3.4.2.1. INSTRUCCIONES ARITMETICAS.

En la Tabla 3.4.1. se listan las instrucciones Aritméticas en orden alfabético de acuerdo a los mnemónicos, observe como aparecen realzadas algunas letras de la función en Inglés que corresponden con los mnemónicos.

Tabla 3.4.1. Lista de Instrucciónes Aritméticas

Mnemónico	Función en Inglés	Función en Español
ABA	Add B to A	Sumar B a A
ABX	Add B to X	Sumar B a X
ABY	Add B to Y	Sumar B a Y
ADCA	Add with Carry to A	Sumar con Cargo a A
ADCB	Add with Carry to B	Sumar con Cargo a B
ADDA	Add Memory to A	Sumar Memoria con A
ADDB	Add Memory to B	Sumar Memoria con B
ADDD	Add Memory to D	Sumar Memoria con D
CBA	Compare B to A	Comparar B con A
CMPA	Compare A to Memory	Comparar A con Memoria
CMPB	Compare B to Memory	Comparar B con Memoria
CPD	Compare D to Memory	Comparar D con Memoria
CPX	Compare X to Memory	Comparar X con Memoria
CPY	Compare Y to Memory	Comparar Y con Memoria
DAA	Decimal Adjust A	Ajuste Decimal de A
DEC	Decrement Memory	Decrementar Memoria
DECA	Decrement A	Decrementar A
DECB	Decrement B	Decrementar B
DES	Decrement Stack	Decrementar Stack
DEX	Decrement X	Decrementar X
DEY	Decrement Y	Decrementar Y
FDIV	Fractional Divide	División Fraccional
IDIV	Integer Divide	División Entera
INC	Increment Memory	Incrementar Memoria
INCA	Increment A	Incrementar A
INCB	Increment B	Incrementar B
INS	Increment Stack	Incrementar Stack
INX	Increment X	Incrementar X
INY	Increment Y	Incrementar Y
MUL	Multiply 8 by 8	Multiplicar de 8 por 8
NEG	Negate (2's Complement) memory	Negativo(Complemento a 2)de Memoria
NEGA	Negate (2's Complement) A	Negativo (Complemento a 2) de A
NEGB	Negate (2's Complement) B	Negativo (Complemento a 2) de B
SBA	Subtract B to A	Resta B de A
SBCA	Subtract with Carry from A	Resta con Acarreo de A
SBCB	Subtract with Carry from B	Resta con Acarreo de B
SUBA	Subtract Memory from A	Resta Memoria de A
SUBB	Subtract Memory from B	Resta Memoria de B
SUBD	Subtract Memory from D	Resta Memoria de D

"ABA" Add Accumulator B to Accumulator A:
Operación: (A) ⇦ (A) + (B)
Descripción: Suma el contenido del acumulador B al contenido del acumulador A y el resultado es colocado en el acumulador A. El contenido del acumulador B no cambia.

	S	X	H	I	N	Z	V	C
CCR	-	-	⇕	-	⇕	⇕	⇕	⇕

Mnemónico: ABA
Código de Operación:1 B

Antes de ejecutar la instrucción:ABA

Localidad de Memoria	Contenido
100	1B
101	

(A)= A3
(B)= 34
S X H I N Z V C
(CCR)= - - - - - - - -

Después de ejecutar la instrucción: ABA

Localidad de Memoria	Contenido
100	1B
101	

(A)= D7
(B)= 34
S X H I N Z V C
(CCR)= - - 0 - 1 0 0 0

Debido a que en la instrucción ABA es posible afectar los 5 bits de status (H, N, Z, V y C) del Registro de Código de Condiciones CCR, se presenta en seguida una serie de ejemplos donde se muestra: la suma en hexadecimal, su equivalente binaria, los bits afectados del CCR y algunas notas aclarativas.

Ejemplo de Suma donde es afectado el bit H del CCR.

SUMA HEXADECIMAL

```
  18
+ 18
  30
```

SUMA BINARIA EQUIVALENTE

```
    1⇦half carry
  00011000
+ 00011000
  00110000
```

CCR después de la operación:

S	X	H	I	N	Z	V	C
NA	NA	1	NA	NC	NC	NC	NC

NA= No Afectado
NC= No Cambia en este ejemplo

Nota: El bit H (half carry) es puesto en 1 porque se produce un acarreo intermedio al realizar la suma de los bit 3 de los sumandos. El bit H se utiliza en conjunto con la instrucción DAA para la corrección de números en código BCD.

Ejemplo de Suma donde son afectados los bits N y V del CCR.

SUMA HEXADECIMAL

```
   78
+  50
   --
   C8
```

SUMA BINARIA EQUIVALENTE

```
   01111000
+  01010000
   --------
   11001000
   ⇧
bit más significativ
o
```

CCR después de la operación:

S	X	H	I	N	V	C
NA	NA	NC	NA	1	1	NC

NA = No Afectado
NC= No Cambia en este ejemplo

Notas:

1. En los números con signo, cuando el bit más significativo vale 0 (cero) indica número positivo, cuando el bit más significativo vale 1 (uno) indica número negativo. En éste ejemplo el bit más significativo tomó el valor de 1 después de la suma, lo cual ocasionó que el bit N se hiciera 1.

2. El bit V de sobreflujo (overflow) se activa porque al operar con números con signo, el máximo número positivo es 7F hex (127 decimal) y el resultado supera este valor.

Ejemplo de suma donde es afectado el bit C del CCR.

SUMA HEXADECIMAL

```
   78
+  A0
   11
    8
```

SUMA BINARIA EQUIVALENTE

```
Carry ⇨ 111
          01111000
        + 10100000
Carry ⇨ 100011000
```

CCR *después de la operación:*

S	X	H	I	N	Z	V	C
NA	NA	NC	NA	NC	NC	NC	1

NA = No Afectado
NC= No Cambia en este ejemplo

Nota: El bit C (carry) es puesto en 1 porque se produce un acarreo *(cargo = carry) al realizar la suma de los bits 7 de los sumandos.*

"ABX" Add accumulator **B** to Index Register **X**.
Operación: (X)⇦ (X) + (B)
Descripción: Suma el contenido del acumulador B no-signado, al contenido del Registro Indice X considerando el acarreo (carry), el resultado es colocado *en el Registro Indice X y el acumulador B no cambia.*

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	-	-	-	-

Mnemónico: ABX
Código de Operación: 3A

Antes de ejecutar la instrucción: ABX

Localidad de Memoria	Contenido
101	3A

(X)=	3124
(B)=	51
X+B=	3175

Después de ejecutar la instrucción: ABX

Localidad de Memoria	Contenido
101	3A

(X)=	3175
(B)=	51

"DECA" Decrement Accumulator A
Operación: (A) ⇦ (A) - 1
Descripción: Resta 1 del contenido del acumulador A y el resultado es colocado en el acumulador A. Los bits N, Z y V en el CCR son puestos en 1 ó 0 de acuerdo al resultado de la operación, el bit C no es afectado por la operación. Cuando opera con valores no signados, solo se puede contar con las instrucciones de valores de complemento a 2, además todas las instrucciones de bifurcación son disponibles.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇕	⇕	⇕	-

Mnemónico: DECA
Código de Operación: 4A

Antes de ejecutar la instrucción: DECA

Localidad de Memoria	Contenido
106	4A
107	

(A)= 38

Después de ejecutar la instrucción: DECA

Localidad de Memoria	Contenido
106	4A
107	

(A)= 37

"DES" Decremento Stack Pointer
Operación: (SP) ⇦ (SP) - $0001
Descripción: resta 1 del contenido del Stack Pointer y el resultado es colocado en el Stack Pointer.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	-	-	-	-

Mnemónico: DES
Código de Operación: 34

Antes de ejecutar la instrucción: DES

Localidad de Memoria	Contenido
107	34
108	

(SP)= 00FF

Después de ejecutar la instrucción: DES

Localidad de Memoria	Contenido
107	34
108	

(SP)= 00FE

"CMPA" Compare Accumulator A to Memory

Operación: (A) - (M)

Descripción: *resta al contenido del acumulador A el contenido de la localidad de memoria M con el propósito de poner en 1 o 0 los bits N, Z, V y C del CCR para su uso en instrucciones de bifurcación.*

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇕	⇕	⇕	⇕

Mnemónico: CMPA

Código de Operación: 34

Modo de Direccionamiento	Código de Operación	Listado Fuente	Notas de Explicación
Inmediato	81	CMPA #$63	1 ver ejem.
Directo	91	CMPA $82	2
Extendido	B1	CMPA $8671	3
Indexado X	A1	CMPA $23, X	4
Indexado Y	18A1	CMPA $23, Y	5

NOTAS:

1.- (A) - 63 hex
2.- (A) - (82 hex)
3.- (A) - (8671 hex)
4.- (A) - ((X) + 23 hex)
5.- (A) - ((Y) + 23 hex)

Antes de ejecutar la instrucción: CMPA #$63

Localidad de Memoria	Contenido
108	81
109	63

(A)= 18 = 00011000
(109)= 63 = 01100011

SXH I NZVC
(CCR) = ————

Después de ejecutar la instrucción: CMPA #$63

Localidad de Memoria	Contenido
108	81
109	63

(A)= 18 = 00011000
(109)= 63 = 01100011

SXHINZVC
(CCR) = —0-1000

"INC" (Increment Memory Byte)
Operación: (M) ⇦ (M) + $01
Descripción: *suma 1 al contenido de la localidad* de memoria M, y el resultado se coloca en la localidad de memoria M. Los bits N, Z y V en el CCR son puestos en 1 ó 0 de acuerdo al resultado de la operación, el bit C, *no es* afectado por la operación. Cuando opera con valores no signados, solo se puede contar con las instrucciones de bifurcación BEQ y BNE. Cuando opera *con valores de complemento a 2, todas las instrucciones de bifurcación son* disponibles.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇧	⇧	⇧	-

Mnemónico: INC

Modo de Direccionamiento	Código de Operación	Listado Fuente	Notas de Explicación
Extendido	7C	INC $1823	1
Indexado X	6C	INC $14, X	2
Indexado Y	186C	INC $ AC,Y	3

NOTAS:

1.- (1823 hex) ⇦ (1823 hex) + 1
2.- ((X) + 14 hex) ⇦ ((X) + 14 hex) + 1
3.- ((Y) + AC hex) ⇦ ((Y) + AC hex) + 1

Antes de ejecutar la instrucción: INC $AC,Y

Localidad de Memoria	Contenido
110	18
111	6C
43B4	42

(Y) = 4308

Y+14 = 43B4

Después de ejecutar la instrucción: INC $AC,Y

Localidad de Memoria	Contenido
110	18
111	6C
43B4	43

(Y) = 4308

Y+14 = 43B4

"NEGB" Negate B
Operación: (B) = - (B) = $00 - (B)
Descripción: Remplaza el contenido del acumulador B con su complemento a 2; si el valor es $80 permanece sin cambio.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇕	⇕	⇕	⇕

Mnemónico: NEGB
Código de Operación: 50

Antes de ejecutar la instrucción: NEGB

Localidad de Memoria	Contenido
112	50
113	

(B)= 60 = 01100000

(CCR) = SXHINZVC ———

Después de ejecutar la instrucción: NEGB

Localidad de Memoria	Contenido
112	50
113	

(B) = 10100000

(CCR) = SXHINZVC ——1001

"SBCA" Subtract with Carry from A
Operación: (A) ⇦ (A) - (M) - (C)
Descripción: resta el contenido de A y coloca el resultado en A. Para esta instrucción el bit C del CCR representa un préstamo (borrow).

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇧	⇧	⇧	⇧

Mnemónico: SBCA

Modo de Direccionamiento	Código de Operación	Listado Fuente	Notas de Explicación
Inmediato	82	SBCA #$23	1 ver ejem.
Directo	92	SBCA $23	2
Extendido	B2	SBCA $D323	3
Indexado X	A2	SBCA $16,X	4
Indexado Y	18A2	SBCA $38,Y	5

NOTAS:
1.- (A) ⇦ (A) - 23 hex - (C)
2.- (A) ⇦ (A) - (23 hex) - (C)
3.- (A) ⇦ (A) - (D323 hex) - (C)
4.- (A) ⇦ (A) - ((X) + 16 hex) - (C)
5.- (A) ⇦ (A) - ((Y) + 38 hex) - (C)

Antes de ejecutar la instrucción: SBCA #$23

Después de ejecutar la instrucción: SBCA #$23

3.4.2.2. INSTRUCCIONES LOGICAS

En la tabla 3.4.2. se listan las instrucciones lógicas en orden alfabético de acuerdo a los mnemónicos, observe como aparecen realzadas algunas letras de la función en inglés que corresponden a los mnemónicos.

Tabla 3.4.2. Instrucciones Lógicas

Mnemónico	Función en Inglés	Función en Español
ASL	Arithmetic Shift Left	Correr Aritmeticamente a la Izq.
ASLA	Arithmetic Shift Left A	Correr A Aritmeticamente a la Izq.
ASLB	Arithmetic Shift Left B	Correr B Aritmeticamente a la Izq
ASLD	Arithmetic Shift Left D	Correr D Aritmeticamente a la Izq.
ASR	Arithmetic Shift Right	Correr Aritmeticamente a la Der.
ASRA	Arithmetic Shift Right A	Correr A Aritmeticamente a la Der.
ASRB	Arithmetic Shift Right B	Correr B Aritmeticamente a la Der.
BITA	Bits Test A with Memory	Prueba los bits de A con Memoria
BITB	Bits Test B with Memory	Prueba los bits de B con Memoria
COM	1's Complement Memory	Complemento a 1 de Memoria
COMA	1's Complement A	Complemento a 1 de A
COMB	1's Complement B	Complemento a 1 de B
EOR	Exclusive OR A with Memory	OR Exclusivo de A con Memoria
EORB	Exclusive OR B with Memory	OR Exclusivo de B con Memoria
LSL	Logical Shift Left	Corrimiento Lógico a la Izquierda
LSLA	Logical Shift Left A	Corrimiento Lógico a la Izq. de A
LSLB	Logical Shift Left B	corrimiento Lógico a la Izq. de B
LSLD	Logical Shift Left D	Corrimiento Lógico a la Izq. de D
LSR	Logical Shift Right	Corrimiento Lógico a la Derecha
LSRA	Logical Shift Right A	Corrimiento Lógico a la Der. de A
LSRB	Logical Shift Right B	Corrimiento Lógico a la Der. de B
LSRD	Logical Shift Right D	Corrimiento Lógico a la Der. de D
ORAA	OR A	OR Inclusive de A con Memoria
ORAB	OR B	OR Inclusive de B con Memoria
ROL	Rotate Left	Rotación a la Izquierda
ROLA	Rotate Left A	Rotar A a la Izquierda
ROLB	Rotate Left B	Rotar B a la Izquierda
ROR	Rotate Right	Rotación a la Derecha
RORA	Rotate Right A	Rotar A a la Derecha
RORB	Rotate Right B	Rotar B a la Derecha

"ANDA" AND A *with Memory*
Operación: (A) ⇦ (A) . (M)
Descripción: Realiza el AND lógico entre el contenido del acumulador A y el contenido de la localidad de memoria M, y coloca el resultado en el acumulador A. Cada bit del acumulador A es "ANDeado" con el correspondiente bit del contenido de la localidad N.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇧	⇧	0	-

Mnemónico: "ANDA"

Modos de Direccionamiento	Código de Operación	Listado Fuente	Notas de Explicación
Inmediato	84	ANDA #$18	1
Directo	94	ANDA $36	2
Extendido	B4	ANDA $1836	3
Indexado X	A4	ANDA $08,X	4 Ver Ejem.
Indexado Y	18A4	ANDA #E9,Y	5

NOTAS: AND Lógico se representa con " · "
1.- (A) ⇦ (A) ·18 hex
2.- (A) ⇦ (A) · (36 hex)
3.- (A) ⇦ (A) · (1836 hex)
4.- (A) ⇦ (A) · ((X) + 08 hex)
5.- (A) ⇦ (A) · ((Y) + E9 hex)

Antes de ejecutar la instrucción: ANDA $08,X

Localidad de Memoria	Contenido
120	A4
121	08
122	
0338	9D

	(X)	=	0330
(A)=	5A	=	01011010
(0338)=	9D	=	10011101
			SXHINZVC
	(CCR)	=	————

Después de ejecutar la instrucción: ANDA $08,X

Localidad de Memoria	Contenido
120	A4
121	08
122	
0338	9D

	(X)	=	0330
(A)=	18	=	00011000
			SXHINZVC
	(CCR)	=	——000–

"ASLD" Arithmetic Shift Left D

Operación: C ⇦ A [b7 - - - - - - b0] ⇦ B [b7 - - - - - - b0] ⇦ 0

Descripción: Desplaza todos los bits del acumulador D un lugar hacia la izquierda. El bit 0 es cargado con un cero. El bit C en el CCR es cargado con el bit más significativo del acumulador D.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇕	⇕	⇕	⇕

Mnemónico: ASLD
Código de Operación: 05

Antes de ejecutar la instrucción: ASLD

Localidad de Memoria	Contenido
122	05
123	

C: 0
D: 1 0 1 1 0 1 0 1 1 1 0 1 0 1 1 1

Después de ejecutar la instrucción: ASLD

Localidad de Memoria	Contenido
122	05
1234	

C: 1
D: 0 1 1 0 1 0 1 1 1 0 1 0 1 1 1 0

"ASRB" Arithmetic Shift Right B

Operación: → [b7 - - - - - - b0] ⇨ C (← ↑ B)

Descripción: Desplazar todos los bits del acumulador B un lugar hacia la derecha. El bit 7 es mantenido constante. El bit 0 es cargado en el bit C del CCR.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇕	⇕	⇕	⇕

Mnemónico: ASRB
Código de Operación: 57

Antes de ejecutar la instrucción: ASRB

Después de ejecutar la instrucción: ASRB

"EORA" Exclusive **OR A** with Memory
Operación: (A) ⇦ A ⊕ M
Descripción: Realiza la función lógica OR exclusiva entre el contenido del acumulador A y el contenido de la localidad de memoria M y coloca el resultado en el acumulador A. Con cada bit del acumulador A se realiza la función OR exclusiva con el correspondiente bit de la localidad de memoria M.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇧	⇧	0	⇧

Mnemónico: EORA

Modo de Direccionamiento	Código de Operación	Listado Fuente	Notas de Explicación
Inmediato	88	EORA #$87	1 ver ejem.
Directo	98	EORA $36	2
Extendido	B8	EORA $D344	3
Indexado X	A8	EORA $31,X	4
Indexado Y	18A8	EORA $71,Y	5

NOTAS: OR exclusive se representa con " ⊕ "
1.- (A) ⇦ (A) ⊕ 87 hex
2.- (A) ⇦ (A) ⊕ (36 hex)
3.- (A) ⇦ (A) ⊕ (D344 hex)
4.- (A) ⇦ (A) ⊕ ((X) + 31 hex)
5.- (A) ⇦ (A) ⊕ ((Y) + 71 hex)

166783

Antes de ejecutar la instrucción: EORA #$87

Localidad de Memoria	Contenido
124	88
125	87
126	

(A)= 01101110
87= 10000111

Después de ejecutar la instrucción: EORA #$87

Localidad de Memoria	Contenido
124	88
125	87
126	

(A)= 11101001

"LSL" Logical Shift Left = ASL

Operación: C ⇦ b7 - - - - - - b0 ⇦ 0 (⇐)

Descripción: Desplaza todos los bits de la localidad de memoria M un lugar a la izquierda. El bit 0 es cargado con cero. El bit C es cargado con el bit más significativo de la localidad de memoria M.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇕	⇕	⇕	⇕

Mnemónico: LSL

Modo de Direccionamiento	Código de Operación	Listado Fuente	Notas de Explicación
Extendido	78	LSL $1243	1 ver ejem.
Indexado X	68	LSL $18,X	2
Indexado Y	1868	LSL $15,Y	3

NOTAS:
1. Desplaza hacia la Izquierda todos los bits de la localidad 1243 hex.
2. Desplaza hacia la Izquierda todos los bits de la localidad X + 18 hex.
3. Desplaza hacia la Izquierda todos los bits de la localidad Y + 15 hex.

Antes de ejecutar la instrucción: LSL $1243

Localidad de Memoria	Contenido
126	78
127	12
128	43
129	
1243	6D

(1243) = 01101101 = 6D

SXHINZVC

(CCR) = - - - - - - - -

Después de ejecutar la instrucción: LSL $1243

Localidad de Memoria	Contenido
126	78
127	12
128	43
129	
1243	DA

(1243) = 11011010 = DA

SXHINZVC

(CCR) = - - - - 1000

"ORAB" OR Accumulator B

Operación: (B) ⇦ (B) + (M)

Descripción: Realiza la función lógica OR entre el contenido de B y M y el resultado es colocado en A. Con cada bit del acumulador B se realiza la función OR con los correspondientes bits del contenido de la localidad M.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇕	⇕	0	-

Mnemónico: ORAB

Modo de Direccionamiento	Código de Operación	Listado Fuente	Notas de Explicación
Inmediato	CA	ORAB #$27	1
Directo	DA	ORAB $48	2 ver ejem.
Extendido	FA	ORAB $E316	3
Indexado X	EA	ORAB $07,X	4
Indexado Y	18EA	ORAB $10,Y	5

NOTAS: OR se representa con " + "

1.- (B) ⇦ (B) + 27 hex
2.- (B) ⇦ (B) + (48 hex)
3.- (B) ⇦ (B) + (E316 hex)
4.- (B) ⇦ (B) + ((X) + 07 hex)
5.- (B) ⇦ (B) + ((Y) + 10 hex)

Antes de ejecutar la instrucción: ORAB $48

Localidad de Memoria	Contenido
129	DA
130	48
131	
132	
48	C6

(B) = 01101101

(48) = 11000110

B+48 = 11101111

(CCR) = SXHINZVC - - - - - - - -

Después de ejecutar la instrucción: ORAB $48

Localidad de Memoria	Contenido
129	DA
130	48
131	
132	
48	C6

(B) = 11101111

(CCR) = SXHINZVC - - - - 1000

"ROLA" Rotate Left Accumulator A

Descripción: Desplaza todos los bits de A un lugar hacia la izquierda. El bit 0 es cargado con el bit C y el bit C es cargado con el bit más significativo del acumulador A.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇕	⇕	⇕	⇕

Mnemónico: ROLA

Código de Operación: 49

Antes de ejecutar la instrucción: ROLA

Localidad de Memoria	Contenido
131	49
132	

C: 0

A: 11011101

Después de ejecutar la instrucción: ROLA

Localidad de Memoria	Contenido
131	49
132	

C: 1

A: 10111010

3.4.2.3. INSTRUCCIONES DE TRANSFERENCIA DE DATOS.

En la Tabla 3.4.3. se listan las instrucciones que se consideran forman parte del grupo de Instrucciones de Transferencia de Datos.

