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	68060
Number of Cores/Bus Width	1 Core, 32-Bit
Speed	50MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	5V
Operating Temperature	0°C ~ 70°C (TA)
Security Features	-
Package / Case	304-LBGA Exposed Pad
Supplier Device Package	304-TBGA (31x31)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68lc060zu50

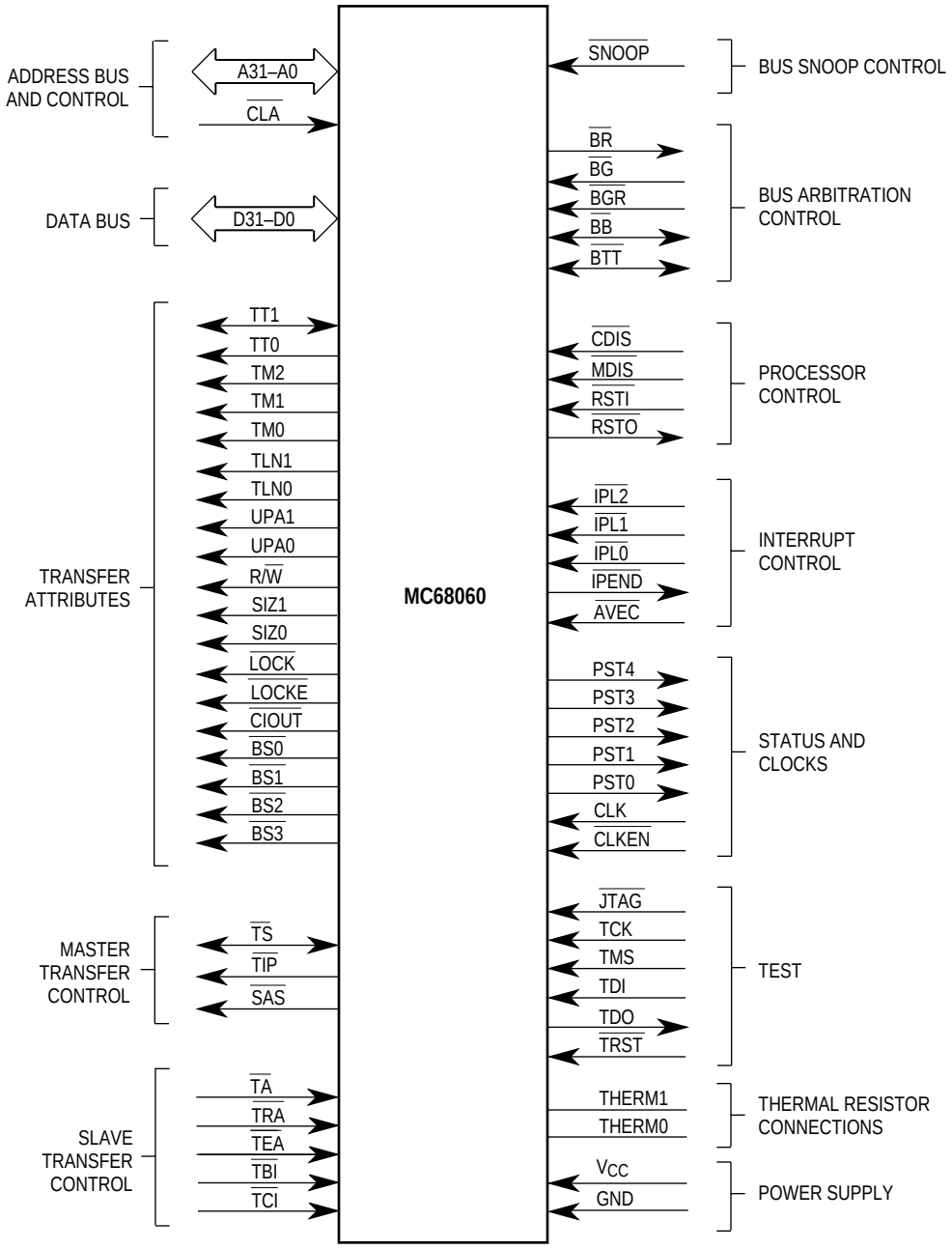


Figure 2-1. Functional Signal Groups

2.1 ADDRESS AND CONTROL SIGNALS

The following paragraphs describe the MC68060 address and control signals.

2.1.1 Address Bus (A31–A0)

These three-state bidirectional signals provide the address of the first item of a bus transfer (except for interrupt acknowledge transfers) when the MC68060 is the bus master. When an alternate bus master is controlling the bus and asserts the $\overline{\text{SNOOP}}$ signal, the address sig-

4.2.4 Variations in Translation Table Structure

Several aspects of the MMU translation table structure are software configurable, allowing the system designer flexibility to optimize the performance of the MMUs for a particular system. The following paragraphs discuss the variations of the translation table structure.

4.2.4.1 INDIRECT ACTION. The MC68060 provides the ability to replace an entry in a page table with a pointer to an alternate entry. The indirection capability allows multiple tasks to share a physical page while maintaining only a single set of history information for the page (i.e., the modified indication is maintained only in the single descriptor). The indirection capability also allows the page frame to appear at arbitrarily different addresses in the logical address spaces of each task.

Using the indirection capability, single entries or entire tables can be shared between multiple tasks. Figure 4-13 illustrates two tasks sharing a page using indirect descriptors.

