

Chapter 7

Central Processor Unit (S08CPUV2)

7.1 Introduction

This section provides summary information about the registers, addressing modes, and instruction set of the CPU of the HCS08 Family. For a more detailed discussion, refer to the *HCS08 Family Reference Manual, volume 1*, Freescale Semiconductor document order number HCS08RMV1/D.

The HCS08 CPU is fully source- and object-code-compatible with the M68HC08 CPU. Several instructions and enhanced addressing modes were added to improve C compiler efficiency and to support a new background debug system which replaces the monitor mode of earlier M68HC08 microcontrollers (MCU).

7.1.1 Features

Features of the HCS08 CPU include:

- Object code fully upward-compatible with M68HC05 and M68HC08 Families
- All registers and memory are mapped to a single 64-Kbyte address space
- 16-bit stack pointer (any size stack anywhere in 64-Kbyte address space)
- 16-bit index register (H:X) with powerful indexed addressing modes
- 8-bit accumulator (A)
- Many instructions treat X as a second general-purpose 8-bit register
- Seven addressing modes:
 - Inherent — Operands in internal registers
 - Relative — 8-bit signed offset to branch destination
 - Immediate — Operand in next object code byte(s)
 - Direct — Operand in memory at 0x0000–0x00FF
 - Extended — Operand anywhere in 64-Kbyte address space
 - Indexed relative to H:X — Five submodes including auto increment
 - Indexed relative to SP — Improves C efficiency dramatically
- Memory-to-memory data move instructions with four address mode combinations
- Overflow, half-carry, negative, zero, and carry condition codes support conditional branching on the results of signed, unsigned, and binary-coded decimal (BCD) operations
- Efficient bit manipulation instructions
- Fast 8-bit by 8-bit multiply and 16-bit by 8-bit divide instructions
- STOP and WAIT instructions to invoke low-power operating modes

7.2 Programmer's Model and CPU Registers

Figure 7-1 shows the five CPU registers. CPU registers are not part of the memory map.

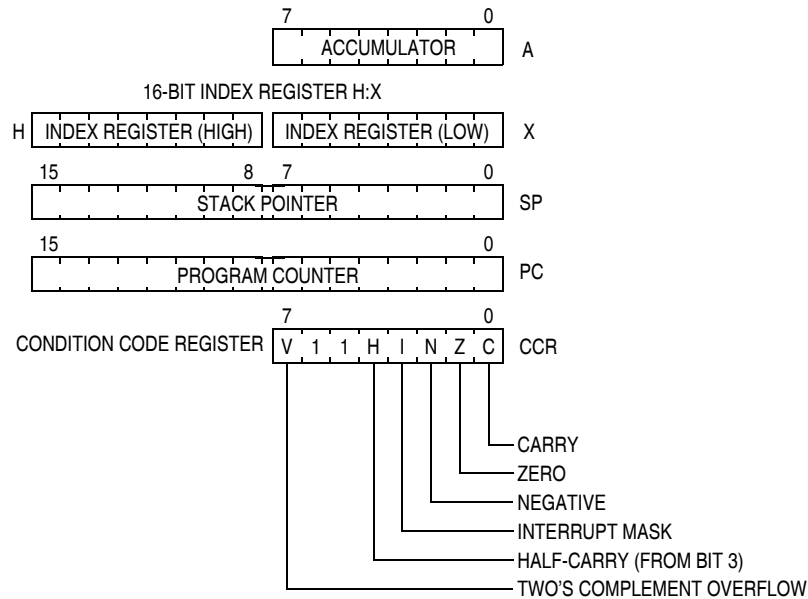


Figure 7-1. CPU Registers

7.2.1 Accumulator (A)

The A accumulator is a general-purpose 8-bit register. One operand input to the arithmetic logic unit (ALU) is connected to the accumulator and the ALU results are often stored into the A accumulator after arithmetic and logical operations. The accumulator can be loaded from memory using various addressing modes to specify the address where the loaded data comes from, or the contents of A can be stored to memory using various addressing modes to specify the address where data from A will be stored.

Reset has no effect on the contents of the A accumulator.

7.2.2 Index Register (H:X)

This 16-bit register is actually two separate 8-bit registers (H and X), which often work together as a 16-bit address pointer where H holds the upper byte of an address and X holds the lower byte of the address. All indexed addressing mode instructions use the full 16-bit value in H:X as an index reference pointer; however, for compatibility with the earlier M68HC05 Family, some instructions operate only on the low-order 8-bit half (X).

Many instructions treat X as a second general-purpose 8-bit register that can be used to hold 8-bit data values. X can be cleared, incremented, decremented, complemented, negated, shifted, or rotated. Transfer instructions allow data to be transferred from A or transferred to A where arithmetic and logical operations can then be performed.

For compatibility with the earlier M68HC05 Family, H is forced to 0x00 during reset. Reset has no effect on the contents of X.

7.2.3 Stack Pointer (SP)

This 16-bit address pointer register points at the next available location on the automatic last-in-first-out (LIFO) stack. The stack may be located anywhere in the 64-Kbyte address space that has RAM and can be any size up to the amount of available RAM. The stack is used to automatically save the return address for subroutine calls, the return address and CPU registers during interrupts, and for local variables. The AIS (add immediate to stack pointer) instruction adds an 8-bit signed immediate value to SP. This is most often used to allocate or deallocate space for local variables on the stack.

SP is forced to 0x00FF at reset for compatibility with the earlier M68HC05 Family. HCS08 programs normally change the value in SP to the address of the last location (highest address) in on-chip RAM during reset initialization to free up direct-page RAM (from the end of the on-chip registers to 0x00FF).

The RSP (reset stack pointer) instruction was included for compatibility with the M68HC05 Family and is seldom used in new HCS08 programs because it only affects the low-order half of the stack pointer.

7.2.4 Program Counter (PC)

The program counter is a 16-bit register that contains the address of the next instruction or operand to be fetched.

During normal program execution, the program counter automatically increments to the next sequential memory location every time an instruction or operand is fetched. Jump, branch, interrupt, and return operations load the program counter with an address other than that of the next sequential location. This is called a change-of-flow.

During reset, the program counter is loaded with the reset vector that is located at 0xFFFFE and 0xFFFF. The vector stored there is the address of the first instruction that will be executed after exiting the reset state.

7.2.5 Condition Code Register (CCR)

The 8-bit condition code register contains the interrupt mask (I) and five flags that indicate the results of the instruction just executed. Bits 6 and 5 are set permanently to 1. The following paragraphs describe the functions of the condition code bits in general terms. For a more detailed explanation of how each instruction sets the CCR bits, refer to the *HCS08 Family Reference Manual, volume 1*, Freescale Semiconductor document order number HCS08RMv1.

