the element most recently added to the list. The code fragment shown in Figure 3-2 illustrates the code for inserting an element. The MOVE instructions load the address in location HEAD into D0 and into the NEXT pointer in the element being inserted, and the address of the new element into D1. The CAS instruction stores the address of the inserted element into location HEAD if the address in HEAD remains unaltered. If HEAD contains a new address, the instruction loads the new address into D0 and branches to the second MOVE instruction to try again.

The CAS2 instruction is similar to the CAS instruction except that it performs two comparisons and updates two variables when the results of the comparisons are equal. If the results of both comparisons are equal, CAS2 copies new values into the destination addresses. If the result of either comparison is not equal, the instruction copies the values in the destination addresses into the compare operands.

```
SINSERT                                     ALLOCATE NEW ENTRY, ADDRESS IN A1
        MOVE.L    HEAD,D0                   MOVE HEAD POINTER VALUE TO D0
SILOOP  MOVE.L    D0,(NEXT,A1)              ESTABLISH FORWARD LINK IN NEW ENTRY
        MOVE.L    A1,D1                     MOVE NEW ENTRY POINTER VALUE TO D1
        CAS.L     D0,D1,HEAD                IF WE STILL POINT TO TOP OF STACK, UPDATE THE HEAD POINTER
        BNE       SILOOP                    IF NOT, TRY AGAIN
```

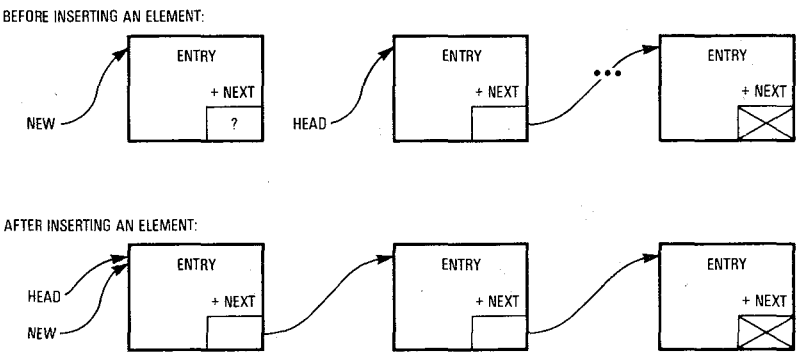


Figure 3-2. Linked List Insertion

5.2 FUNCTION CODE SIGNALS (FC0–FC2)

These three-state outputs identify the address space of the current bus cycle. Table 4-1 shows the relationship of the function code signals to the privilege levels and the address spaces. Refer to **4.2 ADDRESS SPACE TYPES** for more information.

5.3 ADDRESS BUS (A0–A31)

These three-state outputs provide the address for the current bus cycle, except in the CPU address space. Refer to **4.2 ADDRESS SPACE TYPES** for more information on the CPU address space. A31 is the most significant address signal. Refer to **7.1.2 Address Bus** for information on the address bus and its relationship to bus operation.

5

5.4 DATA BUS (D0–D31)

These three-state bidirectional signals provide the general-purpose data path between the MC68030 and all other devices. The data bus can transfer 8, 16, 24, or 32 bits of data per bus cycle. D31 is the most significant bit of the data bus. Refer to **7.1.4 Data Bus** for more information on the data bus and its relationship to bus operation.

5.5 TRANSFER SIZE SIGNALS (SIZ0, SIZ1)

These three-state outputs indicate the number of bytes remaining to be transferred for the current bus cycle. With A0, A1, $\overline{\text{DSACK0}}$, $\overline{\text{DSACK1}}$, and $\overline{\text{STERM}}$, SIZ0 and SIZ1 define the number of bits transferred on the data bus. Refer to **7.2.1 Dynamic Bus Sizing** for more information on the size signals and their use in dynamic bus sizing.

