Transporting M68HC11 Code to M68HC16 Devices

By Michael Greenberg and Harold Roberson

1 INTRODUCTION

Devices in the Freescale M68HC16 modular microcontroller family are built up from standard modules that interface via a common internal bus. Modularity facilitates rapid development of devices tailored for specific applications. The standard central processing unit in the M68HC16 family is the 16-bit CPU16 module. Both the CPU16 programming model and the CPU16 instruction set are designed for compatibility with the M68HC11 CPU, and M68HC11 applications can be ported to the CPU16 with moderate effort. However, because the CPU16 has additional capabilities, the functions of certain M68HC11 instructions have been modified and some M68HC11 CPU instructions have been replaced by instructions specific to the CPU16. In addition, the M68HC11 CPU and CPU16 manage interrupts differently.

This note is intended to assist programmers who wish to transport code from the M68HC11 CPU to the CPU16. It compares the capabilities of the two processors, provides information concerning differences in the respective instruction sets, and discusses cases that need special attention. For more detailed information, please refer to the *M68HC11 Reference Manual* (M68HC11RM/AD) and to the *CPU16 Reference Manual* (CPU16RM/AD).

2 M68HC11 CPU

The M68HC11 CPU treats all peripheral, I/O, and memory locations as addresses in its memory map. There are no special instructions for I/O that are distinct from those used for memory. This architecture also allows accessing an operand from an external memory location with no execution-time penalty.

2.1 Programming Model

M68HC11 CPU registers are an integral part of the processing unit and are not addressed as memory locations. **Figure 1** shows the programming model. The following paragraphs describe the registers.

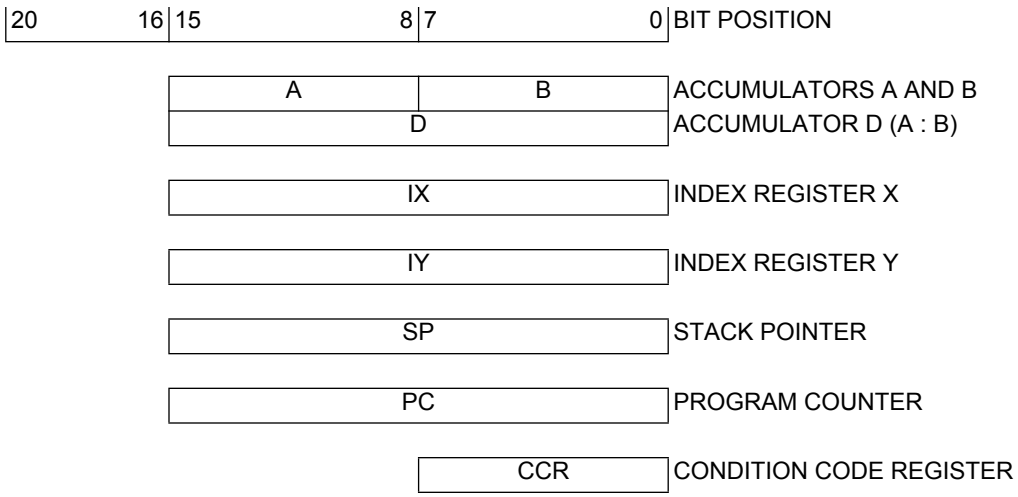


Figure 1 M68HC11 CPU Programming Model

2.1.1 Accumulators

The M68HC11 CPU has two general-purpose 8-bit accumulators (A and B). Accumulators A and B can be concatenated into a general-purpose 16-bit double accumulator (D). Although most operations can use A or B interchangeably, the following exceptions apply:

- The ABX and ABY instructions add the contents of B to the contents of IX or IY, but there are no equivalent instructions that use A rather than B.
- The TAP and TPA instructions transfer data from A to the CCR, or from the CCR to A, but there are no equivalent instructions that use B rather than A.
- The DAA instruction is used to adjust the content of A after BCD arithmetic operations, but there is no equivalent instruction for B.
- Add, subtract, and compare instructions that operate on both A and B (ABA, SBA, and CBA) only operate in one direction, making it important to place an operand in the correct accumulator.

2.1.2 Index Registers

The M68HC11 CPU has two 16-bit index registers (IX and IY). When an indexed addressing mode is used, a 16-bit value contained in an index register is added to an 8-bit offset provided by an instruction to create an effective address. The index registers can also be used as counters or as temporary storage registers. Because of M68HC11 opcode mapping, most instructions that use IY require an extra byte of machine code and an extra cycle of execution time.

2.1.3 Stack Pointer

The M68HC11 CPU stack pointer (SP) is 16 bits wide. The stack can be located anywhere in address space and can be any size up to the amount of memory available in the system. Stack entries are byte-width. The SP contains the 16-bit address of the next free location in the stack, rather than the address of the latest stack entry. SP is decremented each time data is pushed on the stack, and incremented each time data is pulled from the stack. The stack grows downward from high to low memory as it is filled.

2.1.4 Program Counter

The 16-bit program counter (PC) contains the address of the next instruction to be executed. The PC can be initialized with one of six possible vectors, depending on operating mode and the cause of reset.

2.1.5 Condition Code Register

As **Figure 2** shows, this 8-bit register contains five condition code indicators (H, N, Z, V and C), two interrupt masking bits (I and X), and a stop disable bit (S). In the M68HC11 CPU, condition codes are automatically updated by most instructions. However, pushes, pulls, Add B to X (ABX), Add B to Y (ABY), and transfer/exchange instructions do not affect the condition codes.

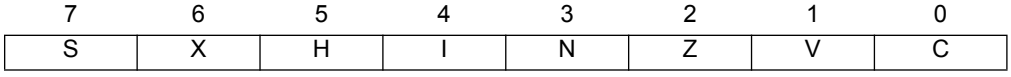


Figure 2 M68HC11 CPU Condition Code Register

S — STOP Enable

0 = Stop clock when STOP instruction is executed.

1 = Perform NOP when STOP instruction is executed.

X — X Interrupt Mask

Setting X disables interrupts from the $\overline{\text{XIRQ}}$ pin. X can be set only by hardware $\overline{\text{RESET}}$ or $\overline{\text{XIRQ}}$ acknowledge. X can be cleared by a TAP instruction or by an RTI (when bit 6 of the value restored from the stack to the CCR is cleared).

H — Half Carry Flag

Set when a carry from A3 or B3 occurs during BCD addition.

I — Interrupt Mask

I is a global mask that disables maskable interrupt sources. While I is set, no maskable interrupts are processed. After reset, I is set and can only be cleared by software. I is normally cleared when CCR content is restored by the RTI instruction at the end of an interrupt service routine.

N — Negative Flag

Set when the MSB of a result register is set.

Z — Zero Flag

Set when all bits of a result register are zero.

V — Overflow Flag

Set when two's complement overflow occurs as the result of an operation.

C — Carry Flag

Set when carry or borrow occurs during arithmetic operation. Also used during shift and rotate

2.2 Memory Management

All M68HC11 devices have a contiguous 64 Kbyte address space that is accessed by means of a 16-line address bus. Some devices have the upper eight address lines multiplexed with the data bus lines, while others have non-multiplexed address and data buses. Some variants also have address extension capabilities — the CPU address space remains 64 Kbytes, but on-chip logic and extra address lines are provided to implement bank-switching in external memory. Extended memory is accessed by means of two windows of a pre-defined size and extend.

2.3 Data Types

The M68HC11 CPU supports the following data types:

- Bit data
- 8-bit and 16-bit signed and unsigned integers
- 16-bit unsigned fractions
- 16-bit addresses

A byte is eight bits wide and can be accessed at any byte location. A word is composed of two consecutive bytes with the most significant byte at the lower value address. Because the M68HC11 CPU is an 8-bit CPU, there are no special requirements for alignment of instructions or operands.

2.4 Addressing Modes

The M68HC11 CPU uses six basic types of addressing. Each type consists of one or more addressing modes. All modes except inherent mode use an effective address. The effective address is the memory address from which an argument is fetched or stored, or the address from which execution is to proceed. An effective address can be specified within an instruction, or it can be calculated. **Table 1** shows the various M68HC11 CPU addressing modes.

Table 1 M68HC11 CPU Addressing Modes

Mode	Mnemonic	Description
Direct	DIR	Low-order byte of effective address follows opcode
Extended	EXT	Effective address follows opcode
Immediate	IMM	Argument follows opcode
Indexed	IND, X IND, Y	Effective address formed by adding unsigned 8-bit offset from instruction to index register content
Inherent	INH	Opcode contains information necessary for execution
Relative	REL	When a branch is taken, effective address formed by adding signed 8-bit offset from instruction to PC content.

2.4.1 Direct Mode

In the direct addressing mode, the low-order byte of the operand address is contained in a single byte following the opcode, and the high-order byte of the address is assumed to be $00. Addresses $00–$FF are thus accessed directly, using two-byte instructions. Execution time is reduced by eliminating the additional memory access required for the high-order address byte. In most applications, this 256-byte area is reserved for frequently referenced data. M68HC11 memory can be configured so that combinations of internal registers, RAM or external memory occupy these addresses.

2.4.2 Extended Mode

In the extended addressing mode, the effective address of the argument is contained in two bytes following the opcode byte.

2.4.3 Immediate Mode

In the immediate addressing mode an argument is contained in the byte(s) immediately following the opcode. The number of bytes following the opcode matches the size of the register or memory location being operated on. The effective address is the address of the byte following the instruction.

2.4.4 Indexed Modes

In the indexed addressing mode, an 8-bit unsigned offset contained in the instruction is added to the value contained in an index register (IX or IY) — the sum is the effective address. This addressing mode allows referencing any memory location in the 64 Kbyte address space.

2.4.5 Inherent Modes

In the inherent addressing mode, all the information necessary to execute the instruction is contained in the opcode. Operations that use only the index registers or accumulators, as well as control instructions with no arguments, are included in this addressing mode.

2.4.6 Relative Mode

The relative addressing mode is used only for branch instructions. If the branch condition is true, an 8-bit signed offset included in the instruction is added to the contents of the program counter to form the effective branch address. Otherwise, control proceeds to the next instruction.

2.5 Instructions

The M68HC11 family of microcontrollers uses 8-bit opcodes. Each opcode identifies a particular instruction and associated addressing mode to the CPU. Several opcodes are required to provide each instruction with a range of addressing capabilities. Only 256 opcodes would be available if the range of values were restricted to the number able to be expressed in 8-bit binary numbers.

A four-page opcode map has been implemented to expand the number of instructions. An additional byte, called a prebyte, directs the processor from page 0 of the opcode map to one of the other three pages. As its name implies, the additional byte precedes the opcode.

A complete instruction consists of a prebyte, if any, an opcode, and zero, one, two, or three operands. The operands contain information the CPU needs for executing the instruction. Complete instructions can be from one to five bytes long.

2.6 Instruction Execution

The M68HC11 CPU fetches and executes instruction bytes sequentially from byte addresses. The program counter is incremented by one after each opcode or operand byte fetch.

2.7 Changes in Program Flow

M68HC11 jump, branch, and subroutine instructions initiate changes in program flow. When program flow changes, instructions are fetched from a new address. When a change in flow is temporary, a return address is stored, so that execution of the original instruction stream can resume after the change in flow.

The jump (JMP) instruction uses direct, extended and indexed addressing modes. Jumps are unconditional changes in flow. No return PC value is stacked prior to executing a jump instruction.

The M68HC11 CPU supports a number of 8-bit relative displacement branch instructions, as well as specialized bit condition branches that use the direct and indexed addressing modes. Branch instructions are conditional changes in flow. A change occurs only if a pre-defined condition is satisfied. No return PC value is stacked prior to executing a branch instruction.

Subroutines are called by special branch (BSR) or jump (JSR) instructions. The RTS instruction returns control to the calling routine after a subroutine has executed. JSR uses the direct, indexed, and extended addressing modes; BSR uses only relative addressing mode. When a subroutine instruction is executed, the PC points to the address of the instruction that follows the instruction that calls the subroutine. Both calling instructions stack the high and low bytes of this return PC value. The return PC is pulled from the stack when RTS is executed at the end of a subroutine.

2.8 Reset And Interrupt Vectors

Reset and interrupt operations load the M68HC11 CPU program counter with a vector that points to a new location from which instructions are to be fetched. **Table 2** shows vector assignments for a typical M68HC11 device.

2.9 Resets

Resets are generally used to initialize the MCU or to recover from catastrophic failure. A reset immediately stops program execution and forces the program counter to a known starting address. Internal registers and control bits are initialized so the MCU can resume operation in a known state. There are four possible sources of reset. External reset and power-on reset share a vector. The computer operating properly system and the clock monitor each have a vector.

The M68HC11 CPU distinguishes between internal and external reset conditions by measuring the time it takes the MCU $\overline{\text{RESET}}$ line to return to logic level one after assertion. When a reset condition is sensed, an internal circuit drives the $\overline{\text{RESET}}$ pin low for four ECLK cycles, then releases it. Two ECLK cycles later, the logic level of the $\overline{\text{RESET}}$ line is sampled. If it is still low, the CPU assumes that an external reset has occurred. If it is high, the CPU assumes that reset was initiated internally.

A positive transition on V_{dd} generates a power-on reset, which is used only for power-up conditions. A 4064 clock cycle delay after the oscillator becomes active allows the clock generator to stabilize. If $\overline{\text{RESET}}$ is low at the end of 4064 clock cycles, the CPU remains in reset condition until $\overline{\text{RESET}}$ goes high.

The MCU includes a computer operating properly (COP) system to help protect against software failures. When the system is enabled, software is responsible for keeping a free-running watchdog timer from timing out. If the software fails to update the timer control register, a system reset occurs.

The clock monitor circuit is based on an internal RC time delay. If no MCU clock edges are detected within the delay period, the clock monitor can generate a system reset.

When a reset condition is recognized, the internal registers and control bits are forced to an initial state. Depending on the cause of the reset and the operating mode, the reset vector can be fetched from one of the six possible locations shown in **Table 3** .

The M68HC11 CPU fetches the appropriate vector during the first three cycles after reset, then begins fetching instructions from the address pointed to by the vector. The stack pointer and other CPU registers are indeterminate immediately after reset, but the X and I interrupt mask bits in the CCR are set to mask interrupt requests.

Table 2 M68HC11 Interrupt and Reset Vector Assignments

Vector Address	Interrupt Source	CCR Mask Bit	Local Mask
FFC0, C1 – FFD4, D5	Reserved	—	—
FFD6, D7	SCI Serial System	I	
	• SCI Transmit Complete		TCIE
	• SCI Transmit Data Register Empty		TIE
	• SCI Idle Line Detect		ILIE
	• SCI Receiver Overrun		RIE
	• SCI Receive Data Register Full		RIE
FFD8, D9	SPI Serial Transfer Complete	I	SPIE
FFDA, DB	Pulse Accumulator Input Edge	I	PAII
FFDC, DD	Pulse Accumulator Overflow	I	PAOVI
FFDE, DF	Timer Overflow	I	TOI
FFE0, E1	Timer Input Capture 4/Output Compare 5	I	I4/O5I
FFE2, E3	Timer Output Compare 4	I	OC4I
FFE4, E5	Timer Output Compare 3	I	OC3I
FFE6, E7	Timer Output Compare 2	I	OC2I
FFE8, E9	Timer Output Compare 1	I	OC1I
FFEA, EB	Timer Input Capture 3	I	IC3I
FFEC, ED	Timer Input Capture 2	I	IC2I
FFEE, EF	Timer Input Capture 1	I	IC1I
FFF0, F1	Real-Time Interrupt	I	RTII
FFF2, F3	Parallel I/O Handshake	I	STAI
	$\overline{\text{IRQ}}$		None
FFF4, F5	$\overline{\text{XIRQ}}$ Pin	X	None
FFF6, F7	Software Interrupt	None	None
FFF8, F9	Illegal Opcode Trap	None	None
FFFA, FB	COP Failure	None	NOCOP
FFFC, FD	Clock Monitor Fail	None	CME
FFFE, FF	$\overline{\text{RESET}}$	None	None

Table 3 Reset Vectors

Cause of Reset	Normal Mode Vector	Special Test or Bootstrap
$\overline{\text{RESET}}$ pin	$FFFE, FFFF	$BFFE, BFFF
Power-on reset	$FFFE, FFFF	$BFFE, BFFF
Clock monitor reset	$FFFC, FFFD	$BFFC, BFFD
COP system reset	$FFFA, FFFB	$BFFA, BFFB

2.10 Interrupts

An interrupt temporarily suspends normal program execution while an interrupt service routine is being executed. After an interrupt has been serviced, the main program resumes as if there had been no interruption. Maskable interrupts are recognized only when the CCR I bit is cleared. Maskable interrupts are generated by on-chip peripheral systems, and are enabled by control bits in MCU registers associated with these systems. Nonmaskable interrupt sources are not masked by the I bit. The three nonmaskable interrupt sources are the illegal opcode trap, the software interrupt instruction, and the $\overline{\text{XIRQ}}$ pin. Operation of the $\overline{\text{XIRQ}}$ pin is enabled by the CCR X bit.

Upon reset, both the X bit and the I bit are set, which inhibits both maskable interrupts and $\overline{\text{XIRQ}}$ interrupts. After reset, software can clear both X and I to enable interrupt recognition. Once cleared, the X bit cannot be set by software.

An interrupt request can be recognized at any time, but the CPU does not respond to a request until completion of the instruction being executed. Interrupt latency varies according to the number of cycles required to complete the current instruction.

When the CPU begins to service an interrupt, the contents of the CPU registers are pushed onto the stack in the order shown in **Table 4** .

Table 4 Stacking Order on Entry to Interrupts

Memory Location	CPU Registers
SP	PCL
SP – 1	PCH
SP – 2	IYL
SP – 3	IYH
SP – 4	IXL
SP – 5	IXH
SP – 6	ACCA
SP – 7	ACCB
SP – 8	CCR

After the CCR value is stacked, the appropriate mask bit is set to inhibit further interrupts. When an I-bit-related interrupt occurs, the I bit is set after stacking, but the X bit is not affected. When an X-bit-related interrupt occurs, both the X and I bits are set after stacking.

After stacking and masking take place, the priority of pending requests is evaluated, and the interrupt vector for the highest priority pending source is fetched. Execution of the interrupt service routine begins at the address pointed to by the vector. At the end of the interrupt service routine, the return from interrupt instruction (RTI) is executed and the stacked registers are restored from the stack (restoring the CCR restores the X and I bits to their pre-interrupt request state), and normal program execution resumes.

2.11 Reset and Interrupt Priority

Resets and interrupts have a hardware priority that determines which reset or interrupt is serviced first when simultaneous requests occur. There are six nonmaskable reset and interrupt sources. The priority hierarchy for these sources is as follows:

1. POR or $\overline{\text{RESET}}$ pin
2. Clock monitor reset
3. COP watchdog reset
4. $\overline{\text{XIRQ}}$ interrupt
5. Illegal opcode interrupt
6. Software interrupt (SWI)

Maskable interrupt sources have the following priorities:

1. $\overline{\text{IRQ}}$ Interrupt
2. Real-time interrupt
3. Timer input capture 1
4. Timer input capture 2
5. Timer input capture 3
6. Timer output compare 1
7. Timer output compare 2
8. Timer output compare 3
9. Timer output compare 4
10. Timer input capture 4/output compare 5
11. Timer overflow
12. Pulse accumulator overflow
13. Pulse accumulator input edge
14. SPI transfer complete
15. SCI system

Any single interrupt source can be designated as the highest-priority interrupt by writing an appropriate value to the PSEL bits in the HPRIO register. Priority relationships of other maskable interrupts remain the same. An interrupt that is assigned highest priority is still subject to global masking by the I bit. Interrupt vectors are not affected by priority assignment.

3 CPU16 MODULE

The M68HC16 central processing unit (CPU16) was designed to provide compatibility with the M68HC11 CPU and to provide additional capabilities associated with 16- and 32-bit data sizes, 20-bit addressing, and digital signal processing.

The CPU16 treats all peripheral, I/O, and memory locations as parts of a pseudolinear 1 Megabyte address space. There are no special instructions for I/O that are separate from instructions for addressing memory. Address space is made up of 16 64-Kbyte banks. Specialized bank addressing techniques and support registers provide transparent access across bank boundaries.