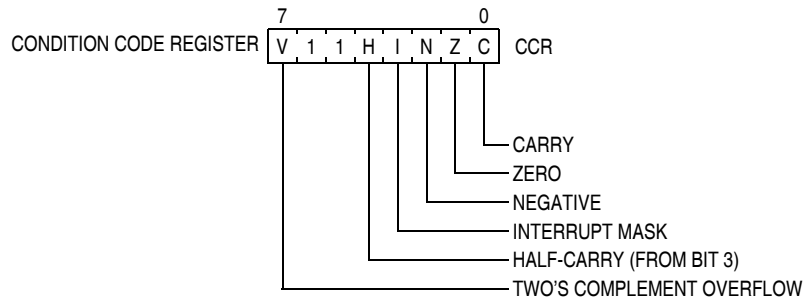


Figure 7-2. Condition Code Register

Table 7-1. CCR Register Field Descriptions

Field	Description
7 V	Two's Complement Overflow Flag — The CPU sets the overflow flag when a two's complement overflow occurs. The signed branch instructions BGT, BGE, BLE, and BLT use the overflow flag. 0 No overflow 1 Overflow
4 H	Half-Carry Flag — The CPU sets the half-carry flag when a carry occurs between accumulator bits 3 and 4 during an add-without-carry (ADD) or add-with-carry (ADC) operation. The half-carry flag is required for binary-coded decimal (BCD) arithmetic operations. The DAA instruction uses the states of the H and C condition code bits to automatically add a correction value to the result from a previous ADD or ADC on BCD operands to correct the result to a valid BCD value. 0 No carry between bits 3 and 4 1 Carry between bits 3 and 4
3 I	Interrupt Mask Bit — When the interrupt mask is set, all maskable CPU interrupts are disabled. CPU interrupts are enabled when the interrupt mask is cleared. When a CPU interrupt occurs, the interrupt mask is set automatically after the CPU registers are saved on the stack, but before the first instruction of the interrupt service routine is executed. Interrupts are not recognized at the instruction boundary after any instruction that clears I (CLI or TAP). This ensures that the next instruction after a CLI or TAP will always be executed without the possibility of an intervening interrupt, provided I was set. 0 Interrupts enabled 1 Interrupts disabled
2 N	Negative Flag — The CPU sets the negative flag when an arithmetic operation, logic operation, or data manipulation produces a negative result, setting bit 7 of the result. Simply loading or storing an 8-bit or 16-bit value causes N to be set if the most significant bit of the loaded or stored value was 1. 0 Non-negative result 1 Negative result
1 Z	Zero Flag — The CPU sets the zero flag when an arithmetic operation, logic operation, or data manipulation produces a result of 0x00 or 0x0000. Simply loading or storing an 8-bit or 16-bit value causes Z to be set if the loaded or stored value was all 0s. 0 Non-zero result 1 Zero result
0 C	Carry/Borrow Flag — The CPU sets the carry/borrow flag when an addition operation produces a carry out of bit 7 of the accumulator or when a subtraction operation requires a borrow. Some instructions — such as bit test and branch, shift, and rotate — also clear or set the carry/borrow flag. 0 No carry out of bit 7 1 Carry out of bit 7

7.3 Addressing Modes

Addressing modes define the way the CPU accesses operands and data. In the HCS08, all memory, status and control registers, and input/output (I/O) ports share a single 64-Kbyte linear address space so a 16-bit binary address can uniquely identify any memory location. This arrangement means that the same instructions that access variables in RAM can also be used to access I/O and control registers or nonvolatile program space.

Some instructions use more than one addressing mode. For instance, move instructions use one addressing mode to specify the source operand and a second addressing mode to specify the destination address. Instructions such as BRCLR, BRSET, CBEQ, and DBNZ use one addressing mode to specify the location of an operand for a test and then use relative addressing mode to specify the branch destination address when the tested condition is true. For BRCLR, BRSET, CBEQ, and DBNZ, the addressing mode listed in the instruction set tables is the addressing mode needed to access the operand to be tested, and relative addressing mode is implied for the branch destination.

7.3.1 Inherent Addressing Mode (INH)

In this addressing mode, operands needed to complete the instruction (if any) are located within CPU registers so the CPU does not need to access memory to get any operands.

7.3.2 Relative Addressing Mode (REL)

Relative addressing mode is used to specify the destination location for branch instructions. A signed 8-bit offset value is located in the memory location immediately following the opcode. During execution, if the branch condition is true, the signed offset is sign-extended to a 16-bit value and is added to the current contents of the program counter, which causes program execution to continue at the branch destination address.

7.3.3 Immediate Addressing Mode (IMM)

In immediate addressing mode, the operand needed to complete the instruction is included in the object code immediately following the instruction opcode in memory. In the case of a 16-bit immediate operand, the high-order byte is located in the next memory location after the opcode, and the low-order byte is located in the next memory location after that.

7.3.4 Direct Addressing Mode (DIR)

In direct addressing mode, the instruction includes the low-order eight bits of an address in the direct page (0x0000–0x00FF). During execution a 16-bit address is formed by concatenating an implied 0x00 for the high-order half of the address and the direct address from the instruction to get the 16-bit address where the desired operand is located. This is faster and more memory efficient than specifying a complete 16-bit address for the operand.

7.3.5 Extended Addressing Mode (EXT)

In extended addressing mode, the full 16-bit address of the operand is located in the next two bytes of program memory after the opcode (high byte first).

7.3.6 Indexed Addressing Mode

Indexed addressing mode has seven variations including five that use the 16-bit H:X index register pair and two that use the stack pointer as the base reference.

7.3.6.1 Indexed, No Offset (IX)

This variation of indexed addressing uses the 16-bit value in the H:X index register pair as the address of the operand needed to complete the instruction.

7.3.6.2 Indexed, No Offset with Post Increment (IX+)

This variation of indexed addressing uses the 16-bit value in the H:X index register pair as the address of the operand needed to complete the instruction. The index register pair is then incremented ($H:X = H:X + 0x0001$) after the operand has been fetched. This addressing mode is only used for MOV and CBEQ instructions.

7.3.6.3 Indexed, 8-Bit Offset (IX1)

This variation of indexed addressing uses the 16-bit value in the H:X index register pair plus an unsigned 8-bit offset included in the instruction as the address of the operand needed to complete the instruction.

7.3.6.4 Indexed, 8-Bit Offset with Post Increment (IX1+)

This variation of indexed addressing uses the 16-bit value in the H:X index register pair plus an unsigned 8-bit offset included in the instruction as the address of the operand needed to complete the instruction. The index register pair is then incremented ($H:X = H:X + 0x0001$) after the operand has been fetched. This addressing mode is used only for the CBEQ instruction.

7.3.6.5 Indexed, 16-Bit Offset (IX2)

This variation of indexed addressing uses the 16-bit value in the H:X index register pair plus a 16-bit offset included in the instruction as the address of the operand needed to complete the instruction.

7.3.6.6 SP-Relative, 8-Bit Offset (SP1)

This variation of indexed addressing uses the 16-bit value in the stack pointer (SP) plus an unsigned 8-bit offset included in the instruction as the address of the operand needed to complete the instruction.

7.3.6.7 SP-Relative, 16-Bit Offset (SP2)

This variation of indexed addressing uses the 16-bit value in the stack pointer (SP) plus a 16-bit offset included in the instruction as the address of the operand needed to complete the instruction.

7.4 Special Operations

The CPU performs a few special operations that are similar to instructions but do not have opcodes like other CPU instructions. In addition, a few instructions such as STOP and WAIT directly affect other MCU circuitry. This section provides additional information about these operations.

7.4.1 Reset Sequence

Reset can be caused by a power-on-reset (POR) event, internal conditions such as the COP (computer operating properly) watchdog, or by assertion of an external active-low reset pin. When a reset event occurs, the CPU immediately stops whatever it is doing (the MCU does not wait for an instruction boundary before responding to a reset event). For a more detailed discussion about how the MCU recognizes resets and determines the source, refer to the [Resets, Interrupts, and System Configuration](#) chapter.

The reset event is considered concluded when the sequence to determine whether the reset came from an internal source is done and when the reset pin is no longer asserted. At the conclusion of a reset event, the CPU performs a 6-cycle sequence to fetch the reset vector from 0xFFFFE and 0xFFFF and to fill the instruction queue in preparation for execution of the first program instruction.

7.4.2 Interrupt Sequence

When an interrupt is requested, the CPU completes the current instruction before responding to the interrupt. At this point, the program counter is pointing at the start of the next instruction, which is where the CPU should return after servicing the interrupt. The CPU responds to an interrupt by performing the same sequence of operations as for a software interrupt (SWI) instruction, except the address used for the vector fetch is determined by the highest priority interrupt that is pending when the interrupt sequence started.