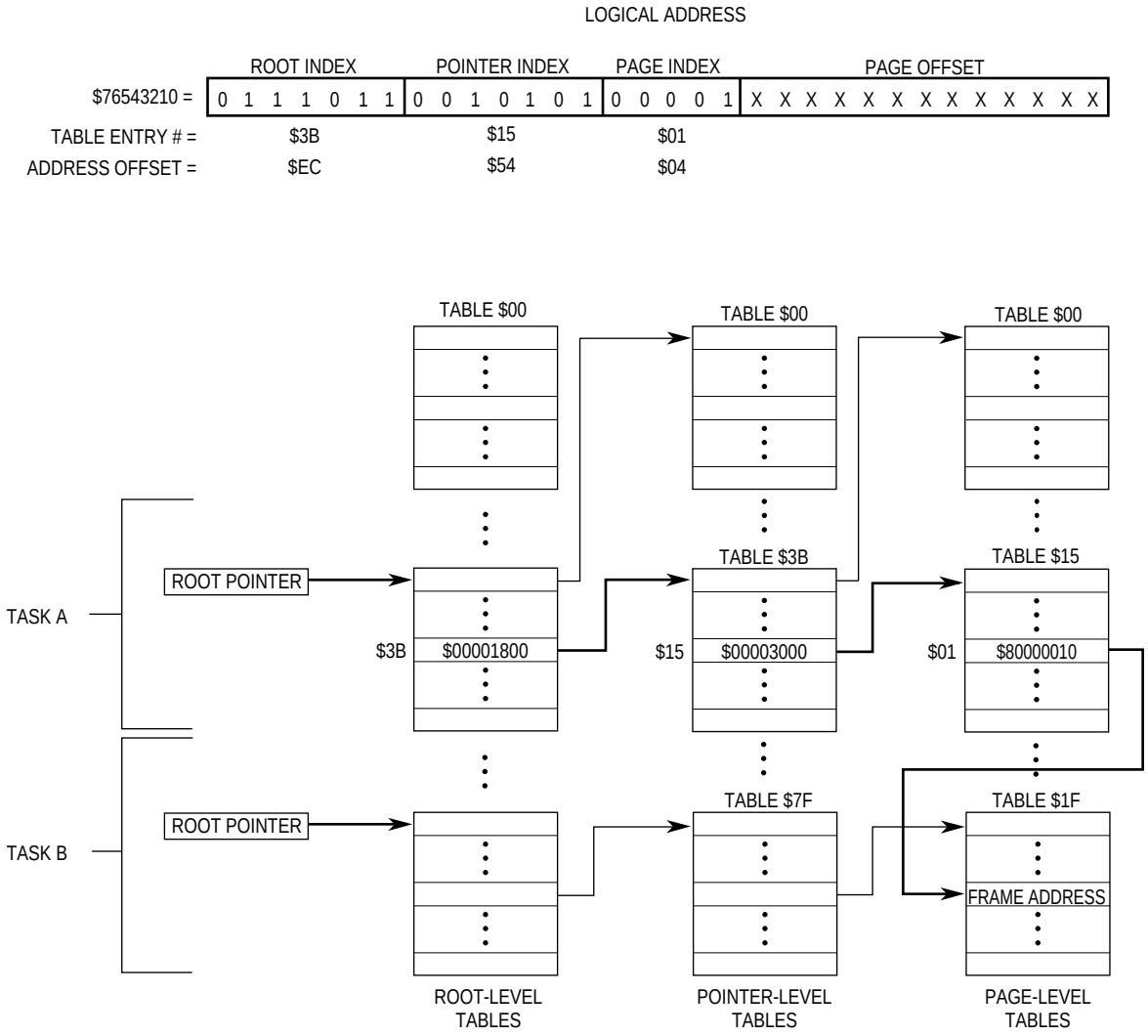


Figure 4-13. Translation Table Using Indirect Descriptors

5.5.3 Read Hit

On a read hit, the appropriate cache provides the data to the requesting pipe unit. In most cases no bus transaction is performed, and the state of the cache line does not change. However, when a writethrough read hit to a line containing dirty data occurs, the dirty line is pushed and the cache line state changes to valid before the data is provided to the requesting pipe unit.

A snooped external read hit invaildates the cache line that is hit.

5.5.4 Write Hit

The cache controller handles processor writes that hit in the cache differently for writethrough and copyback pages. For write hits to a writethrough page, the portions of the cache line(s) corresponding to the size of the access are updated with the data, and the data is also written to external memory. The cache line state does not change. A writethrough access to a line containing dirty data results in the dirty line being pushed and then witten to memory. If the access is copyback, the cache controller updates the cache line and sets the D-bit for the line. An external write is not performed, and the cache line state changes to, or remains in, the dirty state.

An alternate bus master can assert the $\overline{\text{SNOOP}}$ signal for a write that it initiates, which will invalidate any corresponding entry in the internal cache.

5.6 CACHE COHERENCY

The MC68060 provides several different mechanisms to assist in maintaining cache coherency in multimaster systems. Both writethrough and copyback memory update techniques are supported to maintain coherency between the data cache and memory.

Alternate bus master accesses can reference data that the MC68060 may have cached, causing coherency problems if the accesses are not handled properly. The MC68060 snoops the bus during alternate bus master transfers if $\overline{\text{SNOOP}}$ is asserted. Snoop hits invalidate the cache line in all cases (read, write, long word, word, byte) for MOVE16 and normal accesses. Since the processor may be accessing data in its caches even when it does not have the bus, a snoop has priority over the processor, to maintain cache coherency.

The snooping protocol and caching mechanism supported by the MC68060 requires that pages shared with any other bus master be marked cachable writethrough or cache inhibited (either precise or imprecise). 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, cache coherency is not guaranteed.

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

tests that set the BSUN bit in the FPSR status byte if an unordered condition is present when the conditional test is attempted (IEEE nonaware tests), and 16 tests that do not cause the BSUN bit in the FPSR status byte (IEEE aware tests). The set of IEEE nonaware tests is best used:

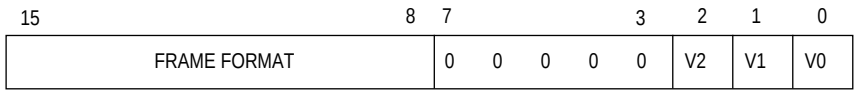
- When porting a program from a system that does not support the IEEE 754 standard to a conforming system, or
- When generating high-level language code that does not support IEEE floating-point concepts (i.e., the unordered condition).

An unordered condition occurs when one or both of the operands in a floating-point compare operation is a NAN. The inclusion of the unordered condition in floating-point branches destroys the familiar trichotomy relationship (greater than, equal, less than) that exists for integers. For example, the opposite of floating-point branch greater than (FBGT) is not floating-point branch less than or equal (FBLE). Rather, the opposite condition is floating-point branch not greater than (FBNGT). If the result of the previous instruction was unordered, FBNGT is true; whereas, both FBGT and FBLE would be false since unordered fails both of these tests. Compiler programmers should be particularly careful of the lack of trichotomy in the floating-point branches since it is common for compilers to invert the sense of conditions.

When using the IEEE nonaware tests, the BSUN bit and the NAN bit are set in the FPSR, unless the branch is an FBEQ or an FBNE. If the BSUN exception is enabled in the FPCR, an exception is taken. Therefore, the IEEE nonaware program may be interrupted if an unexpected condition occurs.

Compilers and programmers who are knowledgeable of the IEEE 754 standard should use the IEEE aware tests in programs that contain ordered and unordered conditions. Since the ordered or unordered attribute is explicitly included in the conditional test, the BSUN bit is not set in the FPSR EXC byte when the unordered condition occurs.

Table 6-9 summarizes the conditional mnemonics, definitions, equations, predicates, and whether the BSUN bit is set in the FPSR EXC byte for the 32 floating-point conditional tests. The equation column lists the combination of FPCC bits for each test in the form of an equation.



Frame Format
- $00—Null Frame
- $60—Idle Frame
- $E0—Exception Frame

V2–V0—Exception Vector
- 000—BSUN
- 001—INEX2 | INEX1
- 010—DZ
- 011—UNFL
- 100—OPERR
- 101—OVFL
- 110—SNAN
- 111—UNSUP

Figure 6-11. Status Word Contents

FSAVE on the MC68060 only generates one size frame (three long words), which creates a significant performance benefit, and one of these three frame types. An attempt to FRESTORE a frame format other than $00, $60, or $E0 results in a format error exception.

The format of the first long word of the MC68060 floating-point frame has changed from that of previous M68000 microprocessors. The MC68060 frame format (bits 15–8) is a consolidation of the version number and size format information (bits 31–16) on previous parts. In addition, on the MC68060, this information resides in the lower word of the long word while the upper word is used for the exception operand exponent in EXCP frames. Therefore, FRESTORE of a frame on an MC68060 created by FSAVE on a non-MC68060 microprocessor and FRESTORE of a frame on a non-MC68060 microprocessor created by FSAVE on an MC68060 will not guarantee a format error exception will be detected and thus must never be attempted.

When an FSAVE is executed, the floating-point frame reflects the state of the FPU at the time of the FSAVE. Internally, the FPU can be in the NULL, IDLE or EXCP states. Upon reset, the FPU is in the NULL state. In the NULL state, all floating-point registers contain nonsignaling NANs and the FPCR, FPSR, and FPIAR contain zeroes. The FPU remains in this state until the execution of an implemented floating-point instruction (except FSAVE). At this point, the FPU transitions from a NULL state to an IDLE state. An FRESTORE of NULL returns the FPU to the NULL state. The EXCP state is entered as a result of either a floating-point exception or an unsupported data type exception. V2–V0 indicates the exception types that are associated with the EXCP state.

