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	66MHz
Co-Processors/DSP	-
RAM Controllers	-
Graphics Acceleration	No
Display & Interface Controllers	-
Ethernet	-
SATA	-
USB	-
Voltage - I/O	3.3V
Operating Temperature	-40°C ~ 70°C (TA)
Security Features	-
Package / Case	206-BPGA
Supplier Device Package	206-PGA (47.25x47.25)
Purchase URL	https://www.e-xfl.com/pro/item?MUrl=&PartUrl=mc68lc060brc66

Table 1-4. Notational Conventions (Continued)

Rn	Any Address or Data Register
Rx, Ry	Any source and destination registers, respectively.
Xn	Index Register—An, Dn, or suppressed.
Data Format and Type	
+ inf	Positive Infinity
<fmt>	Operand Data Format: Byte (B), Word (W), Long (L), Single (S), Double (D), Extended (X), or Packed (P).
B, W, L	Specifies a signed integer data type (twos complement) of byte, word, or long word.
D	Double-precision real data format (64 bits).
k	A twos complement signed integer (–64 to +17) specifying a number's format to be stored in the packed decimal format.
P	Packed BCD real data format (96 bits, 12 bytes).
S	Single-precision real data format (32 bits).
X	Extended-precision real data format (96 bits, 16 bits unused).
– inf	Negative Infinity
Subfields and Qualifiers	
#<xxx> or #<data>	Immediate data following the instruction word(s).
( )	Identifies an indirect address in a register.
[ ]	Identifies an indirect address in memory.
bd	Base Displacement
d_n	Displacement Value, n Bits Wide (example: d_{16} is a 16-bit displacement).
LSB	Least Significant Bit
LSW	Least Significant Word
MSB	Most Significant Bit
MSW	Most Significant Word
od	Outer Displacement
SCALE	A scale factor (1, 2, 4, or 8, for no-word, word, long-word, or quad-word scaling, respectively).
SIZE	The index register's size (W for word, L for long word).
{offset:width}	Bit field selection.
Register Codes	
*	General Case.
C	Carry Bit in CCR
cc	Condition Codes from CCR
FC	Function Code
N	Negative Bit in CCR
U	Undefined, Reserved for Motorola Use.
V	Overflow Bit in CCR
X	Extend Bit in CCR
Z	Zero Bit in CCR
—	Not Affected or Applicable.
Miscellaneous	
<ea>	Effective Address
<label>	Assemble Program Label
<list>	List of registers, for example D3–D0.
LB	Lower Bound
m	Bit m of an Operand
m–n	Bits m through n of Operand
UB	Upper Bound

Figure 4-1 illustrates the MMUs contained in the two memory units, one for instructions (supporting instruction prefetches) and one for data (supporting all other accesses). Each MMU contains a 64-entry ATC, two transparent translation registers (TTRs), and control logic. The ATCs hold recently used logical to physical address translations, cache mode and protection information, and whether or not the page has been written. The TTRs are used for defining the cache modes, enabling protection modes and defining user page attributes for large regions of untranslated address space. Each MMU also allows enabling a default cache mode, protection, and user page attributes for address regions not covered by the ATC or TTRs.

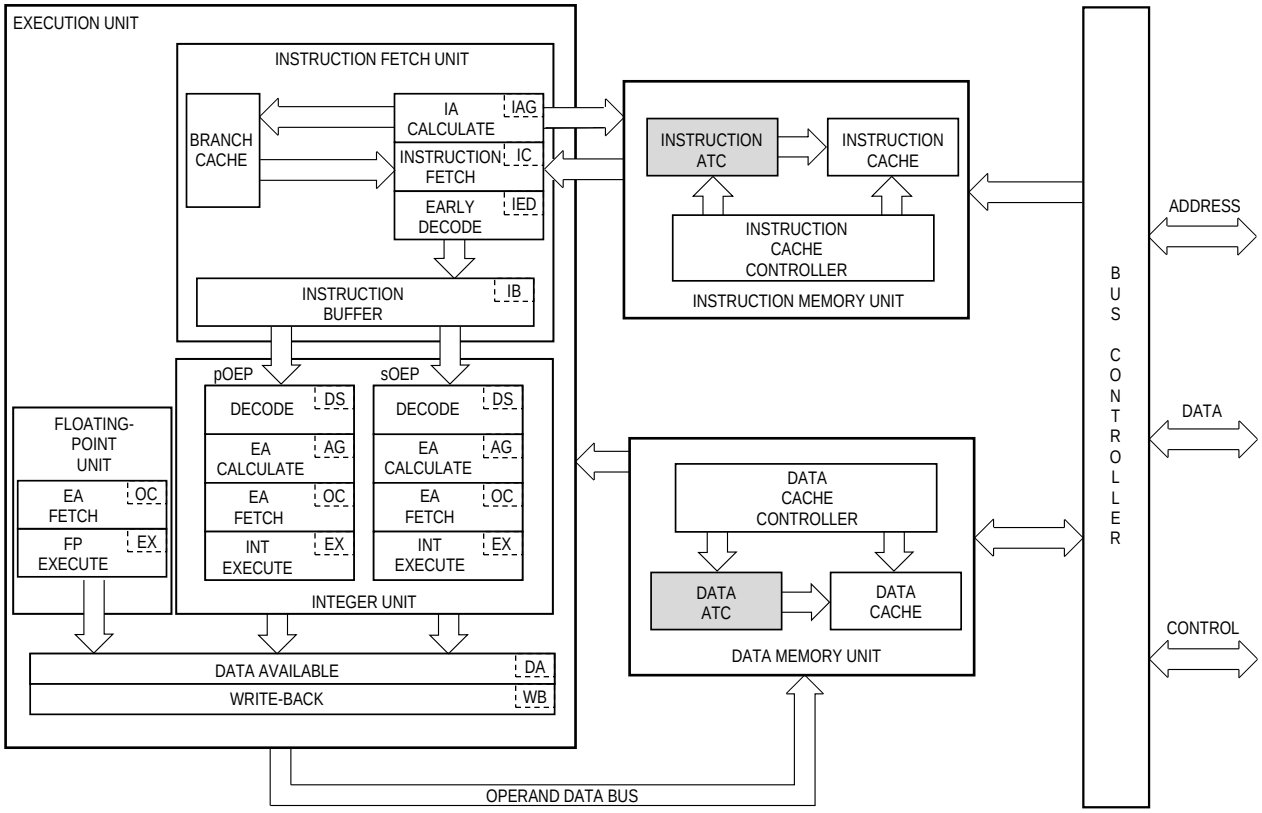


Figure 4-1. Memory Management Unit

One of the principal functions of the MMU is to provide logical to physical address translation using translation tables stored in memory. As an MMU receives a request from the corresponding pipe unit, its ATC is searched for the translation, using the upper logical address bits as a tag. If the translation is resident (or one of the TTRs hit causing transparent translation), the MMU provides the physical address for the corresponding cache lookup. If the translation is not in the ATC (and the TTRs miss), then a table search is done using translation tables stored in memory. When the translation is obtained, it is used for the cache lookup, and is placed in the ATC for future use. The table search is performed automatically by the MC68060 using on-chip logic.

updated before the MC68060 allows a page to be accessed. Table 4-1 lists the page descriptor update operations for each combination of U-bit, M-bit, write-protected, and read or write access type.

