SECTION 3
COLDFIRE CORE

This section describes the organization of the ColdFire 5200 processor core and an overview of the program-visible registers. For detailed information on instructions, see the ColdFire Programmer's Reference Manual.

3.1 PROCESSOR PIPELINES

Figure 3-1 is a block diagram showing the processor pipelines of a ColdFire 5200 core.

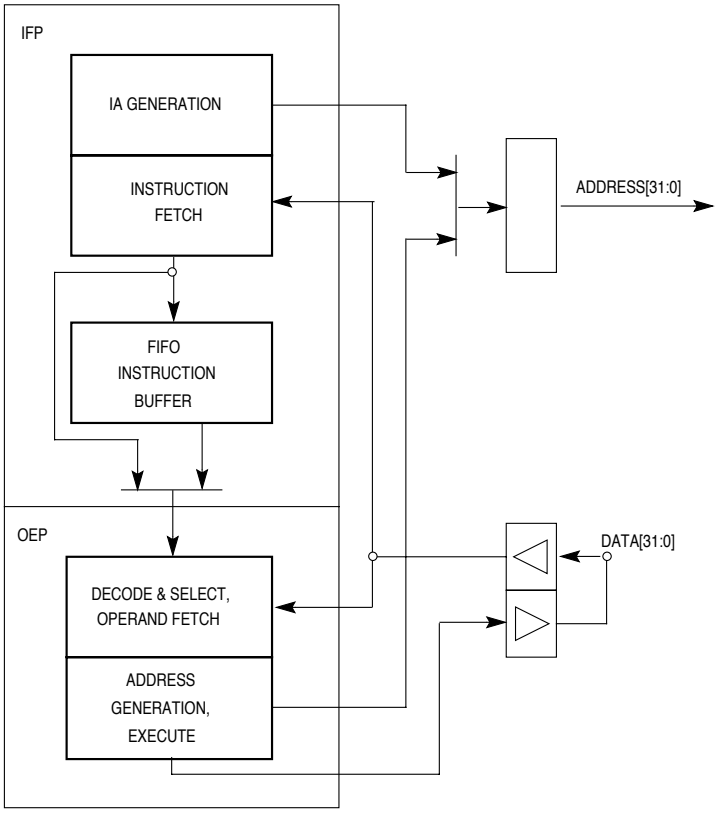


Figure 3-1. ColdFire Processor Core Pipelines

The processor core is comprised of two separate pipelines that are decoupled by an instruction buffer. The Instruction Fetch Pipeline (IFP) is responsible for instruction address generation and instruction fetch. The instruction buffer is a first-in-first-out (FIFO) buffer that holds prefetched instructions awaiting execution in the Operand Execution Pipeline (OEP). The OEP includes two pipeline stages. The first stage decodes instructions and selects operands (DSOC); the second stage (AGEX) performs instruction execution and calculates operand effective addresses, if needed.

3.2 PROCESSOR REGISTER DESCRIPTION

The following paragraphs describe the processor registers in the user and supervisor programming models. The appropriate programming model is selected based on the privilege level (user mode or supervisor mode) of the processor as defined by the S-bit of the status register.

3.2.1 User Programming Model

Figure 3-2 illustrates the user programming model. The model is the same as for M68000 Family microprocessors, consisting of the following registers:

- 16 general-purpose 32-bit registers (D0–D7, A0–A7)
- 32-bit program counter (PC)
- 8-bit condition code register (CCR)

3.2.1.1 DATA REGISTERS (D0–D7). Registers D0–D7 are used as data registers for bit (1 bit), byte (8-bit), word (16-bit) and longword (32-bit) operations and can also be used as index registers.

3.2.1.2 ADDRESS REGISTERS (A0–A6). These registers can be used as software stack pointers, index registers, or base address registers as well as for word and longword operations.

3.2.1.3 STACK POINTER (A7). ColdFire supports a single hardware stack pointer (A7) for explicit references or implicit ones during stacking for subroutine calls and returns and exception handling. The initial value of A7 is loaded from the reset exception vector, address $0. The same register is used for both user and supervisor mode as well as word and longword operations.

