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	Active
Core Processor	68040
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	179-BEPGA
Supplier Device Package	179-PGA (47.24x47.24)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68ec040rc33a

SECTION 1
INTRODUCTION

The MC68040, MC68040V, MC68LC040, MC68EC040, and MC68EC040V (collectively called M68040) are Motorola's third generation of M68000-compatible, high-performance, 32-bit microprocessors. All five devices are virtual memory microprocessors employing multiple concurrent execution units and a highly integrated architecture that provides very high performance in a monolithic HCMOS device. They integrate an MC68030-compatible integer unit (IU) and two independent caches. The MC68040, MC68040V, and MC68LC040 contain dual, independent, demand-paged memory management units (MMUs) for instruction and data stream accesses and independent, 4-Kbyte instruction and data caches. The MC68040 contains an MC68881/MC68882-compatible floating-point unit (FPU). The use of multiple independent execution pipelines, multiple internal buses, and a full internal Harvard architecture, including separate physical caches for both instruction and data accesses, achieves a high degree of instruction execution parallelism on all three processors. The on-chip bus snoop logic, which directly supports cache coherency in multimaster applications, enhances cache functionality.

The M68040 family is user object-code compatible with previous M68000 family members and is specifically optimized to reduce the execution time of compiler-generated code. All five processors implement Motorola's latest HCMOS technology, providing an ideal balance between speed, power, and physical device size.

1.1 DIFFERENCES

Because the functionality of individual M68040 family members are similar, this manual is organized so that the reader will take the following differences into account while reading the rest of this manual. Unless otherwise noted, all references to M68040, with the exception of the differences outlined below, will apply to the MC68040, MC68040V, MC68LC040, MC68EC040, and MC68EC040V. The following paragraphs describe the differences of MC68040V, MC68LC040, MC68EC040, and the MC68EC040V from the MC68040.

1.1.1 MC68040V and MC68LC040

The MC68040V and MC68LC040 are derivatives of the MC68040. They implement the same IU and MMU as the MC68040, but have no FPU. The MC68LC040 is pin compatible with the MC68040. The MC68040V is not pin compatible with the MC68040 and contains some additional features. The following differences exist between the MC68040V, MC68LC040, and MC68040:

1.2 FEATURES

The main features of the M68040 are as follows:

- 6-Stage Pipeline, MC68030-Compatible IU
- MC68881/MC68882-Compatible FPU
- Independent Instruction and Data MMUs
- Simultaneously Accessible, 4-Kbyte Physical Instruction Cache and 4-Kbyte Physical Data Cache
- Low-Latency Bus Accesses for Reduced Cache Miss Penalty
- Multimaster/Multiprocessor Support via Bus Snooping
- Concurrent IU, FPU, MMU, and Bus Controller Operation Maximizes Throughput
- 32-Bit, Nonmultiplexed External Address and Data Buses with Synchronous Interface
- User Object-Code Compatible with All Earlier M68000 Microprocessors
- 4-Gbyte Direct Addressing Range
- Software Support Including Optimizing C Compiler and UNIX® System V Port

The on-chip FPU and large physical instruction and data caches yield improved system performance and increased functionality. The independent instruction and data MMUs and increased internal parallelism also improve performance.

1.3 EXTENSIONS TO THE M68000 FAMILY

The M68040 is compatible with the ANSI/IEEE *Standard 754 for Binary Floating-Point Arithmetic*. The MC68040's FPU has been optimized to execute the most commonly used subset of the MC68881/MC68882 instruction sets and includes additional instruction formats for single- and double-precision rounding results. Software emulates floating-point instructions not directly supported in hardware. Refer to **Appendix E M68040 Floating-Point Emulation (MC68040FPSP)** for details on software emulation. The MOVE16 user instruction is new to the instruction set, supporting efficient 16-byte memory-to-memory data transfers.

1.4 FUNCTIONAL BLOCKS

Figure 1-1 illustrates a simplified block diagram of the MC68040. Refer to **Appendix A MC68LC040** for information on the MC68LC040's and MC68040V's functional blocks; and **Appendix B MC68EC040** for information on the MC68EC040's and MC68EC040V's functional blocks.

The M68040 IU pipeline has been expanded from the MC68030 to include effective address calculation (<ea> calculate) and operand fetch (<ea> fetch) stages with commonly used effective addressing modes. Conditional branches are optimized for the

® UNIX is a registered trademark of AT&T Bell Laboratories.