Table 4-1. Updating U-Bit and M-Bit for Page Descriptors

Previous Status		WP Bit	Access Type	Page Descriptor Update Operation	New Status	
U-Bit	M-Bit				U-Bit	M-Bit
0	0	X	Read	Locked RMW Access to Set U	1	0
0	1			Locked RMW Access to Set U	1	1
1	0			None	1	0
1	1			None	1	1
0	0	0	Write	Write to Set U and M	1	1
0	1			Write to Set U	1	1
1	0			Write to Set M	1	1
1	1			None	1	1
0	0	1		None	0	0
0	1			None	0	1
1	0			None	1	0
1	1			None	1	1

NOTE: WP indicates the accumulated write-protect status.

An alternate address space access is a special case that is immediately used as a physical address without translation. Because the MC68060 implements a merged instruction and data space, instruction address spaces (SFC/DFC = $6 or $2) using the MOVES instruction are converted into data references (SFC/DFC = $5 or $1). The data memory unit handles these translated accesses as normal data accesses. If the access fails due to an ATC fault or a physical bus error, the resulting access error stack frame contains the converted function code in the TM field for the faulted access. If the MOVES instruction is used to write instruction address space, then to maintain cache coherency, the corresponding addresses must be invalidated in the instruction cache. The SFC and DFC values and results for normal (TT = 0) and for MOVES (TT = 10) accesses are listed in Table 4-2.

Table 4-2. SFC and DFC Values

SFC/DFC Value	Results	
	TT	TM
000	10	000
001	00	001
010	00	001
011	10	011
100	10	100
101	00	101
110	00	101
111	10	111

4.2.6 Address Translation Protection

The MC68060 MMUs provide separate translation tables for supervisor and user address spaces. The translation tables contain both mapping and protection information. Each table and page descriptor includes a write-protect (W) bit that can be set to provide write protec-

Operands of locked instructions (CAS and TAS) and operand references while the lock bit in the bus control register is set which miss in the data cache do not allocate for reads or writes regardless of the caching mode, and therefore will bypass the cache. Locked instructions that hit in the data cache invalidate a matching valid entry or will push and invalidate a matching dirty entry. The locked operand access will then bypass the cache.

5.2 CACHE CONTROL REGISTER

The cache control register (CACR) is a 32-bit register which contains control information for the instruction and data caches. A MOVEC sets all of the bits in the CACR. A hardware reset clears the CACR, disabling both caches; however, reset does not affect the tags, state information, and data within the caches. The CACR is illustrated in Figure 5-5.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EDC	NAD	ESB	DPI	FOC	0	0	0	EBC	CABC	CUBC	0	0	0	0	0	EIC	NAI	FIC	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-5. Cache Control Register

EDC—Enable Data Cache

0 = Data cache is disabled.
1 = Data cache is enabled.

NAD—No Allocate Mode (Data Cache)

0 = Read and write misses will allocate in the data cache.
1 = Read and write misses will not allocate in the data cache.

ESB—Enable Store Buffer

0 = All writes to writethrough or cache-inhibited imprecise pages will bypass the store buffer and generate bus cycles directly.
1 = The four entry first-in-first-out (FIFO) store buffer to the MC68060 is enabled. This buffer is used to defer pending writes to writethrough or cache-inhibited imprecise pages to maximize performance.

Locked write accesses and accesses to cache-inhibited precise pages always bypass the store buffer.

DPI—Disable CPUSH Invalidation

0 = Each cache line is invalidated as it is pushed. Affects only the data cache.
1 = CPUSHed lines remain valid in the cache.

FOC—1/2 Cache Operation Mode Enable (Data Cache)

0 = The data cache operates in normal, full-cache mode.
1 = The data cache operates in 1/2-cache mode.

data cache is disabled for the second half of the operand. Internal accesses always bypass the instruction and data caches while $\overline{\text{CDIS}}$ is recognized, and the contents of the caches are unchanged. Disabling the caches with $\overline{\text{CDIS}}$ does not affect snoop operations. $\overline{\text{CDIS}}$ is intended primarily for use by in-circuit emulators to allow swapping between the tags and emulator memories.

The privileged CINV and CPUSH instructions support cache management, by selectively pushing and/or invalidating an individual cache line, a full page, or an entire cache, for either or both instruction and data caches. CINV allows selective invalidation of cache entries. The CPUSH instruction will either push and invalidate all matching lines, or push and leave the line valid, depending on the state of the DPI bit of the CACR register. (Note that only CPUSH instructions which specify the data cache are affected by the DPI bit. Since the instruction cache cannot have dirty data, a CPUSH specifying the instruction cache is interpreted as a CINV instruction.) Because of the size of the caches, pushing pages or an entire cache may incur a significant time penalty. Therefore, the CPUSH instruction may be interrupted to avoid large interrupt latencies. The state of the $\overline{\text{CDIS}}$ signal or the cache enable or no-allocate bits in the CACR does not affect the operation of CINV and CPUSH.

5.4 CACHING MODES

Every cache access has an associated caching mode from the MMU that determines how the cache handles the access. An access can be cachable in either the writethrough or copyback modes, or it can be cache inhibited in precise or imprecise modes. The CM field (from the transparent translation register (TTR) or MMU translation table page descriptor) corresponding to the logical address of the access normally specifies, on a page-by-page basis, one of these caching modes. When the cache is enabled and memory management is disabled, the default caching mode is writethrough.

The MMU provides the cache mode user page attributes (UPAx) and write protection for each access. This information may come from a TTR which matches or from the MMU translation tables via the ATC. If both the TTR and the ATC match the access, the TTR provides the information. If the paging MMU is disabled (TCR bit clear) and neither TTR matches, then the cache mode, UPAx, and write protection will be that which is specified in the default bits of the TCR. After reset, the defaults are writethrough cache mode, UPAx bits are zero, and all addresses may be written.

The TTRs and MMUs allow the defaults to be overridden. In addition, some instructions and integer unit operations perform data accesses that have an implicit caching mode associated with them. The following paragraphs discuss the different caching accesses and their related cache modes.

5.4.1 Cachable Accesses

If the CM field of a page descriptor, TTR, or default field of the TCR indicates writethrough or copyback, then the access is cachable. A read access to a writethrough or copyback page is read from the cache if matching data is found. Otherwise, the data is read from memory and used to update the cache. Since instruction cache accesses are always reads, the selection of writethrough or copyback modes do not affect them. The following paragraphs describe the writethrough and copyback modes in detail.

