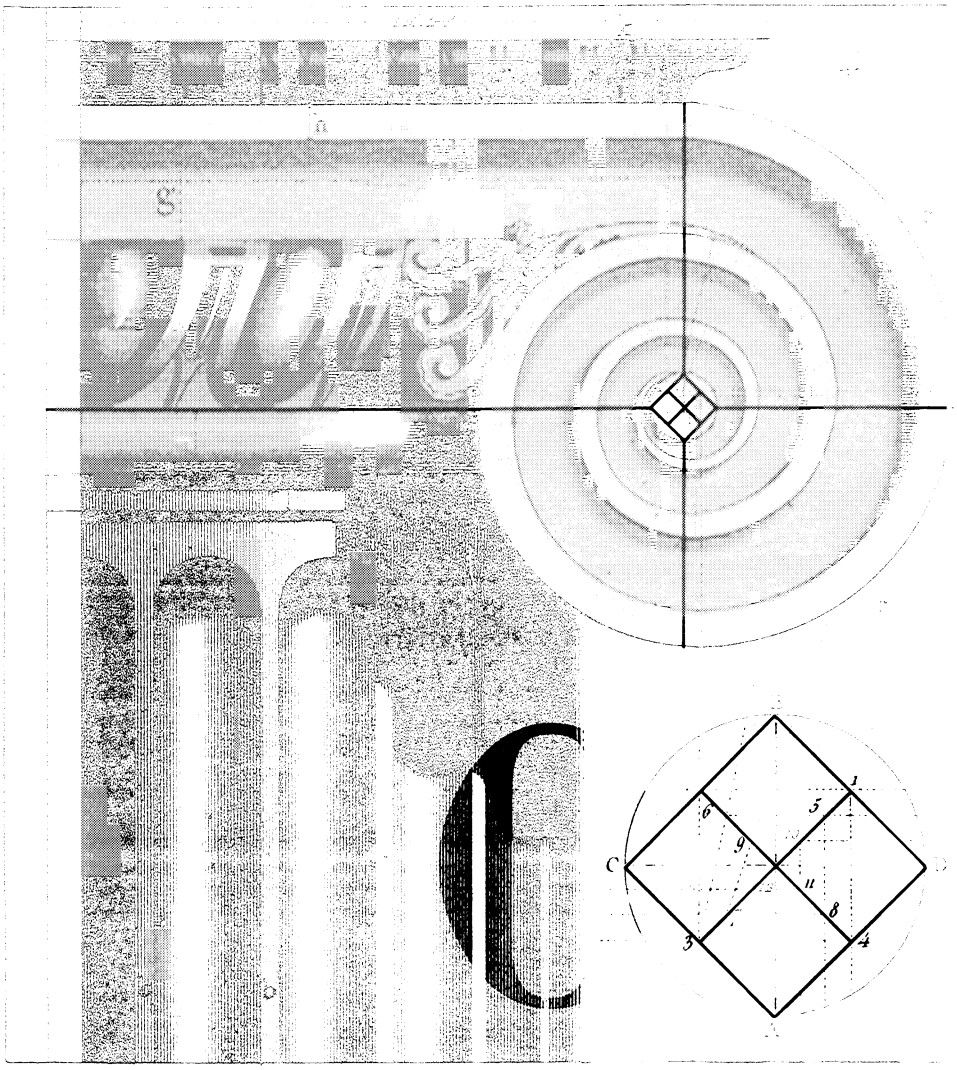
M68HC08RG/AD
Rev. 1

HC08

FAMILY REFERENCE GUIDE



MOTOROLA

M68HC08 INSTRUCTION SET

NEW HC08 INSTRUCTIONS

ADDRESSING MODES

PROGRAMMING MODEL
INTERRUPT STACKING ORDER

OPCODE MAP

ASCII CHART
HEX/DEC CONVERSION

M68HC08 INSTRUCTION SET

NEW HC08 INSTRUCTIONS

ADDRESSING MODES

PROGRAMMING MODEL
INTERRUPT STACKING ORDER

OPCODE MAP

ASCII CHART
HEX/DEC CONVERSION

M68HC08 INSTRUCTION SET

The following legend summarizes the symbols and abbreviations used.

Boolean Operators

( )	Contents of	⊕	Boolean exclusive-OR
-( )	Negate (two's complement)	?	If
←	Is loaded with (read: "gets")	«	Sign extend
↑	Is pulled from stack	×	Multiply
↓	Is pushed onto stack	÷	Divide
&	Boolean AND	:	Concatenate
|	Boolean OR	+	Add
		n	Any bit

CPU Registers

A	Accumulator	PC	Program counter
CCR	Condition code register	PCH	PC high byte
H	Index register, high byte	PCL	PC low byte
X	Index register, low byte	SP	Stack pointer

M68HC08 INSTRUCTION SET

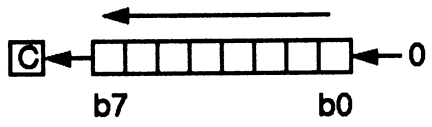
Condition Code Register (CCR) Bits

V	Overflow indicator	0	Bit forced to zero
I	Interrupt mask, bit 3	H	Half carry, bit 4
Z	Zero indicator, bit 1	N	Negative indicator, bit 2
U	Undefined	C	Carry/borrow, bit 0
–	Bit not affected	1	Bit forced to one
		↕	Bit set or cleared

Addressing Modes

INH	Inherent (no operands)	IX2	16-bit indexed with 16-bit offset (ee ff)
IMM	8-bit immediate (ii)	REL	8-bit relative offset (rr)
DIR	8-bit direct (dd, dd rr)	DD	Direct to direct
EXT	16-bit extended (hh ll)	IMD	Immediate to direct
IX	16-bit indexed no offset (no operands)	IX+D	16-bit indexed, post increment source to direct
IX+	16-bit indexed no offset, post increment	DIX+	Direct source to 16-bit indexed, post increment
IX1	16-bit indexed with 8-bit offset (ff)	SP1	Stack pointer with 8-bit offset
IX1+	16-bit indexed with 8-bit offset, post increment	SP2	Stack pointer with 16-bit offset
M	Memory location	*rel*	Relative offset
		opr	Operand

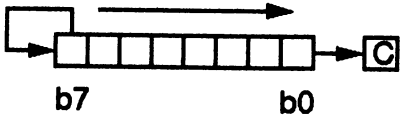
The following table is a listing of the HC08 instructions.

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
AIX #*opr*	Add Immediate Value (Signed) to Index Register (H:X)	H:X ← (H:X) + (16 « M)	–	–	–	–	–	–	IMM	AF	ii	2
AND #*opr*	Logical AND	A ← (A) & (M)	0	–	–	↕	↕	–	IMM	A4	ii	2
AND *opr*									DIR	B4	dd	3
AND *opr*									EXT	C4	hh ll	4
AND *opr*,X									IX2	D4	ee ff	4
AND *opr*,X									IX1	E4	ff	3
AND ,X									IX	F4		2
AND *opr*,SP									SP1	9EE4	ff	4
AND *opr*,SP									SP2	9ED4	ee ff	5
ASL *opr*	Arithmetic Shift Left (Same as LSL)		↕	–	–	↕	↕	↕	DIR	38	dd	4
ASLA									INH	48		1
ASLX									INH	58		1
ASL *opr*,X									IX1	68	ff	4
ASL ,X									IX	78		3
ASL *opr*,SP									SP1	9E68	ff	5

Instruction Set

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
ADC #*opr* ADC *opr* ADC *opr* ADC *opr*,X ADC *opr*,X ADC ,X ADC *opr*,SP ADC *opr*,SP	Add with Carry	A ← (A) + (M) + (C)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A9 B9 C9 D9 E9 F9 9EE9 9ED9	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
ADD #*opr* ADD *opr* ADD *opr* ADD *opr*,X ADD *opr*,X ADD ,X ADD *opr*,SP ADD *opr*,SP	Add without Carry	A ← (A) + (M)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	AB BB CB DB EB FB 9EEB 9EDB	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
AIS #*opr*	Add Immediate Value (Signed) to Stack Pointer	SP ← (SP) + (16 « M)	–	–	–	–	–	–	IMM	A7	ii	2