The CPU16 interacts with external devices and with other modules within the microcontroller via a standardized bus and bus interface. There are bus protocols for memory and peripheral accesses, as well as for managing an hierarchy of interrupt priorities.

3.1 Programming Model

CPU16 registers are an integral part of the CPU and are not addressed as memory locations. The CPU16 register model contains all the resources of the M68HC11 CPU, plus additional resources. **Figure 3** shows the CPU16 programming model. Registers are discussed in detail in the following paragraphs.

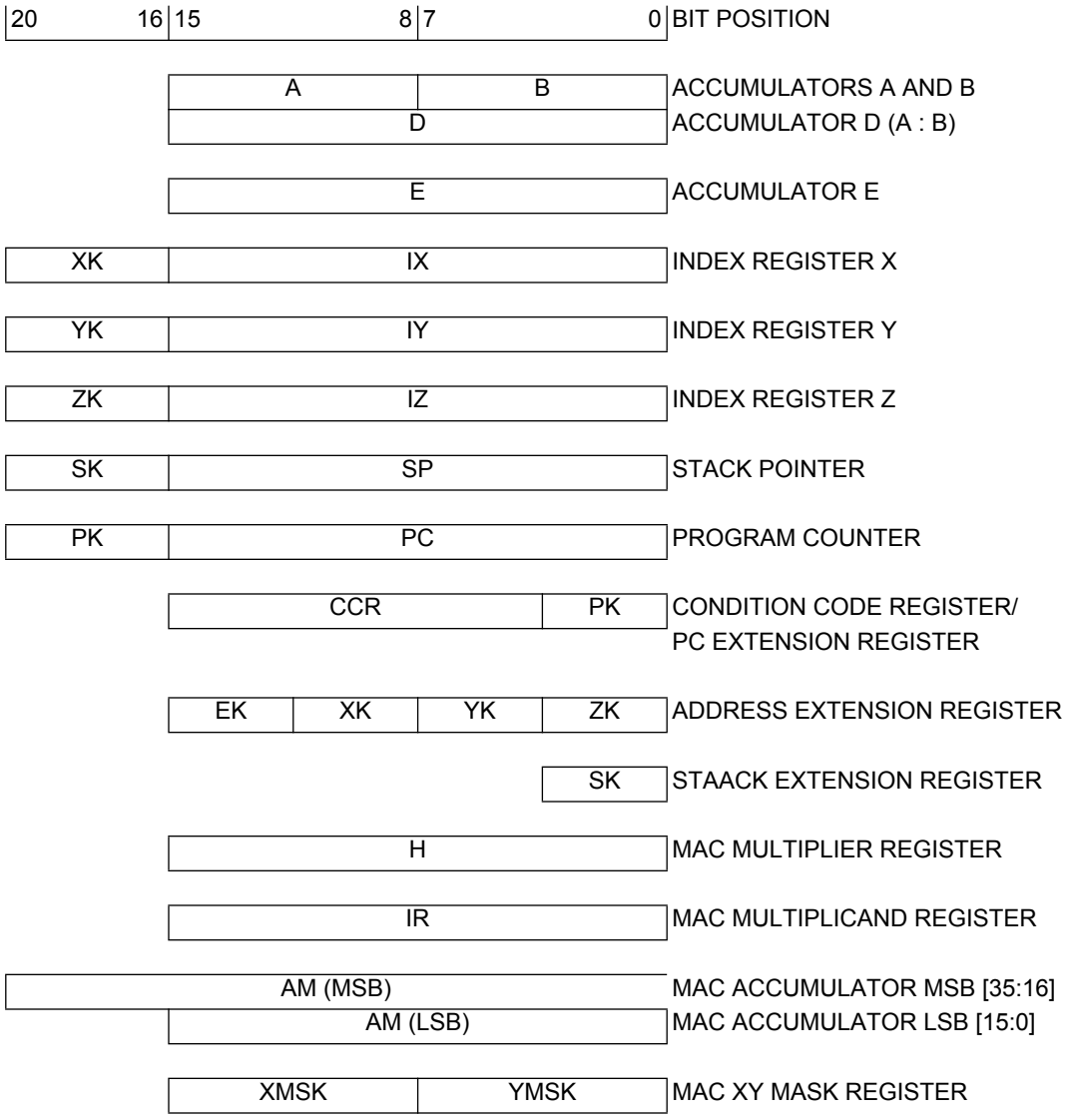


Figure 3 CPU16 Programming Model

3.1.1 Accumulators

The CPU16 has two 8-bit accumulators (A and B) and one 16-bit accumulator (E). In addition, accumulators A and B can be concatenated into a second 16-bit double accumulator (D).

Accumulators A, B, and D are general-purpose registers used to hold operands and results during mathematical and data manipulation operations.

Accumulator E can be used in the same way as accumulator D, and also extends CPU16 capabilities. It allows more data to be held within the CPU16 during operations, simplifies 32-bit arithmetic and digital signal processing, and provides a practical 16-bit accumulator offset indexed addressing mode.

3.1.2 Index Registers

The CPU16 has three 16-bit index registers (IX, IY, and IZ). Each index register has an associated 4-bit extension field (XK, YK, and ZK).

Concatenated registers and extension fields provide 20-bit indexed addressing and support data structure functions anywhere in the CPU16 address space.

,X and IY can perform the same operations as M68HC11 CPU registers of the same names, but the CPU16 instruction set provides additional indexed operations.

IZ can perform the same operations as IX and IY, and also provides an additional indexed addressing capability that replaces M68HC11 CPU direct addressing mode. Initial IZ and ZK extension field values are included in the RESET exception vector, so that ZK : IZ can be used as a direct page pointer out of reset.

3.1.3 Stack Pointer

The CPU16 stack pointer (SP) is 16 bits wide. An associated 4-bit extension field (SK) provides 20-bit stack addressing.

Stack implementation in the CPU16 is from high to low memory. The stack grows downward as it is filled. SK : SP are decremented each time data is pushed on the stack, and incremented each time data is pulled from the stack.

SK : SP point to the next available stack address, rather than to the address of the latest stack entry. Although the stack pointer is normally incremented or decremented by word address, it is possible to push and pull byte-sized data. Setting the stack pointer to an odd value causes misalignment, which affects performance.

3.1.4 Program Counter

The CPU16 program counter (PC) is 16 bits wide. An associated 4-bit extension field (PK) provides 20-bit program addressing.

CPU16 instructions are fetched from even word boundaries. PC0 always has a value of zero, to assure that instruction fetches are made from word-aligned addresses.

3.1.5 Condition Code Register

The 16-bit condition code register can be divided into two functional blocks. The 8 MSB, which correspond to the CCR in the M68HC11 CPU, contain the low-power stop control bit and processor status flags. The 8 LSB contain the interrupt priority field, the DSP saturation mode control bit, and the program counter address extension field.

Figure 4 shows the condition code register. Detailed descriptions of each status indicator and field in the register follow the figure.

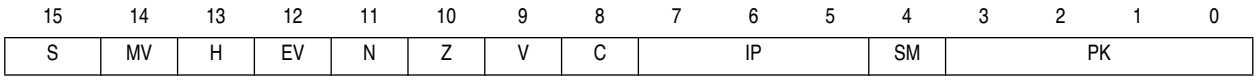


Figure 4 CPU16 Condition Code Register

S — STOP Enable

0 = Stop clock when LPSTOP instruction is executed.

1 = Perform NOP when LPSTOP instruction is executed.

MV — Accumulator M overflow flag

Set when overflow into AM35 has occurred.

H — Half Carry Flag

Set when a carry from A3 or B3 occurs during BCD addition.

EV — Extension Bit Overflow Flag

Set when an overflow into AM31 has occurred.

N — Negative Flag

Set when the MSB of a result register is set.

Z — Zero Flag

Set when all bits of a result register are zero.

✓ — Overflow Flag

Set when two's complement overflow occurs as the result of an operation.

C — Carry Flag

Set when carry or borrow occurs during arithmetic operation. Also used during shift and rotate to facilitate multiple word operations.

IP[2:0] — Interrupt Priority Field

The priority value in this field (0 to 7) is used to mask interrupts.

SM — Saturate Mode Bit

When SM is set, if either EV or MV is set, data read from AM using TMER or TMET will be given maximum positive or negative value, depending on the state of the AM sign bit before overflow.

PK[3:0] — Program Counter Address Extension Field

This field is concatenated with the program counter to form a 20-bit address.

3.1.6 Address Extension Register and Address Extension Fields

There are six 4-bit address extension fields. EK, XK, YK, and ZK are contained by the address extension register, PK is part of the CCR, and SK stands alone.

Extension fields are the bank portions of 20-bit concatenated bank : byte addresses used in the CPU16 pseudolinear memory management scheme.

All extension fields except EK correspond directly to a register. XK, YK, and ZK extend registers IX, IY, and IZ; PK extends the PC; and SK extends the SP. EK holds the 4 MSB of the 20-bit address used by extended addressing mode.

3.1.7 Multiply and Accumulate Registers

The multiply and accumulate (MAC) registers are part of a CPU submodule that performs repetitive signed fractional multiplication and stores the cumulative result. These operations are part of control-oriented digital signal processing.

There are four MAC registers. Register H contains the 16-bit signed fractional multiplier. Register I contains the 16-bit signed fractional multiplicand. Accumulator M is a specialized 36-bit product accumulation register. XMSK and YMSK contain 8-bit mask values used in modulo addressing.

The CPU16 has a special subset of signal processing instructions that manipulate the MAC registers and perform signal processing calculation.

3.2 Memory Management

The CPU16 uses bank switching to provide a 1 Megabyte address space. There are 16 banks within the address space. Each bank is made up of 64 Kbytes addressed from $0000 to $FFFF. Banks are selected by means of address extension fields associated with individual CPU16 registers. CPU16 addressing is pseudolinear — a 20-bit extended address can access any byte location in the appropriate address space.

In addition, address space can be split into discrete 1 Megabyte program and data spaces by externally decoding the SIM function code outputs. When this technique is used, instruction fetches and reset vector fetches access program space, while exception vector fetches (other than for reset), data accesses, and stack accesses are made in data space.

3.2.1 Address Extension

All CPU16 resources that are used to generate addresses are effectively 20 bits wide. These resources include extended index registers, program counter, and stack pointer. All addressing modes use 20-bit addresses. 20-bit addresses are formed from a 16-bit byte address generated by an individual CPU16 register and a 4-bit bank address contained in an associated extension field. The byte address corresponds to ADDR[15:0] and the bank address corresponds to ADDR[19:16].

3.3 Data Types

The CPU16 uses the following types of data:

- Bits
- 4-bit signed integers
- 8-bit (byte) signed and unsigned integers
- 8-bit, 2-digit binary coded decimal numbers
- 16-bit (word) signed and unsigned integers
- 32-bit (long word) signed and unsigned integers
- 16-bit signed fractions
- 32-bit signed fractions
- 36-bit signed fixed-point numbers
- 20-bit effective address consisting of 16-bit byte address and 4-bit extension

There are 8 bits in a byte, 16 bits in a word. Bit set and clear instructions use both byte and word operands. Bit test instructions use byte operands.

Negative integers are represented in two's-complement form. Four-bit signed integers, packed two to a byte, are used only as X and Y offsets in MAC and RMAC operations. Thirty-two-bit integers are used only by extended multiply and divide instructions, and by the associated LDED and STED instructions.

Binary coded decimal numbers are packed, two digits per byte. BCD operations use byte operands.

16-bit fractions are used in both fractional multiplication and division, and as multiplicand and multiplier operands in the MAC unit. Bit 15 is the sign bit. There is an implied radix point between bits 15 and 14. There are 15 bits of magnitude — the range of values is –1 ($8000) to $1 - 2^{-15}$ ($7FFF).

Signed 32-bit fractions are used only by fractional multiplication and division instructions. Bit 31 is the sign bit. An implied radix point lies between bits 31 and 30. There are 31 bits of magnitude — the range of values is –1 ($80000000) to $1 - 2^{-31}$ ($7FFFFFFF).

Signed 36-bit fixed-point numbers are used only by the MAC unit. Bit 35 is the sign bit. Bits [34:31] are sign extension bits. There is an implied radix point between bits 31 and 30. There are 31 bits of magnitude, but use of the extension bits allows representation of numbers in the range –16 ($800000000) to 15.999969482 ($7FFFFFFFF).

20-bit addresses are formed by combining a 16-bit byte address with a 4-bit address extension.

3.4 Memory Organization

A word is composed of two consecutive bytes. A word address is normally an even byte address. Byte 0 of a word has a lower 16-bit address than Byte 1. Long words and 32-bit signed fractions consist of two consecutive words, and are normally accessed at the address of Byte 0 in Word 0.

Instruction fetches always access word addresses. Word operands are normally accessed at even byte addresses, but may be accessed at odd byte addresses, with a substantial performance penalty.

To be compatible with the M68HC11 CPU, misaligned word transfers and misaligned stack accesses are allowed. Transferring a misaligned word requires two successive byte transfer operations.

3.5 Addressing Modes

The CPU16 uses 9 basic types of addressing. There are one or more addressing modes within each type. **Table 5** shows the addressing modes.

Table 5 CPU16 Addressing Modes

Mode	Mnemonic	Description
Accumulator Offset	E,X	Index Register X with Accumulator E offset
	E,Y	Index Register Y with Accumulator E offset
	E,Z	Index Register Z with Accumulator E offset
Extended	EXT	Extended
	EXT20	20-bit Extended
Immediate	IMM8	8-bit Immediate
	IMM16	16-bit Immediate
Indexed 8-Bit	IND8, X	Index Register X with unsigned 8-bit offset
	IND8, Y	Index Register Y with unsigned 8-bit offset
	IND8, Z	Index Register Z with unsigned 8-bit offset
Indexed 16-Bit	IND16, X	Index Register X with signed 16-bit offset
	IND16, Y	Index Register Y with signed 16-bit offset
	IND16, Z	Index Register Z with signed 16-bit offset
Indexed 20-Bit	IND20, X	Index Register X with signed 20-bit offset
	IND20, Y	Index Register Y with signed 20-bit offset
	IND20, Z	Index Register Z with signed 20-bit offset
Inherent	INH	Inherent
Post-Modified Index	IXP	Signed 8-bit offset added to Index Register X after effective address is used
Relative	REL8	8-bit relative
	REL16	16-bit relative

All modes generate ADDR[15:0]. This address is combined with ADDR[19:16] from an operand or an extension field to form a 20-bit effective address. Bank switching is transparent to most instructions. ADDR[19:16] of the effective address are changed to make an access across a bank boundary. However, extension field values do not change as a result of effective address computation.

3.5.1 Immediate Addressing Modes

In the immediate modes, an argument is contained in a byte or word immediately following the instruction. For IMM8 and IMM16 modes, the effective address is the address of the argument.

There are three specialized forms of IMM8 addressing. The AIS, AIX/Y/Z, ADDD and ADDE instructions decrease execution time by sign-extending the 8-bit immediate operand to 16 bits, then adding it to an appropriate register. The MAC and RMAC instructions use an 8-bit immediate operand to specify two signed 4-bit index register offsets. The PSHM and PULM instructions use an 8-bit immediate mask operand to indicate which registers must be pushed to or pulled from the stack.

3.5.2 Extended Addressing Modes

Regular extended mode instructions contain ADDR[15:0] in the word following the opcode. The effective address is formed by concatenating the EK field and the 16-bit byte address. EXT20 mode is used only by the JMP and JSR instructions. These instructions contain a 20-bit effective address that is zero-extended to 24 bits to give the instruction an even number of bytes.

3.5.3 Indexed Addressing Modes

In the indexed modes, registers IX, IY, and IZ, together with their associated extension fields, are used to calculate the effective address. For 8-bit indexed modes an 8-bit unsigned offset contained in the instruction

is added to the value contained in an index register and its extension field. For 16-bit modes, a 16-bit signed offset contained in the instruction is added to the value contained in an index register and its extension field. For 20-bit modes, a 20-bit signed offset (zero-extended to 24 bits) is added to the value contained in an index register. These modes are used for JMP and JSR instructions only.

3.5.4 Inherent Addressing Mode

Inherent mode instructions use information directly available to the processor to determine the effective address. Operands (if any) are system resources and are thus not fetched from memory.

3.5.5 Accumulator Offset Addressing Mode

Accumulator offset modes form an effective address by sign-extending the content of accumulator E to 20 bits, then adding the result to an index register and its associated extension field. This mode allows use of an index register and an accumulator within a loop without corrupting accumulator D.

3.5.6 Relative Addressing Modes

Relative modes are used for branch and long branch instructions. If a branch condition is satisfied, a byte or word signed two's-complement offset is added to the concatenated PK field and program counter. The new PK : PC value is the effective address.

3.5.7 Post-Modified Index Addressing Mode

Post-modified index mode is used by the MOVB and MOVW instructions. A signed 8-bit offset is added to index register X after the effective address formed by XK : IX is used.

3.6 Instructions

The instruction set is based upon that of the M68HC11 CPU, but the opcode map has been rearranged to maximize performance with a 16-bit data bus. Much M68HC11 code can run on the CPU16 following reassembly. The user must take into account changed instruction times, the interrupt mask, and the new interrupt stack frame.

CPU16 instructions consist of an 8-bit opcode, which may be preceded by an 8-bit prebyte and followed by one or more operands.

Opcodes are mapped in four 256-instruction pages. Page 0 opcodes stand alone, but Page 1, 2, and 3 opcodes are pointed to by a prebyte code on Page 0. The prebytes are $17 (Page 1), $27 (Page 2), and $37 (Page 3).

Operands can be 4 bits, 8 bits or 16 bits in length. However, because the CPU16 fetches 16-bit instruction words from even byte boundaries, each instruction must contain an even number of bytes.

Operands are organized as bytes, words, or a combination of bytes and words. Four-bit operands are either zero-extended to 8 bits, or packed two to a byte. The largest instructions are six bytes in length. Size, order, and function of operands are evaluated when an instruction is decoded.

A Page 0 opcode and an 8-bit operand can be fetched simultaneously. Instructions that use 8-bit indexed, immediate, and relative addressing modes have this form. Code written with these instructions is very compact.

3.7 CPU16 Pipeline Mechanism

This description is a simplified model of the mechanism the CPU16 uses to fetch and execute instructions. Functional divisions in the model do not necessarily correspond to distinct architectural subunits of the microprocessor.

There are three functional blocks involved in fetching, decoding, and executing instructions. These are the microsequencer, the instruction pipeline, and the execution unit. These elements function concurrently — at any given time, all three may be active.

3.7.1 Microsequencer

The microsequencer controls the order in which instructions are fetched, advanced through the pipeline, and executed. It increments the program counter and generates multiplexed external tracking signals IPIPE0 and IPIPE1 from internal signals that control execution sequence.

3.7.2 Instruction Pipeline

The pipeline is a three stage FIFO that holds instructions while they are decoded and executed. As many as three instructions can be in the pipeline at one time (single-word instructions, one held in stage C, one being executed in stage B, and one latched in stage A).

3.7.3 Execution Unit

The execution unit evaluates opcodes, interfaces with the microsequencer to advance instructions through the pipeline, and performs instruction operations.

3.8 Execution Process

A prefetch mechanism in the microsequencer reads instruction words from memory and increments the program counter. When instruction execution begins, the program counter points to an address six bytes after the address of the first word of the instruction being executed.

Fetched opcodes are latched into stage A, then advanced to stage B. Opcodes are evaluated in stage B. The execution unit can access operands in either stage A or stage B (stage B accesses are limited to 8-bit operands). When execution is complete, opcodes are moved from stage B to stage C, where they remain until the next instruction is complete. The number of machine cycles necessary to complete an execution sequence varies according to the complexity of the instruction.

3.9 Changes in Program Flow

When program flow changes, instructions are fetched from a new address. Before execution can begin at the new address, instructions and operands from the previous instruction stream must be removed from the pipeline. If a change in flow is temporary, a return address must be stored, so that execution of the original instruction stream can resume after the change in flow.