A subroutine call saves the PC on the stack and the return restores it from the stack. Both the PC and the SR are saved on the stack during the processing of exceptions and interrupts. The return from exception instruction restores the SR and PC values from the stack.

3.2.1.4 PROGRAM COUNTER. The PC contains the address of the currently executing instruction. During instruction execution and exception processing, the processor automatically increments the contents of the PC or places a new value in the PC, as appropriate. For some addressing modes, the PC can be used as a pointer for PC-relative operand addressing.

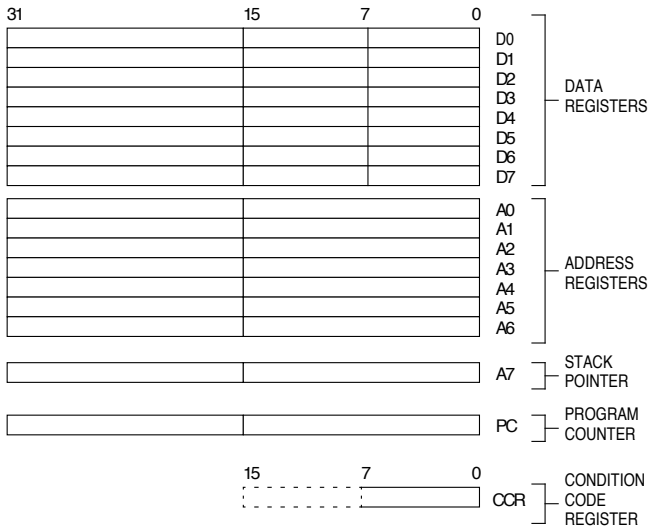


Figure 3-2. User Programming Model

3.2.1.5 CONDITION CODE REGISTER . The CCR is the least significant byte of the processor status register (SR). Bits 4–0 represent indicator flags based on results generated by processor operations. Bit 4, the extend bit (X-bit), is also used as an input operand during multiprecision arithmetic computations.

4	3	2	1	0
X	N	Z	V	C

X— extend condition code bit

N– negative condition code bit

Set if the most significant bit of the result is set; otherwise cleared

Z– zero condition code bit

Set if the result equals zero; otherwise cleared

V– overflow condition code bit

Set if an arithmetic overflow occurs implying that the result cannot be represented in the operand size; otherwise cleared

C– carry condition code bit

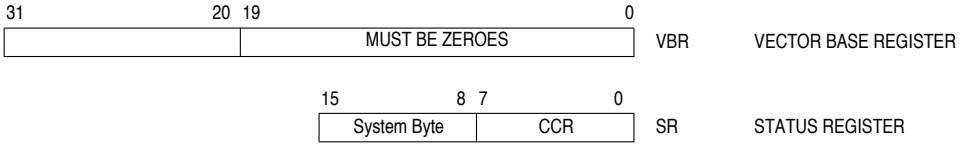
Set if a carryout of the operand MSB occurs for an addition, or if a borrow occurs in a subtraction; otherwise cleared

Set to the value of the C-bit for arithmetic operations; otherwise not affected.

3.2.2 Supervisor Programming Model

Only system programmers use the supervisor programming model to implement sensitive operating system functions, I/O control, and memory management. All accesses that affect the control features of ColdFire 5200 processors are in the supervisor programming model, which consists of the registers available to users as well as the following control registers:

- 16-bit status register (SR)
- 32-bit vector base register (VBR)



Supervisor Programming Model

Additional registers may be supported on a part basis.

The following paragraphs describe the supervisor programming model registers.

3.2.2.1 STATUS REGISTER. The SR stores the processor status and includes the CCR, the interrupt priority mask, and other control bits. In the supervisor mode, software can access the entire SR. In user mode, only the lower 8 bits are accessible (CCR). The control bits indicate the following states for the processor: trace mode (T-bit), supervisor or user mode (S-bit), and master or interrupt state (M).

SYSTEM BYTE								CONDITION CODE REGISTER (CCR)							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T	0	S	M	0	I[2:0]			0	0	0	X	N	Z	V	C

Status Register

T– trace enable

When set, the processor will perform a trace exception after every instruction.

S– supervisor / user state

Denotes whether the processor is in supervisor mode (S=1) or user mode (S=0).

M– master / interrupt state

This bit is cleared by an interrupt exception, and can be set by software during execution of the RTE or move to SR instructions.

I[2:0]– interrupt priority mask

Defines the current interrupt priority. Interrupt requests are inhibited for all priority levels less than or equal to the current priority, except the edge-sensitive level 7 request, which cannot be masked.

3.2.2.2 VECTOR BASE REGISTER (VBR). The VBR contains the base address of the exception vector table in memory. The displacement of an exception vector is added to the value in this register to access the vector table. The lower 20 bits of the VBR are not implemented by ColdFire 5200 processors; they are assumed to be zero, forcing the table to be aligned on a 1 Mbyte boundary.

The ColdFire 5200 processors provide a simplified exception processing model. The next section details the model.

3.3 EXCEPTION PROCESSING OVERVIEW

Exception processing for ColdFire processors is streamlined for performance. Differences from previous 68000 Family processors include:

- A simplified exception vector table
- Reduced relocation capabilities using the Vector Base Register (VBR)
- A fixed-length exception stack frame format
- Use of a single self-aligning system stack

ColdFire 5200 processors use an instruction restart exception model but do require more software support to recover from certain access errors. See subsection **3.5.1 Access Error Exception** for details.

Exception processing is comprised of four major steps and can be defined as the time from the detection of the fault condition until the fetch of the first handler instruction has been initiated.

1.The processor makes an internal copy of the SR and then enters supervisor mode by asserting the S-bit and disabling trace mode by negating the T-bit in the SR. The occurrence of an interrupt exception also forces the master/interrupt bit to be cleared and the interrupt priority mask to be set to the level of the current interrupt request.

2. The processor determines the exception vector number. For all faults except interrupts, the processor performs this calculation based on the exception type. For interrupts, the processor performs an interrupt-acknowledge (IACK) bus cycle to obtain the vector number from a peripheral device. The IACK cycle is mapped to a special acknowledge address space with the interrupt level encoded in the address.

3. The processor saves the current context by creating an exception stack frame on the system stack. ColdFire 5200 processors support a single stack pointer in the A7 address register; therefore, there is no notion of separate supervisor or user stack pointers. As a result, the exception stack frame is created at a 0-modulo-4 address on the top of the current system stack. Additionally, the processor uses a simplified fixed-length stack frame for all exceptions. The exception type determines whether the program counter placed in the exception stack frame defines the location of the faulting instruction (fault) or the address of the next instruction to be executed (next).

4. The processor calculates the address of the first instruction of the exception handler. By definition, the exception vector table is aligned on a 1 Mbyte boundary. This instruction address is generated by fetching an exception vector from the table located at the address defined in the vector base register. The index into the exception table is calculated as (4 x vector_number). Once the exception vector has been fetched, the contents of the vector determine the address of the first instruction of the desired handler. After the instruction fetch for the first opcode of the handler has been initiated, exception processing terminates and normal instruction processing continues in the exception handler.

ColdFire 5200 processors support a 1024-byte vector table aligned on any 1 Mbyte address boundary (see Table 3-1). The table contains 256 exception vectors where the first 64 are defined by Motorola and the remaining 192 are user-defined interrupt vectors.

Table 3-1. Exception Vector Assignments

VECTOR NUMBER(S)	VECTOR OFFSET (HEX)	STACKED PROGRAM COUNTER	ASSIGNMENT
0	$000	-	Initial stack pointer
1	$004	-	Initial program counter
2	$008	Fault	Access error
3	$00C	Fault	Address error
4	$010	Fault	Illegal instruction
5-7	$014-$01C	-	Reserved
8	$020	Fault	Privilege violation
9	$024	Next	Trace
10	$028	Fault	Unimplemented line-a opcode
11	$02C	Fault	Unimplemented line-f opcode
12	$030	Next	Debug interrupt
13	$034	-	Reserved
14	$038	Fault	Format error
15	$03C	Next	Uninitialized interrupt
16-23	$040-$05C	-	Reserved
24	$060	Next	Spurious interrupt
25-31	$064-$07C	Next	Level 1-7 autovectored interrupts
32-47	$080-$0BC	Next	Trap # 0-15 instructions
48-63	$0C0-$0FC	-	Reserved
64-255	$100-$3FC	Next	User-defined interrupts

"Fault" refers to the PC of the instruction that caused the exception
"Next" refers to the PC of the next instruction that follows the instruction that caused the fault.