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
BEQ *rel*	Branch if Equal	PC ← (PC) + $0002 + *rel* ? (Z) = 1	–	–	–	–	–	–	REL	27	rr	3
BGE *opr*	Branch if Greater Than or Equal To (Signed Operands)	PC ← (PC) + $0002 + *rel* ? (N ⊕ V) = 0	–	–	–	–	–	–	REL	90	rr	3
BGT *opr*	Branch if Greater Than (Signed Operands)	PC ← (PC) + $0002 + *rel* ? (Z) | (N ⊕ V) = 0	–	–	–	–	–	–	REL	92	rr	3
BHCC *rel*	Branch if Half Carry Bit Clear	PC ← (PC) + $0002 + *rel* ? (H) = 0	–	–	–	–	–	–	REL	28	rr	3
BHCS *rel*	Branch if Half Carry Bit Set	PC ← (PC) + $0002 + *rel* ? (H) = 1	–	–	–	–	–	–	REL	29	rr	3
BHI *rel*	Branch if Higher	PC ← (PC) + $0002 + *rel* ? (C) | (Z) = 0	–	–	–	–	–	–	REL	22	rr	3
BHS *rel*	Branch if Higher or Same (Same as BCC)	PC ← (PC) + $0002 + *rel* ? (C) = 0	–	–	–	–	–	–	REL	24	rr	3

ASR *opr* ASRA ASRX ASR *opr*,X ASR *opr*,X ASR *opr*,SP	Arithmetic Shift Right		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	37 47 57 67 77 9E67	dd ff ff	4 1 1 4 3 5
BCC *rel*	Branch if Carry Bit Clear	PC ← (PC) + $0002 + rel ? (C) = 0	–	–	–	–	–	–	REL	24	rr	3
BCLR *n*, *opr*	Clear Bit n in Memory	Mn ← 0	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	11 13 15 17 19 1B 1D 1F	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4
BCS *rel*	Branch if Carry Bit Set (Same as BLO)	PC ← (PC) + $0002 + *rel* ? (C) = 1	–	–	–	–	–	–	REL	25	rr	3

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
BMI *rel*	Branch if Minus	PC ← (PC) + $0002 + *rel* ? (N) = 1	–	–	–	–	–	–	REL	2B	rr	3
BMS *rel*	Branch if Interrupt Mask Set	PC ← (PC) + $0002 + *rel* ? (I) = 1	–	–	–	–	–	–	REL	2D	rr	3
BNE *rel*	Branch if Not Equal	PC ← (PC) + $0002 + *rel* ? (Z) = 0	–	–	–	–	–	–	REL	26	rr	3
BPL *rel*	Branch if Plus	PC ← (PC) + $0002 + *rel* ? (N) = 0	–	–	–	–	–	–	REL	2A	rr	3
BRA *rel*	Branch Always	PC ← (PC) + $0002 + *rel*	–	–	–	–	–	–	REL	20	rr	3
BRCLR *n*,*opr*,*rel*	Branch if Bit *n* in Memory Clear	PC ← (PC) + $0003 + *rel* ? (Mn) = 0	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5

BIH *rel*	Branch if $\overline{\text{IRQ}}$ Pin High	PC ← (PC) + $0002 + *rel* ? $\overline{\text{IRQ1}}$ = 1	–	–	–	–	–	–	REL	2F	rr	3
BIL *rel*	Branch if $\overline{\text{IRQ}}$ Pin Low	PC ← (PC) + $0002 + *rel* ? $\overline{\text{IRQ}}$ = 0	–	–	–	–	–	–	REL	2E	rr	3
BIT #*opr* BIT *opr* BIT *opr* BIT *opr*,X BIT *opr*,X BIT ,X BIT *opr*,SP BIT *opr*,SP	Bit Test	(A) & (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A5 B5 C5 D5 E5 F5 9EE5 9ED5	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
BLE *opr*	Branch if Less Than or Equal To (Signed Operands)	PC ← (PC) + $0002 + *rel* ? (Z) | (N ⊕ V) = 1	–	–	–	–	–	–	REL	93	rr	3
BLO *rel*	Branch if Lower (Same as BCS)	PC ← (PC) + $0002 + *rel* ? (C) = 1	–	–	–	–	–	–	REL	25	rr	3
BLS *rel*	Branch if Lower or Same	PC ← (PC) + $0002 + *rel*? (C) | (Z) = 1	–	–	–	–	–	–	REL	23	rr	3
BLT *opr*	Branch if Less Than (Signed Operands)	PC ← (PC) + $0002 + *rel* ? (N ⊕ V) =1	–	–	–	–	–	–	REL	91	rr	3
BMC *rel*	Branch if Interrupt Mask Clear	PC ← (PC) + $0002 + *rel* ? (I) = 0	–	–	–	–	–	–	REL	2C	rr	3

BRN *rel*	Branch Never	PC ← (PC) + $0002	–	–	–	–	–	–	REL	21	rr	3
BRSET *n*, *opr*, *rel*	Branch if Bit *n* in Memory Set	PC ← (PC) + $0003 + *rel* ? (Mn) = 1	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5
BSET *n*, *opr*	Set Bit *n* in Memory	Mn ← 1	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 12 14 16 18 1A 1C 1E	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
BSR *rel*	Branch to Subroutine	PC ← (PC) + $0002; push (PCL) SP ← (SP) – $0001; push (PCH) SP ← (SP) – $0001 PC ← (PC) + *rel*	–	–	–	–	–	–	REL	AD	rr	4
CBEQ *opr,rel* CBEQA #*opr,rel* CBEQX #*opr,rel* CBEQ *opr*,X+,*rel* CBEQ X+,*rel* CBEQ *opr*,SP,*rel*	Compare and Branch if Equal	PC ← (PC) + $0003 + *rel* ? (A) – (M) = $00 PC ← (PC) + $0003 + *rel* ? (A) – (M) = $00 PC ← (PC) + $0003 + *rel* ? (X) – (M) = $00 PC ← (PC) + $0003 + *rel* ? (A) – (M) = $00 PC ← (PC) + $0002 + *rel* ? (A) – (M) = $00 PC ← (PC) + $0004 + *rel* ? (A) – (M) = $00	–	–	–	–	–	–	DIR IMM IMM IX1+ IX+ SP1	31 41 51 61 71 9E61	dd rr ii rr ii rr rr ff rr ff rr	5 4 4 5 4 6
CLC	Clear Carry Bit	C ← 0	–	–	–	–	–	0	INH	98		1
CLI	Clear Interrupt Mask Bit	I ← 0	–	–	0	–	–	–	INH	9A		2

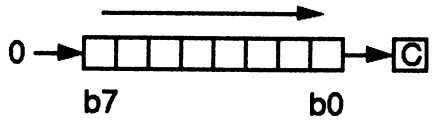
CLR *opr* CLRA CLRX CLRH CLR *opr*,X CLR ,X CLR *opr*,SP	Clear	M ← $00 A ← $00 X ← $00 H ← $00 M ← $00 M ← $00 M ← $00	0	–	–	0	1	–	DIR INH INH INH IX1 IX SP1	3F 4F 5F 8C 6F 7F 9E6F	dd ff ff	3 1 1 1 3 2 4
CMP #*opr* CMP *opr* CMP *opr* CMP *opr*,X CMP *opr*,X CMP ,X CMP *opr*,SP CMP *opr*,SP	Compare Accumulator with Memory	(A) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A1 B1 C1 D1 E1 F1 9EE1 9ED1	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
COM *opr* COMA COMX COM *opr*,X COM ,X COM *opr*,SP	Complement (One's Complement)	M ← ($\overline{M}$) = $FF – (M) A ← ($\overline{A}$) = $FF – (M) X ← ($\overline{X}$) = $FF – (M) M ← ($\overline{M}$) = $FF – (M) M ← ($\overline{M}$) = $FF – (M) M ← ($\overline{M}$) = $FF – (M)	0	–	–	↕	↕	1	DIR INH INH IX1 IX SP1	33 43 53 63 73 9E63	dd ff ff	4 1 1 4 3 5

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
CPHX #*opr* CPHX *opr*	Compare Index Register (H:X) with Memory	(H:X) – (M:M + $0001)	↕	–	–	↕	↕	↕	IMM DIR	65 75	ii ii+1 dd	3 4
CPX #*opr* CPX *opr* CPX *opr* CPX ,X CPX *opr*,X CPX *opr*,X CPX *opr*,SP CPX *opr*,SP	Compare X (Index Register Low) with Memory	(X) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A3 B3 C3 D3 E3 F3 9EE3 9ED3	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
DAA	Decimal Adjust Accumulator	$(A)_{10}$	U	–	–	↕	↕	↕	INH	72		2