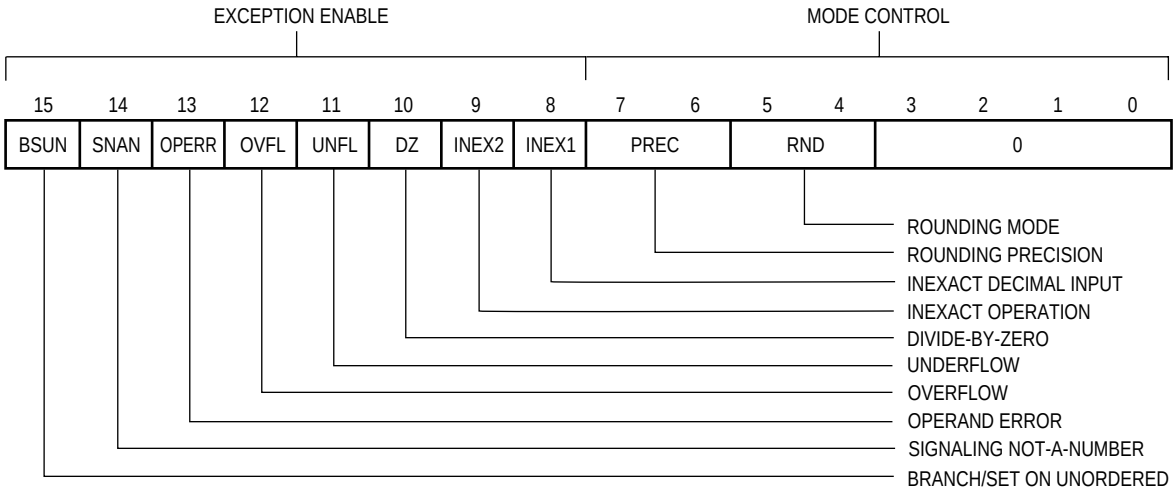


Figure 6-3. Floating-Point Control Register Format

The processor supports four rounding modes specified by the IEEE 754 standard. These modes are round to nearest (RN), round toward zero (RZ), round toward plus infinity (RP), and round toward minus infinity (RM). The RP and RM modes are directed rounding modes that are useful in interval arithmetic. Rounding is accomplished through the intermediate result. Single-precision results are rounded to a 24-bit boundary; double-precision results are rounded to a 53-bit boundary; and extended-precision results are rounded to a 64-bit boundary. Table 6-1 lists the encoding for the rounding mode. Table 6-2 lists the encoding for rounding precision.

Table 6-1. RND Encoding

Encoding		Rounding Mode
0	0	To Nearest (RN)
0	1	Toward Zero (RZ)
1	0	Toward Minus Infinity (RM)
1	1	Toward Plus Infinity (RP)

Table 6-2. PREC Encoding

Encoding		Rounding Precision
0	0	Extend (X)
0	1	Single (S)
1	0	Double (D)
1	1	Undefined

6.1.3 Floating-Point Status Register (FPSR)

The FPSR (see Figure 6-2) contains a floating-point condition code byte (FPCC), a quotient byte, a floating-point exception status byte (EXC), and a floating-point accrued exception byte (AEXC). The user can read or write to all defined bits in the FPSR. Execution of most floating-point instructions modifies this register. The reset function or a restore operation of the null state clears the FPSR. Floating-point conditional operations are not guaranteed if the FPSR is written directly, because the FPSR is only valid as a result of a floating-point instruction.

New AEXC Bit	= Old AEXC Bit	+	EXC Bits
IOP	= IOP	+	(BSUN + SNAN + OPERR)
OVFL	= OVFL	+	(OVFL)
UNFL	= UNFL	+	(UNFL • INEX2)
DZ	= DZ	+	(DZ)
INEX	= INEX	+	(INEX1 + INEX2 + OVFL)

6.1.4 Floating-Point Instruction Address Register (FPIAR)

For the subset of the floating-point instructions that generate exception traps, the FPU loads the 32-bit FPIAR with the logical address of the instruction before executing the instruction. Because the integer unit can execute instructions while the FPU executes floating-point instructions, the program counter (PC) value stacked by the MC68060 in response to a floating-point exception handler may not point to the offending instruction. Therefore, a floating-point exception handler uses the address in the FPIAR to locate a floating-point instruction that has caused an exception. Since the FMOVE to/from the FPCR, FPSR, or FPIAR and FMOVEM instructions cannot generate floating-point exceptions, these instructions do not modify the FPIAR. However, they can be used to read the FPIAR in an exception handler without changing the previous value. A reset or a restore operation of the null state clears the FPIAR.

6.2 FLOATING-POINT DATA FORMATS AND DATA TYPES

The M68000 floating-point model (MC68881, MC68882, MC68040, and MC68060) supports the following floating-point data formats: single precision, double precision, extended precision, and packed decimal. The M68000 floating-point model supports the following data types: normalized, zeros, infinities, unnormalized numbers, denormalized numbers, and NANs. The MC68060 supports part of the M68000 floating-point model in hardware. Table 6-3 lists the floating-point data formats and data types supported by the MC68060. Table 6-4 through Table 6-7 summarize the floating-point data formats and data types details.

Table 6-3. MC68060 FPU Data Formats and Data Types

Number Types	Data Formats						
	Single-Precision Real	Double-Precision Real	Extended-Precision Real	Packed-Decimal Real	Byte Integer	Word Integer	Long-Word Integer
Normalized	*	*	*	†	*	*	*
Zero	*	*	*	†	*	*	*
Infinity	*	*	*	†	—	—	—
NAN	*	*	*	†	—	—	—
Denormalized	†	†	†	†	—	—	—
Unnormalized	—	—	†	†	—	—	—

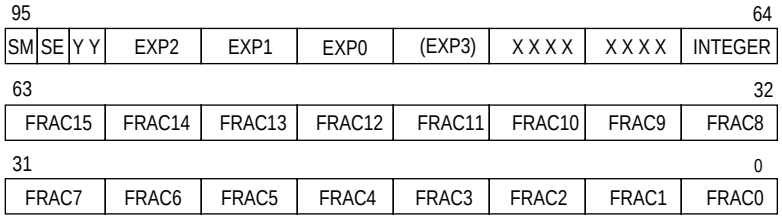
* Data Format/Type Supported by On-Chip MC68060 FPU Hardware

† Data Format/Type Supported by Software (M68060SP)

Table 6-6. Extended-Precision Real Format Summary (Continued)

Approximate Ranges	
Maximum Positive Normalized	1.2×10^{4932}
Minimum Positive Normalized	1.7×10^{-4932}
Minimum Positive Denormalized	1.7×10^{-4951}

Table 6-7. Packed Decimal Real Format Summary



Data Type	SM	SE	Y	Y	3-Digit Exponent	1-Digit Integer	16-Digit Fraction
±Infinity	0/1	1	1	1	$FFF	$XXXX	$00…00
±NAN	0/1	1	1	1	$FFF	$XXXX	Nonzero
±SNAN	0/1	1	1	1	$FFF	$XXXX	Nonzero
+Zero	0	0/1	X	X	$000–$999	$XXX0	$00…00
–Zero	1	0/1	X	X	$000–$999	$XXX0	$00…00
+In-Range	0	0/1	X	X	$000–$999	$XXX0–$XXX9	$00…01–$99…99
–In-Range	1	0/1	X	X	$000–$999	$XXX0–$XXX9	$00…01–$99…99

NOTE: EXP3 is generated only during an FMOVE OUT if the source is too large to be represented with a three-digit exponent. Otherwise, it is a don't care.