At the time an instruction that causes a change in program flow executes, PK : PC point to the address of the first word of the instruction + $0006. During execution of the instruction, PK : PC is loaded with the address of the first word of the new instruction stream. However, stages A and B still contain words from the old instruction stream. The CPU16 prefetches to advance the new instruction to stage C, and fills the pipeline from the new instruction stream.

3.9.1 Jumps

The CPU16 jump instruction uses 20-bit extended and indexed addressing modes. It consists of an 8-bit opcode with a 20-bit argument. No return PK : PC is stacked for a jump.

3.9.2 Branches

The CPU16 supports 8-bit relative displacement (short), and 16-bit relative displacement (long) branch instructions, as well as specialized bit condition branches that use indexed addressing modes. CPU16 short branches are generally equivalent to M68HC11 CPU branches, although opcodes are not identical. M68HC11 BHI and BLO are replaced by CPU16 BCC and BCS.

Short branch instructions consist of an 8-bit opcode and an 8-bit operand contained in one word. Long branch instructions consist of an 8-bit prebyte and an 8-bit opcode in one word, followed by an operand word. Bit condition branches consist of an 8-bit opcode and an 8-bit operand in one word, followed by one or two operand words.

When a branch instruction executes, PK : PC point to an address equal to the address of the first word of the instruction plus $0006. The range of displacement for each type of branch is relative to this value. In addition, because prefetches are automatically aligned to word boundaries, only even offsets are valid. An odd offset value is rounded down.

3.9.3 Subroutines

Subroutines can be called by short (BSR) or long (LBSR) branches, or by a jump (JSR). The RTS instruction returns control to the calling routine. BSR consists of an 8-bit opcode with an 8-bit operand. LBSR consists of an 8-bit prebyte and an 8-bit opcode in one word, followed by an operand word. JSR consists of an 8-bit opcode with a 20-bit argument. RTS consists of an 8-bit prebyte and an 8-bit opcode in one word.

When a subroutine instruction is executed, PK : PC contain the address of the calling instruction plus $0006. All three calling instructions stack return PK : PC values prior to processing instructions from the new instruction stream. In order for RTS to work with all three calling instructions, however, the value stacked by BSR must be adjusted.

LBSR and JSR are two-word instructions. In order for program execution to resume with the instruction immediately following them, RTS must subtract $0002 from the stacked PK : PC value. BSR is a one-word instruction — it subtracts $0002 from PK : PC prior to stacking so that execution will resume correctly.

3.10 Exceptions

An exception is an event that preempts normal instruction process. Exception processing makes the transition from normal instruction execution to execution of a routine that deals with an exception.

Each exception has an assigned vector that points to an associated handler routine. Exception processing includes all operations required to transfer control to a handler routine, but does not include execution of the handler routine itself. Keep the distinction between exception processing and execution of an exception handler in mind while reading this section.

3.10.1 Exception Vectors

An exception vector is the address of a routine that handles an exception. Exception vectors are contained in a data structure called the exception vector table, which is located in the first 512 bytes of Bank 0.

All vectors except the reset vector consist of one word and reside in data space. The reset vector consists of four words that reside in program space. There are 52 predefined or reserved vectors, and 200 user-defined vectors.

Each vector is assigned an 8-bit number. Vector numbers for some exceptions are generated by external devices; others are supplied by the processor. There is a direct mapping of vector number to vector table address. The processor left shifts the vector number one place (multiplies by two) to convert it to an address. **Table 6** shows the exception vectors.

3.10.2 Exception Stack Frame

During exception processing, the contents of the program counter and condition code register are stacked at a location pointed to by SK : SP. Unless it is altered during exception processing, the stacked PK : PC value is the address of the next instruction in the current instruction stream, plus $0006. **Figure 5** shows the exception stack frame.

Table 6 Exception Vector Table

Vector Number	Vector Address	Address Space	Type of Exception
0	0000	P	Reset — Initial ZK, SK, and PK
	0002	P	Reset — Initial PC
	0004	P	Reset — Initial SP
	0006	P	Reset — Initial IZ
4	0008	D	Breakpoint
5	000A	D	Bus Error
6	000C	D	Software Interrupt Instruction (SWI)
7	000E	D	Illegal Instruction
8	0010	D	Division by Zero
9 – E	0012 – 001C	D	Unassigned, Reserved
F	001E	D	Uninitialized Interrupt
10	0020	D	Unassigned, Reserved
11	0022	D	Level 1 Interrupt Autovector
12	0024	D	Level 2 Interrupt Autovector
13	0026	D	Level 3 Interrupt Autovector
14	0028	D	Level 4 Interrupt Autovector
15	002A	D	Level 5 Interrupt Autovector
16	002C	D	Level 6 Interrupt Autovector
17	002E	D	Level 7 Interrupt Autovector
18	0030	D	Spurious Interrupt
19 – 37	0032 – 006E	D	Unassigned, Reserved
38 – FF	0070 – 01FE	D	User-Defined Interrupts

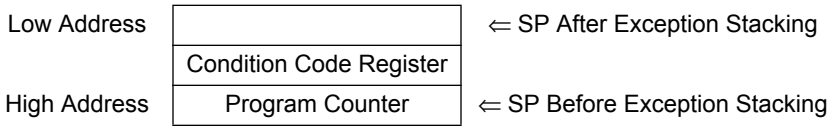


Figure 5 Exception Stack Frame Format

3.10.3 Exception Processing Sequence

Exception processing is performed in four distinct phases.

1. Priority of all pending exceptions is evaluated, and the highest priority exception is processed first.
2. Processor state is stacked, then the CCR PK extension field is cleared.
3. An exception vector number is acquired and converted to a vector address.
4. The content of the vector address is loaded into the PC, and the processor jumps to the exception handler routine.

There are variations within each phase for differing types of exceptions. However, all vectors but the reset vectors contain 16-bit addresses, and the PK field is cleared. Exception handlers must be located within Bank 0 or vectors must point to a jump table.

3.10.4 Types of Exceptions

Exceptions can be either internally or externally generated. External exceptions, which are defined as asynchronous, include interrupts, bus errors, breakpoints, and resets. Internal exceptions, which are defined as synchronous, include the software interrupt (SWI) instruction, the background (BGND) instruction, illegal instruction exceptions, and the divide-by-zero exception.

3.10.4.1 Asynchronous Exceptions

Asynchronous exceptions occur without reference to CPU16 or IMB clocks, but exception processing is synchronized. For all asynchronous exceptions but reset, exception processing begins at the first instruction boundary following recognition of an exception.

Because of pipelining, the stacked return PK : PC value for all asynchronous exceptions, other than reset, is equal to the address of the next instruction in the current instruction stream plus $0006. The RTI instruction, which must terminate all exception handler routines, subtracts $0006 from the stacked value in order to resume execution of the interrupted instruction stream.

3.10.4.2 Synchronous Exceptions

Synchronous exception processing is part of an instruction definition. Exception processing for synchronous exceptions will always be completed, and the first instruction of the handler routine will always be executed, before interrupts are detected.

Because of pipelining, the value of PK : PC at the time a synchronous exception executes is equal to the address of the instruction that causes the exception plus $0006. Since RTI always subtracts $0006 upon return, the stacked PK : PC must be adjusted so that execution will resume with the following instruction. For this reason $0002 is added to the PK : PC value before it is stacked.

3.10.4.3 Multiple Exceptions

Each exception has a hardware priority based upon its relative importance to system operation. Asynchronous exceptions have higher priorities than synchronous exceptions. Exception processing for multiple exceptions is done by priority, from lowest to highest. Priority governs the order in which exception processing occurs, not the order in which exception handlers are executed.

Unless a bus error, a breakpoint, or a reset occurs during exception processing, the first instruction of all exception handler routines is guaranteed to execute before another exception is processed. Since interrupt exceptions have higher priority than synchronous exceptions, this means that the first instruction in an interrupt handler will be executed before other interrupts are sensed.

Bus error, breakpoint, and reset exceptions that occur during exception processing of a previous exception are processed before the first instruction of that exception's handler routine. The converse is not true — if an interrupt occurs during bus error exception processing, for example, the first instruction of the bus error handler is executed before interrupts are sensed. This permits the exception handler to mask interrupts during execution.

3.11 RTI Instruction

The return-from-interrupt (RTI) instruction is used to terminate all exception handlers except the reset handler. RTI restores context so that normal execution can resume. Asynchronous interrupts are serviced at instruction boundaries, and a value of PK : PC + $0006 is stacked when exception processing begins. RTI subtracts $0006 from the stacked value so that the pipeline is refilled from the correct address. RTI is not used in the reset handler because the system is re-initialized and there is no context to restore.

SWI initiates interrupt exception processing without an external service request. The PK : PC value at the time of execution is the first word address of SWI plus $0006. If this value were stacked, execution of RTI at the end of the handler would cause SWI to execute again. To prevent this, SWI adds $0002 to the PK : PC value prior to stacking.

3.12 Resets

Reset procedures handle system initialization and recovery from catastrophic failure. M68HC16 microcontrollers perform resets with a combination of hardware and software. The system integration module determines whether a reset is valid, asserts control signals, performs basic system configuration and boot ROM selection based on hardware mode-select inputs, then passes control to the CPU16.

Reset occurs when an active low logic level on the $\overline{\text{RESET}}$ pin is clocked into the SIM. Resets are gated by the CLKOUT signal. Asynchronous resets are assumed to be catastrophic. An asynchronous reset can occur on any clock edge. Synchronous resets are timed to occur at the end of bus cycles. If there is no clock when $\overline{\text{RESET}}$ is asserted, reset does not occur until the clock starts. Resets are clocked in order to allow completion of write cycles in progress at the time $\overline{\text{RESET}}$ is asserted.

Reset is the highest-priority CPU16 exception. Any processing in progress is aborted by the reset exception, and cannot be restarted. Only essential tasks are performed during reset exception processing. Other initialization tasks must be accomplished by the exception handler routine.

The logic states of certain data bus pins during reset determine SIM operating configuration. In addition, the state of the MODCLK pin determines system clock source and the state of the $\overline{\text{BKPT}}$ pin determines what happens during subsequent breakpoint assertions.

Generally, module pins default to port functions, and input/output ports are set to input state. This is accomplished by disabling pin functions in the appropriate control registers, and by clearing the appropriate port data direction registers.

3.12.1 Reset Timing

The $\overline{\text{RESET}}$ input must be asserted for a specified minimum period in order for reset to occur. External $\overline{\text{RESET}}$ assertion can be delayed internally for a period equal to the longest bus cycle time (or the bus monitor timeout period) in order to protect write cycles from being aborted by reset. While $\overline{\text{RESET}}$ is asserted, SIM pins are either in an inactive, high-impedance state or are driven to their inactive states.

When an external device asserts $\overline{\text{RESET}}$ for the proper period, reset control logic clocks the signal into an internal latch. The control logic drives the $\overline{\text{RESET}}$ pin low for an additional 512 CLKOUT cycles after it detects that the $\overline{\text{RESET}}$ signal is no longer being externally driven, to guarantee this length of reset to the entire system.

If an internal source asserts a reset signal, the reset control logic asserts $\overline{\text{RESET}}$ for a minimum of 512 cycles. If the reset signal is still asserted at the end of 512 cycles, the control logic continues to assert $\overline{\text{RESET}}$ until the internal reset signal is negated.

After 512 cycles have elapsed, the reset input pin goes to an inactive, high-impedance state for 10 cycles. At the end of this 10-cycle period, the reset input is tested. When the input is at logic level one, reset exception processing begins. If, however, the reset input is at logic level zero, the reset control logic drives the pin low for another 512 cycles. At the end of this period, the pin again goes to high-impedance state for 10 cycles, then it is tested again. The process repeats until $\overline{\text{RESET}}$ is released.

During power-on reset, an internal circuit in the SIM drives the IMB internal and external reset lines. The circuit releases the internal reset line as V_{DD} ramps up to the minimum specified value, and SIM pins are initialized. When V_{DD} reaches the specified minimum value, the clock synthesizer VCO begins operation. Clock frequency ramps up to the specified limp mode frequency. The external $\overline{\text{RESET}}$ line remains asserted until the clock synthesizer PLL locks and 512 CLKOUT cycles elapse.

The SIM clock synthesizer provides clock signals to the other MCU modules. After the clock is running and the internal reset signal is asserted for four clock cycles, these modules reset. V_{DD} ramp time and VCO frequency ramp time determine how long these four cycles take. Worst case is approximately 15 milliseconds. During this period, module port pins may be in an indeterminate state. While input-only pins can be put in a known state by means of external pull-up resistors, external logic on input/output or output-only pins must condition the lines during this time. Active drivers require high-impedance buffers or isolation resistors to prevent conflict.

3.13 Interrupts

Interrupt recognition and servicing involve complex interaction between the central processing unit, the system integration module, and a device or module requesting interrupt service.

The CPU16 provides for eight levels of interrupt priority (0–7), seven automatic interrupt vectors, and 200 assignable interrupt vectors. All interrupts with priorities less than 7 can be masked by the interrupt priority (IP) field in the condition code register. The CPU16 handles interrupts as a type of asynchronous exception.

Interrupt recognition is based on the states of interrupt request signals $\overline{\text{IRQ}}$[7:1] and the IP mask value. Each of the signals corresponds to an interrupt priority. $\overline{\text{IRQ1}}$ has the lowest priority, and $\overline{\text{IRQ7}}$ has the highest priority.

The IP field consists of three bits (CCR[7:5]). Binary values %000 to %111 provide eight priority masks. Masks prevent an interrupt request of a priority less than or equal to the mask value (except for $\overline{\text{IRQ7}}$) from being recognized and processed. When IP contains %000, no interrupt is masked. During exception processing, the IP field is set to the priority of the interrupt being serviced.

Interrupt request signals can be asserted by external devices or by microcontroller modules. Request lines are connected internally by means of a wired NOR — simultaneous requests of differing priority can be made. Internal assertion of an interrupt request signal does not affect the logic state of the corresponding MCU pin.

External interrupt requests are routed to the CPU16 via the external bus interface and SIM interrupt control logic. The CPU treats external interrupt requests as though they come from the SIM.

External $\overline{\text{IRQ}}$[6:1] are active-low level-sensitive inputs. External $\overline{\text{IRQ7}}$ is an active-low transition-sensitive input. $\overline{\text{IRQ7}}$ requires both an edge and a voltage level for validity.

$\overline{\text{IRQ}}$[6:1] are maskable. $\overline{\text{IRQ7}}$ is nonmaskable. The $\overline{\text{IRQ7}}$ input is transition-sensitive in order to prevent redundant servicing and stack overflow. A nonmaskable interrupt is generated each time $\overline{\text{IRQ7}}$ is asserted, and each time the priority mask changes from %111 to a lower number while $\overline{\text{IRQ7}}$ is asserted.

Interrupt requests are sampled on consecutive falling edges of the system clock. Interrupt request input circuitry has hysteresis. To be valid, a request signal must be asserted for at least two consecutive clock periods. Valid requests do not cause immediate exception processing, but are left pending. Pending requests are processed at instruction boundaries or when exception processing of higher-priority exceptions is complete.

The CPU16 does not latch the priority of a pending interrupt request. If an interrupt source of higher priority makes a service request while a lower priority request is pending, the higher priority request is serviced. If an interrupt request of equal or lower priority than the current IP mask value is made, the CPU does not recognize the occurrence of the request in any way.

3.13.1 Interrupt Acknowledge and Arbitration

Interrupt acknowledge bus cycles are generated during exception processing. When the CPU16 detects one or more interrupt requests of a priority higher than the interrupt priority mask value, it performs a CPU space read from address $FFFFF : [IP] : 1.

The CPU space read cycle performs two functions: it places a mask value corresponding to the highest priority interrupt request on the address bus, and it acquires an exception vector number from the interrupt source. The mask value also serves two purposes: it is latched into the CCR IP field in order to mask lower-priority interrupts during exception processing, and it is decoded by modules that have requested interrupt service to determine whether the current interrupt acknowledge cycle pertains to them.

Modules that have requested interrupt service decode the IP value placed on the address bus at the beginning of the interrupt acknowledge cycle, and if their requests are at the specified IP level, respond to the cycle. Arbitration between simultaneous requests of the same priority is performed by means of serial contention between module interrupt arbitration (IARB) field bit values.

Each module that can make an interrupt service request, including the SIM, has an IARB field in its configuration register. An IARB field can be assigned a value from %0001 (lowest priority) to %1111 (highest priority). A value of %0000 in an IARB field causes the CPU16 to process a spurious interrupt exception when an interrupt from that module is recognized.

Because the EBI manages external interrupt requests, the SIM IARB value is used for arbitration between internal and external interrupt requests. The reset value of IARB for the SIM is %1111, and the reset IARB value for all other modules is %0000. Initialization software must assign different IARB values in order to implement an arbitration scheme.

Each module must be assigned a unique IARB value. When two or more IARB fields have the same nonzero value, the CPU16 attempts to interpret multiple vector numbers simultaneously, with unpredictable consequences.

Arbitration must always take place, even when a single source requests service. This point is important for two reasons: the CPU interrupt acknowledge cycle is not driven on the external bus unless the SIM wins contention, and failure to contend causes an interrupt acknowledge bus cycle to be terminated by a bus error, which causes a spurious interrupt exception to be taken.

When arbitration is complete, the dominant module must place an interrupt vector number on the data bus and terminate the bus cycle. In the case of an external interrupt request, because the interrupt acknowledge cycle is transferred to the external bus, an external device must decode the mask value and respond with a vector number, then generate bus cycle termination signals. If the device does not respond in time, a spurious interrupt exception is taken.

The periodic interrupt timer (PIT) in the SIM can generate internal interrupt requests of specific priority at predetermined intervals. By hardware convention, PIT interrupts are serviced before external interrupt service requests of the same priority.

3.13.2 Interrupt Processing Summary

A valid interrupt service request has been detected and is pending.

The CPU finishes higher priority exception processing or reaches an instruction boundary.
Processor state is stacked, then the CCR PK extension field is cleared.
FC[2:0] are driven to %111 (CPU space) encoding.
The address bus is driven as follows:
- ADDR[23:20] = %1111;
- ADDR[19:16] = %1111, indicating an interrupt acknowledge CPU space cycle;
- ADDR[15:4] = %111111111111;
- ADDR[3:1] = the priority of the interrupt request being acknowledged;
- ADDR0 = %1.

Request priority is latched into the CCR IP field from the address bus.
Modules or external peripherals that have requested interrupt service decode ADDR[3:1].
IARB contention takes place.
The interrupt vector number is generated, in one of four ways:
- If contention has not produced a dominant interrupt source (IARB = %0000), the CPU16 generates the spurious interrupt vector number.

If contention has produced a dominant interrupt source, it supplies the vector number.
If the autovector signal is asserted, the CPU16 generates a vector number that corresponds to interrupt request priority.
If the bus monitor asserts the bus error signal, the CPU16 generates the spurious interrupt vector number.
The CPU16 converts the vector number to a vector address.
The content of the vector address is loaded into the PC.
The exception handler routine begins to execute.

3.14 Development Support

The CPU16 incorporates powerful tools for tracking program execution and for system debugging. These tools are deterministic opcode tracking, breakpoint exceptions, and background debugging mode. Judicious use of CPU16 capabilities permits in-circuit emulation and system debugging using a bus state analyzer, a simple serial interface, and a terminal. Refer to *CPU16 Reference Manual* (CPU16RM/AD) for more information.

4 COMPARISON OF INSTRUCTION SETS

This section provides detailed analysis of differences between M68HC11 instructions and CPU16 instructions. Topics include functionally equivalent instructions, instructions with the same mnemonic that operate differently, functions that perform the same operation in a different way, and unimplemented instructions.

4.1 Functionally Equivalent Instructions

The CPU16 has a number of instructions that are functionally equivalent to M68HC11 instructions — a CPU16 instruction with a different mnemonic that performs the same task as an M68HC11 instruction. The following paragraphs give the mnemonic of the M68HC11 instruction, then discuss the equivalent CPU16 operation.

4.1.1 BHS

The BHS mnemonic is used in the M68HC11 CPU instruction set to differentiate a branch based on a comparison of unsigned numbers from a branch based on operations that clear the Carry bit. The CPU16 uses only the BCC mnemonic.