The CPU sequence for an interrupt is:

1. Store the contents of PCL, PCH, X, A, and CCR on the stack, in that order.
2. Set the I bit in the CCR.
3. Fetch the high-order half of the interrupt vector.
4. Fetch the low-order half of the interrupt vector.
5. Delay for one free bus cycle.
6. Fetch three bytes of program information starting at the address indicated by the interrupt vector to fill the instruction queue in preparation for execution of the first instruction in the interrupt service routine.

After the CCR contents are pushed onto the stack, the I bit in the CCR is set to prevent other interrupts while in the interrupt service routine. Although it is possible to clear the I bit with an instruction in the

interrupt service routine, this would allow nesting of interrupts (which is not recommended because it leads to programs that are difficult to debug and maintain).

For compatibility with the earlier M68HC05 MCUs, the high-order half of the H:X index register pair (H) is not saved on the stack as part of the interrupt sequence. The user must use a PSHH instruction at the beginning of the service routine to save H and then use a PULH instruction just before the RTI that ends the interrupt service routine. It is not necessary to save H if you are certain that the interrupt service routine does not use any instructions or auto-increment addressing modes that might change the value of H.

The software interrupt (SWI) instruction is like a hardware interrupt except that it is not masked by the global I bit in the CCR and it is associated with an instruction opcode within the program so it is not asynchronous to program execution.

7.4.3 Wait Mode Operation

The WAIT instruction enables interrupts by clearing the I bit in the CCR. It then halts the clocks to the CPU to reduce overall power consumption while the CPU is waiting for the interrupt or reset event that will wake the CPU from wait mode. When an interrupt or reset event occurs, the CPU clocks will resume and the interrupt or reset event will be processed normally.

If a serial BACKGROUND command is issued to the MCU through the background debug interface while the CPU is in wait mode, CPU clocks will resume and the CPU will enter active background mode where other serial background commands can be processed. This ensures that a host development system can still gain access to a target MCU even if it is in wait mode.

7.4.4 Stop Mode Operation

Usually, all system clocks, including the crystal oscillator (when used), are halted during stop mode to minimize power consumption. In such systems, external circuitry is needed to control the time spent in stop mode and to issue a signal to wake the target MCU when it is time to resume processing. Unlike the earlier M68HC05 and M68HC08 MCUs, the HCS08 can be configured to keep a minimum set of clocks running in stop mode. This optionally allows an internal periodic signal to wake the target MCU from stop mode.

When a host debug system is connected to the background debug pin (BKGD) and the ENBDM control bit has been set by a serial command through the background interface (or because the MCU was reset into active background mode), the oscillator is forced to remain active when the MCU enters stop mode. In this case, if a serial BACKGROUND command is issued to the MCU through the background debug interface while the CPU is in stop mode, CPU clocks will resume and the CPU will enter active background mode where other serial background commands can be processed. This ensures that a host development system can still gain access to a target MCU even if it is in stop mode.

Recovery from stop mode depends on the particular HCS08 and whether the oscillator was stopped in stop mode. Refer to the [Modes of Operation](#) chapter for more details.

7.4.5 BGND Instruction

The BGND instruction is new to the HCS08 compared to the M68HC08. BGND would not be used in normal user programs because it forces the CPU to stop processing user instructions and enter the active background mode. The only way to resume execution of the user program is through reset or by a host debug system issuing a GO, TRACE1, or TAGGO serial command through the background debug interface.

Software-based breakpoints can be set by replacing an opcode at the desired breakpoint address with the BGND opcode. When the program reaches this breakpoint address, the CPU is forced to active background mode rather than continuing the user program.

7.5 HCS08 Instruction Set Summary

Table 7-2 provides a summary of the HCS08 instruction set in all possible addressing modes. The table shows operand construction, execution time in internal bus clock cycles, and cycle-by-cycle details for each addressing mode variation of each instruction.

Table 7-2. Instruction Set Summary (Sheet 1 of 9)

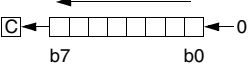
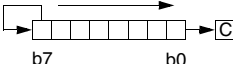
Source Form	Operation	Address Mode	Object Code	Cycles	Cyc-by-Cyc Details	Affect on CCR				
						VH	I	N	Z	C
ADC #opr8i ADC opr8a ADC opr16a ADC oprx16,X ADC oprx8,X ADC ,X ADC oprx16,SP ADC oprx8,SP	Add with Carry $A \leftarrow (A) + (M) + (C)$	IMM DIR EXT IX2 IX1 IX SP2 SP1	A9 ii B9 dd C9 hh ll D9 ee ff E9 ff F9 9E D9 ee ff 9E E9 ff	2 3 4 4 3 3 5 4	pp rpp prpp prpp rpp rfp pprpp prpp	↑↑	-	↑	↑	↑
ADD #opr8i ADD opr8a ADD opr16a ADD oprx16,X ADD oprx8,X ADD ,X ADD oprx16,SP ADD oprx8,SP	Add without Carry $A \leftarrow (A) + (M)$	IMM DIR EXT IX2 IX1 IX SP2 SP1	AB ii BB dd CB hh ll DB ee ff EB ff FB 9E DB ee ff 9E EB ff	2 3 4 4 3 3 5 4	pp rpp prpp prpp rpp rfp pprpp prpp	↑↑	-	↑	↑	↑
AIS #opr8i	Add Immediate Value (Signed) to Stack Pointer $SP \leftarrow (SP) + (M)$	IMM	A7 ii	2	pp	--	-	-	-	-
AIX #opr8i	Add Immediate Value (Signed) to Index Register (H:X) $H:X \leftarrow (H:X) + (M)$	IMM	AF ii	2	pp	--	-	-	-	-
AND #opr8i AND opr8a AND opr16a AND oprx16,X AND oprx8,X AND ,X AND oprx16,SP AND oprx8,SP	Logical AND $A \leftarrow (A) \& (M)$	IMM DIR EXT IX2 IX1 IX SP2 SP1	A4 ii B4 dd C4 hh ll D4 ee ff E4 ff F4 9E D4 ee ff 9E E4 ff	2 3 4 4 3 3 5 4	pp rpp prpp prpp rpp rfp pprpp prpp	0-	-	↑	↑	-
ASL opr8a ASLA ASLX ASL oprx8,X ASL ,X ASL oprx8,SP	Arithmetic Shift Left  (Same as LSL)	DIR INH INH IX1 IX SP1	38 dd 48 58 68 ff 78 9E 68 ff	5 1 1 5 4 6	rfwpp p p rfwpp rfwp prfwpp	↑-	-	↑	↑	↑
ASR opr8a ASRA ASRX ASR oprx8,X ASR ,X ASR oprx8,SP	Arithmetic Shift Right 	DIR INH INH IX1 IX SP1	37 dd 47 57 67 ff 77 9E 67 ff	5 1 1 5 4 6	rfwpp p p rfwpp rfwp prfwpp	↑-	-	↑	↑	↑
BCC rel	Branch if Carry Bit Clear (if C = 0)	REL	24 rr	3	ppp	--	-	-	-	-

Table 7-2. Instruction Set Summary (Sheet 2 of 9)