An FSAVE instruction always clears the internal exception status bit at the completion of the FSAVE. An FRESTORE of EXCP may be used to place the FPU in the exception state.

The FRESTORE of an EXCP state is used in the M68060SP to provide to the user exception handler the illusion that the M68060SP handler never existed at all. The user exception handler is entered with the FPU in the proper exception state. The user

Clock 1 (C1)

The write cycle starts in C1. During C1, the processor places valid values on the address bus and transfer attributes. The processor asserts $\overline{TS}$ during C1 to indicate the beginning of a bus cycle. If not already asserted from a previous bus cycle, the $\overline{TIP}$ signal is also asserted at this time to indicate that a bus cycle is active.

The processor pre-conditions the data bus during C1 to improve AC timing. The process of pre-conditioning involves reinforcing the logic level that is already at the data pin. If the voltage level is originally zero volts, nothing is done; if the voltage level is 3.3 V, that voltage level is reinforced; however, if the voltage level is originally 5 V, the processor drives that data pin from 5 V down to 3.3 V. Note that no active logic change is done at this time. The actual logic level change is done in C2. This pre-conditioning affects operation primarily when using the processor in a 5-V system.

For user and supervisor mode accesses, which the corresponding memory unit translates, the UPAx signals are driven with the values from the U1 and U0 bits for the area. The TTx and TMx signals identify the specific access type. The R/$\overline{W}$ signal is driven low for a write cycle. $\overline{CIOUT}$ is asserted if the access is identified as noncachable or if the access references an alternate address space. Refer to **Section 4 Memory Management Unit** for information on the MC68060 and MC68LC060 memory units and **Appendix B MC68EC060** for information on the MC68EC060 memory unit.

Clock 2 (C2)

During C2, the processor negates $\overline{TS}$ and drives the appropriate bytes of the data bus with the data to be written. All other bytes are driven with undefined values. The selected device uses R/$\overline{W}$, SIZ1, SIZ0, A1, A0, or $\overline{BS3}$–$\overline{BS0}$, and $\overline{CIOUT}$ to register only the required information from the data bus. With the exception of R/$\overline{W}$ and $\overline{CIOUT}$, these signals also select any or all of the bytes (D31–D24, D23–D16, D15–D8, and D7–D0). If C2 is not a wait state (CW), then the selected peripheral device asserts the $\overline{TA}$ signal.

The MC68060 implements a special mode called the acknowledge termination ignore state capability to aid in high-frequency designs. In this mode, the processor begins the sampling of termination signals such as $\overline{TA}$ after a user-programmed number of BCLK rising edges has expired. The $\overline{SAS}$ signal is provided as a status output to indicate which BCLK rising edge the processor begins to sample the termination signals. If this mode is disabled, $\overline{SAS}$ is asserted during C2 to indicate that the processor immediately begins sampling the terminations signals. Refer to **7.14.1 Acknowledge Termination Ignore State Capability** for details on this special mode.

Assuming that the acknowledge termination ignore state capability is disabled, at the end of the C2, the processor samples the level of $\overline{TA}$, terminating the bus cycle if $\overline{TA}$ is asserted. If $\overline{TA}$ is not recognized asserted at the end of the clock cycle, the processor ignores the data and inserts a wait state instead of terminating the transfer. The processor continues to sample $\overline{TA}$ on successive rising edges of BCLK until $\overline{TA}$ is recognized asserted. The data bus then three-states and the bus cycle ends.

When the processor recognizes $\overline{TA}$ at the rising BCLK edge and terminates the bus cycle, $\overline{TIP}$ remains asserted if the processor is ready to begin another bus cycle. Otherwise, the

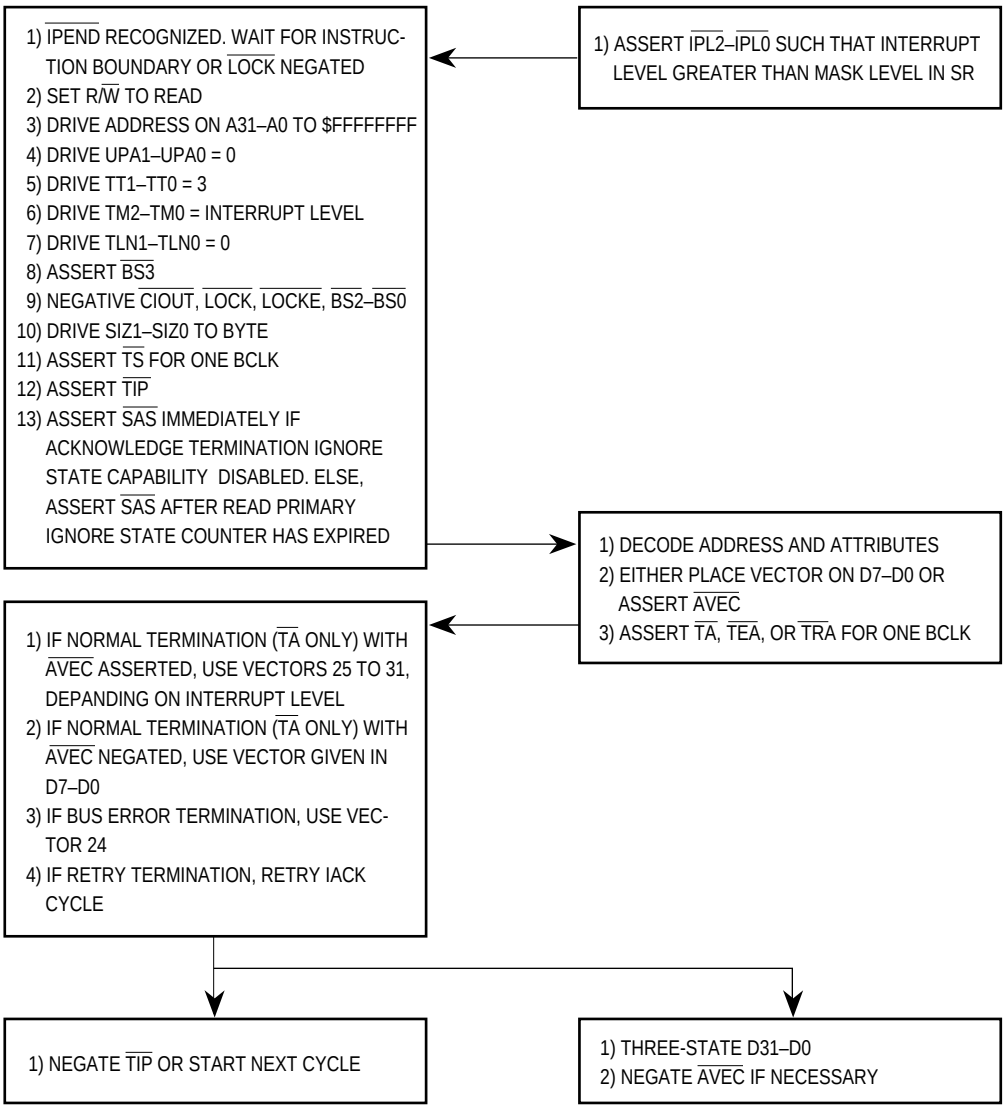


Figure 7-27. Interrupt Acknowledge Cycle Flowchart

and if TA and TEA are both asserted, the processor retries the cycle. If operating in native-MC68060 acknowledge termination mode, a retry is indicated by the assertion of TRA.