Table 1-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
BTST	–(bit number of Destination) ø Z;	BTST Dn,<ea> BTST #<data>,<ea>
CAS	CAS Destination – Compare Operand ø cc; if Z, Update Operand ø Destination else Destination ø Compare Operand	CAS Dc,Du,<ea>
CAS2	CAS2 Destination 1 – Compare 1 ø cc; if Z, Destination 2 – Compare ø cc; if Z, Update 1 ø Destination 1; Update 2 ø Destination 2 else Destination 1 ø Compare 1; Destination 2 ø Compare 2	CAS2 Dc1–Dc2,Du1–Du2,(Rn1)–(Rn2)
CHK	If Dn < 0 or Dn > Source then TRAP	CHK <ea>,Dn
CHK2	If Rn < LB or If Rn > UB then TRAP	CHK2 <ea>,Rn
CINV	If supervisor state then invalidate selected cache lines else TRAP	CINVL <caches>, (An) CINVP <caches>, (An) CINVA <caches>
CLR	0 ø Destination	CLR <ea>
CMP	Destination – Source ø cc	CMP <ea>,Dn
CMPA	Destination – Source	CMPA <ea>,An
CMPI	Destination – Immediate Data	CMPI #<data>,<ea>
CMPM	Destination – Source ø cc	CMPM (Ay)+,(Ax)+
CMP2	Compare Rn < LB or Rn > UB and Set Condition Codes	CMP2 <ea>,Rn
CPUSH	If supervisor state then if data cache push selected dirty data cache lines; invalidate selected cache lines else TRAP	CPUSHL <caches>, (An) CPUSHP <caches>, (An) CPUSHA <caches>
DBcc	If condition false then (Dn–1 ø Dn; If Dn ≠ –1 then PC + d_n ø PC)	DBcc Dn,<label>
DIVS, DIVSL	Destination ÷ Source ø Destination	DIVS.W <ea>,Dn 32 ÷ 16 ø 16r:16q DIVS.L <ea>,Dq 32 ÷ 32 ø 32q DIVS.L <ea>,Dr:Dq 64 ÷ 32 ø 32r:32q DIVSL.L <ea>,Dr:Dq 32 ÷ 32 ø 32r:32q
DIVU, DIVUL	Destination ÷ Source ø Destination	DIVU.W <ea>,Dn 32 ÷ 16 ø 16r:16q DIVU.L <ea>,Dq 32 ÷ 32 ø 32q DIVU.L <ea>,Dr:Dq 64 ÷ 32 ø 32r:32q DIVUL.L <ea>,Dr:Dq 32 ÷ 32 ø 32r:32q
EOR	Source ⊕ Destination ø Destination	EOR Dn,<ea>
EORI	Immediate Data ⊕ Destination ø Destination	EORI #<data>,<ea>
EORI to CCR	Source ⊕ CCR ø CCR	EORI #<data>,CCR
EORI to SR	If supervisor state then Source ⊕ SR ø SR else TRAP	EORI #<data>,SR

SECTION 2
INTEGER UNIT

This section describes the organization of the M68040 integer unit (IU) and presents a brief description of the associated registers. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for details concerning the memory management unit (MMU) programming model, and to **Section 9 Floating-Point Unit (MC68040 Only)** for details concerning the floating-point unit (FPU) programming model.

2.1 INTEGER UNIT PIPELINE

The IU carries out logical and arithmetic operations using six separate subunits. Each unit is dedicated to a different stage of the IU pipeline, handling a total of six separate instructions simultaneously. Pipelining is a technique that overlaps the processing of different parts of several instructions. Pipelining simulates an assembly line with the IU containing a number of instructions in different phases of processing. The IU pipeline consists of six stages:

1. Instruction Fetch—Fetching an instruction from memory.
2. Decode—Converting an instruction into micro-instructions.
3. <ea> Calculate—If the instruction calls for data from memory, the location of the data, its memory address is calculated.
4. <ea> Fetch—Data is fetched from memory.
5. Execute—The data is manipulated during execution.
6. Write-Back—The result of the computation is written back to on-chip caches or external memory.

The pipeline contains special shadow registers that can begin processing future instructions for conditional branches while the main pipeline is processing current instructions. The <ea> calculate stage eliminates pipeline blockage for instructions with postincrement, postdecrement, or immediate add and load to address register for updates that occur in the <ea> calculate stage. The write-back stage can write data over the system bus to store a result in external memory or directly to on-chip caches. These write-backs to memory can be deferred until the most opportune moment because of the M68040 bus interface. Figure 2-1 illustrates the IU pipeline.

0 of an address loaded into the URP or the SRP must be zero. Transfers of data to and from these 32-bit registers are long-word transfers.

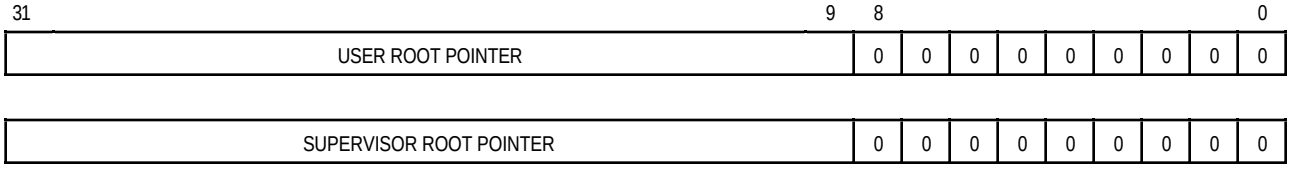
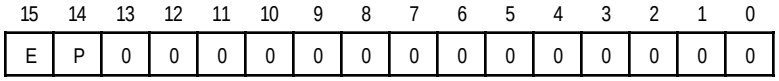


Figure 3-3. URP and SRP Register Formats

3.1.2 Translation Control Register

The 16-bit TCR contains two control bits to enable paged address translation and to select page size. The operating system must flush the ATCs before enabling address translation since the TCR accesses and reset do not flush the ATCs. All unimplemented bits of this register are read as zeros and must always be written as zeros. The M68040 always uses word transfers to access this 16-bit register. The fields of the TCRs are defined following Figure 3-4, which illustrates the TCR.



NOTE: Bits 13–0 are undefined (reserved).

Figure 3-4. Translation Control Register Format

E—Enable

This bit enables and disables paged address translation.