4.1.2 BHO

The BLO mnemonic is used in the M68HC11 CPU instruction set to differentiate a branch based on a comparison of unsigned numbers from a branch based on operations that set the Carry bit. The CPU16 uses only the BCS mnemonic.

4.1.3 CLC

The CLC instruction has been replaced by ANDP. ANDP performs AND between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[3:0]) is not affected.

The following code can be used to clear the C bit in the CCR:

```
ANDP #$FEFF
```

The ANDP instruction can clear the entire CCR, except for the PK extension field, at once.

4.1.4 CLI

The CLI instruction has been replaced by ANDP. ANDP performs AND between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[3:0]) is not affected.

The following code can be used to clear the IP field in the CCR:

```
ANDP #$FF1F
```

The ANDP instruction can clear the entire CCR, except for the PK extension field, at once.

4.1.5 CLV

The CLV instruction has been replaced by ANDP. ANDP performs AND between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[3:0]) is not affected.

The following code can be used to clear the V bit in the CCR:

```
ANDP #$FDFF
```

The ANDP instruction can clear the entire CCR, except for the PK extension field, at once.

4.1.6 DES

The DES instruction has been replaced by AIS. AIS adds a 20-bit value to concatenated SK and SP. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform a DES:

```
AIS –1
```

CPU16 stacking operations normally use 16-bit words and even word addresses, while M68HC11 CPU stacking operations normally use bytes and byte addresses. If the CPU16 stack pointer is misaligned as a result of a byte operation, performance can be degraded.

4.1.7 DEX

The DEX instruction has been replaced by AIX. AIX adds a 20-bit value to concatenated XK and IX. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform a DEX:

```
AIX –1
```

4.1.8 DEY

The DEY instruction has been replaced by AIY. AIY adds a 20-bit value to concatenated YK and IY. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform a DEY:

```
AIY –1
```

4.1.9 INS

The INS instruction has been replaced by AIS. AIS adds a 20-bit value to concatenated SK and SP. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform an INS:

```
AIS –1
```

CPU16 stacking operations normally use 16-bit words and even word addresses, while M68HC11 CPU stacking operations normally use bytes and byte addresses. If the CPU16 stack pointer is misaligned as a result of a byte operation, performance can be degraded.

4.1.10 INX

The INX instruction has been replaced by AIX. AIX adds a 20-bit value to concatenated XK and IX. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform an INX:

```
AIX1
```

4.1.11 INY

The INY instruction has been replaced by AIY. AIY adds a 20-bit value to concatenated YK and IY. The 20-bit value is formed by sign-extending an 8-bit or 16-bit signed immediate operand.

The following code can be used to perform an INY:

```
AIY 1
```

4.1.12 PSHX

The PSHX instruction has been replaced by PSHM. PSHM stores the contents of selected registers on the system stack. Registers are designated by setting bits in a mask byte.

The following code can be used to stack index register X:

```
PSHM X
```

The CPU16 can stack up to seven registers with a single PSHM instruction.

4.1.13 PSHY

The PSHY instruction has been replaced by PSHM. PSHM stores the contents of selected registers on the system stack. Registers are designated by setting bits in a mask byte.

The following code can be used to stack index register Y:

```
PSHM Y
```

The CPU16 can stack up to seven registers with a single PSHM instruction.

4.1.14 PULX

The PULX instruction has been replaced by PULM. PULM restores the contents of selected registers from the system stack. Registers are designated by setting bits in a mask byte.

The following code can be used to restore index register X:

```
PULM X
```

The CPU16 can restore up to seven registers with a single PULM instruction. As a part of normal execution, PULM reads an extra location in memory. The extra data is discarded. A PULM from the highest available location in memory will cause an attempt to read an unimplemented location, with unpredictable results.

4.1.15 PULY

The PULY instruction has been replaced by PULM. PULM restores the contents of selected registers from the system stack. Registers are designated by setting bits in a mask byte.

The following code can be used to restore index register Y:

```
PULM Y
```

The CPU16 can restore up to seven registers with a single PULM instruction. As a part of normal execution, PULM reads an extra location in memory. The extra data is discarded. A PULM from the highest available location in memory will cause an attempt to read an unimplemented location, with unpredictable results.

4.1.16 SEC

The SEC instruction has been replaced by ORP. ORP performs inclusive OR between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[3:0]) is not affected.

The following code can be used to set the CCR C bit:

ORP #$0100

The ORP instruction can set all CCR bits, except the PK extension field, at once.

4.1.17 SEI

The SEI instruction has been replaced by ORP. ORP performs inclusive OR between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[3:0]) is not affected.

The following code can be used to set all the bits in the CCR IP field:

ORP #$00E0

The ORP instruction can set all CCR bits, except the PK extension field, at once.

4.1.18 SEV

The SEV instruction has been replaced by ORP. ORP performs inclusive OR between the content of the condition code register and an unsigned immediate operand, then replaces the content of the CCR with the result. The PK extension field (CCR[3:0]) is not affected.

The following code can be used to set the CCR V bit:

ORP #$0200

The ORP instruction can set all CCR bits, except the PK extension field, at once.

4.1.19 STOP and WAIT instructions

There are two instructions that put the M68HC11 CPU in an inactive state. Both require that either an interrupt or a reset occur before normal execution of instructions resumes. The STOP instruction turns off on-chip clocks and reduces power consumption to a minimum while retaining the contents of RAM. The WAIT instruction suspends processing and reduces power consumption to an intermediate level.

STOP operation is controlled by the S bit in the CCR. If S = 0 when STOP is executed, the MCU goes to stop condition. If S = 1 when STOP is executed, the STOP opcode is treated as a NOP. While the MCU is stopped, all MCU clocks, including the crystal oscillator, are turned off, and all internal peripheral functions stop. The MCU remains stopped until an interrupt or reset occurs. The interrupt can be an internally-generated interrupt, an external $\overline{\text{IRQ}}$, or an $\overline{\text{XIRQ}}$. An internal interrupt or the $\overline{\text{IRQ}}$ pin re-activate the MCU only when the I bit in the CCR is cleared — processing resumes with the instruction that follows the STOP instruction. $\overline{\text{XIRQ}}$ assertion always activates the MCU, but recovery sequence depends upon X-bit state. If X = 0, the MCU executes the stacking sequence leading to normal $\overline{\text{XIRQ}}$ interrupt service when it restarts. If X = 1, processing restarts with the instruction that follows the STOP instruction. When a reset is used to restart the system, a normal reset sequence is performed before processing begins.

When WAI is executed, CPU registers are stored and processing is suspended. The on-chip crystal oscillator remains active. The MCU remains in wait state until an interrupt is detected. The interrupt can be an external $\overline{\text{IRQ}}$, an $\overline{\text{XIRQ}}$, or any of the internally generated interrupts, such as the timer or serial interrupts. CCR interrupt mask bits (I and X) affect interrupt recognition during wait state. Reduction of power during wait depends on how many internal peripheral clock signals are turned off. These clocks must be turned off by means of control bits in the appropriate peripheral system registers. The MCU free-running timer system is shut down only if the I bit is set and the COP system is disabled. WAI does not significantly reduce analog-to-digital converter power consumption. While in the wait state, the address/data bus repeatedly runs read cycles to the address where the CCR contents are stacked.

The CPU16 also has two instructions that put it in an inactive state. Both require that either an interrupt or a reset exception occur before normal execution resumes. LPSTOP minimizes microcontroller power consumption. WAI idles the CPU16, but does not affect operation of other microcontroller modules. To make certain that conditions for termination of LPSTOP and WAI are correct, interrupts are not recognized until after the instruction following ANDP, ORP, TAP, and TDP executes. This prevents interrupt exception processing during the period after the mask changes but before the following instruction executes.

LPSTOP operation is controlled by the S bit in the CCR. If S = 0 when LPSTOP is executed, the IP field from the condition code register is copied into an external bus interface, and the MCU system clock is disabled. If S = 1, LPSTOP operates in the same way as a 4-cycle NOP. The CPU16 initiates low-power stop, but it and other controller modules are deactivated by the microcontroller system integration module. Reactivation is also handled by the integration module. When a reset or an interrupt of higher priority than the IP value occurs, the integration module activates the CPU16, and the appropriate exception processing sequence begins.

When WAI is executed, internal CPU clocks are stopped, and normal execution of instructions ceases. The IP field is not copied to the integration module. System clocks continue to run. The processor waits until a reset or an interrupt of higher priority than the IP value occurs, then begins the appropriate exception processing sequence. Because the system integration module does not restart the CPU16, interrupts are acknowledged more quickly following WAI than following LPSTOP.

4.2 Instructions That Operate Differently

There are a number of CPU16 instructions that have the same mnemonic as an M68HC11 instruction, but operate differently. The following paragraphs discuss the differences in detail.

4.2.1 BSR

The CPU16 stack frame differs from the M68HC11 CPU stack frame. The CPU16 stacks the current PC and CCR, but restores only the return PK : PC. The programmer must designate (PSHM) which other registers are stacked during a subroutine. Because SK : SP point to the next available word address, stacked CPU16 parameters are at a different offset from the stack pointer than stacked M68HC11 CPU parameters. In order for RTS to work with all three calling instructions, the PK : PC value stacked by BSR is decremented by two before being pushed on to the stack. Stacked PC value is the return address + $0002.

4.2.2 JSR

The CPU16 stack frame differs from the M68HC11 CPU stack frame. The CPU16 stacks the current PC and CCR, but restores only the return PK : PC. The programmer must designate (PSHM) which other registers are stacked during a subroutine. Because SK : SP point to the next available word address, stacked CPU16 parameters are at a different offset from the stack pointer than stacked M68HC11 CPU parameters.

4.2.3 PSHA, PSHB

These instructions operate in the same way as the M68HC11 CPU instructions with the same mnemonics. However, because the CPU16 normally pushes words from an even boundary, pushing byte data to the stack can misalign the stack pointer and degrade performance.

4.2.4 PULA, PULB

These instructions operate in the same way as the M68HC11 CPU instructions with the same mnemonics. However, because the CPU16 normally pulls words from the stack, pulling byte data can misalign the stack pointer and degrade performance.

4.2.5 RTI

The CPU16 stack frame differs from the M68HC11 CPU stack frame. The CPU16 stacks only the current PC and CCR before exception processing begins. In order to resume execution after interrupt with the correct instruction, RTI subtracts $0006 from the stacked PK : PC.

4.2.6 SWI

The CPU16 stack frame differs from the M68HC11 CPU stack frame. The PK : PC value at the time of execution is the first word address of SWI plus $0006. If this value were stacked, RTI would cause SWI to execute again. In order to resume execution with the instruction following SWI, $0002 is added to the PK : PC value prior to stacking. PSHM must be used to stack other registers during an interrupt.

4.2.7 TAP

The CPU16 CCR differs from the M68HC11 CPU CCR. The CPU16 interrupt priority scheme differs from that of the M68HC11 CPU, and the CPU16 interrupt priority field cannot be changed by the TAP instruction.

4.2.7.1 M68HC11 CPU Implementation:

7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0
⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓
7	6	5	4	3	2	1	0
S	X	H	I	N	Z	V	C

4.2.7.2 CPU16 Implementation:

7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0
⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			

4.2.8 TPA

The CPU16 CCR and the M68HC11 CPU CCR are different. TPA cannot be used to read CPU16 interrupt priority status. Use TPD to read the CPU16 CCR interrupt priority field.

4.2.8.1 M68HC11 CPU Implementation:

7	6	5	4	3	2	1	0
S	X	H	I	N	Z	V	C
⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓
7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A0

4.2.8.2 CPU16 Implementation:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	MV	H	EV	N	Z	V	C	IP			SM	PK			
⇓	⇓	⇓	⇓	⇓	⇓	⇓	⇓								
7	6	5	4	3	2	1	0								
A7	A6	A5	A4	A3	A2	A1	A0								

4.3 Instructions With Transparent Changes

Some CPU16 instructions with mnemonics identical to M68HC11 instructions function differently than the corresponding M68HC11 instructions, but accomplish the equivalent operation. These instructions are discussed in the following paragraphs.

4.3.1 RTS

The CPU16 stack frame differs from the M68HC11 CPU stack frame. PK : PC is restored during an RTS. The PK field in the CCR is restored, then the PC value read from the stack is decremented by two before being loaded into the PC. The PC value is decremented because LBSR and JSR are two-word instructions. In order for program execution to resume with the instruction immediately following them, RTS must subtract $0002 from the stacked PK : PC value. Because BSR is a one-word instruction, it subtracts $0002 from PK : PC prior to stacking so that execution will resume correctly after RTS.

4.3.2 TSX

The CPU16 adds 2 to SK : SP before the transfer to XK : IX. The M68HC11 CPU adds 1.

4.3.3 TSY

The CPU16 adds 2 to SK : SP before the transfer to YK : IY. The M68HC11 CPU adds 1.

4.3.4 TXS

The CPU16 subtracts 2 from XK : IX before the transfer to SK : SP. The M68HC11 CPU subtracts 1.

4.3.5 TYS

The CPU16 subtracts 2 from YK : IY before the transfer to SK : SP. The M68HC11 CPU subtracts 1.

4.4 Unimplemented Instructions

There is only one M68HC11 instruction that has no CPU16 equivalent.

4.4.1 TEST

Causes the program counter to be continuously incremented.

4.5 Summary of Instruction Set Differences

Table 7 provides a quick reference to differences between the CPU16 and M68HC11 CPU instruction sets. Refer to appropriate paragraphs in the preceding sections for detailed information.

Table 7 CPU16 Implementation of M68HC11 CPU Instructions

M68HC11 CPU Instruction	CPU16 Implementation
BHS	BCC
BLO	BCS
BSR	Generates a different stack frame
CLC	Replaced by ANDP
CLI	Replaced by ANDP
CLV	Replaced by ANDP
DES	Replaced by AIS
DEX	Replaced by AIX
DEY	Replaced by AIY
INS	Replaced by AIS
INX	Replaced by AIX
INY	Replaced by AIY
JMP	IND8 addressing modes replaced by IND20 and EXT modes
JSR	IND8 addressing modes replaced by IND20 and EXT modes Generates a different stack frame
LSL, LSLD	Use ASL instructions*
PSHX	Replaced by PSHM
PSHY	Replaced by PSHM
PULX	Replaced by PULM
PULY	Replaced by PULM
RTI	Reloads PC and CCR only
RTS	Uses two-word stack frame
SEC	Replaced by ORP
SEI	Replaced by ORP
SEV	Replaced by ORP
STOP	Replaced by LPSTOP
TAP	CPU16 CCR bits differ from M68HC11 CPU CPU16 interrupt priority scheme differs from M68HC11
TPA	CPU16 CCR bits differ from M68HC11 CPU CPU16 interrupt priority scheme differs from M68HC11
TSX	Adds 2 to SK : SP before transfer to XK : IX
TSY	Adds 2 to SK : SP before transfer to YK : IY
TXS	Subtracts 2 from XK : IX before transfer to SK : SP
TXY	Transfers XK field to YK field
TYS	Subtracts 2 from YK : IY before transfer to SK : SP
TYX	Transfers YK field to XK field
WAI	Waits indefinitely for interrupt or reset Generates a different stack frame

*Freescale assemblers automatically translate LSL mnemonics

In general, CPU16 addressing modes can be thought of as a superset of M68HC11 CPU addressing modes. The CPU16 has all the capabilities of the M68HC11 CPU, and each mode is enhanced by the pseudolinear addressing scheme. In addition, M68HC11 direct addressing has been replaced by an enhanced form of indexed addressing that can use the IZ register as a pointer out of reset.

5.1 Addressing Mode Differences

The following paragraphs summarize the differences between CPU16 addressing modes and the equivalent M68HC11 CPU addressing modes. In addition, the effects discussed in **3.7 CPU16 Pipeline Mechanism** must be considered when indexed modes are used.

5.1.1 Extended Addressing Mode

In M68HC11 CPU extended addressing mode, the effective address of the instruction appears explicitly in the two bytes following the opcode. In CPU16 extended addressing mode, the effective address is formed by concatenating the EK field and the 16-bit byte address. A 20-bit extended mode (EXT20) is used only by the JMP and JSR instructions. These instructions contain a 20-bit effective address that is zero-extended to 24 bits to give the instruction an even number of bytes.

5.1.2 Indexed Addressing Mode

M68HC11 CPU indexed addressing mode forms the effective address by adding an 8-bit unsigned offset to the index register. In CPU16 indexed addressing mode, a 16-bit offset can be used. However, the 16-bit offset is signed and effective address calculation can yield a negative offset from the index register. An 8-bit unsigned mode is still available on the CPU16. A 20-bit indexed mode is used for JMP and JSR instructions. In 20-bit modes, a 20-bit signed offset is added to the value contained in an index register.

5.1.3 Post-Modified Index Addressing Mode

Post-modified index mode is used with the CPU16 MOVB and MOVW instructions. A signed 8-bit offset is added to index register X after the effective address formed by XK : IX is used.

5.2 Use Of CPU16 Indexed Mode To Replace M68HC11 Direct Mode

In M68HC11 systems, direct addressing mode can be used to perform rapid accesses to RAM or I/O mapped into bank 0 ($0000 to $00FF), but the CPU16 uses the first 512 bytes of bank 0 for exception vectors. To provide an enhanced replacement for direct mode, the ZK field and index register Z have been assigned reset initialization vectors. After ZK : IZ have been initialized, indexed addressing can provide rapid access to any address in the memory map.

6 INSTRUCTION SET REFERENCE

Table 8 and **Table 9** are comprehensive references to the M68HC11 CPU and the CPU16 instructions sets. For more detailed information, please refer to the *M68HC11 Reference Manual* (M68HC11RM/AD) and to the *CPU16 Reference Manual* (CPU16RM/AD).