6.3 COMPUTATIONAL ACCURACY

Whenever an attempt is made to represent a real number in a binary format of finite precision, there is a possibility that the number can not be represented exactly. This is commonly referred to as a round-off error. Furthermore, when two inexact numbers are used in a calculation, the error present in each number is reflected, and possibly aggravated, in the result. All FPU calculations use an intermediate result. When the MC68060 performs an operation, the calculation is carried out using extended-precision inputs, and the intermediate result is calculated as if to produce infinite precision. After the calculation is complete, the intermediate result is rounded to the selected precision and stored in the destination.

The FPCR RND and PREC encodings (see Table 6-1 and Table 6-2) provide emulation for devices that only support single and double precision. By setting the rounding precision to single, the MC68060 will perform all calculations as if only 24 bits of precision were available for the result. Setting the rounding precision to double does the same to 53 bits of precision. The execution speed of all instructions is the same whether using single- or double-precision rounding. When using these two forced rounding precisions, the MC68060 produces the same results as any other device that conforms to the IEEE 754 standard, but does not support extended precision. The results are the same when performing the same operation in extended precision and storing the results in single- or double-precision format.

The snoop state is similar to the AM-explicit own state in that the MC68060 does not have ownership of the bus. The snoop state differs from the AM-explicit own state in that the MC68060 is in the process of performing an internal snoop operation because the processor has detected that $\overline{TS}$ and $\overline{SNOOP}$ are asserted and TT1 = 0. The snoop state always returns to the AM-explicit own state. The implicit ownership state indicates that the MC68060 owns the bus because $\overline{BG}$ is asserted to it. The processor, however, is not ready to begin a bus cycle, and keeps $\overline{BB}$ negated and the bus three-stated until an internal bus request occurs.

The MC68060 explicitly owns the bus when the bus is granted to it ($\overline{BG}$ asserted) and it has initiated at least one bus cycle. Until $\overline{BG}$ is negated, the processor retains explicit ownership of the bus whether or not active bus cycles are being executed. When the processor is ready to relinquish the bus, it goes through the end tenure state to indicate to all alternate masters that it is relinquishing the bus. During the end tenure state, $\overline{BTT}$ is asserted for one BCLK and is actively negated for the next BCLK prior to three-stating. While in this state, if $\overline{RSTI}$ is asserted, the processor proceeds to the end tenure state to inform other bus masters it is relinquishing the bus.

All alternate masters that reside in a system and use the MC68060-arbitration protocol must provide the same functionality as the MC68060 for proper system operation.

7.11.3 External Arbiter Considerations

The bus arbitration state diagrams for the MC68040-arbitration protocol and MC68060-arbitration protocol may be used to approximate the high level behavior of the processor. In either case, it is assumed that all $\overline{TS}$ signals in a system are tied together, all $\overline{BB}$ signals in a system are tied together and to a pullup resistor (MC68040-arbitration protocol), or all $\overline{BTT}$ signals in a system are tied together and to a pullup resistor (MC68060-arbitration protocol). Furthermore, unused $\overline{BB}$ or $\overline{BTT}$ pins must have separate pullup resistors.

If an alternate master loses bus ownership when it is in its implicit ownership state, the processor checks $\overline{TS}$. If $\overline{TS}$ is sampled asserted, the processor interprets this as the alternate master transitioning to its explicit ownership state, and it does not take over bus ownership. This operation is different from that of the MC68040, in that external arbiters are required to check for this boundary condition. However, in order for the processor to properly detect this boundary condition, it is imperative that the $\overline{TS}$ of all alternate bus masters be tied together with the processor's $\overline{TS}$ signal.

When using the MC68040-arbitration protocol, as with the $\overline{TS}$ signal, the $\overline{BB}$ of all alternate bus masters must be tied together to the processor's $\overline{BB}$ signal. Also, when an alternate master becomes bus master, it must assert $\overline{BB}$ if it initiates a bus cycle with the $\overline{TS}$ asserted.

The external arbiter design needs to include the function of $\overline{BR}$. For example, in certain cases associated with conditional branches, the MC68060 can assert $\overline{BR}$ to request the bus from an alternate bus master, then negate $\overline{BR}$ without using the bus, regardless of whether or not the external arbiter eventually asserts $\overline{BG}$. This situation happens when the MC68060 attempts to prefetch an instruction for a conditional branch. To achieve maximum performance, the processor may prefetch the instructions of the forward path for a conditional branch. If the branch prediction is incorrect and if the conditional branch results in a branch-not-taken, the previously issued branch-taken prefetch is then terminated since the prefetch

instruction is restarted, another table search is performed, and the instruction is executed successfully. If the access is not allowed, it is up to the system software designer to determine appropriate action.

For the physical bus error cases, as long as it is not one of the non-recoverable write cases, the exception handler must fix the page descriptor to point to a different physical memory, so that when the restart of the instruction occurs, that bus error does not recur.

It is important to note that the MC68060 performs table searches in hardware, and does not use the fetch table and page descriptors from the cache. The descriptor tables must be placed in non-cachable memory so that when the exception handler touches these descriptors, that the physical image in memory is updated properly.

The sixth step is to handle the default TTR cases. The default TTR is indicated if none of these bits are set: TTR, PTA, PTB, IL and PF. At this point, only the following cases are possible:

- WP = 1 (write protection violation detected by default TTR)
- RE = 1 (bus error on read)
- WE = 1 (bus error on write)

These cases may be handled similarly to step three. If the exception handler has gotten to this point, but none of the WP, RE and WE bits are set, and if the BPE bit is set and has been handled by the first step, then execute an RTE.

8.4.6 Bus Errors and Pending Memory Writes

The MC68060 processor contains two different write buffers for pending memory write operations: the store buffer and the push buffer. The store buffer is used to optimize performance by deferring bus write operations in write through and imprecise cache modes, and the push buffer holds displaced copyback mode cache lines and line write data for the MOVE16 instruction.

The push buffer holds a displaced cache line destined for memory until the cache-miss bus read access that caused the push completes. Imprecise cache modes (cachable write-through and copyback, and cache inhibited, imprecise) use the write buffers of the MC68060 to optimize system performance. Cache inhibited precise mode provides a precise exception model for MC68060 operation, not utilizing the write buffers (store or push).

When the MC68060 detects an exception condition, all instruction execution is aborted and the exception processing state is entered. Upon entering this state, the pipeline stalls until both the store and push buffers are empty before beginning exception processing. If a $\overline{\text{TEA}}$ signal termination occurs during a memory write cycle while emptying the store buffer, ‘a bus error $\overline{\text{TEA}}$ on store buffer’ is recorded and the buffer sequences through all the remaining, pending writes. However, if a $\overline{\text{TEA}}$ signal termination occurs during a memory write cycle while emptying the push buffer, ‘a bus error $\overline{\text{TEA}}$ on push buffer’ is recorded and the memory write operation is aborted immediately.