Source Form	Operation	Address Mode	Object Code	Cycles	Cyc-by-Cyc Details	Affect on CCR				
						VH	I	N	Z	C
BCLR <i>n,opr8a</i>	Clear Bit <i>n</i> in Memory ($M_n \leftarrow 0$)	DIR (b0)	11 dd	5	rfwpp	--	-	-	-	-
		DIR (b1)	13 dd	5	rfwpp					
		DIR (b2)	15 dd	5	rfwpp					
		DIR (b3)	17 dd	5	rfwpp					
		DIR (b4)	19 dd	5	rfwpp					
		DIR (b5)	1B dd	5	rfwpp					
		DIR (b6)	1D dd	5	rfwpp					
		DIR (b7)	1F dd	5	rfwpp					
BCS <i>rel</i>	Branch if Carry Bit Set (if C = 1) (Same as BLO)	REL	25 rr	3	ppp	--	-	-	-	-
BEQ <i>rel</i>	Branch if Equal (if Z = 1)	REL	27 rr	3	ppp	--	-	-	-	-
BGE <i>rel</i>	Branch if Greater Than or Equal To (if $N \oplus V = 0$) (Signed)	REL	90 rr	3	ppp	--	-	-	-	-
BGND	Enter active background if ENBDM=1 Waits for and processes BDM commands until GO, TRACE1, or TAGGO	INH	82	5+	fp...ppp	--	-	-	-	-
BGT <i>rel</i>	Branch if Greater Than (if $Z \mid (N \oplus V) = 0$) (Signed)	REL	92 rr	3	ppp	--	-	-	-	-
BHCC <i>rel</i>	Branch if Half Carry Bit Clear (if H = 0)	REL	28 rr	3	ppp	--	-	-	-	-
BHCS <i>rel</i>	Branch if Half Carry Bit Set (if H = 1)	REL	29 rr	3	ppp	--	-	-	-	-
BHI <i>rel</i>	Branch if Higher (if $C \mid Z = 0$)	REL	22 rr	3	ppp	--	-	-	-	-
BHS <i>rel</i>	Branch if Higher or Same (if C = 0) (Same as BCC)	REL	24 rr	3	ppp	--	-	-	-	-
BIH <i>rel</i>	Branch if IRQ Pin High (if IRQ pin = 1)	REL	2F rr	3	ppp	--	-	-	-	-
BIL <i>rel</i>	Branch if IRQ Pin Low (if IRQ pin = 0)	REL	2E rr	3	ppp	--	-	-	-	-
BIT # <i>opr8i</i> BIT <i>opr8a</i> BIT <i>opr16a</i> BIT <i>opr16,X</i> BIT <i>opr8,X</i> BIT <i>,X</i> BIT <i>opr16,SP</i> BIT <i>opr8,SP</i>	Bit Test (A) & (M) (CCR Updated but Operands Not Changed)	IMM DIR EXT IX2 IX1 IX SP2 SP1	A5 ii B5 dd C5 hh ll D5 ee ff E5 ff F5 9E D5 ee ff 9E E5 ff	2 3 4 4 3 3 5 4	pp rpp prpp prpp rpp rfp pprpp prpp	0-	-	↑	↓	-
BLE <i>rel</i>	Branch if Less Than or Equal To (if $Z \mid (N \oplus V) = 1$) (Signed)	REL	93 rr	3	ppp	--	-	-	-	-
BLO <i>rel</i>	Branch if Lower (if C = 1) (Same as BCS)	REL	25 rr	3	ppp	--	-	-	-	-
BLS <i>rel</i>	Branch if Lower or Same (if $C \mid Z = 1$)	REL	23 rr	3	ppp	--	-	-	-	-
BLT <i>rel</i>	Branch if Less Than (if $N \oplus V = 1$) (Signed)	REL	91 rr	3	ppp	--	-	-	-	-
BMC <i>rel</i>	Branch if Interrupt Mask Clear (if I = 0)	REL	2C rr	3	ppp	--	-	-	-	-
BMI <i>rel</i>	Branch if Minus (if N = 1)	REL	2B rr	3	ppp	--	-	-	-	-
BMS <i>rel</i>	Branch if Interrupt Mask Set (if I = 1)	REL	2D rr	3	ppp	--	-	-	-	-
BNE <i>rel</i>	Branch if Not Equal (if Z = 0)	REL	26 rr	3	ppp	--	-	-	-	-
BPL <i>rel</i>	Branch if Plus (if N = 0)	REL	2A rr	3	ppp	--	-	-	-	-

Table 7-2. Instruction Set Summary (Sheet 3 of 9)

Source Form	Operation	Address Mode	Object Code	Cycles	Cyc-by-Cyc Details	Affect on CCR				
						VH	I	N	Z	C
BRA <i>rel</i>	Branch Always (if I = 1)	REL	20 rr	3	ppp	--	---	---	---	---
BRCLR <i>n,opr8a,rel</i>	Branch if Bit <i>n</i> in Memory Clear (if (Mn) = 0)	DIR (b0)	01 dd rr	5	rpppp	--	---	---	---	↑
		DIR (b1)	03 dd rr	5	rpppp					
		DIR (b2)	05 dd rr	5	rpppp					
		DIR (b3)	07 dd rr	5	rpppp					
		DIR (b4)	09 dd rr	5	rpppp					
		DIR (b5)	0B dd rr	5	rpppp					
		DIR (b6)	0D dd rr	5	rpppp					
		DIR (b7)	0F dd rr	5	rpppp					
BRN <i>rel</i>	Branch Never (if I = 0)	REL	21 rr	3	ppp	--	---	---	---	---
BRSET <i>n,opr8a,rel</i>	Branch if Bit <i>n</i> in Memory Set (if (Mn) = 1)	DIR (b0)	00 dd rr	5	rpppp	--	---	---	---	↑
		DIR (b1)	02 dd rr	5	rpppp					
		DIR (b2)	04 dd rr	5	rpppp					
		DIR (b3)	06 dd rr	5	rpppp					
		DIR (b4)	08 dd rr	5	rpppp					
		DIR (b5)	0A dd rr	5	rpppp					
		DIR (b6)	0C dd rr	5	rpppp					
		DIR (b7)	0E dd rr	5	rpppp					
BSET <i>n,opr8a</i>	Set Bit <i>n</i> in Memory (Mn ← 1)	DIR (b0)	10 dd	5	rfwpp	--	---	---	---	---
		DIR (b1)	12 dd	5	rfwpp					
		DIR (b2)	14 dd	5	rfwpp					
		DIR (b3)	16 dd	5	rfwpp					
		DIR (b4)	18 dd	5	rfwpp					
		DIR (b5)	1A dd	5	rfwpp					
		DIR (b6)	1C dd	5	rfwpp					
		DIR (b7)	1E dd	5	rfwpp					
BSR <i>rel</i>	Branch to Subroutine PC ← (PC) + 0x0002 push (PCL); SP ← (SP) – 0x0001 push (PCH); SP ← (SP) – 0x0001 PC ← (PC) + <i>rel</i>	REL	AD rr	5	ssppp	--	---	---	---	---
CBEQ <i>opr8a,rel</i>	Compare and... Branch if (A) = (M)	DIR	31 dd rr	5	rpppp	--	---	---	---	---
CBEQA # <i>opr8i,rel</i>		IMM	41 ii rr	4	pppp					
CBEQX # <i>opr8i,rel</i>		IMM	51 ii rr	4	pppp					
CBEQ <i>opr8,X+,rel</i>		IX1+	61 ff rr	5	rpppp					
CBEQ <i>,X+,rel</i>		IX+	71 rr	5	rfppp					
CBEQ <i>opr8,SP,rel</i>		SP1	9E 61 ff rr	6	prpppp					
CLC	Clear Carry Bit (C ← 0)	INH	98	1	p	--	---	---	---	0
CLI	Clear Interrupt Mask Bit (I ← 0)	INH	9A	1	p	--	0	---	---	---
CLR <i>opr8a</i>	Clear M ← 0x00 A ← 0x00 X ← 0x00 H ← 0x00 M ← 0x00 M ← 0x00 M ← 0x00	DIR	3F dd	5	rfwpp	0	-	-	0	1
CLRA		INH	4F	1	p					
CLR X		INH	5F	1	p					
CLR H		INH	8C	1	p					
CLR <i>opr8,X</i>		IX1	6F ff	5	rfwpp					
CLR <i>,X</i>		IX	7F	4	rfwp					
CLR <i>opr8,SP</i>		SP1	9E 6F ff	6	prfwpp					