0 = Disable
1 = Enable

A reset operation clears this bit. When translation is disabled, logical addresses are used as physical addresses. The MMU instruction, PFLUSH, can be executed successfully despite the state of the E-bit. PTEST results are undefined if the MMU is disabled and no table search occurs. If translation is disabled and an access does not match a transparent translation register (TTR), the access has the following default attributes on the TTR: the caching mode is cachable/write-through, write protection is disabled, and the user attribute signals (UPA1 and UPA0) are zero.

P—Page Size

This bit selects the memory page size.

0 = 4 Kbytes
1 = 8 Kbytes

A reset operation does not affect this bit. The bit must be initialized after a reset.

implementing multiple MC68040s as bus masters, shared data should be stored in write-through pages. This procedure allows each processor to cache shared data for read access while forcing a processor write to shared data to appear as an external write to memory, which the other processors can snoop.

If shared data is stored in copyback pages, only one processor at a time can cache the data since writes to copyback pages do not access the external bus. If a processor accesses shared data cached by another processor, the slave can source the data to the master without invalidating its own copy only if the transfer to the master is cache inhibited. For the master processor to cache the data, it must force invalidation of the slave processor's copy of the data (by specifying mark invalid for the snoop operation), and the memory controller must monitor the data transfer between the processors and update memory with the transferred data. The memory update is required since the master processor is unaware of the sourced data (valid data from memory or dirty data from a snooping processor) and initially creates a valid cache line, losing dirty status if a snooping processor supplies the data.

Coherency between the instruction cache and the data cache must be maintained in software since the instruction cache does not monitor data accesses. Processor writes that modify code segments (i.e., resulting from self-modifying code or from code executed to load a new page from disk) access memory through the data memory unit. Because the instruction cache does not monitor these data accesses, stale data occurs in the instruction cache if the corresponding data in memory is modified. Invalidating instruction cache lines before writing to the corresponding memory lines can prevent this coherency problem, but only if the data cache line is in write-through mode and the page is marked serialized. A cache coherency problem could arise if the data cache line is configured as copyback and no serialization is done.

To fully support self-modifying code in any situation, it is imperative that a CPUSHA instruction be executed before the execution of the first self-modified instruction. The CPUSHA instruction has the effect of ensuring that there is no stale data in memory, the pipeline is flushed, and instruction prefetches are repeated and taken from external memory.

Another potential coherency problem exists due to the relationship between the cache state information and the translation table descriptors. Because each cache line reflects page state information, a page should be flushed from the cache before any of the page attributes are changed. The presence of a valid or dirty cache line implicitly indicates that accesses to the page containing the line are cachable. The presence of a dirty cache line implies that the page is not write protected and that writes to the page are in copyback mode. A system programming error occurs when page attributes are changed without flushing the corresponding page from the cache, resulting in cache line states inconsistent with their page definitions. Even with these inconsistencies, the cache is defined and predictable.

4.7.1 Instruction Cache

The IU uses the instruction cache to store instruction prefetches as it requests them. Instruction prefetches are normally requested from sequential memory locations except when a change of program flow occurs (e.g., a branch taken) or when an instruction that can modify the status register (SR) is executed, in which case the instruction pipe is automatically flushed and refilled. The instruction cache supports a line-based protocol that allows individual cache lines to be in either the invalid or valid states.

For instruction prefetch requests that hit in the cache, the half-line selected by physical address bit 3 is multiplexed onto the internal instruction data bus. When an access misses in the cache, the cache controller requests the line containing the required data from memory and places it in the cache. If available, an invalid line is selected and updated with the tag and data from memory. The line state then changes from invalid to valid by setting the V-bit. If all lines in the set are already valid, a pseudo-random replacement algorithm is used to select one of the four cache lines replacing the tag and data contents of the line with the new line information. Figure 4-5 illustrates the instruction-cache line state transitions resulting from processor and snoop controller accesses. Transitions are labeled with a capital letter, indicating the previous state, followed by a number indicating the specific case listed in Table 4-3.

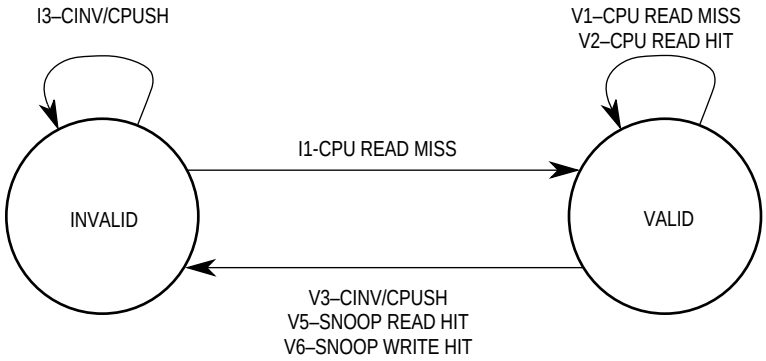


Figure 4-5. Instruction-Cache Line State Diagram

The M68040 takes an interrupt exception for a pending interrupt within one instruction boundary after processing any other pending exception with a higher priority. Thus, the M68040 executes at least one instruction in an interrupt exception handler before recognizing another interrupt request. The following paragraphs describe the various kinds of interrupt acknowledge bus cycles that can be executed as part of interrupt exception processing. Table 7-4 provides a summary of the possible interrupt acknowledge terminations and the exception processing results.