Tabla 3.4.3 Instrucciones de Transferencia de Datos

Mnemónico	Función en Inglés	Función en Español
LDAA	Load A	Cargar A
LDAB	Load B	Cargar B
LDD	Load D	Cargar D
LDS	Load Stack Pointer	Cargar Stack
LDX	Load X	Cargar X
LDY	Load Y	Cargar Y
PSHA	Push A onto Stack	"Empujar" A al Stack
PSHB	Push B onto Stack	"Empujar" B al Stack
PSHX	Push X onto Stack	"Empujar" X al Stack
PSHY	Push Y onto Stack	"Empujar" Y al Stack
PULA	Pull A from Stack	"Jala" A desde el Stack
PULB	Pull B from Stack	"Jala" B desde el Stack
PULX	Pull X from Stack	"Jala" X desde el Stack
PULY	Pull Y from Stack	"Jala" Y desde el Stack
STAA	Store A	Almacenar A
STAB	Store B	Almacenar B
STD	Store D	Almacenar D
STS	Store Stack Pointer	Almacenar Stack
STX	Store X	Almacenar X
STY	Store Y	Almacenar Y
TAB	Transfer A to B	Transferir A a B
TAP	Transfer A to CCR	Transferir A a CCR
TBA	Transfer B to A	Transferir B a A
TPA	Transfer CCR to A	Transferir CCR a A
TSX	Transfer Stack to X	Transferir Stack a X
TSY	Transfer Stack to Y	Transferir Stack a Y
TXS	Transfer X to Stack	Transferir X a Stack
TYS	Transfer Y to Stack	Transferir Y a Stack
XGDX	Exchange D with X	Intercambiar D con X
XGDY	Exchange D with Y	Intercambiar D con Y

"LDAB" Load Accumulator B
Operación: (B) ⇦ M
Descripción: Carga el contenido de la localidad de memoria M en el acumulador B.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇧	⇧	0	-

Mnemónico: LDAB.

Modo de Direccionamiento	Código de Operación	Listado Fuente	Notas de Explicación
Inmediato	C6	LDAB #$18	1 ver ejem.
Directo	D6	LDAB $31	2
Extendido	F6	LDAB $1431	3
Indexado X	E6	LDAB $3,X	4
Indexado Y	18E6	LDAB $8,Y	5

NOTAS:
1.- (B) ⇦ 18 hex
2.- (B) ⇦ (31 hex)
3.- (B) ⇦ (1431 hex)
4.- (B) ⇦ ((X) + 3 hex)
5.- (B) ⇦ ((Y) + 8 hex)

Antes de ejecutar la instrucción: LDAB #18

Localidad de Memoria	Contenido
150	C6
151	18
152	

(B) = -36

SXHINZVC
(CCR) = - - - - - - - -

Después de ejecutar la instrucción: LDAB #$18

Localidad de Memoria	Contenido
150	C6
151	18
152	

(B) = 18

SXHINZVC
(CCR) = - - - - 000-

"LDS" Load Stack Pointer
Operación: SPH ⇦ (M) SPL ⇦ (M+1)
Descripción: Carga el byte más significativo del Stack Pointer con el contenido de la localidad M, y carga el byte menos significativo del Stack Pointer con el contenido de la localidad M + 1.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	⇕	⇕	0	-

Mnemónico: LDS

Modo de Direccionamiento	Código de Operación	Listado Fuente	Notas de Explicación
Inmediato	8E	LDS #$0116	1
Directo	9E	LDS $26	2 ver ejem.
Extendido	BE	LDS $2122	3
Indexado X	AE	LDS $08,X	4
Indexado Y	18AE	LDS $16,Y	5

1.- (SPH) ⇦ 1 hex (SPL) ⇦ 16 hex
2.- (SPH) ⇦ (26 hex) (SPL) ⇦ (27 hex)
3.- (SPH) ⇦ (2122 hex) (SPL) ⇦ (2123) hex)
4.- (SPH) ⇦ ((X) + 08 hex) (SPL ⇦ ((X) + 09 hex)
5.- (SPH) ⇦ ((Y) + 16 hex) (SPL) ⇦ ((Y) + 17 hex)

Antes de ejecutar la instrucción: LDS $26

Localidad de Memoria	Contenido
152	9E
153	26
154	
26	10
27	21

(SP)= 0000

SXHINZVC
(CCR)= - - - - - - - -

Después de ejecutar la instrucción: LDS $26

Localidad de Memoria	Contenido
152	9E
153	26
154	
26	10
27	21

(SP)= 1021

SXHINZVC
(CCR)= - - - - 000-

"TAP" Transfer from Accumulator **A** to Condition Code Register.
Operación: (CCR) ⇦ A
Descripción: Transfiere el contenido del acumulador A desde las posiciones del bit 7 al bit 0 a las correspondientes posiciones del CCR. El contenido del acumulador A permanece sin cambio. El bit X del CCR puede ser puesto en cero como resultado de la instrucción TAP pero no puede ser puesto en uno.

A	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
	↓	↓	↓	↓	↓	↓	↓	↓
CCR	S	X	H	I	N	Z	V	C

	S	X	H	I	N	Z	V	C
CCR	⇕	⇓	⇕	⇕	⇕	⇕	⇕	⇕

Mnemónico: TAP

3.4.2.4. INSTRUCCIONES DE BIFURCACION O SALTO.

Las instrucciones de bifurcación o salto incondicional (Jump) permiten que el control sea transferido a cualquier dirección en el mapa de memoria de 64 kbytes, en cambio las instrucciones de bifurcación o salto condicional (branch) permiten que la CPU tome decisiones basadas en el contenido de uno o varios de los bits del CCR para realizar la bifurcación. Todos los bloques de decisión en una diagrama de flujo deben corresponder a alguna instrucción de salto o bifurcación condicional. En la tabla 3.4.4. aparecen las instrucciones de Bifurcación o Salto.

Las instrucciones de salto o bifurcación condicionales tienen un rango limitado de -128 a +127 localidades. Mediante el uso de etiquetas en la elaboración del programa en lenguaje ensamblador, el cálculo de cuantas localidades hay que saltar es transparente al usuario.

Tabla 3.4.4. Instrucciones de Bifurcación o Salto

Mnemónico	Función en Inglés	Función en Español	Condición de Salto
BCC	Branch if carry clear	Saltar si carry = 0	C = 0
BCS	Branch if carry set	Saltar si carry = 1	C = 1
BEQ	Branch if equal zero	Saltar si es igual a cero	Z = 1
BGE	Branch if greater than or equal to zero	Saltar si es mayor que o igual a cero	N ⊕ V = 0
BGT	Branch if greater than zero	Saltar si es mayor que cero	Z . (N ⊕ V) = 0
BHI	Branch if higher	Salta si es más grande	C + Z = 0
BHS	Branch if higher or same	Salta si es más grande o lo mismo	C = 0
BLE	Branch if less than or equal to zero	Salta si es menor que o igual a cero	Z +(N ⊕ V) = 1
BLO	Branch if lower	Salta si es más pequeño	C = 1
BLS	Branch if lower or same	Salta si es menor o lo mismo	C + Z = 1
BIT	Branch if less than zero	Salta si es menor que cero	N ⊕ V = 1
BMI	Branch if minus	Salta si es negativo	N = 1
BNE	Branch if not equal to zero	Salta si no es igual a cero	Z = 0
BPL	Branch if plus	Salta si es positivo	N = 0
BRA	Branch Always	Salta siempre	
BRCLR	Branch if bits clear	Salta si bits = 0	
BRN	Branch never	No salta nunca	Ninguna
BRSET	Branch if bits set	Salta si bits = 1	
BVC	Branch if overflow clear	Salta si sobre flujo = 0	V = 0
BVS	Branch if overflow set	Salta si sobre flujo = 1	V = 1
JMP	Jump	Salto incondicional	Ninguna

Cuando se avanza a través de la memoria de programa ejecutando una instrucción cada vez, frecuentemente se encuentran instrucciones de salto condicional (BRANCH). El estado de ciertos bits en el Registro de Código de Condición CCR indican al MCU si el salto condicional toma lugar o no. Estos bits mediante una instrucción previa a la instrucción BRANCH deben ser puestos en "1" o "0" . Para que un salto condicional se realice, el estado de los bits debe ser favorable, si no es así, se ejecuta entonces la siguiente instrucción. En seguida se muestran algunos ejemplos donde aparecen las instrucciones de salto condicional.

Instrucciones	Estado de los bits del CCR después de ejecutar la instrucción								¿Ocurre la condición de salto?
	S	X	H	I	N	Z	V	C	
1.-LDAB $FF	NA	NA	NA	NA	1	0	0	NA	
COMB	NA	NA	NA	NA	0	1	0	1	
BCC	NA	NA	NA	NA	0	1	0	1	No
2.- LDAA #$0	NA	NA	NA	NA	0	1	0	NA	
ADDA #$80	NA	NA	0	NA	1	0	0	0	
BHI	NA	NA	0	NA	1	0	0	0	Si
3.- LDAA #$01	NA	NA	NA	NA	0	0	0	NA	
ASRA	NA	NA	NA	NA	0	1	1	1	
BLE	NA	NA	NA	NA	0	1	1	1	Si
4.- LDAA #$85	NA	NA	NA	NA	1	0	0	NA	
LDAB #$05	NA	NA	NA	NA	0	0	0	NA	
SBA	NA	NA	NA	NA	1	0	0	0	
BMI	NA	NA	NA	NA	1	0	0	0	SI
5.- LDX #$01	NA	NA	NA	NA	0	0	0	NA	
DEX	NA	NA	NA	NA	0	1	0	NA	
BEQ	NA	NA	NA	NA	0	1	0	NA	SI
6.-LDAA #$7F	NA	NA	NA	NA	0	0	0	NA	
RORA	NA	NA	NA	NA	0	0	1	1	
BVS	NA	NA	NA	NA	0	0	1	1	SI
7.-LDAA #$7F	NA	NA	NA	NA	0	0	0	NA	
ADDA #$01	NA	NA	NA	NA	1	0	1	0	
COMA	NA	NA	NA	NA	0	0	0	1	
BGE	NA	NA	NA	NA	0	0	0	1	SI
8.-LDAA #$7F	NA	NA	NA	NA	0	0	0	NA	
LDAB #$7E	NA	NA	NA	NA	0	0	0	NA	
SBA	NA	NA	NA	NA	0	0	0	0	
LSRA	NA	NA	NA	NA	0	1	1	1	
BCS	NA	NA	NA	NA	0	1	1	1	SI
9.- LDAB #$FF	NA	NA	NA	NA	1	0	0	NA	
TSTB	NA	NA	NA	NA	1	0	0	0	
BCC	NA	NA	NA	NA	1	0	0	0	SI

3.4.5. INSTRUCCIONES DE LLAMADA Y RETORNO DE SUBRUTINA.

Conceptos Generales:

Las ***Subrutinas*** son un bloque de instrucciones que son utilizadas repetidamente, y permiten de una manera sencilla, dividir el trabajo de un programa en secciones. El MCU automatiza el proceso de recordar la dirección en el Programa Principal, donde la ejecución debe reasumirse después de terminar la subrutina. En la fig. 3.4.5.1. se ilustra el uso repetido de una subrutina.

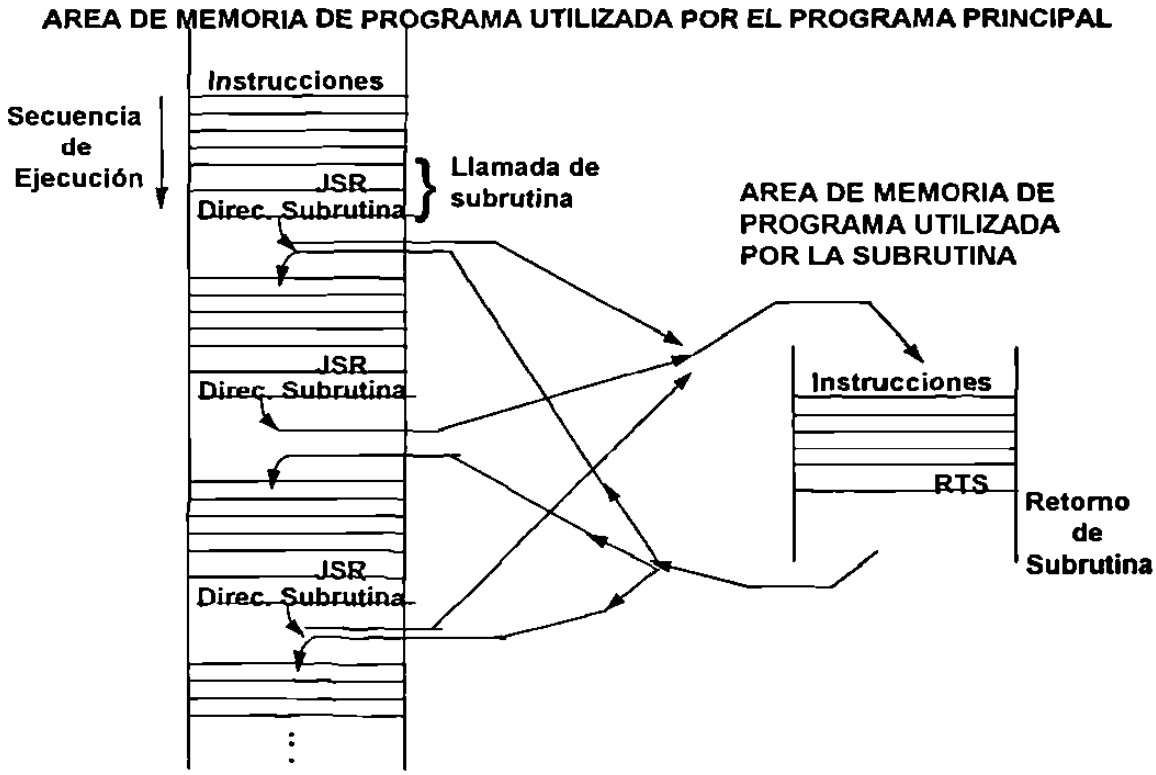


Fig. 3.4.5.1. Uso repetido de una subrutina

El Stack o Pila es un área seleccionada de memoria RAM, utilizado para almacenar en forma temporal datos o direcciones. El Stack o Pila recibe su nombre debido al hecho de que puede visualizarse como una pila (Stack) de datos, en donde los últimos datos en entrar son los primeros en salir.

El Stack Pointer (SP) o Apuntador de Pila es el registro utilizado para direccionar el Stack. El Stack Pointer o SP almacena (apunta hacia) la dirección de *la última localidad vacía utilizable para almacenar datos.* Es

sumamente importante que el Programador en las primeras instrucciones del programa, apunte el Stack Pointer a un área utilizable de la RAM con suficiente espacio, para evitar que se traslape con otros usos de la RAM que pueda dar lugar a fallas en el programa.

Para los MCU's de la familia M68HC11 conviene imaginar que las localidades o direcciones del Stack se numeran en forma semejante a como se numeran los renglones de una hoja de papel. Pero existe una diferencia importante, en el Stack la escritura de datos se empieza en las localidades con números mayores (correspondientes a los renglones inferiores) y se continua hacia las localidades con números menores (que corresponde a los renglones superiores), la lectura o extracción de datos se realiza en forma semejante tanto en el Stack como en una hoja, pero cada vez que se extrae un dato del Stack, se considera que tal localidad queda vacía, esto es ilustrado en la fig. 3.4.5.2.

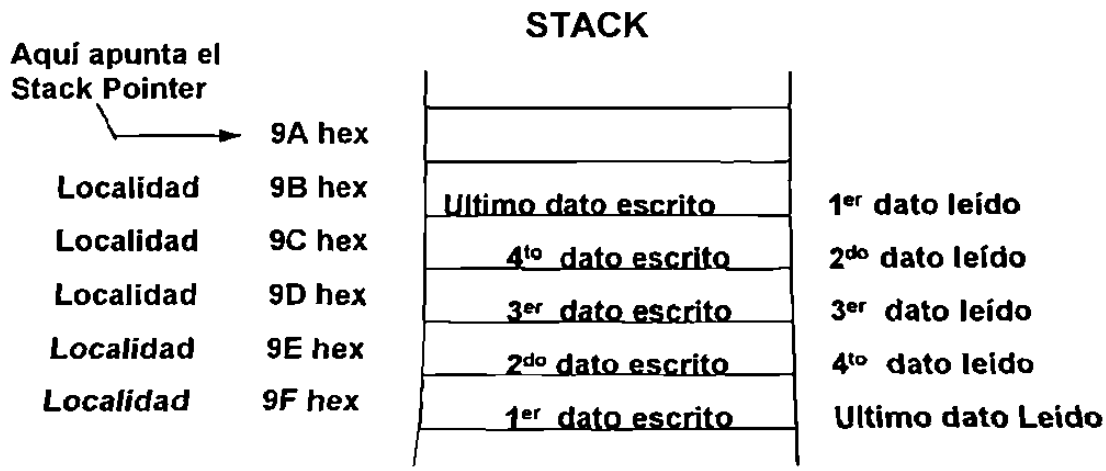


Fig. 3.4.5.2. Lectura y escritura de datos en el STACK

Solo dos operaciones son permitidas en el Stack: 1).- Escritura en la parte de arriba del Stack, a lo que normalmente se le conoce como **push.** 2).- Leer en la parte de arriba del Stack, a lo que normalmente se le conoce como **pop o pull.**

Una operación push o de escritura en el stack causa que un dato de algún registro del CPU sea escrito en la localidad de memoria a la que apunta el Stack Pointer (SP). El Stack Pointer (SP) es entonces automáticamente decrementado apuntando hacia la siguiente localidad libre en la parte de arriba del Stack.

En una operación de lectura (pop o pull) el Stack Pointer o SP es decrementado para que apunte a la dirección del último dato escrito en la parte

de arriba de la pila (stack) y entonces se realiza la operación de lectura y el dato es retornado a algún registro del CPU, después de que se realiza la lectura se considera que la localidad queda vacía.

"JSR" Jump to Subroutine
Operación: paso a paso.

1a.-	(PC)	⇦	(PC)	+	$0003 Para direccionamiento extendido.
1b.-	(PC)	⇦	(PC)	+	$0002 Para direccionamiento Directo o Indexado X.
2.-	((SP))	⇦	PCL		
3.-	(SP)	⇦	(SP)	-	$0001
4.-	((SP))	⇦	PCH		
5.-	(SP)	⇦	(SP)	-	$0001
6.-	(PC)	⇦			Dirección Efectiva.

Descripción: paso a paso.

1. El Contador de Programa PC es incrementado en 3 ó 2 dependiendo del modo de direccionamiento. Esta será la dirección de retorno de la subrutina.

2. El byte menos significativo del Contador de Programa PCL o sea el byte menos significativo de la dirección de retorno de la subrutina es escrito en el Stack en la dirección a la que apunta el Stack Pointer SP.

3. El Stack Pointer es decrementado en uno.

4. El byte más significativo del Contador de Programa PCH o sea el byte más significativo de la dirección de retorno de la subrutina es escrito en el stack, en la localidad a donde apunta el más reciente valor del Stack Pointer SP resultante del paso 3.

5. El Stack Pointer SP es decrementado en uno.

6. La Dirección Efectiva es obtenida de acuerdo a las reglas de *direccionamiento* (Extendido, Directo o Indexado), y es almacenado en el Contador de Programa PC. Un salto ocurre a la dirección efectiva a donde ahora apunta el Contador de Programa y ahí se inicia la ejecución de la subrutina.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	-	-	-	-

Mnemónico: JSR

Modo de Direccionamiento	Código de *Operación*	Listado *Fuente*	Notas de Explicación
Directo	9D	JSR $45	1
Extendido	BD	JSR $B700	2 ver ejem.
Indexado X	AD	JSR $23,X	3
Indexado Y	18AD	JSR $54,Y	4

Antes de ejecutar la instrucción: JSR $B700

PROGRAMA PRINCIPAL

Localidad de Memoria	Contenido
0160	BD
0161	B7
0162	00

(SP)= 01FF

(PC)= 0160

SUBRUTINA

Localidad de Memoria	Contenido
B700	instrucción #1
B701	instrucción #2
	•
	•
	•

STACK

localidad	contenido
01FD	
01FE	
01FF	

⇦aquí apunta el SP

Después de ejecutar la instrucción: JSR $B700

PROGRAMA PRINCIPAL

Localidad de Memoria	Contenido
0160	BD
0161	B7
0162	00

(SP)= 01FD

(PC)= B700

SUBRUTINA

Localidad de Memoria	Contenido
B700	instrucción #1
B701	instrucción #2
	•
	•
	•

STACK

localidad	*contenido*
01FD	
01FE	01
01FF	63

⇦aquí apunta el SP

"RTS" Return to Subrutine
Operación: paso a paso.

1.- (SP) ⇦ (SP) + $0001
2.- (PC) ⇦ PCH = ((SP))
3.- (SP) ⇦ (SP) + $0001
4.- (PC) ⇦ PCL = ((SP))

Descripción: paso a paso.

1. El Stack Pointer SP es incrementado en uno.

2. El contenido de la localidad de memoria hacia donde apunta el nuevo Stack Pointer es el byte más significativo del Contador Programa PCH y corresponde al byte más significativo de la dirección de retorno de la subrutina. Este contenido es leído (pulled) y almacenado como parte del Contador de Programa PC.

3. El Stack Pointer SP es otra vez incrementado en 1.

4. El contenido de la localidad de memoria a donde ahora apunta el Stack Pointer SP es el byte menos significativo del Contador de Programa PCL y corresponde al byte menos significativo de la dirección de retorno. Este contenido es leído (pulled) y almacenado en el Contador de Programa PC que ahora ya tiene su valor completo que corresponde a la dirección de retorno. El programa principal continua entonces su ejecución.

Nota: Una subrutina siempre debe terminar con una instrucción RTS.

	S	X	H	I	N	Z	V	C
CCR	-	-	-	-	-	-	-	-

Mnemónico: RTS.
Código de Operación: 39

Antes de ejecutar la instrucción: RTS

PROGRAMA PRINCIPAL

Localidad de Memoria	Contenido
0160	BD
0161	B7
0162	00

(SP)= 01FD

(PC)= B725

SUBRUTINA

Localidad de Memoria	Contenido
B700	instrucción #1
B701	instrucción #2
	•
	•
	•
B725	39

STACK

localidad	contenido	
01FD		⇦aquí apunta
01FE	01	el SP
01FF	63	

Después de ejecutar la instrucción: JSR $B700

PROGRAMA PRINCIPAL

Localidad de Memoria	Contenido
0160	BD
0161	B7
0162	00
0163	

(SP)= 01FF

(PC)= 0163

SUBRUTINA

Localidad de Memoria	Contenido
B700	instrucción #1
B701	instrucción #2
	•
	•
	•
8725	39

STACK

localidad	contenido	
01FD		
01FE		
01FF		⇦aquí apunta el SP

3.4.2.6. INSTRUCCIONES MISCELANEAS.

Aquí se incluyen algunas instrucciones diversas que no corresponden a las clasificadas anteriormente y que permiten: la manipulación de bits, la manipulación de interrupciones, generar leves retardos, etc. Estas instrucciones son listadas en la Tabla 3.4.6.1.

Tabla 3.4.6.1 Instrucciones Misceláneas

Mnemónico	Función en Inglés	Función en Español
BCLR	Clear Bits in Memory	Ceros en Memoria
BSET	Set Bits in Memory	Unos en Memoria
CLC	Clear Carry bit	Cero en bit de Cargo
CLI	Clear Interrup Mask bit	Cero en bit de Interrupción
CLV	Clear Overflow bit	Cero en bit de Sobreflujo
SEC	Set Carry bit	Uno en bit de Cargo
SEI	Set Interrupt Mask bit	Uno en bit de Interrupción
NOP	No Operation	No Operación
RTI	Return from Interrupt	Retorno de Interrupción
S	Stop	Alto
SWI	Software Interrupt	Interrupción por Software
TEST	Test	Prueba
WAI	Wait for Interrupt	Espera de Interrupción

CAPITULO 4

SISTEMA DE DESARROLLO DE SOFTWARE

4.1. INTRODUCCION.

El objetivo de este capítulo es describir algunos conceptos básicos implicados en el desarrollo de software (programas). Además, aprender el manejo de algunas herramientas que nos faciliten el transferir y depurar un programa para hacer que lo ejecute apropiadamente un MCU.