ColdFire 5200 processors inhibit sampling for interrupts during the first instruction of all exception handlers. This allows any handler to effectively disable interrupts, if necessary, by raising the interrupt mask level contained in the status register.

3.4 EXCEPTION STACK FRAME DEFINITION

The exception stack frame is shown in Figure 3-3. The first longword of the exception stack frame contains the 16-bit format/vector word (F/V) and the 16-bit status register, and the second longword contains the 32-bit program counter address.

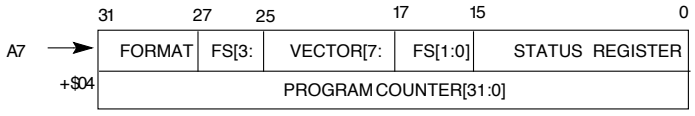


Figure 3-3. Exception Stack Frame Form

The 16-bit format/vector word contains 3 unique fields:

- A 4-bit format field at the top of the system stack is always written with a value of {4,5,6,7} by the processor indicating a two-longword frame format. See Table 3-2.

Table 3-2. Format Field Encodings

ORIGINAL A7 @ TIME OF EXCEPTION, BITS 1:0	A7 @ 1ST INSTRUCTION OF HANDLER	FORMAT FIELD
00	Original A7 - 8	4
01	Original A7 - 9	5
10	Original A7 - 10	6
11	Original A7 - 11	7

- A 4-bit fault status field, FS[3:0], at the top of the system stack. This field is defined for access and address errors only and written as zeros for all other types of exceptions. See Table 3-3.

Table 3-3. Fault Status Encodings

FS[3:0]	DEFINITION
00xx	Reserved
0100	Error on instruction fetch
0101	Reserved
011x	Reserved
1000	Error on operand write
1001	Attempted write to write-protected space
101x	Reserved
1100	Error on operand read
1101	Reserved
111x	Reserved

- The 8-bit vector number, vector[7:0], defines the exception type and is calculated by the processor for all internal faults and represents the value supplied by the peripheral in the case of an interrupt. Refer to Table 3-1.

3.5 PROCESSOR EXCEPTIONS

3.5.1 Access Error Exception

An Access Error Exception vector number $2 occurs when a bus cycle terminates with an error condition. The exact processor response to an access error depends on the type of memory reference being performed.

For an instruction fetch, the processor postpones the error reporting until the faulted reference is needed by an instruction for execution. Therefore, faults that occur during instruction prefetches that are then followed by a change of instruction flow will not generate an exception. When the processor attempts to execute an instruction with a faulted opword and/or extension words, the access error will be signaled and the instruction aborted. For this type of exception, the programming model has not been altered by the instruction generating the access error.

If the access error occurs on an operand read, the processor immediately aborts the current instruction's execution and initiates exception processing. In this situation, any address register updates attributable to the auto-addressing modes, {e.g., (An)+,-(An)}, will already have been performed. So, the programming model contains the updated An value. In addition, if an access error occurs during the execution of a MOVEM instruction loading from memory, any registers already updated before the fault occurs will contain the operands from memory.

The ColdFire processor uses an imprecise reporting mechanism for access errors on write operations. Since the actual write cycle may be decoupled from the processor's issuing of the operation, the signaling of an access error appears to be decoupled from the instruction that generated the write. Accordingly, the PC contained in the exception stack frame merely represents the location in the program when the access error was signaled, not when the offending instruction was executed. All programming model updates associated with the write instruction are completed. The NOP instruction can collect access errors for writes. This instruction delays its execution until all previous operations to internal memory resources, including all pending write operations, are complete. If any previous write terminates with an access error, it is guaranteed to be reported on the NOP instruction.

3.5.2 Address Error Exception

Any attempted execution transferring control to an odd instruction address (i.e., if bit 0 of the target address is set) results in an address error exception, vector number $3.

Any attempted use of a word-sized index register (Xn.w) or a scale factor of 8 on an indexed effective addressing mode generates an address error as does an attempted execution of a full-format indexed addressing mode.