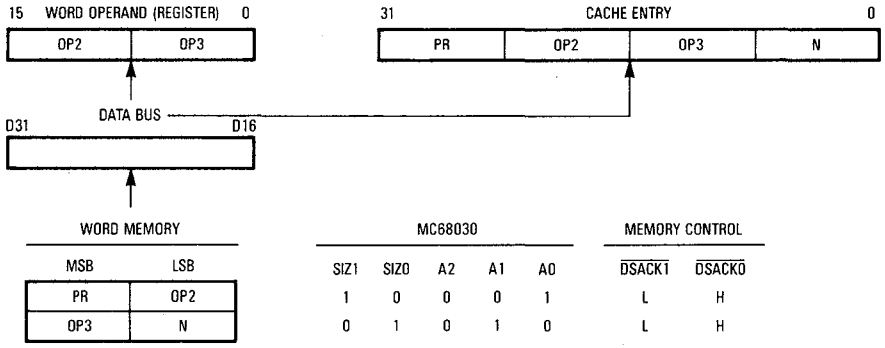


Figure 7-14. Example of Misaligned Cachable Word Transfer from Word Bus

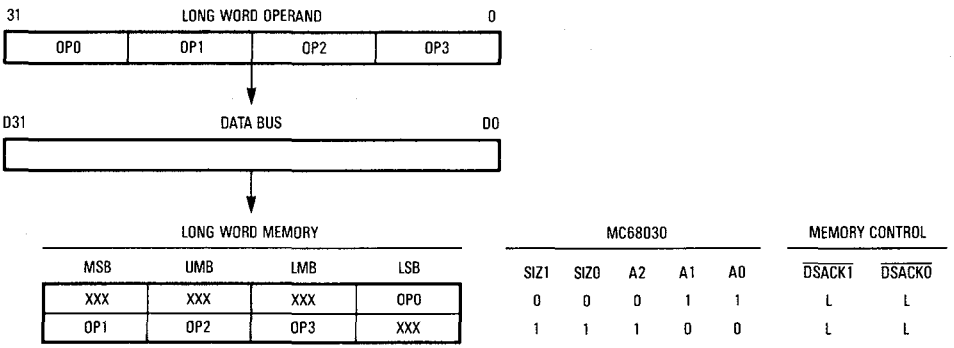


Figure 7-15. Misaligned Long-Word Transfer to Long-Word Port

Table 7-6 shows that the processor always prefetches instructions by reading a long word from a long-word address (A1:A0 = 00), regardless of port size or alignment. When the required instruction begins at an odd-word boundary, the processor attempts to fetch the entire 32 bits and loads both words into the instruction cache, if possible, although the second one is the required word. Even if the instruction access is not cached, the entire 32 bits are latched into an internal cache holding register from which the two instructions words can subsequently be referenced. Refer to **SECTION 11 INSTRUCTION EXECUTION TIMING** for a complete description of the cache holding register and pipeline operation.