Table 7-4. Interrupt Acknowledge Termination Summary

TA	TEA	AVEC	Termination Condition
High	High	Don't Care	Insert Waits
High	Low	Don't Care	Take Spurious Interrupt Exception
Low	High	High	Latch Vector Number on D7–D0 and Take Interrupt Exception
Low	High	Low	Take Autovectored Interrupt Exception
Low	Low	Don't Care	Retry Interrupt Acknowledge Cycle

7.5.1.1 INTERRUPT ACKNOWLEDGE BUS CYCLE (TERMINATED NORMALLY). When the M68040 processes an interrupt exception, it performs an interrupt acknowledge bus cycle to obtain the vector number that contains the starting location of the interrupt exception handler. Some interrupting devices have programmable vector registers that contain the interrupt vectors for the exception handlers they use. Other interrupting conditions or devices cannot supply a vector number and use the autovector bus cycle described in **7.5.1.2 Autovector Interrupt Acknowledge Bus Cycle**.

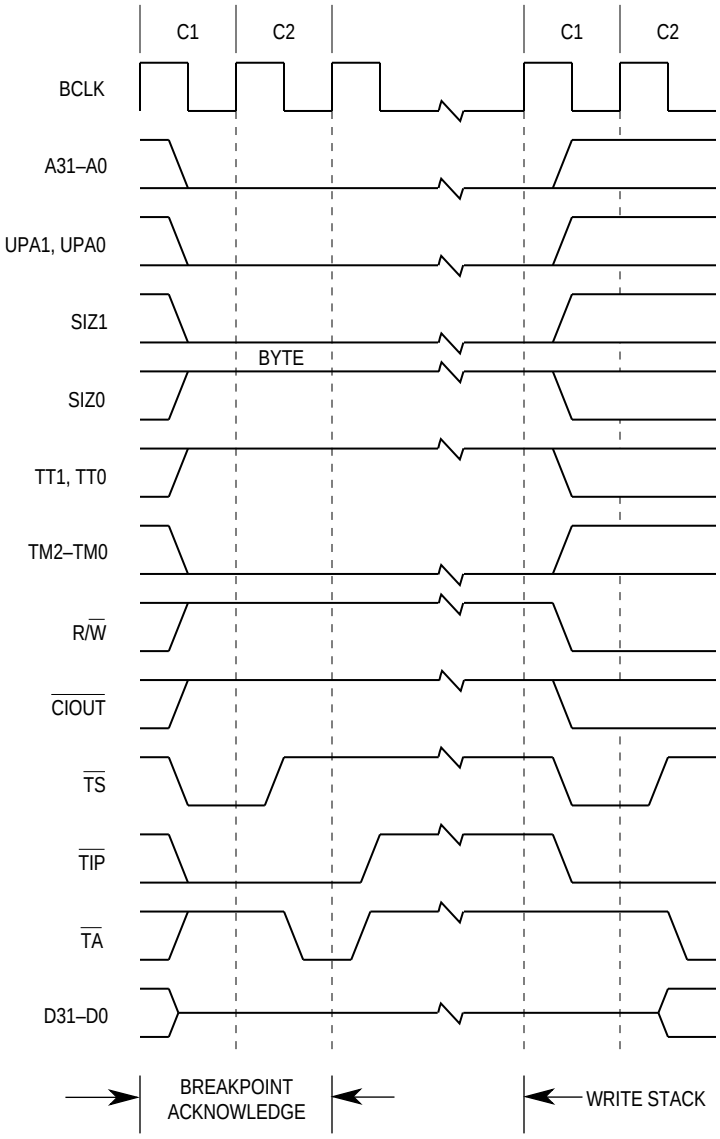


Figure 7-25. Breakpoint Interrupt Acknowledge Bus Cycle Timing

7.6 BUS EXCEPTION CONTROL CYCLES

The M68040 bus architecture requires assertion of $\overline{TA}$ from an external device to signal that a bus cycle is complete. $\overline{TA}$ is not asserted in the following cases:

- The external device does not respond.
- No interrupt vector is provided.
- Various other application-dependent errors occur.

External circuitry can provide $\overline{TEA}$ when no device responds by asserting $\overline{TA}$ within an appropriate period of time after the processor begins the bus cycle. This allows the cycle to terminate and the processor to enter exception processing for the error condition. $\overline{TEA}$ can also be asserted in combination with $\overline{TA}$ to cause a retry of a bus cycle in error.

such as those that generate locked bus transfers or access serialized pages, are allowed to complete before exception processing begins.

Exception processing occurs in four functional steps. However, all individual bus cycles associated with exception processing (vector acquisition, stacking, etc.) are not guaranteed to occur in the order in which they are described in this section. Figure 8-1 illustrates a general flowchart for the steps taken by the processor during exception processing.

During the first step, the processor makes an internal copy of the status register (SR). Then the processor changes to the supervisor mode by setting the S-bit and inhibits tracing of the exception handler by clearing the trace enable (T1 and T0) bits in the SR. For the reset and interrupt exceptions, the processor also updates the interrupt priority mask in the SR.

During the second step, the processor determines the vector number for the exception. For interrupts, the processor performs an interrupt acknowledge bus cycle to obtain the vector number. For all other exceptions, internal logic provides the vector number. This vector number is used in the last step to calculate the address of the exception vector. Throughout this section, vector numbers are given in decimal notation.