3.5.3 Illegal Instruction Exception

Any attempted execution of the $0000 and the $4AFC opwords generates an illegal instruction exception, vector number $4. Additionally, any attempted execution of any line A and most line F opcode generates their unique exception types, vector numbers 10 and 11, respectively. ColdFire 5200 processors do not provide illegal instruction detection on the extension words on any instruction, including MOVEC. If any other nonsupported opcode is executed, the resulting operation is undefined.

3.5.4 Privilege Violation Exception

The attempted execution of a supervisor mode instruction while in user mode generates a privilege violation exception, vector number $8. See the ColdFire Programmer's Reference Manual for lists of supervisor- and user-mode instructions.

3.5.5 Trace Exception

To aid in program development, the ColdFire 5200 processors provide an instruction-by-instruction tracing capability. While in trace mode, indicated by the assertion of the T-bit in the status register (SR[15] = 1), the completion of an instruction execution signals a trace exception, vector number $9. This functionality allows a debugger to monitor program execution.

The single exception to this definition is the STOP instruction. When the STOP opcode is executed, the processor core waits until an unmasked interrupt request is asserted, then aborts the pipeline and initiates interrupt exception processing.

Because ColdFire processors do not support any hardware stacking of multiple exceptions, it is the responsibility of the operating system to check for trace mode after processing other exception types. As an example, consider the execution of a TRAP instruction while in trace mode. The processor will initiate the TRAP exception and then pass control to the corresponding handler. If the system requires that a trace exception be processed, it is the responsibility of the TRAP exception handler to check for this condition (SR[15] in the exception stack frame asserted) and pass control to the trace handler before returning from the original exception.

3.5.6 Debug Interrupt

This special type of program interrupt is discussed in detail in **Debug Support**. This exception is generated in response to a hardware breakpoint register trigger. The processor does not generate an IACK cycle but rather calculates the vector number internally (vector number 12).

3.5.7 RTE and Format Error Exceptions

When an RTE instruction is executed, the processor first examines the 4-bit format field in the exception stack frame on the stack to validate the frame type. For a ColdFire 5200 processor, any attempted execution of an RTE where the format is not equal to {4,5,6,7} generates a format error, vector number $E. The exception stack frame for the format error is created without disturbing the original RTE frame and the stacked PC pointing to the RTE instruction.

The selection of the format value provides some limited debug support for porting code from 68000 applications. On 680x0 family processors, the SR was located at the top of the stack. On those processors, bit[30] of the longword addressed by the system stack pointer is typically zero. Thus, if an RTE is attempted with a 680X0-type exception stack frame, the 5206 will generate a format exception.

If the format field defines a valid type, the processor: (1) reloads the SR operand, (2) fetches the second longword operand, PC, (3) adjusts the stack pointer by adding the format value to the auto-incremented address after the fetch of the first longword, and then (4) transfers control to the instruction address defined by the PC (fetched in step2) second longword operand within the stack frame.

3.5.8 TRAP Instruction Exceptions

The TRAP #n instruction always forces an exception as part of its execution and is useful for implementing system calls.

3.5.9 Interrupt Exception

The interrupt exception processing, with interrupt recognition and vector fetching, includes uninitialized and spurious interrupts as well as those where the requesting device supplies

the 8-bit interrupt vector. Autovectoring may optionally be supported through the System Integration module (SIM). Refer to the SIM section to see if this is supported on the MCF5206.

3.5.10 Fault-on-Fault Halt

If a ColdFire 5200 processor encounters any type of fault during the exception processing of another fault, the processor immediately halts execution with the catastrophic “fault-on-fault” condition. A reset is required to force the processor to exit this halted state.

3.5.11 Reset Exception

Asserting the reset input signal to the processor causes a reset exception, vector number $0. The reset exception has the highest priority of any exception; it provides for system initialization and recovery from catastrophic failure. Reset also aborts any processing in progress when the reset input is recognized, and the aborted processing cannot be recovered.

The reset exception places the processor in the supervisor mode by setting the S-bit and disables tracing by clearing the T-bit in the SR. This exception also clears the M-bit and sets the processor’s interrupt priority mask in the SR to the highest level (level 7). Next, the VBR is initialized to zero ($00000000). The control registers specifying the operation of any memories (e.g., cache and/or RAM modules) connected directly to the processor are disabled.

Note

Other implementation-specific supervisor registers are also affected. Refer to each of the modules in this user’s manual for details on these registers.

Once the processor is granted the bus and it does not detect any other alternate masters taking the bus, the core then performs two longword read bus cycles. The first longword at address 0 is loaded into the stack pointer and the second longword at address 4 is loaded into the program counter. After the initial instruction is fetched from memory, program execution begins at the address in the PC. If an access error or address error occurs before the first instruction is executed, the processor enters the fault-on-fault halted state.

3.6 INSTRUCTION EXECUTION TIMING

This section presents ColdFire 5200 Family processor instruction execution times in terms of processor core clock cycles. The number of operand references for each instruction is enclosed in parentheses following the number of clock cycles. Each timing entry is presented as **C**(r/w) where:

- **C** - number of processor clock cycles, including all applicable operand fetches and writes, and all internal core cycles required to complete the instruction execution.
- r/w - number of operand reads (r) and writes (w) required by the instruction. An operation performing a read-modify-write function is denoted as (1/1).

This section includes the assumptions concerning the timing values and the execution time details.

3.6.1 Timing Assumptions

For the timing data presented in this section, the following assumptions apply:

1. The operand execution pipeline (OEP) is loaded with the opword and all required extension words at the beginning of each instruction execution. This implies that the OEP does not wait for the instruction fetch pipeline (IFP) to supply opwords and/or extension words.
2. The OEP does not experience any sequence-related pipeline stalls. For ColdFire 5200 processors, the most common example of this type of stall involves consecutive store operations, excluding the MOVEM instruction. For all STORE operations (except MOVEM), certain hardware resources within the processor are marked as “busy” for two clock cycles after the final DSOC cycle of the store instruction. If a subsequent STORE instruction is encountered within this 2-cycle window, it will be stalled until the resource again becomes available. Thus, the maximum pipeline stall involving consecutive STORE operations is 2 cycles. The MOVEM instruction uses a different set of resources and this stall does not apply.
3. The OEP completes all memory accesses without any stall conditions caused by the memory itself. Thus, the timing details provided in this section assume that an infinite zero-wait state memory is attached to the processor core.
4. All operand data accesses are aligned on the same byte boundary as the operand size, i.e., 16-bit operands aligned on 0-modulo-2 addresses, 32-bit operands aligned on 0-modulo-4 addresses.

If the operand alignment fails these guidelines, it is misaligned. The processor core decomposes the misaligned operand reference into a series of aligned accesses as shown in Table 3-4.

Table 3-4. Misaligned Operand References

ADDRESS[1:0]	SIZE	KBUS OPERATIONS	ADDITIONAL C(R/W)
X1	Word	Byte, Byte	2(1/0) if read 1(0/1) if write
X1	Long	Byte, Word, Byte	3(2/0) if read 2(0/2) if write
10	Long	Word, Word	2(1/0) if read 1(0/1) if write

3.6.2 MOVE Instruction Execution Times

The execution times for the MOVE.{B,W} instructions are shown in Table 3-5, while Table 3-6 provides the timing for MOVE.L.

For all tables in this section, the execution time of any instruction using the PC-relative effective addressing modes is the same for the comparable An-relative mode.

The nomenclature “xxx.wl” refers to both forms of absolute addressing, xxx.w and xxx.l.

Table 3-5. Move Byte and Word Execution Times

SOURCE	DESTINATION						
	RX	(AX)	(AX)+	-(AX)	(D16,AX)	(D8,AX,XN*SF)	XXX.WL
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(Ay)+	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
-(Ay)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(d16,Ay)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	–	–
(d8,Ay,Xn*SF)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	–	–	–
xxx.w	3(1/0)	3(1/1)	3(1/1)	3(1/1)	–	–	–
xxx.l	3(1/0)	3(1/1)	3(1/1)	3(1/1)	–	–	–
(d16,PC)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	–	–
(d8,PC,Xn*SF)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	–	–	–
#xxx	1(0/0)	3(0/1)	3(0/1)	3(0/1)	–	–	–

Table 3-6. Move Long Execution Times

SOURCE	DESTINATION						
	RX	(AX)	(AX)+	-(AX)	(D16,AX)	(D8,AX,XN*SF)	XXX.WL
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(Ay)+	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
-(Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(d16,Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	–	–
(d8,Ay,Xn*SF)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	–	–	–
xxx.w	2(1/0)	2(1/1)	2(1/1)	2(1/1)	–	–	–
xxx.l	2(1/0)	2(1/1)	2(1/1)	2(1/1)	–	–	–
(d16,PC)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	–	–
(d8,PC,Xn*SF)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	–	–	–
#xxx	1(0/0)	2(0/1)	2(0/1)	2(0/1)	–	–	–

3.7 STANDARD ONE OPERAND INSTRUCTION EXECUTION TIMES

Table 3-7. One Operand Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		RN	(AN)	(AN)+	-(AN)	(D16,AN)	(D8,AN,XN*SF)	XXX.WL	#XXX
CLR.B	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	–
CLR.W	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	–
CLR.L	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	–
EXT.W	Dx	1(0/0)	–	–	–	–	–	–	–
EXT.L	Dx	1(0/0)	–	–	–	–	–	–	–
EXTB.L	Dx	1(0/0)	–	–	–	–	–	–	–
NEG.L	Dx	1(0/0)	–	–	–	–	–	–	–
NEGX.L	Dx	1(0/0)	–	–	–	–	–	–	–
NOT.L	Dx	1(0/0)	–	–	–	–	–	–	–
Scc	Dx	1(0/0)	–	–	–	–	–	–	–
SWAP	Dx	1(0/0)	–	–	–	–	–	–	–
TST.B	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
TST.W	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
TST.L	<ea>	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)

3.8 STANDARD TWO OPERAND INSTRUCTION EXECUTION TIMES

Table 3-8. Two Operand Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		RN	(AN)	(AN)+	-(AN)	(D16,AN) (D16,PC)	(D8,AN,XN*SF) (D8,PC,XN*SF)	XXX.WL	#XXX
ADD.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
ADD.L	Dy,<ea>	–	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	–
ADDI.L	#imm,Dx	1(0/0)	–	–	–	–	–	–	–
ADDQ.L	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	–
ADDX.L	Dy,Dx	1(0/0)	–	–	–	–	–	–	–
AND.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
AND.L	Dy,<ea>	–	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	–
ANDI.L	#imm,Dx	1(0/0)	–	–	–	–	–	–	–
ASL.L	<ea>,Dx	1(0/0)	–	–	–	–	–	–	1(0/0)
ASR.L	<ea>,Dx	1(0/0)	–	–	–	–	–	–	1(0/0)
BCHG	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	–
BCHG	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	–	–	–
BCLR	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	–
BCLR	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	–	–	–
BSET	Dy,<ea>	2(0/0)	4(1/1)	41/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	–
BSET	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	–	–	–
BTST	Dy,<ea>	2(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	–
BTST	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	–	–	1(0/0)
CMP.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
CMPI.L	#imm,Dx	1(0/0)	–	–	–	–	–	–	–
EOR.L	Dy,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	–
EORI.L	#imm,Dx	1(0/0)	–	–	–	–	–	–	–
LEA	<ea>,Ax	–	1(0/0)	–	–	1(0/0)	2(0/0)	1(0/0)	–
LSL.L	<ea>,Dx	1(0/0)	–	–	–	–	–	–	1(0/0)
LSR.L	<ea>,Dx	1(0/0)	–	–	–	–	–	–	1(0/0)
MOVEQ	#imm,Dx	–	–	–	–	–	–	–	1(0/0)
MULS.W	<ea>,Dx	9(0/0)	11(1/0)	11(1/0)	11(1/0)	11(1/0)	12(1/0)	11(1/0)	9(0/0)
MULU.W	<ea>,Dx	9(0/0)	11(1/0)	11(1/0)	11(1/0)	11(1/0)	12(1/0)	11(1/0)	9(0/0)
MULS.L[1]	<ea>,Dx	£ 18(0/0)	£ 20(1/0)	£ 20(1/0)	£ 20(1/0)	£ 20(1/0)	–	–	–
MULU.L[1]	<ea>,Dx	£ 18(0/0)	£ 20(1/0)	£ 20(1/0)	£ 20(1/0)	£ 20(1/0)	–	–	–
OR.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
OR.L	Dy,<ea>	–	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	–
ORI.L	#imm,Dx	1(0/0)	–	–	–	–	–	–	–
SUB.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
SUB.L	Dy,<ea>	–	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	–
SUBI.L	#imm,Dx	1(0/0)	–	–	–	–	–	–	–
SUBQ.L	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	–
SUBX.L	Dy,Dx	1(0/0)	–	–	–	–	–	–	–

3.9 MISCELLANEOUS INSTRUCTION EXECUTION TIMES

Table 3-9. Miscellaneous Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		RN	(AN)	(AN)+	-(AN)	(D16,AN)	(D8,AN,XN*SF)	XXX.WL	#XXX
LINK.W	Ay,#imm	2(0/1)	–	–	–	–	–	–	–
MOVE.W	CCR,Dx	1(0/0)	–	–	–	–	–	–	–
MOVE.W	<ea>,CCR	1(0/0)	–	–	–	–	–	–	1(0/0)
MOVE.W	SR,Dx	1(0/0)	–	–	–	–	–	–	–
MOVE.W	<ea>,SR	7(0/0)	–	–	–	–	–	–	7(0/0) [2]
MOVEC	Ry,Rc	9(0/1)	–	–	–	–	–	–	–
MOVEM.L	<ea>,&list	–	1+n(n/0)	–	–	1+n(n/0)	–	–	–
MOVEM.L	&list,<ea>	–	1+n(0/n)	–	–	1+n(0/n)	–	–	–
NOP		3(0/0)	–	–	–	–	–	–	–
PEA	<ea>	–	2(0/1)	–	–	2(0/1) [4]	3(0/1) [5]	2(0/1)	–
PULSE		1(0/0)	–	–	–	–	–	–	–
STOP	#imm	–	–	–	–	–	–	–	3(0/0) [3]
TRAP	#imm	–	–	–	–	–	–	–	15(1/2)
TRAPF		1(0/0)	–	–	–	–	–	–	–
TRAPF.W		1(0/0)	–	–	–	–	–	–	–
TRAPF.L		1(0/0)	–	–	–	–	–	–	–
UNLK	Ax	2(1/0)	–	–	–	–	–	–	–
WDDATA	<ea>	–	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	3(1/0)
WDEBUG	<ea>	–	5(2/0)	–	–	5(2/0)	–	–	–

n is the number of registers moved by the MOVEM opcode.

[1] £ indicates that long multiplies have early termination after 9 cycles; thus, actual cycle count is operand independent

[2] If a MOVE.W #imm,SR instruction is executed and imm[13] = 1, the execution time is 1(0/0).

[3] The execution time for STOP is the time required until the processor begins sampling continuously for interrupts.

[4] PEA execution times are the same for (d16,PC)

[5] PEA execution times are the same for (d8,PC,Xn*SF)

3.10 BRANCH INSTRUCTION EXECUTION TIMES

Table 3-10. General Branch Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		RN	(AN)	(AN)+	-(AN)	(D16,AN) (D16,PC)	(D8,AN,Xi*SF) (D8,PC,Xi*SF)	XXX.WL	#XXX
BSR		–	–	–	–	3(0/1)	–	–	–
JMP	<ea>	–	3(0/0)	–	–	3(0/0)	4(0/0)	3(0/0)	–
JSR	<ea>	–	3(0/1)	–	–	3(0/1)	4(0/1)	3(0/1)	–
RTE		–	–	10(2/0)	–	–	–	–	–
RTS		–	–	5(1/0)	–	–	–	–	–

Table 3-11. BRA, Bcc Instruction Execution Times

OPCODE	FORWARD TAKEN	FORWARD NOT TAKEN	BACKWARD TAKEN	BACKWARD NOT TAKEN
BRA	2(0/0)	–	2(0/0)	–
Bcc	3(0/0)	1(0/0)	2(0/0)	3(0/0)