Table 8 M68HC11 Instruction Set (Sheet 1 of 6)

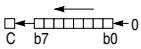
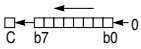
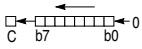
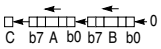
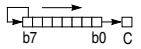
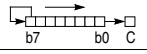
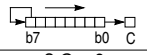
Mnemonic	Operation	Description	Addressing Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	X	H	I	N	Z	V	C
ABA	Add Accumulators	A + B ⇒ A	INH	1B	—	2	—	—	Δ	—	Δ	Δ	Δ	Δ
ABX	Add B to X	IX + (00 : B) ⇒ IX	INH	3A	—	3	—	—	—	—	—	—	—	—
ABY	Add B to Y	IY + (00 : B) ⇒ IY	INH	18 3A	—	4	—	—	—	—	—	—	—	—
ADCA (opr)	Add with Carry to A	A + M + C ⇒ A	A IMM A DIR A EXT A IND,X A IND,Y	89 99 B9 A9 18 A9	ii dd hh ll ff ff	2 3 4 4 5	—	—	Δ	—	Δ	Δ	Δ	Δ
ADCB (opr)	Add with Carry to B	B + M + C ⇒ B	B IMM B DIR B EXT B IND,X B IND,Y	C9 D9 F9 E9 18 E9	ii dd hh ll ff ff	2 3 4 4 5	—	—	Δ	—	Δ	Δ	Δ	Δ
ADDA (opr)	Add Memory to A	A + M ⇒ A	A IMM A DIR A EXT A IND,X A IND,Y	8B 9B BB AB 18 AB	ii dd hh ll ff ff	2 3 4 4 5	—	—	Δ	—	Δ	Δ	Δ	Δ
ADDB (opr)	Add Memory to B	B + M ⇒ B	B IMM B DIR B EXT B IND,X B IND,Y	CB DB FB EB 18 EB	ii dd hh ll ff ff	2 3 4 4 5	—	—	Δ	—	Δ	Δ	Δ	Δ
ADDD (opr)	Add 16-Bit to D	D + (M : M + 1) ⇒ D	IMM DIR EXT IND,X IND,Y	C3 D3 F3 E3 18 E3	jj kk dd hh ll ff ff	4 5 6 6 7	—	—	—	—	Δ	Δ	Δ	Δ
ANDA (opr)	AND A with Memory	A • M ⇒ A	A IMM A DIR A EXT A IND,X A IND,Y	84 94 B4 A4 18 A4	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	0	—
ANDB (opr)	AND B with Memory	B • M ⇒ B	B IMM B DIR B EXT B IND,X B IND,Y	C4 D4 F4 E4 18 E4	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	0	—
ASL (opr)	Arithmetic Shift Left		EXT IND,X IND,Y	78 68 18 68	hh ll ff ff	6 6 7	—	—	—	—	Δ	Δ	Δ	Δ
ASLA	Arithmetic Shift Left A		A INH	48	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASLB	Arithmetic Shift Left B		B INH	58	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASLD	Arithmetic Shift Left D		INH	05	—	3	—	—	—	—	Δ	Δ	Δ	Δ
ASR	Arithmetic Shift Right		EXT IND,X IND,Y	77 67 18 67	hh ll ff ff	6 6 7	—	—	—	—	Δ	Δ	Δ	Δ
ASRA	Arithmetic Shift Right A		A INH	47	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASRB	Arithmetic Shift Right B		B INH	57	—	2	—	—	—	—	Δ	Δ	Δ	Δ
BCC (rel)	Branch if Carry Clear	? C = 0	REL	24	rr	3	—	—	—	—	—	—	—	—
BCLR (opr) (msk)	Clear Bit(s)	M • (mm) ⇒ M	DIR IND,X IND,Y	15 1D 18 1D	dd mm ff mm ff mm	6 7 8	—	—	—	—	Δ	Δ	0	—
BCS (rel)	Branch if Carry Set	? C = 1	REL	25	rr	3	—	—	—	—	—	—	—	—
BEQ (rel)	Branch if = Zero	? Z = 1	REL	27	rr	3	—	—	—	—	—	—	—	—
BGE (rel)	Branch if Δ Zero	? N ⊕ V = 0	REL	2C	rr	3	—	—	—	—	—	—	—	—
BGT (rel)	Branch if > Zero	? Z + (N ⊕ V) = 0	REL	2E	rr	3	—	—	—	—	—	—	—	—

Table 8 M68HC11 Instruction Set (Sheet 2 of 6)

Mnemonic	Operation	Description	Addressing Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	X	H	I	N	Z	V	C
BHI (rel)	Branch if Higher	? C + Z = 0	REL	22	rr	3	—	—	—	—	—	—	—	—
BHS (rel)	Branch if Higher or Same	? C = 0	REL	24	rr	3	—	—	—	—	—	—	—	—
BITA (opr)	Bit(s) Test A with Memory	A • M	A IMM A DIR A EXT A IND,X A IND,Y	85 95 B5 A5 18 A5	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	0	—
BITB (opr)	Bit(s) Test B with Memory	B • M	B IMM B DIR B EXT B IND,X B IND,Y	C5 D5 F5 E5 18 E5	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	0	—
BLE (rel)	Branch if Δ Zero	? Z + (N ⊕ V) = 1	REL	2F	rr	3	—	—	—	—	—	—	—	—
BLO (rel)	Branch if Lower	? C = 1	REL	25	rr	3	—	—	—	—	—	—	—	—
BLS (rel)	Branch if Lower or Same	? C + Z = 1	REL	23	rr	3	—	—	—	—	—	—	—	—
BLT (rel)	Branch if < Zero	? N ⊕ V = 1	REL	2D	rr	3	—	—	—	—	—	—	—	—
BMI (rel)	Branch if Minus	? N = 1	REL	2B	rr	3	—	—	—	—	—	—	—	—
BNE (rel)	Branch if not = Zero	? Z = 0	REL	26	rr	3	—	—	—	—	—	—	—	—
BPL (rel)	Branch if Plus	? N = 0	REL	2A	rr	3	—	—	—	—	—	—	—	—
BRA (rel)	Branch Always	? 1 = 1	REL	20	rr	3	—	—	—	—	—	—	—	—
BRCLR(opr) (msk) (rel)	Branch if Bit(s) Clear	? M • mm = 0	DIR IND,X IND,Y	13 1F 18 1F	dd mm rr ff mm rr ff mm rr	6 7 8	—	—	—	—	—	—	—	—
BRN (rel)	Branch Never	? 1 = 0	REL	21	rr	3	—	—	—	—	—	—	—	—
BRSET(opr) (msk) (rel)	Branch if Bit(s) Set	? (M) • mm = 0	DIR IND,X IND,Y	12 1E 18 1E	dd mm rr ff mm rr ff mm rr	6 7 8	—	—	—	—	—	—	—	—
BSET (opr) (msk)	Set Bit(s)	M + mm ⇒ M	DIR IND,X IND,Y	14 1C 18 1C	dd mm ff mm ff mm	6 7 8	—	—	—	—	Δ	Δ	0	—
BSR (rel)	Branch to Subroutine	See Figure 3–2	REL	8D	rr	6	—	—	—	—	—	—	—	—
BVC (rel)	Branch if Overflow Clear	? V = 0	REL	28	rr	3	—	—	—	—	—	—	—	—
BVS (rel)	Branch if Overflow Set	? V = 1	REL	29	rr	3	—	—	—	—	—	—	—	—
CBA	Compare A to B	A – B	INH	11	—	2	—	—	—	—	Δ	Δ	Δ	Δ
CLC	Clear Carry Bit	0 ⇒ C	INH	0C	—	2	—	—	—	—	—	—	—	0
CLI	Clear Interrupt Mask	0 ⇒ I	INH	0E	—	2	—	—	—	0	—	—	—	—
CLR (opr)	Clear Memory Byte	0 ⇒ M	EXT IND,X IND,Y	7F 6F 18 6F	hh ll ff ff	6 6 7	—	—	—	—	0	1	0	0
CLRA	Clear Accumulator A	0 ⇒ A	A INH	4F	—	2	—	—	—	—	0	1	0	0
CLRB	Clear Accumulator B	0 ⇒ B	B INH	5F	—	2	—	—	—	—	0	1	0	0
CLV	Clear Overflow Flag	0 ⇒ V	INH	0A	—	2	—	—	—	—	—	—	0	—
CMPA (opr)	Compare A to Memory	A – M	A IMM A DIR A EXT A IND,X A IND,Y	81 91 B1 A1 18 A1	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	Δ	Δ
CMPB (opr)	Compare B to Memory	B – M	B IMM B DIR B EXT B IND,X B IND,Y	C1 D1 F1 E1 18 E1	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	Δ	Δ
COM (opr)	Ones Complement Memory Byte	$FF – M ⇒ M	EXT IND,X IND,Y	73 63 18 63	hh ll ff ff	6 6 7	—	—	—	—	Δ	Δ	0	1

Table 8 M68HC11 Instruction Set (Sheet 3 of 6)

Mnemonic	Operation	Description	Addressing Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	X	H	I	N	Z	V	C
COMA	Ones Complement A	$FF – A ⇒ A	A INH	43	—	2	—	—	—	—	Δ	Δ	0	1
COMB	Ones Complement B	$FF – B ⇒ B	B INH	53	—	2	—	—	—	—	Δ	Δ	0	1
CPD (opr)	Compare D to Memory 16-Bit	D – M : M + 1	IMM DIR EXT IND,X IND,Y	1A 83 1A 93 1A B3 1A A3 CD A3	jj kk dd hh ll ff ff	5 6 7 7 7	—	—	—	—	Δ	Δ	Δ	Δ
CPX (opr)	Compare X to Memory 16-Bit	IX – M : M + 1	IMM DIR EXT IND,X IND,Y	8C 9C BC AC CD AC	jj kk dd hh ll ff ff	4 5 6 6 7	—	—	—	—	Δ	Δ	Δ	Δ
CPY (opr)	Compare Y to Memory 16-Bit	IY – M : M + 1	IMM DIR EXT IND,X IND,Y	18 8C 18 9C 18 BC 1A AC 18 AC	jj kk dd hh ll ff ff	5 6 7 7 7	—	—	—	—	Δ	Δ	Δ	Δ
DAA	Decimal Adjust A	Adjust Sum to BCD	INH	19	—	2	—	—	—	—	Δ	Δ	Δ	Δ
DEC (opr)	Decrement Memory Byte	M – 1 ⇒ M	EXT IND,X IND,Y	7A 6A 18 6A	hh ll ff ff	6 6 7	—	—	—	—	Δ	Δ	Δ	—
DECA	Decrement Accumulator A	A – 1 ⇒ A	A INH	4A	—	2	—	—	—	—	Δ	Δ	Δ	—
DECB	Decrement Accumulator B	B – 1 ⇒ B	B INH	5A	—	2	—	—	—	—	Δ	Δ	Δ	—
DES	Decrement Stack Pointer	SP – 1 ⇒ SP	INH	34	—	3	—	—	—	—	—	—	—	—
DEX	Decrement Index Register X	IX – 1 ⇒ IX	INH	09	—	3	—	—	—	—	—	Δ	—	—
DEY	Decrement Index Register Y	IY – 1 ⇒ IY	INH	18 09	—	4	—	—	—	—	—	Δ	—	—
EORA (opr)	Exclusive OR A with Memory	A ⊕ M ⇒ A	A IMM A DIR A EXT A IND,X A IND,Y	88 98 B8 A8 18 A8	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	0	—
EORB (opr)	Exclusive OR B with Memory	B ⊕ M ⇒ B	B IMM B DIR B EXT B IND,X B IND,Y	C8 D8 F8 E8 18 E8	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	0	—
FDIV	Fractional Divide 16 by 16	D / IX ⇒ IX; r ⇒ D	INH	03	—	41	—	—	—	—	—	Δ	Δ	Δ
IDIV	Integer Divide 16 by 16	D / IX ⇒ IX; r ⇒ D	INH	02	—	41	—	—	—	—	—	Δ	0	Δ
INC (opr)	Increment Memory Byte	M + 1 ⇒ M	EXT IND,X IND,Y	7C 6C 18 6C	hh ll ff ff	6 6 7	—	—	—	—	Δ	Δ	Δ	—
INCA	Increment Accumulator A	A + 1 ⇒ A	A INH	4C	—	2	—	—	—	—	Δ	Δ	Δ	—
INCB	Increment Accumulator B	B + 1 ⇒ B	B INH	5C	—	2	—	—	—	—	Δ	Δ	Δ	—
INS	Increment Stack Pointer	SP + 1 ⇒ SP	INH	31	—	3	—	—	—	—	—	—	—	—
INX	Increment Index Register X	IX + 1 ⇒ IX	INH	08	—	3	—	—	—	—	—	Δ	—	—
INY	Increment Index Register Y	IY + 1 ⇒ IY	INH	18 08	—	4	—	—	—	—	—	Δ	—	—
JMP (opr)	Jump	See Figure 3–2	EXT IND,X IND,Y	7E 6E 18 6E	hh ll ff ff	3 3 4	—	—	—	—	—	—	—	—

Table 8 M68HC11 Instruction Set (Sheet 4 of 6)

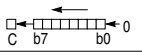
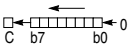
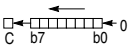
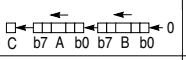
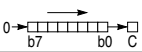
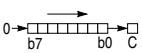
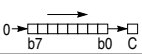
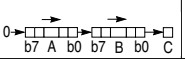
Mnemonic	Operation	Description	Addressing Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	X	H	I	N	Z	V	C
JSR (opr)	Jump to Subroutine	See Figure 3–2	DIR	9D	dd	5	—	—	—	—	—	—	—	—
			EXT	BD	hh ll	6								
			IND,X	AD	ff	6								
			IND,Y	18 AD	ff	7								
LDAA (opr)	Load Accumulator A	M ⇒ A	A IMM	86	ii	2	—	—	—	—	Δ	Δ	0	—
			A DIR	96	dd	3								
			A EXT	B6	hh ll	4								
			A IND,X	A6	ff	4								
			A IND,Y	18 A6	ff	5								
LDAB (opr)	Load Accumulator B	M ⇒ B	B IMM	C6	ii	2	—	—	—	—	Δ	Δ	0	—
			B DIR	D6	dd	3								
			B EXT	F6	hh ll	4								
			B IND,X	E6	ff	4								
			B IND,Y	18 E6	ff	5								
LDD (opr)	Load Double Accumulator D	M ⇒ A,M + 1 ⇒ B	IMM	CC	jj kk	3	—	—	—	—	Δ	Δ	0	—
			DIR	DC	dd	4								
			EXT	FC	hh ll	5								
			IND,X	EC	ff	5								
			IND,Y	18 EC	ff	6								
LDS (opr)	Load Stack Pointer	M : M + 1 ⇒ SP	IMM	8E	jj kk	3	—	—	—	—	Δ	Δ	0	—
			DIR	9E	dd	4								
			EXT	BE	hh ll	5								
			IND,X	AE	ff	5								
			IND,Y	18 AE	ff	6								
LDX (opr)	Load Index Register X	M : M + 1 ⇒ IX	IMM	CE	jj kk	3	—	—	—	—	Δ	Δ	0	—
			DIR	DE	dd	4								
			EXT	FE	hh ll	5								
			IND,X	EE	ff	5								
			IND,Y	CD EE	ff	6								
LDY (opr)	Load Index Register Y	M : M + 1 ⇒ IY	IMM	18 CE	jj kk	4	—	—	—	—	Δ	Δ	0	—
			DIR	18 DE	dd	5								
			EXT	18 FE	hh ll	6								
			IND,X	1A EE	ff	6								
			IND,Y	18 EE	ff	6								
LSL (opr)	Logical Shift Left		EXT	78	hh ll	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND,X	68	ff	6								
			IND,Y	18 68	ff	7								
LSLA	Logical Shift Left A		A INH	48	—	2	—	—	—	—	Δ	Δ	Δ	Δ
LSLB	Logical Shift Left B		B INH	58	—	2	—	—	—	—	Δ	Δ	Δ	Δ
LSLD	Logical Shift Left Double		INH	05	—	3	—	—	—	—	Δ	Δ	Δ	Δ
LSR (opr)	Logical Shift Right		EXT	74	hh ll	6	—	—	—	—	0	Δ	Δ	Δ
			IND,X	64	ff	6								
			IND,Y	18 64	ff	7								
LSRA	Logical Shift Right A		A INH	44	—	2	—	—	—	—	0	Δ	Δ	Δ
LSRB	Logical Shift Right B		B INH	54	—	2	—	—	—	—	0	Δ	Δ	Δ
LSRD	Logical Shift Right Double		INH	04	—	3	—	—	—	—	0	Δ	Δ	Δ
MUL	Multiply 8 by 8	A * B ⇒ D	INH	3D	—	10	—	—	—	—	—	—	—	Δ
NEG (opr)	Two's Complement Memory Byte	0 – M ⇒ M	EXT	70	hh ll	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND,X	60	ff	6								
			IND,Y	18 60	ff	7								
NEGA	Two's Complement A	0 – A ⇒ A	A INH	40	—	2	—	—	—	—	Δ	Δ	Δ	Δ
NEGB	Two's Complement B	0 – B ⇒ B	B INH	50	—	2	—	—	—	—	Δ	Δ	Δ	Δ
NOP	No operation	No Operation	INH	01	—	2	—	—	—	—	—	—	—	—
ORAA (opr)	OR Accumulator A (Inclusive)	A + M ⇒ A	A IMM	8A	ii	2	—	—	—	—	Δ	Δ	0	—
			A DIR	9A	dd	3								
			A EXT	BA	hh ll	4								
			A IND,X	AA	ff	4								
			A IND,Y	18 AA	ff	5								

Table 8 M68HC11 Instruction Set (Sheet 5 of 6)

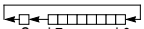
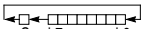
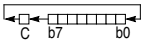
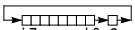
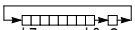
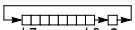
Mnemonic	Operation	Description	Addressing Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	X	H	I	N	Z	V	C
ORAB (opr)	OR Accumulator B (Inclusive)	B + M ⇒ B	B IMM B DIR B EXT B IND,X B IND,Y	CA DA FA EA 18 EA	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	0	—
PSHA	Push A onto Stack	A ⇒ Stk,SP = SP – 1	A INH	36	—	3	—	—	—	—	—	—	—	—
PSHB	Push B onto Stack	B ⇒ Stk,SP = SP – 1	B INH	37	—	3	—	—	—	—	—	—	—	—
PSHX	Push X onto Stack (Lo First)	IX ⇒ Stk,SP = SP – 2	INH	3C	—	4	—	—	—	—	—	—	—	—
PSHY	Push Y onto Stack (Lo First)	IY ⇒ Stk,SP = SP – 2	INH	18 3C	—	5	—	—	—	—	—	—	—	—
PULA	Pull A from Stack	SP = SP + 1, A ⇐ Stk	A INH	32	—	4	—	—	—	—	—	—	—	—
PULB	Pull B from Stack	SP = SP + 1, B ⇐ Stk	B INH	33	—	4	—	—	—	—	—	—	—	—
PULX	Pull X From Stack (Hi First)	SP = SP + 2, IX ⇐ Stk	INH	38	—	5	—	—	—	—	—	—	—	—
PULY	Pull Y from Stack (Hi First)	SP = SP + 2, IY ⇐ Stk	INH	18 38	—	6	—	—	—	—	—	—	—	—
ROL (opr)	Rotate Left		EXT IND,X IND,Y	79 69 18 69	hh ll ff ff	6 6 7	—	—	—	—	Δ	Δ	Δ	Δ
ROLA	Rotate Left A		A INH	49	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ROLB	Rotate Left B		B INH	59	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ROR (opr)	Rotate Right		EXT IND,X IND,Y	76 66 18 66	hh ll ff ff	6 6 7	—	—	—	—	Δ	Δ	Δ	Δ
RORA	Rotate Right A		A INH	46	—	2	—	—	—	—	Δ	Δ	Δ	Δ
RORB	Rotate Right B		B INH	56	—	2	—	—	—	—	Δ	Δ	Δ	Δ
RTI	Return from Interrupt	See Figure 3–2	INH	3B	—	12	Δ	↓	Δ	Δ	Δ	Δ	Δ	Δ
RTS	Return from Subroutine	See Figure 3–2	INH	39	—	5	—	—	—	—	—	—	—	—
SBA	Subtract B from A	A – B ⇒ A	INH	10	—	2	—	—	—	—	Δ	Δ	Δ	Δ
SBCA (opr)	Subtract with Carry from A	A – M – C ⇒ A	A IMM A DIR A EXT A IND,X A IND,Y	82 92 B2 A2 18 A2	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	Δ	Δ
SBCB (opr)	Subtract with Carry from B	B – M – C ⇒ B	B IMM B DIR B EXT B IND,X B IND,Y	C2 D2 F2 E2 18 E2	ii dd hh ll ff ff	2 3 4 4 5	—	—	—	—	Δ	Δ	Δ	Δ
SEC	Set Carry	1 ⇒ C	INH	0D	—	2	—	—	—	—	—	—	—	1
SEI	Set Interrupt Mask	1 ⇒ I	INH	0F	—	2	—	—	—	1	—	—	—	—
SEV	Set Overflow Flag	1 ⇒ V	INH	0B	—	2	—	—	—	—	—	—	1	—
STAA (opr)	Store Accumulator A	A ⇒ M	A DIR A EXT A IND,X A IND,Y	97 B7 A7 18 A7	dd hh ll ff ff	3 4 4 5	—	—	—	—	Δ	Δ	0	—
STAB (opr)	Store Accumulator B	B ⇒ M	B DIR B EXT B IND,X B IND,Y	D7 F7 E7 18 E7	dd hh ll ff ff	3 4 4 5	—	—	—	—	Δ	Δ	0	—
STD (opr)	Store Accumulator D	A ⇒ M, B ⇒ M + 1	DIR EXT IND,X IND,Y	DD FD ED 18 ED	dd hh ll ff ff	4 5 5 6	—	—	—	—	Δ	Δ	0	—

Table 8 M68HC11 Instruction Set (Sheet 6 of 6)