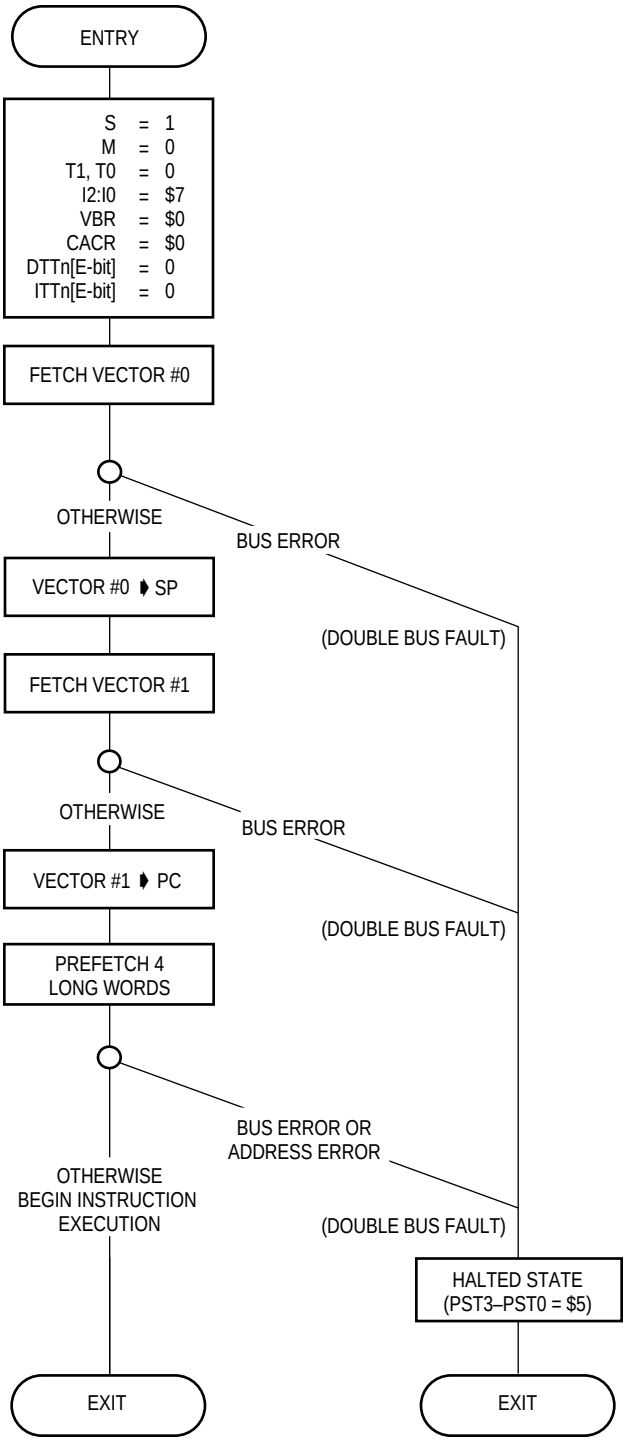


Figure 8-5. Reset Exception Processing Flowchart

After the initial instruction is prefetched, program execution begins at the address in the PC. The reset exception does not flush the ATCs or invalidate entries in the instruction or data caches; it does not save the value of either the PC or the SR. If an access fault or address error occurs during the exception processing sequence for a reset, a double bus fault is generated. The processor halts, and the processor status (PST3–PST0) signals indicate $5. Execution of the reset instruction does not cause a reset exception, or affect

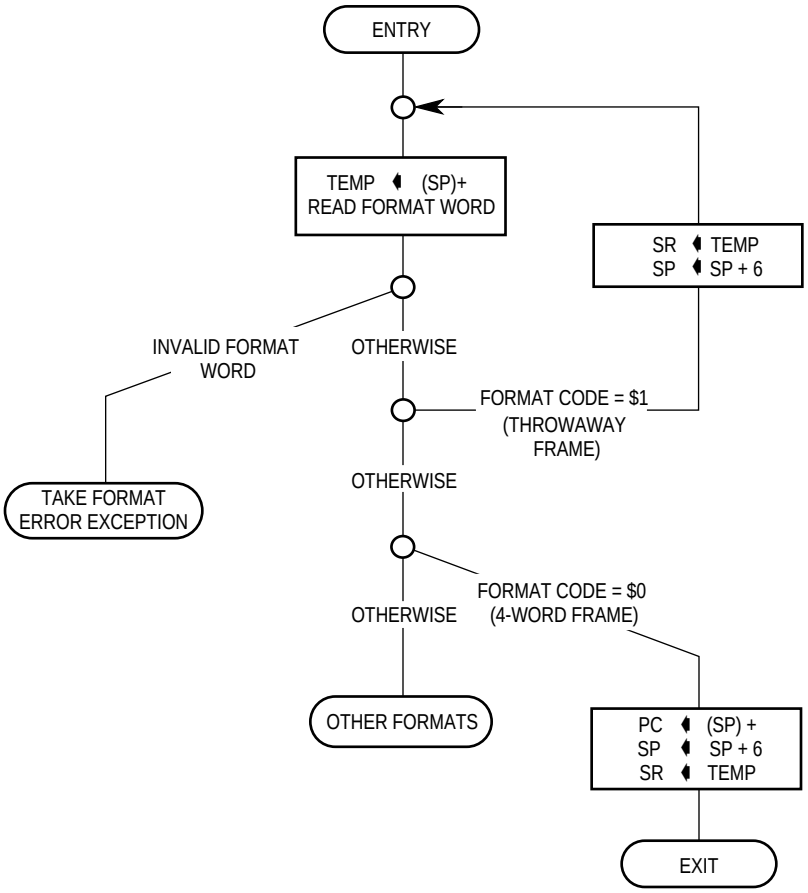
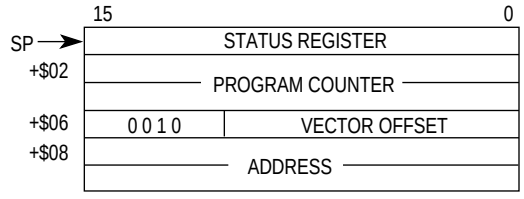


Figure 8-6. Flowchart of RTE Instruction for Throwaway Four-Word Frame

8.4.3 Six-Word Stack Frame (Format $2)

If a six-word throwaway stack frame is on the active stack and an RTE instruction is encountered, the processor restores the SR and PC values from the stack, increments the active supervisor stack pointer by $C, and resumes normal instruction execution.

Stack Frames	Exception Types	Stacked PC Points To
 SIX-WORD STACK FRAME–FORMAT $2	• CHK, CHK2, TRAPcc, FTRAPcc, TRAPV, Trace, or Zero Divide • Unimplemented Floating-Point Instruction • Address Error	• Next Instruction: address is the address of the instruction that caused the exception. • Next Instruction: address is the calculated <ea> for the floating-point instruction. • Instruction that caused the address error, address is the reference address – 1.

the MC68040 does not directly support packed decimal real operands, the processor never sets INEX1 bit in the FPSR EXC byte, but provides it as a latch so that emulation software can report the exception.

A floating-point arithmetic exception is taken in one of two situations. The first situation occurs when the user program enables an arithmetic exception by setting a bit in the FPCR ENABLE byte and the corresponding bit in the FPSR EXC byte matches the bit in the FPCR ENABLE byte as a result of program execution; this is referred to as maskable exception conditions. A user write operation to the FPSR, which sets a bit in the EXC byte, does not cause an exception to be taken, regardless of the value in the ENABLE byte. When a user writes to the ENABLE byte that enables a class of floating-point exceptions, a previously generated floating-point exception does not cause an exception to be taken, regardless of the value in the FPSR EXC byte. The user can clear a bit in the FPCR ENABLE byte, disabling each corresponding exception.

The second situation occurs when the processor encounters a nonmaskable SNAN, OPERR, OVFL, and UNFL condition; this is referred to as nonmaskable exception conditions. This allows a supervisor exception handler to correct a defaulting result generated by the MC68040 that is different from the result generated by an MC68881/MC68882 executing the same code. After correcting the result, the supervisor exception handler calls a user-defined exception handler if the exception has been enabled in the FPCR ENABLE byte or returns to the main program flow if the exception is disabled.

A single instruction execution can generate dual and triple exceptions. When multiple exceptions occur with exceptions enabled for more than one exception class, the highest priority exception is reported; the lower priority exceptions are never reported or taken. The previous list of arithmetic floating-point exceptions is in order of priority. The bits of the ENABLE byte are organized in decreasing priority, with bit 15 being the highest and bit 8 the lowest. The exception handler must check for multiple exceptions. The address of the exception handler is derived from the vector number corresponding to the exception. The following is a list of multiple instruction exceptions that can occur:

- SNAN and INEX1
- OPERR and INEX2
- OPERR and INEX1
- OVFL and INEX2 and/or INEX1
- UNFL and INEX2 and/or INEX1

9.7.1 Branch/Set On Unordered (BSUN)

The BSUN exception is the result of performing an IEEE nonaware conditional test associated with the FBcc, FDBcc, FTRAPcc, and FScc instructions when an unordered condition is present. Refer to **9.5.2 Conditional Testing** for information on conditional tests.

10.1 OVERVIEW

Refer to **Section 2 Integer Unit** for information on the integer unit pipeline. The <ea> fetch timing is not listed in the following tables because most instructions require one clock in the <ea> fetch stage for each memory access to obtain an operand. An instruction requires one clock to pass through the <ea> fetch stage even if no operand is fetched. Table 10-2 summarizes the number of memory fetches required to access an operand using each addressing mode for long-word aligned accesses. The user must perform his own calculations for <ea> fetch timing for misaligned accesses.

Table 10-2. Number of Memory Accesses

Addressing Mode	Evaluate <ea> And Fetch Operand	Evaluate <ea> And Send To Execution Stage
Dn	0	0
An	0	0
(An)	1	0
(An)+	1	0
–(An)	1	0
(d_{16},An)	1	0
(d_{16},PC)	1	0
(xxx).W, (xxx).L	1	0
#<xxx>	0	0
(d_8,An,Xn)	1	0
(d_8,PC,Xn)	1	0
(BR,Xn)	1	0
(bd,BR,Xn)	1	0
([bd,BR,Xn])	2	1
([bd,BR,Xn],od)	2	1
([bd,BR],Xn)	2	1
([bd,BR],Xn,od)	2	1

In the instruction timing tables, the <ea> calculate column lists the number of clocks required for the instruction to execute in the <ea> calculate stage of the integer unit pipeline. Dual effective address instructions such as ABCD –(Ay),–(Ax) require two calculations in the <ea> calculate stage and two memory fetches. Due to pipelining, the fetch of the first operand occurs in the same clock as the <ea> calculation for the second operand.

The execute column lists the number of clocks required for the instruction to execute in the execute stage of the integer unit pipeline. This number is presented as a lead time and a base time. The lead time is the number of clocks the instruction can stall when entering the execution stage without delaying the instruction execution. If the previous instruction is still executing in the execution stage when the current instruction is ready to move from the <ea> fetch stage, the current instruction stalls until the previous one completes. For

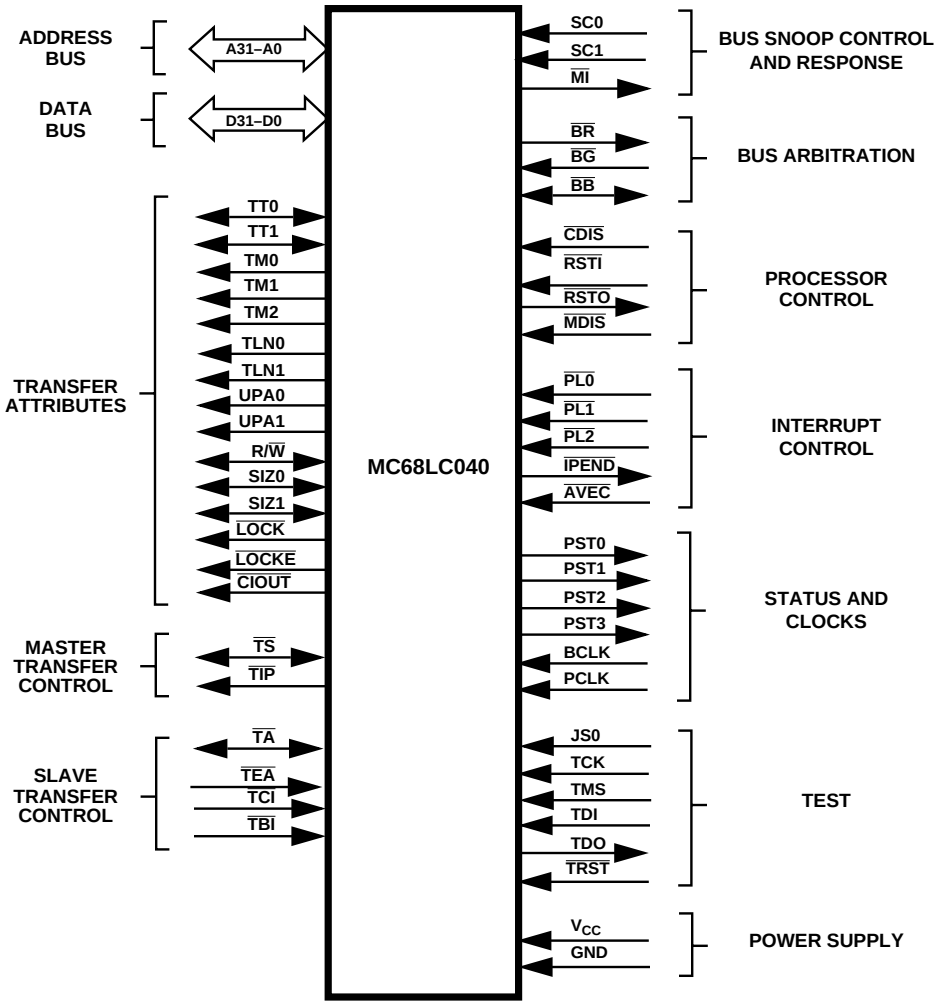


Figure A-3. MC68LC040 Functional Signal Groups

A.1 MC68LC040 DIFFERENCES

The following differences exist between the MC68LC040 and MC68040:

- The MC68LC040 does not implement the small output bufferr impedance selection mode.
- The DLE pin name has been changed to JS0
- The MC68LC040 does not implement the data latch (DLE) or multiplexed bus modes of operation. All timing and drive capabilities of the MC68LC040 are equivalent to those of the MC68040 in small output buffer impedance mode.
- The MC68LC040 does not contain an FPU, which causes unimplemented floating-point exceptions to occur using a new eight-word stack frame format.

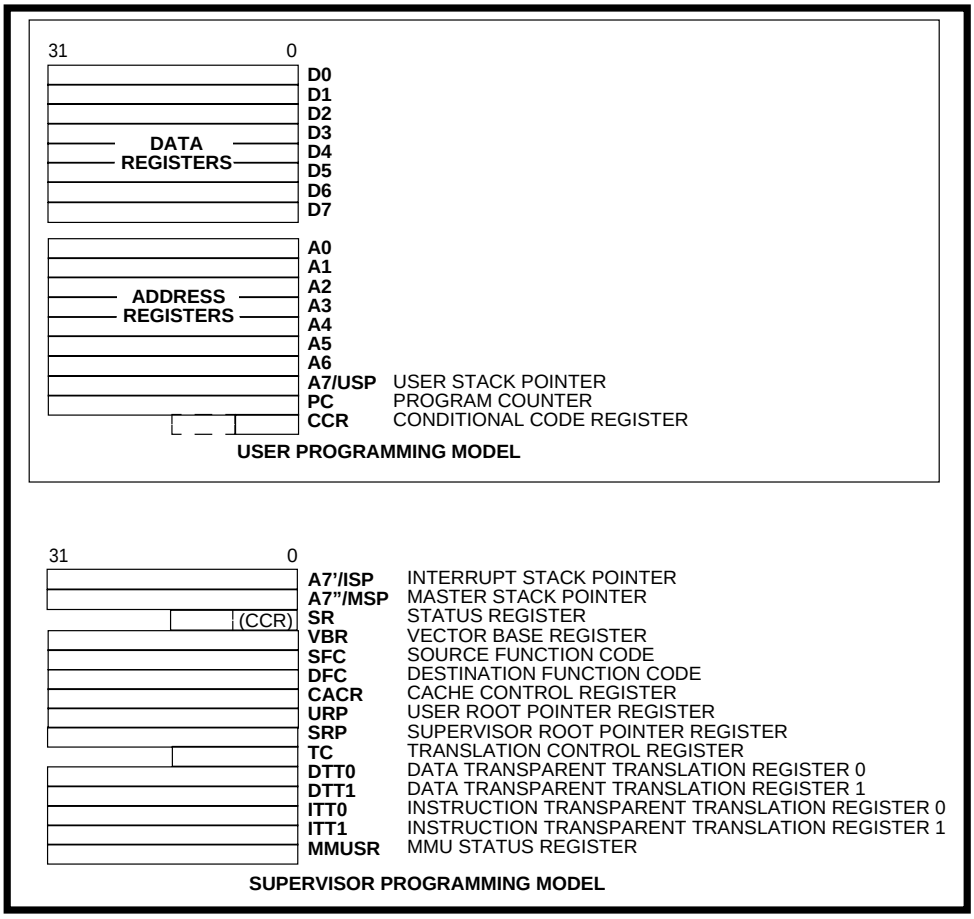


Figure B-2. MC68EC040 Programming Model

- PTEST and PFLUSH instructions cause an indeterminate result (i.e., an undetermined number of bus cycles); the user should not execute them on the MC68EC040.
- The MC68EC040 does not contain an FPU which causes unimplemented floating-point exceptions to occur using a new stack frame format.
- The DLE and $\overline{\text{MDIS}}$ pin names have been changed to JS0 and JS1, respectively.
- The MC68EC040 does not implement the DLE mode, multiplexed, or output buffer impedance selection modes of operation. The MC68EC040 implements only the small output buffer mode of operation. All timing and drive capabilities of the MC68EC040 are equivalent to those of the MC68040 in the small buffer mode of operation.

B.2 JTAG SCAN (JS1–JS0)

The MC68040 $\overline{\text{MDIS}}$ and DLE pin names have been changed to JS1 and JS0 respectively. During normal operation, the JS1 and JS0 pin cannot float, they must be tied to GND or Vcc directly or through a resistor. During board testing, these pins retain the functionality of the JTAG scan of the MC68040 for compatibility purposes. Refer to **Section 6 IEEE 1149.1A**

B.3.1 Access Control Registers

Each ACU has two independent access control registers (ACRs). The instruction ACU contains the instruction access control registers (IACR0 and IACR1). The data ACU contains the data access control registers (DACR0 and DACR1). Both ACRs provide and control status information for access control of memory in the MC68EC040. Only programs that execute in the supervisor mode using the MOVEC instruction can directly access the ACRs.

The 32-bit ACRs each define blocks of MC68EC040:address space for access control. These blocks of address space can overlap or be separate, and are a minimum of 16 Mbytes. Three blocks are used with two user-defined attributes, cachability control and optional write protection. The ACRs specify a block of address space as serialized noncachable for peripheral selections and as write-through for shared blocks of address space in multi-processing applications. The ACRs can be configured to support many embedded control applications while maintaining cache integrity. Refer to **Section 4 Instruction and Data Caches** for details concerning cachability. Figure B-4 illustrates the ACR format.

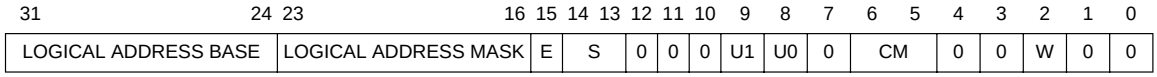


Figure B-4. MC68EC040 Access Control Register Format

ADDRESS BASE

This 8-bit field is compared with physical address bits A31–A24. Addresses that match in this comparison (and are otherwise eligible) are accessible.

ADDRESS MASK

This 8-bit field contains a mask for the ADDRESS BASE field. Setting a bit in the ADDRESS MASK field causes the processor to ignore the corresponding bit in the ADDRESS BASE field. Setting some of the ADDRESS MASK bits to ones obtains blocks of memory larger than 16 Mbytes. The low-order bits of this field are normally set to define contiguous blocks larger than 16 Mbytes, although contiguous blocks are not required.

E—Enable

This bit enables and disables transparent translation of the block defined by this register. Refer to **Section 3 Memory Management Unit (Except MC68EC040 and MC68EC040V)** for details on transparent translation.

0 = Access control disabled.
1 = Access control enabled.

S—Supervisor/User Mode

This field specifies the way FC2 is used in matching an address:

00 =Match only if FC2 = 0 (user mode access).
01 =Match only if FC2 = 1 (supervisor mode access).
10, 11 =Ignore FC2 when matching.

C.2.4 LPSTOP Instruction Summary

Operation: If Supervisor State
Immediate Data ➧ SR
SR ➧ Broadcast Cycle
STOP
Else TRAP

Assembler Syntax: LPSTOP #<data>

Attributes: Privileged Word Sized

Condition Codes: Set according to the immediate operand.

Description: See **C.2 Low Power Stop Mode**.

Instruction Format:

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

Instruction Fields: Immediate field—Specifies the data to be loaded into the status register.

–C–

–D–

–E–

–S–

–T–