Table 9-3. Boundary Scan Bit Definitions (Continued)

Bit	Cell Type	Pin/Cell Name	Pin Type
179	I.Pin	D15	I/O
180	O.Pin	D14	I/O
181	I.Pin	D14	I/O
182	IO.Ctl	D15–D12 ena	—
183	O.Pin	D13	I/O
184	I.Pin	D13	I/O
185	O.Pin	D12	I/O
186	I.Pin	D12	I/O
187	O.Pin	D11	I/O
188	I.Pin	D11	I/O
189	O.Pin	D10	I/O
190	I.Pin	D10	I/O
191	IO.Ctl	D11–D8 ena	—
192	O.Pin	D9	I/O
193	I.Pin	D9	I/O
194	O.Pin	D8	I/O
195	I.Pin	D8	I/O
196	O.Pin	D7	I/O
197	I.Pin	D7	I/O
198	O.Pin	D6	I/O
199	I.Pin	D6	I/O
200	IO.Ctl	D7–D4 ena	—
201	O.Pin	D5	I/O
202	I.Pin	D5	I/O
203	O.Pin	D4	I/O
204	I.Pin	D4	I/O
205	O.Pin	D3	I/O
206	I.Pin	D3	I/O
207	O.Pin	D2	I/O
208	I.Pin	D2	I/O
209	IO.Ctl	D3–D0 ena	—
210	O.Pin	D1	I/O
211	I.Pin	D1	I/O
212	O.Pin	D0	I/O
213	I.Pin	D0	I/O

9.1.3.3 BYPASS REGISTER. An IEEE-1149.1-compliant bypass register has been included on the MC68060. This register is a single bit in depth when connected between TDI and TDO. The register element is in the shift path which operates during rising edges of TCK while the TAP state machine is in the shift-DR state or captures a default state of logic 0 during the rising edge of TCK while the TAP state machine is in the capture-DR state.

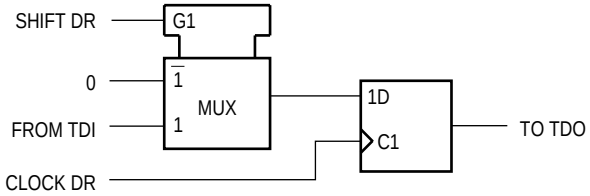


Figure 9-8. JTAG Bypass Register

9.1.4 Restrictions

The test logic is implemented using static logic design, and TCK can be stopped in either a high or low state without loss of data. The system logic, however, operates on a different system clock which is not synchronized to TCK internally. Any mixed operation requiring the use of 1149.1 test logic in conjunction with system functional logic that uses both clocks, must have coordination and synchronization of these clocks done externally to the MC68060.

The MC68060 also includes an internal instruction known as LPSTOP which can place the output pins in a high-impedance state, isolate the input pins from their internal signals, and stop the internal clock. Special care must be taken to ensure that the JTAG logic does not consume excess power during this mode if it is to be left inactive (see **9.1.5 Disabling the IEEE 1149.1 Standard Operation**).

9.1.5 Disabling the IEEE 1149.1 Standard Operation

There are two methods by which the device can be used without the IEEE 1149.1 test logic being active: 1) non-use of the JTAG test logic by either non-termination (disconnection) or intentional fixing of TAP logic values, and 2) intentional disabling of the JTAG test logic by assertion of the $\overline{\text{JTAG}}$ signal.