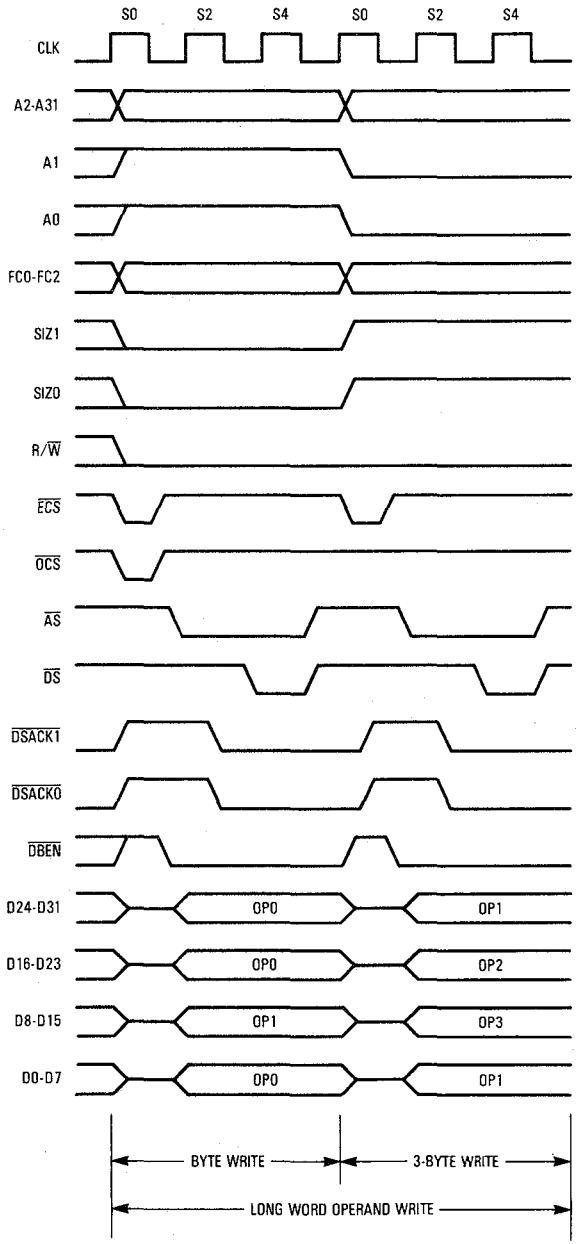


Figure 7-16. Misaligned Write Cycles to Long-Word Port

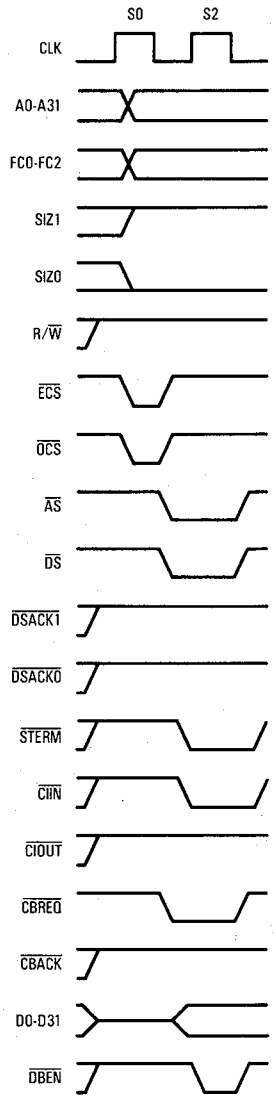


Figure 7-32. Synchronous Read with CIIN Asserted and CBACK Negated

State 7

During this state, the processor negates $\overline{\text{CBREQ}}$, and the memory device may negate $\overline{\text{CBACK}}$. Aside from this, all other bus signals driven by the processor remain driven. The same hold times for $\overline{\text{STERM}}$ and data described for S3 apply here.

State 8

This state is identical to S4 except that $\overline{\text{CBREQ}}$ is negated, indicating that the processor cannot continue to accept more data after this. The data latched at the end of S8 corresponds to the fourth long word of the burst.

State 9

The processor negates $\overline{\text{AS}}$, $\overline{\text{DS}}$, and $\overline{\text{DBEN}}$ during S9. It holds the address, R/$\overline{\text{W}}$, SIZ0–SIZ1, and FC0–FC2 valid throughout S9. The same hold times for data described for S3 apply here.

Note that the address bus of the MC68030 remains driven to a constant value for the duration of a burst transfer operation (including the first transfer before burst mode is entered). If an external memory system requires incrementing of the long-word base address to supply successive long words of information, this function must be performed by external hardware. Additionally, in the case of burst transfers that cross a 16-byte boundary (i.e., the first long word transferred is not located at A3/A2 = 00), the external hardware must correctly control the continuation or termination of the burst transfer as desired. The burst may be terminated by negating $\overline{\text{CBACK}}$ during the transfer of the most significant long word of the 16-byte image (A3/A2 = 11) or may be continued (with $\overline{\text{CBACK}}$ asserted) by providing the long word located at A3/A2 = 00 (i.e., the count sequence wraps back to zero and continues as necessary). The MC68030 caches assume the higher order address lines (A4–A31) remain unchanged as the long-word accesses wrap back around to A3/A2 = 00.

7.4 CPU SPACE CYCLES

FC0–FC2 select user and supervisor program and data areas as listed in Table 4-1. The area selected by FC0–FC2 = $7 is classified as the CPU space. The interrupt acknowledge, breakpoint acknowledge, and coprocessor communication cycles described in the following sections utilize CPU space.

9.5.1.10 LONG-FORMAT INVALID DESCRIPTOR. The long-format invalid descriptor is used in pointer and page tables that contain long-format descriptors. It is used in the same way as the short-format invalid descriptor in the preceding section. The first long word contains the DT field in the lowest order bits. The second long word is an unused field, also available to the operating system. Figure 9-16 shows the format of the long-format invalid descriptor.

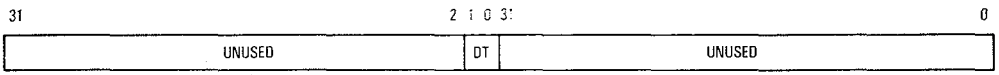


Figure 9-16. Long-Format Invalid Descriptor

9.5.1.11 SHORT-FORMAT INDIRECT DESCRIPTOR. The short-format indirect descriptor does not have a unique descriptor-type code. Rather, it resides in a page table (the bottom level of the address translation tree) that contains short-format descriptors and is neither a page descriptor nor an invalid descriptor. The descriptor-type field contains either the code for a valid 4-byte descriptor or for a valid 8-byte descriptor, depending upon the size of the referenced page descriptor. The field descriptions in **9.5.1.1 DESCRIPTOR FIELD DEFINITIONS** apply to the corresponding fields of this descriptor. Figure 9-17 shows the format of a short-format indirect descriptor.

Figure 9-17. Short-Format Indirect Descriptor

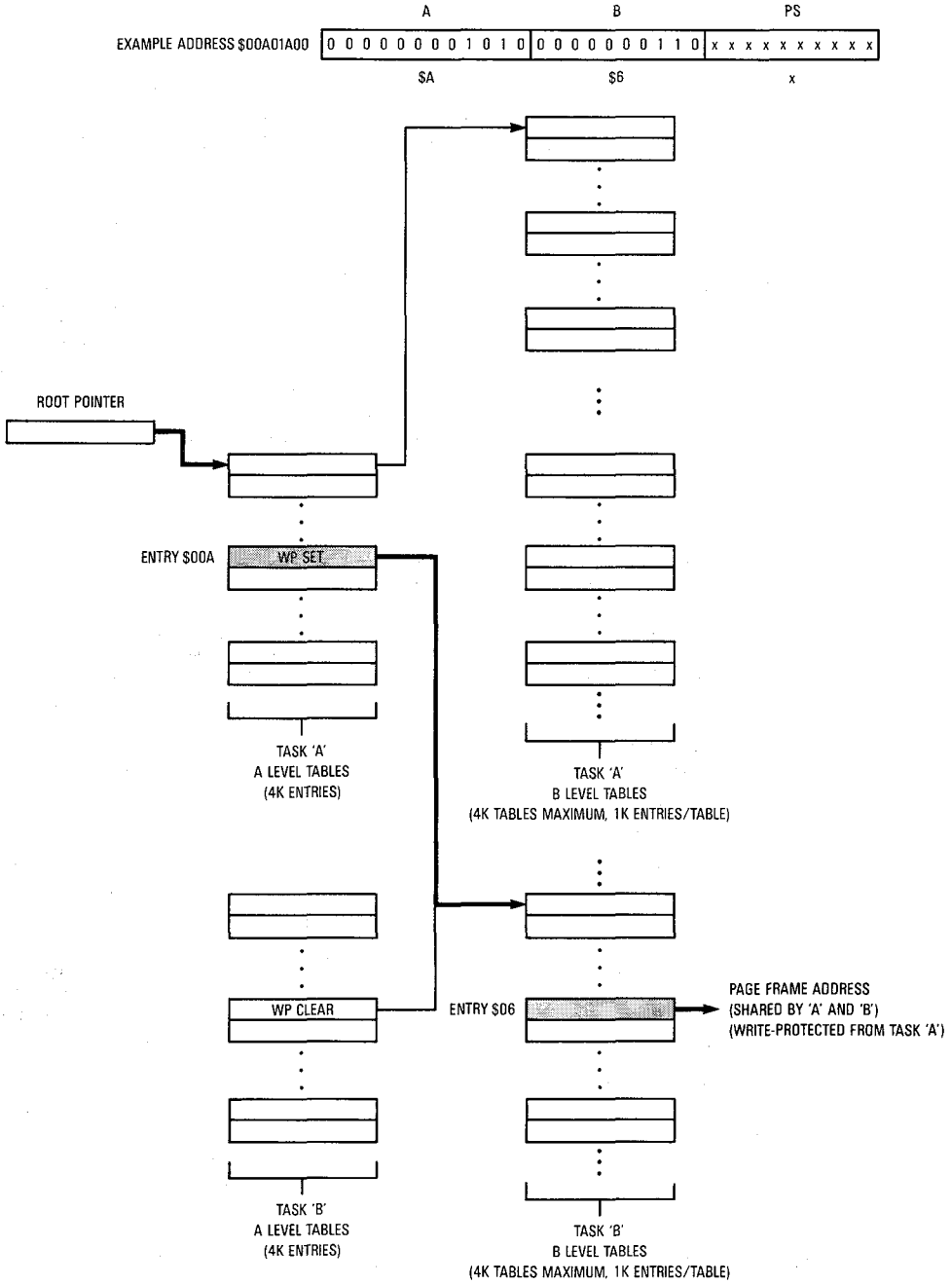
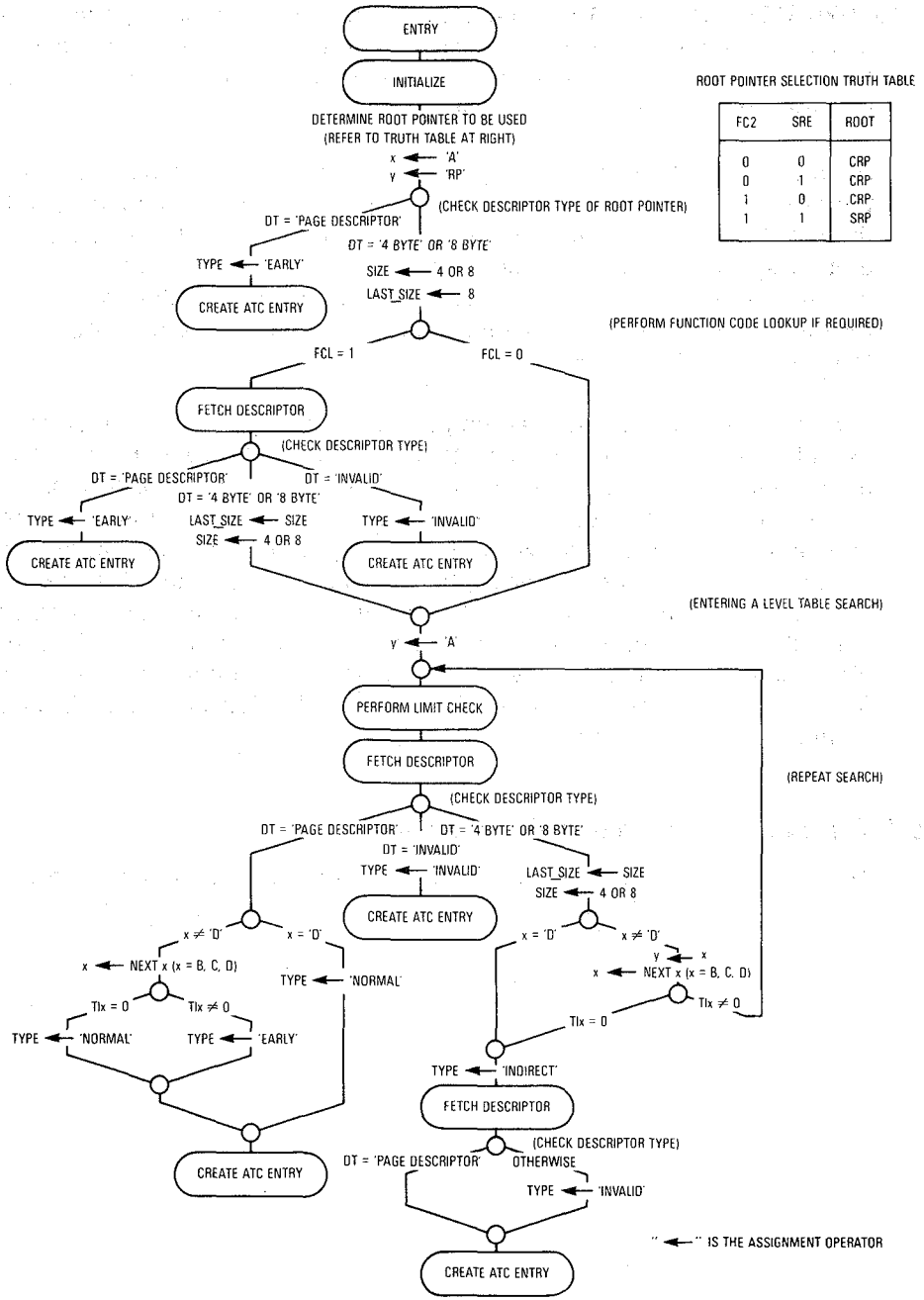