Table 7-2. Instruction Set Summary (Sheet 4 of 9)

Source Form	Operation	Address Mode	Object Code	Cycles	Cyc-by-Cyc Details	Affect on CCR				
						VH	I	N	Z	C
CMP #opr8i CMP opr8a CMP opr16a CMP oprx16,X CMP oprx8,X CMP ,X CMP oprx16,SP CMP oprx8,SP	Compare Accumulator with Memory A – M (CCR Updated But Operands Not Changed)	IMM DIR EXT IX2 IX1 IX SP2 SP1	A1 ii B1 dd C1 hh ll D1 ee ff E1 ff F1 9E D1 ee ff 9E E1 ff	2 3 4 4 3 3 5 4	pp rpp prpp prpp rpp rfp pprpp prpp	↓- ↓- ↓- ↓- ↓- ↓- ↓- ↓-	- - - - - - - -	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
COM opr8a COMA COMX COM oprx8,X COM ,X COM oprx8,SP	Complement (One's Complement) M ← (\overline{M}) = 0xFF – (M) A ← (\overline{A}) = 0xFF – (A) X ← (\overline{X}) = 0xFF – (X) M ← (\overline{M}) = 0xFF – (M) M ← (\overline{M}) = 0xFF – (M) M ← (\overline{M}) = 0xFF – (M)	DIR INH INH IX1 IX SP1	33 dd 43 53 63 ff 73 9E 63 ff	5 1 1 5 4 6	rfwpp p p rfwpp rfwp prfwpp	0- 0- 0- 0- 0- 0-	- - - - - -	↑ ↑ ↑ ↑ ↑ ↑	↑ ↑ ↑ ↑ ↑ ↑	1 1 1 1 1 1
CPHX opr16a CPHX #opr16i CPHX opr8a CPHX oprx8,SP	Compare Index Register (H:X) with Memory (H:X) – (M:M + 0x0001) (CCR Updated But Operands Not Changed)	EXT IMM DIR SP1	3E hh ll 65 jj kk 75 dd 9E F3 ff	6 3 5 6	prrfpp ppp rrfpp prrfpp	↓- ↓- ↓- ↓-	- - - -	↑ ↑ ↑ ↑	↑ ↑ ↑ ↑	↑ ↑ ↑ ↑
CPX #opr8i CPX opr8a CPX opr16a CPX oprx16,X CPX oprx8,X CPX ,X CPX oprx16,SP CPX oprx8,SP	Compare X (Index Register Low) with Memory X – M (CCR Updated But Operands Not Changed)	IMM DIR EXT IX2 IX1 IX SP2 SP1	A3 ii B3 dd C3 hh ll D3 ee ff E3 ff F3 9E D3 ee ff 9E E3 ff	2 3 4 4 3 3 5 4	pp rpp prpp prpp rpp rfp pprpp prpp	↓- ↓- ↓- ↓- ↓- ↓- ↓- ↓-	- - - - - - - -	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
DAA	Decimal Adjust Accumulator After ADD or ADC of BCD Values	INH	72	1	p	U- U- U- U-	- - - -	↑ ↑ ↑ ↑	↑ ↑ ↑ ↑	↑ ↑ ↑ ↑
DBNZ opr8a,rel DBNZA rel DBNZX rel DBNZ oprx8,X,rel DBNZ ,X,rel DBNZ oprx8,SP,rel	Decrement A, X, or M and Branch if Not Zero (if (result) ≠ 0) DBNZX Affects X Not H	DIR INH INH IX1 IX SP1	3B dd rr 4B rr 5B rr 6B ff rr 7B rr 9E 6B ff rr	7 4 4 7 6 8	rfwpppp fppp fppp rfwpppp rfwppp prfwpppp	-- -- -- -- -- --	- - - - - -	↑ ↑ ↑ ↑ ↑ ↑	↑ ↑ ↑ ↑ ↑ ↑	↑ ↑ ↑ ↑ ↑ ↑
DEC opr8a DECA DECX DEC oprx8,X DEC ,X DEC oprx8,SP	Decrement M ← (M) – 0x01 A ← (A) – 0x01 X ← (X) – 0x01 M ← (M) – 0x01 M ← (M) – 0x01 M ← (M) – 0x01	DIR INH INH IX1 IX SP1	3A dd 4A 5A 6A ff 7A 9E 6A ff	5 1 1 5 4 6	rfwpp p p rfwpp rfwp prfwpp	↓- ↓- ↓- ↓- ↓- ↓-	- - - - - -	↑ ↑ ↑ ↑ ↑ ↑	↑ ↑ ↑ ↑ ↑ ↑	- - - - - -
DIV	Divide A ← (H:A) ÷ (X); H ← Remainder	INH	52	6	fffffp	-- -- -- --	- - - -	↑ ↑ ↑ ↑	↑ ↑ ↑ ↑	↑ ↑ ↑ ↑
EOR #opr8i EOR opr8a EOR opr16a EOR oprx16,X EOR oprx8,X EOR ,X EOR oprx16,SP EOR oprx8,SP	Exclusive OR Memory with Accumulator A ← (A ⊕ M)	IMM DIR EXT IX2 IX1 IX SP2 SP1	A8 ii B8 dd C8 hh ll D8 ee ff E8 ff F8 9E D8 ee ff 9E E8 ff	2 3 4 4 3 3 5 4	pp rpp prpp prpp rpp rfp pprpp prpp	0- 0- 0- 0- 0- 0- 0- 0-	- - - - - - - -	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑	- - - - - - - -

Table 7-2. Instruction Set Summary (Sheet 6 of 9)

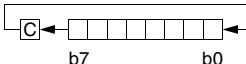
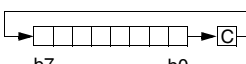
Source Form	Operation	Address Mode	Object Code	Cycles	Cyc-by-Cyc Details	Affect on CCR			
						VH	I	N	ZC
MOV <i>opr8a,opr8a</i> MOV <i>opr8a,X+</i> MOV <i>#opr8i,opr8a</i> MOV <i>,X+,opr8a</i>	Move $(M)_{\text{destination}} \leftarrow (M)_{\text{source}}$ In IX+/DIR and DIR/IX+ Modes, $H:X \leftarrow (H:X) + 0x0001$	DIR/DIR DIR/IX+ IMM/DIR IX+/DIR	4E dd dd 5E dd 6E ii dd 7E dd	5 5 4 5	rpwpp rfwpp pwpp rfwpp	0-	-	↑	↓
MUL	Unsigned multiply $X:A \leftarrow (X) \times (A)$	INH	42	5	fffffp	-0	-	-	0
NEG <i>opr8a</i> NEGA NEGX NEG <i>opr8,X</i> NEG <i>,X</i> NEG <i>opr8,SP</i>	Negate Two's Complement $M \leftarrow (M) = 0x00 - (M)$ $A \leftarrow (A) = 0x00 - (A)$ $X \leftarrow (X) = 0x00 - (X)$ $M \leftarrow (M) = 0x00 - (M)$ $M \leftarrow (M) = 0x00 - (M)$ $M \leftarrow (M) = 0x00 - (M)$	DIR INH INH IX1 IX SP1	30 dd 40 50 60 ff 70 9E 60 ff	5 1 1 5 4 6	rfwpp p p rfwpp rfwp prfwpp	↑-	-	↑	↓
NOP	No Operation — Uses 1 Bus Cycle	INH	9D	1	p	--	-	-	-
NSA	Nibble Swap Accumulator $A \leftarrow (A[3:0]:A[7:4])$	INH	62	1	p	--	-	-	-
ORA <i>#opr8i</i> ORA <i>opr8a</i> ORA <i>opr16a</i> ORA <i>opr16,X</i> ORA <i>opr8,X</i> ORA <i>,X</i> ORA <i>opr16,SP</i> ORA <i>opr8,SP</i>	Inclusive OR Accumulator and Memory $A \leftarrow (A) \mid (M)$	IMM DIR EXT IX2 IX1 IX SP2 SP1	AA ii BA dd CA hh ll DA ee ff EA ff FA 9E DA ee ff 9E EA ff	2 3 4 4 3 3 5 4	pp rpp prpp prpp rpp rfp pprpp prpp	0-	-	↑	↓
PSHA	Push Accumulator onto Stack $\text{Push } (A); SP \leftarrow (SP) - 0x0001$	INH	87	2	sp	--	-	-	-
PSHH	Push H (Index Register High) onto Stack $\text{Push } (H); SP \leftarrow (SP) - 0x0001$	INH	8B	2	sp	--	-	-	-
PSHX	Push X (Index Register Low) onto Stack $\text{Push } (X); SP \leftarrow (SP) - 0x0001$	INH	89	2	sp	--	-	-	-
PULA	Pull Accumulator from Stack $SP \leftarrow (SP + 0x0001); \text{Pull } (A)$	INH	86	3	ufp	--	-	-	-
PULH	Pull H (Index Register High) from Stack $SP \leftarrow (SP + 0x0001); \text{Pull } (H)$	INH	8A	3	ufp	--	-	-	-
PULX	Pull X (Index Register Low) from Stack $SP \leftarrow (SP + 0x0001); \text{Pull } (X)$	INH	88	3	ufp	--	-	-	-
ROL <i>opr8a</i> ROLA ROLX ROL <i>opr8,X</i> ROL <i>,X</i> ROL <i>opr8,SP</i>	Rotate Left through Carry 	DIR INH INH IX1 IX SP1	39 dd 49 59 69 ff 79 9E 69 ff	5 1 1 5 4 6	rfwpp p p rfwpp rfwp prfwpp	↑-	-	↑	↓
ROR <i>opr8a</i> RORA RORX ROR <i>opr8,X</i> ROR <i>,X</i> ROR <i>opr8,SP</i>	Rotate Right through Carry 	DIR INH INH IX1 IX SP1	36 dd 46 56 66 ff 76 9E 66 ff	5 1 1 5 4 6	rfwpp p p rfwpp rfwp prfwpp	↑-	-	↑	↓