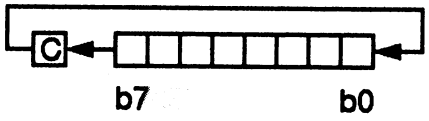
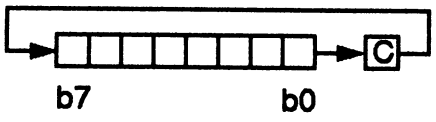
DBNZ *opr,rel* DBNZA *rel* DBNZX *rel* DBNZ *opr,X,rel* DBNZ *X,rel* DBNZ *opr,SP,rel*	Decrement and Branch if Not Zero	A ← (A) –$0001 or M ← (M) – $0001 or X ← (X) – $0001 PC ← (PC) + $0003 + *rel* if (result) ≠ 0 for DBNZ direct, IX1 PC ← (PC) + $0002 + *rel* if (result) ≠ 0 for DBNZA, DBNZX, or IX PC ← (PC) + $0004 + *rel* if (result) ≠ 0 for DBNZ SP1	–	–	–	–	–	–	DIR INH INH IX1 IX SP1	3B 4B 5B 6B 7B 9E6B	dd rr rr rr ff rr rr ff rr	5 3 3 5 4 6
DEC *opr* DECA DECX DEC *opr,X* DEC ,X DEC *opr,SP*	Decrement	M ← (M) – $01 A ← (A) – $01 X ← (X) – $01 M ← (M) – $01 M ← (M) – $01 M ← (M) – $01	↕	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3A 4A 5A 6A 7A 9E6A	dd ff ff	4 1 1 4 3 5

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
DIV	Divide	A ← (H:A)/(X) H ← Remainder	–	–	–	–	↕	↕	INH	52		7
EOR #*opr*	Exclusive OR Memory with Accumulator	A ← (A) ⊕ (M)	0	–	–	↕	↕	–	IMM	A8	ii	2
EOR *opr*									DIR	B8	dd	3
EOR *opr*									EXT	C8	hh ll	4
EOR *opr*,X									IX2	D8	ee ff	4
EOR *opr*,X									IX1	E8	ff	3
EOR ,X									IX	F8		2
EOR *opr*,SP									SP1	9EE8	ff	4
EOR *opr*,SP									SP2	9ED8	ee ff	5
INC *opr*	Increment	M ← (M) + $01	↕	–	–	↕	↕	–	DIR	3C	dd	4
INCA		A ← (A) + $01							INH	4C		1
INCX		X ← (X) + $01							INH	5C		1
INC opr,X		M ← (M) + $01							IX	7C		3
INC ,X		M ← (M) + $01							IX1	6C	ff	4
INC *opr*,SP		M ← (M) + $01							SP1	9E6C	ff	5

JMP *opr* JMP *opr* JMP *opr*,X JMP *opr*,X JMP ,X	Jump	PC ← Jump Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BC CC DC EC FC	dd hh ll ee ff ff	2 3 4 3 2
JSR *opr* JSR *opr* JSR *opr*,X JSR *opr*,X JSR ,X	Jump to Subroutine	PC ← (PC) + *n* (*n* = 1, 2, or 3) Push (PCL); SP ← (SP) – $0001 Push (PCH); SP ← (SP) – $0001 PC ← Unconditional Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff	4 5 6 5 4
LDA #*opr* LDA *opr* LDA *opr* LDA *opr*,X LDA *opr*,X LDA ,X LDA *opr*,SP LDA *opr*,SP	Load Accumulator from Memory	A ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A6 B6 C6 D6 E6 F6 9EE6 9ED6	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

LSR *opr* LSRA LSR*X* LSR *opr*,X LSR ,X LSR *opr*,SP	Logical Shift Right		↕	–	–	0	↕	↕	DIR INH INH IX1 IX SP1	34 54 44 64 74 9E64	dd ff ff	4 1 1 4 3 5
MOV *opr,opr* MOV *opr*,X+ MOV #*opr,opr* MOV X+,*opr*	Move	$(M)_{destination} \leftarrow (M)_{source}$ H:X ← (H:X) + $001 in IX+D and DIX+ Modes	0	–	–	↕	↕	–	DD DIX+ IMD IX+D	4E 5E 6E 7E	dd dd dd ii dd dd	5 4 4 4
MUL	Unsigned Multiply	X : A ← (X) × (A)	–	0	–	–	–	0	INH	42		5
NEG *opr* NEGA NEGX NEG *opr*,X NEG ,X NEG *opr*,SP	Negate (Two's Complement)	M ← −(M) = $00 − (M) A ← −(A) = $00 − (A) X ← −(X) = $00 − (X) M ← −(M) = $00 − (M) M ← −(M) = $00 − (M)	↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	30 40 50 60 70 9E60	dd ff ff	4 1 1 4 3 5

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
NOP	No Operation	None	–	–	–	–	–	–	INH	9D		1
NSA	Nibble Swap Accumulator	A ← (A[3:0]:A[7:4])	–	–	–	–	–	–	INH	62		3
ORA #*opr* ORA *opr* ORA *opr* ORA *opr*,X ORA *opr*,X ORA ,X ORA *opr*,SP ORA *opr*,SP	Inclusive OR Accumulator and Memory	A ← (A) | (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	AA BA CA DA EA FA 9EEA 9EDA	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
PSHA	Push Accumulator onto Stack	Push (A); SP ← (SP) – $0001	–	–	–	–	–	–	INH	87		2
PSHH	Push Index Register H onto Stack	Push (H); SP ← (SP) – $0001	–	–	–	–	–	–	INH	8B		2

PSHX	Push Index Register X onto Stack	Push (X); SP ← (SP) – $0001	–	–	–	–	–	–	INH	89		2
PULA	Pull Accumulator from Stack	SP ← (SP + $0001); Pull (A)	–	–	–	–	–	–	INH	86		2
PULH	Pull H from Stack	SP ← (SP + $0001); Pull (H)	–	–	–	–	–	–	INH	8A		2
PULX	Pull X from Stack	SP ← (SP + $0001); Pull (X)	–	–	–	–	–	–	INH	88		2
ROL *opr* ROLA ROLX ROL *opr*,X ROL ,X ROL *opr*,SP	Rotate Left through Carry		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	39 49 59 69 79 9E69	dd ff ff	4 1 1 4 3 5
ROR *opr* RORA RORX ROR *opr*,X ROR ,X ROR *opr*,SP	Rotate Right through Carry		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	36 46 56 66 76 9E66	dd ff ff	4 1 1 4 3 5

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
SUB #*opr* SUB *opr* SUB *opr* SUB *opr*,X SUB *opr*,X SUB ,X SUB *opr*,SP SUB *opr*,SP	Subtract	A ← (A) - (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A0 B0 C0 D0 E0 F0 9EE0 9ED0	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5
SWI	Software Interrupt	PC ← (PC) + $0001; Push (PCL) SP ← (SP) - $0001; Push (PCH) SP ← (SP) - $0001; Push (X) SP ← (SP) - $0001; Push (A) SP ← (SP) - $0001; Push (CCR) SP ← (SP) - $0001; I ← 1 PCH ← Interrupt Vector High Byte PCL ← Interrupt Vector Low Byte	–	–	1	–	–	–	INH	83		9
TAP	Transfer Accumulator to CCR	CCR ← (A)	↕	↕	↕	↕	↕	↕	INH	84		2

SEC	Set Carry Bit	C ← 1	–	–	–	–	–	1	INH	99		1
SEI	Set Interrupt Mask Bit	I ← 1	–	–	1	–	–	–	INH	9B		2
STA *opr* STA *opr* STA *opr*,X STA *opr*,X STA ,X STA *opr*,SP STA *opr*,SP	Store Accumulator in Memory	M ← (A)	0	–	–	↕	↕	–	DIR EXT IX2 IX1 IX SP1 SP2	B7 C7 D7 E7 F7 9EE7 9ED7	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5
STHX *opr*	Store Index Reg. (H:X)	(M:M + $0001) ← (H:X)	0	–	–	↕	↕	–	DIR	35	dd	4
STOP	Enable $\overline{\text{IRQ}}$ Pin; Stop Osc.	I bit ← 0; Stop Oscillator	–	–	0	–	–	–	INH	8E		1
STX *opr* STX *opr* STX *opr*,X STX *opr*,X STX ,X STX *opr*,SP STX *opr*,SP	Store X (Index Register Low) in Memory	M ← (X)	0	–	–	↕	↕	–	DIR EXT IX2 IX1 IX SP1 SP2	BF CF DF EF FF 9EEF 9EDF	dd hh ll ee ff ff ff ee ff	3 4 4 3 2 4 5