Mnemonic	Operation	Description	Addressing Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	X	H	I	N	Z	V	C
STOP	Stop Internal Clocks	—	INH	CF	—	2	—	—	—	—	—	—	—	—
STS (opr)	Store Stack Pointer	SP ⇒ M : M + 1	DIR	9F	dd	4	—	—	—	—	Δ	Δ	0	—
			EXT	BF	hh ll	5								
			IND,X	AF	ff	5								
			IND,Y	18 AF	ff	6								
STX (opr)	Store Index Register X	IX ⇒ M : M + 1	DIR	DF	dd	4	—	—	—	—	Δ	Δ	0	—
			EXT	FF	hh ll	5								
			IND,X	EF	ff	5								
			IND,Y	CD EF	ff	6								
STY (opr)	Store Index Register Y	IY ⇒ M : M + 1	DIR	18 DF	dd	5	—	—	—	—	Δ	Δ	0	—
			EXT	18 FF	hh ll	6								
			IND,X	1A EF	ff	6								
			IND,Y	18 EF	ff	6								
SUBA (opr)	Subtract Memory from A	A – M ⇒ A	A IMM	80	ii	2	—	—	—	—	Δ	Δ	Δ	Δ
			A DIR	90	dd	3								
			A EXT	B0	hh ll	4								
			A IND,X	A0	ff	4								
			A IND,Y	18 A0	ff	5								
SUBB (opr)	Subtract Memory from B	B – M ⇒ B	A IMM	C0	ii	2	—	—	—	—	Δ	Δ	Δ	Δ
			A DIR	D0	dd	3								
			A EXT	F0	hh ll	4								
			A IND,X	E0	ff	4								
			A IND,Y	18 E0	ff	5								
SUBD (opr)	Subtract Memory from D	D – M : M + 1 ⇒ D	IMM	83	jj kk	4	—	—	—	—	Δ	Δ	Δ	Δ
			DIR	93	dd	5								
			EXT	B3	hh ll	6								
			IND,X	A3	ff	6								
			IND,Y	18 A3	ff	7								
SWI	Software Interrupt	See Figure 3–2	INH	3F	—	14	—	—	—	1	—	—	—	—
TAB	Transfer A to B	A ⇒ B	INH	16	—	2	—	—	—	—	Δ	Δ	0	—
TAP	Transfer A to CC Register	A ⇒ CCR	INH	06	—	2	Δ	↓	Δ	Δ	Δ	Δ	Δ	Δ
TBA	Transfer B to A	B ⇒ A	INH	17	—	2	—	—	—	—	Δ	Δ	0	—
TEST	TEST (Only in Test Modes)	Address Bus Counts	INH	00	—	*	—	—	—	—	—	—	—	—
TPA	Transfer CC Register to A	CCR ⇒ A	INH	07	—	2	—	—	—	—	—	—	—	—
TST (opr)	Test for Zero or Minus	M – 0	EXT	7D	hh ll	6	—	—	—	—	Δ	Δ	0	0
			IND,X	6D	ff	6								
			IND,Y	18 6D	ff	7								
TSTA	Test A for Zero or Minus	A – 0	A INH	4D	—	2	—	—	—	—	Δ	Δ	0	0
TSTB	Test B for Zero or Minus	B – 0	B INH	5D	—	2	—	—	—	—	Δ	Δ	0	0
TSX	Transfer Stack Pointer to X	SP + 1 ⇒ IX	INH	30	—	3	—	—	—	—	—	—	—	—
TSY	Transfer Stack Pointer to Y	SP + 1 ⇒ IY	INH	18 30	—	4	—	—	—	—	—	—	—	—
TXS	Transfer X to Stack Pointer	IX – 1 ⇒ SP	INH	35	—	3	—	—	—	—	—	—	—	—
TYS	Transfer Y to Stack Pointer	IY – 1 ⇒ SP	INH	18 35	—	4	—	—	—	—	—	—	—	—
WAI	Wait for Interrupt	Stack Regs & WAIT	INH	3E	—	**	—	—	—	—	—	—	—	—
XGDX	Exchange D with X	IX ⇒ D, D ⇒ IX	INH	8F	—	3	—	—	—	—	—	—	—	—
XGDY	Exchange D with Y	IY ⇒ D, D ⇒ IY	INH	18 8F	—	4	—	—	—	—	—	—	—	—

Table 9 CPU16 Instruction Set Summary (Sheet 1 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
ABA	Add B to A	(A) + (B) ⇒ A	INH	370B	—	2	—	—	Δ	—	Δ	Δ	Δ	Δ
ABX	Add B to X	(XK : IX) + (000 : B) ⇒ XK : IX	INH	374F	—	2	—	—	—	—	—	—	—	—
ABY	Add B to Y	(YK : IY) + (000 : B) ⇒ YK : IY	INH	375F	—	2	—	—	—	—	—	—	—	—
ABZ	Add B to Z	(ZK : IZ) + (000 : B) ⇒ ZK : IZ	INH	376F	—	2	—	—	—	—	—	—	—	—
ACE	Add E to AM[31:15]	(AM[31:15]) + (E) ⇒ AM	INH	3722	—	2	—	Δ	—	Δ	—	—	—	—
ACED	Add concatenated E and D to AM	(E : D) + (AM) ⇒ AM	INH	3723	—	4	—	Δ	—	Δ	—	—	—	—
ADCA	Add with Carry to A	(A) + (M) + C ⇒ A	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	43 53 63 73 1743 1753 1763 1773 2743 2753 2763	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	Δ	—	Δ	Δ	Δ	Δ
ADCB	Add with Carry to B	(B) + (M) + C ⇒ B	IND8, X IND8, Y IND8, Z IMM8 E, X E, Y E, Z IND16, X IND16, Y IND16, Z EXT	C3 D3 E3 F3 27C3 27D3 27E3 17C3 17D3 17E3 17F3	ff ff ff ii — — — gggg gggg gggg hh ll	6 6 6 2 6 6 6 6 6 6 6	—	—	Δ	—	Δ	Δ	Δ	Δ
ADCD	Add with Carry to D	(D) + (M : M + 1) + C ⇒ D	IND8, X IND8, Y IND8, Z E, X E, Y E, Z IMM16 IND16, X IND16, Y IND16, Z EXT	83 93 A3 2783 2793 27A3 37B3 37C3 37D3 37E3 37F3	ff ff ff — — — jj kk gggg gggg gggg hh ll	6 6 6 6 6 6 4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
ADCE	Add with Carry to E	(E) + (M : M + 1) + C ⇒ E	IMM16 IND16, X IND16, Y IND16, Z EXT	3733 3743 3753 3763 3773	jj kk gggg gggg gggg hh ll	4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
ADDA	Add to A	(A) + (M) ⇒ A	IND8, X IND8, Y IND8, Z IMM8 E, X E, Y E, Z IND16, X IND16, Y IND16, Z EXT	41 51 61 71 2741 2751 2761 1741 1751 1761 1771	ff ff ff ii — — — gggg gggg gggg hh ll	6 6 6 2 6 6 6 6 6 6 6	—	—	Δ	—	Δ	Δ	Δ	Δ
ADDB	Add to B	(B) + (M) ⇒ B	IND8, X IND8, Y IND8, Z IMM8 E, X E, Y E, Z IND16, X IND16, Y IND16, Z EXT	C1 D1 E1 F1 27C1 27D1 27E1 17C1 17D1 17E1 17F1	ff ff ff ii — — — gggg gggg gggg hh ll	6 6 6 2 6 6 6 6 6 6 6	—	—	Δ	—	Δ	Δ	Δ	Δ

Table 9 CPU16 Instruction Set Summary (Sheet 2 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction Opcode	Instruction Operand	Instruction Cycles	Condition Codes S	MV	H	EV	N	Z	V	C
ADDD	Add to D	(D) + (M : M + 1) ⇒ D	IND8, X	81	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	91	ff	6								
			IND8, Z	A1	ff	6								
			IMM8	FC	ii	2								
			E, X	2781	—	6								
			E, Y	2791	—	6								
			E, Z	27A1	—	6								
			IMM16	37B1	jjkk	4								
			IND16, X	37C1	gggg	6								
			IND16, Y	37D1	gggg	6								
			IND16, Z	37E1	gggg	6								
			EXT	37F1	hh ll	6								
ADDE	Add to E	(E) + (M : M + 1) ⇒ E	IMM8	7C	ii	2	—	—	—	—	Δ	Δ	Δ	Δ
			IMM16	3731	jj kk	4								
			IND16, X	3741	gggg	6								
			IND16, Y	3751	gggg	6								
			IND16, Z	3761	gggg	6								
			EXT	3771	hh ll	6								
ADE	Add D to E	(E) + (D) ⇒ E	INH	2778	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ADX	Add D to X	(XK : IX) + («D) ⇒ XK : IX	INH	37CD	—	2	—	—	—	—	—	—	—	—
ADY	Add D to Y	(YK : IY) + («D) ⇒ YK : IY	INH	37DD	—	2	—	—	—	—	—	—	—	—
ADZ	Add D to Z	(ZK : IZ) + («D) ⇒ ZK : IZ	INH	37ED	—	2	—	—	—	—	—	—	—	—
AEX	Add E to X	(XK : IX) + («E) ⇒ XK : IX	INH	374D	—	2	—	—	—	—	—	—	—	—
AEY	Add E to Y	(YK : IY) + («E) ⇒ YK : IY	INH	375D	—	2	—	—	—	—	—	—	—	—
AEZ	Add E to Z	(ZK : IZ) + («E) ⇒ ZK : IZ	INH	376D	—	2	—	—	—	—	—	—	—	—
AIS	Add Immediate Data to SP	SK : SP + «IMM ⇒ SK : SP	IMM8	3F	ii	2	—	—	—	—	—	—	—	—
			IMM16	373F	jj kk	4								
AIX	Add Immediate Value to X	XK : IX + «IMM ⇒ XK : IX	IMM8	3C	ii	2	—	—	—	—	—	Δ	—	—
			IMM16	373C	jj kk	4								
AIY	Add Immediate Value to Y	YK : IY + «IMM ⇒ YK : IY	IMM8	3D	ii	2	—	—	—	—	—	Δ	—	—
			IMM16	373D	jj kk	4								
AIZ	Add Immediate Value to Z	ZK : IZ + «IMM ⇒ ZK : IZ	IMM8	3E	ii	2	—	—	—	—	—	Δ	—	—
			IMM16	373E	jj kk	4								
ANDA	AND A	(A) • (M) ⇒ A	IND8, X	46	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	56	ff	6								
			IND8, Z	66	ff	6								
			IMM8	76	ii	2								
			IND16, X	1746	gggg	6								
			IND16, Y	1756	gggg	6								
			IND16, Z	1766	gggg	6								
			EXT	1776	hh ll	6								
			E, X	2746	—	6								
			E, Y	2756	—	6								
			E, Z	2766	—	6								
ANDB	AND B	(B) • (M) ⇒ B	IND8, X	C6	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	D6	ff	6								
			IND8, Z	E6	ff	6								
			IMM8	F6	ii	2								
			IND16, X	17C6	gggg	6								
			IND16, Y	17D6	gggg	6								
			IND16, Z	17E6	gggg	6								
			EXT	17F6	hh ll	6								
			E, X	27C6	—	6								
			E, Y	27D6	—	6								
			E, Z	27E6	—	6								
ANDD	AND D	(D) • (M : M + 1) ⇒ D	IND8, X	86	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	96	ff	6								
			IND8, Z	A6	ff	6								
			E, X	2786	—	6								
			E, Y	2796	—	6								
			E, Z	27A6	—	6								
			IMM16	37B6	jj kk	4								
			IND16, X	37C6	gggg	6								
			IND16, Y	37D6	gggg	6								
			IND16, Z	37E6	gggg	6								
			EXT	37F6	hh ll	6								
ANDE	AND E	(E) • (M : M + 1) ⇒ E	IMM16	3736	jj kk	4	—	—	—	—	Δ	Δ	0	—
			IND16, X	3746	gggg	6								
			IND16, Y	3756	gggg	6								
			IND16, Z	3766	gggg	6								
			EXT	3776	hh ll	6								
ANDP[1]	AND CCR	(CCR) • IMM16⇒ CCR	IMM16	373A	jj kk	4	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ

Table 9 CPU16 Instruction Set Summary (Sheet 3 of 15)

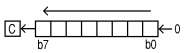
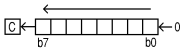
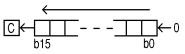
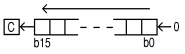
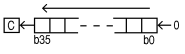
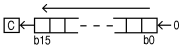
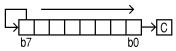
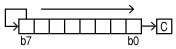
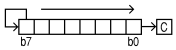
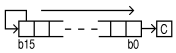
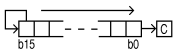
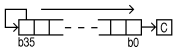
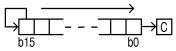
Mnemonic	Operation	Description	Address Mode	Instruction Opcode	Instruction Operand	Instruction Cycles	Condition Codes S	MV	H	EV	N	Z	V	C
ASL	Arithmetic Shift Left		IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	04 14 24 1704 1714 1724 1734	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
ASLA	Arithmetic Shift Left A		INH	3704	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASLB	Arithmetic Shift Left B		INH	3714	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASLD	Arithmetic Shift Left D		INH	27F4	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASLE	Arithmetic Shift Left E		INH	2774	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASLM	Arithmetic Shift Left AM		INH	27B6	—	4	—	Δ	—	Δ	Δ	—	—	Δ
ASLW	Arithmetic Shift Left Word		IND16, X IND16, Y IND16, Z EXT	2704 2714 2724 2734	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
ASR	Arithmetic Shift Right		IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	0D 1D 2D 170D 171D 172D 173D	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
ASRA	Arithmetic Shift Right A		INH	370D	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASRB	Arithmetic Shift Right B		INH	371D	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASRD	Arithmetic Shift Right D		INH	27FD	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASRE	Arithmetic Shift Right E		INH	277D	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ASRM	Arithmetic Shift Right AM		INH	27BA	—	4	—	—	—	Δ	Δ	—	—	Δ
ASRW	Arithmetic Shift Right Word		IND16, X IND16, Y IND16, Z EXT	270D 271D 272D 273D	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
BCC[4]	Branch if Carry Clear	If C = 0, branch	REL8	B4	rr	6, 2	—	—	—	—	—	—	—	—
BCLR	Clear Bit(s)	(M) • ($\overline{\text{Mask}}$) ⇒ M	IND16, X IND16, Y IND16, Z EXT IND8, X IND8, Y IND8, Z	08 18 28 38 1708 1718 1728	mm gggg mm gggg mm gggg mm hh ll mm ff mm ff mm ff	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	0	—
BCLRW	Clear Bit(s) Word	(M : M + 1) • ($\overline{\text{Mask}}$) ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2708 2718 2728 2738	gggg mmmm gggg mmmm gggg mmmm hh ll mmmm	10 10 10 10	—	—	—	—	Δ	Δ	0	—

Table 9 CPU16 Instruction Set Summary (Sheet 4 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction Opcode	Instruction Operand	Instruction Cycles	Condition Codes S	MV	H	EV	N	Z	V	C
BCS[4]	Branch if Carry Set	If C = 1, branch	REL8	B5	rr	6, 2	—	—	—	—	—	—	—	—
BEQ[4]	Branch if Equal	If Z = 1, branch	REL8	B7	rr	6, 2	—	—	—	—	—	—	—	—
BGE[4]	Branch if Greater Than or Equal to Zero	If N ⊕ V = 0, branch	REL8	BC	rr	6, 2	—	—	—	—	—	—	—	—
BGND	Enter Background Debug Mode	If BDM enabled enter BDM; else, illegal instruction	INH	37A6	—	—	—	—	—	—	—	—	—	—
BGT[4]	Branch if Greater Than Zero	If Z + (N ⊕ V) = 0, branch	REL8	BE	rr	6, 2	—	—	—	—	—	—	—	—
BHI[4]	Branch if Higher	If C + Z = 0, branch	REL8	B2	rr	6, 2	—	—	—	—	—	—	—	—
BITA	Bit Test A	(A) • (M)	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	49 59 69 79 1749 1759 1769 1779 2749 2759 2769	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
BITB	Bit Test B	(B) • (M)	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C9 D9 E9 F9 17C9 17D9 17E9 17F9 27C9 27D9 27E9	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
BLE[4]	Branch if Less Than or Equal to Zero	If Z + (N ⊕ V) = 1, branch	REL8	BF	rr	6, 2	—	—	—	—	—	—	—	—
BLS[4]	Branch if Lower or Same	If C + Z = 1, branch	REL8	B3	rr	6, 2	—	—	—	—	—	—	—	—
BLT[4]	Branch if Less Than Zero	If N ⊕ V = 1, branch	REL8	BD	rr	6, 2	—	—	—	—	—	—	—	—
BMI[4]	Branch if Minus	If N = 1, branch	REL8	BB	rr	6, 2	—	—	—	—	—	—	—	—
BNE[4]	Branch if Not Equal	If Z = 0, branch	REL8	B6	rr	6, 2	—	—	—	—	—	—	—	—
BPL[4]	Branch if Plus	If N = 0, branch	REL8	BA	rr	6, 2	—	—	—	—	—	—	—	—
BRA	Branch Always	If 1 = 1, branch	REL8	B0	rr	6	—	—	—	—	—	—	—	—
BRCLR[4]	Branch if Bit(s) Clear	If (M) • (Mask) = 0, branch	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	CB DB EB 0A 1A 2A 3A	mm ff rr mm ff rr mm ff rr mm gggg rrrr mm gggg rrrr mm gggg rrrr mm hh ll rrrr	10, 12 10, 12 10, 12 10, 14 10, 14 10, 14 10, 14	—	—	—	—	—	—	—	—
BRN	Branch Never	If 1 = 0, branch	REL8	B1	rr	2	—	—	—	—	—	—	—	—
BRSET[4]	Branch if Bit(s) Set	If (M) • (Mask) = 0, branch	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	8B 9B AB 0B 1B 2B 3B	mm ff rr mm ff rr mm ff rr mm gggg rrrr mm gggg rrrr mm gggg rrrr mm hh ll rrrr	10, 12 10, 12 10, 12 10, 14 10, 14 10, 14 10, 14	—	—	—	—	—	—	—	—
BSET	Set Bit(s)	(M) • (Mask) ⇒ M	IND16, X IND16, Y IND16, Z EXT IND8, X IND8, Y IND8, Z	09 19 29 39 1709 1719 1729	mm gggg mm gggg mm gggg mm hh ll mm ff mm ff mm ff	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	0	—