Note that the acknowledge termination ignore state capability is applicable to the interrupt acknowledge cycle. If enabled, TA, TEA, TRA, and other acknowledge termination signals are ignored for a user-programmed number of BCLK cycles.

7.8.2 Breakpoint Acknowledge Cycle

The execution of a BKPT instruction generates the breakpoint acknowledge cycle. An acknowledged access is a read bus cycle and is indicated with TT1, TT0 = $3, address A31–A0 = $00000000, and TM2–TM0 = $0. When the external device terminates the cycle with either TA or TEA, the processor takes an illegal instruction exception. A retry termination simply retries the breakpoint acknowledge cycle. Figure 7-30 and Figure 7-31 illustrate a flowchart and functional timing diagram for a breakpoint acknowledge bus cycle.

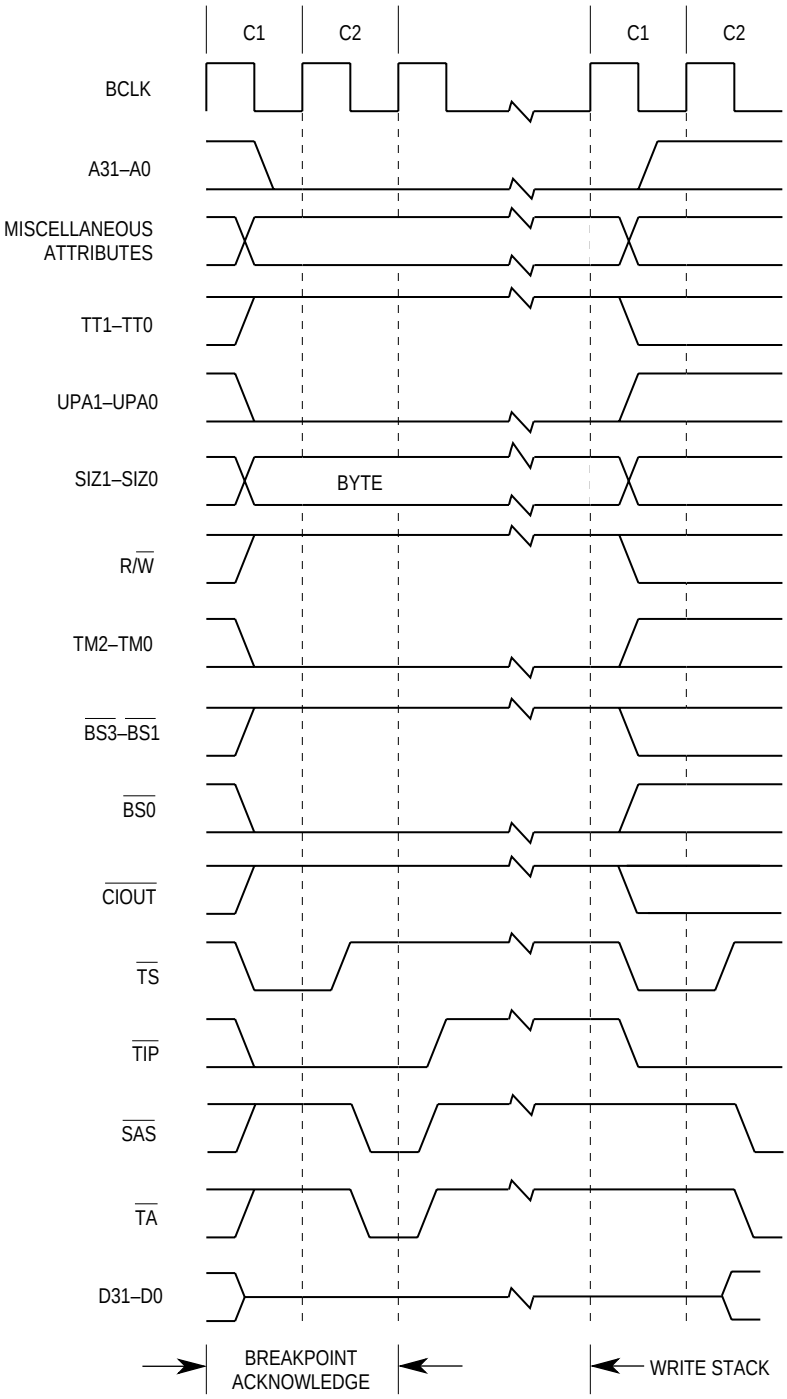


Figure 7-31. Breakpoint Interrupt Acknowledge Bus Cycle Timing

To exit the LPSTOP mode, the processor CLK must be restarted for at least eight CLK and two BCLK periods prior to asserting either the RSTI or generating an interrupt. It is imperative before asserting RSTI or generating the interrupt no alternate master activity be done until the processor begins exception processing for either the reset or interrupt. Additionally, the following control signals must be pulled-up or negated during this time: BB, TRA, TA, TEA, CLA, BGR, BG, SNOOP, AVEC, MDIS, CDIS, TCI, and TBI. The processor uses the PSTx encoding of $18 to indicate exception processing.