La **Programación** es la preparación de la lista de instrucciones que una computadora, microprocesador o microcontrolador debe ejecutar. La programación es un trabajo de diseño que requiere un alto grado de creatividad. Cuando un programa funciona inapropiadamente generalmente se debe a que está mal elaborado y pocas veces se debe a fallas del equipo.

El **Software** o programa está constituido por la lista de instrucciones y se almacena temporal o permanentemente en la memoria de programa. Software es un término generalmente utilizado para denominar a todos los programas. **Firmware** es el nombre dado al software que es almacenado permanentemente en la memoria ROM.

El **Sistema de Desarrollo de Software** es *el conjunto* de técnicas y herramientas que el programador utiliza en el proceso de desarrollo de software y consta de:

- Técnicas de Programación.
- Lenguajes de Programación.
- Editor de Texto.
- Programa Ensamblador
- Programa Monitor.

En la figura 4.1 se muestra la secuencia en el Sistema de Desarrollo de Software. El procedimiento puede seguirse en la figura por medio de los números dentro de los círculos y los pasos que a continuación se detallan.

Paso 1.- Aplicando técnicas de programación se elabora el diagrama de flujo estableciendo las actividades que se desea realice el programa.

Paso 2.- Utilizando una PC que tenga instalado un Editor de Texto se teclea el listado de instrucciones para obtener el Programa Fuente (Programa.ASM) y se guarda en un disco *para usarlo después.*

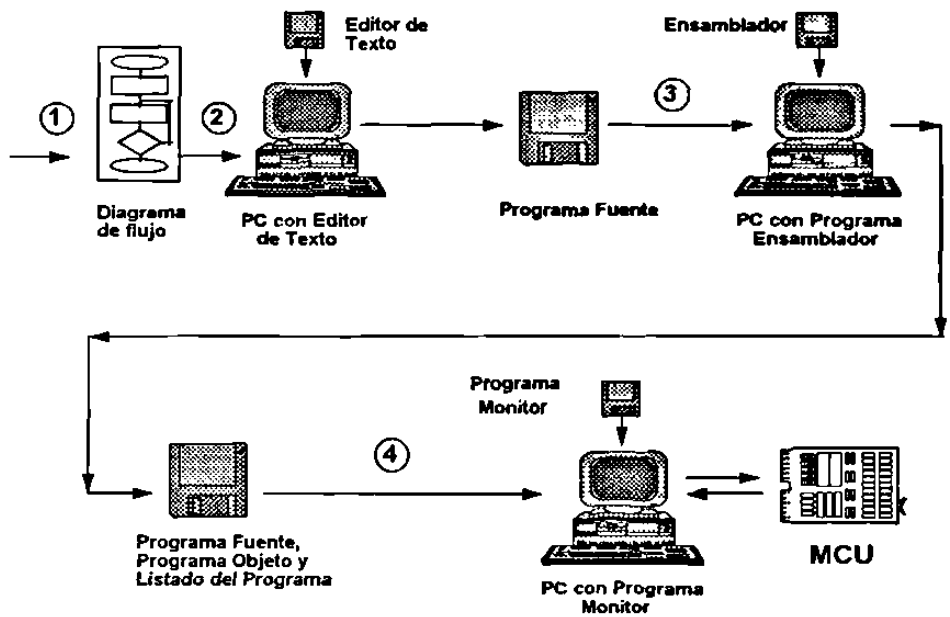


Figura 4.1. Secuencia en el Desarrollo de Software.

Paso 3.- Utilizando una PC que tenga instalado el Programa Ensamblador, se transforma o ensambla el Programa Fuente para obtener: El Programa Objeto (Programa.S19) y el Listado del Programa (Programa.LST)

Paso 4.- El Programa Monitor instalado en la PC permite depurar y corregir el Software desarrollado hasta obtener el comportamiento adecuado. El Programa Monitor realiza las siguientes funciones:

- Carga el Programa Objeto en el *MCU*
- Corre el Programa Objeto
- Lee y modifica memorias y registros internos del MCU

4.2. TECNICAS DE PROGRAMACION.

La primer pregunta que hay que responder antes de hacer cualquier esfuerzo de programación es: ¿Qué se quiere que haga el programa? La falta de responder esta pregunta con propiedad y eficiencia es con frecuencia uno de los obstáculos más graves y equivale a iniciar un viaje sin conocer el destino. ¡Este es un error muy común que se comete en el desarrollo de software¡ Después de decidir lo que hará el programa, se puede continuar con el proceso de desarrollo del software.

La **Programación Estructurada** es la técnica de programación más utilizada, mediante un diagrama muy parecido a la estructura jerárquica de una corporación, se establece en forma descendente como se desea que el programa realice su trabajo. El organizar y codificar programas en forma estructurada permite reducir su complejidad, mejora la claridad y facilita su depuración y modificación.

El **Diagrama de Flujo** es una representación gráfica que ilustra los pasos lógicos, cálculos y decisiones que en secuencia deben ser realizados para cumplir con una tarea específica. Antes de empezar a escribir un programa, es práctica común generar un Diagrama de Flujo de las actividades que se desea que realice el programa. Después de generar un diagrama de flujo, la escritura del programa es relativamente simple. En la figura 4.2. aparecen los símbolos comunes de un diagrama de flujo.

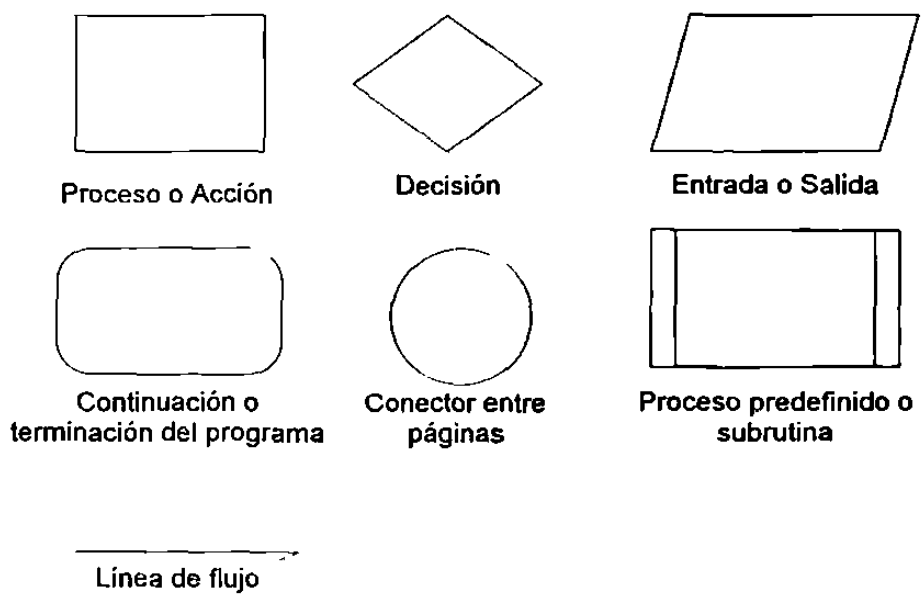


Fig. 4.2 . Símbolos comunes de un diagrama de flujo

4.3 LENGUAJES DE PROGRAMACION

En la actualidad existen diferentes tipos de lenguajes que son utilizados por el programador para realizar software, pero al aumentar el nivel de complejidad del lenguaje utilizado, los detalles de la estructura interna del microprocesador o microcontrolador se escapan a la vista del usuario. Los Lenguajes de Programación pueden ser clasificados como:

- Lenguaje de Máquina (de bajo nivel)
- Lenguaje Ensamblador (de nivel medio)
- Lenguaje de Alto Nivel

El **Lenguaje de Máquina** es un lenguaje binario pues está constituido exclusivamente por unos y ceros, y es el único lenguaje que reconocen los MCU's.

Un Programa en Lenguaje de Máquina contiene los códigos binarios (normalmente expresados en notación hexadecimal) que representan las instrucciones y datos de un MCU. **Programa Objeto** es el nombre dado a un programa ya expresado en lenguaje de máquina.

El **Lenguaje Ensamblador** es una colección de símbolos mnemónicos que representan: operaciones (mnemónicos de instrucciones del MCU y directivas para el ensamblador), etiquetas, operadores y símbolos especiales.

Un **Programa Fuente o Programa en Lenguaje Ensamblador** es el texto donde los mnemónicos (ayudas de memorización) del fabricante son utilizados para representar las instrucciones del MCU. Programa Fuente es otro nombre dado a un Programa en Lenguaje Ensamblador y no es ejecutable por el MCU.

Los **Lenguajes de Alto Nivel** utilizan proposiciones en inglés que representan desde unas cuantas, hasta muchas instrucciones equivalentes de lenguaje ensamblador o lenguaje de máquina. Algunos lenguajes de alto nivel son: BASIC, FORTRAN, Pascal, C, COBOL, etc. El nombre de Programa Fuente también se aplica al programa escrito mediante las instrucciones de los lenguajes de alto nivel.

4.4. EDITOR DE TEXTO.

Un **Editor de Texto** es un programa de computadora que es utilizado para escribir el Programa Fuente. Un Programa en Lenguaje Ensamblador es el texto de una Programa Fuente. Para que el Programa Fuente sea útil y legible

al MCU, debe ser transformado en Programa Objeto mediante un Programa Ensamblador.

El formato o sintaxis utilizado al escribir el programa fuente mediante el Editor de Texto, debe corresponder al utilizado por el Programa Ensamblador.

El Editor de Texto es un programa residente en computadora que permite usarla como una máquina de escribir eléctrica. En lugar de un Editor de Texto puede ser utilizado un Procesador de Palabras que produzca la salida de texto ASCII, también conocida como salida no-documentada. El Modo no-documentado produce un archivo sin los caracteres de control no imprimibles que son usados en documentos formateados, los cuales pueden producir errores al ensamblar el programa.

Debido a las diferencias de cada Editor de Textos, el usuario deberá elegir el de su preferencia para aprender a operarlo, pues aquí no será explicado ninguno de ellos por no ser parte del curso.

4.5. PROGRAMA ENSAMBLADOR.

El Ensamblador es un programa que procesa el Programa Fuente (escrito en lenguaje ensamblador) y lo traduce en Programa Objeto (escrito en lenguaje de máquina) el cual es ejecutable por el MCU, además produce un listado del Programa.

Motorola proporciona como herramienta un Ensamblador de 8 bit de dos pasos llamado AS11.EXE, el cual puede ser utilizado en computadoras IBM-PC o compatibles y produce el Programa Objeto para los MCU's de la familia M68HC11, existen otros ensambladores diferentes pero aquí solo veremos el AS11.EXE.

4.5.1. FORMATO O SINTAXIS DE LAS LINEAS FUENTE DEL PROGRAMA.

El Programa Fuente consiste de una secuencia de líneas o estatutos. En cada línea solamente es permitida una instrucción o directiva de ensamblado. Cada línea de ensamblado termina al oprimir la tecla ENTER. El formato o sintaxis de las líneas puede incluir algunos o todos los cuatro campos siguientes:

Etiqueta **Operación** **Operando(s)** **Comentarios**

4.5.1.1. El Campo de la Etiqueta. El primer campo de una línea o estatuto es el Campo de la Etiqueta y tiene las siguientes características:

1. Si no se requiere ninguna etiqueta en el estatuto, entonces el primer carácter debe ser un espacio en blanco, lo cuál indica que el campo de la etiqueta esta vacío.

2. Si el primer carácter en el campo de la etiqueta es un asterisco (*), entonces el resto del estatuto es un comentario. Los comentarios son ignorados por el Ensamblador y son impresos en el listado del programa solamente para información.

3. Los símbolos o caracteres utilizados para escribir etiquetas son:
   - Las letras mayúsculas y minúsculas del alfabeto, las cuales son consideradas distintas.
   - Los dígitos del 0 al 9.
   - Los caracteres especiales: Punto (.), signo de dólar ($) y subrayado (_).

4. Las etiquetas consisten de 1 a 16 caracteres y el primero debe ser alfabético, punto (.) o subrayado (_).

5. Las etiquetas no deben ser repetidas en otras líneas o un error será indicado.

6. Opcionalmente la etiqueta puede terminar con dos puntos (:), pero estos no forman parte de la etiqueta y solo sirven para separarla del resto de la línea. Los siguientes fragmentos de programa son equivalentes:

```
AQUI: DECA
      BNE AQUI

AQUI  DECA
      BNE AQUI
```

4.5.1.2. El Campo de Operación se encuentra enseguida del Campo de la Etiqueta y tiene las siguientes características:

1. Debe ser precedido al menos por un espacio en blanco.

2. El Campo de Operación debe contener el mnemónico de una instrucción del MCU o alguna Directiva del Ensamblador (las directivas son explicadas más adelante).

3. Las letras mayúsculas son convertidas a letras minúsculas antes de ser checadas como operaciones válidas, por lo tanto "NOP", "NoP" y "nop" son reconocidos como el mismo mnemónico.

4.5.1.3. El Campo del Operando es utilizado para establecer el operando de una instrucción o para especificar el modo de direccionamiento de una instrucción. El Campo del Operando tiene las siguientes características:

1. Debe estar enseguida del Campo de Operación y debe ser precedido al menos por un espacio en blanco.

2. El Campo del Operando puede contener: una etiqueta, una expresión, o una combinación de etiquetas y expresiones separadas por comas (,).

3. El formato de los operandos utilizados por los MCU's de la familia M68HC11 es mostrado en la Tabla 4.5.1.3.

Tabla 4.5.1.3. Formato de los Operandos

Formato del Operando	**Modo de Direccionamiento**
No operando	Inherente
expresión	Directo, Extendido, Relativo
# expresión,	Inmediato
expresión, X	Indexado X
expresión, Y	Indexado Y
expresión 1 expresión 2	Bits en 1's o O's
expresión 1 expresión 2 expresión 3	Prueba de bits o Salto Condicional

Notas: expresión1 expresa direccionamiento directo o indexado.
expresión2 establece el byte de máscara.
expresión3 genera una dirección relativa.

4.5.1.3.1. Expresiones. Una expresión es una combinación de: operadores (algebraicos o lógicos), etiquetas y constantes. Las expresiones son evaluadas de izquierda a derecha y no esta previsto el uso de paréntesis. La aritmética es realizada con signo en complemento a 2 y con precisión entera de 16 bits.

Los operadores algebraicos y lógicos utilizados en las expresiones, así como la función que representan, son mostrados enseguida:

+	suma
-	resta
x	multiplicación
/	división
%	residuo después de la división
&	AND
I	OR
^	OR EXCLUSIVE

Las **etiquetas** son asociadas con un valor entero de 16 bit, el cual es usado en lugar de la etiqueta durante la evaluación de la expresión.

Las **constantes** representan cantidades que no varían en valor durante le ejecución del programa. Las constantes pueden ser representadas en alguno de los siguientes formatos: decimal, hexadecimal, binario, octal o ASCII.

Los *prefijos* utilizados antes de cada número para indicar su formato son:

$	Hexadecimal
%	Binario
@	Octal
'	ASCII

El Ensamblador interpreta las constantes sin prefijo como decimales. Además, convierte todas las constantes a código binario y las muestra en el listado en forma hexadecimal. Las constantes debe caer en el rango decimal de 0 a 65535 inclusive o su equivalente en cualquier otro sistema numérico.

Ejemplos de constantes decimales:

válidos	inválidos	razón de invalidez
12	123456	fuera de rango
12345	12.3	Carácter inválido

Ejemplos de constantes hexadecimales

válidos	inválidos	razón de invalidez
$12	ABCD	falta prefijo $
$ABCD	$G2A	carácter inválido
$001F	$2F018	fuera de rango

Ejemplos de constantes binarias

válidos	inválidos	razón de invalidez
%00101	1011	falta prefijo %
%1	%210101	dígito inválido

Ejemplos de constantes octales

válidos	inválidos	razón de invalidez
@17634	@2317234	fuera de rango
@377	@23914	dígito inválido

Ejemplos de constantes ASCII

valido	inválido	razón de invalidez
'*	'VALID	demasiado largo

4.5.1.4. El Campo de Comentarios es el último campo de un estatuto, este campo es opcional, y es impreso solamente en el listado del programa para propósitos de información. El Campo de Comentarios debe ser separado por al menos un espacio en blanco del campo que le precede, además puede contener cualquier carácter ASCII imprimible.

4.5.2. DIRECTIVAS DEL ENSAMBLADOR.

Las Directivas controlan el funcionamiento del Ensamblador y no son transferidas al Programa Objeto. La notación utilizada para describir las Directivas es la siguiente:

XYZ Las letras mayúsculas realzadas se usan para indicar el mnemónico de las directivas.

( ) Los paréntesis denotan un elemento opcional.

< > Los paréntesis angulares se usan para nombrar elementos y son escritos con letras minúsculas. Todos los elementos fuera de estos paréntesis '< >' deben ser especificados como están.

Por ejemplo: (<número>,), número es un elemento opcional y se requiere especificar la coma.

Lo que significan los siguientes elementos es mostrado a continuación:

<comentario>	Estatuto del Campo de Comentario.
<etiqueta>	Etiqueta.
<expresión>	Expresión del Ensamblador.
<expr>	Expresión del Ensamblador.
<número>	Constante numérica.
<string>	Cadena de Caracteres ASCII.
<delimitador>	Delimitador de string.
<opción>	Opción del Ensamblador.

En la descripción de las directivas se sigue el siguiente orden:

- Mnemónico y Nombre de la Directiva en Inglés.
- Formato o sintaxis.
- Descripción de la directiva.

4.5.2.1. BSZ (Block Storage Zeros)

(<etiqueta>) BSZ <expresión> (<comentario>).

La directivas BSZ y ZMB causan que el Ensamblador distribuya un bloque de bytes, y a cada byte le es asignado el valor inicial de cero. El número de bytes es especificado por el valor de la expresión en el campo del operando. Un error será indicado si la expresión tiene un valor de cero y sí contiene etiquetas indefinidas más adelante.

4.5.2.2. EQU (Equate label to a value)

<etiqueta> EQU <expresión> (<comentario>).

La directiva EQU asigna a la etiqueta el valor de la expresión del Campo del Operando. Las etiquetas no deben ser redefinidas. Las expresiones con referencias adelante serán indicadas como error de fase.

4.5.2.3. FCB (Form Constant Byte)

(<etiqueta> FCB <expr> (,<expr>,...,<expr>) (<comentario>)

La directiva FCB puede tener uno o más operandos separados por comas. El valor de cada operando es truncado a 8 bits y es almacenado en un solo byte del Programa Objeto. Los caracteres múltiples son almacenado en bytes sucesivos. Los operandos pueden ser constantes numéricas, constantes de carácter, etiquetas o una expresión.

Ejemplo:

```
TABLA1  FCB  $00,$30,$5F.
```

4.5.2.4. FCC (Form Constant Character String)

(<etiqueta>) FCC <delimitador><string><delimitador>(<comentario>).

La directiva FCC es usada para almacenar una cadena de caracteres ASCII en bytes consecutivos de memoria. El almacenaje de bytes empieza en el valor actual del Contador de Programa y a la etiqueta se le asigna el valor del primer byte. Cualquier carácter ASCII imprimible puede formar parte de la cadena de caracteres (string). Como delimitadores se pueden usar dos caracteres idénticos ASCII.

Ejemplo:

```
Etiqueta1  FCC  ,ABC,
```

4.5.2.5. FDB (Form Double Byte Constant)

(<etiqueta>) FDB <expr>(,<expr>,...,<expr>) (<comentario>).

La directiva FDB puede tener uno o más operandos separados por comas. Un valor de 16 bits correspondiente a cada operando es almacenado en dos bytes consecutivos del Programa Objeto. El almacenaje empieza en el valor actual del Contador de Programa, y a la etiqueta se le asigna el primer valor de 16 bits. Los operandos múltiples son almacenados en bytes sucesivos. Los operandos pueden ser: constantes numéricas, constantes de carácter, etiquetas, o una expresión.

Ejemplo:

TABLA2 FDB $1004,$1030,$10FD.

4.5.2.6. FILL (Fill memory)

(<etiqueta>) FILL <expr>, <expr>.

La directiva FILL causa que el ensamblador inicialice un área de memoria con un valor constante. La primera expresión significa el valor de un byte que será colocado en memoria, y debe estar en el rango de 0 a 255. La segunda expresión indica el total de bytes que serán inicializados. Las expresiones no pueden contener etiquetas indefinidas o referenciadas más adelante en el programa.

4.5.2.7. OPT (Assembler Output Options).

OPT <opción>(,<opción>,...,<opción>) (<comentario>).

La directiva OPT es usada para controlar el formato de la salida del Ensamblador. Las opciones especificadas en el Campo del Operando son separadas por comas. Todas las opciones tienen una condición por omisión (default). Algunas opciones pueden ser inicializadas en la línea de comando que se utiliza para correr el Ensamblador, sin embargo las opciones contenidas en el Programa Fuente toman prioridad. En las siguientes descripciones aparece entre paréntesis (DEFAULT) si la opción es una condición por omisión. Todas las opciones deben ser escritas con letra minúscula.

c Habilita la cuenta de ciclos en el listado.

cre Imprime una tabla de referencia de cruce al final del listado. Esta opción debe ser especificada antes de la primera etiqueta utilizada en el programa Fuente.

l Imprime el listado desde el punto en que aparece.

noc (DEFAULT) Deshabilita el conteo de ciclos desde este punto en el listado.

nol (DEFAULT) No imprime el listado desde el punto en que aparece.

s Imprime la tabla de etiquetas al final del listado del Programa Fuente.

4.5.2.8. ORG (Set Program Counter to Origin).

ORG <expresión> (<comentario).

La directiva ORG cambia el valor del Contador de Programa al especificado en el Campo del Operando. Los estatutos que se encuentran después de la directiva ORG son ensamblados en localidades de memoria que empiezan en el nuevo valor del Contador de Programa. El Contador de Programa se inicializa en cero si no se encuentra la directiva ORG. La expresión no puede contener etiquetas indefinidas ni referencias hacia adelante.

4.5.2.9. PAGE (Top of Page).

PAGE

La directiva PAGE hace que el Ensamblador avance el papel hasta la parte superior de la siguiente página. Si el listado del programa no es producido, la directiva PAGE no tiene efecto. Esta directiva no es impresa en el listado del programa.

4.5.2.10. RMB (Reserve Memory Byte).

(<etiqueta>) RMB <expresión> (<comentario>).

La directiva RMB es utilizada para reservar espacio de memoria para etiquetas y es colocada después de la directiva ORG para que el Ensamblador asigne el espacio de memoria en la dirección especificada por la directiva ORG.

Ejemplo:

```
ORG $100
TEMP RMB 3
```

Descripción: la localidad 100 se *reserva para* la etiqueta TEMP, la localidad 101 se reserva para la etiqueta TEMP1 y la localidad 102 para la etiqueta TEMP2.

4.5.2.11. ZMB (Zero Memory Bytes).

(<etiqueta>) ZMB <expresión> (<comentario>).

La directiva ZMB es idéntica a la directiva BSZ.