Table 7-2. Instruction Set Summary (Sheet 7 of 9)

Source Form	Operation	Address Mode	Object Code	Cycles	Cyc-by-Cyc Details	Affect on CCR				
						VH	I	N	Z	C
RSP	Reset Stack Pointer (Low Byte) SPL ← 0xFF (High Byte Not Affected)	INH	9C	1	p	--	---	---	---	---
RTI	Return from Interrupt SP ← (SP) + 0x0001; Pull (CCR) SP ← (SP) + 0x0001; Pull (A) SP ← (SP) + 0x0001; Pull (X) SP ← (SP) + 0x0001; Pull (PCH) SP ← (SP) + 0x0001; Pull (PCL)	INH	80	9	uuuuufppp	↑↑		↑↑↑↑		
RTS	Return from Subroutine SP ← SP + 0x0001; Pull (PCH) SP ← SP + 0x0001; Pull (PCL)	INH	81	5	ufppp	--		---	---	---
SBC #opr8i SBC opr8a SBC opr16a SBC oprx16,X SBC oprx8,X SBC ,X SBC oprx16,SP SBC oprx8,SP	Subtract with Carry A ← (A) – (M) – (C)	IMM DIR EXT IX2 IX1 IX SP2 SP1	A2 ii B2 dd C2 hh ll D2 ee ff E2 ff F2 9E D2 ee ff 9E E2 ff	2 3 4 4 3 3 5 4	pp rpp prpp prpp rpp rpp pprpp prpp			↑-	-↑↑↑	
SEC	Set Carry Bit (C ← 1)	INH	99	1	p	--		---	---	1
SEI	Set Interrupt Mask Bit (I ← 1)	INH	9B	1	p	--	1	---	---	---
STA opr8a STA opr16a STA oprx16,X STA oprx8,X STA ,X STA oprx16,SP STA oprx8,SP	Store Accumulator in Memory M ← (A)	DIR EXT IX2 IX1 IX SP2 SP1	B7 dd C7 hh ll D7 ee ff E7 ff F7 9E D7 ee ff 9E E7 ff	3 4 4 3 2 5 4	wpp pwpp pwpp wpp wp ppwpp pwpp		0-	-↑↑-		
STHX opr8a STHX opr16a STHX oprx8,SP	Store H:X (Index Reg.) (M:M + 0x0001) ← (H:X)	DIR EXT SP1	35 dd 96 hh ll 9E FF ff	4 5 5	wwpp pwwpp pwwpp		0-	-↑↑-		
STOP	Enable Interrupts: Stop Processing Refer to MCU Documentation I bit ← 0; Stop Processing	INH	8E	2	fp...	--	0	---	---	---
STX opr8a STX opr16a STX oprx16,X STX oprx8,X STX ,X STX oprx16,SP STX oprx8,SP	Store X (Low 8 Bits of Index Register) in Memory M ← (X)	DIR EXT IX2 IX1 IX SP2 SP1	BF dd CF hh ll DF ee ff EF ff FF 9E DF ee ff 9E EF ff	3 4 4 3 2 5 4	wpp pwpp pwpp wpp wp ppwpp pwpp		0-	-↑↑-		

Table 7-2. Instruction Set Summary (Sheet 8 of 9)

Source Form	Operation	Address Mode	Object Code	Cycles	Cyc-by-Cyc Details	Affect on CCR				
						VH	I	N	Z	C
SUB #opr8i SUB opr8a SUB opr16a SUB oprx16,X SUB oprx8,X SUB ,X SUB oprx16,SP SUB oprx8,SP	Subtract $A \leftarrow (A) - (M)$	IMM DIR EXT IX2 IX1 IX SP2 SP1	A0 ii B0 dd C0 hh ll D0 ee ff E0 ff F0 9E D0 ee ff 9E E0 ff	2 3 4 4 3 3 5 4	pp rpp prpp prpp rpp rfp pprpp prpp		↓-			-↑↑↑
SWI	Software Interrupt $PC \leftarrow (PC) + 0x0001$ Push (PCL); $SP \leftarrow (SP) - 0x0001$ Push (PCH); $SP \leftarrow (SP) - 0x0001$ Push (X); $SP \leftarrow (SP) - 0x0001$ Push (A); $SP \leftarrow (SP) - 0x0001$ Push (CCR); $SP \leftarrow (SP) - 0x0001$ $I \leftarrow 1$; PCH ← Interrupt Vector High Byte PCL ← Interrupt Vector Low Byte	INH	83	11	sssssvvfppp	--	1	--	--	--
TAP	Transfer Accumulator to CCR $CCR \leftarrow (A)$	INH	84	1	p	↑↑			↑↑↑↑	
TAX	Transfer Accumulator to X (Index Register Low) $X \leftarrow (A)$	INH	97	1	p	--			--	--
TPA	Transfer CCR to Accumulator $A \leftarrow (CCR)$	INH	85	1	p	--			--	--
TST opr8a TSTA TSTX TST oprx8,X TST ,X TST oprx8,SP	Test for Negative or Zero (M) – 0x00 (A) – 0x00 (X) – 0x00 (M) – 0x00 (M) – 0x00 (M) – 0x00	DIR INH INH IX1 IX SP1	3D dd 4D 5D 6D ff 7D 9E 6D ff	4 1 1 4 3 5	rfpp p p rfpp rfp prfpp	0-			-↑↑-	
TSX	Transfer SP to Index Reg. $H:X \leftarrow (SP) + 0x0001$	INH	95	2	fp	--			--	--
TXA	Transfer X (Index Reg. Low) to Accumulator $A \leftarrow (X)$	INH	9F	1	p	--			--	--

Table 7-2. Instruction Set Summary (Sheet 9 of 9)

Source Form	Operation	Address Mode	Object Code	Cycles	Cyc-by-Cyc Details	Affect on CCR				
						VH	I	N	Z	C
TXS	Transfer Index Reg. to SP SP ← (H:X) – 0x0001	INH	94	2	f _p	--	---	---	---	---
WAIT	Enable Interrupts; Wait for Interrupt I bit ← 0; Halt CPU	INH	8F	2+	f _p . . .	--	0	---	---	---

Source Form: Everything in the source forms columns, *except expressions in italic characters*, is literal information which must appear in the assembly source file exactly as shown. The initial 3- to 5-letter mnemonic and the characters (#, () and +) are always a literal characters.

n Any label or expression that evaluates to a single integer in the range 0-7.

opr8i Any label or expression that evaluates to an 8-bit immediate value.

opr16i Any label or expression that evaluates to a 16-bit immediate value.

opr8a Any label or expression that evaluates to an 8-bit direct-page address (0x00xx).

opr16a Any label or expression that evaluates to a 16-bit address.

opr8 Any label or expression that evaluates to an unsigned 8-bit value, used for indexed addressing.

opr16 Any label or expression that evaluates to a 16-bit value, used for indexed addressing.

rel Any label or expression that refers to an address that is within –128 to +127 locations from the start of the next instruction.

Operation Symbols:

A Accumulator
 CCR Condition code register
 H Index register high byte
 M Memory location
n Any bit
opr Operand (one or two bytes)
 PC Program counter
 PCH Program counter high byte
 PCL Program counter low byte
rel Relative program counter offset byte
 SP Stack pointer
 SPL Stack pointer low byte
 X Index register low byte
 & Logical AND
 | Logical OR
 ⊕ Logical EXCLUSIVE OR
 () Contents of
 + Add
 – Subtract, Negation (two's complement)
 × Multiply
 ÷ Divide
 # Immediate value
 ← Loaded with
 : Concatenated with

CCR Bits:

V Overflow bit
 H Half-carry bit
 I Interrupt mask
 N Negative bit
 Z Zero bit
 C Carry/borrow bit

Addressing Modes:

DIR Direct addressing mode
 EXT Extended addressing mode
 IMM Immediate addressing mode
 INH Inherent addressing mode
 IX Indexed, no offset addressing mode
 IX1 Indexed, 8-bit offset addressing mode
 IX2 Indexed, 16-bit offset addressing mode
 IX+ Indexed, no offset, post increment addressing mode
 IX1+ Indexed, 8-bit offset, post increment addressing mode
 REL Relative addressing mode
 SP1 Stack pointer, 8-bit offset addressing mode
 SP2 Stack pointer 16-bit offset addressing mode

Cycle-by-Cycle Codes:

f Free cycle. This indicates a cycle where the CPU does not require use of the system buses. An f cycle is always one cycle of the system bus clock and is always a read cycle.
 p Program fetch; read from next consecutive location in program memory
 r Read 8-bit operand
 s Push (write) one byte onto stack
 u Pop (read) one byte from stack
 v Read vector from 0xFFxx (high byte first)
 w Write 8-bit operand

CCR Effects:

↑ Set or cleared
 – Not affected
 U Undefined

Table 7-3. Opcode Map (Sheet 1 of 2)

Bit-Manipulation		Branch		Read-Modify-Write				Control				Register/Memory																			
00 5 3	BRSET0 DIR	10 5 2	BSET0 DIR	20 3 2	BRA REL	30 5 2	NEG DIR	40 1 1	NEGA INH	50 1 1	NEGX INH	60 5 2	NEG IX1	70 4 1	NEG IX	80 9 1	RTI INH	90 2 2	BGE REL	A0 2 2	SUB IMM	B0 3 2	SUB DIR	C0 4 3	SUB EXT	D0 4 3	SUB IX2	E0 3 2	SUB IX1	F0 3 1	SUB IX
01 5 3	BRCLR0 DIR	11 5 2	BCLR0 DIR	21 3 2	BRN REL	31 5 3	CBEQ DIR	41 4 3	CBEQA IMM	51 4 3	CBEQX IMM	61 5 3	CBEQ IX1+	71 5 2	CBEQ IX+	81 6 1	RTS INH	91 3 2	BLT REL	A1 2 2	CMP IMM	B1 3 2	CMP DIR	C1 4 3	CMP EXT	D1 4 3	CMP IX2	E1 3 2	CMP IX1	F1 3 1	CMP IX
02 5 3	BRSET1 DIR	12 5 2	BSET1 DIR	22 3 2	BHI REL	32 5 3	LDHX EXT	42 5 1	MUL INH	52 6 1	DIV INH	62 1 2	NSA INH	72 1 1	DAA INH	82 5+ 1	BGND INH	92 3 2	BGT REL	A2 2 2	SBC IMM	B2 3 2	SBC DIR	C2 4 3	SBC EXT	D2 4 3	SBC IX2	E2 3 2	SBC IX1	F2 3 1	SBC IX
03 5 3	BRCLR1 DIR	13 5 2	BCLR1 DIR	23 3 2	BLS REL	33 5 3	COM DIR	43 1 1	COMA INH	53 1 1	COMX INH	63 5 2	COM IX1	73 4 1	COM IX	83 11 1	SWI INH	93 3 2	BLE REL	A3 2 2	CPX IMM	B3 3 2	CPX DIR	C3 4 3	CPX EXT	D3 4 3	CPX IX2	E3 3 2	CPX IX1	F3 3 1	CPX IX
04 5 3	BRSET2 DIR	14 5 2	BSET2 DIR	24 3 2	BCC REL	34 5 2	LSR DIR	44 1 1	LSRA INH	54 1 1	LSRX INH	64 5 2	LSR IX1	74 4 1	LSR IX	84 1 1	TAP INH	94 2 2	TXS INH	A4 2 2	AND IMM	B4 3 2	AND DIR	C4 4 3	AND EXT	D4 4 3	AND IX2	E4 3 2	AND IX1	F4 3 1	AND IX
05 5 3	BRCLR2 DIR	15 5 2	BCLR2 DIR	25 3 2	BCS REL	35 4 3	STHX DIR	45 3 3	LDHX IMM	55 4 2	LDHX DIR	65 3 3	CPHX IMM	75 5 3	CPHX DIR	85 1 1	TPA INH	95 2 1	TSX INH	A5 2 2	BIT IMM	B5 3 2	BIT DIR	C5 4 3	BIT EXT	D5 4 3	BIT IX2	E5 3 2	BIT IX1	F5 3 1	BIT IX
06 5 3	BRSET3 DIR	16 5 2	BSET3 DIR	26 3 2	BNE REL	36 5 2	ROR DIR	46 1 1	RORA INH	56 1 1	RORX INH	66 5 2	ROR IX1	76 4 1	ROR IX	86 3 1	PULA INH	96 5 3	STHX EXT	A6 2 2	LDA IMM	B6 3 2	LDA DIR	C6 4 3	LDA EXT	D6 4 3	LDA IX2	E6 3 2	LDA IX1	F6 3 1	LDA IX
07 5 3	BRCLR3 DIR	17 5 2	BCLR3 DIR	27 3 2	BEQ REL	37 5 2	ASR DIR	47 1 1	ASRA INH	57 1 1	ASRX INH	67 5 2	ASR IX1	77 4 1	ASR IX	87 2 1	PSHA INH	97 1 1	TAX INH	A7 2 2	AIS IMM	B7 3 2	STA DIR	C7 4 3	STA EXT	D7 4 3	STA IX2	E7 3 2	STA IX1	F7 3 1	STA IX
08 5 3	BRSET4 DIR	18 5 2	BSET4 DIR	28 3 2	BHCC REL	38 5 2	LSL DIR	48 1 1	LSLA INH	58 1 1	LSLX INH	68 5 2	LSL IX1	78 4 1	LSL IX	88 3 1	PULX INH	98 1 1	CLC INH	A8 2 2	EOR IMM	B8 3 2	EOR DIR	C8 4 3	EOR EXT	D8 4 3	EOR IX2	E8 3 2	EOR IX1	F8 3 1	EOR IX
09 5 3	BRCLR4 DIR	19 5 2	BCLR4 DIR	29 3 2	BHCS REL	39 5 2	ROL DIR	49 1 1	ROLA INH	59 1 1	ROLX INH	69 5 2	ROL IX1	79 4 1	ROL IX	89 2 1	PSHX INH	99 1 1	SEC INH	A9 2 2	ADC IMM	B9 3 2	ADC DIR	C9 4 3	ADC EXT	D9 4 3	ADC IX2	E9 3 2	ADC IX1	F9 3 1	ADC IX
0A 5 3	BRSET5 DIR	1A 5 2	BSET5 DIR	2A 3 2	BPL REL	3A 5 2	DEC DIR	4A 1 1	DECA INH	5A 1 1	DECX INH	6A 5 2	DEC IX1	7A 4 1	DEC IX	8A 3 1	PULH INH	9A 1 1	CLI INH	AA 2 2	ORA IMM	BA 3 2	ORA DIR	CA 4 3	ORA EXT	DA 4 3	ORA IX2	EA 3 2	ORA IX1	FA 3 1	ORA IX
0B 5 3	BRCLR5 DIR	1B 5 2	BCLR5 DIR	2B 3 2	BMI REL	3B 7 3	DBNZ DIR	4B 4 2	DBNZA INH	5B 4 2	DBNZX INH	6B 7 3	DBNZ IX1	7B 6 2	DBNZ IX	8B 2 1	PSHH INH	9B 1 1	SEI INH	AB 2 2	ADD IMM	BB 3 2	ADD DIR	CB 4 3	ADD EXT	DB 4 3	ADD IX2	EB 3 2	ADD IX1	FB 3 1	ADD IX
0C 5 3	BRSET6 DIR	1C 5 2	BSET6 DIR	2C 3 2	BMC REL	3C 5 2	INC DIR	4C 1 1	INCA INH	5C 1 1	INCX INH	6C 5 2	INC IX1	7C 4 1	INC IX	8C 1 1	CLRH INH	9C 1 1	RSP INH	AC 2 2	JMP	BC 3 2	JMP DIR	CC 4 3	JMP EXT	DC 4 3	JMP IX2	EC 3 2	JMP IX1	FC 3 1	JMP IX
0D 5 3	BRCLR6 DIR	1D 5 2	BCLR6 DIR	2D 3 2	BMS REL	3D 4 3	TST DIR	4D 1 1	TSTA INH	5D 1 1	TSTX INH	6D 4 2	TST IX1	7D 3 1	TST IX	8D 2 1	STOP INH	9D 1 1	NOP INH	AD 5 2	BSR REL	BD 5 2	JSR DIR	CD 6 3	JSR EXT	DD 6 3	JSR IX2	ED 5 2	JSR IX1	FD 5 1	JSR IX
0E 5 3	BRSET7 DIR	1E 5 2	BSET7 DIR	2E 3 2	BIL REL	3E 6 3	CPHX EXT	4E 5 3	MOV DD	5E 5 2	MOV DIX+	6E 4 3	MOV IMD	7E 5 2	MOV IX+D	8E 2+ 1	STOP INH	9E 2 2	Page 2	AE 2 2	LDX IMM	BE 3 2	LDX DIR	CE 4 3	LDX EXT	DE 4 3	LDX IX2	EE 3 2	LDX IX1	FE 3 1	LDX IX
0F 5 3	BRCLR7 DIR	1F 5 2	BCLR7 DIR	2F 3 2	BIH REL	3F 5 2	CLR DIR	4F 1 1	CLRA INH	5F 1 1	CLR INH	6F 5 2	CLR IX1	7F 4 1	CLR IX	8F 2+ 1	WAIT INH	9F 1 1	TXA INH	AF 2 2	AIX IMM	BF 3 2	STX DIR	CF 4 3	STX EXT	DF 4 3	STX IX2	EF 3 2	STX IX1	FF 3 1	STX IX