TAX	Transfer Accumulator to X (Index Register Low)	X ← (A)	–	–	–	–	–	–	INH	97		1
TPA	Transfer CCR to Accumulator	A ← (CCR)	–	–	–	–	–	–	INH	85		1
TST *opr* TSTA TSTX TST *opr*,X TST ,X TST *opr*,SP	Test for Negative or Zero	(A) – $00 (X) – $00 (M) – $00	0	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3D 4D 5D 6D 7D 9E6D	dd ff ff	3 1 1 3 2 4
TSX	Transfer Stack Pointer to Index Register	H:X ← (SP + $0001)	–	–	–	–	–	–	INH	95		2
TXA	Transfer X (Index Register Low) to Accumulator	A ← (X)	–	–	–	–	–	–	INH	9F		1
TXS	Transfer Index Register to Stack Pointer	(SP) ← (H:X) – $0001	–	–	–	–	–	–	INH	94		2
WAIT	Enable Interrupts; Stop Processor	I Bit ← 0	–	–	0	–	–	–	INH	8F		1

NEW HC08 INSTRUCTIONS

The M68HC08 Family instruction set is an extension of the M68HC05 Family instruction set. This section contains the instructions unique to the M68HC08 Family. Following is a list of new instructions.

- Add Immediate Value (Signed) to Stack Pointer (AIS)
- Add Immediate Value (Signed) to Index Register (AIX)
- Branch if Greater Than or Equal To (BGE)
- Branch if Greater Than (BGT)
- Branch if Less Than or Equal To (BLE)
- Branch if Less Than (BLT)
- Compare and Branch if Equal (CBEQ)
- Compare Accumulator with Immediate, Branch if Equal (CBEQA)
- Compare Index Register Low with Immediate, Branch if Equal (CBEQX)
- Clear Index Register High (CLRH)
- Compare Index Register with Memory (CPHX)
- Decimal Adjust Accumulator (DAA)
- Decrement and Branch if Not Zero (DBNZ)
- Divide (DIV)
- Load Index Register with Memory (LDHX)
- Move (MOV)
- Nibble Swap Accumulator (NSA)
- Push Accumulator onto Stack (PSHA)
- Push Index Register High onto Stack (PSHH)
- Push Index Register Low onto Stack (PSHX)
- Pull Accumulator from Stack (PULA)
- Pull Index Register High from Stack (PULH)
- Pull Index Register Low from Stack (PULX)
- Store Index Register (STHX)
- Transfer Accumulator to Condition Code Register (TAP)
- Transfer Condition Code Register to Accumulator (TPA)
- Transfer Stack Pointer to Index Register (TSX)
- Transfer Index Register to Stack Pointer (TXS)

Add Immediate Value (Signed) to Stack Pointer

Operation:

SP ← (SP) + (16 « M)

Description:

Adds the immediate operand to the stack pointer. The immediate value is an 8-bit two's complement signed operand. The 8-bit operand is sign-extended to 16 bits prior to the addition. The AIS instruction can be used to create and remove a stack frame buffer that is used to store temporary variables.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
AIS *#opr*	IMM	A7	ii ii	2

Add Immediate Value (Signed) to Index Register

Operation:

H:X ← (H:X) + (16 « M)

Description:

Adds an immediate operand to index register (H:X) formed by the concatenation of the H and X registers. The immediate operand is an 8-bit two's complement signed offset. The 8-bit operand is sign-extended to 16 bits prior to the addition.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
AIX #*opr*	IMM	AF	ii	2

Branch if Greater Than or Equal To (Signed Operands)

Operation:

PC ← (PC) + $0002 + *rel* if (N ⊕ V) = 0
i.e., if (A) (M) (two's complement signed numbers)

Description:

If the BGE instruction is executed immediately after execution of a compare or subtract instruction, branch occurs if and only if the two's complement number represented by the appropriate internal register (A, X, or H:X) was greater than or equal to the two's complement number represented by M.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
BGE *opr*	REL	90	rr	3

BGT

Branch if Greater Than (Signed Operands)

Operation:

PC ← (PC) + $0002 + *rel* if Z | (N ⊕ V) = 0
i.e., if (A) > (M) (two's complement signed numbers)

Description:

If the BGT instruction is executed immediately after execution of CMP, CPX, CPHX, or SUB, branch will occur if and only if the two's complement number represented by the appropriate internal register (A, X, or H:X) was greater than the two's complement number represented by M.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
BGT *opr*	REL	92	rr	3

Branch if Less Than or Equal To (Signed Operands)

Operation:

PC ← (PC) + $0002 + *rel* if Z | (N ⊕ V) = 1
i.e., if (A) (M) (two's complement signed numbers)

Description:

If the BLE instruction is executed immediately after execution of CMP, CPX, CPHX, or SUB, branch will occur if and only if the two's complement number represented by the appropriate internal register (A, X, or H:X) was less than or equal to the two's complement number represented by M.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
BLE *opr*	REL	93	rr	3

BLT

Branch if Less Than (Signed Operands)

Operation:

PC ← (PC) + $0002 + *rel* if (N ⊕ V) = 1
i.e., if (A) < (M) (two's complement signed numbers)

Description:

If the BLT instruction is executed immediately after execution of CMP, CPX, CPHX, or SUB, branch will occur if and only if the two's complement number represented by the appropriate internal register (A, X, or H:X) was less than the two's complement number represented by M.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
BLT *opr*	REL	91	rr	3

Compare and Branch if Equal

Operation:

(A) – (M); PC ← (PC) + $0003 + *rel*
if result is $00
or: for IX+ mode: (A) – (M); PC ← (PC) + $0002 + *rel*
if result is $00
or: for SP1 mode: PC ← (PC) + $0004 + *rel*
if result is $00

Description:

CBEQ compares the operand with the accumulator (A) and causes a branch if the result is zero. The CBEQ instruction combines CMP and BEQ for faster table lookup routines.

CBEQ_IX+ compares the operand addressed by H:X to the A and causes a branch if the result is zero. H:X is then incremented regardless of whether a branch is taken. CBEQ_IX1+ operates the same way except that an 8-bit offset is added to the effective address of the operand.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
CBEQ *opr*	DIR	31	dd rr	5
CBEQA *#opr, rel*	IMM	41	ii rr	4
CBEQX *#opr, rel*	IMM	51	ii rr	4
CBEQ X+, *rel*	IX+	71	rr	4
CBEQ *opr*, X+, *rel*	IX1+	61	ff rr	5
CBEQ *opr*, SP, *rel*	SP1	9E61	ff rr	6

CBEQA

Compare A with Immediate, Branch if Equal

Operation:

(A) – (M); PC ← (PC) + $0003 + *rel*
if result is $00

Description:

CBEQ compares an immediate operand with the accumulator (A) and causes a branch if the result is zero. The CBEQA instruction combines CPX and BEQ for faster table lookup routines.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
CBEQ *opr*	DIR	31	dd rr	5
CBEQA *#opr, rel*	IMM	41	ii rr	4
CBEQX *#opr, rel*	IMM	51	ii rr	4
CBEQ X+, *rel*	IX+	71	rr	4
CBEQ *opr*, X+, *rel*	IX1+	61	ff rr	5
CBEQ *opr*, SP, *rel*	SP1	9E61	ff rr	5

Compare X with Immediate, Branch if Equal

Operation:

(X) – (M); PC ← (PC) + $0003 + *rel*
if result is $00

Description:

CBEQX compares an immediate operand with X (index register low) and causes a branch if the result is zero. The CBEQX instruction combines CMX and BEQ for faster loop counter control.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
CBEQ *opr*	DIR	31	dd rr	5
CBEQA *#opr, rel*	IMM	41	ii rr	4
CBEQX *#opr, rel*	IMM	51	ii rr	4
CBEQ X+, *rel*	IX+	71	rr	4
CBEQ *opr*, X+, *rel*	IX1+	61	ff rr	5
CBEQ *opr*, SP, *rel*	SP1	9E61	ff rr	5

CLRH

Clear H (Index Register High)

Operation:

H ← $00

Description:

The contents of H are replaced with zeros.

Condition Codes and Boolean Formulae:

V			H	I	N	Z	C
0	1	1	—	—	0	1	—

V: Cleared.

N: Cleared.

Z: Set.

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
CLRH	INH (H)	8C		1
CLR *opr*, SP	SP1	9E6F	ff	4

Compare Index Register with Memory

Operation:

(H:X) – (M:M + $0001)

Description:

CPHX compares index register (H:X) with the 16-bit value in memory and sets the condition code register accordingly.

Condition Codes and Boolean Formulae:

V			H	I	N	Z	C
↕	1	1	—	—	↕	↕	↕

V: $H7 \& \overline{M15} \& \overline{R15} \mid \overline{H7} \& M15 \& R15$
Set if a two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set is MSB of result is one; cleared otherwise.

Z: $\overline{R15} \& \overline{R14} \& \overline{R13} \& \overline{R12} \& \overline{R11} \& \overline{R10} \& \overline{R9} \& \overline{R8} \& \overline{R7} \& \overline{R6} \& \overline{R5} \& \overline{R4} \& \overline{R3} \& \overline{R2} \& \overline{R1}$
Set if the result is $0000; cleared otherwise.

C: $\overline{H7} \& M15 \mid M15 \& R15 \mid R15 \& \overline{H7}$
Set if the absolute value of the contents of memory is larger than the absolute value of the index register; cleared otherwise.

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
CPHX *#opr*	IMM	65	ii ii+1	3
CPHX *opr*	DIR	75	dd	4

Decimal Adjust Accumulator

Operation:

$(A)_{10}$

Description:

Adjusts contents of the accumulator (A) and the state of the CCR carry bit after binary-coded decimal (BCD) operations, so that there is a correct BCD sum and an accurate carry indication. The state of the CCR half carry bit affects operation. (Refer to the **DAA Function Summary** table on the following page for details of operation.)

Condition Codes and Boolean Formulae:

V			H	I	N	Z	C
U	1	1	—	—	↕	↕	↕

V: U
Undefined.

N: R7
Set is MSB of result is one; cleared otherwise.

Z: $\overline{R7}$ & $\overline{R6}$ & $\overline{R5}$ & $\overline{R4}$ & $\overline{R3}$ & $\overline{R2}$ & $\overline{R1}$ & $\overline{R0}$
Set if the result is $0000; cleared otherwise.

C: (Refer to the **DAA Function Summary** table on the following page.)

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
DAA	INH	72		2

Decimal Adjust Accumulator

The **DAA Function Summary** table below shows DAA operation for all legal combinations of input operands. Columns 1–4 represent the results of ADC or ADD operations on BCD operands. The correction factor in column 5 is added to the accumulator to restore the result of an operation on two BCD operands to a valid BCD value and to set or clear the C bit. All values are in hexadecimal.

DAA Function Summary

1	2	3	4	5	6
Initial C-Bit Value	Value of A[7:4]	Initial H-Bit Value	Value of A [3:0]	Correction Factor	Corrected C-Bit Value
0	0–9	0	0–9	00	0
0	0–8	0	A–F	06	0
0	0–9	1	0–3	06	0
0	A–F	0	0–9	60	1
0	9–F	0	A–F	66	1
0	A–F	1	0–3	66	1
1	0–2	0	0–9	60	1
1	0–2	0	A–F	66	1
1	0–3	1	0–3	66	1

DBNZ

Decrement and Branch if Not Zero

Operation:

A ← (A) – $0001
or: M ← (M) – $0001
or: X ← (X) – $0001; PC ← (PC) + $0003 + *rel*
if (result) 0 for DBNZ DIR or IX1
PC ← (PC) + $0002 + *rel*
if (result) 0 for DBNZA, DBNZX, or IX
PC ← (PC) + $0004 + *rel*
if (result) 0 for DBNZ SP1

Description:

Subtract one from the contents of A, X, or M; then branch using the relative offset if the result of the subtract is not zero.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
DBNZ *opr*	DIR	3B	dd rr	5
DBNZA *opr*	INH	4B	rr	3
DBNZX *opr*	INH	5B	rr	3
DBNZ *opr*	IX	7B	rr	4
DBNZ *opr*	IX1	6B	rr rr	5
DBNZ *opr*	SP1	9E6B	rr rr	6

Divide

Operation:

$A \leftarrow (H:A) \div (X)$ $H \leftarrow$ Remainder

Description:

Divides a 16-bit unsigned dividend contained in the concatenated registers H (index register high) and accumulator (A) by an 8-bit divisor contained in X (index register low). The quotient is placed in A, and the divisor is left unchanged.

An overflow (quotient > $FF) or divide-by-zero sets the C bit, and the quotient and remainder are indeterminate.

Condition Codes and Boolean Formulae:

V			H	I	N	Z	C
—	1	1	—	—	—	↕	↕

Z: $\overline{M7} \,\&\, \overline{M6} \,\&\, \overline{M5} \,\&\, \overline{M4} \,\&\, \overline{M3} \,\&\, \overline{M2} \,\&\, \overline{M1} \,\&\, \overline{M0}$
Set if the result (quotient) is $00; cleared otherwise.

C: Set if a divide by zero was attempted or if an overflow overflow occurred; cleared otherwise.

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
DIV	INH	52		7

LDHX

Load Index Register with Memory

Operation:

H:X ← (M:M + $0001)

Description:

Loads the contents of the specified memory location into index register (H:X). The condition codes are set according to the data.

Condition Codes and Boolean Formulae:

V			H	I	N	Z	C
0	1	1	—	—	↕	↕	—

V: 0
Cleared.

N: R15
Set if MSB of result is one; cleared otherwise.

Z: $\overline{R15}$ & $\overline{R14}$ & $\overline{R13}$ & $\overline{R12}$ & $\overline{R11}$ & $\overline{R10}$ & $\overline{R9}$ & $\overline{R8}$ & $\overline{R7}$ & $\overline{R6}$ & $\overline{R5}$ & $\overline{R4}$ & $\overline{R3}$ & $\overline{R2}$ & $\overline{R1}$ & $\overline{R0}$
Set if the result is $0000; cleared otherwise.

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
LDHX #*opr*	IMM	45	ii ii	3
LDHX *opr*	DIR	55	dd	4

Move

Operation:

$(M)_{destination} \leftarrow (M)_{source}$

Description:

Moves a byte of data from a source address to a destination address. Data is examined as it is moved, and condition codes are set. Source data is not changed. The accumulator is not affected.

There are four addressing modes for the MOV instruction:

1) IMD moves an immediate byte to a direct memory location.
2) DD moves a direct location byte to another direct location.
3) IX+D moves a byte from a location addressed by the index register (H:X) to a direct location. H:X is incremented after the move.
4) DIX+ moves a byte from a direct location to one addressed by H:X. H:X is incremented after the move.

Condition Codes and Boolean Formulae:

V			H	I	N	Z	C
0	1	1	—	—	↕	↕	—

V: 0
Cleared.

N: R7
Set if MSB of result is one; cleared otherwise.

Z: $\overline{R7}$ & $\overline{R6}$ & $\overline{R5}$ & $\overline{R4}$ & $\overline{R3}$ & $\overline{R2}$ & $\overline{R1}$ & $\overline{R0}$
Set if the result is $00; cleared otherwise.

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
MOV *opr*	IMD	6E	ii dd	4
MOV *opr*	DD	4E	dd dd	5
MOV *opr*	IX+D	7E	dd	4
MOV *opr*	DIX+	5E	dd	4

Nibble Swap Accumulator

Operation:

A ← (A [3:0] : A [7:4])

Description:

Swaps upper and lower nibbles (4 bits) of the accumulator. The NSA instruction is used for more efficient storage and use of binary-coded decimal operands.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
NSA	INH	62		3

Push Accumulator onto Stack

Operation:

↓ (A), SP ← (SP) – $0001

Description:

The contents of the accumulator (A) are pushed onto the stack at the address contained in the stack pointer. The stack pointer is then decremented to point to the next available location in the stack. The contents of the accumulator remain unchanged.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
PSHA	INH	87		2

PSHH

Push H (Index Register High) onto Stack

Operation:

↓ (H), SP ← (SP) – $0001

Description:

The contents of H are pushed onto the stack at the address contained in the stack pointer (SP). SP is then decremented to point at the next available location in the stack. The contents of H remain unchanged.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
PSHH	INH	8B		2

Push X (Index Register Low) onto Stack

Operation:

↓ (X), SP ← (SP) – $0001

Description:

The contents of X are pushed onto the stack at the address contained in the stack pointer (SP). SP is then decremented to point to the next available location in the stack. The contents of X remain unchanged.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
PSHX	INH	89		2

Pull Accumulator from Stack

Operation:

SP ← (SP + $0001); ↑ (A)

Description:

The stack pointer (SP) is incremented to address the last operand on the stack. The accumulator (A) is then loaded with the contents of the address pointed to by SP.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
PULA	INH	86		2

Pull H (Index Register High) from Stack

Operation:

SP ← (SP + $0001); ↑ (H)

Description:

The stack pointer (SP) is incremented to address the last operand on the stack. H is then loaded with the contents of the address pointed to by SP.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
PULH	INH	8A		2

PULX

Pull X (Index Register Low) from Stack

Operation:

SP ← (SP + $0001); ↑ (X)

Description:

The stack pointer (SP) is incremented to address the last operand on the stack. X is then loaded with the contents of the address pointed to by SP.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
PULX	INH	88		2

Store Index Register

Operation:

(M:M + $0001) ← (H:X)

Description:

Stores index register (H:X) to the specified memory location. The condition codes are set according to the data.