There are several considerations that must be addressed if the IEEE 1149.1 logic is not going to be used once the MC68060 is assembled onto a board. The prime consideration is to ensure that the IEEE 1149.1 test logic remains transparent and benign to the system logic during functional operation. This requires the minimum of either connecting the $\overline{\text{TRST}}$ pin to logic 0, or connecting the TCK clock pin to a clock source that will supply five rising edges and the falling edge after the fifth rising edge, to ensure that the part enters the test-logic-reset state. The recommended solution is to connect $\overline{\text{TRST}}$ to logic 0 since logic was included to ensure that unterminated or fixed-value terminated pins consume the least power during the LPSTOP functional state. Another consideration is that the TCK pin does not have a pullup as is required on the TMS, TDI, and $\overline{\text{TRST}}$ pins; therefore, it should not be left unterminated to preclude mid-level input values.

```
attribute INSTRUCTION_CAPTURE of MC68060: entity is "0101";
attribute INSTRUCTION_PRIVATE of MC68060:entity is "PRIVATE";
attribute REGISTER_ACCESS of MC68060:entity is
  "BOUNDARY (LPSAMPLE)";

attribute IDCODE_REGISTER of MC68060: entity is
  "0001" &        -- version
  "000001" &      -- design center
  "0000110000" &  -- sequence number
  "00000001110" & -- Motorola
  "1";            -- required by 1149.1

attribute BOUNDARY_CELLS of MC68060:entity is
"BC_1, BC_2, BC_4";

attribute BOUNDARY_LENGTH of MC68060:entity is 214;

attribute BOUNDARY_REGISTER of MC68060:entity is
--num cell  port    function safe  ccell dsval rslt
"0   (BC_1, D(0),   input,    X),   " &
"1   (BC_2, D(0),   output3,  X,      4,   0,   Z), " &
"2   (BC_1, D(1),   input,    X),   " &
"3   (BC_2, D(1),   output3,  X,      4,   0,   Z), " &
"4   (BC_2, *,      control,  0),   " & -- d[3:0]
"5   (BC_1, D(2),   input,    X),   " &
"6   (BC_2, D(2),   output3,  X,      4,   0,   Z), " &
"7   (BC_1, D(3),   input,    X),   " &
"8   (BC_2, D(3),   output3,  X,      4,   0,   Z), " &
"9   (BC_1, D(4),   input,    X),   " &
"10  (BC_2, D(4),   output3,  X,     13,   0,   Z), " &
"11  (BC_1, D(5),   input,    X),   " &
"12  (BC_2, D(5),   output3,  X,     13,   0,   Z), " &
"13  (BC_2, *,      control,  0),   " & -- d[7:4]
"14  (BC_1, D(6),   input,    X),   " &
"15  (BC_2, D(6),   output3,  X,     13,   0,   Z), " &
"16  (BC_1, D(7),   input,    X),   " &
"17  (BC_2, D(7),   output3,  X,     13,   0,   Z), " &
"18  (BC_1, D(8),   input,    X),   " &
"19  (BC_2, D(8),   output3,  X,     22,   0,   Z), " &
--num  cell port    function safe  ccell dsval rslt
"20  (BC_1, D(9),   input,    X),   " &
"21  (BC_2, D(9),   output3,  X,     22,   0,   Z), " &
"22  (BC_2, *,      control,  0),   " & -- d[11:8]
"23  (BC_1, D(10),  input,    X),   " &
"24  (BC_2, D(10),  output3,  X,     22,   0,   Z), " &
"25  (BC_1, D(11),  input,    X),   " &
"26  (BC_2, D(11),  output3,  X,     22,   0,   Z), " &
"27  (BC_1, D(12),  input,    X),   " &
"28  (BC_2, D(12),  output3,  X,     31,   0,   Z), " &
"29  (BC_1, D(13),  input,    X),   " &
"30  (BC_2, D(13),  output3,  X,     31,   0,   Z), " &
"31  (BC_2, *,      control,  0),   " & -- d[15:12]
"32  (BC_1, D(14),  input,    X),   " &
"33  (BC_2, D(14),  output3,  X,     31,   0,   Z), " &
```

```
"86  (BC_2, *,      control,  0),   " & -- a[3:2]
"87  (BC_1, A(3),   input,    X),   " &
"88  (BC_2, A(3),   output3,  X,     86,   0,   Z), " &
"89  (BC_1, A(2),   input,    X),   " &
"90  (BC_2, A(2),   output3,  X,     86,   0,   Z), " &
"91  (BC_1, CLA,    input,    X),   " &
"92  (BC_2, *,      control,  0),   " & -- a[1:0]
"93  (BC_1, A(1),   input,    X),   " &
"94  (BC_2, A(1),   output3,  X,     92,   0,   Z), " &
"95  (BC_1, A(0),   input,    X),   " &
"96  (BC_2, A(0),   output3,  X,     92,   0,   Z), " &
"97  (BC_2, TM(2),  output3,  X,     99,   0,   Z), " &
"98  (BC_2, TM(1),  output3,  X,     99,   0,   Z), " &
"99  (BC_2, *,      control,  0),   "   & -- tln(1),tm[2:0]
--num cell  port    function safe  ccell dsval rslt
"100 (BC_2, TM(0),  output3,  X,     99,   0,   Z), " &
"101 (BC_2, TLN(1), output3,  X,     99,   0,   Z), " &
"102 (BC_2, R_W,    output3,  X,    104,   0,   Z), " &
"103 (BC_2, SIZ(1), output3,  X,    104,   0,   Z), " &
"104 (BC_2, *,      control,  0),   "   &   -- tln(0),siz[1:0]
"105 (BC_2, SIZ(0), output3,  X,    104,   0,   Z), " &
"106 (BC_2, TLN(0), output3,  X,    104,   0,   Z), " &
"107 (BC_2, LOCKE,  output3,  X,    109,   0,   Z), " &
"108 (BC_2, LOCK,   output3,  X,    109,   0,   Z), " &
"109 (BC_2, *,      control,  0),   "   & -- lock,locke
"110 (BC_2, BR,     output3,  X,    127,   0,   Z), " &
"111 (BC_1, BB,     input,    X),   " &
"112 (BC_2, *,      control,  0),   " & -- bb
"113 (BC_2, BB,     output3,  X,    112,   0,   Z), " &
"114 (BC_1, SNOOP,  input,    X),   " &
"115 (BC_2, *,      control,  0),   " & -- tip
"116 (BC_2, TIP,    output3,  X,    115,   0,   Z), " &
"117 (BC_1, TS,     input,    X),   " &
"118 (BC_2, *,      control,  0),   " & -- ts
"119 (BC_2, TS,     output3,  X,    118,   0,   Z), " &
--num cell  port    function safe  ccell dsval rslt
"120 (BC_1, BTT,    input,    X),   " &
"121 (BC_2, *,      control,  0),   " & -- btt
"122 (BC_2, BTT,    output3,  X,    121,   0,   Z), " &
"123 (BC_2, *,      control,  0),   " & -- sas
"124 (BC_2, SAS,    output3,  X,    123,   0,   Z), " &
"125 (BC_2, PST(4), output3,  X,    127,   0,   Z), " &
"126 (BC_2, PST(3), output3,  X,    127,   0,   Z), " &
"127 (BC_2, *,      control,  0),   " & -- pst[4:0],br
"128 (BC_2, PST(2), output3,  X,    127,   0,   Z), " &
"129 (BC_2, PST(1), output3,  X,    127,   0,   Z), " &
"130 (BC_2, PST(0), output3,  X,    127,   0,   Z), " &
"131 (BC_1, TA,     input,    X),   " &
"132 (BC_1, TEA,    input,    X),   " &
"133 (BC_1, TRA,    input,    X),   " &
"134 (BC_1, BG,     input,    X),   " &
"135 (BC_1, BGR,    input,    X),   " &
"136 (BC_1, TBI,    input,    X),   " &
"137 (BC_1, AVEC,   input,    X),   " &
"138 (BC_1, TCI,    input,    X),   " &
```

Table 9-5. Command Summary

Command	Command Operation
$00	No operation
$01	Restart the processor This command restarts the processor after it had been halted by the execution of a HALT instruction (opcode = $4AC8), or receipt of the $02 (Halt the processor) command.This command must be issued only when the processor is halted.
$02	Halt the processor This command forces the processor to gracefully halt. The processor samples for halts once per instruction and if this command is present, the processor halts execution. The halted state is reflected in the PST encoding (PST = 11100).
$03	Enable the PULSE instruction to toggle non-pipelined mode This command enables the PULSE instruction (opcode = $4acc) to toggle the processor between the non-pipelined mode (allowing single-pipe dispatches) and normal pipeline mode. The PULSE instruction must be followed by a NOP to ensure proper operation. Refer to command $07 for details of non-pipelined mode, single-pipe dispatch operation. The $04 command negates the effect of this command. This command must be issued only when the processor is halted.
$04	Reset all non-pipelined modes This command forces the processor to normal pipeline operation and negates the effect of the $03, $06, and $07 commands. The $04 command negates the effect of this command. This command must be issued only when the processor is halted.
$05	Reserved
$06	Enable non-pipelined mode (allowing superscalar dispatches) This command forces the processor into a non-pipelined mode of operation, while allowing superscalar dispatches (if PCR0 = 1). After an instruction pair is dispatched into the primary and secondary OEPs, execution of the subsequent instructions is delayed until the original instruction(s) complete execution and the pipeline is synchronized. The synchronization requires the processor to be in a quiescent state with all pending memory cycles complete. This implies all write buffers (push and store) are empty. The $04 command negates the effect of this command. This command must be issued only when the processor is halted.
$07	Enable non-pipelined mode (allowing single-pipe dispatches) This command forces the processor into a non-pipelined mode of operation, while allowing instruction dispatches into the primary OEP only. After an instruction has been dispatched into the primary OEP, execution of the subsequent instructions is delayed until the original instruction complete execution and the pipeline is synchronized. The synchronization requires the processor to be in a quiescent state with all pending memory cycles complete. This implies all write buffers (push and store) are empty. The $04 command negates the effect of this command. This command must be issued only when the processor is halted.
$08	Perform CINVA IC operation This command causes a CINVAIC instruction to be inserted into the primary OEP. This command must be received while the processor is halted.
$09	Perform CINVA DC operation This command causes a CINVA DC instruction to be inserted into the primary OEP. This command must be received while the processor is halted.
$0A	Perform CPUSHA IC,DC operation This command causes a CPUSHA IC,DC instruction to be inserted into the primary OEP. This command must be issued only when the processor is halted.
$0B	Perform CPUSHA DC operation This command causes a CPUSHA DC instruction to be inserted into the primary OEP. This command must be issued only when the processor is halted.
$0C	Perform PFLUSHA operation This command causes a PFLUSHA instruction to be inserted into the primary OEP. This command must be received while the processor is halted.