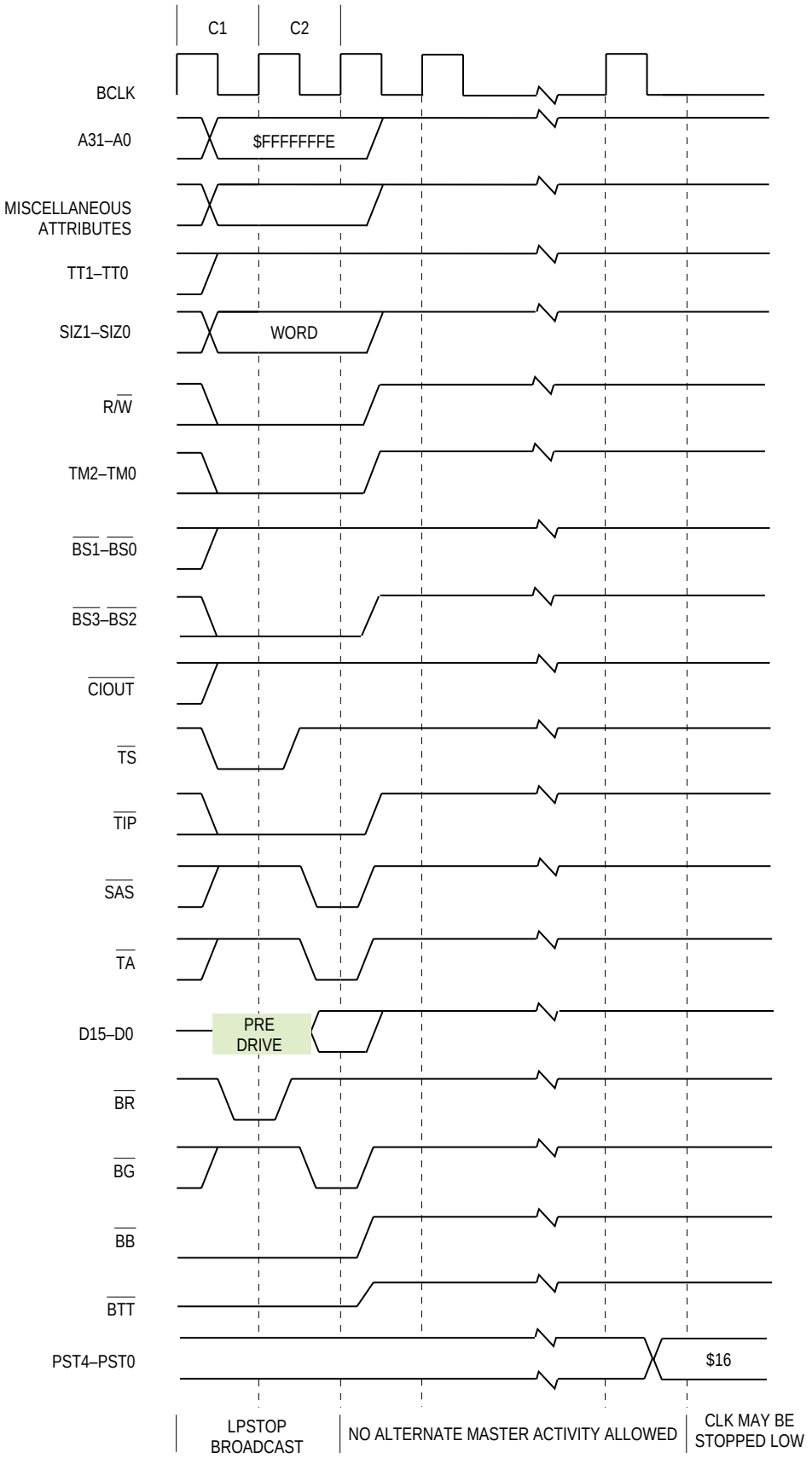
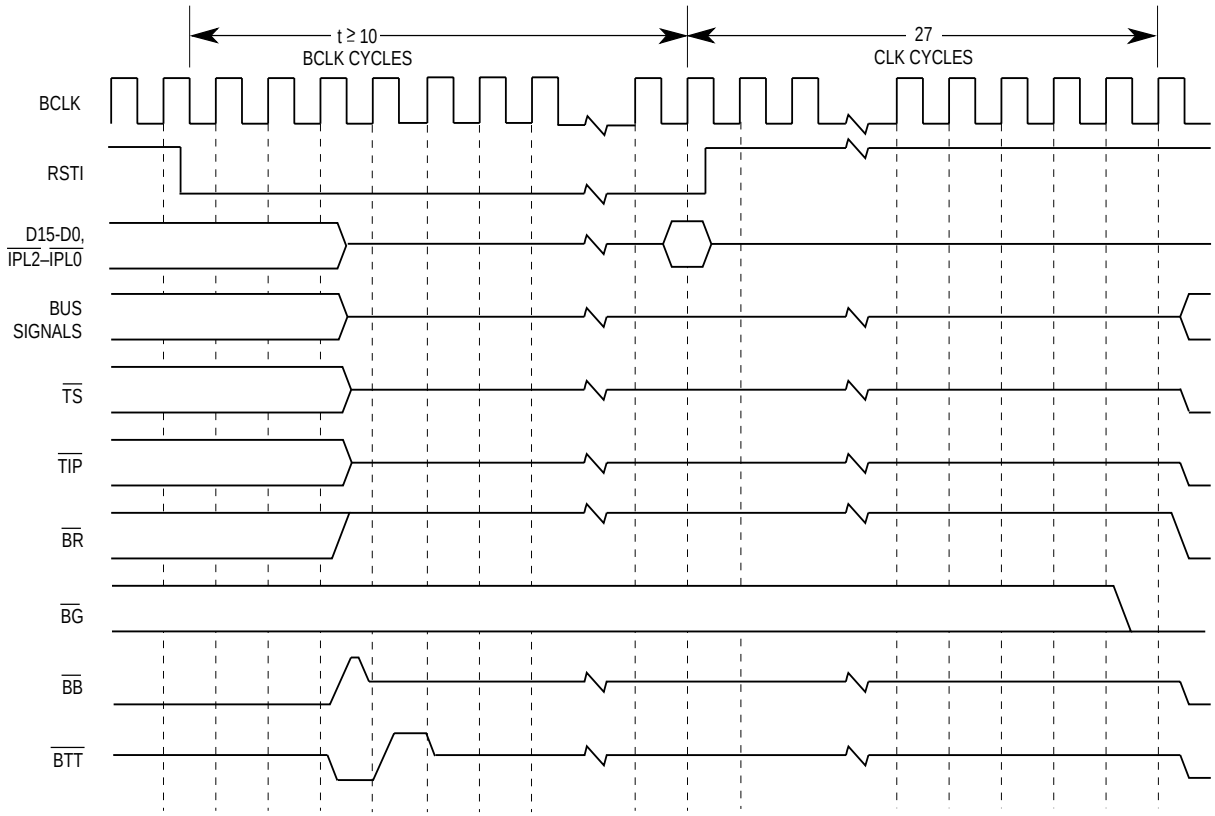


Figure 7-34. LPSTOP Broadcast Bus Cycle Timing, $\overline{BG}$ Asserted

For processor resets after the initial power-on reset, RSTI should be asserted for at least ten BCLK periods. Figure 7-49 illustrates timings associated with a reset when the processor is executing bus cycles. BB and TIP are negated before transitioning to a three-state level.



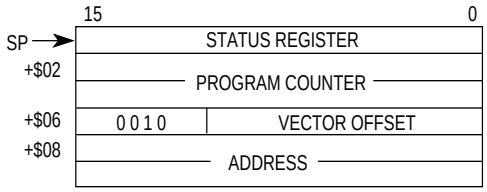
NOTE: For the processor to reset begin bus cycles after reset, BG must be asserted, TS must be negated or pulled up. BTT must be asserted (or BTT transition from asserted to negated) eventually to indicate an end to the alternate master's tenure.

Figure 7-49. Normal Reset Timing

Resetting the processor causes any bus cycle in progress to terminate as if TEA had been asserted. In addition, the processor initializes registers appropriately for a reset exception. **Section 8 Exception Processing** describes reset exception processing. When a RESET bus operation instruction is executed, the processor drives the reset out (RSTO) signal for 512 CLK cycles. In this case, the processor can be used to reset external devices in a system, and the internal registers of the processor are unaffected. The external devices connected to the RSTO signal are reset at the completion of the RESET instruction. An RSTI signal that is asserted to the processor during execution of a RESET instruction immediately resets the processor and causes the RSTO signal to negate. RSTO can be logically ANDed with the external signal driving RSTI to derive a system reset signal that is asserted for both an external processor reset and execution of a RESET instruction.

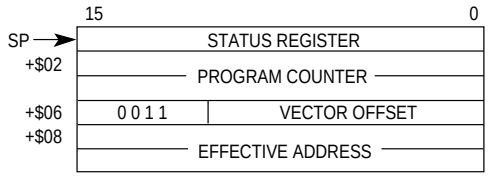
8.4.2 Six-Word Stack Frame (Format $2)

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

Stack Frames	Exception Types	Stacked PC Points To; Address Field Has
 SIX-WORD STACK FRAME–FORMAT $2	• CHK, CHK2 (Emulated), TRAPcc, FTRAPcc(Emulated), TRAPV, Trace, or Zero Divide • Unimplemented Floating-Point Instruction • Address Error	• Next Instruction; Address field has the address of the instruction that caused the exception. • Next Instruction; Address field has the calculated <ea> for the floating-point instruction. • Instruction that caused the address error; Address field has the branch target address with A0=0.

8.4.3 Floating-Point Post-Instruction Stack Frame (Format $3)

In this case, the processor restores the SR and PC values from the stack and increments the supervisor stack pointer by $C. If another floating-point post-instruction exception is pending, exception processing begins immediately for the new exception; otherwise, the processor resumes normal instruction execution.

Stack Frames	Exception Types	Stacked PC Points To; Effective Address Field
 FLOATING-POINT POST-INSTRUCTION STACK FRAME–FORMAT $3	• Floating-Point Post-Instruction	• Next Instruction; Effective Address field is the calculated effective address for the floating-point instruction.

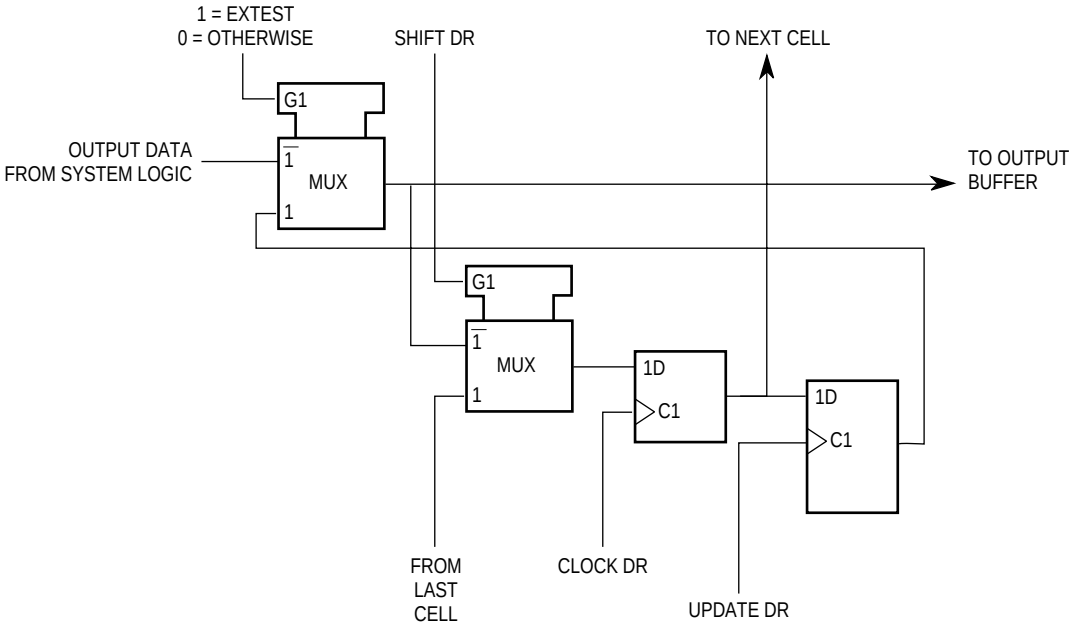


Figure 9-3. Output Pin Cell (O.Pin)

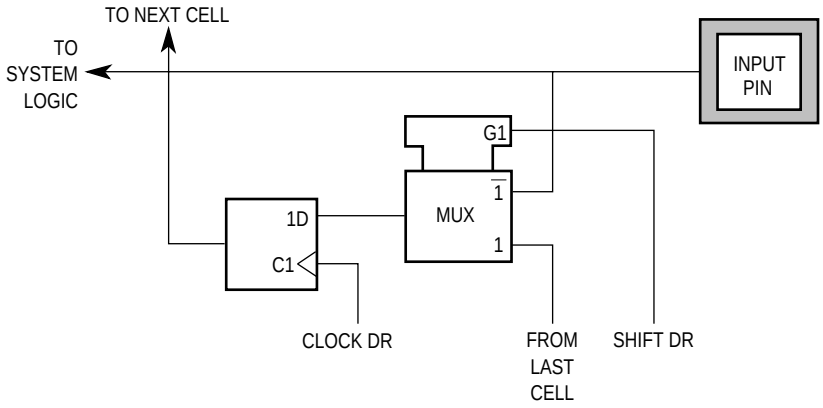


Figure 9-4. Observe-Only Input Pin Cell (I.Obs)

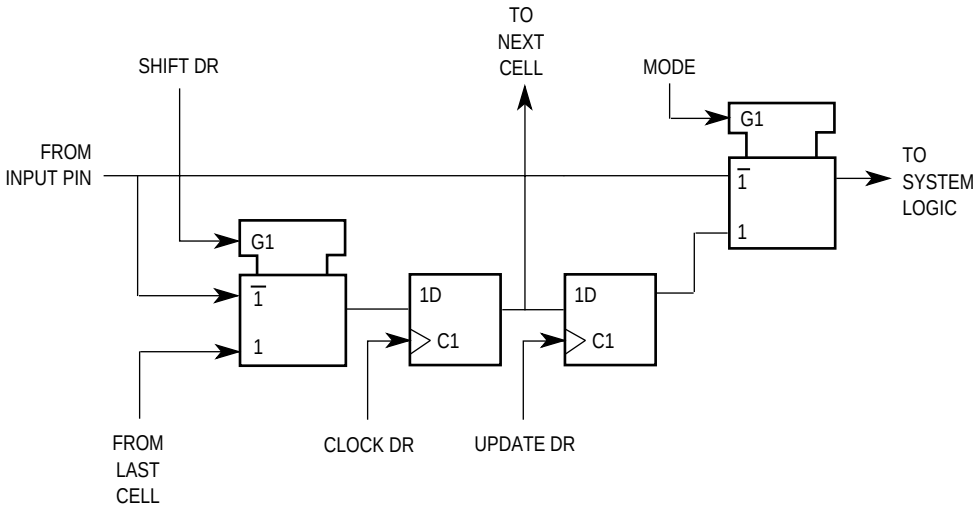


Figure 9-5. Input Pin Cell (I.Pin)

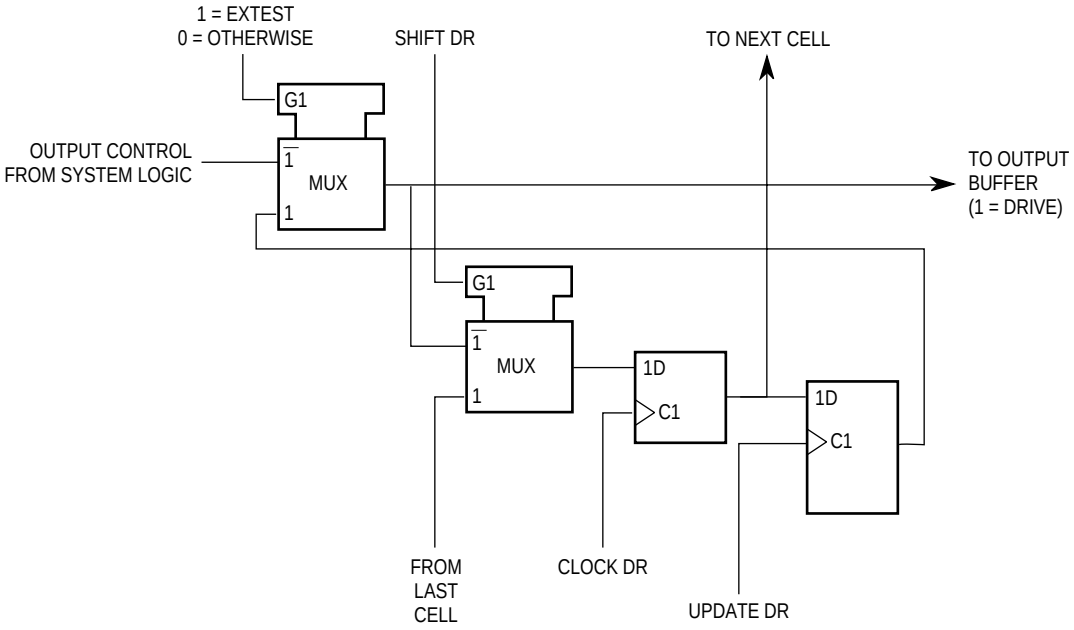


Figure 9-6. Output Control Cell (IO.Ctl)

Table 10-25. Floating-Point Instruction Execution Times (Continued)

Instruction	Effective Address, <ea>									
	FPn	Dn	(An)	(An)+	–(An)	(d16,An) (d16,PC)	(d8,An,Xi∗SF) (d8,PC,Xi∗SF)	(bd,An,XI∗SF) (bd,PC,XI∗SF)	(xxx).WL	#<imm>
FSGLDIV	37(0/0)	39(0/0)	37(1/0)	37(1/0)	37(1/0)	37(1/0)	38(1/0)	39(1/0)	38(1/0)	38(0/0)
FSGLMUL	3(0/0)	5(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)
FMOVEM ,FPx *	—	—	1+3n (3n/0)	1+3n (3n/0)	—	1+3n(3n/0)	2+3n(3n/0)	3+3n(3n/0)	2+3n(3n/0)	—
FMOVEM FPy, *	—	—	1+3n (0/3n)	—	1+3n (0/3n)	1+3n(0/3n)	2+3n(0/3n)	3+3n(0/3n)	2+3n(0/3n)	—
FMUL	3(0/0)	5(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)
FDMUL	3(0/0)	5(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)
FSMUL	3(0/0)	5(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)
FNEG	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	2(0/0)
FDNEG	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	2(0/0)
FSNEG	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	2(0/0)
FSUB	3(0/0)	5(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	5(1/0)	4(1/0)	3(0/0)
FDSUB	3(0/0)	5(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	5(1/0)	4(1/0)	3(0/0)
FSSUB	3(0/0)	5(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	5(1/0)	4(1/0)	3(0/0)
FTST	1(0/0)	3(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)
FSQRT	68(0/0)	70(0/0)	68(1/0)	68(1/0)	68(1/0)	68(1/0)	69(1/0)	70(1/0)	69(1/0)	69(0/0)
FSSQRT	68(0/0)	70(0/0)	68(1/0)	68(1/0)	68(1/0)	68(1/0)	69(1/0)	70(1/0)	69(1/0)	69(0/0)
FDSQRT	68(0/0)	70(0/0)	68(1/0)	68(1/0)	68(1/0)	68(1/0)	69(1/0)	70(1/0)	69(1/0)	69(0/0)
FSAVE	—	—	3(0/3)	—	—	—	—	—	—	—
FRE-STORE	—	—	6(3/0)	—	—	—	—	—	—	—
FMOVEM ,FPxR	—	—	7(n/0)	—	—	—	—	—	—	—
FMOVEM FPxR,	—	—	5(0/n)	—	—	—	—	—	—	—

NOTES:

“n” is the number of registers being moved.

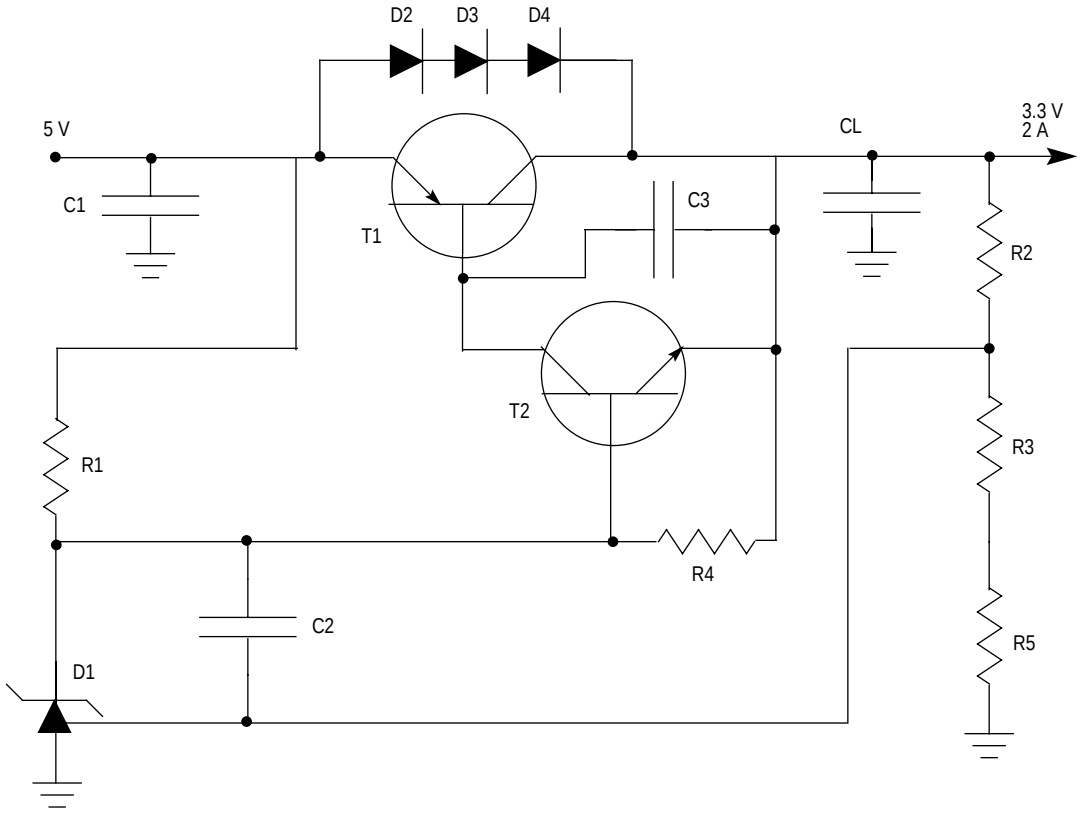
For all FPU operations, if the external operand format is byte, word, or long, add three cycles to the execution time.

*For all FPU operations except FMOVEM, if the external operand format is extended precision, add two cycles to the execution time.

Add 2(1/0) cycles to the (bd,An,Xi*SF) time for a memory indirect address.

Add 1(0/0) cycle if the <ea> specifies a double precision immediate operand.

11.2.1.1.1 Linear Voltage Regulator Solution. This solution uses a linear voltage regulator to supply 2 A at 3.3 V. This solution is inexpensive; however, conversion efficiency of only up to 65% can be achieved. Figure 11-1 shows a solution using power BJTs. This solution would be used primarily for applications that are cost sensitive, but not power sensitive. The suggested linear solution meets the 3.3 V± 5% MC68060 specifications.



T1	=	TIP32
T2	=	2N2222A
D1	=	TL431CLP
D2,D3,D4	=	IN4001
R1	=	220Ω
R2,R3,R4	=	1KΩ
R5	=	2.2KΩ
C1	=	47µF
C2	=	0.01µF
C3	=	0.033µF
CL	=	47µF

Figure 11-1. Linear Voltage Regulator Solution

11.2.1.1.2 Switching Regulator Solution. This solution uses switching regulators. Linear Technologies offers two parts, the LTC1147 and LTC1148 and MAXIM offers the MAX767. The main difference among these parts is a trade-off between price, part count, and conversion efficiency.

The LTC1147 solution is less expensive and has one fewer MOSFET than the LTC1148 solution. The LTC1148 is less than $5 in 1000 piece quantities, and the LTC1147 is less than $4 in 1000 piece quantities. In either of these solutions there are around 15 discrete

```
* _dmem_write():
* Writes to data memory while in supervisor mode.
* INPUTS:
*      a0 - supervisor source address
*      a1 - user destination address
*      d0 - number of bytes to write
*      $4(a6),bit5 - 1 = supervisor mode, 0 = user mode
* OUTPUTS:
*      d1 - 0 = success, !0 = failure

* _imem_read(), _dmem_read():
* Reads from data/instruction memory while in supervisor mode.
* INPUTS:
*      a0 - user source address
*      a1 - supervisor destination address
*      d0 - number of bytes to read
*      $4(a6),bit5 - 1 = supervisor mode, 0 = user mode
* OUTPUTS:
*      d1 - 0 = success, !0 = failure

* _dmem_read_byte(), _dmem_read_word(), _dmem_read_long():
* Read a data byte/word/long from user memory.
* INPUTS:
*      a0 - user source address
*      $4(a6),bit5 - 1 = supervisor mode, 0 = user mode
* OUTPUTS:
*      d0 - data byte/word/long in d0
*      d1 - 0 = success, !0 = failure

* _dmem_write_byte(), dmem_write_word(), dmem_write_long():
* Write a data byte/word/long to user memory.
* INPUTS:
*      a0 - user destination address
*      d0 - data byte/word/long in d0
*      $4(a6),bit5 - 1 = supervisor mode, 0 = user mode
* OUTPUTS:
*      d1 - 0 = success, !0 = failure

* _imem_read_word(), _imem_read_long():
* Read an instruction word/long from user memory.
* INPUTS:
*      a0 - user source address
*      $4(a6),bit5 - 1 = supervisor mode, 0 = user mode
* OUTPUTS:
*      d0 - instruction word/long in d0
*      d1 - 0 = success, !0 = failure
```

Figure C-11. Register Usage of {i,d}mem_{read,write}_{b,w,l}

C.4.2 Instructions Not Recommended

Emulated instructions that use the pre-decrement and post-increment addressing mode on the system stack must not contradict the basic definition of a stack. An operation that uses input operands below the stack (using the pre-decrement addressing mode) exhibits poor programming structure since the instruction is using a value before it has been defined. In addition, instructions that place a result on the stack using the post-increment addressing mode exhibit poor programming structure since an unexpected exception such as an interrupt or an unimplemented instruction exception would corrupt the result. The M68060SP does not handle these instruction cases properly, and unpredictable behavior will be exhibited when executing code of this type.

The M68060SP does not recover gracefully from these instruction cases because a performance penalty would be incurred to handle them properly. To avoid imposing this performance penalty on well-behaved systems, the task of avoiding these cases has been left outside the M68060SP. If the system absolutely requires that these cases be handled gracefully, the system software envelope can pre-filter these cases prior to entering the M68060SP. Table C-6 outlines these instructions.

Table C-6. Instructions Not Handled by the M68060SP

Instruction	Exception	Address Mode
div{u,s}.l (64-bit)	Integer Unimplemented	–(ssp), dr:dq
mul{u,s}.l (64-bit)	Integer Unimplemented	–(ssp), dr:dq
cas.{w,l} (mis)	Integer Unimplemented	dc:du,–(ssp)
cas.{w,l} (mis)	Integer Unimplemented	dc:du,(ssp)+
f<op>.p (all)	Floating-Point Unimplemented Data Type	–(ssp), fpn
f<op>.p	Floating-Point Unimplemented Data Type	fpn, (ssp)+
f<op>.{b,w,l,s,d,x}	Floating-Point Unimplemented Instruction	–(ssp), fpn
fs<cc>.b	Floating-Point Unimplemented Instruction	–(ssp)
fmovem.x	Floating-Point Unimplemented Instruction	–(ssp), dn
fmovem.x	Floating-Point Unimplemented Instruction	dn, (ssp)+
f<op>.x	Underflow,SNAN	fpn, (ssp)+
f<op>.{b,w,l}	Enabled OPERR	fpn, (ssp)+

C.5 INSTALLATION NOTES

This section provides a guide on how to install the M68060SP. The files provided in an M68060SP release are shown on Table C-7.

Table C-7. Files Provided in an M68060SP Release

File	Description
fpsp.sa	Full floating-point kernel module
pfpsp.sa	Partial floating-point kernel module
isp.sa	Integer unimplemented exception handler module
fplsp.sa	Floating-point library module
ilsp.sa	Integer library module
fskeleton.s	Sample call-outs needed by fpsp.sa and pfpsp.sa
iskeleton.s	Sample call-outs needed by isp.sa
os.s	Sample call-outs needed by fpsp.sa, pfpsp.sa and isp.sa
fpsp.doc	Release documentation for fpsp.sa and pfpsp.sa
isp.doc	Release documentation for isp.sa
fplsp.doc	Release documentation for fplsp.sa
ilsp.doc	Release documentation for ilsp.sa
fpsp.s	Source code of fpsp.sa
pfpsp.s	Source code of pfpsp.sa
isp.s	Source code of isp.sa
fplsp.s	Source code of fplsp.sa
ilsp.s	Source code of ilsp.sa

C.5.1 Installing the Library Modules

The integer and floating-point library modules (files ilsp.sa and fplsp.sa) require a very simple installation procedure. A symbolic label needs to be added to the module top so that calling routines can use this to reference the other entry-points supplied by these modules as an offset from the top of the module. It is the responsibility of the calling routine to enter the package through the proper offset relative to the top of the module.

C.5.2 Installing the Kernel Modules

The unimplemented integer instruction exception handler and full and/or partial floating-point kernel modules (files fpsp.sa, isp.sa, pfpsp.sa) may require additional steps. To aid in installation, three assembly language files are made available in the M68060SP release. These files contain the sample call-out routines and call-out dispatch tables for the unimplemented integer instruction exception handler module (iskeleton.s file), full or partial floating-point kernel modules (fskeleton.s file), and call-outs common to both (os.s file). When mod-

C.5.4 AESOP Electronic Bulletin Board

Motorola's AESOP electronic bulletin board contains the most current release of the M68060SP, as well as older releases of the M68060SP. The source code used to create the five pseudo-assembly files is provided for documentation purposes only and should not be used for generating a customized software package. Doing so would create versions of the package that is untested and unsupported by Motorola. Motorola will not create an assembly-to-assembly conversion software to provide a different assembler syntax than is already available. AESOP requires VT100 terminal emulation, 9600 Baud, 8 bits, no parity, and 1 stop bit. The modem supports V.32bis and V.42bis and MNP5 protocols. The kermit protocol is needed to download from AESOP. AESOP can be reached at (800)843-3451 or (512)891-3650.