Condition Codes and Boolean Formulae:

V			H	I	N	Z	C
0	1	1	—	—	↕	↕	—

V: 0
Cleared.

N: R7
Set if MSB of result is one; cleared otherwise.

Z: $\overline{R15} \& \overline{R14} \& \overline{R13} \& \overline{R12} \& \overline{R11} \& \overline{R10} \& \overline{R9} \& \overline{R8} \& \overline{R7} \& \overline{R6} \& \overline{R5} \& \overline{R4} \& \overline{R3} \& \overline{R2} \& \overline{R1} \& \overline{R0}$
Set if the result is $0000; cleared otherwise.

Source Forms, Addressing Modes, Machine Code, and Cycles:

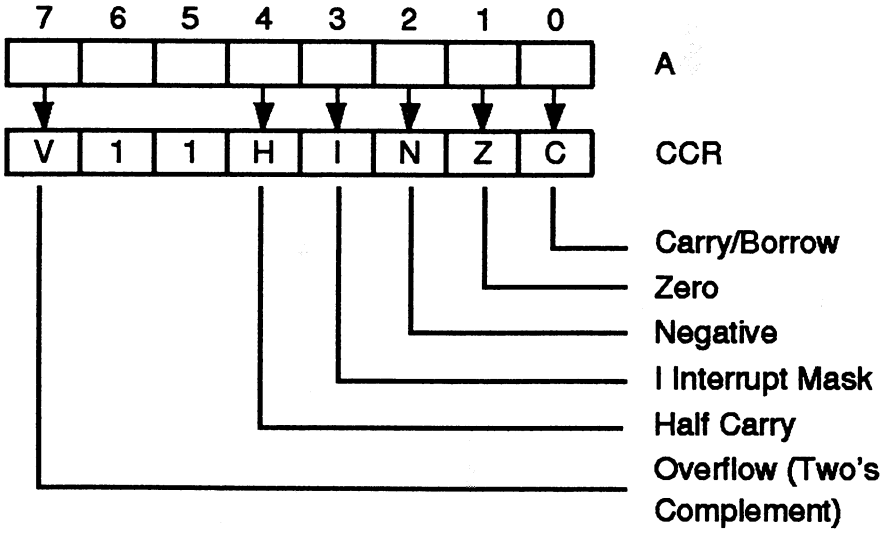
Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
STHX *opr*	DIR	35	dd	4

TAP

Transfer Accumulator to Condition Code Register

Operation:

CCR ← (A)



Description:

Transfers the contents of the accumulator (A) to the condition code register (CCR).

Condition Codes and Boolean Formulae:

V			H	I	N	Z	C
↕	1	1	↕	↕	↕	↕	↕

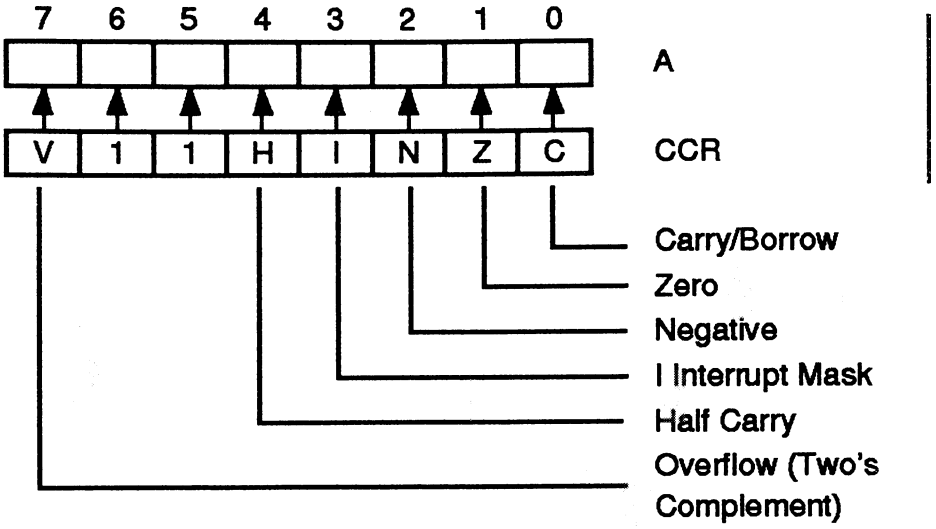
Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
TAP	INH	84		2

Transfer Condition Code Register to Accumulator

Operation:

A ← (CCR)



Description:

Transfers the contents of the condition code register (CCR) into the accumulator (A).

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
TPA	INH	85		1

Transfer Stack Pointer to Index Register

Operation:

H:X ← (SP + $0001)

Description:

Loads index register (H:X) with 1 plus the contents of the stack pointer (SP). The contents of SP remain unchanged. After a TSX instruction, H:X points to the last value that was stored on the stack.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
TSX	INH	95		2

Transfer Index Register to Stack Pointer

Operation:

SP ← (H:X – $0001)

Description:

Loads the stack pointer (SP) with the contents of the index register (H:X) minus one. The contents of H:X are not altered.

Condition Codes and Boolean Formulae:

None affected.

V			H	I	N	Z	C
—	1	1	—	—	—	—	—

Source Forms, Addressing Modes, Machine Code, and Cycles:

Source Forms	Addr Mode	Machine Code		HC08 Cycles
		Opcode	Operand(s)	
TXS	INH	94		2

ADDRESSING MODES

INHERENT (INH)

The inherent addressing mode has no operand because the opcode contains all information necessary to carry out the instruction. Most inherent instructions are one byte long.

IMMEDIATE (IMM)

The operand in immediate mode instructions is contained in the byte(s) immediately following the opcode. The immediate value is one or two bytes, depending on the size of the register involved in the instruction.

DIRECT (DIR)

Most direct mode instructions can access any of the first 256 memory addresses with two bytes. The first byte is the opcode, and the second byte is the low byte of the operand address. The high byte of the address is assumed to be $00.

EXTENDED (EXT)

Extended mode instructions are three bytes in length and can access any address in a 64-Kbyte memory map. The first byte is the opcode. The following two bytes are the operand address.

INDEXED (IX, IX1, IX2)