INH Inherent
 IMM Immediate
 DIR Direct
 EXT Extended
 DD DIR to DIR
 IX+D IX+ to DIR
 REL Relative
 IX Indexed, No Offset
 IX1 Indexed, 8-Bit Offset
 IX2 Indexed, 16-Bit Offset
 IMM to DIR
 DIR to IX+
 SP1 Stack Pointer, 8-Bit Offset
 SP2 Stack Pointer, 16-Bit Offset
 IX+ Indexed, No Offset with Post Increment
 IX1+ Indexed, 1-Byte Offset with Post Increment

Opcode in Hexadecimal F0 SUB 3
 Number of Bytes 1 SUB IX HCS08 Cycles Instruction Mnemonic Addressing Mode

Table 7-3. Opcode Map (Sheet 2 of 2)

Bit-Manipulation	Branch	Read-Modify-Write			Control			Register/Memory					
				9E60 NEG 3 SP1					9ED0 SUB 4 SP2	9EE0 SUB 3 SP1			
				9E61 CBEQ 4 SP1					9ED1 CMP 4 SP2	9EE1 CMP 3 SP1			
									9ED2 SBC 4 SP2	9EE2 SBC 3 SP1			
				9E63 COM 3 SP1					9ED3 CPX 4 SP2	9EE3 CPX 3 SP1	9EF3 CPHX 3 SP1		
				9E64 LSR 3 SP1					9ED4 AND 4 SP2	9EE4 AND 3 SP1			
									9ED5 BIT 4 SP2	9EE5 BIT 3 SP1			
				9E66 ROR 3 SP1					9ED6 LDA 4 SP2	9EE6 LDA 3 SP1			
				9E67 ASR 3 SP1					9ED7 STA 4 SP2	9EE7 STA 3 SP1			
				9E68 LSL 3 SP1					9ED8 EOR 4 SP2	9EE8 EOR 3 SP1			
				9E69 ROL 3 SP1					9ED9 ADC 4 SP2	9EE9 ADC 3 SP1			
				9E6A DEC 3 SP1					9EDA ORA 4 SP2	9EEA ORA 3 SP1			
				9E6B DBNZ 4 SP1					9EDB ADD 4 SP2	9EEB ADD 3 SP1			
				9E6C INC 3 SP1									
				9E6D TST 3 SP1									
								9EAE LDHX 2 IX	9EBE LDHX 4 IX2	9ECE LDHX 3 IX1	9EDE LDX 4 SP2	9EEE LDX 3 SP1	9EFE LDHX 3 SP1
				9E6F CLR 3 SP1					9EDF STX 4 SP2	9EEF STX 3 SP1	9EFF STHX 3 SP1		

INH Inherent REL Relative SP1 Stack Pointer, 8-Bit Offset
 IMM Immediate IX Indexed, No Offset SP2 Stack Pointer, 16-Bit Offset
 DIR Direct IX1 Indexed, 8-Bit Offset IX+ Indexed, No Offset with
 EXT Extended IX2 Indexed, 16-Bit Offset Post Increment
 DD DIR to DIR IMD IMM to DIR IX1+ Indexed, 1-Byte Offset with
 IX+D IX+ to DIR DIX+ DIR to IX+ Post Increment

Note: All Sheet 2 Opcodes are Preceded by the Page 2 Prebyte (9E)

Prebyte (9E) and Opcode in
 Hexadecimal

9E60	6
NEG	
3	SP1

 HCS08 Cycles
 Instruction Mnemonic
 Addressing Mode