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	68040
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	25MHz
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	184-BCQFP
Supplier Device Package	184-CQFP (31.3x31.3)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68lc040fe25a

Table 1-4. Instruction Set Summary

Opcode	Operation	Syntax
ABCD	BCD Source + BCD Destination + X ø Destination	ABCD Dy,Dx ABCD –(Ay),–(Ax)
ADD	Source + Destination ø Destination	ADD <ea>,Dn ADD Dn,<ea>
ADDA	Source + Destination ø Destination	ADDA <ea>,An
ADDI	Immediate Data + Destination ø Destination	ADDI #<data>,<ea>
ADDQ	Immediate Data + Destination ø Destination	ADDQ #<data>,<ea>
ADDX	Source + Destination + X ø Destination	ADDX Dy,Dx ADDX –(Ay),–(Ax)
AND	Source Λ Destination ø Destination	AND <ea>,Dn AND Dn,<ea>
ANDI	Immediate Data Λ Destination ø Destination	ANDI #<data>,<ea>
ANDI to CCR	Source Λ CCR ø CCR	ANDI #<data>,CCR
ANDI to SR	If supervisor state then Source Λ SR ø SR else TRAP	ANDI #<data>,SR
ASL, ASR	Destination Shifted by count ø Destination	ASd Dx,Dy[1] ASd #<data>,Dy[1] ASd <ea>[1]
Bcc	If condition true then PC + d_n ø PC	Bcc <label>
BCHG	~(bit number of Destination) ø Z; ~(bit number of Destination) ø (bit number) of Destination	BCHG Dn,<ea> BCHG #<data>,<ea>
BCLR	~(bit number of Destination) ø Z; 0 ø bit number of Destination	BCLR Dn,<ea> BCLR #<data>,<ea>
BFCHG	~(bit field of Destination) ø bit field of Destination	BFCHG <ea>{offset:width}
BFCLR	0 ø bit field of Destination	BFCLR <ea>{offset:width}
BFEXTS	bit field of Source ø Dn	BFEXTS <ea>{offset:width},Dn
BFEXTU	bit offset of Source ø Dn	BFEXTU <ea>{offset:width},Dn
BFFFO	bit offset of Source Bit Scan ø Dn	BFFFO <ea>{offset:width},Dn
BFINS	Dn ø bit field of Destination	BFINS Dn,<ea>{offset:width}
BFSET	1s ø bit field of Destination	BFSET <ea>{offset:width}
BFTST	bit field of Destination	BFTST <ea>{offset:width}
BKPT	Run breakpoint acknowledge cycle; TRAP as illegal instruction	BKPT #<data>
BRA	PC + d_n ø PC	BRA <label>
BSET	~(bit number of Destination) ø Z; 1 ø bit number of Destination	BSET Dn,<ea> BSET #<data>,<ea>
BSR	SP – 4 ø SP; PC ø (SP); PC + d_n ø PC	BSR <label>

Table 1-4. Instruction Set Summary (Continued)

Opcode	Operation	Syntax
LPSTOP[6]	If supervisor state immediate data ø SR SR ø broadcast cycle STOP else TRAP	LPSTOP #<data>
LSL, LSR	Destination Shifted by count ø Destination	LSd Dx,Dy[1] LSd #<data>,Dy[1] LSd <ea>[1]
MOVE	Source ø Destination	MOVE <ea>,<ea>
MOVEA	Source ø Destination	MOVEA <ea>,An
MOVE from CCR	CCR ø Destination	MOVE CCR,<ea>
MOVE to CCR	Source ø CCR	MOVE <ea>,CCR
MOVE from SR	If supervisor state then SR ø Destination else TRAP	MOVE SR,<ea>
MOVE to SR	If supervisor state then Source ø SR else TRAP	MOVE <ea>,SR
MOVE USP	If supervisor state then USP ø An or An ø USP else TRAP	MOVE USP,An MOVE An,USP
MOVE16	Source block ø Destination block	MOVE16 (Ax)+, (Ay)+[7] MOVE16 (xxx).L, (An) MOVE16 (An), (xxx).L MOVE16 (An)+, (xxx).L
MOVEC	If supervisor state then Rc ø Rn or Rn øRc else TRAP	MOVEC Rc,Rn MOVEC Rn,Rc
MOVEM	Registers ø Destination Source ø Registers	MOVEM <list>,<ea>[4] MOVEM <ea>,<list>[4]
MOVEP	Source ø Destination	MOVEP Dx,(d_n,Ay) MOVEP (d_n,Ay),Dx
MOVEQ	Immediate Data ø Destination	MOVEQ #<data>,Dn
MOVES	If supervisor state then Rn ø Destination [DFC] or Source [SFC] ø Rn else TRAP	MOVES Rn,<ea> MOVES <ea>,Rn
MULS	Source × Destination ø Destination	MULS.W <ea>,Dn 16 × 16 ø 32 MULS.L <ea>,Dl 32 × 32 ø 32 MULS.L <ea>,Dh–Dl 32 × 32 ø 64
MULU	Source × Destination ø Destination	MULU.W <ea>,Dn 16 × 16 ø 32 MULU.L <ea>,Dl 32 × 32 ø 32 MULU.L <ea>,Dh–Dl 32 × 32 ø 64
NBCD	0 – (Destination$_{10}$) – X ø Destination	NBCD <ea>
NEG	0 – (Destination) ø Destination	NEG <ea>
NEGX	0 – (Destination) – X ø Destination	NEGX <ea>

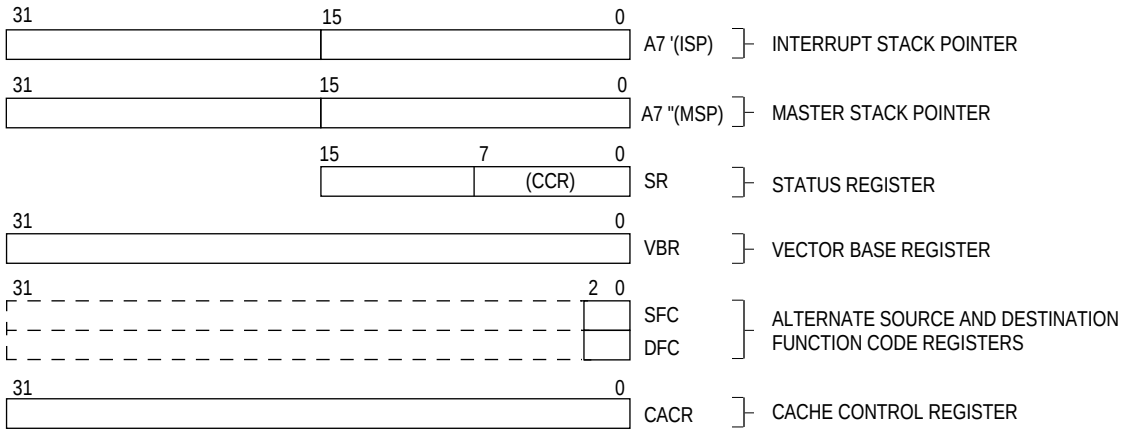


Figure 2-4. Integer Unit Supervisor Programming Model

The supervisor programming model consists of the registers available to the user as well as the following control registers:

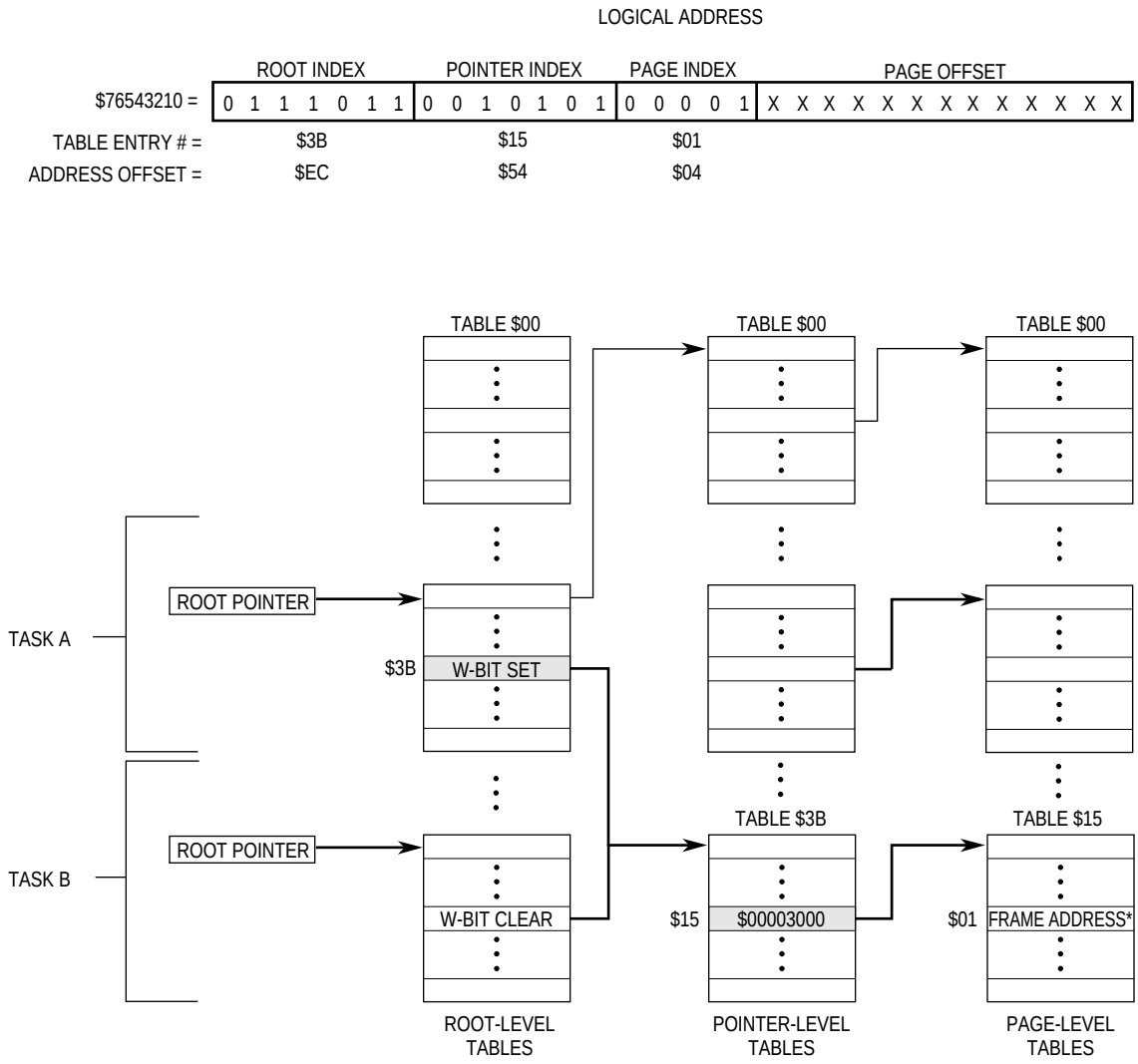
- Two 32-Bit Supervisor Stack Pointers (ISP, MSP)
- 16-Bit Status Register (SR)
- 32-Bit Vector Base Register (VBR)
- Two 32-Bit Alternate Function Code Registers: Source Function Code (SFC) and Destination Function Code (DFC)
- 32-Bit Cache Control Register (CACR)

The following paragraphs describe the supervisor programming model registers. Additional information on the ISP, MSP, SR, and VBR registers can be found in **Section 8 Exception Processing.**

2.2.2.1 INTERRUPT AND MASTER STACK POINTERS. In a multitasking operating system, it is more efficient to have a supervisor stack pointer associated with each user task and a separate stack pointer for interrupt-associated tasks. The M68040 provides two supervisor stack pointers, master and interrupt. Explicit references to the SSP refer to either the MSP or ISP while the processor is operating in the supervisor mode. All instructions that use the SSP implicitly reference the active stack pointer. The ISP and MSP are general-purpose registers and can be used as software stack pointers, index registers, or base address registers. The ISP and MSP can be used for word and long-word operations.

The M-bit of the SR selects whether the ISP or MSP is active. SSP references access the ISP when the M-bit is clear, putting the processor into the interrupt mode. If an exception being processed is an interrupt and the M-bit is set, the M-bit is cleared, putting the processor into the interrupt mode. The interrupt mode is the default condition after reset, and all SSP references access the ISP. The ISP can be used for interrupt control information and for workspace area as interrupt exception handling requires.

SSP references access the MSP when the M-bit is set. The operating system uses the MSP for each task pointing to a task-related area of supervisor data space. This

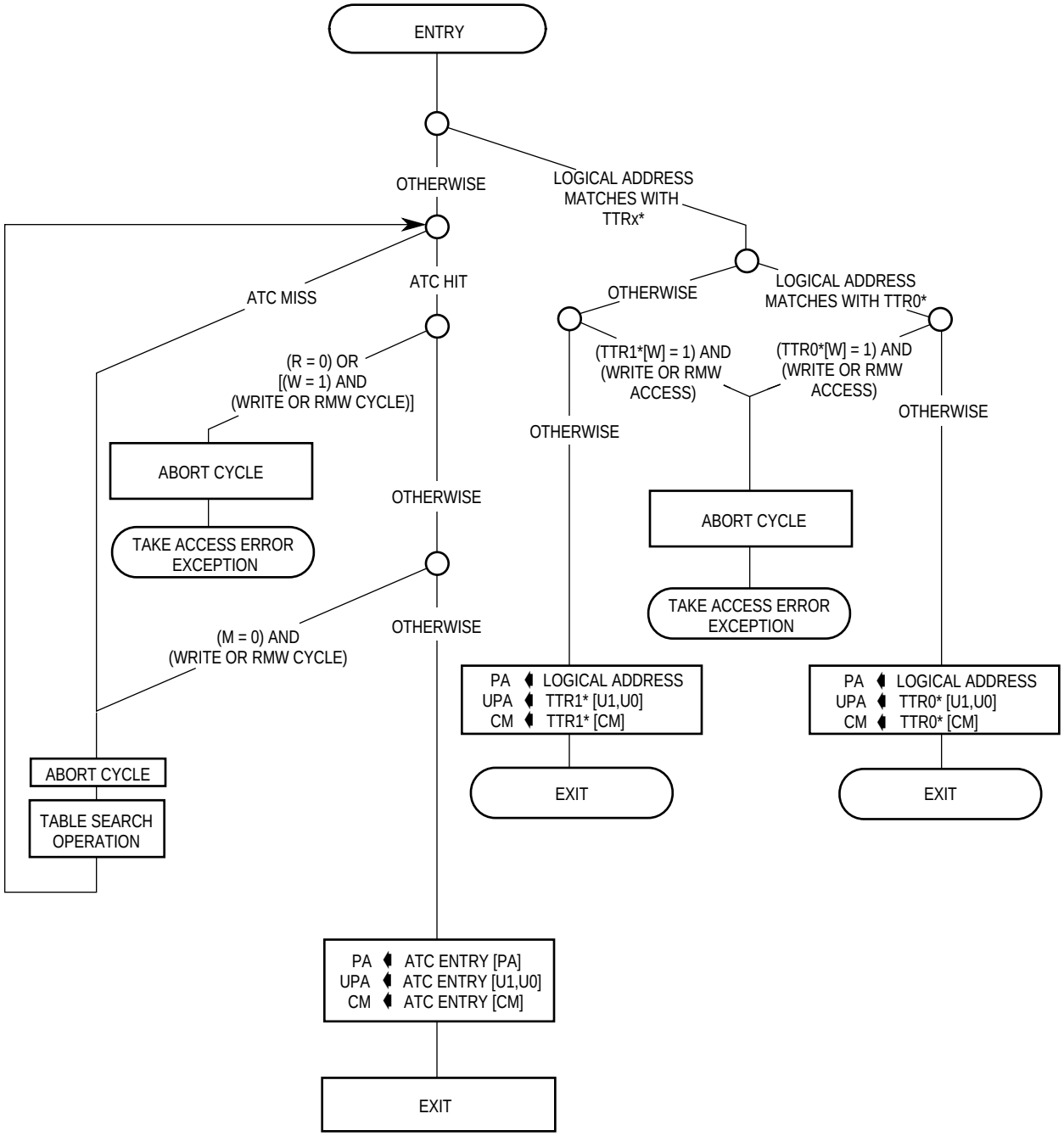


* Page frame address shared by task A and B; write protected from task A.

Figure 3-15. Translation Table Using Shared Tables

3.2.4.3 TABLE PAGING. The entire translation table for an active task need not be resident in main memory. In the same way that only the working set of pages must be allocated in main memory, only the tables that describe the resident set of pages need be available. Placing the invalid code ($0 or $1) in the UDT field of the table descriptor that points to the absent table(s) implements this paging of tables. When a task attempts to use an address that an absent table would translate, the M68040 is unable to locate a translation and takes access error exception when the execution unit retries the bus access that caused the table search to be initiated.

The operating system determines that the invalid code in the descriptor corresponds to nonresident tables. This determination can be facilitated by using he unused bits in the descriptor to store status information concerning the invalid encoding. The M68040 does not interpret or modify an invalid descriptor's fields except for the UDT field. This



* Refers to either instruction or data transparent translation register.

Figure 3-22. Address Translation Flowchart

6.1 OVERVIEW

Figure 6-1 illustrates a block diagram of the M68040 implementation of IEEE standard 1149.1A. The test logic includes a 16-state dedicated TAP controller. These 16 controller states are defined in detail in the IEEE standard 1149.1A, but only 8 are included in this section.

Test-Logic-Reset	Run-Test/Idle
Capture-IR	Capture-DR
Update-IR	Update-DR
Shift-IR	Shift-DR

The TAP controller provides access to five dedicated signal pins:

TCK—A test clock input that synchronizes the test logic.

TMS—A test mode select input with an internal pullup resistor sampled on the rising edge of TCK to sequence the TAP controller.

TDI—A test data input with an internal pullup resistor sampled on the rising edge of TCK.

TDO—A three-state test data output actively driven only in the shift-IR and shift-DR controller states that changes on the falling edge of TCK.

TRST —An active-low asynchronous reset with an internal pullup resistor that forces the TAP controller into the test-logic-reset state.

The test logic also includes an instruction shift register and two test data registers, a boundary scan register and a bypass register. The boundary scan register links all device signal pins into the instruction shift register.

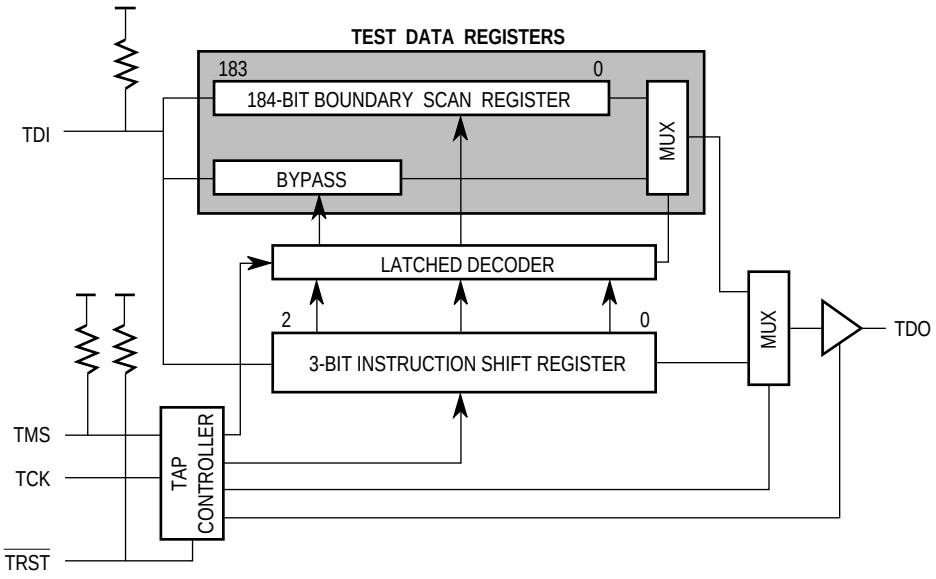


Figure 6-1. M68040 Test Logic Block Diagram

num	cell	port	function	safe	ccell	dsval	rslt			
"114	(BC_4,	D(29),	input,	X),				"	&	
"115	(BC_4,	D(30),	input,	X),				"	&	
"116	(BC_4,	D(31),	input,	X),				"	&	
"117	(BC_2,	A(9),	output3,	X,	150,	0,	Z),	"	&	—150 = io.ab
"118	(BC_4,	A(9),	input,	X),				"	&	
"119	(BC_2,	A(8),	output3,	X,	150,	0,	Z),	"	&	
"120	(BC_4,	A(8),	input,	X),				"	&	
"121	(BC_2,	A(7),	output3,	X,	150,	0,	Z),	"	&	
"122	(BC_4,	A(7),	input,	X),				"	&	
"123	(BC_2,	A(6),	output3,	X,	150,	0,	Z),	"	&	
"124	(BC_4,	A(6),	input,	X),				"	&	
"125	(BC_2,	A(5),	output3,	X,	150,	0,	Z),	"	&	
"126	(BC_4,	A(5),	input,	X),				"	&	
"127	(BC_2,	A(4),	output3,	X,	150,	0,	Z),	"	&	
"128	(BC_4,	A(4),	input,	X),				"	&	
"129	(BC_2,	A(3),	output3,	X,	150,	0,	Z),	"	&	
"130	(BC_4,	A(3),	input,	X),				"	&	
"131	(BC_2,	A(2),	output3,	X,	150,	0,	Z),	"	&	
"132	(BC_4,	A(2),	input,	X),				"	&	
"133	(BC_2,	A(1),	output3,	X,	150,	0,	Z),	"	&	
"134	(BC_4,	A(1),	input,	X),				"	&	
"135	(BC_2,	A(0),	output3,	X,	150,	0,	Z),	"	&	
"136	(BC_4,	A(0),	input,	X),				"	&	
"137	(BC_2,	TM(2),	output3,	X,	156,	0,	Z),	"	&	—156 = io.0
"138	(BC_2,	TM(1),	output3,	X,	156,	0,	Z),	"	&	
"139	(BC_2,	TM(0),	output3,	X,	156,	0,	Z),	"	&	
"140	(BC_2,	TLN(1),	output3,	X,	156,	0,	Z),	"	&	
"141	(BC_2,	TLN(0),	output3,	X,	156,	0,	Z),	"	&	
"142	(BC_2,	SIZ(0),	output3,	X,	156,	0,	Z),	"	&	
"143	(BC_4,	SIZ(0),	input,	X),				"	&	
"144	(BC_2,	R_W,	output3,	X,	156,	0,	Z),	"	&	
"145	(BC_4,	R_W,	input,	X),				"	&	
"146	(BC_2,	LOCKE,	output3,	X,	156,	0,	Z),	"	&	
"147	(BC_2,	SIZ(1),	output3,	X,	156,	0,	Z),	"	&	
"148	(BC_4,	SIZ(1),	input,	X),				"	&	
"149	(BC_2,	LOCK,	output3,	X,	156,	0,	Z),	"	&	
"150	(BC_2,	*,	controlr,	0),				"	&	— io.ab
"151	(BC_2,	*,	controlr,	0),				"	&	— io.db
"152	(BC_2,	MI,	output2,	X),				"	&	
"153	(BC_2,	BR,	output2,	X),				"	&	
"154	(BC_2,	*,	controlr,	0),				"	&	— io.2
"155	(BC_2,	*,	controlr,	0),				"	&	— io.1
"156	(BC_2,	*,	controlr,	0),				"	&	— io.0
"157	(BC_2,	TS,	output3,	X,	156,	0,	Z),	"	&	— 156 = io.0
"158	(BC_4,	TS,	input,	X),				"	&	
"159	(BC_2,	BB,	output3,	X,	155,	0,	Z),	"	&	— 155 = io.1
"160	(BC_4,	BB,	input,	X),				"	&	
"161	(BC_2,	TIP,	output3,	X,	155,	0,	Z),	"	&	— 155 = io.1
"162	(BC_2,	PST(3),	output2,	X),				"	&	
"163	(BC_2,	PST(2),	output2,	X),				"	&	
"164	(BC_2,	PST(1),	output2,	X),				"	&	
"165	(BC_2,	PST(0),	output2,	X),				"	&	
"166	(BC_2,	TA,	output3,	X,	154,	0,	Z),	"	&	— 154 = io.2
"167	(BC_4,	TA,	input,	X),				"	&	
"168	(BC_4,	TEA,	input,	X),				"	&	
"169	(BC_4,	BG,	input,	X),				"	&	
"170	(BC_4,	SC(1),	input,	X),				"	&	

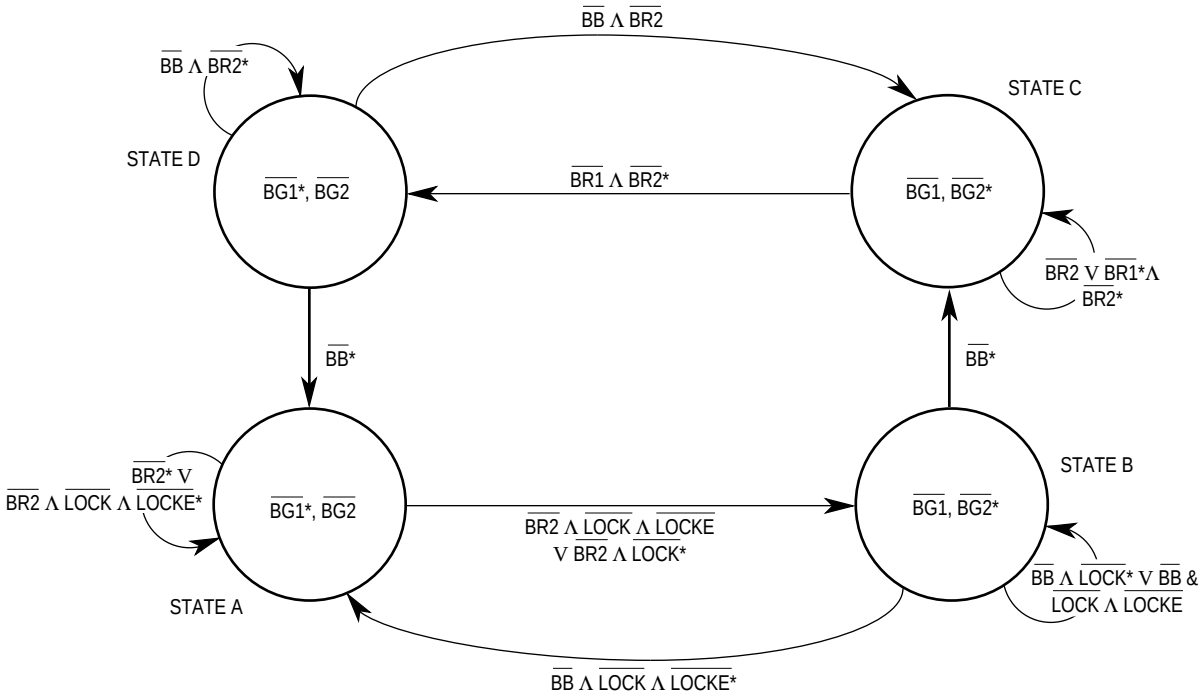
give the bus to processor 2. The arbiter remains in state B until $\overline{BB}$ is negated, signifying the end of the bus cycle.

Once state C is reached, depending on whether or not processor 2 asserts $\overline{BR2}$ and then negates $\overline{BR2}$ because of a disregard request condition, processor 1 may or may not actively begin a bus cycle. If no other bus requests are pending by the time state C is reached, processor 2 is in the implicit ownership state. If processor 1 asserts $\overline{BR1}$, then it is possible for state C to persist for only one clock. In this case, processor 2 does not have a chance to run any active bus cycles.

A null bus cycle tenure is better than having the external bus arbiter wait for processor 2 to perform at least one bus cycle before returning bus ownership to processor 1, even though this appears to be a waste of bus arbitration overhead. Note that once processor 2 enters the disregard request condition, processor 2 reasserts $\overline{BR}$ anywhere from one clock to an undetermined number of clocks before running another bus cycle. Waiting for processor 2 to run a bus cycle can result in a temporary bus arbitration lockup.

This bus arbitration scheme is restricted if the system supports the relinquish and retry operation that can occur for the last write cycle of a locked transfer. In this case, $\overline{LOCKE}$ cannot be used. Assuming that $\overline{LOCKE}$ is always negated excludes the need for $\overline{LOCKE}$ in an arbitration similar to this example. The reason for this restriction is that the external bus arbiter gives up the bus to the other processor once $\overline{LOCKE}$ is asserted. If a relinquish and retry operation were to occur, then the next bus cycle would be from the other processor violating the integrity of the locked transfer.

7.8.2.2 DUAL M68040 PRIORITIZED ARBITRATION. This example is very similar to the dual M68040 fairness arbitration example, except that one processor is assigned higher priority over the other. Processor 2 can own the bus only if there are no processor 1 pending requests. It is important to note that when the processor asserts the $\overline{LOCK}$ signal, it also asserts $\overline{BR1}$. This implementation replaces $\overline{LOCK}$ with $\overline{BR}$ because $\overline{BR}$ is more demanding than using $\overline{LOCK}$. Only when processor 2 is in the middle of a locked operation does it have higher priority than processor 1. Similar to the M68040 fairness arbitration example, the restriction on using $\overline{LOCKE}$ applies to this example. Figure 7-36 illustrates the state diagram for dual M68040 prioritized arbitration.

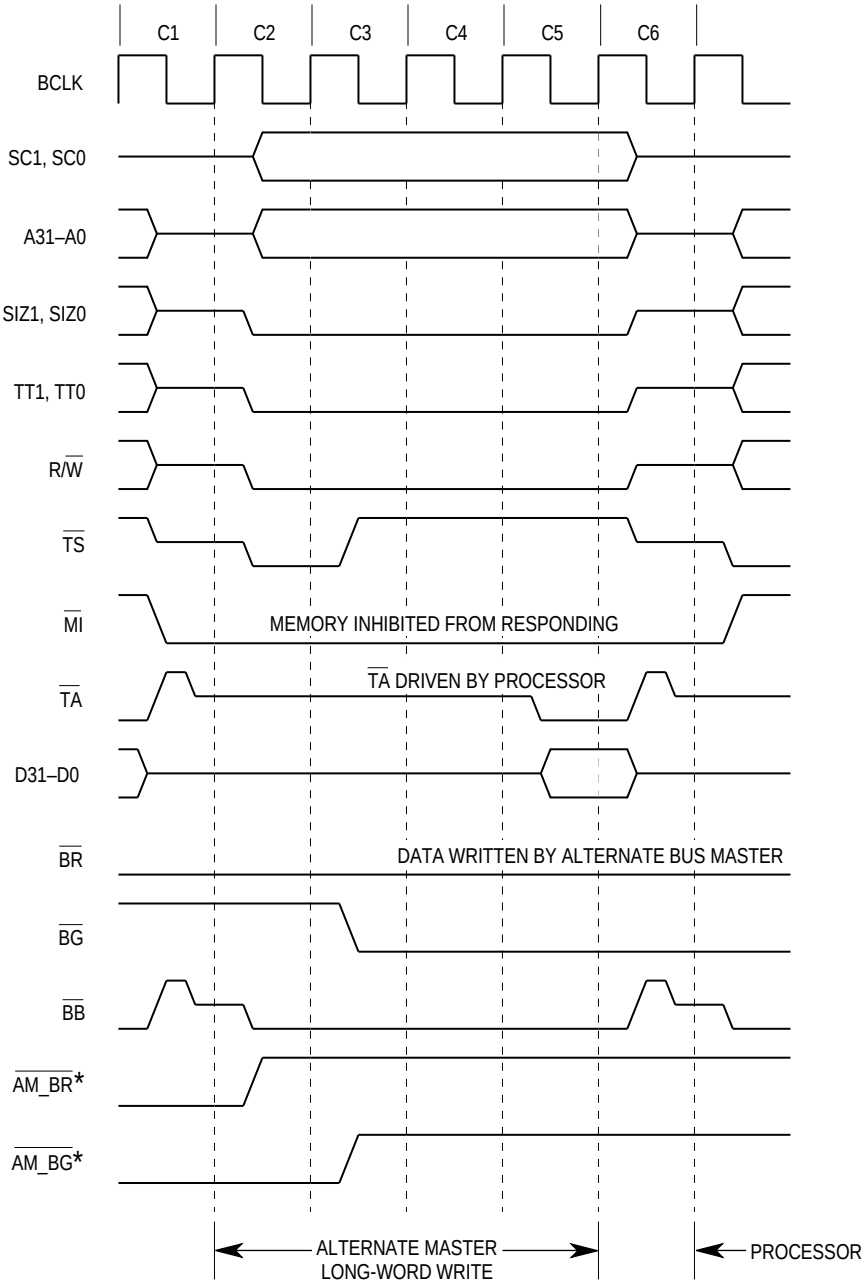


NOTES:

1. Because this example uses two MC68040s, 1 or 2 refers to the processor and its signals.
2. *Indicates the signal is asserted for that device.

Figure 7-36. Dual M68040 Prioritized Arbitration State Diagram

7.8.2.3 M68040 SYNCHRONOUS DMA ARBITRATION. Figure 7-37 illustrates a system with an M68040 and a synchronous direct memory access (DMA) that contains an M68040 interface. Figure 7-37(a) illustrates that the DMA owning the bus only when the M68040 has no pending requests, and Figure 7-37(b) illustrates the DMA having higher priority than the M68040 causing the M68040 to yield the bus to the DMA at any time except when the M68040 is performing a locked bus operation. In either case, the M68040 is the default bus master; if there are no pending requests from either device, the external arbiter gives the bus to the M68040. Similar to the M68040 fairness arbitration example, the restriction on using $\overline{\text{LOCKE}}$ applies to this example.



*AM indicates the alternate bus master.

Figure 7-43. Snooped Long-Word Write, Memory Inhibited

7.10 RESET OPERATION

An external device asserts the reset input signal ($\overline{RSTI}$) to reset the processor. When power is applied to the system, external circuitry should assert $\overline{RSTI}$ for a minimum of 10 BCLK cycles after V_{CC} is within tolerance. Figure 7-44 is a functional timing diagram of the power-on reset operation, illustrating the relationships among V_{CC}, $\overline{RSTI}$, mode selects, and bus signals. The BCLK and PCLK clock signals are required to be stable by the time V_{CC} reaches the minimum operating specification. The V_{IH} levels of the clocks

7.11 SPECIAL MODES OF OPERATION

The MC68LC040 and MC68EC040 do not support the following three modes of operation, which for the M68040 are selectively enabled during processor reset and remain in effect until the next processor reset. Refer to **Appendix A MC68LC040** and **Appendix B MC68EC040** for differences in the special modes of operation for the MC68LC040 and MC68EC040.

7.11.1 Output Buffer Impedance Selection

All output drivers in the M68040 can be configured to operate in either a large buffer mode (low-impedance driver) or small buffer mode (high-impedance driver). Large buffers have a nominal output impedance of 6 Ω for both high and low drive, resulting in minimum output delays. Signal traces driven by large buffers usually require transmission line effects to be considered in their design, including the use of signal termination. Small buffers have a nominal impedance of 25 Ω for high and low drive, resulting in longer output delays and less critical board-design requirements. Refer to **Section 11 MC68040 Electrical and Thermal Characteristics** for further information on electrical specifications, buffer characteristics, and transmission line design examples. The output drivers are configured in three groups. Each group of signals is configured depending on the corresponding $\overline{\text{IPLx}}$ signal level during processor reset (see Table 5-5).

7.11.2 Multiplexed Bus Mode

The multiplexed bus mode changes the timing of the three-state control logic for the address and data buses to support generation of a multiplexed address/data bus. When the M68040 is operating in this mode, the address and data bus signals can be hardwired together to form a single 32-bit bus, with address and data information time-multiplexed on the bus. This configuration minimizes the number of pins required to interface to peripheral devices without requiring additional discrete multiplexing logic. This mode is enabled during a processor reset by a logic zero on the $\overline{\text{CDIS}}$ signal.

Figure 7-46 illustrates a line write with multiplexed bus mode enabled. The address bus drivers are enabled during C1 and disabled during C2. Later in C2, the data bus drivers are enabled to drive the data bus with the data to be written. The address bus is only driven for the BCLK rising edge at the start of each bus cycle.

- Trace
- Format Error
- Breakpoint Instruction
- Interrupt
- Reset

8.2.1 Access Fault Exception

An access fault exception occurs when a data or instruction prefetch access faults due to either an external bus error or an internal access fault. Both types of access faults are treated identically and the access fault exception handler or a status bit in the access fault stack frame distinguishes them. An access fault exception may or may not be taken immediately, depending on whether the faulted access specifically references data required by the execution unit or whether there are any other exceptions that can occur, allowing the execution pipeline to idle.

An external access fault (bus error) occurs when external logic aborts a bus cycle and asserts the $\overline{\text{TEA}}$ input signal. A bus error on a data write access always results in an access fault exception, causing the processor to begin exception processing immediately. A bus error on a data read also causes exception processing to begin immediately if the access is a byte, word, or long-word access or if the bus error occurs on the first transfer of a line read. Bus errors on the second, third, or fourth transfers for a data line read cause the transfer to be aborted, but result in a bus error only if the execution unit is specifically requesting the long word being transferred. For example, if a misaligned operand spans the first two long words in the line being read, a bus error on the second transfer causes an exception, but a bus error on the third or last transfer does not, unless the execution unit has generated another operand access that references data in these transfers.

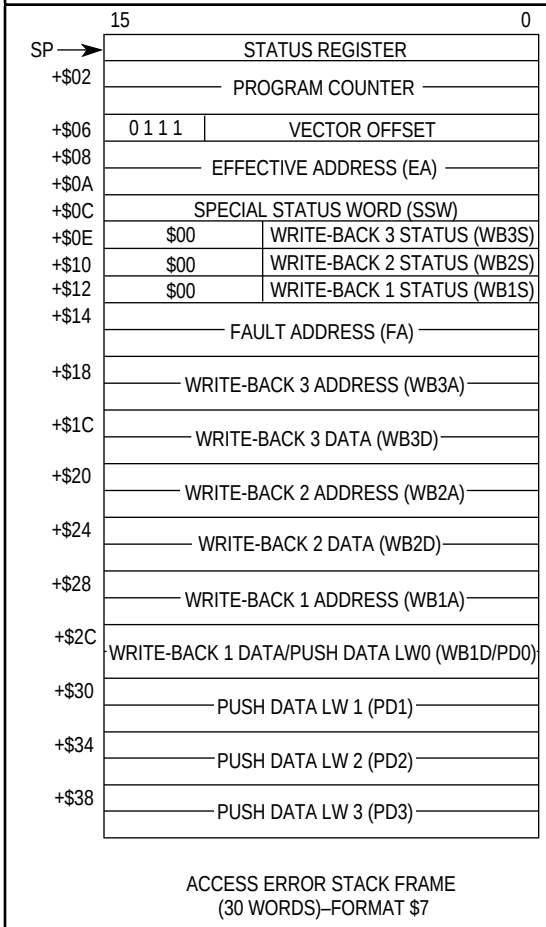
Bus errors that occur during instruction prefetches are deferred until the processor attempts to use the information. For instance, if a bus error occurs while prefetching other instructions after a change-of-flow instruction (BRA, JMP, JSR, TRAP#n, etc.), BRA, JMP, JSR, TRAP#n execution of the new instruction flow clears the exception condition. This also applies to the not-taken branch for a conditional branch instruction, even though both sides of the branch are decoded.

Processor accesses for either data or instructions can result in internal access faults. Internal access faults must be corrected to complete execution of the current context. Four types of internal access faults can occur:

1. Push transfer faults occur when the execution unit is idle, the integer unit pipeline is frozen, the instruction and data cache requests are cancelled (however, writes are not lost), and pending writes are stacked.
2. Data access faults occur when the bus controller and the execution unit are idle. A data access fault freezes the pipeline and cancels any pending instruction cache accesses. Pending writes are stacked because the data cache is deadlocked until stacking transfers are initiated.

8.4.6 Access Error Stack Frame (Format $7)

A 30-word access error stack frame is created for data and instruction access faults other than instruction address errors. In addition to information about the current processor status and the faulted access, the stack frame also contains pending write-backs that the access error exception handler must complete. The following paragraphs describe in detail the format for this frame and how the processor uses it when returning from exception processing.

Stack Frames	Exception Types	Stacked PC Points To
	• Data or Instruction Access Fault (ATC Fault or Bus Error)	• Next Instruction

8.4.6.1 EFFECTIVE ADDRESS. The effective address contains address information when one of the continuation flags CM, CT, CU, or CP in the SSW is set.

8.4.6.2 SPECIAL STATUS WORD (SSW). The SSW information indicates whether an access to the instruction stream or the data stream (or both) caused the fault and contains status information for the faulted access. Figure 8-7 illustrates the SSW format.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CP	CU	CT	CM	MA	ATC	LK	RW	X	SIZE		TT		TM		

Figure 8-7. Special Status Word Format

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.

4. Examining the conditional predicate and setting the FPCC NAN bit accordingly prevents the exception from being taken again. This technique gives the most control since it is possible to pre-determine the direction of program flow. Bit 7 of the F-line operation word indicates where the conditional predicate is located. If bit 7 is set, the conditional predicate is the lower six bits of the F-line operation word. Otherwise, the conditional predicate is the lower six bits of the instruction word, which immediately follows the F-line operation word. Using the conditional predicate and the table for IEEE nonaware test in **9.5.2 Conditional Testing**, the condition codes can be set to return a known result indication when the conditional instruction is reexecuted.

Prior to exiting the user BSUN exception handler, the exception handler discards the floating-point state frame.

9.7.1.2 NONMASKABLE EXCEPTION CONDITIONS. There are no conditions.

9.7.2 Signaling Not-a-Number (SNAN)

An SNAN is used as an escape mechanism for a user-defined, non-IEEE data type. The processor never creates an SNAN as a result of an operation; a NAN created by an operand error exception is always a nonsignaling NAN. When an operand is an SNAN involved in an arithmetic instruction, the SNAN bit is set in the FPSR EXC byte. Since the FMOVEM, FMOVE FPCR, and FSAVE instructions do not modify the status bits, they cannot generate exceptions. Therefore, these instructions are useful for manipulating SNANs.

9.7.2.1 MASKABLE EXCEPTION CONDITIONS. When an SNAN is encountered, if the destination is a floating-point data register or is in memory (or an integer data register) and the format is single, double, or extended precision, the SNAN is maskable and may or may not take an exception.

a. If the user SNAN exception is disabled, the processor clears the SNAN bit in the NAN data format and the resulting nonsignaling NAN is transferred to the destination. No bits other than the SNAN bit of the NAN data format are modified, although the input NAN is truncated if necessary. Instruction execution continues without taking any exceptions.

b. If the user SNAN exception handler is enabled, the processor posts an exception and another floating-point instruction is eventually encountered; a pre-instruction exception is reported at that time. The SNAN entry in the processor's vector table points to the M68040FPSP SNAN exception handler. Once the M68040FPSP SNAN exception handler recognizes the operand error as a maskable condition, it does not modify the destination or pass control to the user SNAN exception handler.

9.7.2.2 NONMASKABLE EXCEPTION CONDITIONS. When an SNAN is encountered, if the destination is either in memory or an integer data register and the format is byte, word, or long word, a nonmaskable post-instruction exception occurs and is taken immediately. The SNAN entry in the processor's vector table points to the M68040FPSP SNAN exception handler.

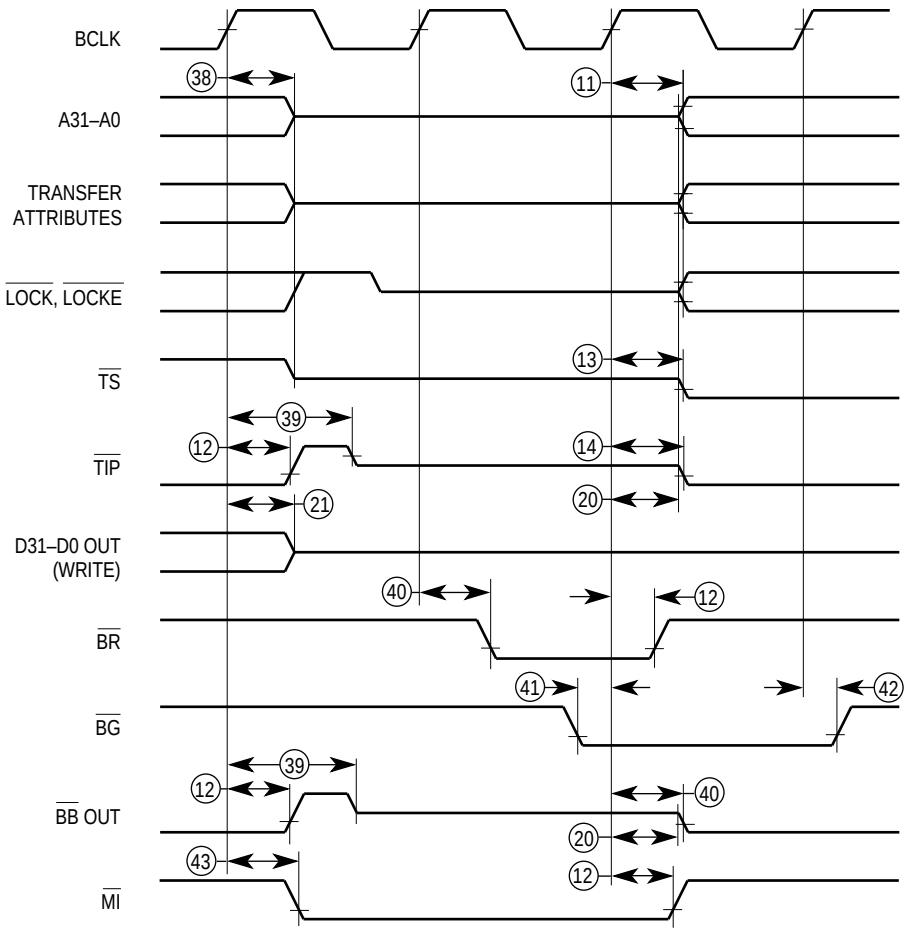
10.7 FLOATING-POINT UNIT INSTRUCTION TIMINGS

For floating-point instructions in the MC68040, the integer pipeline passes the decoded instruction to the floating-point unit for execution, then supports the floating-point unit by calculating effective addresses and transferring operands to and from this unit. For these instructions, the execution times listed in the integer unit timing section show the overhead required by the integer unit to support the floating-point unit, assuming the floating-point unit is not busy with the previous floating-point instructions.

Times in parentheses are the total time that that stage uses to execute an instruction even though the stage can pass data to the next stage early. The order of operands is generally not significant for timing purposes. Different rounding modes (i.e., round to zero, etc.) never incur a time penalty. Instructions with an S or D (e.g., FSADD) have the same effect as setting the rounding precision to S or D. All FMOVEM instructions wait for the pipe to idle before starting. Refer to **Section 9 Floating-Point Unit (MC68040 Only)** for details on the operation of the floating-point unit pipeline.

10.7.1 Miscellaneous Integer Unit Support Timings

Instruction	Condition	<ea> Calculate	Execute
FBcc	Taken Not Taken	7 6	7 6
FDBcc	cc True cc False	9 11	1_L + 7 1_L + 9
FNOP	FPU Idle	6	6
FTRAPcc	Not Taken	6	1_L + 5



NOTE: Transfer Attribute Signals = UPAx, SIZx, TTx, TMx, TLNx, R/W, CIOUT

Figure 11-4. Bus Arbitration Timing

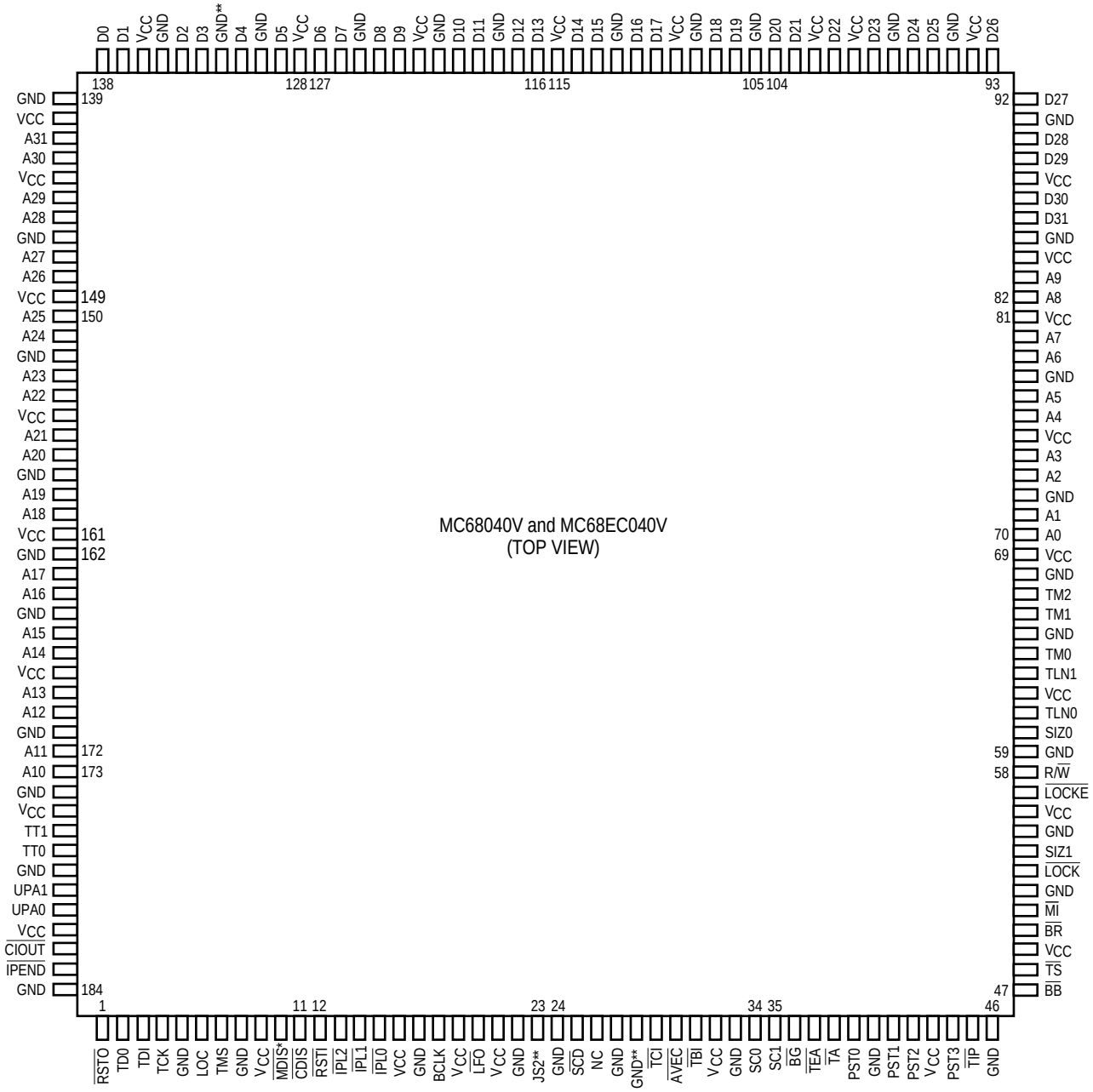
11.9.2 Forced Air

In this study, a small sample of MC68040 packages was tested in forced-air cooling in a wind tunnel with no heat sink. The test was performed with approximately 6 W of power being dissipated from within the package. As previously mentioned, since the variance in θ_{JA} within the possible power range is negligible, it can be assumed for calculation purposes that θ_{JA} is constant at all power levels. Using the previous equations, Table 11-3 lists the results of the maximum power dissipation at maximum θ_{JC} with airflow and no heat sink for the MC68040, and Table 11-4 lists the results for the MC68LC040 and MC68EC040. Refer to Table 11-1 for calculating other power dissipation values.

Table 11-3. Thermal Parameters with Forced Airflow and No Heat Sink for the MC68040

	Thermal Mgmt. Technique	Defined Parameters			Measured	Calculated		
MHz	Airflow Velocity	P_D	T_J	θ_{JC}	θ_{JA}	θ_{CA}	T_C	T_A
25	100 LFM	6.3 W	110 °C	3 °C/W	12.7 °C/W	9.7 °C/W	91.1 °C	29.90 °C
25		6.6 W					90.2 °C	26.18 °C
25		8.6 W					84.9 °C	00.76 °C
33		7.7 W					86.9 °C	12.21 °C
33		8.0 W					86.0 °C	08.40 °C
33		10.0 W					80.0 °C	00.00°C
25	250 LFM	6.3 W	110 °C	3 °C/W	11.0 °C/W	8.0 °C/W	91.1 °C	40.70 °C
25		6.6 W					90.2 °C	37.40 °C
25		8.6 W					84.2 °C	15.40 °C
33		7.7 W					86.9 °C	25.30 °C
33		8.0 W					86.0 °C	22.00 °C
33		10.0 W					80.0 °C	00.00 °C
25	500 LFM	6.3 W	110 °C	3 °C/W	9.9 °C/W	6.9 °C/W	91.1 °C	47.63 °C
25		6.6 W					90.2 °C	44.66 °C
25		8.6 W					84.2 °C	24.86 °C
33		7.7 W					86.9 °C	33.77 °C
33		8.0 W					86.0 °C	30.80 °C
33		10.0 W					80.0 °C	11.00 °C
25	750 LFM	6.3 W	110 °C	3 °C/W	9.5 °C/W	6.5 °C/W	91.1 °C	50.15 °C
25		6.6 W					90.2 °C	47.30 °C
25		8.6 W					84.2 °C	28.30 °C
33		7.7 W					86.9 °C	36.85 °C
33		8.0 W					86.0 °C	34.00 °C
33		10.0 W					80.0 °C	15.00 °C
25	1000 LFM	6.3 W	110 °C	3 °C/W	9.3 °C/W	6.3 °C/W	91.1 °C	51.41 °C
25		6.6 W					90.2 °C	48.62 °C
25		8.6 W					84.2 °C	30.02 °C
33		7.7 W					86.9 °C	38.39 °C
33		8.0 W					80.0 °C	17.00 °C
33		10.0 W					81.8 °C	22.58 °C

12.2.7 MC68040V and MC68EC040V Quad Flat Pack



NOTES:
* On MC68EC040V this pin is called JS1.
** All these pins are in the JTAG scan chain. On an MC68040 design JS2 = GND; on an MC68060 design JS2 = CLK.

$\overline{RSTO}$ are reset at the completion of the RESET instruction. An $\overline{RSTI}$ signal that is asserted to the processor during execution of a RESET instruction immediately resets the processor and causes $\overline{RSTO}$ to negate. $\overline{RTSO}$ can be logically ANDed with the external signal driving $\overline{RTSI}$ to derive a system reset signal that is asserted for both an external processor reset and execution of a RESET instruction.

B.5 EXCEPTION PROCESSING

The MC68EC040 provides five different stack frames for exception processing and allows for a MC68040-specific stack frame. Refer to **Section 8 Exception Processing** for details on exception processing.

B.5.1 Unimplemented Floating-Point Instructions and Exceptions

All legal MC68040 and MC68881/MC68882 floating-point instructions are defined as unimplemented floating-point instructions on the MC68EC040. These instructions generate an eight-word stack frame (format $4) during exception processing before taking an F-line exception. These instructions trap as an F-line exception and can be emulated in software by the F-line exception handler to maintain user-object-code compatibility.

The MC68EC040 assists the emulation process by distinguishing unimplemented floating-point instructions from other unimplemented F-line instructions. To aid emulation, the effective address is calculated and saved in the format $4 stack frame. This simplifies and speeds up the emulation process by eliminating the need for the emulation routine to determine the effective address and by providing information required to emulate the instruction on the exception stack frame in the supervisor address space. However, the floating-point instruction can reside in user space; therefore, the floating-point unimplemented exception handler may need to access user instruction space. The following processing steps occur for an unimplemented floating-point instruction:

1. When an unimplemented floating-point instruction is encountered, the instruction is partially decoded, and the effective address is calculated, if required.
2. The processor waits for all previous integer instructions, write-backs, and associated exception processing to complete before beginning exception processing for the unimplemented floating-point instruction. Any access error that occurs in completing the write-backs causes an access error exception, and the resulting stack frame indicates a pending unimplemented floating-point instruction exception. The access error exception handler then completes the write-backs in software, and exception processing for the unimplemented floating-point instruction exception begins immediately after return from the access error handler.
3. The processor begins exception processing for the unimplemented floating-point instruction by making an internal copy of the current SR. The processor then enters the supervisor mode and clears the trace bits (T1 and T0). It creates a format $4 stack frame and saves the internal copy of the SR, PC, vector offset, calculated effective address, and PC value of the faulted instruction in the stack frame.

   The effective address field of the format $4 stack frame contains the calculated effective address of the operand for the faulted floating-point instruction using the addressing mode in which the effective address is calculated. For immediate and register