Table 9-5. Command Summary (Continued)

Command	Command Operation
$0D	Force all the processor outputs to high-impedance state This command causes the processor to three-state all output pins and ignore all input pins. This command does not apply to the debug command interface pins. This forces the processor into a state where an emulator can generate system bus cycles by driving the appropriate pins. This command must be issued only when the processor is halted.
$0E	Release all the processor outputs from high-impedance state This command causes the processor to re-enable all output pins and begin sampling all the input pins. This command must be issued only when the processor is halted.
$0F	Negate the effects of the Disable commands This command causes the processor to disable the effects of the commands from $10 to $17.
$10	Disable instruction cache This command forces the processor to run with the instruction cache disabled. The $0F command negates the effect of this command. This command must be issued only when the processor is halted.
$11	Disable data cache This command forces the processor to run with the data cache disabled. The $0F command negates the effect of this command. This command must be issued only when the processor is halted.
$12	Disable instruction ATC This command forces the processor to run with the instruction ATC disabled. The $0F command negates the effect of this command. This command must be issued only when the processor is halted.
$13	Disable data ATC This command forces the processor to run with the data ATC disabled. The $0F command negates the effect of this command. This command must be issued only when the processor is halted.
$14	Disable write buffer This command forces the processor to run with the store buffers disabled. This command operation is equivalent to that provided by the cache control register (CACR) bit 29. The $0F command negates the effect of this command. This command must be issued only when the processor is halted.
$15	Disable branch cache This command forces the processor to run with the branch cache disabled. The $0F command negates the effect of this command. This command must be issued only when the processor is halted.
$16	Disable FPU This command forces the FPU-disabled operation. The $0F command negates the effect of this command. This command must be issued only when the processor is halted.
$17	Disable secondary OEP This command disables superscalar operation. The $0F command negates the effect of this command. This command must be issued only when the processor is halted.
$18	trace -> normal trace; bkpt -> normal breakpoint Both the trace and breakpoint exceptions operate normally. This command must be issued only when the processor is halted.
$19	trace -> normal trace; bkpt -> bkpt with emulator mode entry The trace exception operates normally. A breakpoint exception operates using vector offset $30, in addition, the processor enters the emulator mode. This command must be issued only when the processor is halted.
$1A	trace -> normal trace with emulator mode entry; bkpt -> normal breakpoint The breakpoint exception operates normally. A trace exception operates normally; in addition, the processor enters the emulator mode. This command must be issued only when the processor is halted.
$1B	trace -> normal trace with emulator mode entry; bkpt -> bkpt with emulator mode entry The trace exception operates normally. The breakpoint exception operates using vector offset $30. In addition, when either of these exceptions are taken, the processor enters the emulator mode. This command must be issued only when the processor is halted.
$1C–$1F	Generate an emulator interrupt Take an emulator interrupt exception.

If the first instruction of each pair is contained in the pOEP and the second in the sOEP, test 5 fails for both pairs. For the first example, the base resource required by the sOEP conflicts with the execute result generated by the pOEP instruction. In the second example, the index resource required by the sOEP conflicts with the execute result from the pOEP instruction.

10.1.6 Dispatch Test 6: No Register Conflicts on sOEP.IEE Resources

This test validates that the register resources of the sOEP.IEE (A, B) do not conflict with the execute result being generated by the instruction in the pOEP. Recall the most significant bit of the resource name is asserted to indicate a register resource. Thus, this test can be stated as:

```
test6 = 1        /* set test6 as okay
if (sOEP.A > 15) /* indicates a valid register
/* if the sOEP.A equals the pOEP's Execute_result, a conflict exists
    if ((sOEP.A = pOEP.Execute_result))
          test6 = 0/* test6 has register conflict
if (sOEP.B > 15) /* indicates a valid register
/* if the sOEP.B equals the pOEP's Execute_result, a conflict exists
    if ((sOEP.B = pOEP.Execute_result))
          test6 = 0/* test6 has register conflict
```

There are two very important exceptions to this rule involving the MOVE instruction:

1. If the primary OEP instruction is a simple “move long to register” (MOVE.L,Rx) and the destination register Rx is required as either the sOEP.A or sOEP.B input, the MC68060 bypasses the data as required and the test succeeds.
2. In the following sequence of instructions:

```
<op>.l,Dx
mov.l Dx,<mem>
```

the result of the pOEP instruction is needed as an input to the sOEP.IEE and the sOEP instruction is a move instruction. The destination operand for the memory write is sourced directly from the pOEP execute result and the test succeeds.

Consider the following examples:

```
asl.l &k,d0       Execute_result = d0
add.l d0,d1       A = d0
add.l <ea>,d1     Execute_result = d1
sub.l d0,d1       B = d1
mov.l <ea>,d0     Execute_result = d0
add.l d0,d1       A = d0
```

For all the examples, let the first instruction be loaded into the primary OEP and the second loaded into the secondary OEP.

In the first and second examples, the result of the pOEP instruction is required as an input to the sOEP.IEE. Since the pOEP instruction is not a simple MOVE operation, the test fails in each case.

In the third example, the result of the pOEP operation is needed as an input to the sOEP.IEE, but since the pOEP is executing the register-load MOVE instruction, the desti-

SECTION 11
APPLICATIONS INFORMATION

This section describes various applications topics relating to the MC68060.

11.1 GUIDELINES FOR PORTING SOFTWARE TO THE MC68060

The following paragraphs describe the issues involved in using the MC68060 in an existing MC68040 system from a software perspective. Although this section focuses on the MC68060, many of these items apply also to the MC68EC060 and MC68LC060.

11.1.1 User Code

The MC68060 is 100% user-mode compatible with the MC68040 when utilized with the MC68060 software package (M68060SP) provided by Motorola. The M68060SP is available free of charge. **Appendix C MC68060 Software Package** discusses the procedure for porting the M68060SP.

All user-mode instructions are handled in the M68060SP, except the “TRAPF #immediate” instruction, in which the immediate value is a valid branch opcode. Use of this construct results in a branch prediction error and an access error exception is taken. This exception is indicated by the BPE bit in the fault status long word (FSLW). Although this error is recoverable in the access error handler by flushing the branch cache, performance is compromised.

In addition, the CAS (misaligned operands) and CAS2 emulation may need special handling in the access error handler. Furthermore, CAS and CAS2 emulation must not be interrupted by level 7 interrupts to prevent data corruption. Refer to **Appendix C MC68060 Software Package** for additional information.