Figure 9-23. Example Translation Tree Using Shared Tables



ROOT POINTER SELECTION TRUTH TABLE

FC2	SRE	ROOT
0	0	CRP
0	1	CRP
1	0	CRP
1	1	SRP

Figure 9-25. Detailed Flowchart of MMU Table Search Operation

The protection mechanisms can be used individually or in any combination to protect:

- Supervisor program and data spaces from access by user programs.
- User program and data spaces from access by other user programs or supervisor programs (except with the MOVES instruction).
- Supervisor and user program spaces from write accesses (except by the supervisor using the MOVES instruction).
- One or more pages of memory from write accesses.

9.5.5.1 FUNCTION CODE LOOKUP. One way of protecting supervisor and user spaces from unauthorized access is to set the FCL bit in the TC register. This effectively segments the logical address space into a supervisor program space, a supervisor data space, a user program space, and a user data space, as shown in Figure 9-30. Each task has an address translation tree with unique mappings for the logical addresses in its user spaces. The translation tables for mapping the supervisor spaces can be copied into each task's translation tree. Figure 9-31 shows a translation tree using function code lookup, and Figure 9-32 shows translation trees for two tasks that share common supervisor spaces.

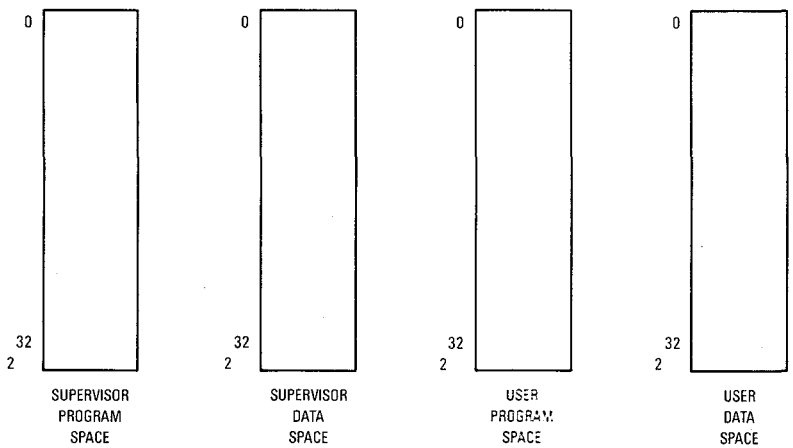


Figure 9-30. Logical Address Map Using Function Code Lookup

9.7.5 Register Programming Considerations

If the entries in the address translation cache (ATC) are no longer valid when a reset operation occurs, an explicit flush operation must be specified by the software. The assertion of $\overline{\text{RESET}}$ disables translations by clearing the E bits of the TC and TTx registers, but it does not flush the ATC. Flushing of the ATC is optional under control of the FD bit of the PMOVE instruction that loads a new value into the SRP, CRP, TT0, TT1, or TC register.

The programmer of the MMU must be aware of effects resulting from loading certain registers. A subsequent section describes these effects. The MMUSR values lend themselves to the use of a case structure for branching to appropriate routines in a bus error handler. An example of a flowchart that implements this technique is shown in another section. A third section describes the conditions that result in MMU exceptions.

9.7.5.1 REGISTER SIDE EFFECTS. The PMOVE instruction is used to load or read any of the MMU registers (CRP, SRP, TC, MMUSR, TT0, and TT1). Since loading the root pointers, the translation control register, or the transparent translation registers with new values can cause some or all of the address translations to change, it may be desired to flush the ATC of its contents any time these registers are written. The opcodes of the PMOVE instructions that write to CRP, SRP, TC, TT0, and TT1 contain a flush disable (FD) bit that optionally flushes the ATC when these instructions are executed. If the FD bit equals one, the ATC is not flushed when the instruction is executed. If the FD bit equals zero, the ATC is flushed during the execution of the PMOVE instruction.

9.7.5.2 MMU STATUS REGISTER DECODING. The seven status bits in the MMU status register (MMUSR) indicate conditions to which the operating system should respond. In a typical bus error handler routine, the flows shown in Figures 9-39 and 9-40 can be used to determine the cause of an MMU fault. The PTEST instructions set the bits in the MMUSR appropriately, and the program can branch to the appropriate code segment for the condition. Figure 9-39 shows the flow for a PTEST instruction for the ATC (level 0), and Figure 9-40 shows the flow for a PTEST instruction that accesses an address translation tree (levels 1–7).

10.4.15 Transfer Multiple Main Processor Registers Primitive

The transfer multiple main processor registers primitive transfers long-word operands between one or more of its data or address registers and the coprocessor. This primitive applies to general and conditional category instructions. Figure 10-35 shows the format of the transfer multiple main processor registers primitive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CA	PC	DR	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 10-35. Transfer Multiple Main Processor Registers Primitive Format

This primitive uses the CA, PC, and DR bits as previously described. If the coprocessor issues this primitive with CA=0 during a conditional category instruction, the main processor initiates protocol violation exception processing.

When the main processor receives this primitive, it reads a 16-bit register select mask from the register select CIR. The format of the register select mask is shown in Figure 10-36. A register is transferred if the bit corresponding to the register in the register select mask is set to one. The selected registers are transferred in the order D0–D7 and then A0–A7.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0

Figure 10-36. Register Select Mask Format

If DR=0, the main processor writes the contents of each register indicated in the register select mask to the operand CIR using a sequence of long-word transfers. If DR=1, the main processor reads a long-word operand from the operand CIR into each register indicated in the register select mask. The registers are transferred in the same order, regardless of the direction of transfer indicated by the DR bit.

10.4.16 Transfer Multiple Coprocessor Registers Primitive

The transfer multiple coprocessor registers primitive transfers from 0–16 operands between the effective address specified in the coprocessor instruction and the coprocessor. This primitive applies to general category instruc-

sequencer activity. The execution of an instruction that only accesses on-chip registers can be overlapped entirely with a concurrent data write generated by a previous instruction, if prefetches generated by that instruction are resident in the instruction cache.

11.2 RESOURCE SCHEDULING

Some of the variability in instruction execution timings results from the overlap of resource utilization. The processor can be viewed as consisting of eight independently scheduled resources. Since very little of the scheduling is directly related to instruction boundaries, it is impossible to make accurate estimates of the time required to execute a particular instruction without knowing the complete context within which the instruction is executing. The position of these resources within the MC68030 is shown in Figure 11-1.

11.2.1 Microsequencer

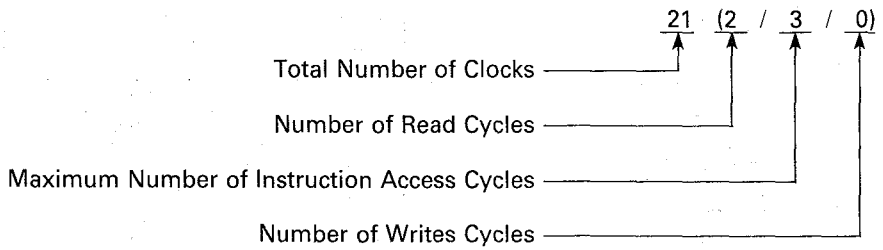
The microsequencer is either executing microinstructions or awaiting completion of accesses that are necessary to continue executing microcode. The bus controller is responsible for all bus activity. The microsequencer controls the bus controller, instruction execution, and internal processor operations such as calculation of effective addresses and setting of condition codes. The microsequencer initiates instruction word prefetches and controls the validation of instruction words in the instruction pipe.

11.2.2 Instruction Pipe

The MC68030 contains a three-word instruction pipe where instruction opcodes are decoded. As shown in Figure 11-1, instruction words (instruction operation words and all extension words) enter the pipe at stage B and proceed to stages C and D. An instruction word is completely decoded when it reaches stage D of the pipe. Each of the pipe stages has a status bit that reflects whether the word in the stage was loaded with data from a bus cycle that was terminated abnormally. Stages of the pipe are only filled in response to specific prefetch requests issued by the microsequencer.

Words are loaded into the instruction pipe from the cache holding register. While the individual stages of the pipe are only 16 bits wide, the cache holding register is 32 bits wide and contains the entire long word. This long word is obtained from the instruction cache or the external bus in response to a prefetch request from the microsequencer. When the microsequencer re-

The instruction-cache-case and average no-cache-case columns of the instruction timing tables contain four sets of numbers, three of which are enclosed in parentheses. The outer number is the total number of clocks for the given cache case and instruction. The first number inside the parentheses is the number of operand read cycles performed by the instruction. The second value inside the parentheses is the maximum number of instruction bus cycles performed by the instruction, including all prefetches to keep the instruction pipe filled. Because the second value is the average of the odd-word-aligned case and the even-word-aligned case (rounded up to an integral number of bus cycles), it is always greater than or equal to the actual number of bus cycles (one bus cycle per two instruction prefetches). The third value within the parentheses is the number of write cycles performed by the instruction. One example from the instruction timing table is:



The total numbers of bus-activity clocks and internal clocks (not overlapped by bus activity) of the instruction in this example are derived as follows:

(2 Reads•2 Clocks/Read) + (3 Instruction Accesses•2 Clocks/Access) +
(0 Writes•2 Clocks/Write) = 10 Clocks of Bus Activity
21 Total Clocks − 10 Bus Activity Clocks = 11 Internal Clocks

The example used here is taken from a no-cache-case 'fetch effective address' time. The addressing mode is ([d_{32},B],I,d_{32}). The same addressing mode under the instruction-cache-case execution time entry is 18(2/0/0). For the instruction-cache-case execution time, no instruction accesses are required because the cache is enabled and the sequencer does not have to access external memory for the instruction words.

The first five timing tables deal exclusively with fetching and calculating effective addresses and immediate operands. The remaining tables are instruction and operation timings. Some instructions use addressing modes that are not included in the corresponding instruction timings. These cases refer to footnotes that indicate the additional table needed for the timing calculation. All read and write accesses are assumed to take two clock periods.

11.6.13 Bit Manipulation Instructions

The bit manipulation instruction table indicates the number of clock periods needed for the processor to perform the specified bit operation on the given addressing mode. Footnotes indicate when it is necessary to account for the appropriate effective address time. For instruction-cache case and for no-cache case, the total number of clock cycles is outside the parentheses. The number of read, prefetch, and write cycles is given inside the parentheses as (r/p/w). The read, prefetch, and write cycles are included in the total clock cycle number.

All timing data assumes two-clock reads and writes.

	Instruction		Head	Tail	I-Cache Case	No-Cache Case
	BTST	#⟨data⟩,Dn	4	0	4(0/0/0)	4(0/1/0)
	BTST	Dn,Dn	4	0	4(0/0/0)	4(0/1/0)
#	BTST	#⟨data⟩,Mem	0	0	4(0/0/0)	4(0/1/0)
*	BTST	Dn,Mem	0	0	4(0/0/0)	4(0/1/0)
	BCHG	#⟨data⟩,Dn	6	0	6(0/0/0)	6(0/1/0)
	BCHG	Dn,Dn	6	0	6(0/0/0)	6(0/1/0)
#	BCHG	#⟨data⟩,Mem	0	0	6(0/0/1)	6(0/1/1)
*	BCHG	Dn,Mem	0	0	6(0/0/1)	6(0/1/1)
	BCLR	#⟨data⟩,Dn	6	0	6(0/0/0)	6(0/1/0)
	BCLR	Dn,Dn	6	0	6(0/0/0)	6(0/1/0)
#	BCLR	#⟨data⟩,Mem	0	0	6(0/0/1)	6(0/1/1)
*	BCLR	Dn,Mem	0	0	6(0/0/1)	6(0/1/1)
	BSET	#⟨data⟩,Dn	6	0	6(0/0/0)	6(0/1/0)
	BSET	Dn,Dn	6	0	6(0/0/0)	6(0/1/0)
#	BSET	#⟨data⟩,Mem	0	0	6(0/0/1)	6(0/1/1)
*	BSET	Dn,Mem	0	0	6(0/0/1)	6(0/1/1)

*Add Fetch Effective Address Time
#Add Fetch Immediate Effective Address Time

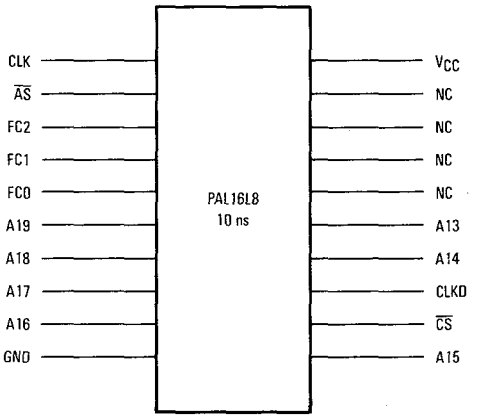


Figure 12-3. Chip-Select Generation PAL

```
PAL16I8
FPCP CS GENERATION CIRCUITRY FOR 25 MHz OPERATION
MOTOROLA INC., AUSTIN, TEXAS
CLK     AS      FC2     FC1     FC0     A19     A18     A17     A16     GND
A15     /CS     /CLKD   A14     A13     NC      NC      NC      NC      VCC

CS   = FC2     * FC1     * FC0                 ;cpu space = $7
       * /A19   * /A18    * A17     * /A16     ;coprocessor access = $2
       * /A15   * /A14    * A13                ;coprocessor id = $1
       * /CLK                                  ;qualified by MPU clock low

       + FC2     * FC1     * FC0               ;cpu space = $7
       * /A19   * /A18    * A17     * /A16     ;coprocessor access = $2
       * /A15   * /A14    * A13                ;coprocessor id = $1
       * /AS                                   ;qualified by address strobe low

       + FC2     * FC1     * FC0
       * /A19   * /A18    * A17     * /A16     ;coprocessor access = $2
       * /A15   * /A14    * A13                ;coprocessor id = $1
       * /CLKD                                 ;qualified by CLKD (delayed CLK)

CLKD = CLK
```

Description: There are three terms to the CS generation. The first term denotes the earliest time CS can be asserted. The second term is used to assert CS until the end of the FPCP access. The third term is to ensure that no race condition occurs in case of a late AS.

Figure 12-4. PAL Equations

During read operations, M68000 processors latch data on the last falling clock edge of the bus cycle, one-half clock before the bus cycle ends (burst mode is a special case). Latching data here, instead of the next rising clock edge, helps to avoid data bus contention with the next bus cycle and allows the MC68030 to receive the data into its execution unit sooner for a net performance increase.

Write operations also use this data bus timing to allow data hold times from the negating strobes and to avoid any bus contention with the following bus cycle. This usually allows the system to be designed with a minimum of bus buffers and latches.

One of the benefits of the MC68030's on-chip caches is that the effect of external wait states on performance is lessened because the caches are always accessed in fewer than "no wait states", regardless of the external memory configuration. This feature makes the MC68030 (and MC68020) unique among other general-purpose microprocessors.

12.4.1 Access Time Calculations

The timing paths that are typically critical in any memory interface are illustrated and defined in Figure 12-8. For burst transfers, the first long word transferred also uses these parameters, but the subsequent transfers are different and are discussed in **12.4.2 Burst Mode Cycles**.

The type of device that is interfaced to the MC68030 determines exactly which of the paths is most critical. The address-to-data paths are typically the critical paths for static devices since there is no penalty for initiating a cycle to these devices and later validating that access with the appropriate bus control signal. Conversely, the address-strobe-to-data-valid path is often most critical for dynamic devices since the cycle must be validated before an access can be initiated. For devices that signal termination of a bus cycle before data is validated (e.g., error detection and correction hardware and some external caches) to improve performance, the critical path may be from the address or strobes to the assertion of $\overline{\text{BERR}}$ (or $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$). Finally, the address-valid-to-$\overline{\text{DSACKx}}$-or-$\overline{\text{STERM}}$-asserted path is most critical for very fast devices and external caches, since the time available between when the address is valid and when $\overline{\text{DSACKx}}$ or $\overline{\text{STERM}}$ must be asserted to terminate the bus cycle is minimal. Table 12-2 provides the equations required to calculate the various memory access times assuming a 50-percent duty cycle clock.

— N —

— O —

— P —

— T —