4.5.3. INVOCACION DEL ENSAMBLADOR.

Para correr el Programa Ensamblador teclee en la computadora la siguiente línea de comando:

AS11 *FUENTE1* (FUENTE2...) (-OPCION1 OPCION2...) >FUENTE1.LST

- **FUENTE1.ASM, FUENTE2.ASM**, etc. son los nombres de los programas fuente que se desean ensamblar. Convencionalmente utilizaremos la extensión .ASM para los nombres de los programas fuente

- Los elementos entre paréntesis "( )" son opcionales. Use un espacio en blanco antes del signo menos para separar las opciones del último nombre del programa fuente.

- Las **OPCIONES** son una o más de las siguientes:

l *Habilita el listado del programa.*

nol Deshabilita el listado del programa (default).

cre Habilita la creación de una tabla de referencia cruzada.

s Habilita la generación de la tabla de etiquetas.

c Habilita la cuenta de ciclos.

noc Deshabilita la cuenta de ciclos (default)

- El **Listado del Programa** o los mensajes de error pueden ser salvados como un archivo para su examen posterior. El Listado del Programa se obtiene agregando el comando de redireccionamiento indicado por el símbolo > (mayor que), seguido por el nombre del programa. Por costumbre se le coloca la extensión .LST al listado del programa.

El Programa Objeto creado, es salvado automáticamente con la extensión .S19, y se le da el nombre del primer programa fuente en la línea de comando. Si el Programa Objeto no es creado, entonces debe consultar el Listado del Programa para verificar que errores de ensamblado se produjeron. Si hay algún error aparece un mensaje explicativo. Los errores marcados Warning no cancelan el ensamblado pero son indicativos de un posible problema.

El Listado del Programa tiene el siguiente formato:

#LINEA DIR BYTES CODIGO OBJ. [#CICLOS] LINEA FUENTE

#LINEA es un número decimal de 4 dígitos usado como referencia en la referencia cruzada.

DIR es el valor hexadecimal de la dirección para el primer byte del código objeto de ésta instrucción.

BYTES CODIGO OBJ. son el código objeto de la línea fuente en hexadecimal. Si una línea fuente causa más de 6 bytes, estos son listados en líneas sucesivas sin que les preceda ninguna dirección.

#CICLOS aparece solo si está la opción "c". Los ciclos aparecen entre paréntesis rectangulares para distinguirlos de listado fuente.

LINEA FUENTE es reimpresa como en el Programa Fuente..

La Tabla de Etiquetas tiene el siguiente formato:

ETIQUETA DIR

La **ETIQUETA** es tomada directamente del campo de la etiqueta.

DIR es la dirección hexadecimal de la localidad que corresponde a la etiqueta.

La Tabla de Referencia Cruzada es impresa cuando es seleccionada la opción "cre", y tiene el siguiente formato:

ETIQUETA DIR *LOC1 LOC2 LOC3...

La ETIQUETA y DIR tienen el mismo significado que fue descrito antes. La primer localidad LOC1 en el listado, es marcada con un asterisco para indicar la línea donde la etiqueta fue definida. Las siguientes localidades LOC2 LOC3... son el número de la línea decimal del listado donde ocurre la etiqueta.

4.5.4. EJERCICIO DE ESCRITURA Y ENSAMBLADO DE UN PROGRAMA.

Mediante el Editor de Texto de su preferencia copie la escritura en letra realzada del siguiente programa que llamaremos SUMAYCCR.ASM. La escritura es realizada aplicando las reglas de sintaxis del ensamblador AS11 y se han hecho las siguientes consideraciones:

1. Se han separado ocho columnas para el Campo de la Etiqueta.

2. Se deja al menos un espacio en blanco de separación entre la etiqueta y la instrucción o directiva.

3. Se han separado ocho columnas para el Campo de Operación (instrucción o directiva).

4. Se deja al menos un espacio en blanco de separación entre la instrucción o directiva y el operando.

5. Se han separado ocho columnas para el Campo del Operando.

6. El Campo de Comentarios no es utilizado.

ETIQUETA		OPERACION		OPERANDO

*PROGRAMA QUE SUMA DOS NÚMEROS Y				
*MUESTRA EL EFECTO EN EL CCR				
		ORG		$40
		LDS		$9F
PUERTOB		EQU		$1004
REPITE		LDAA		#$18
		LDAB		#$13
		ABA		
		TPA		
		STAA		PUERTOB
		JMP		REPITE
FIN		BRA		FIN

Guarde en un disco el programa con el nombre SUMAYCCR.ASM

A continuación es revisado y descrito línea por línea este programa de ejemplo:

Línea 1 y línea 2.- *PROGRAMA QUE SUMA DOS NUMEROS Y *MUESTRA EL EFECTO EN EL CCR. Estas dos líneas empiezan con un asterisco en la columna 1 porque las líneas completas son comentarios.

Línea 3.- ORG $40. El Campo de la Etiqueta está vacío (en total 9 espacios en blanco). La Directiva Origen (ORG) indica que el programa empieza en la localidad $40.

Línea 4.- LDS $9F. El Campo de la Etiqueta está vacío. La instrucción LDS establece el valor inicial del Stack Pointer en la localidad $9F.

Línea 5.- PUERTOB EQU $1004. La etiqueta PUERTOB empieza en la columna 1 y es separada por 2 espacios en blanco de la directiva EQU que está en el Campo de Operación. La directiva EQU hace que la etiqueta PUERTOB sea igual; al valor $1004.

Línea 6.- REPITE LDAA #$18. La etiqueta REPITE empieza en la columna 1 y se utiliza para formar un ciclo en el programa. La etiqueta es separada por medio de 2 espacios en blanco de la instrucción LDAA. La instrucción LDAA es utilizada para cargar el acumulador A en forma inmediata con el número $18.

Línea 7.- LDAB #$13. El Campo de la Etiqueta está vacío. La instrucción LDAB permite cargar el acumulador B en forma inmediata con la cantidad $13.

Línea 8.- ABA. La instrucción ABA no requiere operando y es utilizada para sumar los contenidos de los acumuladores A y B. El resultado es colocado en el acumulador A y el registro CCR cambia debido a la operación de suma.

Línea 9.- TPA. Esta instrucción realiza la transferencia del contenido del registro CCR al acumulador A.

Línea 10.- STAA PUERTOB. La instrucción STAA envía el contenido del acumulador A al PUERTOB de salida.

Línea 11.- JMP REPITE. La instrucción JMP (jump) se utiliza para formar un ciclo al realizar un salto hacia donde se encuentra la etiqueta REPITE.

Línea 12.- FIN BRA FIN. Es conveniente utilizar ésta línea al final de un programa cuando se utiliza el Programa Monitor PCbug11.

Mediante el Programa Ensamblador AS11 ensamble el programa de ejemplo SUMAYCCR.ASM, utilice la siguiente línea de comando en la computadora:

AS11 SUMAYCCR.ASM -L S C CRE >SUMAYCCR.LST

Consulte el listado del programa y si existe algún error de ensamblado corríjalo hasta que no aparezcan errores.

A continuación se muestra como aparece el listado del programa después de ejecutar la línea de comando anterior:

```
0001                                        *PROGRAMA QUE SUMA DOS NUMEROS Y
0002                                        *MUESTRA EL EFECTO EN EL CCR
0003   0040                                        ORG      $40
0004   0040   9e   9f                  [ 4 ]    LDS      $9F
0005   1004                                     PUERTOB   EQU      $1004
0006   0042   86   18                  [ 2 ]    REPITE    LDAA     #$18
0007   0044   c6   13                  [ 2 ]              LDAB     #$13
0008   0046   1b                       [ 2 ]    ABA
0009   0047   07                       [ 2 ]    TPA
0010   0048   b7   10   04             [ 4 ]    STAA     PUERTOB
0011   004b   7e   00   42             [ 3 ]    JMP      REPITE
0012   004e   20   fe                  [ 3 ]    FIN      BRA      FIN
0013                                                               BRA      FIN

FIN        004e   *0012
PUERTOB
REPITE     0042   *0006
```

4.6 PROGRAMA MONITOR.

4.6.1. INTRODUCCION.

El Programa Monitor es colocado como herramienta final del Sistema de Desarrollo de Software, porque solamente puede ser utilizado hasta después de que el Programa Fuente escrito con un Editor de Texto, ha sido trasladado a Programa Objeto mediante un Programa Ensamblador. El Programa Monitor es utilizado para modificar y verificar que el software desarrollado por el usuario realiza la función esperada.

El **Programa Monitor PCbug11 de Motorola** es un paquete de software que permite correr los programas del usuario en cualquier MCU de la familia M68HC11, además facilita el examen de los periféricos internos bajo condiciones específicas, para la depuración de los programas hasta obtener el comportamiento deseado.

4.6.2. COMO TRABAJA EL PCBUG11.

El Programa Monitor PCbug11 es un paquete de software que permite las siguientes funciones:

- Realiza la interacción entre el MCU y la PC por medio de un programa de comunicación de bajo nivel que se instala en el MCU.

- Realiza la función de Monitoreo es decir: correr programas, leer y modificar memorias y registros. Y lo hace mediante un programa simple instalado en la PC que aprovecha las ventajas del hardware que sirve de soporte al MCU y lo sofisticado de la microcomputadora.

Existen dos métodos generales para usar los programas de comunicación que se instalan en el MCU:

1. El Método Cargador de RAM interna.

2. El Método Cargador de ROM.

4.6.2.1. Método Cargador de RAM Interna

En el **Método Cargador de RAM interna** el programa de comunicación es instalado en la RAM interna del MCU cada vez que se corre el PCbug11. Esto se realiza operando el MCU en el Modo de Operación Special Bootstrap

(El hardware de la figura 4.6.4.1 establece este modo de operación al conectar las terminales MODA y MODB a tierra). En este modo de operación, cada vez que se resetea el MCU se corre un programa instalado permanentemente en la ROM interna en las localidades $BF40 a $BFFF. Este programa establece la comunicación SCI (Serial Communications Interface) y hace posible que el PCbug11 instale el programa de comunicación en la RAM interna a partir de la localidad $00 y el programa de comunicación asume el control. Con el programa de comunicación instalado en la RAM es posible instalar programas en las memorias RAM, EEPROM y EPROM.

4.6.2.2. Método Cargador en ROM

En el Método Cargador en ROM, el programa de comunicación se instala en algún tipo de memoria ROM interna o externa. Para el MC68HC11A8 se dispone de memoria RAM desde la localidad $00 hasta $FF y memoria EEPROM desde $B600 hasta $B7FF. Debido a este limitado espacio de memoria, se recomienda para el desarrollo de prácticas instalar el programa de comunicación en la EEPROM, y éste será el método descrito. El MCU opera en el Modo de Operación Special Bootstrap pero en lugar de cargar un programa de comunicación que empiece en la localidad $00, al correr el PCbug11 con la **opción "programa de comunicación ya instalado"**, automáticamente se hace un salto a la localidad $B600 que es donde se inicia el programa de comunicación que previamente debe ser instalado en la EEPROM.

Más adelante en la sección 4.6.8. describe el procedimiento para instalar en la EEPROM el programa de comunicación TALKEREE.S19. Este programa se instala en las localidades de $B600 a $B6FF de la EEPROM, y utiliza parte de la RAM para instalar los vectores de interrupción en las localidades de $AO a $FE, además de establecer por default el Stack Pointer en la localidad $3F, dejando como AREA del Stack de $00 a $3F. Los programas del usuario deben respetar el uso de estas áreas para evitar fallas de comunicación. Vea el contenido del archivo TALKEREE.MAP para checar esas condiciones.

Estando instalado el Programa de Comunicación TALKEREE.S19 en la EEPROM, entonces el usuario dispone desde la localidad $40 hasta $9F en la RAM para poder correr, verificar y depurar sus programas hasta obtener el comportamiento deseado. Recuerde que los programas y datos almacenados en la RAM se pierden al desenergizar el MCU y que la EEPROM se utiliza para instalar programas en forma permanente (o casi permanente) y que no se pierden al desenergizar el MCU.

4.6.3. COMANDOS DEL PCBUG11.

A continuación se describen brevemente los comandos de PCbug11, deberá referirse al Manual de Usuario del PCbug11 para una explicación más detallada de los comandos. En la Tabla 4.6.3. los términos entre paréntesis "( )" son opcionales y las teclas especiales se representan entre paréntesis angulares "< >".

TABLA 4.6.3. Comandos del PCbug11.

Comando	Descripción
ASM addr (mneldir)	Ensamblador de líneas con opción de insertar mnemónicos o directivas.
BAUD (rate).	Muestra o fija el Baud rate (velocidad de comunicación).
BF addr1(addr2) byte/word	Llena un bloque de memoria con byte o palabra.
BL	Lista los puntos de quiebre (breakpoints).
BR (addr (macroname)).	Muestra o establece puntos de quiebre (con opcional ejecución de macros).
CALL addr.	Ejecuta la subrutina en la dirección addr.
CLRM	Elimina todos los macros en el MCU.
CLS	Limpia la ventana principal.
CONTROL (parámetros).	Muestra o cambia los parámetros del sistema PCbug11.
DASM addr1 (addr2).	Desensambla desde la dirección addr1 (hasta la dirección addr2).
DB startaddr (endaddr).	Muestra la Memoria del MCU desde la dirección inicial startaddr (hasta la dirección final endaddr).
DEBUG	Palabra reservada.
DEFINE symbol value/address.	Define el valor de un símbolo o etiqueta.
DEFM macroname/TRACE/ AUTOSTART	Define un comando, el macro de Trazo o el macro de AUTO arranque.
DELM macroname/TRACE/ AUTOSTART	Borra un comando, el macro Trazo o el macro de Auto arranque.
DIR (mask).	Muestra el directorio del disco o el especificado por mask.
DOS (command).	Llama al sistema operativo DOS o ejecuta un comando de DOS.
EDIT macroname.	Edita un macro.
EEPROM (startaddr (endaddr)).	Muestra, desactiva o fija un rango de direcciones al cual se aplica el algoritmo EEPROM para modificación o escritura de esta memoria.

TABLA 4.6.3. Comandos del PCbug11. Continuación...

Comando	Descripción
EEPROM DELAY option.	Fija el tiempo de borrado o programación en la EEPROM.
EEPROM ERASE (option)(addr).	Muestra o cambia la función de borrar antes de escribir en la EEPROM.
EPROM (startaddr (endaddr)).	Muestra, desactiva o fija un rango de direcciones al cual se aplica el algoritmo EPROM para modificación o escritura de esta memoria.
EPROM DELAY option.	Fija el tiempo de borrado o programación en la EPROM.
FIND byte/word addr1 addr2.	Encuentra las veces que ocurre un byte o palabra entre las direcciones addr1 y addr2.
FIND mnemonic addr1 addr2.	Encuentra las veces que ocurre un mnemónico entre las direcciones addr1 y addr2.
G (addr).	Ejecuta el programa del usuario que empieza en la dirección addr.
HELP (command).	Muestra información de ayuda.
KLE	Elimina el último mensaje de error.
LOADM (filename (macroname)).	Carga definiciones de macros del usuario o por omisión del archivo PCBUG11.MCR.
LOADS filename (loadaddr).	Carga en la memoria de MCU los Programas Objeto en Código Máquina.
LS symbol.	Lista los símbolos o etiquetas.
LSTM (Mname/TRACE/ AUTOSTART).	Lista los nombres o definiciones de macros.
MD staraddr (endaddr).	Muestra la memoria del MCU.
MM addr.	Modifica la memoria en la dirección addr.
MOVE addr1 addr2 addr3.	Mueve la memoria del MCU entre las direcciones addr1 y addr2 a la dirección addr3.
MS addr byte/word (byte/word).	Fija en la memoria bytes o palabras.
MSG (string).	Muestra mensajes en la ventana principal.
NOBR (address).	Remueve todos los puntos de quiebre o los especificados.
PAUSE (ms).	Espera que cualquier tecla sea oprimida o espera un tiempo de retardo.
PRINT.	Muestra el número de la versión del PCbug11.
PROTECT (startaddr (endaddr)).	Muestra, desactiva o activa un rango de direcciones protegido contra escritura.
QUIT (Y).	Termina la sesión del PCbug11 (sin requerir confirmación).

TABLA 4.6.3. Comandos del PCbug11. Continuación...

Comando	Descripción
RD (T).	Muestra el valor actual en la ventana de registro, con el parámetro T los actualiza en el modo de Trazo.
RESET (addr).	Resetea el hardware de MCU con nuevos o actuales vectores de reset.
RESTART (option).	Inicializa el PCbug11 con las mismas o nuevas opciones.
RM.	Modifica los Registros.
RS.	Fija valores de los registros.
S.	Detiene la ejecución del programa.
SAVEM (filame).	Salva los macros.
SHELL (command).	Ejecuta comandos del Sistema Operativo.
T (addr).	Este comando permite la ejecución de las instrucciones deteniéndose en cada una.
TERM (X1 Y1 X2 Y2).	Emulador de la terminal en las coordenadas X1,Y1,X2,Y2
TYPE filename.	Muestra el contenido de los archivos en la ventana principal.
UNDEF symbol.	Remueve símbolos o etiquetas.
VER.	Muestra el número de la versión del PCbug11.
VERF filename (memaddr).	Verifica el programa objeto en el disco contra la memoria.
VERF ERASE addr1 (addr2).	Verifica que la memoria este borrada (su contenido debe ser FF hex).

4.6.4. PREPARACION DEL MCU Y DEL PCBUG11.

Antes de usar el MCU con el paquete de software de PCbug11 debe hacer lo siguiente:

- Preparar el circuito (hardware) del MCU. El circuito debe suministrar un voltaje regulado de 5v al MCU y debe tener instalado un circuito de interfaz RS-232. En la fig. 4.6.4.1. se muestra un posible circuito.

- Conectar la interfase al puerto 1 ó 2 de una computadora IBM PC o compatible.

- Instalar el paquete de software del Programa Monitor PCbug11 en la computadora.

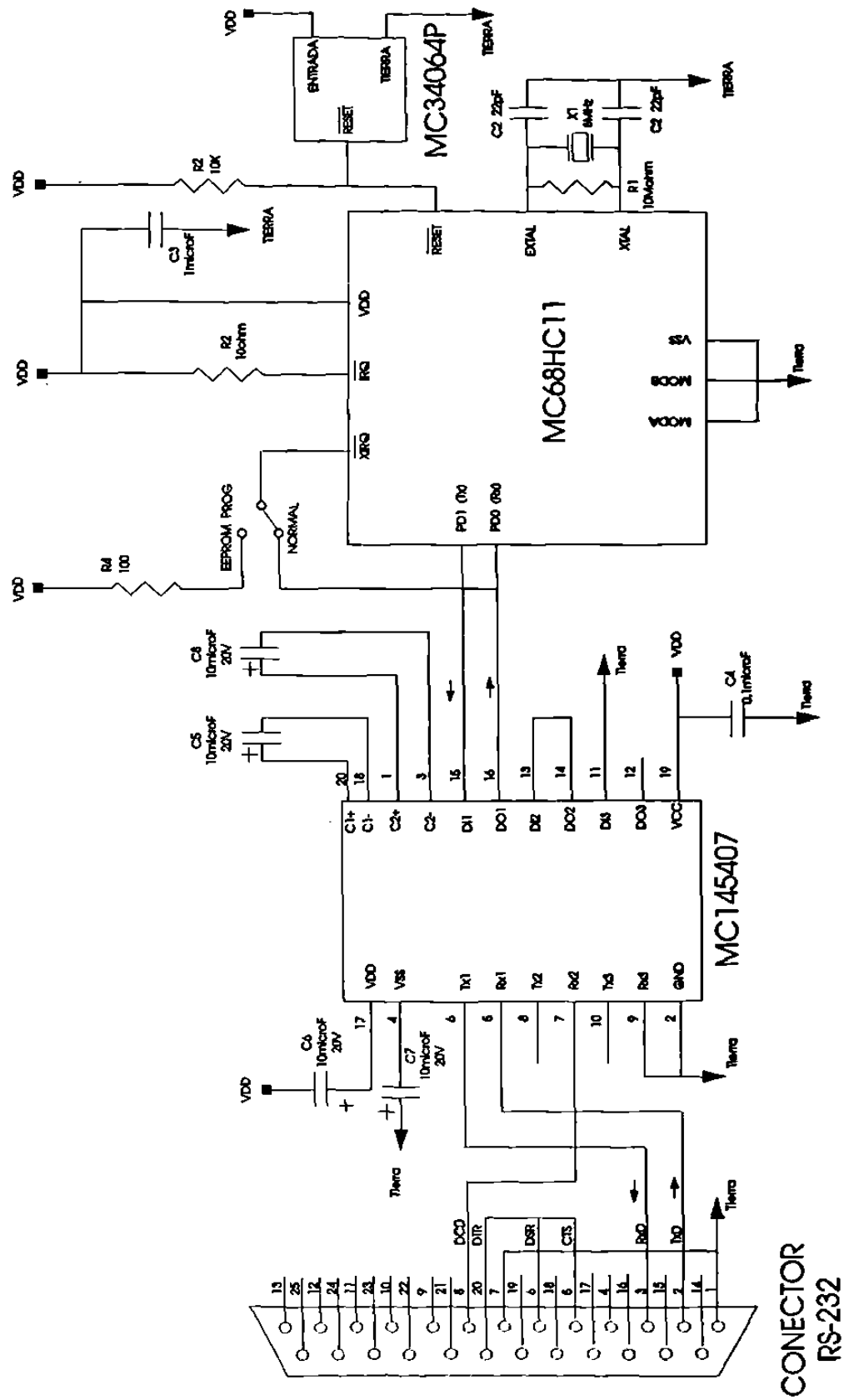


FIGURA 4.6.4.1. Diagrama del Hardware de Soporte del MC68HC11 para usarse con el Programa Monitor PCbug11.

4.6.5. INVOCACION DEL PCBUG11.

El PCbug11 es un paquete de software que hace posible muchas opciones: la elección del puerto utilizado, la elección de cristales de diferente frecuencia, el manejo de macros, la instalación de los programas del usuario en la memoria (RAM, EPROM o EEPROM) del MCU, etc.

En las descripciones siguientes: la *letra itálica* representa el texto de la computadora, la **letra negrita (Bold)** representa el texto que debe teclear como respuesta (seguido de la tecla ENTER en algunos casos) y el texto en letra normal precedido por una flecha, proporciona una explicación extra para el PCbug11 o el texto tecleado.

Para correr el PCbug11 en la computadora haga lo siguiente:

a. Realice la preparación del MCU y del PCbug11 indicada en el punto 6.4.

b. Encienda la computadora con el sistema operativo.

c. Coloque el disco con el paquete del software del PCbug11 en el drive A, B o C, aquí asumiremos que se instala en el drive B y que estamos en el drive C.

C:\> **B:**	⇦ se hace el cambio al drive B.
B:\> **PCBUG11**	⇦ se inicializa el PCbug11 y después de algunos segundos aparecen en la pantalla unos letreros solicitando respuesta a las opciones.

PCbug11 Ver 3.2.4a -M68HC11 Monitor for PC Hosts (c) Motorola Ltd 1991.

PCBUG11 Command Line Compiler.

Is the talker installed on your board? (Y/N) : ___	⇦ debido a que es la primera vez que usamos el PCbug11 supondremos que no hay instalado ningún talker previamente, teclee la letra **N.**
Do you wish to use the XIRQ interrupt? (Y/N) : ___	⇦ teclee la letra **N** (en el hardware del MCU deben estar conectadas juntas las terminales XIRQ y PD0).

MCU boot talkers available.

