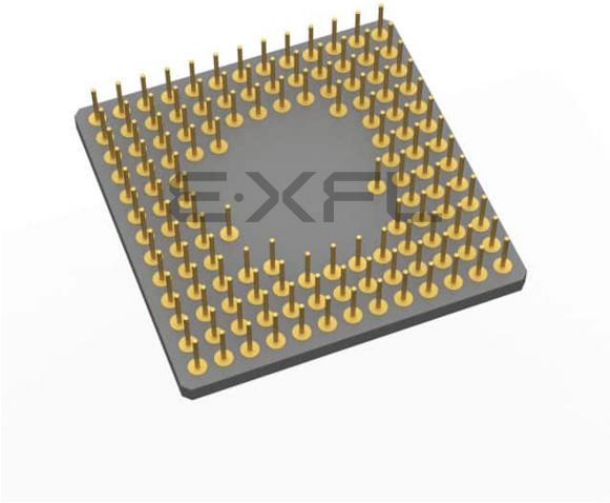


Welcome to E-XFL.COM

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	20MHz
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/product-detail/nxp-semiconductors/mc68030rc20c

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

1.7 THE MEMORY MANAGEMENT UNIT

The MMU supports virtual memory systems by translating logical addresses to physical addresses using translation tables stored in memory. The MMU stores address mappings in an address translation cache (ATC) that contains the most recently used translations. When the ATC contains the address for a bus cycle requested by the CPU, a translation table search is not performed. Features of the MMU include:

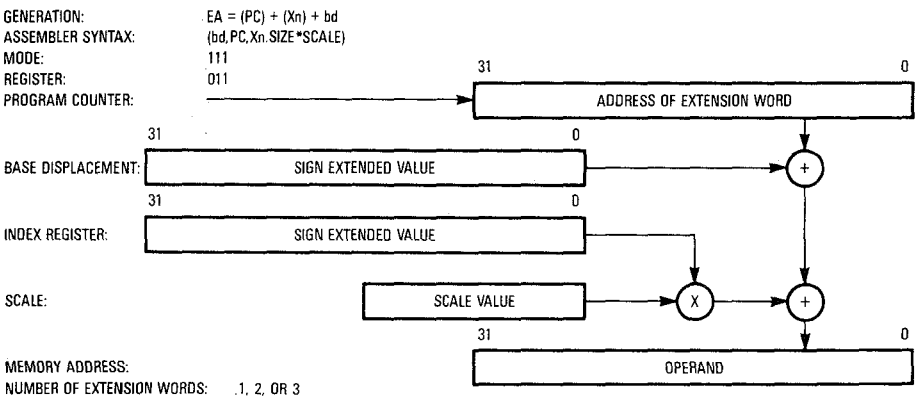
- Multiple Level Translation Tables with Short- and Long-Format Descriptors for Efficient Table Space Usage
- Table Searches Automatically Performed in Microcode
- 22-Entry Fully Associative ATC
- Address Translations and Internal Instruction and Data Cache Accesses Performed in Parallel
- Eight Page Sizes Available Ranging from 256 to 32K Bytes
- Two Optional Transparent Blocks
- User and Supervisor Root Pointer Registers
- Write Protection and Supervisor Protection Attributes
- Translations Enabled/Disabled by Software
- Translations Can Be Disabled with External $\overline{\text{MMUDIS}}$ Signal
- Used and Modified Bits Automatically Maintained in Tables and ATC
- Cache Inhibit Output ($\overline{\text{CIOUT}}$) Signal Can Be Asserted on a Page-by-Page Basis
- 32-Bit Internal Logical Address with Capability To Ignore as many as 15 Upper Address Bits
- 3-Bit Function Code Supports Separate Address Spaces
- 32-Bit Physical Address

The memory management function performed by the MMU is called demand paged memory management. Since a task specifies the areas of memory it requires as it executes, memory allocation is supported on a demand basis. If a requested access to memory is not currently mapped by the system, then the access causes a demand for the operating system to load or allocate the required memory image. The technique used by the MC68030 is paged memory management because physical memory is managed in blocks of a specified number of bytes, called page frames. The logical address space is divided

2.4.13 Program Counter Indirect with Index (Base Displacement) Mode

This mode is similar to the address register indirect with index (base displacement) mode described in **2.4.8 Address Register Indirect with Index (Base Displacement) Mode**, but the PC is used as the base register. It requires an index register indicator and an optional 16- or 32-bit sign-extended base displacement. The operand is in memory. The address of the operand is the sum of the contents of the PC, the scaled contents of the sign-extended index register, and the base displacement. The value of the PC is the address of the first extension word. The reference is a program space reference and is only allowed for reads (refer to **4.2 ADDRESS SPACE TYPES**).

In this mode, the PC, the index register, and the displacement are all optional. However, the user must supply the assembler notation "ZPC" (zero value is taken for the PC) to indicate that the PC is not used. This allows the user to access the program space without using the PC in calculating the effective address. The user can access the program space with a data register indirect access by placing ZPC in the instruction and specifying a data register (Dn) as the index register.



3.3.1 Condition Code Computation

Most operations take a source operand and a destination operand, compute, and store the result in the destination location. Single-operand operations take a destination operand, compute, and store the result in the destination location. Table 3-12 lists each instruction and how it affects the condition code bits.

Table 3-12. Condition Code Computations (Sheet 1 of 2)

Operations	X	N	Z	V	C	Special Definition
ABCD	*	U	?	U	?	C = Decimal Carry Z = Z ∧ $\overline{Rm}$ ∧ . . . ∧ $\overline{R0}$
ADD, ADDI, ADDQ	*	*	*	?	?	V = Sm ∧ Dm ∧ $\overline{Rm}$ ∨ $\overline{Sm}$ ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ Dm ∨ $\overline{Rm}$ ∧ Dm ∨ Sm ∧ $\overline{Rm}$
ADDX	*	*	?	?	?	V = Sm ∧ Dm ∧ $\overline{Rm}$ ∨ $\overline{Sm}$ ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ Dm ∨ $\overline{Rm}$ ∧ Dm ∨ Sm ∧ $\overline{Rm}$ Z = Z ∧ $\overline{Rm}$ ∧ . . . ∧ $\overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, NOT, TAS, TST	—	*	*	0	0	
CHK	—	*	U	U	U	
CHK2, CMP2	—	U	?	U	?	Z = (R = LB) ∨ (R = UB) C = (LB < = UB) ∧ (IR < LB) ∨ (R > UB)) ∨ (UB < LB) ∧ (R > UB) ∧ (R < LB)
SUB, SUBI, SUBQ	*	*	*	?	?	V = $\overline{Sm}$ ∧ Dm ∧ $\overline{Rm}$ ∨ Sm ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ $\overline{Dm}$ ∨ Rm ∧ $\overline{Dm}$ ∨ Sm ∧ Rm
SUBX	*	*	?	?	?	V = $\overline{Sm}$ ∧ Dm ∧ $\overline{Rm}$ ∨ Sm ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ $\overline{Dm}$ ∨ Rm ∧ $\overline{Dm}$ ∨ Sm ∧ Rm Z = Z ∧ $\overline{Rm}$ ∧ . . . ∧ $\overline{R0}$
CAS, CAS2, CMP, CMPI, CMPM	—	*	*	?	?	V = $\overline{Sm}$ ∧ Dm ∧ $\overline{Rm}$ ∨ Sm ∧ $\overline{Dm}$ ∧ Rm C = Sm ∧ $\overline{Dm}$ ∨ Rm ∧ $\overline{Dm}$ ∨ Sm ∧ Rm
DIVS, DUVI	—	*	*	?	0	V = Division Overflow
MULS, MULU	—	*	*	?	0	V = Multiplication Overflow
SBCD, NBCD	*	U	?	U	?	C = Decimal Borrow Z = Z ∧ $\overline{Rm}$ ∧ . . . ∧ $\overline{Ro}$
NEG	*	*	*	?	?	V = Dm ∧ Rm C = Dm ∨ Rm
NEGX	*	*	?	?	?	V = Dm ∧ Rm C = Dm ∨ Rm Z = Z ∧ $\overline{Rm}$ ∧ . . . ∧ $\overline{R0}$

Another widely used application for bit field instructions is bit-mapped graphics. Because byte boundaries are ignored in these areas of memory, the field definitions used with bit field instructions are very helpful.

3.5.4 Pipeline Synchronization with the NOP Instruction

Although the no operation (NOP) instruction performs no visible operation, it serves an important purpose. It forces synchronization of the integer unit pipeline by waiting for all pending bus cycles to complete. All previous integer instructions and floating-point external operand accesses complete execution before the NOP begins. The NOP instruction does not synchronize the FPU pipeline; floating-point instructions with floating-point register operand destinations can be executing when the NOP begins.

Table 5-1. Signal Index (Sheet 2 of 2)

Signal Name	Mnemonic	Function
Data Transfer and Size Acknowledge	$\overline{\text{DSACK0}}$ $\overline{\text{DSACK1}}$	Bus response signals that indicate the requested data transfer operation is completed. In addition, these two lines indicate the size of the external bus port on a cycle-by-cycle basis and are used for asynchronous transfers.
Synchronous Termination	$\overline{\text{STERM}}$	Bus response signal that indicates a port size of 32 bits and that data may be latched on the next falling clock edge.
Cache Inhibit In	$\overline{\text{CIIN}}$	Prevents data from being loaded into the MC68030 instruction and data caches.
Cache Inhibit Out	$\overline{\text{CIOUT}}$	Reflects the CI bit in ATC entries or TTx register; indicates that external caches should ignore these accesses.
Cache Burst Request	$\overline{\text{CBREQ}}$	Indicates a burst request for the instruction or data cache.
Cache Burst Acknowledge	$\overline{\text{CBACK}}$	Indicates that the accessed device can operate in burst mode.
Interrupt Priority Level	$\overline{\text{IPL0}}$–$\overline{\text{IPL2}}$	Provides an encoded interrupt level to the processor.
Interrupt Pending	$\overline{\text{IPEND}}$	Indicates that an interrupt is pending.
Autovector	$\overline{\text{AVEC}}$	Requests an autovector during an interrupt acknowledge cycle.
Bus Request	$\overline{\text{BR}}$	Indicates that an external device requires bus mastership.
Bus Grant	$\overline{\text{BG}}$	Indicates that an external device may assume bus mastership.
Bus Grant Acknowledge	$\overline{\text{BGACK}}$	Indicates that an external device has assumed bus mastership.
Reset	$\overline{\text{RESET}}$	System reset.
Halt	$\overline{\text{HALT}}$	Indicates that the processor should suspend bus activity.
Bus Error	$\overline{\text{BERR}}$	Indicates that an erroneous bus operation is being attempted.
Cache Disable	$\overline{\text{CDIS}}$	Dynamically disables the on-chip cache to assist emulator support.
MMU Disable	$\overline{\text{MMUDIS}}$	Dynamically disables the translation mechanism of the MMU.
Pipe Refill	$\overline{\text{REFILL}}$	Indicates when the MC68030 is beginning to fill pipeline.
Microsequencer Status	$\overline{\text{STATUS}}$	Indicates the state of the microsequencer.
Clock	CLK	Clock input to the processor.
Power Supply	V_{CC}	Power supply.
Ground	GND	Ground connection.

5.11 EMULATOR SUPPORT SIGNALS

The following signals support emulation by providing a means for an emulator to disable the on-chip caches and memory management unit and by supplying internal status information to an emulator. Refer to **SECTION 12 APPLICATIONS INFORMATION** for more detailed information on emulation support.

5.11.1 Cache Disable ($\overline{\text{CDIS}}$)

The cache disable signal dynamically disables the on-chip caches to assist emulator support. Refer to **6.1 ON-CHIP CACHE ORGANIZATION AND OPERATION** for information about the caches; refer to **SECTION 12 APPLICATIONS INFORMATION** for a description of the use of this signal by an emulator. $\overline{\text{CDIS}}$ does not flush the data and instruction caches; entries remain unaltered and become available again when $\overline{\text{CDIS}}$ is negated.

5.11.2 MMU Disable ($\overline{\text{MMUDIS}}$)

The MMU disable signal dynamically disables the translation of addresses by the MMU. Refer to **9.4 ADDRESS TRANSLATION CACHE** for a description of address translation; refer to **SECTION 12 APPLICATIONS INFORMATION** for a description of the use of this signal by an emulator. The assertion of $\overline{\text{MMUDIS}}$ does not flush the address translation cache (ATC); ATC entries become available again when $\overline{\text{MMUDIS}}$ is negated.

5.11.3 Pipeline Refill ($\overline{\text{REFILL}}$)

The pipeline refill signal indicates that the MC68030 is beginning to refill the internal instruction pipeline. Refer to **SECTION 12 APPLICATIONS INFORMATION** for a description of the use of this signal by an emulator.

5.11.4 Internal Microsequencer Status ($\overline{\text{STATUS}}$)

The microsequencer status signal indicates the state of the internal microsequencer. The varying number of clocks for which this signal is asserted indicates instruction boundaries, pending exceptions, and the halted condition. Refer to **SECTION 12 APPLICATIONS INFORMATION** for a description of the use of this signal by an emulator.

8.1.1 Reset Exception

Assertion by external hardware of the $\overline{\text{RESET}}$ signal causes a reset exception. For details on the requirements for the assertion of $\overline{\text{RESET}}$, refer to **7.8 RESET OPERATION**.

The reset exception has the highest priority of any exception; it provides for system initialization and recovery from catastrophic failure. When reset is recognized, it aborts any processing in progress, and that processing cannot be recovered. Figure 8-1 is a flowchart of the reset exception, which performs the following operations:

1. Clears both trace bits in the status register to disable tracing.
2. Places the processor in the interrupt mode of the supervisor privilege level by setting the supervisor bit and clearing the master bit in the status register.
3. Sets the processor interrupt priority mask to the highest priority level (level 7).
4. Initializes the vector base register to zero ($00000000).
5. Clears the enable, freeze, and burst enable bits for both on-chip caches and the write-allocate bit for the data cache in the cache control register.
6. Invalidates all entries in the instruction and data caches.
7. Clears the enable bit in the translation control register and the enable bits in both transparent translation registers of the MMU.
8. Generates a vector number to reference the reset exception vector (two long words) at offset zero in the supervisor program address space.
9. Loads the first long word of the reset exception vector into the interrupt stack pointer.
10. Loads the second long word of the reset exception vector into the program counter.

After the initial instruction prefetches, program execution begins at the address in the program counter. The reset exception does not flush the address translation cache (ATC), nor does it save the value of either the program counter or the status register.

Table 8-3. Tracing Control

T1	T0	Tracing Function
0	0	No Tracing
0	1	Trace on Change of Flow (BRA, JMP, etc.)
1	0	Trace on Instruction Execution (Any Instruction)
1	1	Undefined, Reserved

In general terms, a trace exception is an extension to the function of any traced instruction — that is, the execution of a traced instruction is not complete until the trace exception processing is completed. If an instruction does not complete due to a bus error or address error exception, trace exception processing is deferred until after the execution of the suspended instruction is resumed and the instruction execution completes normally. If an interrupt is pending at the completion of an instruction, the trace exception processing occurs before the interrupt exception processing starts. If an instruction forces an exception as part of its normal execution, the forced exception processing occurs before the trace exception is processed. See **8.1.12 Multiple Exceptions** for a more complete discussion of exception priorities.

When the processor is in the trace mode and attempts to execute an illegal or unimplemented instruction, that instruction does not cause a trace exception since it is not executed. This is of particular importance to an instruction emulation routine that performs the instruction function, adjusts the stacked program counter to skip the unimplemented instruction, and returns. Before returning, the trace bits of the status register on the stack should be checked. If tracing is enabled, the trace exception processing should also be emulated for the trace exception handler to account for the emulated instruction.

The exception processing for a trace starts at the end of normal processing for the traced instruction and before the start of the next instruction. The processor makes an internal copy of the status register and enters the supervisor privilege level. It also clears the T0 and T1 bits of the status register, disabling further tracing. The processor supplies vector number 9 for the trace exception and saves the trace exception vector offset, program counter value, and the copy of the status register on the supervisor stack. The saved value of the program counter is the logical address of the next instruction to be executed. Instruction execution resumes after the required prefetches from the address in the trace exception vector.

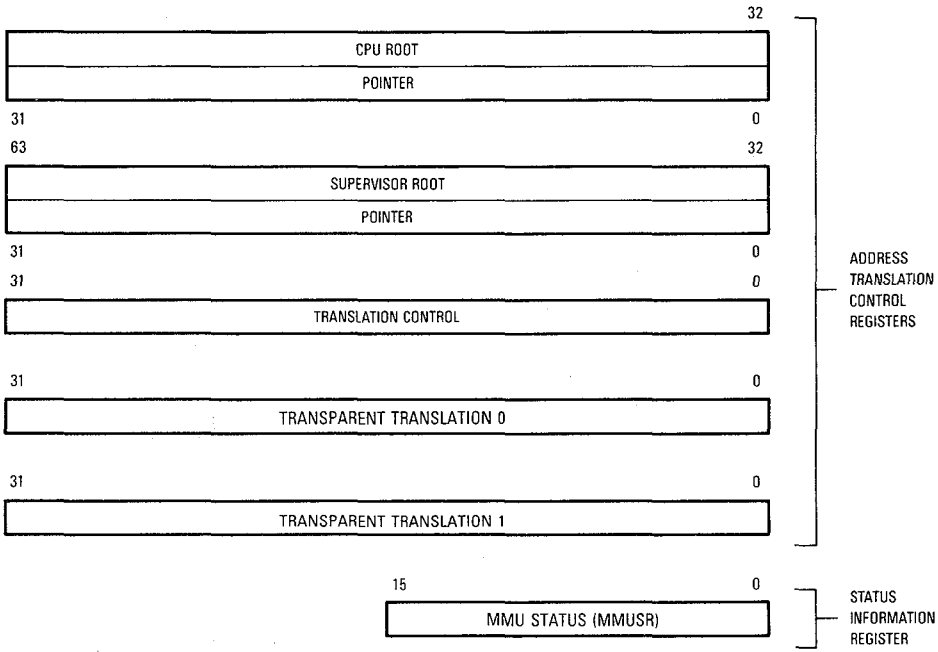


Figure 9-2. MMU Programming Model

9

The ATC in the MMU is a fully associative cache that stores 22 logical-to-physical address translations and associated page information. It compares the logical address and function code internally supplied by the processor with all tag entries in the ATC. When the access address and function code matches a tag in the ATC (a hit occurs) and no access violation is detected, the ATC outputs the corresponding physical address to the bus controller, which continues the external bus cycle. Function codes are routed to the bus controller unmodified.

Each ATC entry contains a logical address, a physical address, and status bits. Among the status bits are the write protect and cache inhibit bits.

When the ATC does not contain the translation for a logical address (a miss occurs) and an external bus cycle is required, the MMU aborts the access and causes the processor to initiate bus cycles that search the translation tables in memory for the correct translation. If the table search completes without any errors, the MMU stores the translation in the ATC and provides the physical address for the access, allowing the bus controller to retry the original bus cycle.

The page size (PS) field of the TC register specifies the page size for the system. The number of pages in the system is equal to the logical address space divided by the page size. The maximum number of pages that can be defined by a translation tree is greater than 16 million ($2^{32}/2^{8}$). The minimum number is 4 ($2^{17}/2^{15}$). The function code can also be used in the table lookup, defining as many as seven regions of the above size (FC = 0–6). The entire range of the logical address space(s) can be defined by translation tables of many sizes. The MC68030 provides flexibility that simplifies the implementation of large translation tables.

The use of a tree structure with as many as five levels of tables provides granularity in translation table design. The LIMIT field of the root pointer can limit the value of the first index and limits the actual number of descriptors required. Optionally, the top level of the structure can be indexed by function code bits. In this case, the pointer table at this level contains eight descriptors. The next level of the structure (or the top level when the FCL bit of the TC register is set to zero) is indexed by the most significant bits of the logical address (disregarding the number of bits specified by the IS field). The number of logical address bits used for this index is specified by the TIA field of the TC register. If, for example, the TIA field contains the value 5, the index for this level contains five bits, and the pointer table at this level contains at most 32 descriptors.

Similarly, the TIB, TIC, and TID fields of the TC register define the indexes for lower levels of the translation table tree. When one of these fields contains *zero, the remaining TIx fields are ignored; the last nonzero TIx field defines* the index into the lowest level of the tree structure. The tables selected by the index at this level are page tables; every descriptor in these tables is (or represents) a page descriptor. Figure 9-6 shows how the TIx fields of the TC register apply to a function code and logical address.

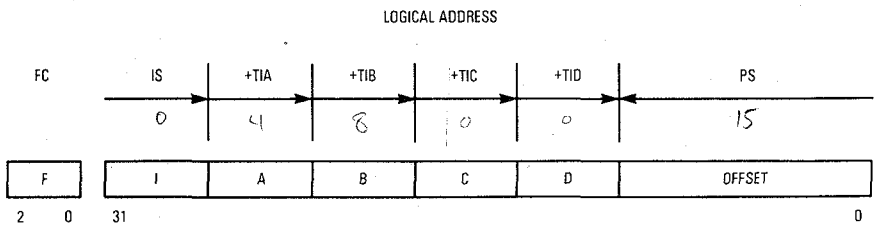


Figure 9-6. Derivation of Table Index Fields

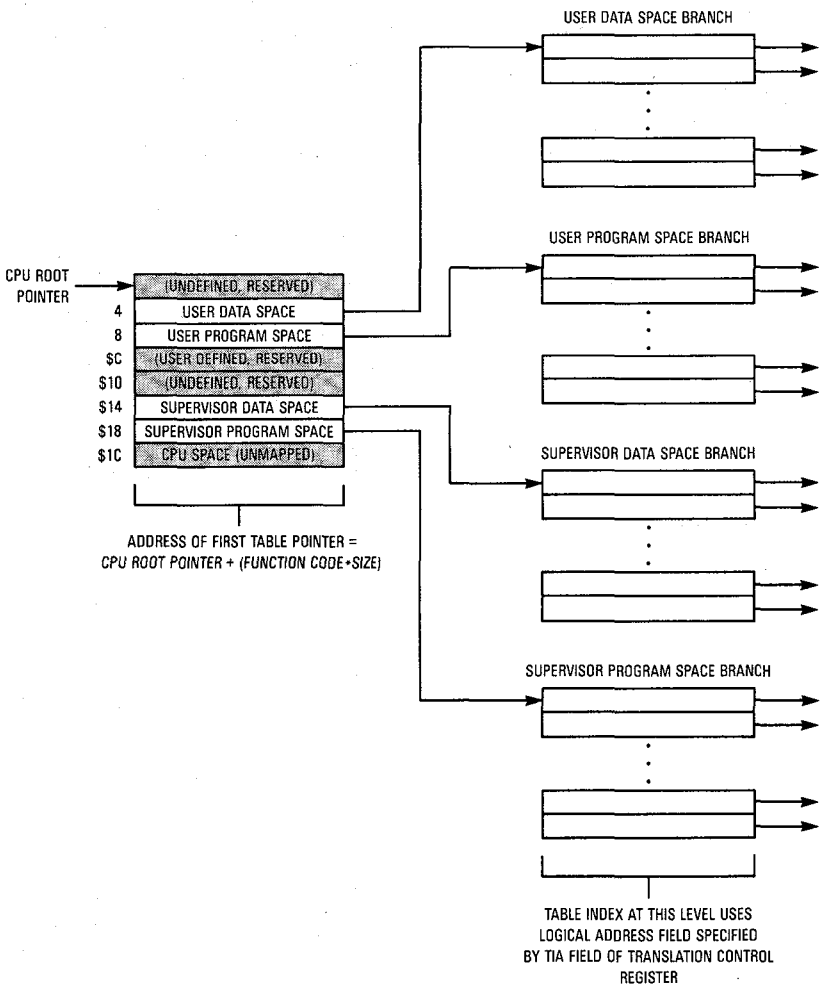


Figure 9-31. Example Translation Tree Using Function Code Lookup

9.6 MC68030 AND MC68851 MMU DIFFERENCES

The MC68851 paged memory management unit provides memory management for the MC68020 as a coprocessor. The on-chip MMU of the MC68030 provides many of the features of the MC68020/MC68851 combination. The following functions of the MC68851 are not available in the MC68030 MMU:

- Access Levels
- Breakpoint Registers
- Root Pointer Table
- Aliases for Tasks
- Lockable Entries in the ATC
- ATC Entries Defined as Shared Globally

In addition, the following features of the MC68030 MMU differ from the MC68020/MC68851 pair:

- 22-Entry ATC
- Reduced Instruction Set
- Only Control-Alterable Addressing Modes Supported for MMU Instructions

In general, the MC68030 is program compatible with the MC68020/MC68851 combination. However, in a program for the MC68030, the following instructions must be avoided or emulated in the exception routine for F-line unimplemented instructions: PVALID, PFLUSHR, PFLUSHS, PBcc, PDBcc, PScc, PTRAPcc, PSAVE, PRESTORE, and PMOVE for unsupported registers (CAL, VAL, SCC, BAD, BACx, DRP, and AC). Additionally, the effective addressing modes supported on the MC68851 that are not emulated by the MC68030 must be simulated or avoided.

10.4.5 Supervisor Check Primitive

The supervisor check primitive verifies that the main processor is operating in the supervisor state while executing a coprocessor instruction. This primitive applies to instructions in the general and conditional coprocessor instruction categories. Figure 10-25 shows the format of the supervisor check primitive.

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

Figure 10-25. Supervisor Check Primitive Format

This primitive uses the PC bit as previously described. Bit [15] is shown as one, but during execution of a general category instruction, this primitive performs the same operations regardless of the value of bit [15]. If this primitive is issued with bit [15]=0 during a conditional category instruction, however, the main processor initiates protocol violation exception processing.

When the main processor reads the supervisor check primitive from the response CIR, it checks the value of the S bit in the status register. If S=0 (main processor operating at user privilege level), the main processor aborts the coprocessor instruction by writing an abort mask (refer to **10.3.2 Control CIR**) to the control CIR. The main processor then initiates privilege violation exception processing (refer to **10.5.2.3 PRIVILEGE VIOLATIONS**). If the main processor is at the supervisor privilege level when it receives this primitive, it reads the response CIR again.

The supervisor check primitive allows privileged instructions to be defined in the coprocessor general and conditional instruction categories. This primitive should be the first one issued by the coprocessor during the dialog for an instruction that is implemented as privileged.

10.4.6 Transfer Operation Word Primitive

The transfer operation word primitive requests a copy of the coprocessor instruction operation word for the coprocessor. This primitive applies to general and conditional category instructions. Figure 10-26 shows the format of the transfer operation word primitive.

10.4.20 Take Post-Instruction Exception Primitive

The take post-instruction exception primitive initiates exception processing using a coprocessor-supplied exception vector number and the post-instruction exception stack frame format. This primitive applies to general and conditional category instructions. Figure 10-44 shows the format of the take post-instruction exception primitive.

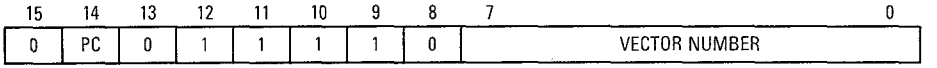


Figure 10-44. Take Post-Instruction Exception Primitive Format

This primitive uses the PC bit as previously described. Bits [0–7] contain the exception vector number used by the main processor to initiate exception processing.

When the main processor receives this primitive, it acknowledges the coprocessor exception request by writing an exception acknowledge mask (refer to **10.3.2 Control CIR**) to the control CIR. The MC68030 then performs exception processing as described in **8.1 EXCEPTION PROCESSING SEQUENCE**. The vector number for the exception is taken from bits [0-7] of the primitive, and the MC68030 uses the six-word stack frame format shown in Figure 10-45.

The value in the main processor scanPC at the time this primitive is received is saved in the scanPC field of the post-instruction exception stack frame. The value of the program counter saved is the F-line operation word address of the coprocessor instruction during which the primitive is received.

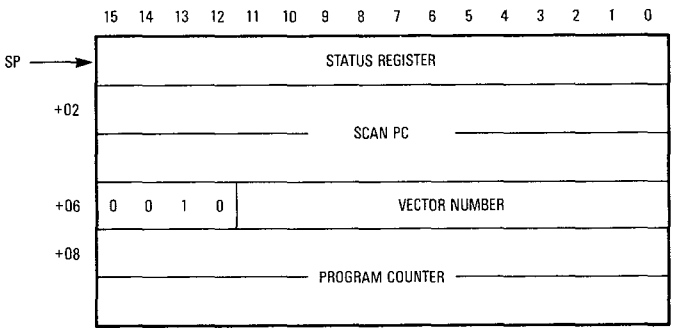


Figure 10-45. MC68030 Post-Instruction Stack Frame

The MC68000 coprocessor response primitive formats are shown in this section. Any response primitive with bits [13:8] = $00 or $3F causes a protocol violation exception. Response primitives with bits [13:8] = $0B, $18–$1B, $1F, $28–$2B, and $38–3B currently cause protocol violation exceptions; they are undefined and reserved for future use by Motorola.

BUSY

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

TRANSFER MULTIPLE COPROCESSOR REGISTERS

15	14	13	12	11	10	9	8	7	0
CA	PC	DR	0	0	0	0	1	LENGTH	

TRANSFER STATUS REGISTER AND SCANPC

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

SUPERVISOR CHECK

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

TAKE ADDRESS AND TRANSFER DATA

15	14	13	12	11	10	9	8	7	0
CA	PC	DR	0	0	1	0	1	LENGTH	

TRANSFER MULTIPLE MAIN PROCESSOR REGISTERS

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

TRANSFER OPERATION WORD

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

11.3.2 Overlap and Best Case

Overlap is the time, measured in clock periods, that an instruction executes concurrently with the previous instruction. In Figure 11-2, a portion of instructions A and B execute simultaneously. The overlap time decreases the overall execution time for the two instructions. Similarly, an overlap period between instructions B and C reduces the overall execution time of these two instructions.

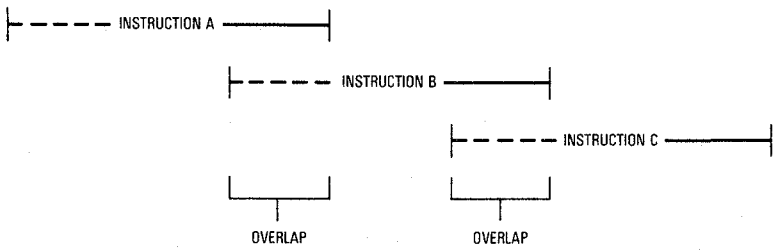


Figure 11-2. Simultaneous Instruction Execution

Each instruction contributes to the total overlap time. As shown in Figure 11-2, a portion of time at the beginning of the execution of instruction B can overlap the end of the execution time of instruction A. This time period is called the head of instruction B. The portion of time at the end of instruction A that can overlap the beginning of instruction B is called the tail of instruction A. The total overlap time between instructions A and B consists of the lesser of the tail of instruction A or the head of instruction B. Refer to the instruction timing tables in **11.6 INSTRUCTION TIMING TABLES** for head and tail times.

Figure 11-3 shows the timing relationship of the factors that comprise the instruction-cache case time for either an effective address calculation (CCea) or for an operation (CCop). In Figure 11-12, the best case execution time for instruction B occurs when the instruction-cache-case times for instruction B and instruction A overlap so that the head of instruction B is completely overlapped with the tail of instruction A.

11.6.2 Fetch Immediate Effective Address (fiea)

The fetch immediate effective address table indicates the number of clock periods needed for the processor to fetch the immediate source operand and to calculate and fetch the specified destination operand. In the case of two-word instructions, this table indicates the number of clock periods needed for the processor to fetch the second word of the instruction and to calculate and fetch the specified source operand or single operand. The effective addresses are divided by their formats (refer to **2.5 Effective Address Encoding Summary**). 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.

Address Mode	Head	Tail	I-Cache Case	No-Cache Case
SINGLE EFFECTIVE ADDRESS INSTRUCTION FORMAT				
% #⟨data⟩.W,Dn	2+op head	0	**2**(0/0/0)	**2**(0/1/0)
% #⟨data⟩.L,Dn	4+op head	0	**4**(0/0/0)	**4**(0/1/0)
#⟨data⟩.W,(An)	1	1	**3**(1/0/0)	**4**(1/1/0)
#⟨data⟩.L,(An)	1	0	**4**(1/0/0)	**5**(1/1/0)
#⟨data⟩.W,(An)+	2	1	**5**(1/0/0)	**5**(1/1/0)
#⟨data⟩.L,(An)+	4	1	**7**(1/0/0)	**7**(1/1/0)
#⟨data⟩.W,−(An)	2	2	**4**(1/0/0)	**4**(1/1/0)
#⟨data⟩.L,−(An)	2	0	**4**(1/0/0)	**5**(1/1/0)
#⟨data⟩.W,(d_{16},An)	2	0	**4**(1/0/0)	**5**(1/1/0)
#⟨data⟩.L,(d_{16},An)	4	0	**6**(1/0/0)	**8**(1/2/0)
#⟨data⟩.W,$XXX.W	4	2	**6**(1/0/0)	**6**(1/1/0)
#⟨data⟩.L,$XXX.W	6	2	**8**(1/0/0)	**8**(1/2/0)
#⟨data⟩.W,$XXX.L	3	0	**6**(1/0/0)	**7**(1/2/0)
#⟨data⟩.L,$XXX.L	5	0	**8**(1/0/0)	**9**(1/2/0)
#⟨data⟩.W,#⟨data⟩.L	6+op head	0	**6**(0/0/0)	**6**(0/2/0)
BRIEF FORMAT EXTENSION WORD				
#⟨data⟩.W,(d_8,An,Xn) or (d_8,PC,Xn)	6	2	**8**(1/0/0)	**8**(1/2/0)
#⟨data⟩.L,(d_8,An,Xn) or (d_8,PC,Xn)	8	2	**10**(1/0/0)	**10**(1/2/0)

— D —

— E —