11.1.2 Supervisor Code

Unlike the MC68040, the MC68060 implements a single supervisor stack. System software that requires the use of two supervisor stacks can still do so, but with some software overhead.

The MC68060 aids in distinguishing between an interrupt exception and a non-interrupt exception by implementing the M-bit in the status register (SR). The MC68060 does not internally use the M-bit, but it is provided for system software. The MC68060 clears the M-bit of the SR when an interrupt exception is taken. Otherwise, it is up to the system software to set the M-bit and to examine it as needed. Also note, when the MC68060 takes an exception, a minimum of one instruction is always processed before a pending interrupt is taken.

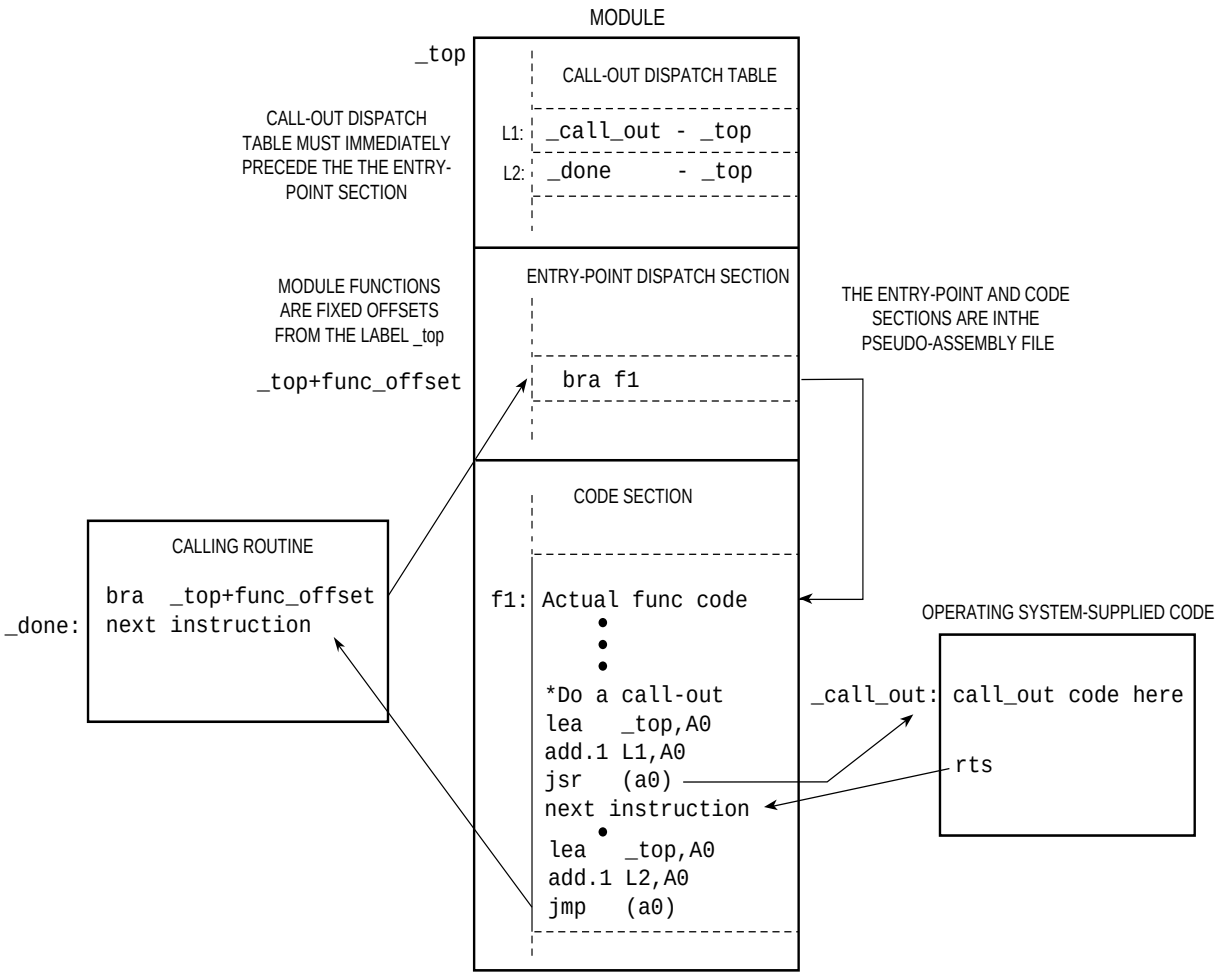


Figure C-3. Module Call-In, Call-Out Example

Table C-1 shows the code size of each module.

Table C-1. Call-Out Dispatch Table and Module Size

Module Name	Call-Out Dispatch Table Size	Entry-Point + Code Section Size	Total Module Size
Unimplemented Integer	128 bytes	8K-128 bytes	8K bytes
Unimplemented Integer Instruction Library	0 bytes	4K bytes	4K bytes
Full Floating-Point Kernel	128 bytes	56K-128 bytes	56K bytes
Floating-Point Library	0 bytes	34K bytes	34K bytes
Partial Floating-Point Kernel	128 bytes	35K-128 bytes	35K bytes

C.2 UNIMPLEMENTED INTEGER INSTRUCTIONS

The MC68060 left some low-use integer instructions unimplemented to streamline internal operations. This results in overall system performance improvement at the expense of software emulation of the unimplemented integer instructions. The M68060SP provides user object-code compatibility by providing the code needed to emulate these instructions via the unimplemented integer instruction exception. The M68060SP also provides a software

CPUSH Push and Possibly Invalidate Cache Line CPUSH

(MC68060, MC68LC060, MC68EC060)

Operation:

```
If Supervisor State, Then
   If Data Cache, Then
        Push Selected Dirty Data Cache Lines
        If DPI bit of CACR = 0, Then
             Invalidate Selected Cache Lines
        Endif
   Endif
   If Instruction Cache, Then
        Invalidate Selected Cache lines
   Endif
Endif
Else TRAP
```

Assembler Syntax:

CPUSHL<caches>,(An)
CPUSHP<caches>,(An)
CPUSHA<caches>

Where <caches> specifies the instruction cache, data cache, both caches, or neither cache.

Attributes: Unsized

Description: Pushes and possibly invalidates selected cache lines. The data cache, instruction cache, both caches, or neither cache can be specified. When the data cache is specified, the selected data cache lines are first pushed to memory (if they contain dirty data) and then invalidated if the DPI bit of the CACR is cleared. Otherwise, the selected data cache lines remain valid. Selected instruction cache lines are invalidated. The CACR is accessed via the MOVEC instruction.

Specific cache lines can be selected in three ways:

1. CPUSHL pushes and possibly invalidates the cache line (if any) matching the physical address in the specified address register.
2. CPUSHP pushes and possibly invalidates the cache lines (if any) matching the physical memory page in the specified address register. For example, if 4K-byte page sizes are selected and An contains $12345000, all cache lines matching page $12345000 are selected.
3. CPUSHA pushes and possibly invalidates all cache entries.