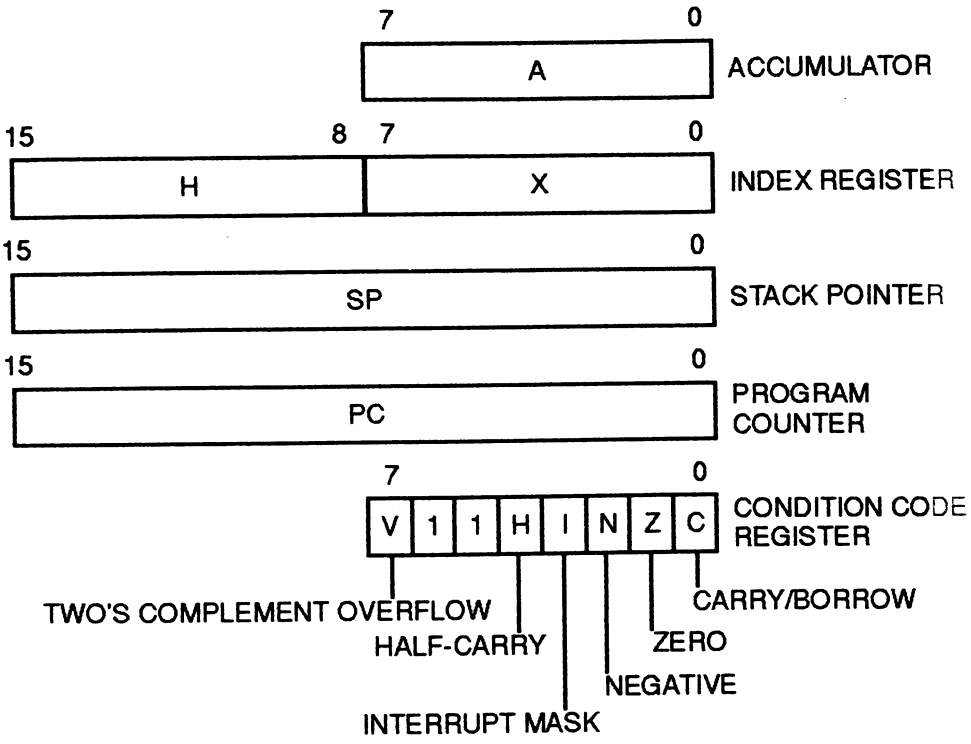
Indexed mode instructions access data with variable addresses. The effective address (EA) of the operand is determined by the contents of the register (H:X) added to a zero, 8-bit, or 16-bit offset. For one-byte, zero-offset mode instructions (IX), X (index register low) contains the low byte of the EA of the operand. The value of H (index register high) is $00 if none of the HC08 instructions that modify H are used, assuring source code compatibility with HC05 Family instructions. The sum of H:X is the EA of the operand. For two-byte, 8-bit offset mode instructions (IX1), the unsigned bytes in H:X added to the unsigned byte following the opcode constitutes the EA of the operand. For three-byte,16-bit offset mode instructions (IX2), the unsigned bytes in H:X added to the 16-bit unsigned word following the opcode constitute the EA of the operand.

ADDRESSING MODES

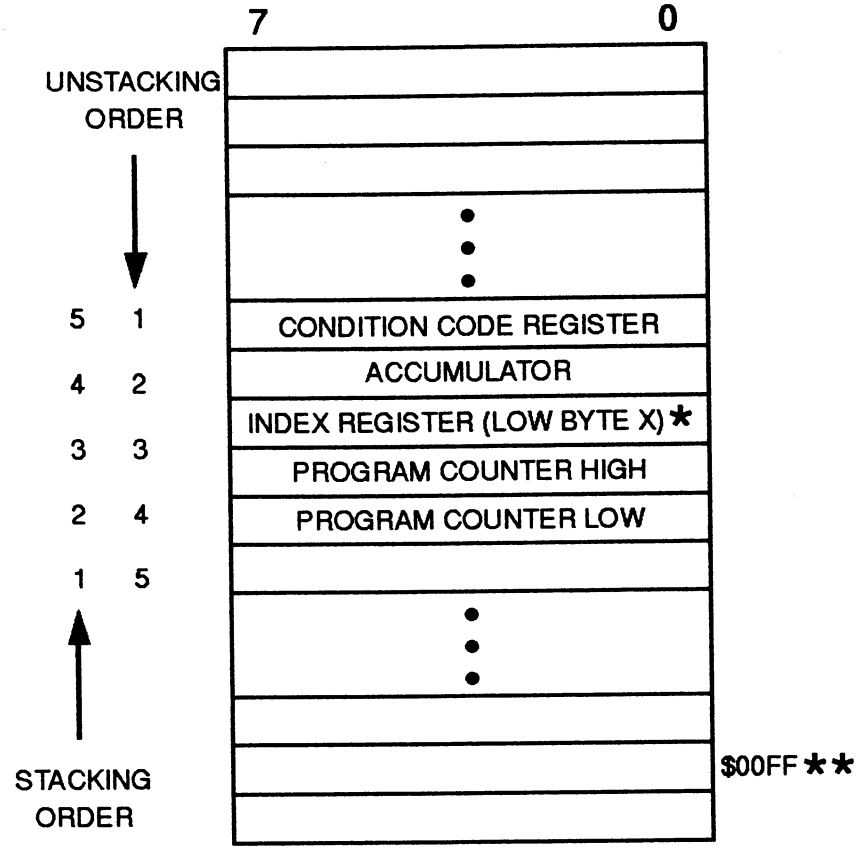
INDEXED, AND INDEXED, 8-BIT OFFSET WITH POST INCREMENT (IX+, IX1+)

Indexed, no offset with post increment mode instructions (IX+) are two-byte instructions that address operands, then increment H:X. The EA of the operand is derived by adding X (low byte) to H (high byte). Indexed, 8-bit offset with post increment mode instructions (IX1+) are three-byte instructions that address operands with variable addresses, then increment H:X. The EA of the operand is derived by adding X (low byte) with H (high byte).

PROGRAMMING MODEL



INTERRUPT STACKING ORDER



* High byte (H) of index register is not stacked.

** Default address on reset.

OPCODE MAP

	Bit Manipulation		Branch	Read-Modify-Write					
	DIR	DIR	REL	DIR	INH	INH	IX1	SP1	IX
HIGH / LOW	0	1	2	3	4	5	6	9E6	7
0	5 BRSET0 3 DIR	4 BSET0 2 DIR	3 BRA 2 REL	4 NEG 2 DIR	1 NEGA 1 INH	1 NEGX 1 INH	4 NEG 2 IX1	5 NEG 3 SP1	3 NEG 1 IX
1	5 BRCLR0 3 DIR	4 BCLR0 2 DIR	3 BRN 2 REL	5 CBEQ 3 DIR	4 CBEQA 3 IMM	4 CBEQX 3 IMM	5 CBEQ 3 IX1+	6 CBEQ 4 SP1	4 CBEQ 2 IX+
2	5 BRSET1 3 DIR	4 BSET1 2 DIR	3 BHI 2 REL		5 MUL 1 INH	7 DIV 1 INH	3 NSA 1 INH		2 DAA 1 INH
3	5 BRCLR1 3 DIR	4 BCLR1 2 DIR	3 BLS 2 REL	4 COM 2 DIR	1 COMA 1 INH	1 COMX 1 INH	4 COM 2 IX1	5 COM 3 SP1	3 COM 1 IX
4	5 BRSET2 3 DIR	4 BSET2 2 DIR	3 BCC 2 REL	4 LSR 2 DIR	1 LSRA 1 INH	1 LSRX 1 INH	4 LSR 2 IX1	5 LSR 3 SP1	3 LSR 1 IX
5	5 BRCLR2 3 DIR	4 BCLR2 2 DIR	3 BCS 2 REL	4 STHX 2 DIR	3 LDHX 3 IMM	4 LDHX 2 DIR	3 CPHX 3 IMM		4 CPHX 2 DIR
6	5 BRSET3 3 DIR	4 BSET3 2 DIR	3 BNE 2 REL	4 ROR 2 DIR	1 RORA 1 INH	1 RORX 1 INH	4 ROR 2 IX1	5 ROR 3 SP1	3 ROR 1 IX

OPCODE MAP

7	BRCLR3 5 3 DIR	BCLR3 4 2 DIR	BEQ 3 2 REL	ASR 4 2 DIR	ASRA 1 1 INH	ASRX 1 1 INH	ASR 4 2 IX1	ASR 5 3 SP1	ASR 3 1 IX
8	BRSET4 5 3 DIR	BSET4 4 2 DIR	BHCC 3 2 REL	LSL 4 2 DIR	LSLA 1 1 INH	LSLX 1 1 INH	LSL 4 2 IX1	LSL 5 3 SP1	LSL 3 1 IX
9	BRCLR4 5 3 DIR	BCLR4 4 2 DIR	BHCS 3 2 REL	ROL 4 2 DIR	ROLA 1 1 INH	ROLX 1 1 INH	ROL 4 2 IX1	ROL 5 3 SP1	ROL 3 1 IX
A	BRSET5 5 3 DIR	BSET5 4 2 DIR	BPL 3 2 REL	DEC 4 2 DIR	DECA 1 1 INH	DECX 1 1 INH	DEC 4 2 IX1	DEC 5 3 SP1	DEC 3 1 IX
B	BRCLR5 5 3 DIR	BCLR5 4 2 DIR	BMI 3 2 REL	DBNZ 5 3 DIR	DBNZA 3 2 INH	DBNZX 3 2 INH	DBNZ 5 3 IX1	DBNZ 6 4 SP1	DBNZ 4 2 IX
C	BRSET6 5 3 DIR	BSET6 4 2 DIR	BMC 3 2 REL	INC 4 2 DIR	INCA 1 1 INH	INCX 1 1 INH	INC 4 2 IX1	INC 5 3 SP1	INC 3 1 IX
D	BRCLR6 5 3 DIR	BCLR6 4 2 DIR	BMS 3 2 REL	TST 3 2 DIR	TSTA 1 1 INH	TSTX 1 1 INH	TST 3 2 IX1	TST 4 3 SP1	TST 2 1 IX
E	BRSET7 5 3 DIR	BSET7 4 2 DIR	BIL 3 2 REL		MOV 5 3 DD	MOV 4 2 DIX+	MOV 4 3 IMD		MOV 4 2 IX+D
F	BRCLR7 5 3 DIR	BCLR7 4 2 DIR	BIH 3 2 REL	CLR 3 2 DIR	CLRA 1 1 INH	CLRX 1 1 INH	CLR 3 2 IX1	CLR 4 3 SP1	CLR 2 1 IX

OPCODE MAP

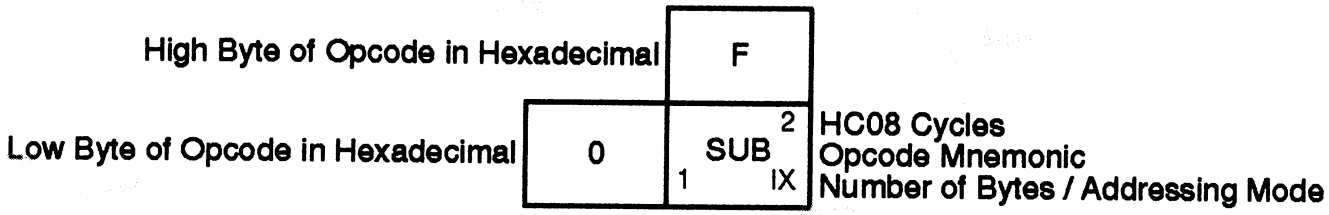
	Control		Register/Memory							
	INH	INH	IMM	DIR	EXT	IX2	SP2	IX1	SP1	IX
HIGH / LOW	8	9	A	B	C	D	9ED	E	9EE	F
0	RTI 7 1 INH	BGE 3 2 REL	SUB 2 2 IMM	SUB 3 2 DIR	SUB 4 3 EXT	SUB 4 3 IX2	SUB 5 4 SP2	SUB 3 2 IX1	SUB 4 3 SP1	SUB 2 1 IX
1	RTS 4 1 INH	BLT 3 2 REL	CMP 2 2 IMM	CMP 3 2 DIR	CMP 4 3 EXT	CMP 4 3 IX2	CMP 5 4 SP2	CMP 3 2 IX1	CMP 4 3 SP1	CMP 2 1 IX
2		BGT 3 2 REL	SBC 2 2 IMM	SBC 3 2 DIR	SBC 4 3 EXT	SBC 4 3 IX2	SBC 5 4 SP2	SBC 3 2 IX1	SBC 4 3 SP1	SBC 2 1 IX
3	SWI 9 1 INH	BLE 3 2 REL	CPX 2 2 IMM	CPX 3 2 DIR	CPX 4 3 EXT	CPX 4 3 IX2	CPX 5 4 SP2	CPX 3 2 IX1	CPX 4 3 SP1	CPX 2 1 IX
4	TAP 2 1 INH	TXS 2 1 INH	AND 2 2 IMM	AND 3 2 DIR	AND 4 3 EXT	AND 4 3 IX2	AND 5 4 SP2	AND 3 2 IX1	AND 4 3 SP1	AND 2 1 IX
5	TPA 1 1 INH	TSX 2 1 INH	BIT 2 2 IMM	BIT 3 2 DIR	BIT 4 3 EXT	BIT 4 3 IX2	BIT 5 4 SP2	BIT 3 2 IX1	BIT 4 3 SP1	BIT 2 1 IX
6	PULA 2 1 INH		LDA 2 2 IMM	LDA 3 2 DIR	LDA 4 3 EXT	LDA 4 3 IX2	LDA 5 4 SP2	LDA 3 2 IX1	LDA 4 3 SP1	LDA 2 1 IX

OPCODE MAP

7	PSHA 2 1 INH	TAX 1 1 INH	AIS 2 2 IMM	STA 3 2 DIR	STA 4 3 EXT	STA 4 3 IX2	STA 5 4 SP2	STA 3 2 IX1	STA 4 3 SP1	STA 2 1 IX
8	PULX 2 1 INH	CLC 1 1 INH	EOR 2 2 IMM	EOR 3 2 DIR	EOR 4 3 EXT	EOR 4 3 IX2	EOR 5 4 SP2	EOR 3 2 IX1	EOR 4 3 SP1	EOR 2 1 IX
9	PSHX 2 1 INH	SEC 1 1 INH	ADC 2 2 IMM	ADC 3 2 DIR	ADC 4 3 EXT	ADC 4 3 IX2	ADC 5 4 SP2	ADC 3 2 IX1	ADC 4 3 SP1	ADC 2 1 IX
A	PULH 2 1 INH	CLI 2 1 INH	ORA 2 2 IMM	ORA 3 2 DIR	ORA 4 3 EXT	ORA 4 3 IX2	ORA 5 4 SP2	ORA 3 2 IX1	ORA 4 3 SP1	ORA 2 1 IX
B	PSHH 2 1 INH	SEI 2 1 INH	ADD 2 2 IMM	ADD 3 2 DIR	ADD 4 3 EXT	ADD 4 3 IX2	ADD 5 4 SP2	ADD 3 2 IX1	ADD 4 3 SP1	ADD 2 1 IX
C	CLRH 1 1 INH	RSP 1 1 INH		JMP 2 2 DIR	JMP 3 3 EXT	JMP 4 3 IX2		JMP 3 2 IX1		JMP 2 1 IX
D		NOP 1 1 INH	BSR 4 2 REL	JSR 4 2 DIR	JSR 5 3 EXT	JSR 6 3 IX2		JSR 5 2 IX1		JSR 4 1 IX
E	STOP 1 1 INH	*	LDX 2 2 IMM	LDX 3 2 DIR	LDX 4 3 EXT	LDX 4 3 IX2	LDX 5 4 SP2	LDX 3 2 IX1	LDX 4 3 SP1	LDX 2 1 IX
F	WAIT 1 1 INH	TXA 1 1 INH	AIX 2 2 IMM	STX 3 2 DIR	STX 4 3 EXT	STX 4 3 IX2	STX 5 4 SP2	STX 3 2 IX1	STX 4 3 SP1	STX 2 1 IX

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 Post Increment
EXT	Extended	IX2	Indexed, 16-Bit Offset		
DD	Direct-Direct	IMD	Immediate-Direct	IX1+	Indexed, 1-Byte Offset with Post Increment
IX+D	Indexed-Direct	DIX+	Direct-Indexed		

*Prebyte for stack pointer indexed instructions



ASCII CHART

ASCII CHARACTER SET (7-BIT CODE)								
MS DIGIT / LS DIGIT	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	—	o	DEL

HEX/DEC CONVERSION

How to use:

Conversion to Decimal: Find the decimal weights for corresponding hexadecimal characters beginning with the least significant character. The sum of the decimal weights is the decimal value of the hexadecimal number.

Conversion to Hexadecimal: Find the highest decimal value in the table which is lower than or equal to the decimal number to be converted. The corresponding hexadecimal character is the most significant. Subtract the decimal value found from the decimal number to be converted. With the difference, repeat the process to find subsequent hexadecimal characters.

15	Byte		8	7			0
15	Char12		11	8	Char4	3	Char0
Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,228	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

M68HC08 INSTRUCTION SET

NEW HC08 INSTRUCTIONS

ADDRESSING MODES

PROGRAMMING MODEL
INTERRUPT STACKING ORDER

OPCODE MAP

ASCII CHART
HEX/DEC CONVERSION

M68HC08 INSTRUCTION SET

NEW HC08 INSTRUCTIONS

ADDRESSING MODES

PROGRAMMING MODEL
INTERRUPT STACKING ORDER

OPCODE MAP

ASCII CHART
HEX/DEC CONVERSION

Literature Distribution Centers:

USA: Motorola Literature Distribution
P.O. Box 20912; Phoenix, Arizona 85036.

EUROPE: Motorola Ltd.; European Literature Centre
88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.

JAPAN: Nippon Motorola Ltd.
4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.

ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.
Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong.

MOTOROLA

M68HC08RG/AD

1ATX31333–1 PRINTED IN USA 9/94 IMPERIAL LITHO 99573 14,000 MCU YGACAA