A 68HCIIK4	*B 68HC8IIA2*	*C 68HCIIE0*	*D 68HCIIE1*	*E 68HCIIE9*
F 68HC7IIE9	*G 68HCIIFI*	*H 68HCIIG5*	*I 68HCIIG7*	*J 68HCIIL6*
K 68HCIID3	*L 68HCIID0*	*M 68HCIIA0*	*N 68HCIIA1*	*O 68HCIIA8*
P 68HCIIE2	*Q 68HCIIA8*	*R 68HCIIE9*		

Wich device are you using?: ___

⇦ oprima la letra correspondiente al MCU que usa, no se preocupe si aparece otra letra en la pantalla, por ejemplo si oprimimos N aparece - A.

Do you wish to load a macro automatically? (Y/N): ___

⇦ por ser la primera vez que usamos el PCbug11 no debe haber ningún macro instalado, teclee la letra N.

PC communications part:

1. COM 1
2. COM 2

Wich communications port are you using?: __

⇦ teclee 1 ó 2 de acuerdo a su instalación y aparecerá en la pantalla port = 1 o port = 2 según corresponda.

Are you using an 8 MHz cristal? (Y/N): __

⇦ si la respuesta es N entonces preguntará la frecuencia en KHz del cristal utilizado en el Hardware. Supondremos que la frecuencia es 8 MHz, por lo tanto teclee la letra Y.

Command Line: **PCBUG11 - A**

⇦ Tecleando lo que aparece aquí subrayado despues del prompt *B:\>*, se puede correr el PCbug11 sin necesidad de teclear nuevamente todas las opciones: pero si cambia de MCU, de cristal, etc., entonces conteste las opciones. El botón de RESET del hardware del MCU debe ser oprimido antes de oprimir la tecla ENTER en la PC.

Press ESC to quit or any key to run PCBUG11

⇦ Antes de correr el PCbug11 debe oprimir el botón RESET del hardware de MCU. Presione cualquier tecla para correrlo (o la tecla ESC para salir).

Al correr el PCbug11 aparecen las siguientes ventanas en la pantalla.

```
PC bugII Ver 3.24a - M68HCII Monitor for PC Hosts (c) Motorola Ltd 1991.
                    Interal RAM Bootstrap Mode.

PC     ACCA ACCB X     Y     CCR (SXHINZVC SP      MCU     : 68HCIIA8
$0000  $40  $2C  $1000 $0100 $40  %.1......  $00EE State   : STOPPED
                                              RTS Level    :ON
                                              User RST  $XXXX
                                              User SWI  $XXXX
                                              User XIRG $XXXX

>>
```

4.6.6. Ventanas del PCbug11 en la Pantalla.

El estado del MCU y del hardware donde está soportado aparece en la pantalla de la computadora. Esta pantalla consiste de cuatro grandes áreas llamadas "Ventanas" que se describen enseguida:

Ventana Principal. Esta ventana está en la mitad superior de la pantalla, en ella aparece el texto en blanco fondo azul. En esta ventana aparece la mayor cantidad de información acerca de la operación del PCbug11, por ejemplo: el resultado de comandos, contenido de la memoria, ensamblado de códigos de operación (mnemónicos de instrucciones), macros, etc.

Ventana de Registros. Esta ventana se encuentra en el centro de la pantalla, con el texto en color blanco o amarillo sobre fondo rojo. En esta ventana se muestra el último contenido grabado en los registros del MCU. Tome nota que los valores en la Ventana de Registros son solamente los últimos datos ya sea después de arranque o cuando son requeridos por el usuario. Los comandos del PCbug11 le permiten modificar el contenido de los registros.

Ventana de Status se encuentra al centro a la derecha de la pantalla y en ella aparece el texto en color blanco con fondo morado. Esta ventana muestra: el tipo de MCU en uso, el estado del MCU (corriendo, detenido, trazo), el estado de la línea RS232 y el estado actual del conjunto de los vectores de interrupción del usuario.

Ventana de Comandos está en la parte inferior izquierda de la pantalla y en ella aparece el texto en color blanco sobre fondo negro. Use esta ventana para introducir y leer los comandos del PCbug11. El cursor de comandos es el carácter ">>" y aparece en la línea de abajo de esta ventana; los comandos que se introducen mediante el teclado, aparecen después del curso de comandos. Los comandos previos y el último mensaje de error también aparecen en la ventana de comandos.

Hay dos ventanas adicionales temporales las cuales aparecen sobrepuestas en la ventana principal.

Ventana de Error. Esta ventana indica cualquier error u operación de comunicación incorrecta con el MCU. El texto aparece en color rojo con el fondo negro. Para limpiar la ventana inmediatamente, presione cualquier tecla o espere algunos segundos y se limpiará sola.

Ventana de Ayudas. Esta ventana muestra información de ayuda requerida por medio del comando HELP, el texto aparece en color blanco sobre fondo negro. Para desplazarse a través de la información use las teclas: ⇧ (up-arrow); ⇩ (down arrow), page-up, page-down. Para limpiar la pantalla use la tecla ESC.

4.6.7. EDICION DE LA LINEA DE COMANDOS.

Para hacer uso del Programa Monitor PCbug11 se utiliza el teclado de la computadora para escribir y editar en la línea de comandos. En la tabla 4.6.7.1. aparecen las teclas de edición con una breve explicación, un buffer retiene los últimos 16 comandos introducidos.

Las cuatro líneas arriba de la línea de comandos sirven para hacer un rastreo de los últimos cuatro comandos introducidos. La quinta línea arriba de la línea de comando muestra los puntos de quiebre (break points) y los códigos de error:

0: No error
1: Error de Verificación
2: Error MS o BF
3: Falla de comunicación del talker.

TABLA 4.6.7.1. Teclas de Edición de la Línea de Comandos.

TECLA	FUNCION
⇦ "left arrow"	Mueve el cursor un carácter hacia atrás.
⇨ "right arrow"	Mueve el cursor un carácter hacia adelante.
Home	Mueve el cursor al primer carácter.
End	Mueve el cursor al último carácter.
Delete left	Borra el carácter a la izquierda del cursor.
Del	Borra en la posición del cursor.
<CTRL> End	Borra desde la posición del cursor hasta el final de la línea.
Ins	Inserta en la posición del cursor.
⇧ Up arrow	Llama al comando previo en orden inverso
⇩ Down arrow	Llama al último comando en el buffer.
ESC	Limpia la línea de comando o termina muchos comandos en progreso (ASM, DASM, MD, etc.

4.6.8. EJEMPLO DE COMO INSTALAR UN PROGRAMA EN LA EEPROM.

En este ejemplo se instalará el programa de comunicación TALKEREE.S19 en la EEPROM, y se verá el uso de algunos comandos seguidos de una breve explicación. En las descripciones siguientes: la *letra itálica* representa el texto de la computadora, la **letra negrita (Bold)** representa el texto que debe teclear como respuesta, seguido de la tecla ENTER en algunos casos. El texto en letra normal precedido por una flecha, proporciona una explicación extra para el PCbug11 o el texto tecleado.

Procedimiento para instalar un programa en la EEPROM:

- Realice todos los pasos de la sección 4.6.5. Invocación de PCbug11, hasta que aparezcan las ventanas.

- Introduzca los siguientes comandos.

>>**CONTROL BASE HEX** ⇦ cambia la base numérica por default que es 10, a base hexadecimal y hace innecesario el poner el carácter $ antes de escribir cada número.

>>**EEPROM B600 B6FF** ⇦ habilita la escritura en un rango de B600 a B6FF (256 bytes para el programa de comunicación en la EEPROM) y hace que el proceso *de escritura sea automático* (transparente al usuario).

>>EEPROM ERASE ENABLE ⇦ habilita la función borrar antes de escribir. Si bien, ésta es una función por omisión (default).

>>LOADS TALKEREE.S19 ⇦ carga el programa de comunicación TALKEREE.S19 en la EEPROM (cuando un programa ya tiene la extensión .S19, no es necesario teclear esta extensión).

>>VERF TALKEREE.S19 ⇦ verifica que el programa fue cargado sin errores.

>>QUIT Y ⇦ sale de PCBUG11 sin pedir confirmación.

Ahora ya esta instalado el programa de comunicación TALKEREE en la EEPROM y se ha dejado un espacio libre en la EEPROM (de B700 en adelante) para programas del usuario.

Para usar el talker de comunicación TALKEREE instalado en la EEPROM, teclee en la computadora.

B:\> **PCBUG11 TALKEREE** ⇦ automáticamente se realiza la comunicación por medio del TALKEREE instalado en la EEPROM.

CAPITULO 5

PUERTOS PARALELO

5.1. INTRODUCCION.

La interacción del microcontrolador con el mundo real, se realiza por medio de la terminales de entrada/salida, que también son conocidas como terminales **I/O** (Input/Output). Todas estas terminales realizan funciones múltiples, que dependen de los datos en los Registros de Control y del Modo de Operación, permitiendo que la comunicación puede ser realizada en las diferentes modalidades de Comunicación. En las figuras 5.1.1. y 5.1.2. se puede observar la ubicación de las terminales.

5.2. PUERTOS.

Un **Puerto** es un conjunto de terminales utilizado para entrada o salida de datos en un MCU. Los Registros de Control permiten configurar los Puertos para que realicen diversas funciones, eso permite una mayor versatilidad y flexibilidad en el uso de las terminales, dando una mayor eficiencia al MCU.

Los puertos en los microcontroladores de la familia M68HC11, están caracterizados por estar localizados en el mapa de memoria, y pueden ser direccionados de la misma forma que cualquier localidad de memoria. Los Puertos y los Registros de Control forman un bloque de 64 localidades, después de resetear el MCU en el Modo de Operación Cargador (Special Bootstrap), ocupan desde la dirección $1000 hasta $103F. En la tabla 5.2. se muestra la ubicación de los Puertos, Registros de Control y la asignación de los bits de control.

Las Terminales I/O se agrupan en:

a. Terminales de habilitación o señalización (Strobe A y Strobe B).
b. Puertos (A, B, C, D y E).

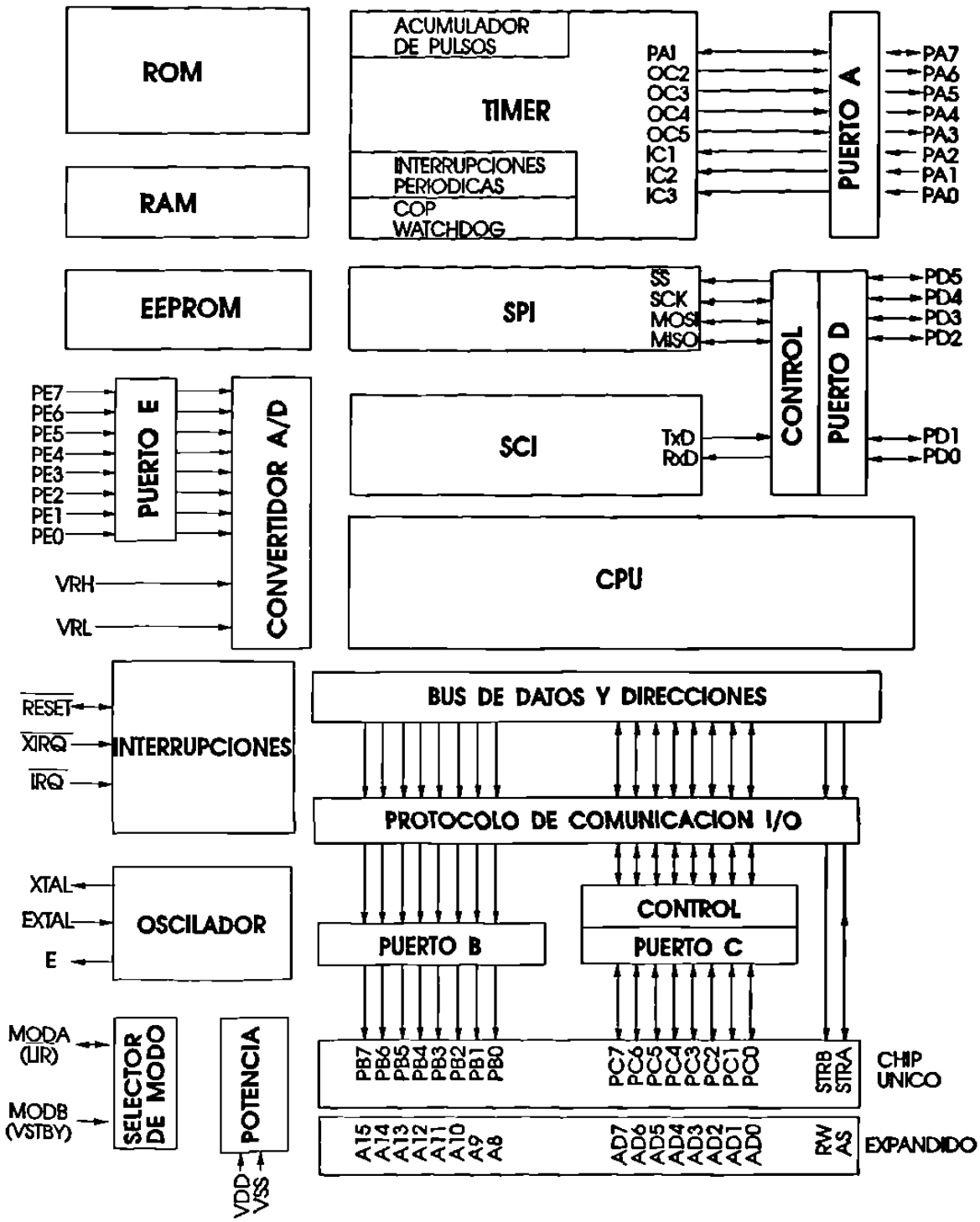
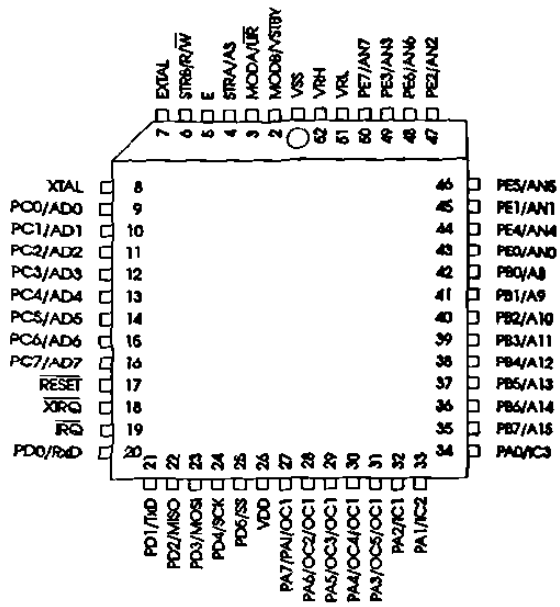


Figura 5.1.1. Diagrama de Bloques.



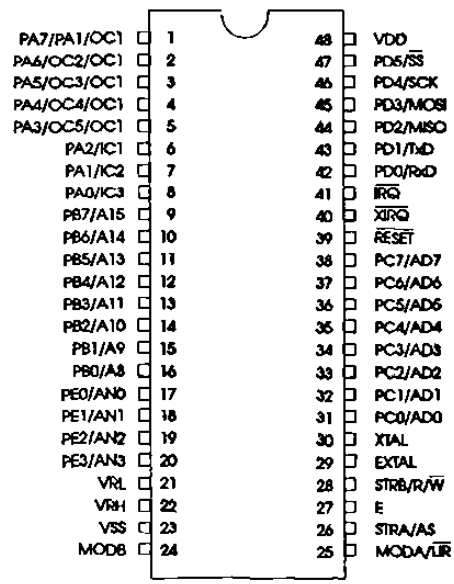


Figura5.1.2. Asignación de terminales MC68HC11A8

TABLA 5.2. BLOQUE DE REGISTROS DEL MC68HC11

localidad	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	Registro
$1000	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	PUERTO A
$1001									RESERVADO
$1002	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB	PIOC
$1003	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	PUERTO C
$1004	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	PUERTO B
$1005	PCL7	PLC6	PCL5	PCL4	PPCL3	PCL2	PCL1	PCL0	PUERTO CL
$1006									RESERVADO
$1007	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDCR
$1008	0	0	PD5	PD4	PD3	PD2	PD1	PD0	PUERTO D
$1009	0	0	DDR5	DDR4	DDR3	DDR2	DDR1	DDR0	DDRD
$100A	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0	PUERTO E
$100B	FOC1	FOC2	FOC3	FOC4	FOC5	0	0	0	CFORC
$100C	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3	0	0	0	OC1M
$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3	0	0	0	OC1D
$100E	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	TCNT(high)
$100F	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	TCNT(low)
$1010	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	TIC1(high)
$1011	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	TIC1(low)
$1012	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	TIC2(high)
$1013	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	TIC2(low)
$1014	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	TIC3(high)
$1015	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	TIC3(low)
$1016	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	TOC1(high)
$1017	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	TOC1(low)
$1018	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	TOC2(high)
$1019	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	TOC2(low)
$101A	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	TOC3(high)
$101B	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	TOC3(low)
$101C	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	TOC4(high)
$101D	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	TOC4(low)
$101E	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8	TI4O5(high)
$101F	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	TI4O5(low)
$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5	TCTL1
$1021	EDG4B	EDG4A	EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A	TCTL2
$1022	OC1I	OC2I	OC3I	OC4I	I4O5I	IC1I	IC2I	IC3I	TMSK1
$1023	OC1F	OC2F	OC3F	OC4F	I4O5F	IC1F	IC2F	IC3F	TFLG1
$1024	TOI	RTII	PAOVI	PAII	0	0	PR1	PR0	TMSK2
$1025	TOF	RTIF	PAOVF	PAIF	0	0	0	0	TFLG2
$1026	DDRA7	PAEN	PAMOD	PEDGE	DDRA3	I4/O5	RTR1	RTR0	PACTL
$1027	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	PACNT
$1028	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
$1029	SPIF	WCOL	0	MODF	0	0	0	0	SPSR

Continuación...

localidad	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	Registro
$102A	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	SPDR
$102B	TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0	BAUD
$102C	R8	T8	0	M	WAKE	0	0	0	SCCR1
$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCCR2
$102E	TDRE	TC	RDRF	IDLE	OR	NF	FE	0	SCSR
$102F	R7/T7	R6/T6	R5/T5	R4/T4	R3/T32	R2/T2	R1/T1	R0/T0	SCDR
$1030	CCF	0	SCAN	MULT	CD	CC	CB	CA	ADCTL
$1031	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	ADR1
$1032	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	ADR2
$1033	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	ADR3
$1034	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	ADR4
$1035	0	0	0	PTCON	BPRT3	BPTR2	BPRT1	BPTR0	BPROT
$1036-8									RESERVAD0
$1039	ADPU	CSEL	IRQE	DLY	CME	0	CR1	CR0	OPTION
$103A	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	COPRST
$103B	ODD	EVEN	0	BYTE	ROW	ERASE	EELAT	EEPGM	PPROG
$103C	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0	HPRIO
$103D	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0	INIT
$103E	TILOP	0	OCCR	CBYP	DISR	FCM	FCOP	TCON	TEST1
$103F	EE3	EE2	EE1	EE0	1	NOCOP	1	EEON	CONFIG

5.2.1. PUERTO A.

EL PUERTO A consta de 8 terminales; de PA0 a PA7, que corresponden del bit 0 al bit 7 (ver las figuras 5.1.1. y 5.1.2. para su ubicación). Estas terminales realizan funciones alternativas que incluyen:

1. I/O de Propósito General.
2. Operación en el Sistema del Timer (Temporizador).
3. Sistema Acumulador de Pulsos.

La principal aplicación del PUERTO A y sus Registros de Control es en operaciones que involucran funciones que dependen del tiempo como: frecuencia, sincronización, cuenta de eventos, etc.

5.2.2. PUERTO B.

EL PUERTO B consta de 8 terminales de salida de dirección fija, que corresponden de PB0 a PB7 (ver las figuras 5.1.1. y 5.1.2. para su ubicación) y realizan funciones alternativas que incluyen:

1. I/O de Propósito General.
2. Comunicación Paralelo de Control (Strobe) Simple.

El Puerto B es direccionado en la localidad $1004 como se muestra en la Tabla 5.2.1. En la sección 5.3. de este capítulo se hace una explicación más detallada de este puerto y sus registros de control.

5.2.3. PUERTO C.

El Puerto C consta de 8 terminales bidireccionales que corresponden de PC0 a PC7, además posee un registro interno de captura de datos (latch) llamado PUERTO CL. Las terminales del PUERTO C realizan funciones alternativas que incluyen:

1. I/O bidireccionales de Propósito General.
2. Comunicación Paralelo de Control Simple o Strobe Simple.
3. Comunicación Paralelo con Protocolo Total (full handshake).

El Puerto C es direccionado en la localidad $1003 y el PUERTO CL en la localidad $1005. En la sección 5.3. de este capítulo se da una explicación más detallada de estos puertos y sus registros de control.

5.2.4. PUERTO D.

EL PUERTO D consta de 6 terminales bidireccionales que corresponden de PD0 a PD5, las funciones alternativas que pueden realizar son:

1. I/O bidireccionales de propósito general.
2. Dos terminales (D0 y D1) para Comunicación Serie Asíncrona.
3. Cuatro terminales (D2 a D5) para Comunicación Serie Síncrona.

EL PUERTO D es direccionado en la localidad $1003 como muestra la Tabla 5.2.1. La función principal de este puerto es para comunicación serie.

5.2.5. PUERTO E.

EL PUERTO E consta de 8 terminales de entrada de dirección fija que corresponden de PE0 a PE7. Las funciones alternativas que puede realizar este puerto son:

1. I (Input) de propósito general.
2. Canales de Entrada para conversión ANALOGO-DIGITAL (A/D).

EL PUERTO E es direccionado en la localidad $100A como muestra la Tabla 5.2.1. En el capítulo 6 se verá una explicación más detallada de este puerto y sus registros de control. La aplicación principal de este puerto es como convertidor Análogo/Digital.

5.3. COMUNICACION PARALELO.

La Comunicación Paralelo permite que varias señales de salida o de entrada sean manipuladas simultáneamente. Por ejemplo: las 8 terminales del PUERTO B manejan simultáneamente cada una de ellas una señal de salida.

La Comunicación Paralelo puede subdividirse en dos grandes modos:

1. Comunicación Directa.
2. Protocolo de Comunicación o Handshake.

5.3.1. COMUNICACION PARALELO EN MODO DIRECTO.

La Comunicación Paralelo en Modo Directo tiene las siguientes características:

a Es el método de comunicación más simple y es el que se utiliza en las prácticas complementarias de este curso.

b Las terminales de los puertos son utilizadas como I/O de Propósito General. Es decir, que en las terminales de los puertos configurados como salidas, el MCU escribe los datos o señales de salida; y en las terminales configuradas como entradas, el MCU lee los datos o señales de entrada.

c El software establece en que momento se leen o escriben los datos en las terminales sin necesidad de señales externas adicionales.

d Los dispositivos (leds, relevadores, actuadores, etc.) conectados a las terminales de salida, deben ser lo suficientemente rápidos para "seguir" los cambios de las señales de salida.

e Las variaciones en las señales de entrada deben ser lo suficientemente lentas, para que el MCU las puede procesar mediante control del software antes de que cambien.

Hay un cuidado que se debe tener, y es que el MCU no debe enviar o escribir datos en un puerto habilitado como entrada, pues no tiene sentido. Sin embargo el PUERTO C es un caso especial, pues posee un registro de captura (latch) donde se almacena el dato escrito, y se envía a las terminales cuando se habilitan como salida.

En la Tabla 5.3.1. se muestra la configuración de las terminales de cada puerto después de resetear el M68HC11 en el Modo de Operación Cargador (Special Bootstrap).

TABLA 5.3.1. Configuración de las terminales después de resetear en el Modo de Operación Cargador (Special Bootstrap).

PUERTO	TERMINALES	CONFIGURACION
A	A0, A1, A2, A3, A7	Entradas.
	A4, A5, A6	Salidas.
B	B0 a B7	Salidas.
C	C0 a C7	Entradas.
D	D0, D1	Entradas y Salidas de la interfaz de Comunicación Serie SCI.
	D2 a D5	Entradas
E	E0 a E7	Entradas.

5.3.2. MODO DE COMUNICACION PARALELO CON PROTOCOLO DE COMUNICACION.

Este modo de comunicación está caracterizado porque, además de las terminales utilizadas como salidas o entradas, se requieren otras terminales para señalización o habilitación.

Los Modos de Comunicación Paralelo con Protocolo de Comunicación o Handshake, puede subdividirse en 2 tipos:

1 Control Simple o Strobe Simple.
2 Protocolo Total (Full handshake).

En el Modo de Comunicación Paralelo con Protocolo de Comunicación o Handshake se utilizan: El Registro de Control Paralelo I/O **PIOC** (Parallel **I/O** Control Register), el PUERTO B y el PUERTO C.

Debido al efecto que producen en el control y configuración de la comunicación paralelo, a continuación se describe el Registro PIOC y la función de cada uno de sus bits.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
PIOC	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB	$1002
reset	0	0	0	0	0	0	1	1	

INVB-Invert Strobe B. Este bit selecciona que nivel de la señal de salida en la terminal STRB (strobe B) se considera activa

INVB=0. La señal strobe B es activa en baja (0 lógico).

INVB=1. La señal strobe B es activa en alta (1 lógico).

EGA-Active Edge for Strobe A. Este bit selecciona cuál transición de la señal es reconocida en la terminal de entrada STRA (strobe A).

EGA=0. La transición de caída es reconocida, y la de elevación ignorada.

EGA=1. La transición de elevación es reconocida, y la de caida ignorada.

PLS-Pulse/Interlocked Handshake Operation. Este bit permite seleccionar dos tipos de señales de salida en la terminal STRB (strobe B). Este bit no tiene significado si el bit HNDS=0

PLS=0. La operación Interlocked Handshake es seleccionada. La señal STROB (Strobe B) una vez activada, permanece activa hasta que la transición seleccionada de strobe A es detectada.

PLS=1. La Operación Pulsed Handshake es seleccionada. La señal strobe B es un pulso que permanece durante 2 ciclos de reloj E.

OIN-*Output or Input Handshaking. Este bit se utiliza para seleccionar:* Entrada con Protocolo Total,o Salida con Protocolo Total. No tiene significado cuando el bit HNDS=0.

OIN=0. Selecciona Entrada con Protocolo Total.

OIN=1. Selecciona Salida con Protocolo Total.

HNDS-Handshake Mode. Este bit se utiliza para seleccionar el tipo de protocolo de comunicación.

HNDS=0. Selecciona el Modo de Strobe Simple. La señal strobe A actúa para habilitar la captura (latch) de datos en el registro PUERTO CL. La señal strobe B produce un pulso de salida después de escribir en el PUERTO B.

HNDS=1. Selecciona el Modo de Entrada o Salida de Protocolo Total, involucra el PUERTO C, el bit OIN y las terminales STRA y STRB.

CWOM-Port C Wire-Or Mode. Este bit afecta las 8 terminales del Puerto C, y selecciona si las salidas actúan en forma normal CMOS para operación en Wire-Or, o funcionan como salidas drenador abierto.

CWON=0. Las salidas del Puerto C son CMOS normales.

CWOM=1. Las salidas del Puerto C actúan como drenador abierto.

STAI-Strobe A Interrup Enable Mask. Este bit permite habilitar interrupciones.

STAI=1. Cuando el bit I del registro CCR es cero, al activar el bit STAF se hace una requisición de interrupción.

STAI=0. Deshabilita la requisición de interrupción.

SATF-Strobe A Interrup Status Flag. Este bit se activa cuando una transición seleccionada ocurre en la terminal Strobe A.

5.3.2.1. Modo de Comunicación Paralelo con Strobe Simple.

A continuación mediante un ejemplo, se describe la operación de salida del PUERTO B del MCU1 y la operación de entrada del PUERTO C del MCU2 en Strobe Simple.

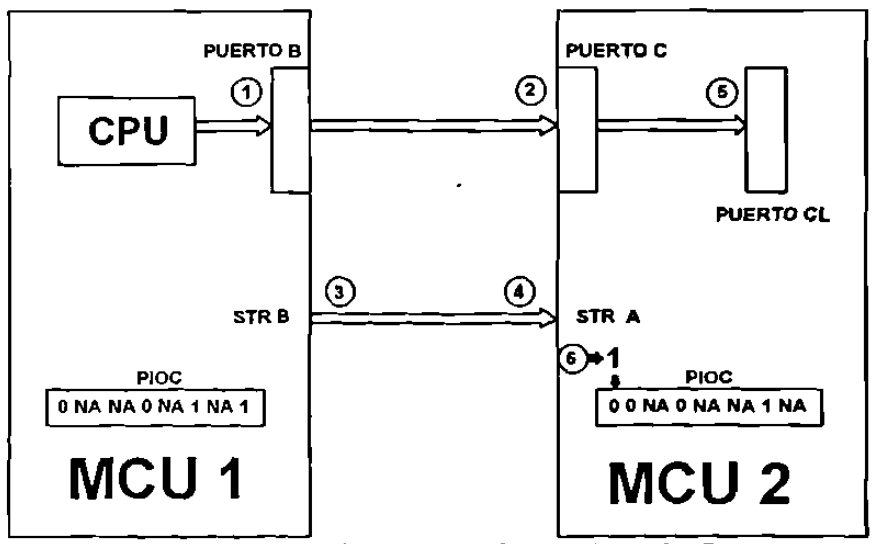


Fig.5.3.2.1.1.Diagrama de conexiones para Comunicación Paralelo

con Control Simple.

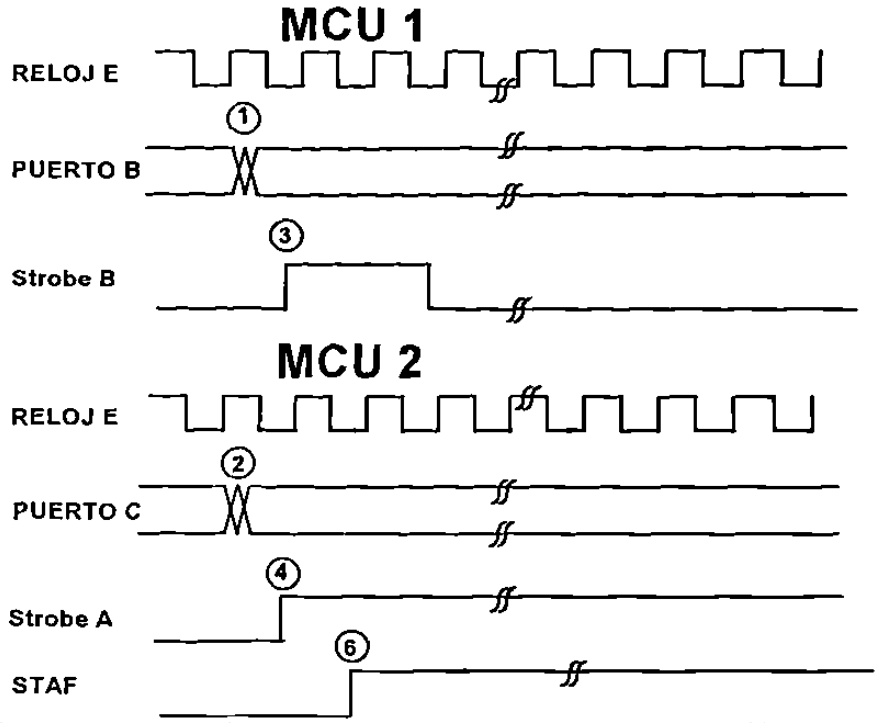


Figura 5.3.2.1.2 Diagrama de tiempos para las señales de los MCU y MCU2

En la tabla 5.3.2.3. se muestra el efecto de los valores de los bits de control de los Registro PIOC, que establecen las condiciones de operación para el Modo de Comunicación Strobe Simple de los MCU's de la figura 5.3.2.1.

Tabla 5.3.2.3. Efecto de los valores del los bits del Registro PIOC

M C U 1

Valores de los bits del registro PIOC	Efecto
STAF=0	Se mantiene en cero pues no recibe señal STRA
STAI=NA	No afecta
CWOM=NA	No afecta al Puerto B
HNDS=0	Comunicación Strobe Simple
OIN=NA	No afecta
PLS=1	Señal Strobe B pulsada
EGA=NA	No afecta la señal de salida.
INVB=1	Señal Strobe B activa en alta.

M C U 2

Valores de los bits del registro PIOC	Efecto
STAF=0	Se pone en 1 al recibir la señal STRA
STAI=0	No solicita interrupción al activarse STAF
CWOM=NA	No afecta
HNDS=0	Comunicación Strobe Simple
OIN=NA	No afecta.
PLS=NA	No afecta.
EGA=1	Captura (latch) la señal de entrada en el Puerto CL en la transición de elevación de la señal STRA.
INVB=NA	No afecta.

Para la descripción de la operación en el Modo de Comunicación Strobe Simple refiérase a las figuras 5.3.2.1.1. y 5.3.2.1.2. Obsérvese la concordancia entre los números encirculados con los siguientes incisos:

1. El programa del MCU1 escribe un dato en las terminales de salida del PUERTO B.

2. El dato es recibido en las terminales del PUERTO C del MCU2.

3. El MCU1 produce la señal strobe B con duración de 2 ciclos de reloj E.

4. La señal de salida Strobe B en la terminal STRB del MCU1 es recibida como señal de strobe A en la terminal STRA del MCU2. Como EGA=1 en el registro PIOC del MCU2, entonces se reconoce como señal la transición de elevación.

5. La señal strobe A ocasiona que los datos sean capturados (latched) en los registros del PUERTO CL.

6. Para el MCU2 las señales recibidas están fuera de sincronía respecto a sus señales de reloj E, por lo que activa el bit STAF para sincronizarla,e indicar que el dato ha sido recibido y está disponible.

5.3.2.2. Modo de Comunicación Paralelo con Protocolo Total.

Este modo de comunicación permite una comunicación más eficiente con otros dispositivos como: impresoras, computadoras, otros MCU's, etc. Pues además de los datos puede incluir señales que contienen toda o parte de la siguiente información:

- Que el MCU está listo para enviar el dato.

- El receptor contesta que está listo para recibir el dato.

- El MCU coloca el dato en las terminales de salida.

- El MCU envía una señal para que el receptor capture el dato.

- El receptor envía una señal confirmando haber recibido el dato.

- Después de procesar el dato, el receptor envía una señal indicando que esta listo para recibir otro dato.

A continuación se describe un ejemplo de Comunicación Paralelo con Protocolo Total (Full Handshake). En este ejemplo, la salida del MCU se utiliza como señal para una impresora. En la tabla 5.3.2.2.1. se muestra el efecto de los valores del Registro PIOC para configurar el microcontrolador para Comunicación con Protocolo Total.

Tabla 5.3.2.2.1. Efecto de los valores de los bits del Registro PIOC.

Valores de los bits del registro PIOC	Efecto
STAF=1	Este bit es puesto en 1 después de recibir la transición seleccionada en la terminal Strobe A
STAI=1	Habilita una interrupción para que al activarse el bit STAF otro dato sea transmitido.
CWON=0	Configura el Puerto C para salidas normales.
HNDS=1	Comunicación Paralelo con Handshake
OIN=1	Selecciona Salida con Protocolo Total.
PLS=1	La señal Strobe B es un pulso
EGA=0	La transición de caída es reconocida en la señal de Strobe A.
INVB=0	La señal Strobe B es activa en baja

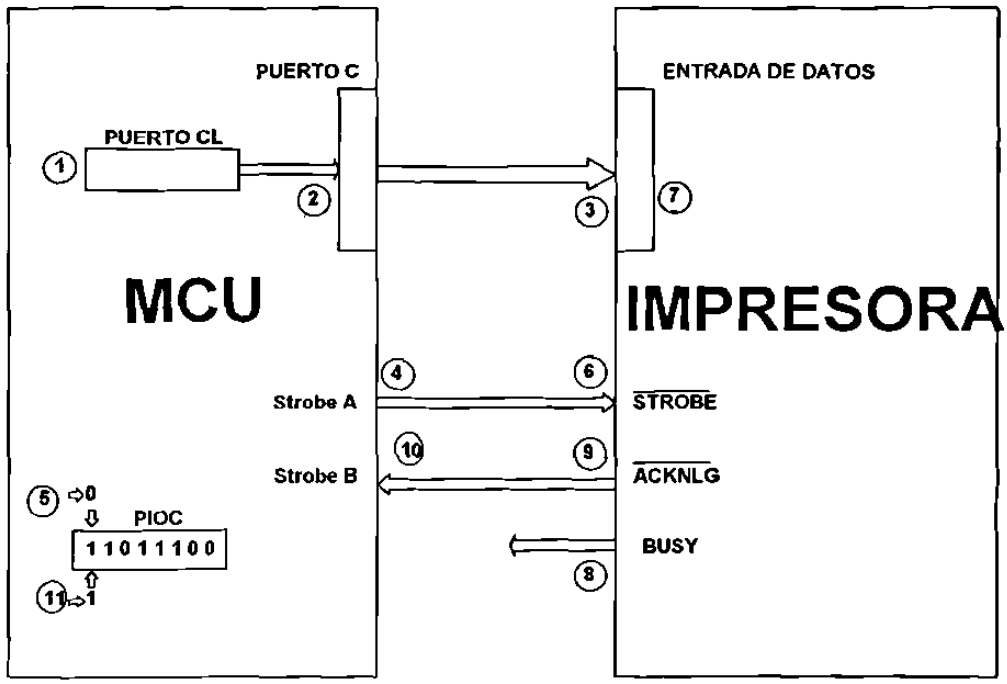


Figura 5.3.2.2.1. Diagrama de conexiones entre el MCU y la Impresora.

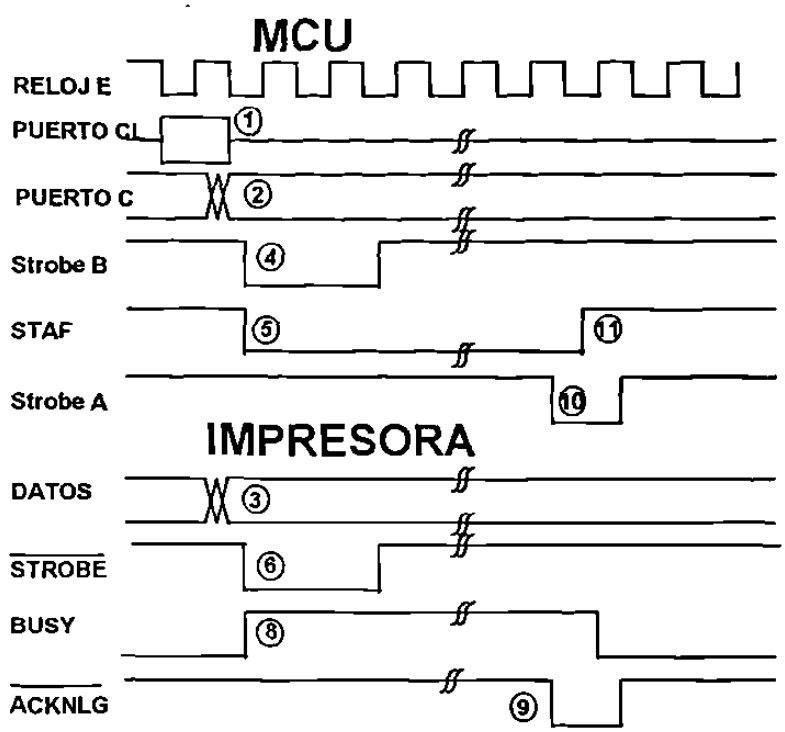


Figura.5.3.2.2.2. Diagrama de tiempos para el MCU y la Impresora.

En la figura 5.3.2.2.1. se muestra el diagrama de conexiones entre el microcontrolador y la impresora. En la figura 5.3.2.2.2. se muestra el diagrama de tiempos para las señales transmitidas y recibidas en el MCU y la impresora. Observe la concordancia entre los números encirculados en ambas figuras con la descripción en los siguientes incisos.

1. El programa escribe un dato en el PUERTO CL .

2. Se envía el dato a las terminales del PUERTO C.

3. El dato aparece en las terminales de ENTRADA DE DATOS de la impresora.

4. Después de un tiempo suficiente para que se estabilice el dato, el MCU genera la señal Strobe B.

5. Se desactiva el bit STAF.

6. La señal Strobe B es recibida en la terminal $\overline{\text{STROBE}}$ de la impresora.

7. El dato es capturado (latch) por la impresora.

8. Se activa la señal BUSY indicando que la impresora está ocupada.

9. Después de imprimir el dato, la señal $\overline{\text{ACKNLG}}$ se activa en baja, indicando que puede recibir otro dato la impresora.

10. La señal $\overline{\text{ACKNLG}}$ se recibe en la terminal Strobe A, indicándo al MCU que puede enviar otro dato. Nota: la señal BUSY puede ser utilizada en lugar de la señal ACKNLG.

11. Debido a que la señal en Strobe A puede estar fuera de sincronía, hay que esperar a que el bit STAF se active en sincronía para que el MCU pueda enviar otro dato.

CAPITULO 6

CONVERTIDOR ANALOGO-DIGITAL

6.1.- INTRODUCCION.

Todas las variables físicas son de naturaleza analógica, y nos hemos acostumbrado a movernos en este amplio espectro de valores analógicos. Sin embargo, los circuitos internos de los MCU's son de naturaleza digital y manejan señales digitales. Por lo que se requiere que las señales pasen por un convertidor de valores analógicos a valores digitales.

6.2.- CONCEPTOS GENERALES.

En la figura 6.2.1. se puede observar una señal formada por un Tren de Pulsos de duración ts y período T. Estas dos señales después de ser manipuladas por un Modulador de Pulsos, producen una señal análoga muestreada, que consiste de un tren de pulsos modulados en amplitud y que también tienen una duración ts y período T. Precisamente este tren de pulsos modulados en amplitud, son manipulados por el Convertidor A/D (Análogo/Digital) para obtener la señal digitalizada, presentada generalmente en código binario o hexadecimal.

Es conveniente observar las siguientes diferencias entre la señal análoga continua y la señal digitalizada resultante:

a) La Señal análoga continua se representa gráficamente, y su valor puede ser conocido en cualquier instante de tiempo.

b) La Señal digitalizada es una lista de números, que nos proporciona el valor de la señal en instantes de tiempo específicos, en este caso cada T segundos.

En la figura 6.2.2. se muestra una vista amplificada de la señal análoga muestreada. El punto a corresponde al valor instantáneo que deseamos convertir a valor digital, y ese sería el valor convertido si el muestreador fuera ideal, es decir si ts=0. Pero el convertidor A/D del MC68HC11A8 utiliza un valor de ts=12 ciclos de Reloj E (ts=6μs para una frecuencia de Reloj E = 2 MHz). El valor en el punto b al final del tiempo de muestreo, es el valor que realmente es convertido a valor digital. Esta diferencia en valores debe ser tomada en cuenta cuando la señal análoga cambia muy rápidamente de valor.

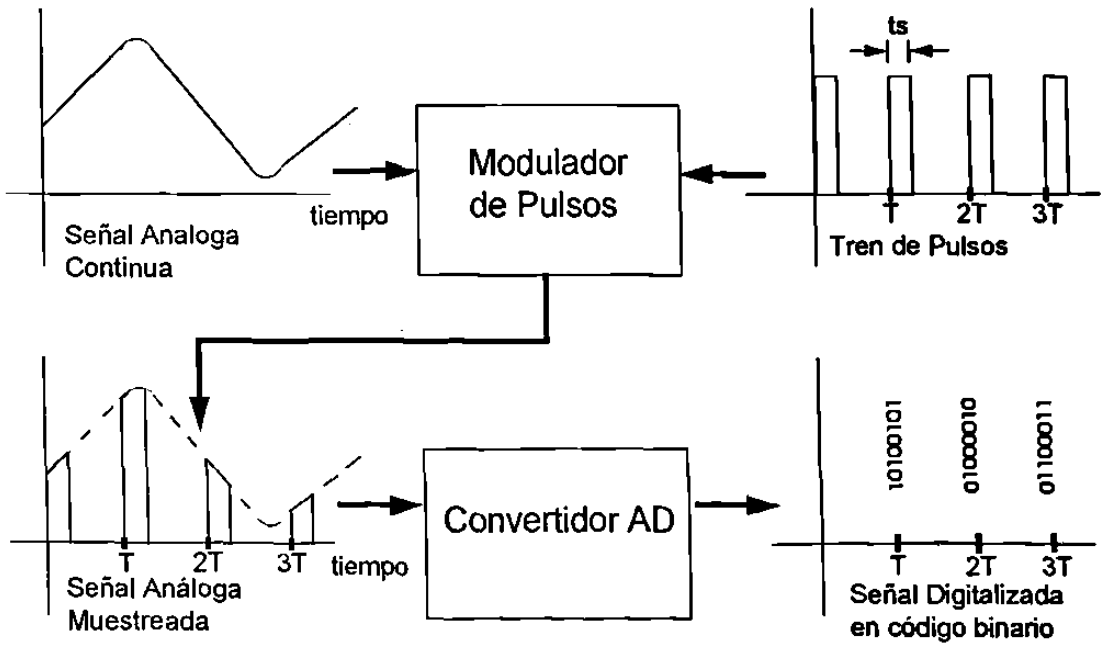


Figura 6.2.1 Proceso de Conversión Análogo Digital

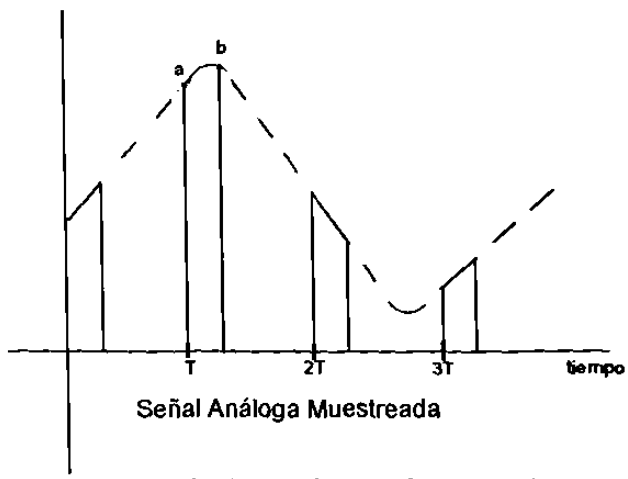


Figura 6.2.2. Señal Análoga Muestreada.

Después de muestrear la señal, el valor del punto b es mantenido durante todo el tiempo que toma la conversión, es decir 18 ciclos de Reloj E (para una frecuencia de Reloj E = 2MHz, el tiempo de conversión es de 9 µs). Después de realizada la conversión, el MCU utiliza 2 ciclos de Reloj E para almacenar el resultado en el Registro de Resultados. Un total de 32 ciclos de Reloj E, es el tiempo total que toma el proceso de muestreo-conversión-almacén de resultados (16 µs para una frecuencia de Reloj E de 2MHz).

Existen dos entradas para establecer voltajes de referencia analógicos para la conversión: V_{RH} y V_{RL}. El rango de conversión se establece de desde V_{RL} hasta V_{RH} que se convierte a un valor $FF de 8 bits (256 en decimal). Todos los valores analógicos que son convertidos digitalmente se expresan como un valor de 8 bits que puede variar de $00 a $FF. Esto corresponde a realizar un conteo de 256 partes, por lo que el error de conversión es de 1 bit que equivale a 1/256.

En la Tabla 6.2.1. se puede observar que el valor de salida digitalizado equivale a diferentes valores reales, lo cual depende de los valores de V_{RH} y V_{RL}.

TABLA 6.2.1.

V_{RH} = 5V V_{RL} = 0V	V_{RH} = 5V V_{RL} = 2.5 V	Valor Digital en Binario	Valor Digital en Hexadecimal
Valor Real Rango 0 - 5	Valor Real Rango 2.5. - 5		
5	5.0	11111111	FF
4	4.5	11001100	CC
3	4.0	10011001	99
2	3.5	01100110	66
1	3.0	00110011	33
0	2.5	00000000	00

El sistema A/D (Análogo/Digital) del MC68HC11A8 consiste de:

a. Dos terminales V_{RL} y V_{RH} utilizadas como entradas de voltaje de referencia.

b. Un solo convertidor A/D de redistribución de cargas totalmente capacitivo de aproximaciones sucesivas.

c. Un Multiplexador de Entrada para seleccionar uno de 16 canales (incluyendo los 8 canales asociados con cada terminal del PUERTO E).

d. Un Circuito de Control.

e. Cuatro Registro de Resultados.

f. Un oscilador interno RC.

6.3.- TERMINALES V_{RL} Y V_{RH}.

Estas terminales son utilizadas para proporcionar los voltajes de referencia para el Convertidor A/D y deben mantenerse dentro de los siguientes rangos para mayor precisión.

$V_{RH} \leq V_{DD}$ $V_{RL} \geq V_{SS}$ $5 \geq V_{RH} - V_{RL} \geq 2.5.$

6.4.- MULTIPLEXADOR DE ENTRADA.

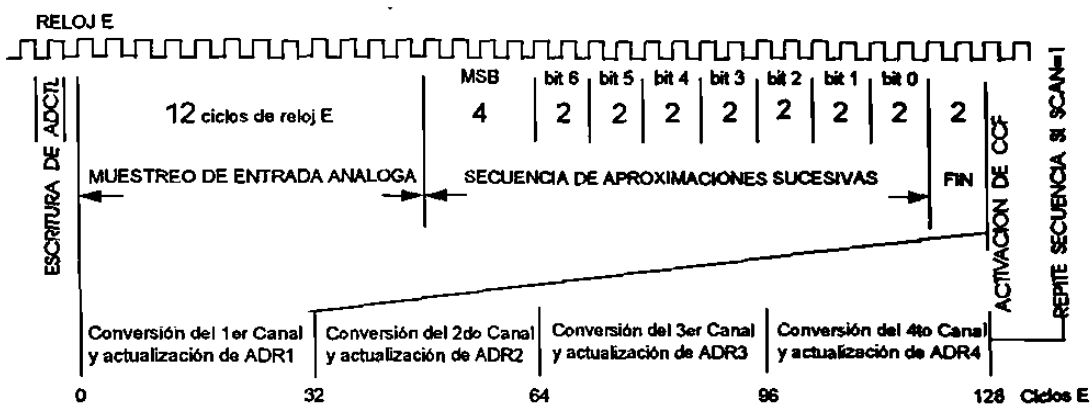
El Sistema Convertidor A/D posee un Multiplexador de Entrada que permite que el único Convertidor A/D seleccione una de 16 señales análogas, 8 de estos canales corresponden a las líneas de entrada del PUERTO E. De los 8 restantes: 4 son para puntos de referencia interna para pruebas y 4 son reservadas para usos futuros. Vea la tabla 6.5.1.1. para la asignación de canales.

6.5.- CIRCUITO DE CONTROL.

El Circuito de Control del Sistema de Conversión A/D permite que mediante diversos registros se realicen las siguientes funciones:

- Selección de canales para conversión A/D.
- Operación de Conversión en un Solo Canal o Multicanal, ambos con las modalidades: sin barrido continuo o con barrido continuo.
- Almacenamiento de Resultados A/D.
- Activación del Sistema A/D y Selección de un Oscilador RC interno.

En la figura 6.5.1. se muestra un diagrama de tiempos para la secuencia de Conversiones A/D, donde se puede observar que se toman como base de sincronía los pulsos de reloj E.



NOTAS: -El proceso de conversión se inicia después de escribir en el registro ADCTL
-Los resultados de las conversiones son colocados en el registro SAR y son transferidos al registro ADRX (X=1,2,3,4) al FINal del periodo de cada conversión
-La bandera CCF es puesta en 1 durante el FINal de la cuarta conversión.
-En esta figura se asume que CSEL=0 en el registro OPTION, por lo que los pulsos de reloj E actúan como sincronizadores.
-Si MULT=0 las cuatro conversiones en secuencia se realizan en el mismo canal

Figura 6.5.1 Secuencia de Conversión Análogo-Digital

Los registros involucrados en el proceso de conversión A/D son mostrados enseguida con la asignación de sus bits.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
$ 1030	CCF		SCAN	MULT	CD	CC	CB	CA	ADCTL
reset	0	0	I	I	I	I	I	I	I=Indefinido
$ 1031	-	-	-	-	-	-	-	-	ADR1
$ 1032	-	-	-	-	-	-	-	-	ADR2
$ 1033	-	-	-	-	-	-	-	-	ADR3
$ 1034	-	-	-	-	-	-	-	-	ADR4
$ 1039	ADPU	CSEL	IRQE	DLY	CME	-	CR1	CR0	OPTION
reset	0	0	0	1	0	0	0	0	

6.5.1.- SELECCION DE CANALES PARA CONVERSION A/D.

En el Registro A/D de Control y Status **ADCTL** (**A/D CONTROL/STATUS REGISTER**), se encuentran cuatro bits selectores de canales: CA, CB, CC y CD, que permiten seleccionar 1 de 16 canales de conversión A/D. En la tabla 6.5.1. se muestra la asignación de cada bit para seleccionar cada canal.

Cuando el modo multicanal es seleccionado haciendo el bit MULT=1, los dos bits menos significativos selectores de canal CA y CB, no tienen efecto, y los bits CC y CD especifican cual grupo de 4 canales es convertido.

Tabla 6.5.1.1. Asignación de canales Análogo-Digital.

CD	CC	CB	CA	Señal del Canal	Resultado en ADRx si MULT=1
0	0	0	0	AN0	ADR1
0	0	0	1	AN1	ADR2
0	0	1	0	AN2	ADR3
0	0	1	1	AN3	ADR4
0	1	0	0	AN4 **	ADR1
0	1	0	1	AN5 **	ADR2
0	1	1	0	AN6 **	ADR3
0	1	1	1	AN7 **	ADR4
1	0	0	0	Reservada	ADR1
1	0	0	1	Reservada	ADR2
1	0	1	0	Reservada	ADR3
1	0	1	1	Reservada	ADR4
1	1	0	0	VRH Pin*	ADR1
1	1	0	1	VRL Pin*	ADR2
1	1	1	0	(VRH)/2*	ADR3
1	1	1	1	Reservada	ADR4

* Este grupo de canales es usado durante pruebas de fábicación.
**No disponibles en versión empacada en 48 terminales.

6.5.2.- MODOS DE OPERACION DE CONVERSION.

La **Operación de Conversión en un Solo Canal** tiene dos variaciones: Sin Barrido Continuo y con Barrido Continuo.

La **Conversión A/D en un Solo Canal Sin Barrido Continuo** se obtiene por medio del registro ADCTL (los bits MULT=0 y SCAN=0). En este caso un solo canal es convertido 4 veces consecutivas, almacenando el primer resultado en el registro de resultados **ADR1** (**A/D** Result Register **1**), el segundo resultado en ADR2, el tercero en ADR3 y el cuarto en ADR4. Después de que se termina la cuarta conversión, se activa el bit de bandera de Conversión Completa **CCF** (Conversions Complete Flag) y se detiene toda actividad de conversión hasta que un nuevo proceso de conversión es ordenado al escribir en el registro ADCTL.

La **Conversión A/D de un Solo Canal Con Barrido Continuo** se obtiene con el registro ADCTL (los bits MULT=0 y SCAN=1). En esta variación la conversión continua es realizada, y el resultado del canal seleccionado en la quinta conversión es almacenado en el Registro de Resultados ADR1, remplazando el primer resultado de la conversión, la sexta conversión remplaza a la que existía en ADR2, y así sucesivamente.

La **Operación de Conversión Multicanal** tiene dos variantes: sin barrido continuo y con barrido continuo.

La **Conversión A/D Multicanal Sin Barrido Continuo** se obtiene con el registro ADCTL haciendo los bits MULT=1 y SCAN=0. El grupo de 4 canales es seleccionado con los bits CC y CD de acuerdo a la Tabla 6.5.1.1. y los bit CA y CB no tienen efecto. Los 4 canales seleccionados son convertidos uno cada vez, con el primer resultado almacenado en el registro ADR1, avanzando hasta que la cuarta conversión es almacenada en ADR4.

La **Conversión A/D Multicanal con Rastreo Continuo** se obtiene con el registro ADCTL (los bits MULT=1 y SCAN=1).

6.5.3. ACTIVACION DEL SISTEMA A/D

La activación del Sistema A/D se realiza mediante el bit 7 **ADPU (A/D** Power Up) del registro OPTION. Cuando ADPU es 1, el sistema A/D es habilitado y un retardo de 100 µs se requiere para que se estabilice.

6.5.4. SELECCION DEL OSCILADOR RC INTERNO.

La Selección de un Oscilador RC interno es requerida cuando la frecuencia de Reloj E es menos de 750KHz. Mediante el bit 6 **CSEL** (Clock Select) del Registro OPTION se selecciona la operación del Oscilador RC interno. Cuando CSEL=0 el sistema A/D usa el sistema de Reloj E, cuando CSEL=1 se activa el Oscilador RC interno para que maneje el sistema A/D a una frecuencia de 1.5 MHz.

GLOSARIO

Acumuladores A, B y D. Los acumuladores A y B son registros de 8 bit de propósito general usados para mantener los operandos y los resultados de cálculos aritméticos o de manipulación de datos. Estos dos acumuladores pueden ser concatenados en un solo acumulador de 16 bit llamado Acumulador D.

Apuntador de Pila SP (Stack Pointer) es un registro de 16 bit que contiene la dirección de la siguiente localidad libre en el stack (pila).

Arquitectura de la Computadora es el nombre dado a la organización funcional de un sistema básico de computadora y consta de 5 unidades:CPU o Unidad de Procesamiento Central. RAM o Memoria de Acceso Aleatorio. ROM o Memoria de Solo Lectura. BUSES o Conjunto de alambres que transportan información con un propósito común, y PUERTOS I/O o Puertos de Entrada/Salida.

AS11.EXE es un programa ensamblador de la compañia Motorola el cual puede ser utilizado en computadoras IBM-PC o compatibles y produce el Programa Objeto para los MCU's de la familia M68HC11,

Bit es un dígito binario, esta palabra se forma funcionando **binary digit** y es la mínima cantidad de información que se puede almacenar o manejar.

Bus de control es un conjunto de líneas con una mezcla de señales, cada una con un rol específico en el control de la actividad del sistema. Como regla, las señales de control son señales de sincronización proporcionadas por el CPU hacia los circuitos externos para sincronizar el movimiento de información entre los buses de dirección y de datos.

Bus de datos es el conjunto de líneas de doble sentido o dirección, que se utiliza para introducir o sacar datos de la CPU. El bus de datos transporta información entre el CPU y la memoria o entre el CPU y dispositivos I/O (Entrada/Salida).

Bus de direcciones es un conjunto de líneas que selecciona una cierta localidad de memoria, puerto de entrada o puerto de salida. El bus de direcciones es utilizado para operaciones de lectura y escritura,

Bus es un conjunto de alambres que transportan información con un propósito común. El acceso a los circuitos de control alrededor del CPU es proporcionado por tres buses: El bus de direcciones, el bus de datos y el bus de control.

Byte es el conjunto de 8 bits o dos nibble.

CCR (Condition Code Register) Registro de Código de Condiciones también recibe el nombre de **Registro de Banderas**, pues se les da el nombre de **Banderas** (flags) a los bits contenidos en el registro de condiciones. El registro CCR tiene: 5 indicadores de status (H, N, Z, V, C), 2 bits de interrupciones (I,X) y un bit habilitador de STOP (S). En seguida se describe cada una de las banderas.

Computadora puede ser definida como un sistema de procesamiento de datos, que puede ser programada para operar sin la intervención humana, y que tiene además la habilidad de almacenar y obtener datos.

Comunicación Paralelo permite que varias señales de salida o de entrada sean manipuladas simultáneamente.

Conjunto de Instrucciones es el grupo de instrucciones que un microprocesador o microcontrolador específico puede ejecutar. Está formado por un conjunto de códigos binarios que el CPU (Inherentemente en su diseño) es capaz de entender y ejecutar.

Constantes representan cantidades que no varían en valor durante le ejecución del programa.

Convertidor A/D (Análogo/Digital) es un dispositivo utilizado para obtener la señal digitalizada, presentada generalmente en código binario o hexadecimal, apartir de un tren de pulsos modulados en amplitud.

CPU (Central Procesing Unit) o Unidad de Procesamiento Central es el "cerebro" de la computadora, administra todas las actividades en el sistema y realiza todas las operaciones con los datos. La CPU es solamente una colección de circuitos lógicos que realizan en forma continua solo dos operaciones: reconocer instrucciones (fetch instructions) y ejecutar instrucciones (executing).

Diagrama de Flujo es una representación gráfica que ilustra los pasos lógicos, cálculos y decisiones que en secuencia deben ser realizados para cumplir con una tarea específica.

DIP (Dual In-line Package) tipo de encapsulado de circuitos integrados

Directivas son instrucciones que controlan el funcionamiento del Programa Ensamblador y no son transferidas al Programa Objeto.

E señal de reloj usada como referencia de tiempo o de sincronización.

Editor de Texto es un programa residente en computadora que permite usarla como una máquina de escribir eléctrica. y es utilizado para escribir el Programa Fuente.

EEPROM: (Electrical Eraseble Programmable ROM) tipo de memoria puede ser borrada y programada electricamente.

Ensamblador es un programa que procesa el Programa Fuente (escrito en lenguaje ensamblador) y lo traduce en Programa Objeto (escrito en lenguaje de máquina) el cual es ejecutable por el MCU, además produce un listado del Programa.

EPROM: Memorias de Lectura Exclusiva Reprogramables, esta memoria pueden ser borrada con luz ultravioleta para ser reprogramada con nuevos patrones de bits.

Escritura, Operación de el CPU saca un dato y lo coloca en el bus de datos, debido a la señal de control, la memoria reconoce la operación como un ciclo de escritura y almacena el dato en la localidad especificada.

Expresión es una combinación de: operadores (algebraicos o lógicos), etiquetas y constantes.

Firmware es el nombre dado al software que es almacenado permanentemente en la memoria de programa.

Hardware es el equipo y constituye las unidades físicas

HCMOS = (High-density Complementary Metal Oxide Semiconductor) tecnología CMOS de alta densidad utilizada para fabricar circuitos integrados.

IRQ o de Requisición de Interrupción, terminal que proporciona un medio para requerir una interrupción asíncrona para el MCU.

K es la abreviación de kilo y es estandarizada para 2^{10}=1024,

Lectura Operación de recupera un byte de dato de una localidad especifica en la memoria y la coloca en el bus de datos. El CPU lee el dato y lo coloca en uno de sus registros internos.

Lenguaje de Máquina es un lenguaje binario que está constituido exclusivamente por unos y ceros, y es el único lenguaje que reconocen los MCU's.

Lenguaje Ensamblador es una colección de símbolos mnemónicos que representan: operaciones (mnemónicos de instrucciones del MCU y directivas para el ensamblador), etiquetas, operadores y símbolos especiales.

Lenguajes de Alto Nivel utilizan proposiciones en inglés que representan desde unas cuantas, hasta muchas instrucciones equivalentes de lenguaje ensamblador o lenguaje de máquina. algunos lenguajes de alto nivel son: BASIC, FORTRAN, Pascal, C, COBOL, etc.

LIFO (Last-In-First-Out) memoria,de lectura/escritura en donde el último dato en entrar es el primero en salir.

Localidad es el nombre dado a cada uno de estos registros de la memoria

LSI (Large Scale Integration) de gran escala de integración.

Memorias son circuitos capaces de almacenar grandes cantidades de información mediante un determinado número de registros que utilizan entradas y salidas comunes para el acceso a todos ellos.

Micro computadoras tienen como característica principal su tamaño reducido y el empacado del CPU, el cuál está contenido en un solo circuito integrado llamado microprocesador.

Microcontrolador es un circuito integrado que contiene: CPU, RAM, ROM, interfaz serie, interfaz paralelo, timer, y circuitos con un programa de interrupciones,

Microcontroladores son circuitos integrados LSI (Large Scale Integration) altamente desarrollados, cuyo diseño esta enfocado hacia actividades de control.

Minicomputadoras son computadoras que tienen un CPU consistente de varios circuitos integrados, lo cuál es necesario para obtener mayor velocidad y potencia de cómputo, además las minicomputadoras son multiusuario y ejecutan simultáneamente varios programas, lo cual es una ilusión resultante de un recurso de la CPU llamado "tiempo compartido".

Mnemónicos son abreviaturas o siglas de palabras (en inglés) que facilitan el recordar la función que realiza una instrucción y se utilizan para aproximar la lectura y escritura de programas al lenguaje habitual (en inglés).

Modelo del Programador es el nombre dado al conjunto de registros internos a que tiene acceso el programador. (Acumulador A, B y D, Registros Indice X e Y, Apuntador de Pila SP, Contador de Programa PC , Registro del Código de Condición CCR.

Modos de Direccionamiento son las diferentes formas en que se especifican el origen o destino de los datos que son manipulados por la CPU.

MSI (Medium Scale Intregration) de media escala de integración

Nibble es un conjunto de 4 bits, por lo tanto un byte tiene dos nibble.

PC (Program Counter) Contador de Programa es un registro de 16 bit que apunta siempre a la localidad de la siguiente instrucción que va a ser ejecutada.

PLCC (Plastic Leaded Chip Carrier) tipo de encapsulado de circuitos integrados

Pop es leer en la parte de arriba del Stack,

Programa en Lenguaje Ensamblador es el texto de una Programa Fuente.

Programa Fuente o Programa en Lenguaje Ensamblador es el texto donde los mnemónicos (ayudas de memorización) del fabricante son utilizados para representar las instrucciones del MCU.

Programa Monitor es un paquete de software utilizado para modificar y verificar que el software desarrollado por el usuario realiza la función esperada.

Programa Objeto es el nombre dado a un programa ya expresado en lenguaje de máquina.

Programación es la preparación de la lista de instrucciones que una computadora, microprocesador o microcontrolador debe ejecutar.

Programación Estructurada es una técnica de programación muy parecido a la estructura jerárquica de una corporación, donde se establece en forma descendente como se desea que el programa realice su trabajo.,

PROM: Memorias programadas por el usuario en el momento de diseñar el sistema. Solo pueden ser programadas una vez.

Puerto es un conjunto de terminales utilizado para entrada o salida de datos en un MCU.

PUERTO I/O es un conjunto de terminales utilizadas para entrada (Input) o salida (Output) de datos en un microcontrolador.

Pull es leer en la parte de arriba del Stack,

Push es escribir en la parte de arriba del Stack,

RAM (Random Access Memory) o memoria de acceso aleatorio es una memoria de lectura/escritura en donde se almacena en forma temporal los datos o algunos programas de naturaleza temporal.

RAM (Random Access Memory) son memorias de acceso aleatorio en las que es posible la escritura y lectura de datos.

Read o lectura es la operación de obtener el valor previamente almacenado en una localidad de memoria.

Registros Indice X e Y son de 16 bit, y son usados para el Modo de Direccionamiento Indexado.

RESET. señal usada como entrada para inicializar en un estado de arranque conocido

ROM (Read Only Memory) o Memoria de Solo Lectura habitualmente se utiliza para almacenar el programa en forma permanente. \

Señal digitalizada es una lista de números, que nos proporciona el valor de la señal en instantes de tiempo específicos,

Símbolo ⇦ indica que "es colocado en" o "transferido a".

Símbolo ☜ indica la siguiente instrucción que será ejecutada por la CPU.
símbolo #, indica direccionamiento inmediato.

Sistema de Desarrollo de Software es el conjunto de técnicas y herramientas que el programador utiliza en el proceso de desarrollo de software y consta de: Técnicas de Programación, Lenguajes de Programación, Editor de Texto, Programa Ensamblador, Programa Monitor.

Software es un término general utilizado para denominar a todos los programas, está constituido por una serie de instrucciones combinadas para realizar un trabajo importante.

Stack o Pila es un área seleccionada de memoria RAM, utilizado para almacenar en forma temporal datos o direcciones.

Stack Pointer (SP) o Apuntador de Pila es el registro utilizado para direccionar el Stack. El Stack Pointer o SP almacena (apunta hacia) la dirección de la última localidad vacía utilizable para almacenar datos.

Strobe es una señal de control.

Subrutinas son un bloque de instrucciones que son utilizadas repetidamente, y permiten de una manera sencilla, dividir el trabajo de un programa en secciones.

Talker es el nombre dado al programa de comunicación usado por el Programa Monitor PCbug11

Write o escritura es la operación de almacenar datos en memoria.

XIRQ o Interrupción No Enmascarable, terminal que proporciona un medio para requerir interrupciones no enmascarables

APENDICE. CONJUNTO DE INSTRUCCIONES DEL MC68HC11

Mnemonic	Operation	Boolean Espression	Addresing Mode for Operand	Machine Coding (hexadecimal)		Condition Codes							
				Opcode	Operand(s)	S	X	H	I	N	Z	V	C
ABA	Add Accumulators	A+B⇨A	INH	1B	dd mm	-	-	⇕	-	⇕	⇕	⇕	⇕
ABX	Add B to X	IX+00:B⇨IX	INH	3A		-	-	-	-	-	-	-	-
ABY	Add B to Y	IY+00:B⇨IY	INH	183A		-	-	-	-	-	-	-	-
ADCA	Add with Carry to A	A+M+C⇨A	A IMM	89	ii	-	-	⇕	-	⇕	⇕	⇕	⇕
			A DIR	99	dd								
			A EXT	B9	hh ll								
			A IND,X	A9	ff								
			A IND,Y	18 A9	ff								
ADCB	Add with Carry to B	B+M+C⇨B	B IMM	C9	ii	-	-	⇕	-	⇕	⇕	⇕	⇕
			B DIR	D9	dd								
			BEXT	F9	hh ll								
			B IND,X	E9	ff								
			B IND,Y	18 E9	ff								
ADDA	Add Memory to A	A+M⇨A	A IMM	8B	ii	-	-	⇕	-	⇕	⇕	⇕	⇕
			A DIR	9B	dd								
			A EXT	BB	hh ll								
			A IND,X	AB	ff								
			A IND,Y	18 AB	ff								
ADDB	Add Memory to B	B+M⇨B	B IMM	CB	ii	-	-	⇕	-	⇕	⇕	⇕	⇕
			B DIR	DB	dd								
			B EXT	FB	hh ll								
			B IND.X	EB	ff								
			B IND,Y	18 EB	ff								
ADDD	Add 16-bit to D	D+M:M+1⇨D	IMM	C3	jj kk	-	-	-	-	⇕	⇕	⇕	⇕
			DIR	D3	dd								
			EXT	F3	hh ll								
			IND,X	E3	ff								
			IND,Y	18 E3	ff								
ANDA	AND A with Memory	A•M⇨A	A IMM	84	ii	-	-	-	-	⇕	⇕	0	-
			A DIR	94	dd								
			A EXT	B4	hh ll								
			A IND,X	A4	ff								
			A IND,Y	18A4	ff								

CONJUNTO DE ISTRUCCIONES continuación...

Mnemonic	Operation	Boolean Expression	Addresing Mode for Operand	Machine Coding (hexadecimal)		Condition Codes							
				Opcode	Operand(s)	S	X	H	I	N	Z	V	C
ANDB	AND B with Memory	B•M⇨B	B IMM	C4	ii	-	-	-	-	⇕	⇕	0	-
			B DIR	D4	dd								
			B EXT	F4	hh ll								
			B IND,X	E4	ff								
			B IND,Y	18E4	ff								
ASL	Arithmetic Shift Left		EXT	78	hh ll	-	-	-	-	⇕	⇕	⇕	⇕
			IND,X	68	ff								
			IND,Y	18 68	ff								
ASLA			A INH	48									
ASLB			B INH	58									
ASLD	Arithmetic Shift Left Double		INH	05		-	-	-	-	⇕	⇕	⇕	⇕
ASR	Arithmetic Shift Rigth		EXT	77	hh ll	-	-	-	-	⇕	⇕	⇕	⇕
			IND,X	67	ff								
			IND,Y	18 67	ff								
ASRA	Arithmetic Shift Rigth		A INH	47									
ASRB	Arithmetic Shift Rigth		B INH	57									
BCC	Branch if Carry Clear	?C=0	REL	24	rr	-	-	-	-	-	-	-	-
BCLR	Clear Bit(s)	M•(mm)⇨M	DIR	15	dd mm	-	-	-	-	⇕	⇕	0	-
			IND,X	1D	ff mm								
			IND,Y	18 1D	ff mm								
BCS	Branch if Carry Set	?C=1	REL	25	rr	-	-	-	-	-	-	-	-
BEQ	Branch if = Zero	?Z=1	REL	27	rr	-	-	-	-	-	-	-	-
BGE	Branch if ≥ = Zero	?N⊕V=0	REL	2C	rr	-	-	-	-	-	-	-	-
BGT	Branch > Zero	?Z+(N⊕V)=0	REL	2E	rr	-	-	-	-	-	-	-	-
BHI	Branch if Higher	?C+Z=0	REL	22	rr	-	-	-	-	-	-	-	-
BHS	Branch if Higher or Same	?C=0	REL	24	rr	-	-	-	-	-	-	-	-
BITA	Bit(s) Test A with Memory	A•M	A IMM	85	ii	-	-	-	-	⇕	⇕	0	-
			A DIR	95	dd								
			A EXT	B5	hh ll								
			A IND,X	A5	ff								
			A IND,Y	18A5	ff								

CONJUNTO DE INSTRUCCIONES continuación..

Mnemonic	Operation	Boolean Expression	Addresing Mode for Operand	Machine Coding (hexadecimal)		Condition Codes							
				Opcode	Operand(s)	S	X	H	I	N	Z	V	C
BITB	Bit(s) Test B with Memory	B•M	B IMM B DIR B EXT B IND,X B IND,Y	C5 D5 F5 E5 18 E5	ii dd hh ll ff ff	-	-	-	-	⇕	⇕	0	-
BLE	Branch if ≤ Zero	?Z+(N⊕V)=1	REL	2F	rr	-	-	-	-	-	-	-	-
BLO	Branch if Lower	?C=1	REL	25	rr	-	-	-	-	-	-	-	-
BLS	Branch if Lower or Same	?C+Z=1	REL	23	rr	-	-	-	-	-	-	-	-
BLT	Branch if < Zero	?N⊕V=1	REL	2D	rr	-	-	-	-	-	-	-	-
BMI	Branch if Minus	?N=1	REL	2B	rr	-	-	-	-	-	-	-	-
BNE	Branch if Not = Zero	?Z=0	REL	26	rr	-	-	-	-	-	-	-	-
BPL	Branch if Plus	?N=0	REL	2A	rr	-	-	-	-	-	-	-	-
BRA	Branch Always	?1=1	REL	20	rr	-	-	-	-	-	-	-	-
BRCLR	Branch if Bit(s) Clear	?M•mm=0	DIR IND,X IND,Y	13 1F 18 1F	dd mm rr ff mm rr ff mm rr	-	-	-	-	-	-	-	-
BRN	Branch Never	?1=0	REL	21	rr	-	-	-	-	-	-	-	-
BRSET	Branch if Bit(s) Set	?(M)•mm=0	DIR IND,X IND,Y	12 1E 18 1E	dd mm rr ff mm rr ff mm rr	-	-	-	-	-	-	-	-
BSET	Set Bit(s)	M+mm⇨M	DIR IND,X IND,Y	14 1C 18 1C	dd mm ff mm ff mm	-	-	-	-	⇕	⇕	0	-
BSR	Branch to Subroutine		REL	8D	rr	-	-	-	-	-	-	-	-
BVC	Branch if Overflow Clear	?V=0	REL	28	rr	-	-	-	-	-	-	-	-
BVS	Branch if Overflow Set	?V=1	REL	29	rr	-	-	-	-	-	-	-	-
CBA	Compare A to B	A-B	INH	11		-	-	-	-	⇕	⇕	⇕	⇕
CLC	Clear Carry Bit	0⇨C	INH	0C		-	-	-	-	-	-	-	0
CLC	Clear Carry Bit	0⇨C	INH	0C		-	-	-	-	-	-	-	0
CLI	Clear Interrupt Mask	0⇨I	INH	0E		-	-	-	0	-	-	-	-
CLR	Clear Memory Byte	0⇨M	EXT IND,X IND,Y	7F 6F 18 6F	hh ll ff ff	-	-	-	-	0	1	0	0

CONJUNTO DE INSTRUCCIONES continuación...

Mnemonic	Operation	Boolean Expression	Addresing Mode for Operand	Machine Coding (hexadecimal)		Condition Codes							
				Opcode	Operand(s)	S	X	H	I	N	Z	V	C
CLRA		0⇨A	A INH	4F		-	-	-	-	0	1	0	0
CLRB		0⇨B	B INH	5F		-	-	-	-	0	1	0	0
CLV		0⇨V	INH	0A		-	-	-	-	-	-	0	-
CMPA	Compare A to Memory	A-M	A IMM	81	ii	-	-	-	-	⇕	⇕	⇕	⇕
			A DIR	91	dd								
			A EXT	B1	hh ll								
			A IND,X	A1	ff								
			A IND,Y	18 A1	ff								
CMPB	Compare B to Memory	B-M	B IMM	C1	ii	-	-	-	-	⇕	⇕	⇕	⇕
			B DIR	D1	dd								
			B EXT	F1	hh ll								
			B IND,X	E1	ff								
			B IND,Y	18 E1	ff								
COM	1's Complement Memory Byte	$FF-M⇨M	EXT	73	hh ll	-	-	-	-	⇕	⇕	0	1
			IND,X	63	ff								
			IND,Y	18 63	ff								
COMA	1's Complement A	$FF-A⇨A	A INH	43		-	-	-	-	⇕	⇕	0	1
COMB	1,s Complement B	$FF-B⇨B	B INH	53		-	-	-	-	⇕	⇕	0	1
CPD	Compare D to Memory	D-M:M+1	IMM	1A 83	jj kk	-	-	-	-	⇕	⇕	⇕	⇕
			DIR	1A 93	dd								
			EXT	1A B3	hh ll								
			IND,X	1A A3	ff								
			IND,Y	CD A3	ff								
CPD	Compare D to Memory	D-M:M+1	IMM	1A 83	jj kk	-	-	-	-	⇕	⇕	⇕	⇕
			DIR	1A 93	dd								
			EXT	1A B3	hh ll								
			IND,X	1A A3	ff								
			IND,Y	CD A3	ff								
CPX	Compare X to Memory	IX-M:M+1	IMM	8C	jj kk	-	-	-	-	⇕	⇕	⇕	⇕
			DIR	9C	dd								
			EXT	BC	hh ll								
			IND,X	AC	ff								
			IND,Y	CD AC	ff								

CONJUNTO DE INSTRUCCIONES continuación...

Mnemonic	Operation	Boolean Expression	Addresing Mode for Operand	Machine Coding (hexadecimal)		Condition Codes							
				Opcode	Operand(s)	S	X	H	I	N	Z	V	C
CPY	Compare Y to Memory	IY-M:M+1	IMM	18 8C	jj kk	-	-	-	-	⇕	⇕	⇕	⇕
			DIR	18 9C	dd								
			EXT	18 BC	hh ll								
			IND,X	1A AC	ff								
			IND,Y	18 AC	ff								
DAA	Decimal Adjust A	Adjust sum	INH	19		-	-	-	-	⇕	⇕	⇕	⇕
DEC	Decrement Memory Byte	M-1⇨M	EXT	7A	hh ll	-	-	-	-	⇕	⇕	⇕	-
			IND,X	6A	ff								
			IND,Y	18 6A	ff								
DECA	Decrement Accumulator A	A-1⇨A	A INH	4A		-	-	-	-	⇕	⇕	⇕	-
DECB	Decrement Accumulator B	B-1⇨B	B INH	5A		-	-	-	-	⇕	⇕	⇕	-
DES	Decrement Stack Pointer	SP-1⇨SP	INH	34		-	-	-	-	-	-	-	-
LDD	Load Double Accumulator D	M⇨A,M+1⇨B	IMM	CC	jj kk	-	-	-	-	⇕	⇕	0	-
			DIR	DC	dd								
			EXT	FC	hh ll								
			IND,X	EC	ff								
			IND,Y	18 EC	ff								
LDS	Load Stack Pointer	M:M+1⇨SP	IMM	8E	jj kk	-	-	-	-	⇕	⇕	0	-
			DIR	9E	dd								
			EXT	BE	hh ll								
			IND,X	AE	ff								
			IND,Y	18 AE	ff								
LDX	Load Index Register X	M:M+1⇨IX	IMM	CE	jj kk	-	-	-	-	⇕	⇕	0	-
			DIR	DE	dd								
			EXT	FE	hh ll								
			IND,X	EE	ff								
			IND,Y	CD EE	ff								
LDY	Load Index Register Y	M:M+1⇨IY	IMM	18 CE	jj kk	-	-	-	-	⇕	⇕	0	-
			DIR	18 DE	dd								
			EXT	18 FE	hh ll								
			IND,X	1A EE	ff								
			IND,Y	18 EE	ff								

CONJUNTO DE INSTRUCCIONES continuación...

Mnemonic	Operation	Boolean Expression	Addresing Mode for Operand	Machine Coding (hexadecimal)		Condition Codes							
				Opcode	Operand(s)	S	X	H	I	N	Z	V	C
LSL	Logical Shift Left		EXT	78	hh ll	-	-	-	-	⇕	⇕	⇕	⇕
			IND,X	68	ff								
			IND,Y	18 68	ff								
			A INH	48									
			B INH	58									
LSLD	Logical Shift Left Double		INH	05		-	-	-	-	⇕	⇕	⇕	⇕
LSR			EXT	74	hh ll	-	-	-	-	0	⇕	⇕	⇕
			IND,X	64	ff								
			IND,Y	18 64	ff								
LSRA			A INH	44									
LSRB			B INH	54									
LSRD	Logical Shift Left Double		INH	04		-	-	-	-	0	⇕	⇕	⇕
MUL	Multiply 8 by 8	AxB⇨D	INH	3D		-	-	-	-	-	-	-	⇕
NEG	2's Complement Memory Byte	0-M⇨M	EXT	70	hh ll	-	-	-	-	⇕	⇕	⇕	⇕
			IND,X	60	ff								
			IND,Y	18 60	ff								
NEGA	2's Complement A	0-A⇨A	A INH	40		-	-	-	-	⇕	⇕	⇕	⇕
NEGB	2,s Complement B	0-B⇨B	B INH	50		-	-	-	-	⇕	⇕	⇕	⇕
NOP	No Operation	No Operation	INH	01		-	-	-	-	-	-	-	-
ORAA	OR Accumulator A (inclusive)	A+M⇨A	A IMM	8A	ii	-	-	-	-	⇕	⇕	0	-
			A DIR	9A	dd								
			A EXT	BA	hh ll								
			A IND,X	AA	ff								
			A IND,Y	18 AA	ff								
ORAB	OR Accumulator B (Inclusive)	B+M⇨B	B IMM	CA	ii	-	-	-	-	⇕	⇕	0	-
			B DIR	DA	dd								
			B EXT	FA	hh ll								
			B IND,X	EA	ff								
			B IND,Y	18 EA	ff								
PSHA	Push A onto Stack	A⇨Stk,SP=SP-1	A INH	36		-	-	-	-	-	-	-	-
PSHB	Push B onto Stack	B⇨Stk,SP=SP-1	B INH	37		-	-	-	-	-	-	-	-
PSHX	Push X onto Stack (Lo First)	IX⇨Stk,SP=SP-2	INH	3C		-	-	-	-	-	-	-	-

CONJUNTO DE INSTRUCCIONES continuación...

Mnemonic	Operation	Boolean Expression	Addresing Mode for Operand	Machine Coding (hexadecimal)		Condition Codes							
				Opcode	Operand(s)	S	X	H	I	N	Z	V	C
PSHY	Push Y onto Stack (Lo First)	IY⇨Stk,SP=SP-2	INH	18 3C		-	-	-	-	-	-	-	-
PULA	Pull A from Stack	SP=SP+1,A⇦Stk	A INH	32		-	-	-	-	-	-	-	-
PULB	Pull B from Stack	SP=SP+1,B⇦Stk	B INH	33		-	-	-	-	-	-	-	-
PULX	Pull X from Stack (Hi First)	SP=SP+2,IX⇦Stk	INH	38		-	-	-	-	-	-	-	-
PULY	Pull Y from Stack (Hi First)	SP=SP+2,IY⇦Stk	INH	18 38		-	-	-	-	-	-	-	-
ROL	Rotate Left		EXT	79	hh ll	-	-	-	-	⇕	⇕	⇕	⇕
			IND,X	69	ff								
			IND,Y	18 69	ff								
ROLA			A INH	49									
ROLB			B INH	59									
ROR	Rotate Right		EXT	76	hh ll	-	-	-	-	⇕	⇕	⇕	⇕
			IND,X	66	ff								
			IND,Y	18 66	ff								
RORA			A INH	46									
RORB			B INH	56									
RTI	Return from Interrup		INH	3B		⇕	⇕	⇕	⇕	⇕	⇕	⇕	⇕
RTS	Return from Subroutine		INH	39		-	-	-	-	-	-	-	-
SBA	Subtract B from A	A-B⇨A	INH	10		-	-	-	-	⇕	⇕	⇕	⇕
SBCA	Subtract with Carry From A	A-M-C⇨A	A IMM	82	ii	-	-	-	-	⇕	⇕	⇕	⇕
			A DIR	92	dd								
			A EXT	B2	hh ll								
			A IND,X	A2	ff								
			A IND,Y	18 A2	ff								
SBCB	Subtract with Carry From B	B-M-C⇨B	B IMM	C2	ii	-	-	-	-	⇕	⇕	⇕	⇕
			B DIR	D2	dd								
			B EXT	F2	hh ll								
			B IND,X	E2	ff								
			B IND,Y	18 E2	ff								
SEC	Set Carry	1⇨C	INH	0D		-	-	-	-	-	-	-	1
SEI	Set Interrupt Mask	1⇨I	INH	OF		-	-	-	1	-	-	-	-
SEV	Set Overflow Flag	1⇨V	INH	OB		-	-	-	-	-	-	1	-
STAA	Store Accumulator A	A⇨M	A DIR	97	dd	-	-	-	-	⇕	⇕	0	-
			A EXT	B7	hh ll								
			A IND,X	A7	ff								
			A IND,Y	18 A7	ff								

CONJUNTO DE INSTRUCCIONES continuación...

Mnemonic	Operation	Boolean Expression	Addresing Mode for Operand	Machine Coding (hexadecimal)		Condition Codes							
				Opcode	Operand(s)	S	X	H	I	N	Z	V	C
STAB	Store Accumulator B	B⇨M	B DIR	D7	dd	-	-	-	-	⇕	⇕	0	-
			B EXT	F7	hh ll								
			B IND,X	E7	ff								
			B IND,Y	18 E7	ff								
STD	Store Accumulator D	A⇨M,B⇨M+1	DIR	DD	dd	-	-	-	-	⇕	⇕	0	-
			EXT	FD	hh ll								
			IND,X	ED	ff								
			IND,Y	18 ED	ff								
STOP	Stop Internal Clocks		INH	CF		-	-	-	-	-	-	-	-
STS	Store Stack Pointer	SP⇨M:M+1	DIR	9F	dd	-	-	-	-	⇕	⇕	0	-
			EXT	BF	hh ll								
			IND,X	AF	ff								
			IND,Y	18 AF	ff								
STX	Store Index Register X	IX⇨M:M+1	DIR	DF	dd	-	-	-	-	⇕	⇕	0	-
			EXT	FF	hh ll								
			IND,X	EF	ff								
			IND,Y	CD EF	ff								
STY	Store Index Register Y	IY⇨M:M+1	DIR	18 DF	dd	-	-	-	-	⇕	⇕	0	-
			EXT	18 FF	hh ll								
			IND,X	1A EF	ff								
			IND,Y	18 EF	ff								
SUBA	Subtract Memory from A	A-M⇨A	A IMM	80	ii	-	-	-	-	⇕	⇕	⇕	⇕
			A DIR	90	dd								
			A EXT	B0	hh ll								
			A IND,X	A0	ff								
			A IND,Y	18 A0	ff								
SUBB	Subtract Memory from B	B-M⇨B	B IMM	C0	ii	-	-	-	-	⇕	⇕	⇕	⇕
			B DIR	D0	dd								
			B EXT	F0	hh ll								
			B IND,X	E0	ff								
			B IND,Y	18 E0	ff								

CONJUNTO DE INSTRUCCIONES continuación...

Mnemonic	Operation	Boolean Expression	Addresing Mode for Operand	Machine Coding (hexadecimal)		Condition Codes							
				Opcode	Operand(s)	S	X	H	I	N	Z	V	C
SUBD	Subtract Memory from D	D-M:M+1⇨D	IMM DIR EXT IND,X IND,Y	83 93 B3 A3 18 A3	jj kk dd hh ll ff ff	-	-	-	-	⇕	⇕	⇕	⇕
SWI	Software Interrupt		INH	3F		-	-	-	1	-	-	-	-
TAB	Transfer A to B	A⇨B	INH	16		-	-	-	-	⇕	⇕	0	-
TAP	Transfer A to CCR	A⇨CCR	INH	06		⇕	⇕	⇕	⇕	⇕	⇕	⇕	⇕
TBA	Transfer B to A	B⇨A	INH	17		-	-	-	-	⇕	⇕	0	-
TEST	TEST (Only in Test Modes)	Address bus counts	INH	00		-	-	-	-	-	-	-	-
TPA	Transfer CCR to A	CCR⇨A	INH	07		-	-	-	-	-	-	-	-
TST	Test for Zero or Minus	M-0	EXT IND,X IND,Y	7D 6D 18 6D	hh ll ff ff	-	-	-	-	⇕	⇕	0	0
TSTA		A-0	A INH	4D		-	-	-	-	⇕	⇕	0	0
TSTB		B-0	B INH	5D		-	-	-	-	⇕	⇕	0	0
TSX	Transfer Stack Pointer to X	SP+1⇨IX	INH	30		-	-	-	-	-	-	-	-
TSY	Transfer Stack Pointer to Y	SP+1⇨IY	INH	18 30		-	-	-	-	-	-	-	-
TXS	Transfer X to Stack Pointer	IX-1⇨SP	INH	35		-	-	-	-	-	-	-	-
TYS	Transfer Y tp Stack Pointer	IY-1⇨SP	INH	18 35		-	-	-	-	-	-	-	-
WAI	Wait for Interrupt	Stack Regs&Wait	INH	3E		-	-	-	-	-	-	-	-
XGDX	Exchange D with X	IX⇨D,D⇨X	INH	8F		-	-	-	-	-	-	-	-
XGDY	Exchange D with Y	IY⇨D,D⇨IY	INH	18 8F		-	-	-	-	-	-	-	-

VIII.4. OTROS ASPECTOS.

Como este trabajo está basado en el uso de información y equipo de Motorola, traté de evitar conflictos legales con respecto a propiedad intelectual y otros relacionados, para lo cual se enviaron cartas solicitando permiso de su uso.

Como hasta la fecha no he recibido contestación, se sugiere que los posibles usuarios de estos apuntes revisen ese aspecto, e incluyo a continuación copia de las cartas enviadas.

TELEFAX

Monterrey, N.L., Abril 22 de 1994

MOTOROLA
Fax: (5) 282 2864

At'n: Ing. Arturo Muñoz

De acuerdo con nuestra conversación telefónica de hoy, le agradeceré su ayuda para conseguir de Motorola la autorización para el uso de la información de su M68HC11 Micro Controller Manual en la elaboración de mi tésis titulada "CURSO DE MICRO CONTROLADORES " para la obtención del Grado de Maestría en Ciencias en la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autonóma de Nuevo León.

Aradezco de antemano su atención y espero su respuesta.

A t e n t a m e n t e

René Briones

Ing. René Briones L.

Fax: (8) 339 2721
Tel: (8) 337 8333
Dirección:: Calle 7 No. 2133
Col. La Quinta
Cd. Guadalupe N.L:

TELEFAX

Monterrey, N.L., México; March 23, 1994

MOTOROLA
Fax: (512) 505 8811

Dear Sirs:

I will appreciate you help me to know the right people on Motorola Corporation in order to get authorization to copyrights some information from your M68HC11 Micro controller Manuals and use it to elaborate my thesis for Master Degree "CURSO DE MICROCONTROLADORES" from Universidad Autónoma de Nuevo León, Facultad de Ingeniería Mecánica y Eléctrica.

Thanks a lot

René Briones L.

Ing. René Briones L.
Phone & Fax: 01152 (8) 339 2721
Address: Calle 7 No. 2133
Col. La Quinta
Cd. Guadalupe N.L.

TX REPORT

Apr.22 1994 4:49PM

RECEIVING FACSIMILE	MODE	PAGES	START TIME	USAGE TIME	RESULT
MOTOROLA DE MEXI	TX	01	Apr.22 4:47PM	00'24	OK

TX REPORT

Mar.23 1994 11:31AM

RECEIVING FACSIMILE	MODE	PAGES	START TIME	USAGE TIME	RESULT
512 3328811	TX	01	Mar.23 11:31AM	00'33	OK