Table 9 CPU16 Instruction Set Summary (Sheet 5 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction Opcode	Instruction Operand	Instruction Cycles	Condition Codes S	MV	H	EV	N	Z	V	C
BSETW	Set Bit(s) in Word	(M : M + 1) • (Mask) ⇒ M : M + 1	IND16, X	2709	gggg mmmm	10	—	—	—	—	Δ	Δ	0	—
			IND16, Y	2719	gggg mmmm	10								
			IND16, Z	2729	gggg mmmm	10								
			EXT	2739	hh ll mmmm	10								
BSR	Branch to Subroutine	(PK : PC) – 2 ⇒ PK : PC Push (PC) (SK : SP) – 2 ⇒ SK : SP Push (CCR) (SK : SP) – 2 ⇒ SK : SP (PK:PC) + Offset ⇒ PK:PC	REL8	36	rr	10	—	—	—	—	—	—	—	—
BVC[4]	Branch if Overflow Clear	If V = 0, branch	REL8	B8	rr	6, 2	—	—	—	—	—	—	—	—
BVS[4]	Branch if Overflow Set	If V = 1, branch	REL8	B9	rr	6, 2	—	—	—	—	—	—	—	—
CBA	Compare A to B	(A) – (B)	INH	371B	—	2	—	—	—	—	Δ	Δ	Δ	Δ
CLR	Clear Memory	$00 ⇒ M	IND8, X	05	ff	4	—	—	—	—	0	1	0	0
			IND8, Y	15	ff	4								
			IND8, Z	25	ff	4								
			IND16, X	1705	gggg	6								
			IND16, Y	1715	gggg	6								
			IND16, Z	1725	gggg	6								
			EXT	1735	hh ll	6								
CLRA	Clear A	$00 ⇒ A	INH	3705	—	2	—	—	—	—	0	1	0	0
CLRB	Clear B	$00 ⇒ B	INH	3715	—	2	—	—	—	—	0	1	0	0
CLRD	Clear D	$0000 ⇒ D	INH	27F5	—	2	—	—	—	—	0	1	0	0
CLRE	Clear E	$0000 ⇒ E	INH	2775	—	2	—	—	—	—	0	1	0	0
CLRM	Clear AM	$000000000 ⇒ AM[32:0]	INH	27B7	—	2	—	0	—	0	—	—	—	—
CLRW	Clear Memory Word	$0000 ⇒ M : M + 1	IND16, X	2705	gggg	6	—	—	—	—	0	1	0	0
			IND16, Y	2715	gggg	6								
			IND16, Z	2725	gggg	6								
			EXT	2735	hh ll	6								
CMPA	Compare A to Memory	(A) – (M)	IND8, X	48	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	58	ff	6								
			IND8, Z	68	ff	6								
			IMM8	78	ii	2								
			IND16, X	1748	gggg	6								
			IND16, Y	1758	gggg	6								
			IND16, Z	1768	gggg	6								
			EXT	1778	hh ll	6								
			E, X	2748	—	6								
			E, Y	2758	—	6								
			E, Z	2768	—	6								
CMPB	Compare B to Memory	(B) – (M)	IND8, X	C8	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	D8	ff	6								
			IND8, Z	E8	ff	6								
			IMM8	F8	ii	2								
			IND16, X	17C8	gggg	6								
			IND16, Y	17D8	gggg	6								
			IND16, Z	17E8	gggg	6								
			EXT	17F8	hh ll	6								
			E, X	27C8	—	6								
			E, Y	27D8	—	6								
			E, Z	27E8	—	6								
COM	One's Complement	$FF – (M) ⇒ M	IND8, X	00	ff	8	—	—	—	—	Δ	Δ	0	1
			IND8, Y	10	ff	8								
			IND8, Z	20	ff	8								
			IND16, X	1700	gggg	8								
			IND16, Y	1710	gggg	8								
			IND16, Z	1720	gggg	8								
			EXT	1730	hh ll	8								
COMA	One's Complement A	$FF – (A) ⇒ A	INH	3700	—	2	—	—	—	—	Δ	Δ	0	1
COMB	One's Complement B	$FF – (B) ⇒ B	INH	3710	—	2	—	—	—	—	Δ	Δ	0	1
COMD	One's Complement D	$FFFF – (D) ⇒ D	INH	27F0	—	2	—	—	—	—	Δ	Δ	0	1
COME	One's Complement E	$FFFF – (E) ⇒ E	INH	2770	—	2	—	—	—	—	Δ	Δ	0	1
COMW	One's Complement Word	$FFFF – M : M + 1 ⇒ M : M + 1	IND16, X	2700	gggg	8	—	—	—	—	Δ	Δ	0	1
			IND16, Y	2710	gggg	8								
			IND16, Z	2720	gggg	8								
			EXT	2730	hh ll	8								

Table 9 CPU16 Instruction Set Summary (Sheet 6 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
CPD	Compare D to Memory	(D) – (M : M + 1)	IND8, X IND8, Y IND8, Z E, X E, Y E, Z IMM16 IND16, X IND16, Y IND16, Z EXT	88 98 A8 2788 2798 27A8 37B8 37C8 37D8 37E8 37F8	ff ff ff — — — jj kk gggg gggg gggg hh ll	6 6 6 6 6 6 4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
CPE	Compare E to Memory	(E) – (M : M + 1)	IMM16 IND16, X IND16, Y IND16, Z EXT	3738 3748 3758 3768 3778	jjkk gggg gggg gggg hhll	4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
CPS	Compare SP to Memory	(SP) – (M : M + 1)	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT IMM16	4F 5F 6F 174F 175F 176F 177F 377F	ff ff ff gggg gggg gggg hh ll jj kk	6 6 6 6 6 6 6 4	—	—	—	—	Δ	Δ	Δ	Δ
CPX	Compare IX to Memory	(IX) – (M : M + 1)	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT IMM16	4C 5C 6C 174C 175C 176C 177C 377C	ff ff ff gggg gggg gggg hh ll jj kk	6 6 6 6 6 6 6 4	—	—	—	—	Δ	Δ	Δ	Δ
CPY	Compare IY to Memory	(IY) – (M : M + 1)	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT IMM16	4D 5D 6D 174D 175D 176D 177D 377D	ff ff ff gggg gggg gggg hh ll jj kk	6 6 6 6 6 6 6 4	—	—	—	—	Δ	Δ	Δ	Δ
CPZ	Compare IZ to Memory	(IZ) – (M : M + 1)	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT IMM16	4E 5E 6E 174E 175E 176E 177E 377E	ff ff ff gggg gggg gggg hh ll jj kk	6 6 6 6 6 6 6 4	—	—	—	—	Δ	Δ	Δ	Δ
DAA	Decimal Adjust A	$(A)_{10}$	INH	3721	—	2	—	—	—	—	Δ	Δ	U	Δ
DEC	Decrement Memory	(M) – $01 ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	01 11 21 1701 1711 1721 1731	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	—
DECA	Decrement A	(A) – $01 ⇒ A	INH	3701	—	2	—	—	—	—	Δ	Δ	Δ	—
DECB	Decrement B	(B) – $01 ⇒ B	INH	3711	—	2	—	—	—	—	Δ	Δ	Δ	—
DECW	Decrement Memory Word	(M : M + 1) – $0001 ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2701 2711 2721 2731	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	—
EDIV	Extended Unsigned Divide	(E : D) / (IX) Quotient ⇒ IX Remainder ⇒ D	INH	3728	—	24	—	—	—	—	Δ	Δ	Δ	Δ
EDIVS	Extended Signed Divide	(E : D) / (IX) Quotient ⇒ IX Remainder ⇒ ACCD	INH	3729	—	38	—	—	—	—	Δ	Δ	Δ	Δ
EMUL	Extended Unsigned Multiply	(E) * (D) ⇒ E : D	INH	3725	—	10	—	—	—	—	Δ	Δ	—	Δ
EMULS	Extended Signed Multiply	(E) * (D) ⇒ E : D	INH	3726	—	8	—	—	—	—	Δ	Δ	—	Δ

Table 9 CPU16 Instruction Set Summary (Sheet 7 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
EORA	Exclusive OR A	(A) ⊕ (M) ⇒ A	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	44 54 64 74 1744 1754 1764 1774 2744 2754 2764	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
EORB	Exclusive OR B	(B) ⊕ (M) ⇒ B	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C4 D4 E4 F4 17C4 17D4 17E4 17F4 27C4 27D4 27E4	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
EORD	Exclusive OR D	(D) ⊕ (M : M + 1) ⇒ D	IND8, X IND8, Y IND8, Z E, X E, Y E, Z IMM16 IND16, X IND16, Y IND16, Z EXT	84 94 A4 2784 2794 27A4 37B4 37C4 37D4 37E4 37F4	ff ff ff — — — jjkk gggg gggg gggg hhll	6 6 6 6 6 6 4 6 6 6 6	—	—	—	—	Δ	Δ	0	—
EORE	Exclusive OR E	(E) ⊕ (M : M + 1) ⇒ E	IMM16 IND16, X IND16, Y IND16, Z EXT	3734 3744 3754 3764 3774	jj kk gggg gggg gggg hh ll	4 6 6 6 6	—	—	—	—	Δ	Δ	0	—
FDIV	Fractional Unsigned Divide	(D) / (IX) ⇒ IX Remainder ⇒ D	INH	372B	—	22	—	—	—	—	—	Δ	Δ	Δ
FMULS	Fractional Signed Multiply	(E) ∗ (D) ⇒ E : D[31:1] 0 ⇒ D[0]	INH	3727	—	8	—	—	—	—	Δ	Δ	Δ	Δ
IDIV	Integer Divide	(D) / (IX) ⇒ IX; Remainder ⇒ D	INH	372A	—	22	—	—	—	—	—	Δ	0	Δ
INC	Increment Memory	(M) + $01 ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	03 13 23 1703 1713 1723 1733	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	—
INCA	Increment A	(A) + $01 ⇒ A	INH	3703	—	2	—	—	—	—	Δ	Δ	Δ	—
INCB	Increment B	(B) + $01 ⇒ B	INH	3713	—	2	—	—	—	—	Δ	Δ	Δ	—
INCW	Increment Memory Word	(M : M + 1) + $0001 ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2703 2713 2723 2733	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	—
JMP	Jump	⟨ea⟩ ⇒ PK : PC	IND20, X IND20, Y IND20, Z EXT20	4B 5B 6B 7A	zg gggg zg gggg zg gggg zb hh ll	8 8 8 6	—	—	—	—	—	—	—	—
JSR	Jump to Subroutine	Push (PC) (SK : SP) – 2 ⇒ SK : SP Push (CCR) (SK : SP) – 2 ⇒ SK : SP ⟨ea⟩ ⇒ PK : PC	IND20, X IND20, Y IND20, Z EXT20	89 99 A9 FA	zg gggg zg gggg zg gggg zb hh ll	12 12 12 10	—	—	—	—	—	—	—	—
LBCC[4]	Long Branch if Carry Clear	If C = 0, branch	REL16	3784	rrrr	6, 4	—	—	—	—	—	—	—	—
LBCS[4]	Long Branch if Carry Set	If C = 1, branch	REL16	3785	rrrr	6, 4	—	—	—	—	—	—	—	—
LBEQ[4]	Long Branch if Equal	If Z = 1, branch	REL16	3787	rrrr	6, 4	—	—	—	—	—	—	—	—
LBEV[4]	Long Branch if EV Set	If EV = 1, branch	REL16	3791	rrrr	6, 4	—	—	—	—	—	—	—	—
LBGE[4]	Long Branch if Greater Than or Equal to Zero	If N ⊕ V = 0, branch	REL16	378C	rrrr	6, 4	—	—	—	—	—	—	—	—

Table 9 CPU16 Instruction Set Summary (Sheet 8 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
LBGT [4]	Long Branch if Greater Than Zero	If Z ✣ (N ⊕ V) = 0, branch	REL16	378E	rrrr	6, 4	—	—	—	—	—	—	—	—
LBHI [4]	Long Branch if Higher	If C ✣ Z = 0, branch	REL16	3782	rrrr	6, 4	—	—	—	—	—	—	—	—
LBLE [4]	Long Branch if Less Than or Equal to Zero	If Z ✣ (N ⊕ V) = 1, branch	REL16	378F	rrrr	6, 4	—	—	—	—	—	—	—	—
LBLS [4]	Long Branch if Lower or Same	If C ✣ Z = 1, branch	REL16	3783	rrrr	6, 4	—	—	—	—	—	—	—	—
LBLT [4]	Long Branch if Less Than Zero	If N ⊕ V = 1, branch	REL16	378D	rrrr	6, 4	—	—	—	—	—	—	—	—
LBMI [4]	Long Branch if Minus	If N = 1, branch	REL16	378B	rrrr	6, 4	—	—	—	—	—	—	—	—
LBMV [4]	Long Branch if MV Set	If MV = 1, branch	REL16	3790	rrrr	6, 4	—	—	—	—	—	—	—	—
LBNE [4]	Long Branch if Not Equal	If Z = 0, branch	REL16	3786	rrrr	6, 4	—	—	—	—	—	—	—	—
LBPL [4]	Long Branch if Plus	If N = 0, branch	REL16	378A	rrrr	6, 4	—	—	—	—	—	—	—	—
LBRA	Long Branch Always	If 1 = 1, branch	REL16	3780	rrrr	6	—	—	—	—	—	—	—	—
LBRN	Long Branch Never	If 1 = 0, branch	REL16	3781	rrrr	6	—	—	—	—	—	—	—	—
LBSR	Long Branch to Subroutine	Push (PC) (SK : SP) – 2 ⇒ SK : SP Push (CCR) (SK : SP) – 2 ⇒ SK : SP (PK : PC) + Offset ⇒ PK : PC	REL16	27F9	rrrr	10	—	—	—	—	—	—	—	—
LBVC [4]	Long Branch if Overflow Clear	If V = 0, branch	REL16	3788	rrrr	6, 4	—	—	—	—	—	—	—	—
LBVS [4]	Long Branch if Overflow Set	If V = 1, branch	REL16	3789	rrrr	6, 4	—	—	—	—	—	—	—	—
LDAA	Load A	(M) ⇒ A	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	45 55 65 75 1745 1755 1765 1775 2745 2755 2765	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
LDAB	Load B	(M) ⇒ B	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C5 D5 E5 F5 17C5 17D5 17E5 17F5 27C5 27D5 27E5	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
LDD	Load D	(M : M + 1) ⇒ D	IND8, X IND8, Y IND8, Z E, X E, Y E, Z IMM16 IND16, X IND16, Y IND16, Z EXT	85 95 A5 2785 2795 27A5 37B5 37C5 37D5 37E5 37F5	ff ff ff — — — jj kk gggg gggg gggg hh ll	6 6 6 6 6 6 4 6 6 6 6	—	—	—	—	Δ	Δ	0	—
LDE	Load E	(M : M + 1) ⇒ E	IMM16 IND16, X IND16, Y IND16, Z EXT	3735 3745 3755 3765 3775	jj kk gggg gggg gggg hh ll	4 6 6 6 6	—	—	—	—	Δ	Δ	0	—
LDED	Load Concatenated E and D	(M : M + 1) ⇒ E (M + 2 : M + 3) ⇒ D	EXT	2771	hh ll	8	—	—	—	—	—	—	—	—
LDHI	Initialize H and I	$(M : M + 1)_X$ ⇒ H R $(M : M + 1)_Y$ ⇒ I R	EXT	27B0	—	8	—	—	—	—	—	—	—	—

Table 9 CPU16 Instruction Set Summary (Sheet 9 of 15)

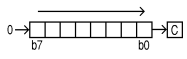
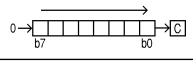
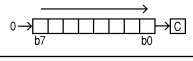
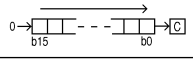
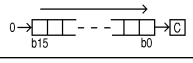
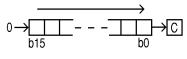
Mnemonic	Operation	Description	Address Mode	Instruction Opcode	Instruction Operand	Instruction Cycles	Condition Codes S	MV	H	EV	N	Z	V	C
LDS	Load SP	(M : M + 1) ⇒ SP	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT IMM16	CF DF EF 17CF 17DF 17EF 17FF 37BF	ff ff ff gggg gggg gggg hh ll jj kk	6 6 6 6 6 6 6 4	—	—	—	—	Δ	Δ	0	—
LDX	Load IX	(M : M + 1) ⇒ IX	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT IMM16	CC DC EC 17CC 17DC 17EC 17FC 37BC	ff ff ff gggg gggg gggg hh ll jj kk	6 6 6 6 6 6 6 4	—	—	—	—	Δ	Δ	0	—
LDY	Load IY	(M : M + 1) ⇒ IY	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT IMM16	CD DD ED 17CD 17DD 17ED 17FD 37BD	ff ff ff gggg gggg gggg hh ll jj kk	6 6 6 6 6 6 6 4	—	—	—	—	Δ	Δ	0	—
LDZ	Load IZ	(M : M + 1) ⇒ IZ	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT IMM16	CE DE EE 17CE 17DE 17EE 17FE 37BE	ff ff ff gggg gggg gggg hh ll jj kk	6 6 6 6 6 6 6 4	—	—	—	—	Δ	Δ	0	—
LPSTOP	Low Power Stop	If $\overline{S}$ then STOP else NOP	INH	27F1	—	4, 20	—	—	—	—	—	—	—	—
LSR	Logical Shift Right		IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	0F 1F 2F 170F 171F 172F 173F	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	0	Δ	Δ	Δ
LSRA	Logical Shift Right A		INH	370F	—	2	—	—	—	—	0	Δ	Δ	Δ
LSRB	Logical Shift Right B		INH	371F	—	2	—	—	—	—	0	Δ	Δ	Δ
LSRD	Logical Shift Right D		INH	27FF	—	2	—	—	—	—	0	Δ	Δ	Δ
LSRE	Logical Shift Right E		INH	277F	—	2	—	—	—	—	0	Δ	Δ	Δ
LSRW	Logical Shift Right Word		IND16, X IND16, Y IND16, Z EXT	270F 271F 272F 273F	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	0	Δ	Δ	Δ
MAC	Multiply and Accumulate Signed 16-Bit Fractions	(HR) * (IR) ⇒ E : D (AM) + (E : D) ⇒ AM Qualified (IX) ⇒ IX Qualified (IY) ⇒ IY (HR) ⇒ IZ $(M : M + 1)_X$ ⇒ HR $(M : M + 1)_Y$ ⇒ IR	IMM8	7B	xoyo	12	—	Δ	—	Δ	—	—	Δ	—
MOVB	Move Byte	$(M_1) \Rightarrow M_2$	IXP to EXT EXT to IXP EXT to EXT	30 32 37FE	ff hh ll ff hh ll hh ll hh ll	8 8 10	—	—	—	—	Δ	Δ	0	—
MOVW	Move Word	$(M : M + 1_1) \Rightarrow M : M + 1_2$	IXP to EXT EXT to IXP EXT to EXT	31 33 37FF	ff hh ll ff hh ll hh ll hh ll	8 8 10	—	—	—	—	Δ	Δ	0	—
MUL	Multiply	(A) * (B) ⇒ D	INH	3724	—	10	—	—	—	—	—	—	—	Δ

Table 9 CPU16 Instruction Set Summary (Sheet 10 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction Opcode	Instruction Operand	Instruction Cycles	Condition Codes S	MV	H	EV	N	Z	V	C
NEG	Negate Memory	$00 – (M) ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	02 12 22 1702 1712 1722 1732	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
NEGA	Negate A	$00 – (A) ⇒ A	INH	3702	—	2	—	—	—	—	Δ	Δ	Δ	Δ
NEGB	Negate B	$00 – (B) ⇒ B	INH	3712	—	2	—	—	—	—	Δ	Δ	Δ	Δ
NEGD	Negate D	$0000 – (D) ⇒ D	INH	27F2	—	2	—	—	—	—	Δ	Δ	Δ	Δ
NEGE	Negate E	$0000 – (E) ⇒ E	INH	2772	—	2	—	—	—	—	Δ	Δ	Δ	Δ
NEGW	Negate Memory Word	$0000 – (M : M + 1) ⇒ M : M + 1	IND16, X IND16, Y IND16, Z EXT	2702 2712 2722 2732	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
NOP	Null Operation	—	INH	274C	—	2	—	—	—	—	—	—	—	—
ORAA	OR A	(A) ✣ (M) ⇒ A	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	47 57 67 77 1747 1757 1767 1777 2747 2757 2767	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
ORAB	OR B	(B) ✣ (M) ⇒ B	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C7 D7 E7 F7 17C7 17D7 17E7 17F7 27C7 27D7 27E7	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	—
ORD	OR D	(D) ✣ (M : M + 1) ⇒ D	IND8, X IND8, Y IND8, Z E, X E, Y E, Z IMM16 IND16, X IND16, Y IND16, Z EXT	87 97 A7 2787 2797 27A7 37B7 37C7 37D7 37E7 37F7	ff ff ff — — — jj kk gggg gggg gggg hh ll	6 6 6 6 6 6 4 6 6 6 6	—	—	—	—	Δ	Δ	0	—
ORE	OR E	(E) ✣ (M : M + 1) ⇒ E	IMM16 IND16, X IND16, Y IND16, Z EXT	3737 3747 3757 3767 3777	jj kk gggg gggg gggg hh ll	4 6 6 6 6	—	—	—	—	Δ	Δ	0	—
ORP [1]	OR Condition Code Register	(CCR) ✣ IMM16 ⇒ CCR	IMM16	373B	jj kk	4	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
PSHA	Push A	(SK : SP) + 1 ⇒ SK : SP Push (A) (SK : SP) – 2 ⇒ SK : SP	INH	3708	—	4	—	—	—	—	—	—	—	—
PSHB	Push B	(SK : SP) + 1 ⇒ SK : SP Push (B) (SK : SP) – 2 ⇒ SK : SP	INH	3718	—	4	—	—	—	—	—	—	—	—
PSHM	Push Multiple Registers Mask bits: 0 = D 1 = E 2 = IX 3 = IY 4 = IZ 5 = K 6 = CCR 7 = (reserved)	For mask bits 0 to 7: If mask bit set Push register (SK : SP) – 2 ⇒ SK : SP	IMM8	34	ii	4 + 2N N = number of iterations	—	—	—	—	—	—	—	—
PSHMAC	Push MAC State	MAC Registers ⇒ Stack	INH	27B8	—	14	—	—	—	—	—	—	—	—

Table 9 CPU16 Instruction Set Summary (Sheet 11 of 15)

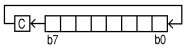
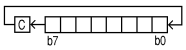
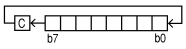
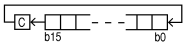
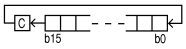
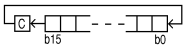
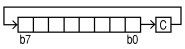
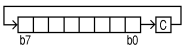
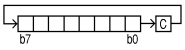
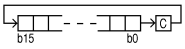
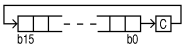
Mnemonic	Operation	Description	Address Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
PULA	Pull A	(SK : SP) + 2 ⇒ SK : SP Pull (A) (SK : SP) – 1 ⇒ SK : SP	INH	3709	—	6	—	—	—	—	—	—	—	—
PULB	Pull B	(SK : SP) + 2 ⇒ SK : SP Pull (B) (SK : SP) – 1 ⇒ SK : SP	INH	3719	—	6	—	—	—	—	—	—	—	—
PULM [1]	Pull Multiple Registers Mask bits: 0 = CCR[15:4] 1 = K 2 = IZ 3 = IY 4 = IX 5 = E 6 = D 7 = (reserved)	For mask bits 0 to 7: If mask bit set (SK : SP) + 2 ⇒ SK : SP Pull register	IMM8	35	ii	4+2(N+1) N = number of iterations	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
PULMAC	Pull MAC State	Stack ⇒ MAC Registers	INH	27B9	—	16	—	—	—	—	—	—	—	—
RMAC	Repeating Multiply and Accumulate Signed 16-Bit Fractions	Repeat until (E) < 0 (AM) + (H) * (I) ⇒ AM Qualified (IX) ⇒ IX; Qualified (IY) ⇒ IY; $(M : M + 1)_X \Rightarrow H$; $(M : M + 1)_Y \Rightarrow I$ (E) – 1 ⇒ E	IMM8	FB	xoyo	6 + 12 per iteration	—	Δ	—	Δ	—	—	—	—
ROL	Rotate Left		IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	0C 1C 2C 170C 171C 172C 173C	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
ROLA	Rotate Left A		INH	370C	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ROLB	Rotate Left B		INH	371C	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ROLD	Rotate Left D		INH	27FC	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ROLE	Rotate Left E		INH	277C	—	2	—	—	—	—	Δ	Δ	Δ	Δ
ROLW	Rotate Left Word		IND16, X IND16, Y IND16, Z EXT	270C 271C 272C 273C	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
ROR	Rotate Right		IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	0E 1E 2E 170E 171E 172E 173E	ff ff ff gggg gggg gggg hh ll	8 8 8 8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
RORA	Rotate Right A		INH	370E	—	2	—	—	—	—	Δ	Δ	Δ	Δ
RORB	Rotate Right B		INH	371E	—	2	—	—	—	—	Δ	Δ	Δ	Δ
RORD	Rotate Right D		INH	27FE	—	2	—	—	—	—	Δ	Δ	Δ	Δ
RORE	Rotate Right E		INH	277E	—	2	—	—	—	—	Δ	Δ	Δ	Δ

Table 9 CPU16 Instruction Set Summary (Sheet 12 of 15)

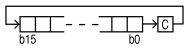
Mnemonic	Operation	Description	Address Mode	Instruction Opcode	Instruction Operand	Instruction Cycles	Condition Codes S	MV	H	EV	N	Z	V	C
RORW	Rotate Right Word		IND16, X IND16, Y IND16, Z EXT	270E 271E 272E 273E	gggg gggg gggg hh ll	8 8 8 8	—	—	—	—	Δ	Δ	Δ	Δ
RTI[2]	Return from Interrupt	(SK : SP) + 2 ⇒ SK : SP Pull CCR (SK : SP) + 2 ⇒ SK : SP Pull PC (PK : PC) – 6 ⇒ PK : PC	INH	2777	—	12	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
RTS[3]	Return from Subroutine	(SK : SP) + 2 ⇒ SK : SP Pull PK (SK : SP) + 2 ⇒ SK : SP Pull PC (PK : PC) – 2 ⇒ PK : PC	INH	27F7	—	12	—	—	—	—	—	—	—	—
SBA	Subtract B from A	(A) – (B) ⇒ A	INH	370A	—	2	—	—	—	—	Δ	Δ	Δ	Δ
SBCA	Subtract with Carry from A	(A) – (M) – C ⇒ A	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	42 52 62 72 1742 1752 1762 1772 2742 2752 2762	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
SBCB	Subtract with Carry from B	(B) – (M) – C ⇒ B	IND8, X IND8, Y IND8, Z IMM8 IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	C2 D2 E2 F2 17C2 17D2 17E2 17F2 27C2 27D2 27E2	ff ff ff ii gggg gggg gggg hh ll — — —	6 6 6 2 6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
SBCD	Subtract with Carry from D	(D) – (M : M + 1) – C ⇒ D	IND8, X IND8, Y IND8, Z E, X E, Y E, Z IMM16 IND16, X IND16, Y IND16, Z EXT	82 92 A2 2782 2792 27A2 37B2 37C2 37D2 37E2 37F2	ff ff ff — — — jj kk gggg gggg gggg hh ll	6 6 6 6 6 6 4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
SBCE	Subtract with Carry from E	(E) – (M : M + 1) – C ⇒ E	IMM16 IND16, X IND16, Y IND16, Z EXT	3732 3742 3752 3762 3772	jj kk gggg gggg gggg hh ll	4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
SDE	Subtract D from E	(E) – (D)⇒ E	INH	2779	—	2	—	—	—	—	Δ	Δ	Δ	Δ
STAA	Store A	(A) ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	4A 5A 6A 174A 175A 176A 177A 274A 275A 276A	ff ff ff gggg gggg gggg hh ll — — —	4 4 4 6 6 6 6 4 4 4	—	—	—	—	Δ	Δ	0	—
STAB	Store B	(B) ⇒ M	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT E, X E, Y E, Z	CA DA EA 17CA 17DA 17EA 17FA 27CA 27DA 27EA	ff ff ff gggg gggg gggg hh ll — — —	4 4 4 6 6 6 6 4 4 4	—	—	—	—	Δ	Δ	0	—

Table 9 CPU16 Instruction Set Summary (Sheet 13 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction Opcode	Instruction Operand	Instruction Cycles	Condition Codes S	MV	H	EV	N	Z	V	C
STD	Store D	(D) ⇒ M : M + 1	IND8, X	8A	ff	6	—	—	—	—	Δ	Δ	0	—
			IND8, Y	9A	ff	6								
			IND8, Z	AA	ff	6								
			E, X	278A	—	6								
			E, Y	279A	—	6								
			E, Z	27AA	—	6								
			IND16, X	37CA	gggg	4								
			IND16, Y	37DA	gggg	4								
			IND16, Z	37EA	gggg	4								
			EXT	37FA	hh ll	6								
STE	Store E	(E) ⇒ M : M + 1	IND16, X	374A	gggg	6	—	—	—	—	Δ	Δ	0	—
			IND16, Y	375A	gggg	6								
			IND16, Z	376A	gggg	6								
			EXT	377A	hh ll	6								
STED	Store Concatenated D and E	(E) ⇒ M : M + 1 (D) ⇒ M + 2 : M + 3	EXT	2773	hh ll	8	—	—	—	—	—	—	—	—
STS	Store SP	(SP) ⇒ M : M + 1	IND8, X	8F	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	9F	ff	4								
			IND8, Z	AF	ff	4								
			IND16, X	178F	gggg	6								
			IND16, Y	179F	gggg	6								
			IND16, Z	17AF	gggg	6								
			EXT	17BF	hh ll	6								
STX	Store IX	(IX) ⇒ M : M + 1	IND8, X	8C	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	9C	ff	4								
			IND8, Z	AC	ff	4								
			IND16, X	178C	gggg	6								
			IND16, Y	179C	gggg	6								
			IND16, Z	17AC	gggg	6								
			EXT	17BC	hh ll	6								
STY	Store IY	(IY) ⇒ M : M + 1	IND8, X	8D	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	9D	ff	4								
			IND8, Z	AD	ff	4								
			IND16, X	178D	gggg	6								
			IND16, Y	179D	gggg	6								
			IND16, Z	17AD	gggg	6								
			EXT	17BD	hh ll	6								
STZ	Store Z	(IZ) ⇒ M : M + 1	IND8, X	8E	ff	4	—	—	—	—	Δ	Δ	0	—
			IND8, Y	9E	ff	4								
			IND8, Z	AE	ff	4								
			IND16, X	178E	gggg	6								
			IND16, Y	179E	gggg	6								
			IND16, Z	17AE	gggg	6								
			EXT	17BE	hh ll	6								
SUBA	Subtract from A	(A) – (M) ⇒ A	IND8, X	40	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	50	ff	6								
			IND8, Z	60	ff	6								
			IMM8	70	ii	2								
			IND16, X	1740	gggg	6								
			IND16, Y	1750	gggg	6								
			IND16, Z	1760	gggg	6								
			EXT	1770	hh ll	6								
			E, X	2740	—	6								
			E, Y	2750	—	6								
			E, Z	2760	—	6								
SUBB	Subtract from B	(B) – (M) ⇒ B	IND8, X	C0	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	D0	ff	6								
			IND8, Z	E0	ff	6								
			IMM8	F0	ii	2								
			IND16, X	17C0	gggg	6								
			IND16, Y	17D0	gggg	6								
			IND16, Z	17E0	gggg	6								
			EXT	17F0	hh ll	6								
			E, X	27C0	—	6								
			E, Y	27D0	—	6								
			E, Z	27E0	—	6								
SUBD	Subtract from D	(D) – (M : M + 1) ⇒ D	IND8, X	80	ff	6	—	—	—	—	Δ	Δ	Δ	Δ
			IND8, Y	90	ff	6								
			IND8, Z	A0	ff	6								
			E, X	2780	—	6								
			E, Y	2790	—	6								
			E, Z	27A0	—	6								
			IMM16	37B0	jj kk	4								
			IND16, X	37C0	gggg	6								
			IND16, Y	37D0	gggg	6								
			IND16, Z	37E0	gggg	6								
			EXT	37F0	hh ll	6								

Table 9 CPU16 Instruction Set Summary (Sheet 14 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction Opcode	Instruction Operand	Instruction Cycles	Condition Codes S	MV	H	EV	N	Z	V	C
SUBE	Subtract from E	(E) – (M : M + 1) ⇒ E	IMM16 IND16, X IND16, Y IND16, Z EXT	3730 3740 3750 3760 3770	jj kk gggg gggg gggg hh ll	4 6 6 6 6	—	—	—	—	Δ	Δ	Δ	Δ
SWI	Software Interrupt	(PK : PC) + 2 ⇒ PK : PC Push (PC) (SK : SP) – 2 ⇒ SK : SP Push (CCR) (SK : SP) – 2 ⇒ SK : SP $0 ⇒ PK SWI Vector ⇒ PC	INH	3720	—	16	—	—	—	—	—	—	—	—
SXT	Sign Extend B into A	If B7 = 1 then A = $FF else A = $00	INH	27F8	—	2	—	—	—	—	Δ	Δ	—	—
TAB	Transfer A to B	(A) ⇒ B	INH	3717	—	2	—	—	—	—	Δ	Δ	0	—
TAP	Transfer A to CCR	(A[7:0]) ⇒ CCR[15:8]	INH	37FD	—	4	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
TBA	Transfer B to A	(B) ⇒ A	INH	3707	—	2	—	—	—	—	Δ	Δ	0	—
TBEK	Transfer B to EK	(B) ⇒ EK	INH	27FA	—	2	—	—	—	—	—	—	—	—
TBSK	Transfer B to SK	(B) ⇒ SK	INH	379F	—	2	—	—	—	—	—	—	—	—
TBXK	Transfer B to XK	(B) ⇒ XK	INH	379C	—	2	—	—	—	—	—	—	—	—
TBYK	Transfer B to YK	(B) ⇒ YK	INH	379D	—	2	—	—	—	—	—	—	—	—
TBZK	Transfer B to ZK	(B) ⇒ ZK	INH	379E	—	2	—	—	—	—	—	—	—	—
TDE	Transfer D to E	(D) ⇒ E	INH	277B	—	2	—	—	—	—	Δ	Δ	0	—
TDMSK	Transfer D to XMSK : YMSK	(D[15:8]) ⇒ X MASK (D[7:0]) ⇒ Y MASK	INH	372F	—	2	—	—	—	—	—	—	—	—
TDP[1]	Transfer D to CCR	(D) ⇒ CCR[15:4]	INH	372D	—	4	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
TED	Transfer E to D	(E) ⇒ D	INH	27FB	—	2	—	—	—	—	Δ	Δ	0	—
TEDM	Transfer E and D to AM[31:0] Sign Extend AM	(D) ⇒ AM[15:0] (E) ⇒ AM[31:16] AM[35:32] = AM31	INH	27B1	—	4	—	0	—	0	—	—	—	—
TEKB	Transfer EK to B	$0 ⇒ B[7:4] (EK) ⇒ B[3:0]	INH	27BB	—	2	—	—	—	—	—	—	—	—
TEM	Transfer E to AM[31:16] Sign Extend AM Clear AM LSB	(E) ⇒ AM[31:16] $00 ⇒ AM[15:0] AM[35:32] = AM31	INH	27B2	—	4	—	0	—	0	—	—	—	—
TMER	Transfer AM to E Rounded	Rounded (AM) ⇒ Temp If (SM • (EV ✣ MV)) then Saturation ⇒ E else Temp[31:16] ⇒ E	INH	27B4	—	6	—	Δ	—	Δ	Δ	Δ	—	—
TMET	Transfer AM to E Truncated	If (SM • (EV ✣ MV)) then Saturation ⇒ E else AM[31:16] ⇒ E	INH	27B5	—	2	—	—	—	—	Δ	Δ	—	—
TMXED	Transfer AM to IX : E : D	AM[35:32] ⇒ IX[3:0] AM35 ⇒ IX[15:4] AM[31:16] ⇒ E AM[15:0] ⇒ D	INH	27B3	—	6	—	—	—	—	—	—	—	—
TPA	Transfer CCR MSB to A	(CCR[15:8]) ⇒ A	INH	37FC	—	2	—	—	—	—	—	—	—	—
TPD	Transfer CCR to D	(CCR) ⇒ D	INH	372C	—	2	—	—	—	—	—	—	—	—
TSKB	Transfer SK to B	(SK) ⇒ B[3:0] $0 ⇒ B[7:4]	INH	37AF	—	2	—	—	—	—	—	—	—	—
TST	Test for Zero or Minus	(M) – $00	IND8, X IND8, Y IND8, Z IND16, X IND16, Y IND16, Z EXT	06 16 26 1706 1716 1726 1736	ff ff ff gggg gggg gggg hh ll	6 6 6 6 6 6 6	—	—	—	—	Δ	Δ	0	0
TSTA	Test A for Zero or Minus	(A) – $00	INH	3706	—	2	—	—	—	—	Δ	Δ	0	0
TSTB	Test B for Zero or Minus	(B) – $00	INH	3716	—	2	—	—	—	—	Δ	Δ	0	0
TSTD	Test D for Zero or Minus	(D) – $0000	INH	27F6	—	2	—	—	—	—	Δ	Δ	0	0
TSTE	Test E for Zero or Minus	(E) – $0000	INH	2776	—	2	—	—	—	—	Δ	Δ	0	0

Table 9 CPU16 Instruction Set Summary (Sheet 15 of 15)

Mnemonic	Operation	Description	Address Mode	Instruction			Condition Codes							
				Opcode	Operand	Cycles	S	MV	H	EV	N	Z	V	C
TSTW	Test for Zero or Minus Word	(M : M + 1) – $0000	IND16, X IND16, Y IND16, Z EXT	2706 2716 2726 2736	gggg gggg gggg hh ll	6 6 6 6	—	—	—	—	Δ	Δ	0	0
TSX	Transfer SP to X	(SK : SP) + 2 ⇒ XK : IX	INH	274F	—	2	—	—	—	—	—	—	—	—
TSY	Transfer SP to Y	(SK : SP) + 2 ⇒ YK : IY	INH	275F	—	2	—	—	—	—	—	—	—	—
TSZ	Transfer SP to Z	(SK : SP) + 2 ⇒ ZK : IZ	INH	276F	—	2	—	—	—	—	—	—	—	—
TXKB	Transfer XK to B	$0 ⇒ B[7:4] (XK) ⇒ B[3:0]	INH	37AC	—	2	—	—	—	—	—	—	—	—
TXS	Transfer X to SP	(XK : IX) – 2 ⇒ SK : SP	INH	374E	—	2	—	—	—	—	—	—	—	—
TXY	Transfer X to Y	(XK : IX) ⇒ YK : IY	INH	275C	—	2	—	—	—	—	—	—	—	—
TXZ	Transfer X to Z	(XK : IX) ⇒ ZK : IZ	INH	276C	—	2	—	—	—	—	—	—	—	—
TYKB	Transfer YK to B	$0 ⇒ B[7:4] (YK) ⇒ B[3:0]	INH	37AD	—	2	—	—	—	—	—	—	—	—
TYS	Transfer Y to SP	(YK : IY) – 2 ⇒ SK : SP	INH	375E	—	2	—	—	—	—	—	—	—	—
TYX	Transfer Y to X	(YK : IY) ⇒ XK : IX	INH	274D	—	2	—	—	—	—	—	—	—	—
TYZ	Transfer Y to Z	(YK : IY) ⇒ ZK : IZ	INH	276D	—	2	—	—	—	—	—	—	—	—
TZKB	Transfer ZK to B	$0 ⇒ B[7:4] (ZK) ⇒ B[3:0]	INH	37AE	—	2	—	—	—	—	—	—	—	—
TZS	Transfer Z to SP	(ZK : IZ) – 2 ⇒ SK : SP	INH	376E	—	2	—	—	—	—	—	—	—	—
TZX	Transfer Z to X	(ZK : IZ) ⇒ XK : IX	INH	274E	—	2	—	—	—	—	—	—	—	—
TZY	Transfer Z to Y	(ZK : IZ) ⇒ ZK : IY	INH	275E	—	2	—	—	—	—	—	—	—	—
WAI	Wait for Interrupt	WAIT	INH	27F3	—	8	—	—	—	—	—	—	—	—
XGAB	Exchange A with B	(A) ⇔ (B)	INH	371A	—	2	—	—	—	—	—	—	—	—
XGDE	Exchange D with E	(D) ⇔ (E)	INH	277A	—	2	—	—	—	—	—	—	—	—
XGDX	Exchange D with X	(D) ⇔ (IX)	INH	37CC	—	2	—	—	—	—	—	—	—	—
XGDY	Exchange D with Y	(D) ⇔ (IY)	INH	37DC	—	2	—	—	—	—	—	—	—	—
XGDZ	Exchange D with Z	(D) ⇔ (IZ)	INH	37EC	—	2	—	—	—	—	—	—	—	—
XGEX	Exchange E with X	(E) ⇔ (IX)	INH	374C	—	2	—	—	—	—	—	—	—	—
XGEY	Exchange E with Y	(E) ⇔ (IY)	INH	375C	—	2	—	—	—	—	—	—	—	—
XGEZ	Exchange E with Z	(E) ⇔ (IZ)	INH	376C	—	2	—	—	—	—	—	—	—	—

NOTES:

1. CCR[15:4] change according to results of operation. The PK field is not affected.
2. CCR[15:0] change according to copy of CCR pulled from stack.
3. PK field changes according to state pulled from stack. The rest of the CCR is not affected.
4. Cycle times for conditional branches are shown in "taken, not taken" order.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor