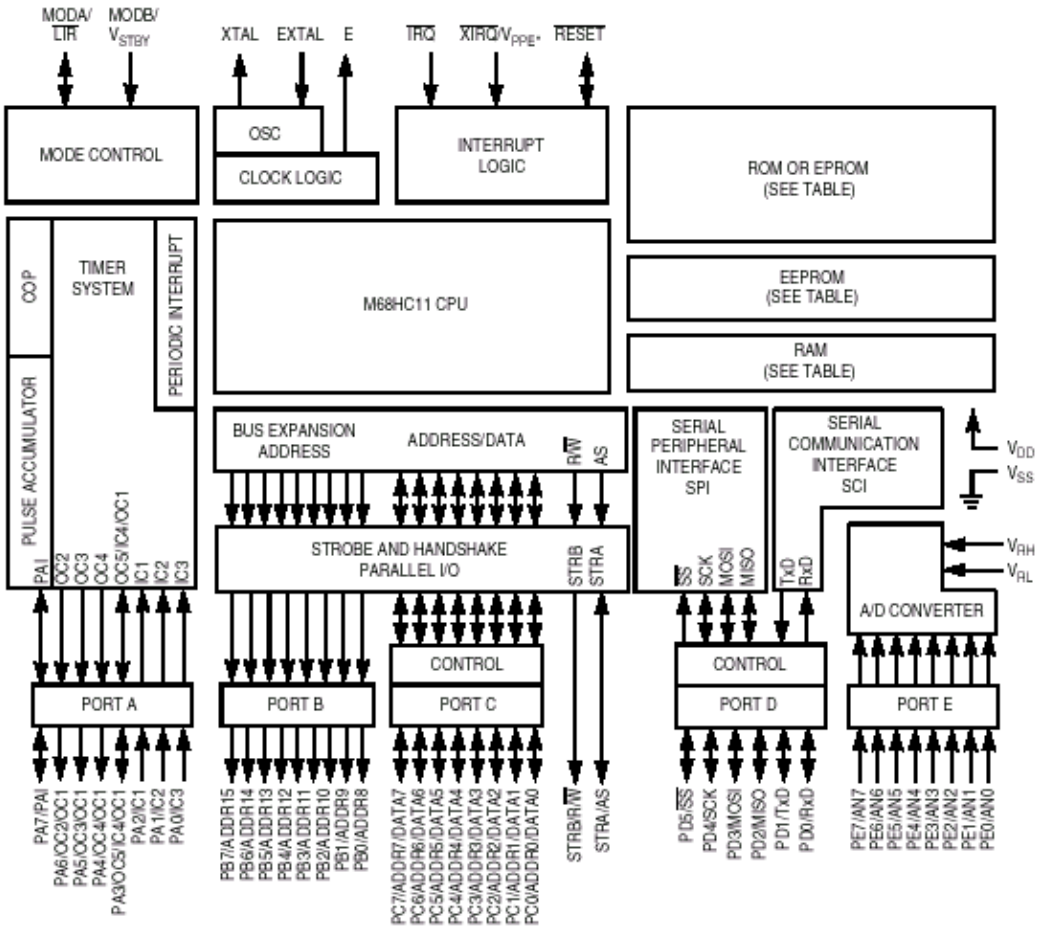
CARACTERÍSTICAS HARDWARE Motorola 68HC11

Familia 68HC11

DISPOSITIVO	RAM	ROM	EPROM	EEPROM
MC68HC11A0	256	0	0	0
MC68HC11A1	256	0	0	512
MC68HC11A7	256	8K	0	0
MC68HC11A8	256	8K	0	512
MC69HC11D0	192	0	0	0
MC68HC11D3	192	4K	0	0
MC68HC711D3	192	0	4K	0
MC68HC11ED0	512	0	0	0
MC68HC11E0	512	0	0	0
MC68HC11E1	512	0	0	512
MC68HC11E8	512	12K	0	0
MC68HC11E9	512	12K	0	512
MC68HC711E9	512	0	12K	512
MC68HC811E2	256	0	0	2K
MC68HC11E20	768	20K	0	512
MC68HC711E20	768	0	20K	512
MC68HC11F1	1K	0	0	512



LOS REGISTROS DEL 68HC11

La CPU del microcontrolador 68HC11 dispone de un registro D de 16 bits. La parte alta del registro D es el acumulador A de 8 bits, y la parte baja es el acumulador B de 8 bits:

Registro D (16 bits)

Acumulador A (8 bits)	Acumulador B (8 bits)

Dispone además de 2 registros de 16 bits para direccionamiento indexado denominados X e Y:

Registro índice X (16 bits)
Registro índice Y (16 bits)

El puntero de pila (SP) y el contador de programa (PC) son también de 16 bits lo que limita la longitud de un programa a 64KBytes, espacio máximo direccionable por el 68HC11:

Puntero de pila SP (16 bits)
Contador de programa PC (16 bits)

El puntero de pila (SP) debe ser inicializado por el usuario. Esta crece desde direcciones altas hacia direcciones bajas, por lo que al añadir un elemento en la pila, el puntero SP se decrementa en 1 ó 2 bytes dependiendo del tamaño del dato que se introduce en la pila. Del mismo modo, al sacar un elemento de la pila, SP se incrementa.

El contador de programa (PC) se incrementa para apuntar a la instrucción que se va a ejecutar. Es por esto que los programas se ejecutan desde direcciones bajas hacia direcciones altas al contrario que el puntero de pila, por lo que tendremos que impedir que la pila se solape con el código (si este se encuentra en RAM y no en la EEPROM). Tendremos que dar pues al puntero SP un valor seguro.

El registro CCR de 8 bits, llamado registro de estado, contiene unos bits que describen el estado de la CPU:

CCR (8 bits)

S	X	H	I	N	Z	V	C

- S: stop deshabilitado
- X: máscara de interrupción especial
- H: medio acarreo
- I: máscara de interrupción normal
- N: negativo
- Z: cero
- V: overflow
- C: acarreo

MODOS DE DIRECCIONAMIENTO DEL 68HC11

Existen 6 maneras de acceder (llamadas modos de direccionamiento) a los datos que están en memoria:

1. Direccionamiento inmediato
2. Direccionamiento extendido
3. Direccionamiento directo
4. Direccionamiento indexado
5. Direccionamiento relativo
6. Direccionamiento inherente

DIRECCIONAMIENTO INMEDIATO:

La instrucción contiene el dato al que hacemos referencia por lo cual no tenemos que acceder a la memoria. Este dato puede ser de 1 ó 2 bytes y debe ser precedido por el símbolo # (almohadilla, sostenido).

Ejemplo:

- LDAA #01

  Carga en el acumulador A el valor 01

DIRECCIONAMIENTO EXTENDIDO:

En este caso tenemos que buscar el dato en una dirección de memoria que se obtiene de la instrucción. La dirección de memoria ocupa 2 bytes y puede tomar valores entre $0000 (0 en decimal) y $FFFF (65535 en decimal).

Ejemplo:

- LDAA $FA0E

  Carga en el acumulador A el valor almacenado en la posición de memoria $FA0E (el contenido de la dirección $FA0E).

DIRECCIONAMIENTO DIRECTO:

Igual que el modo anterior (direccionamiento extendido) salvo que utiliza direcciones comprendidas entre $00 y $FF (256 bytes de memoria). Este modo tiene como ventaja que sólo necesita 1 byte para determinar la dirección del dato.

Ejemplo:

- LDAA $0E

  Carga en el acumulador A el valor almacenado en la posición de memoria $0E (el contenido de la dirección $0E).

DIRECCIONAMIENTO INDEXADO:

Este modo de direccionamiento es utilizado para acceder a tablas en la memoria. Para obtener la dirección de memoria donde está almacenado el dato se toma la dirección del registro índice (X o Y) y se le suma un desplazamiento de 8 bits.

Ejemplo:

- LDAB 5,X

  Carga en el acumulador B el valor almacenado en la posición de memoria especificada por X más el desplazamiento 5.

DIRECCIONAMIENTO RELATIVO:

Este modo de direccionamiento se utiliza sólo para las instrucciones de bifurcación. Sirve para indicar a la CPU que efectúe un salto de cierto número de bytes hacia delante o hacia detrás. Este desplazamiento tiene signo y es de un byte por lo que las bifurcaciones sólo se pueden realizar de 128 bytes hacia detrás ó 127 bytes hacia delante.

No tienes que especificar explícitamente la dirección de salto pues de esto se encarga automáticamente el ensamblador. Sin embargo has de tener en cuenta que sólo se pueden realizar saltos con instrucciones BR (BRanch) hacia direcciones de memoria que estén a menos de 128 bytes por debajo y a menos de 127 bytes por arriba. En caso de no respetar este rango el ensamblador deberá dar un mensaje de error.

Ejemplo:

- *correcto*:

  bucle ... ;dirección 0
  ...
  ...
  BE bucle ;dirección 10 (bytes)

- *incorrecto*:

  bucle ... ;dirección 0
  ...
  ...
  BE bucle ;dirección 255 (bytes)

DIRECCIONAMIENTO INHERENTE:

Los operando no se encuentran en memoria sino que son registros. Por el código de la intrucción la CPU sabe de qué registros se trata.

Ejemplo:

- INCA

  Incrementa el valor del acumulador A en una unidad.

- INX

  Incrementa el valor del registro índice X en una unidad.

CONJUNTO DE INSTRUCCIONES

INSTRUCCIONES DE CARGA, ALMACENAMIENTO Y TRANSFERENCIA:

- **Instrucciones de carga**: permiten introducir un nuevo valor en los registros, leer una posición de memoria...

  LDAA, LDAB, LDD, LDX, LDY, LDS, CLRA, CLRB

  - **LDAA**: Introduce un dato de 8 bits en el acumulador A

    Ejemplos:

    LDAA #30 ; A:=30 (direccionamiento inmediato)

    LDAA $1000 ; mete en el acumulador A el contenido de la dirección $1000.

  - **LDAB**: Introduce un dato de 8 bits en el acumulador B

    Ejemplos:

    LDAB #30 ; B:=30 (direccionamiento inmediato)

    LDAB $1000 ; mete en el acumulador B el contenido de la dirección $1000.

  - **LDD**: Introduce un dato de 8 ó 16 bits en el doble acumulador D
  - **Ejemplos:**

    LDD #$FFCC ; D:=$FFCC , A:=$FF, B:=$CC

    LDD #$20 ; D:=$0020 , A:=$00, B:=$20

  - **LDX**: Introduce un dato de 16 bits en el registro índice X

    Ejemplos:

    LDX #$3000 ; X:=$3000

    LDX 4,Y ; mete en el registro índice X el contenido de la dirección Y+4

    LDX 4,X ; mete en el registro índice X el contenido de la dirección X+4

  - **LDY**: Introduce un dato de 16 bits en el registro índice Y

    Ejemplos:

    LDY #$3000 ; X:=$3000

    LDY 4,X ; mete en el registro índice Y el contenido de la dirección X+4

    LDY 4,Y ; mete en el registro índice Y el contenido de la dirección Y+4

  - **LDS**: Introduce un dato de 16 bits en el registro SP (puntero de pila).

Ejemplo:

LDS #$FC00 ; inicializa la pila para que vaya de la dirección $FC00 hacia abajo

- **CLRA**: Borra el contenido del acumulador A. Equivale a LDAA #0 con la diferencia que el direccionamiento es inherente y sólo ocupa 1 byte mientras que LDAA #0 ocupa 2 bytes.

  Ejemplo:

  CLRA ; A:=0

- **CLRB**: Borra el contenido del acumulador B. Equivale a LDAB #0 con la diferencia que el direccionamiento es inherente y sólo ocupa 1 byte mientras que LDAB #0 ocupa 2 bytes.

  Ejemplo:

  CLRB ; B:=0

- **Instrucciones de almacenamiento**: permiten alterar una posición de memoria, un puerto...

  STAA, STAD, STD, STX, STY, STS, CLR

  - **STAA**: Almacena el acumulador A en una dirección de memoria.

    Ejemplos:

    STAA $1000 ; almaceno el acumulador A en la dirección de memoria $1000

  - **STAB**: Almacena el acumulador B en una dirección de memoria.

    Ejemplos:

    STAB $1000 ; almaceno el acumulador B en la dirección de memoria $1000

  - **STD**: Almacena el doble acumulador D (16 bits) en una dirección de memoria.

    Ejemplos:

    STD $1000 ; almaceno el doble acumulador D en la dirección de memoria $1000

  - **STX**: Almacena el registro X (16 bits) en una dirección de memoria.

    Ejemplos:

    STX $1000 ; almaceno el registro X en la dirección de memoria $1000

  - **STY**: Almacena el registro Y (16 bits) en una dirección de memoria.

    Ejemplos:

    STY $1000 ; almaceno el registro Y en la dirección de memoria $1000

  - **STS**: Almacena el puntero de pila (SP) en una dirección de memoria.

Ejemplos:

STS $1000 ; almaceno SP en la dirección de memoria $1000

- **CLR**: Pone a cero el contenido de una dirección de memoria.

Ejemplos:

CLR $1000 ; pone a ceros la dirección de memoria $1000

- **Instrucciones de transferencia**: permiten transferir datos entre registros y registros o entre registros y memoria. El direccionamiento es inherente por lo que no es necesario especificar dirección.

PSHA, PSHB, PSHX, PSHY, PULA, PULB, PULX, PULY, TAB, TBA, TSX, TSY, TXS, TYS, XGDX,XGDY

- **PSHA**: Introduce el acumulador A en la pila. Se introduce A en la dirección especificada por SP. SP se decrementa en 1.
- **PSHB:** Introduce el acumulador B en la pila.
- **PSHX:** Introduce el registro de índice X en la pila. Se introduce X en la pila. SP se decrementa en 2 unidades puesto que X es de 16 bits.
- **PSHY:** Introduce el registro de índice Y en la pila.
- **PULA:** Saca A de la pila.
- **PULB:** Saca B de la pila.
- **PULX:** Saca X de la pila. SP se incrementa en 2 unidades ya que X es de 16 bits.
- **PULY:** Saca Y de la pila. SP se incrementa en 2 unidades ya que Y es de 16 bits.
- **TAB**: Introduce el valor de A en B.
- **TBA**: Introduce el valor de B en A.
- **TSX:** Introduce el valor de SP en X.
- **TSY:** Introduce el valor de SP en Y.
- **TXS:** Introduce el valor de X en SP.
- **TYS:** Introduce el valor de Y en SP.
- **XGDX:** Intercambia el registro D con el X.
- **XGDY**: Intercambia el registro D con el Y.

Instrucciones aritméticas: suma, resta, comparaciones, complemento a dos, multiplicaciones y divisiones:

- **Suma**:

ADDA, ADDB, ADDD, ADCA, ADCB, ABA, ABX, ABY, INCA, INCB, INC, INX, INY, INAS

- **ADDA**: Añade un dato al acumulador A.

  Ejemplos:

  ADDA #5 ; suma 5 al acumulador A

  ADDA $C000 ; suma el contenido de la dirección de memoria $C000 al acumulador A

- **ADDB**: Añade un dato al acumulador B.

  Ejemplos:

  ADDB #6 ; suma 6 al acumulador B

  ADDB $D0C0 ; suma el contenido de la dirección de memoria $D0C0 al acumulador B

- **ADDD**: Añade un dato al acumulador D (16 bits).

  Ejemplos:

  ADDD #$FFFF ; suma $FFFF al acumulador D

  ADDD $C000 ; suma el contenido (dato de 16 bits) de la dirección de memoria $C000 al acumulador D

- **ADCA**: Añade al acumulador A un dato y el contenido del acarreo.
- **ADCB:** Añade al acumulador B un dato y el contenido del acarreo.
- **ABA:** Suma el contenido de B con el de A y almacena el resultado en A
- **ABX:** Suma el contenido de B con el de X y almacena el resultado en X
- **ABY:** Suma el contenido de B con el de Y y almacena el resultado en Y
- **INCA:** Incrementa el contenido del acumulador A en una unidad
- **INCB:** Incrementa el contenido del acumulador B en una unidad
- **INC:** Incrementa el contenido de una dirección de memoria
- **INX:** Incrementa el contenido del registro índice X en una unidad
- **INY:** Incrementa el contenido del registro índice Y en una unidad
- **INS:** Incrementa el puntero de pila SP

LISTADO DE LOS NEMÓNICOS DEL 68HC11

ABA	añade el contenido del acumulador B al acumulador A.
ABX	añade el contenido del acumulador B (sin signo) al contenido del registro X.
ABY	añade el contenido del acumulador B (sin signo) al contenido del registro Y.
ADCA	añade al acumulador A un dato y el bit de acarreo.
ADCB	añade al acumulador B un dato y el bit de acarreo.
ADDA	añade un dato al registro A.
ADDB	añade un dato al registro B.
ADDD	añade un dato de 16 bits al registro.
ANDA	realiza una operación lógica AND entre un dato y el acumulador A. Resultado en A.
ANDB	realiza una operación lógica AND entre un dato y el acumulador B. Resultado en B.
ASLA	desplaza un bit a la izquierda el acumulador A.
ASLB	desplaza un bit a la izquierda el acumulador B.
ALSD	desplaza un bit a la izquierda el acumulador D.
ASRA	desplaza un bit a la derecha el acumulador A.
ASRB	desplaza un bit a la derecha el acumulador B.
BCC	salta si no hay acarreo.
BCLR	pone a cero bits de la memoria.
BCS	salta si hay acarreo.
BEQ	salta si igual.
BGE	salta si mayor que o igual a cero.
BGT	salta si mayor que cero.
BHI	salta si es mayor.
BHS	salta si mayor o igual.
BITA	comprobar el bit específico del acumulador A.
BITB	comprobar el bit específico del acumulador B.
BLE	salta si es menor que o igual a cero.
BLO	salta si es menor (lo mismo que BCS).
BLS	salta si es menor o igual.
BLT	salta si es menor que cero.
BMI	salta si es negativo
BNE	salta si no es igual
BPL	salto si es positivo
BRA	salto incondicional
BRCLR	salta si los bits especificados estan a cero
BRN	no saltar nunca (es equivalente a una operación NOP de 2 bytes)
BRSET	salta si los bits especificados estan a uno
BSET	pone los bits especificados a uno
BSR	salta a una subrutina
BVC	salta si no ha habido desbordamiento (overflow)
BVS	salta si ha habido overflow
CBA	compara el acumulador A con el B
CLC	pone a cero el bit de acarreo
CLI	habilita las interrupciones
CLR	pone a cero el contenido de memoria especificado
CLRA	pone a cero el acumulador A
CLRB	pone a cero el acumulador B
CLV	pone a cero el bit de overflow
CMPA	compara el acumulador A con un dato
CMPB	compara el acumulador B con un dato
COMA	complementa a uno el acumulador A
COMB	complementa a uno el acumulador B

COM	complementa a uno el contenido de la memoria especificado
CPD	compara el registro D con un dato
CPX	compara el registro X con un dato
CPY	compara el registro Y con un dato
DAA	ajuste decimal
DEC	decrementa una posición de memoria especificada
DECA	decrementa el acumulador A
DECB	decrementa el acumulador B
DES	decrementa el puntero de pila SP
DEX	decrementa el registro X
DEY	decrementa el registro Y
EORA	operación XOR entre un dato y el acumulador A
EORB	operación XOR entre un dato y el acumulador B
FDIV	división
IDIV	división entera
INC	incrementa el contenido de una posición de memoria
INCA	incrementa el acumulador A
INCB	incrementa el acumulador B
INS	incrementa el puntero de pila SP
INX	incrementa el registro X
INY	incrementa el registro Y
JMP	salto incondicional
JSR	salto a una subrutina
LDAA	carga un dato en el acumulador A
LDAB	carga un dato en el acumulador B
LDD	carga un dato de 16 bits en el registro D
LDS	carga un dato de 16 bits en el puntero de pila SP
LDX	carga un dato de 16 bits en el registro X
LDY	carga un dato de 16 bits en el registro Y
LSL	desplaza un bit hacia la izquierda el contenido de una posición de memoria
LSLA	desplaza un bit hacia la izquierda el acumulador A
LSLB	desplaza un bit hacia la izquierda el acumulador B
LSLD	desplaza un bit hacia la izquierda el acumulador D
LSR	desplaza un bit hacia la derecha el contenido de una posición de memoria
LSRA	desplaza un bit hacia la derecha el acumulador A
LSRB	desplaza un bit hacia la derecha el acumulador B
LSRD	desplaza un bit hacia la derecha el acumulador D
MUL	multiplicación sin signo
NEG	complementa a dos el contenido de una posición de memoria
NEGA	complementa a dos el contenido del acumulador A
NEGB	complementa a dos el contenido del acumulador B
NOP	no opera
ORAA	realiza la operación lógica OR entre un dato y el acumulador A
ORAB	realiza la operación lógica OR entre un dato y el acumulador B
PSHA	mete el acumulador A en la pila
PSHB	mete el acumulador B en la pila
PSHX	mete el registro X en la pila
PSHY	mete el registro Y en la pila
PULA	saca el acumulador A de la pila
PULB	saca el acumulador B de la pila
PULX	saca el registro X de la pila
PULY	saca el registro Y de la pila
ROL	rota a la izquierda el contenido de una posición de memoria

ROLA	rota a la izquierda el acumulador A
ROLB	rota a la izquierda el acumulador B
ROR	rota a la derecha el contenido de una posición de memoria
RORA	rota a la derecha el acumulador A
RORB	rota a la derecha el acumulador B
RTI	retorno de interrupción
RTS	retorno de subrutina
SBA	resta un dato al acumulador A
SBCA	resta un dato y el bit de acarreo al acumulador A
SBCB	resta un dato y el bit de acarreo al acumulador B
SEC	pone a uno el bit de acarreo
SEI	deshabilita todas las interrupciones
SEV	pone a uno el bit de overflow
STAA	almacena el acumulador A en una posición de memoria
STAB	almacena el acumulador B en una posición de memoria
STD	almacena el registro D
STOP	para el reloj del sistema
STS	almacena el puntero de pila SP
STX	almacena el registro X
STY	almacena el registro Y
SUBA	resta un dato al acumulador A
SUBB	resta un dato al acumulador B
SUBD	resta un dato al registro D
SWI	provoca una interrupción software
TAB	transfiere el acumulador A al acumulador B
TAP	transfiere el acumulador A al registro CCR
TAB	transfiere el acumulador B al acumulador A
TEST	instrucción de test (sólo se puede ejecutar en el modo de test)
TPA	transfiere el registro CCR al acumulador A
TST	comprueba si una posición de memoria está a cero
TSTA	comprueba si el acumulador A está a cero
TSTB	comprueba si el acumulador B está a cero
TSX	transfiere el puntero de pila al registro X
TSY	transfiere el puntero de pila al registro Y
TXS	transfiere el registro X al puntero de pila
TYS	transfiere el registro Y al puntero de pila
WAI	espera a que se produzca una interrupción
XHDX	intercambia los valores de los registros D y X
XGDY	intercambia los valores de los registro D e Y

68HC11 PROGRAMMING CARD

Programming Model

7	Accum A	0	7	Accum B	0
15	D (A & B combined)				0
15	Index register X				0
15	Index register Y				0
15	Stack Pointer SP				0
15	Program counter PC				0

Addressing modes

direct : adr8 — extended : adr16
immediate: data8/data16 — indexed : off,i
relative : rel

16 bit information is stored with msbyte in low address & lsbyte in high address

Abbreviations:

a = accumulators A or B
ccr = condition code register
i = index registers X or Y
opr8 = 8 bit operand
opr16 = 16 bit operand
* = infinity or till reset
B = number of bytes (X/Y)
CC = number of E cycles (X/Y)

E frequency is 1/4 that of crystal frequency

Flags:

S Stop disable — X X-interrupt mask
H Half carry (from bit 3) — I I-interrupt mask
N Negative — Z Zero
V Overflow — C Carry/Borrow from msb

Flag status after instr:

\- unchanged — ? not defined
| changed accordingly — 0 cleared
1 set — \ may be 0, cannot become 1

SP points to the next unused byte at the top of the stack

8 BIT LOAD REGISTER

			SXHINZVC	B	CC
LDAa	#data8	a ← opr8	----||0-	2	2
LDAa	adr8			2	3
LDAa	adr16			3	4
LDAa	off,i			2/3	4/5

16-BIT LOAD REGISTER

			SXHINZVC	B	CC
LDD	#data16	D ← opr16	----||0-	3	3
LDD	adr8			2	4
LDD	adr16			3	5
LDD	off,i			2/3	5/6
LDS	#data16	SP ← opr16	----||0-	3	3
LDS	adr8			2	4
LDS	adr16			3	5
LDS	off,i			2/3	5/6
LDi	#data16	i ← opr16	----||0-	3/4	3/4
LDi	adr8			2/3	4/5
LDi	adr16			3/4	5/6
LDi	off,X			2/3	5/6
LDi	off,Y			3	6

8-BIT STORE TO MEMORY

			SXHINZVC	B	CC
STAa	adr8	opr8 ← a	----||0-	2	3
STAa	adr16			3	4
STAa	off,i			2/3	4/5

16-BIT STORE TO MEMORY

			SXHINZVC	B	CC
STD	adr8	opr16 ← D	----||0-	2	4
STD	adr16			3	5
STD	off,i			2/3	5/6
STS	adr8	opr16 ← SP	----||0-	2	4
STS	adr16			3	5
STS	off,i			2/3	5/6
STi	adr8	opr16 ← i	----||0-	2/3	4/5
STi	adr16			3/4	5/6
STi	off,X			2/3	5/6
STi	off,Y			3	6

8-BIT TRANSFER

		SXHINZVC	B	CC
TAB	B ← A	----||0-	1	2
TBA	A ← B	----||0-	1	2
TAP	ccr ← A	|\?|||||	1	2
TPA	A ← ccr	--------	1	2

16-BIT TRANSFER

		SXHINZVC	B	CC
TSi	i ← SP+1	--------	1/2	3/4
TiS	SP ← i-1	--------	1/2	3/4

16-BIT EXCHANGE

		SXHINZVC	B	CC
XGDi	i ↔ D	--------	1/2	3/4

8-BIT PUSH/PULL

		SXHINZVC	B	CC
PSHa	(stk) ← a; SP ← SP-1	--------	1	3
PULa	SP ← SP+1; a ← (stk)	--------	1	4

16-BIT PUSH/PULL

		SXHINZVC	B	CC
PSHi	(stk) ← i; SP ← SP-2	--------	1/2	4/5
PULi	SP ← SP+2; i ← (stk)	--------	1/2	5/6

8-BIT CLEAR

			SXHINZVC	B	CC
CLRa		a ← 0	----0100	1	2
CLR	adr16	opr8 ← 0		3	6
CLR	off,i			2/3	6/7

8-BITS ADDITIONS

			SXHINZVC	B	CC
ADCa	#data8	a ← a+opr8+C	--|-||||	2	2
ADCa	adr8			2	3
ADCa	adr16			3	4
ADCa	off,i			2/3	4/5
ADDa	#data8	a ← a+opr8	--|-||||	2	2
ADDa	adr8			2	3
ADDa	adr16			3	4
ADDa	off,i			2/3	4/5
ABA		A ← A+B	--|-||||	1	2

16-BITS ADDITIONS

			SXHINZVC	B	CC
ABi		i ← i+B	--------	1/2	3/4
ADDD	#data16	D ← D+opr16	----||||	3	4
ADDD	adr8			2	5
ADDD	adr16			3	6
ADDD	off,i			2/3	6/7

8-BIT SUBTRACTIONS

			SXHINZVC	B	CC
SUBa	#data8	a ← a - opr8	----||||	2	2
SUBa	adr8			2	3
SUBa	adr16			3	4
SUBa	off,i			2/3	4/5
SBA		A ← A-B	----||||	1	2
SBCa	#data8	a ← a-opr8-C	----||||	2	2
SBCa	adr8			2	3
SBCa	adr16			3	4
SBCa	off,i			2/3	4/5

16-BIT SUBTRACTION

			SXHINZVC	B	CC
SUBD	#data16	D ← D - opr16	----||||	3	4
SUBD	adr8			2	5
SUBD	adr16			3	6
SUBD	off,i			2/3	6/7

8-BIT INCREMENT/DECREMENT

			SXHINZVC	B	CC
INCa		opr8 ← opr8+1	----|||-	1	2
INC	adr16			3	6
INC	off,i			2/3	6/7
DECa		opr8 ← opr8-1	----|||-	1	2
DEC	adr16			3	6
DEC	off,i			2/3	6/7

16-BIT INCREMENT/DECREMENT

INS		SP←SP+1	--------	1	3
INi		i ← i+1	-----|--	1/2	3/4
DES		SP ← SP-1	--------	1	3
DEi		i ← i-1	-----|--	1/2	3/4

MULTIPLY & DIVIDE

			SXHINZVC	B	CC
MUL		D ← A * B	-------|	1	10
FDIV	fractional	D/X;X←q, D←r	-----|||	1	41
IDIV	integer	D/IX;IX←q, D←r	-----|0|	1	41

8-BIT BCD ADJUST

		SXHINZVC	B	CC
DAA	dec adjust A to BCD	----||?|	1	2

8-BIT COMPARE

			SXHINZVC	B	CC
CBA		A - B	----||||	1	2
CMPa	#data8	a - opr8	----||||	2	2
CMPa	adr8			2	3
CMPa	adr16			3	4
CMPa	off,i			2/3	4/5

16-BIT COMPARE

			SXHINZVC	B	CC
CPD	#data16	D - opr16	----||||	4	5
CPD	adr8			3	6
CPD	adr16			4	7
CPD	off,i			3	7

CPi	#data16	i - opr16	----||||	3/4 4/5
CPi	adr8			2/3 5/6
CPi	adr16			3/4 6/7
CPi	off,X			2/3 6/7
CPi	off,Y			3 7

8-BIT TEST — SXHINZVC B CC

TSTa		opr8 - 0	----||00	1 2
TST	adr16			3 6
TST	off,i			2/3 6/7

8-BIT LOGICAL BIT AND TEST — SXHINZVC B CC

BITa	#data8	a & opr8	----||0-	2 2
BITa	adr8			2 3
BITa	adr16			3 4
BITa	off,i			2/3 4/5

8-BIT LOGICAL OPERATIONS — SXHINZVC B CC

ANDa	#data8	a ← a & opr8	----||0-	2 2
ANDa	adr8			2 3
ANDa	adr16			3 4
ANDa	off,i			2/3 4/5
EORa	#data8	a ← a ^ opr8	----||0-	2 2
EORa	adr8			2 3
EORa	adr16			3 4
EORa	off,i			2/3 4/5
ORAa	#data8	a ← a | opr8	----||0-	2 2
ORAa	adr8			2 3
ORAa	adr16			3 4
ORAa	off,i			2/3 4/5

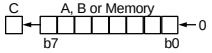
8-BIT NEGATE & COMPLEMENT — SXHINZVC B CC

NEGa		opr8 ← -opr8	----||||	1 2
NEG	adr16			3 6
NEG	off,i			2/3 6/7
COMa		opr8 ← ~opr8	----||01	1 2
COM	adr16			3 6
COM	off,i			2/3 6/7

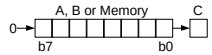
8-BIT BIT SET/CLEAR — SXHINZVC B CC

BCLR	adr8,msk	opr8 ← opr8 & ~msk	----||0-	3 6
BCLR	off,i,msk			3/4 7/8
BSET	adr8,msk	opr8 ← opr8 | msk	----||0-	3 6
BSET	off,i,msk			3/4 7/8

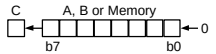
8-BIT SHIFTS — SXHINZVC B CC



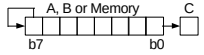
LSLa		----||||	1 2
LSL	adr16		3 6
LSL	off,i		2/3 6/7



LSRa		----0|||	1 2
LSR	adr16		3 6
LSR	off,i		2/3 6/7

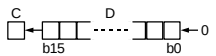


ASLa		----||||	1 2
ASL	adr16		3 6
ASL	off,i		2/3 6/7

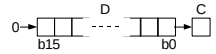


ASRa		----||||	1 2
ASR	adr16		3 6
ASR	off,i		2/3 6/7

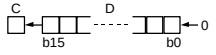
16-BIT SHIFTS — SXHINZVC B CC



LSLD		----||||	1 3

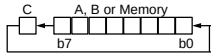


LSRD		----0|||	1 3

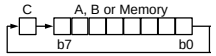


ASLD		----||||	1 3

8-BIT ROTATE — SXHINZVC B CC



ROLa		----||||	1 2
ROL	adr16		3 6
ROL	off,i		2/3 6/7



RORa		----||||	1 2
ROR	adr16		3 6
ROR	off,i		2/3 6/7

CONDITION CODE BIT SET/CLEAR

CLC		C bit ← 0	-------0	1 2
CLI		I bit ← 0	---0----	1 2
CLV		V bit ← 0	------0-	1 2
SEC		C ← 1	-------1	1 2
SEI		I ← 1	---1----	1 2
SEV		V ← 1	------1-	1 2

UNCONDITIONAL BRANCH/JUMPS

BRA	rel		--------	2 3
JMP	adr16			3 3
JMP	off,i			2/3 3/4

SIMPLE CONDITIONAL BRANCHES

BMI	rel	if minus (N=1)	--------	2 3
BPL	rel	if plus (N=0)		2 3
BEQ	rel	if equal (Z=1)		2 3
BNE	rel	if not equal 0 (Z=0)		2 3
BVS	rel	if overflow set (V=1)		2 3
BVC	rel	if overflow clear (V=0)		2 3
BCS	rel	if carry set (C=1)		2 3
BCC	rel	if carry clear (C=0)		2 3

SIGNED CONDITIONAL BRANCHES

BGT	rel	if r > m	--------	2 3
BGE	rel	if r >= m		2 3
BEQ	rel	if r == m		2 3
BNE	rel	if r !=m		2 3
BLE	rel	if r <= m		2 3
BLT	rel	if r < m		2 3

UNSIGNED CONDITIONAL BRANCHES

BHI	rel	if r > m	--------	2 3
BHS	rel	if r >= m (or BCC)		2 3
BEQ	rel	if r == m		2 3
BNE	rel	if r != m		2 3
BLS	rel	if r <= m		2 3
BLO	rel	if r < m (or BCS)		2 3

BRANCH IF BIT(S) SET/CLEAR — SXHINZVC B CC

BRCLR	adr8,msk,rel	bit(s) indicated	--------	4 6
BRCLR	off,i,msk,rel	in the msk byte		4/5 7/8
BRSET	adr8,msk,rel	branch if all	--------	3 6
BRSET	off,i,msk,rel	indicated bits set/clr		4/5 7/8

SUBROUTINE (ISR) BRANCH/JUMP/RETURN

BSR	rel	psh PC, PC ← opr	--------	2 6
JSR	adr8			2 5
JSR	adr16			3 6
JSR	off,i			2/3 6/7
RTS		pull PC	--------	1 5
SWI		psh PC,Y,X,A,B,CCR;I←1;PC←vector		
			---1----	1 14
		(same as hardware I- & X-interrupt)		
RTI		pull CCR,B,A,X,Y,PC	|||||||	1 12

OTHERS — SXHINZVC B CC

BRN	rel	never branch	--------	2 3
WAI		psh PC,Y,X,A,B,ccr; wait interr		
			--------	1 14+n
NOP		no operation	--------	1 2
TEST		addr bus count in test mode		
			--------	1 *
STOP		stop internal clock	--------	1 2

hex	***msd***	0	1	2	3	4	5	6	7
lsd	bin	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL