MICROCONTROLADORES: Arquitectura básica del 68HC11

Organización del 68hc11

El C.I. 68HC11 puede operar como un sistema completo ya que dispone de los elementos necesarios para diseñar una máquina programada de tipo Von Newmann; Unidad de Proceso (controlador y operador de datos), memoria (RAM, ROM/PROM/EPROM, y EEPROM) y puertos de entrada-salida (I/O). Cada uno de estos módulos se presentan en el siguiente diagrama de bloques

Desde un punto de vista de tecnología electrónica, este CI se basa en puertas HCMOS, y de forma general utiliza un reloj de hasta 2 MHz. La tecnología CMOS permite bajos consumos. Se puede elegir diferentes tipos de configuraciones, tanto en tamaño y tipo de memoria (ROM, OTP EPROM, EEPROM…) como en el tipo de puertos de entrada salida. En los mismos se incluyen

- Convertidores analógico-digitales de 8 bits de resolución.
- Contadores (timers) con una gran flexibilidad de operación
- Puertos paralelo de entrada/salida.
- Puertos serie (asíncronos y síncronos)

esto es, este sistema permite implementar las operaciones básicas de cualquier sistema programado.

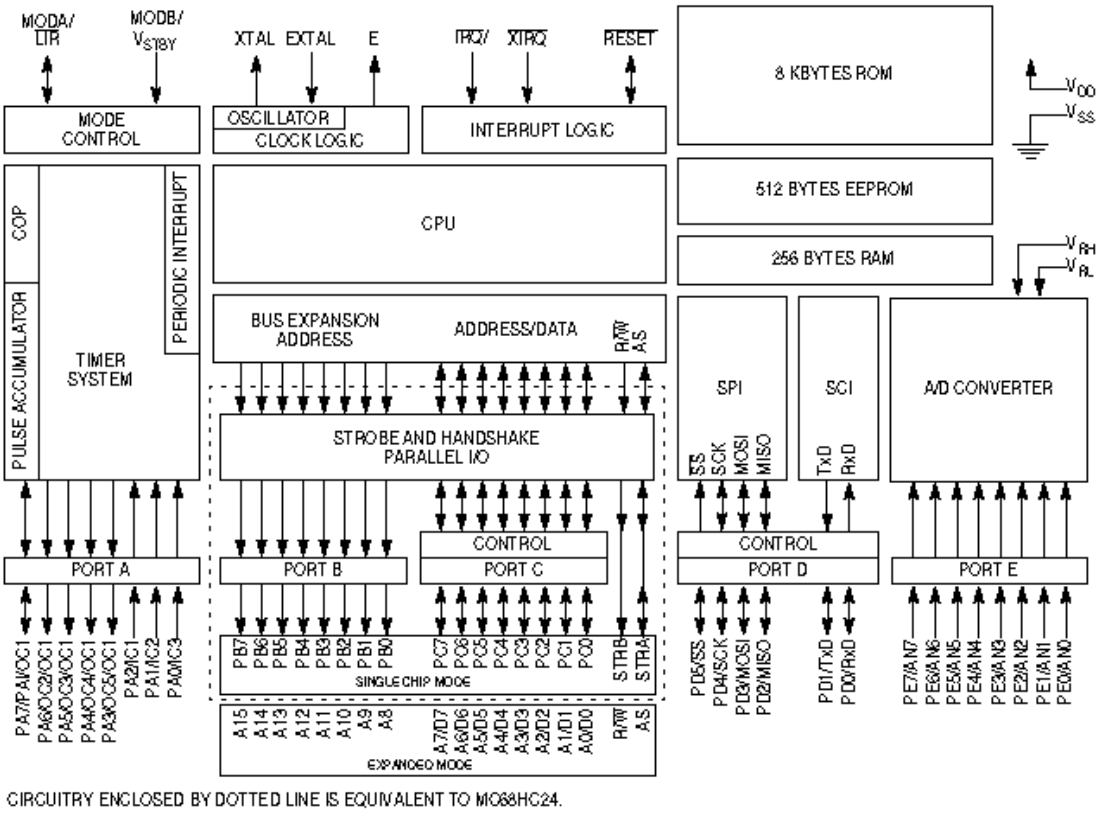


Fig. 1 Diagrama de bloques del 68HC11A8 (Motorola ®)

Los modos de funcionamiento básico

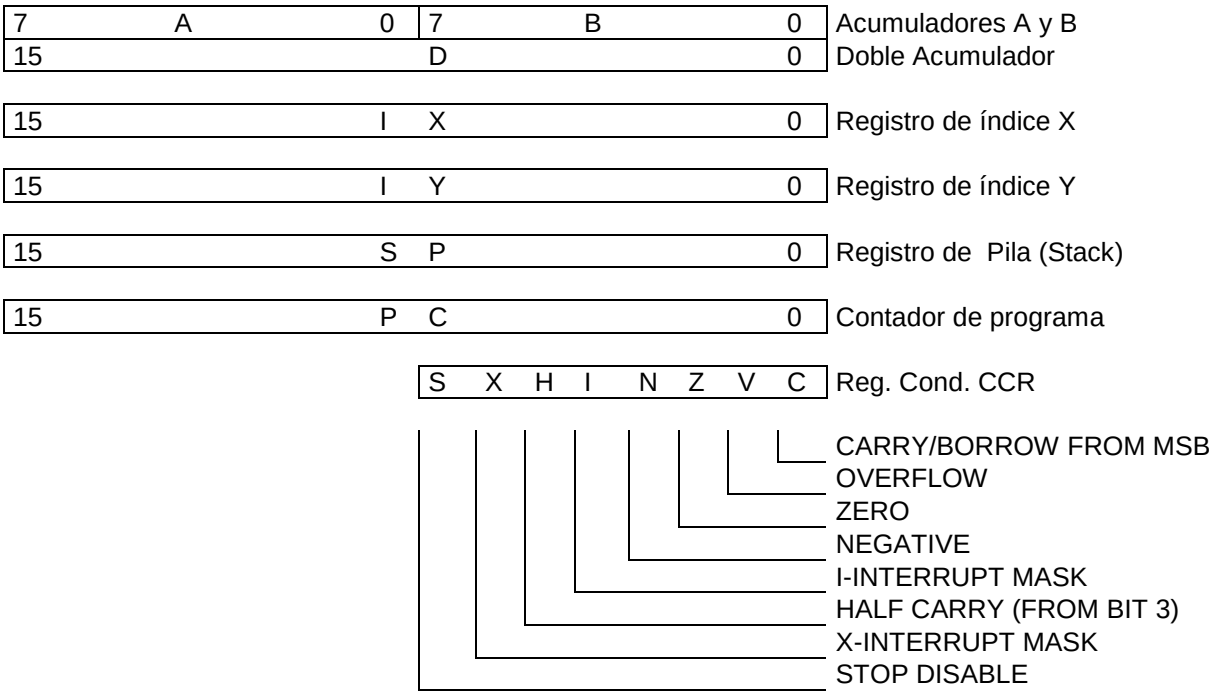
Dependiendo del tipo de la aplicación se puede elegir entre los siguientes modos de funcionamiento:

- **Single chip:** Todo los recursos necesarios se encuentran en el propio circuito integrado. Es el recomendado para aplicaciones en las que el programa de control se puede grabar en la memoria interna (ROM) y las necesidades de almacenamiento de variables temporales no supera el tamaño de la memoria RAM. Se consiguen una mayor fiabilidad en el "hardware", y simplicidad en el diseño de la placa, además de disponer de todos los puertos
- **Modo Expandido:** Utilizando dos puertos de entrada salida, se dispone del bus de direcciones y de datos de, de forma que se puede utilizar zonas del mapa de memoria que en el modo "single chip" no se usan. La técnica de acceso a los dos buses se basa en la multiplexación de señales (LSByte del bus de direcciones y el bus de datos) por un puerto
- **Bootstrap**: Permite una transferencia rápida de datos entre un sistema externo y la memoria interna del HC11. En este modo los vectores de interrupción se encuentran en otras posiciones de memoria.
- **Test:** Utilizado únicamente en factoría

Registros de la CPU

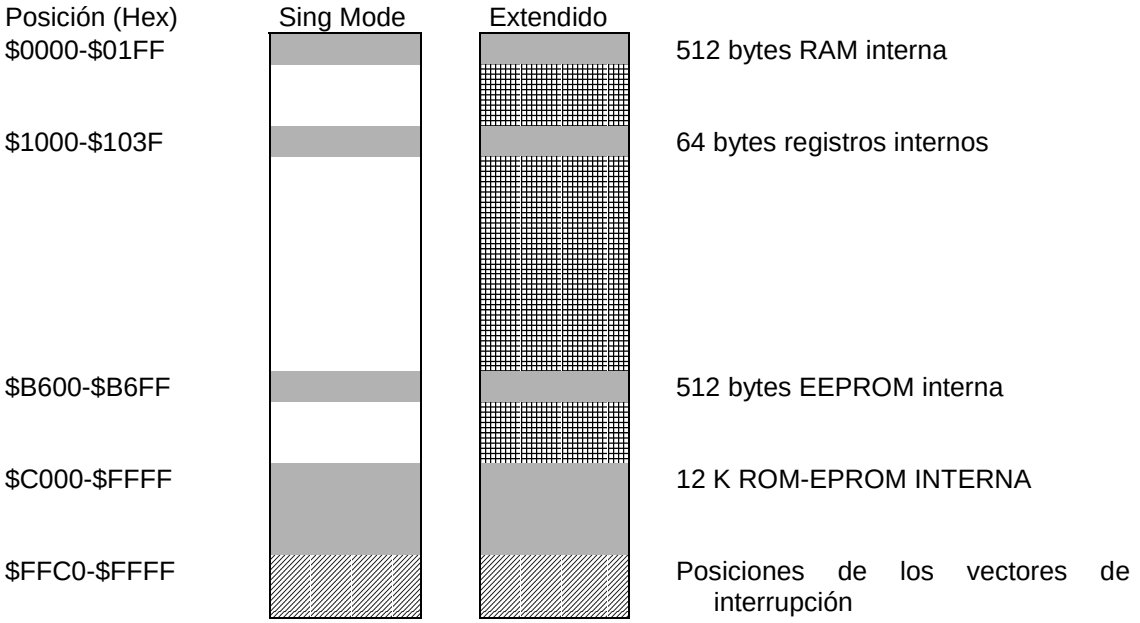
Los registros de la CPU del HC11 son dos acumuladores A y B, de 8 bits que pueden operar como un único registro de 16 bits (D). Dos registros de índice IX, IY un registro de ountero de pila (Stack) y un contador de programa, todos ellos de 16 bits. El registro de Condición de código (CCR) permite monitorizar el resultado de algunas operaciones de la CPU.

Seguidamente se muestran de forma esquemática, para más detalles nos remitimos al reference manual de estos microcontroladores.



Mapa de memoria del 68HC(7)11E9

El mapa de memoria es una descripción gráfica que permite mostrar que rangos de direcciones están utilizadas por periféricos de I/O, memorias RAM, EEPROM, etc. En función del modo de operación el mapa varía ligeramente. Seguidamente mostramos el mapa de memoria del 68HC11 para los modos mas usuales de operación. En el modo extendido, la zona cuadriculada indica la posibilidad de posicionar perifericos externos.



Los registros de configuración y de datos de los bloques internos se encuentran en las posiciones $1000 a $103F. En los siguientes capítulos se describirán con más detalle.

Modos de direccionamiento del MC68HC11.

El microcontrolador MC68HC11 tiene seis modos de direccionamiento diferentes que permiten acceder a posiciones del mapa de memoria y a los registros internos. No todas las instrucciones pueden utilizarse con todos los modos. Entonces el modo de direccionamiento es una parte implícita de cada instrucción

Direccionamiento directo (DIR) y extendido (EXT)

En el primero se utiliza el bit menos significativo de la dirección efectiva. Por este motivo sólo se puede direccionar 256 palabra de la página cero "direct page" (0000-00FF). Un ejemplo del mismo se refleja en el siguiente código

```
      SUBD    CAT       ;FWD REF TO CAT
CAT   EQU     $12       ;DEFINE CAT=$12
      SUBD    CAT       ;BKWD REF TO CAT
      CLR     CAT       ;EXTENDED ONLY
```

A partir del mismo se genera el código máquina indicado en la primera columna de la siguiente tabla. En cada fila, el primer byte indica el código de operación mientras que los siguientes indican la dirección

Código máquina	Etiqueta	Instrucción	Operando	Comentarios
B3 00 12		SUBD	CAT	FWD REF TO CAT
	CAT	EQU	$12	DEFINE CAT=$12
93 12		SUBD	CAT	BKWD REF TO CAT
7F 00 12		CLR	CAT	EXTENDED ONLY

En el direccionamiento extendido se utiliza toda la palabra correspondiente a la posición de memoria direccionada, por lo que se puede acceder a cualquiera de las 64 K posiciones.

Direccionamiento Indexado (INX) (INY)

En el mismo la información almacenada en el registro de índice X o Y es utilizada para conocer la dirección real a la que se accede. A este valor se le añade un "offset" que es una palabra de 8 bits. Los siguientes ejemplos ilustran el resultado para diferentes condiciones del offset.

```
        ADDD    X              ;EA=(X)
        ADDD    ,X             ;EA=(X)
        ADDD    0,X            ;EA=(X)
        ADDD    4,X            ;EA=(X)+4
CAT     EQU     7              ;DEFINE CAT=7
        ADDD    CAT,X          ;EA=(X)+7
        ADDD    $22,X          ;EA=(X)+$22
        ADDD    CAT*8/2+6,X    ;EA=(X)+(CAT*8•2+6)
```

Código máquina	Etiqueta	Instrucción	Operando	Comentarios
E3 00		ADDD	X	EA=(X)
E3 00		ADDD	,X	EA=(X)
E3 00		ADDD	0,X	EA=(X)
E3 04		ADDD	4,X	EA=(X)+4
	CAT	EQU	7	DEFINE CAT=7
E3 07		ADDD	CAT,X	EA=(X)+7
E3 22		ADDD	$22,X	EA=(X)+$22
E3 22		ADDD	CAT*8/2+6,X	EA=(X)+(CAT*8÷2+6)

Este modo es muy útil para acceder a tablas de valores de forma iterativa.

Direccionamiento Inmediato (IMM)

En el mismo una parte de la instrucción es el byte de dato, y no la dirección del mismo. El carácter # se utiliza en el lenguaje ensamblador para identificar el direccionamiento inmediato. Observar en el siguiente ejemplo que el código máquina incluye el valor hexadecimal del operando.

Código máquina	Etiqueta	Instrucción	Operando	Comentarios
	CAT	EQU	7	ETIQUETA CAT
		ORG	$1000	POSICION DEL PROGRAMA
	REGS	EQU	*	ADDR(REGS) IS $1000
86 16		LDAA	#22	DECIMAL 22 ⇒ ACCA ($16)
C8 34		EORB	#$34	XOR ($34,ACCB) ⇒ ACCB
81 24		CMPA	#%100100	VALOR BINARIO
86 07		LDAA	#CAT	7⇒ ACCA
CC 12 34		LDD	#$1234	
CC 00 07		LDD	#7	7⇒ ACCA:ACCB
86 12		LDAA	#@22	OCTAL
86 41		LDAA	#'A	ASCII
CE 10 00		LDX	#REGS	ADDR(REGS) ⇒ X

Direccionamiento Inherente (INH)

En este caso, la CPU "sabe" a que posición de memoria o registro debe ir. Este modo está asociado a instrucciones muy concretas, que manipulan registros internos de la CPU, interrupciones, etc.

Código máquina	Etiqueta	Instrucción	Operando	Comentarios
1B		ABA		A+B⇒A
5C		INCB		B+1⇒B
08		INX		X+1⇒X

Direccionamiento Relativo (REL)

Se utiliza únicamente para las operaciones de bifurcación (salto condicional) y es necesario un byte de "offset", que introduce un desplazamiento en el contador de programa. Debido a que este desplazamiento puede ser positivo o negativo el offset esta codificado como una palabra en complemento a 2. Seguidamente mostramos algunos ejemplos

Código máquina	Etiqueta	Instrucción	Operando	Comentarios
20 00	LAB1	BRA	LAB2	SALTO CORTO
22 FC	LAB2	BHI	LAB1	SALTO CORTO
24 04		BCC	LBCC	SALTO L-O-N-G
27 FE	LAB3	BEQ	LAB3	BRANCH TO SELF
27 FE		BEQ	*	"*" SIGNIFICA "AQUÍ"
7E 10 00	LBCC	JMP	$1000	
8D F7		BSR	LAB3	

En la siguiente tabla se resume los tipos de direccionamiento.

Modo	Ejemplo	Uso
Inherente	SWI	Mejorar la eficiencia
Registros	INCA	Mejorar la eficiencia
Inmediato	LDAA #12	Inicializar registros, generar constantes…
Directo	LDAA alfa	Almacenar datos globales
Extendido	LDAA alfa	Acceder a cualquier posición de memoria
Indexado	LDAA 5,X	Direccionamiento de matrices
Relativo	BRA alfa	Independizar el programa de la posición.

Conjunto de instrucciones

No se pretende dar una información exhaustiva de las instrucciones, ya que existe suficiente documentación técnica al respecto (ej 68hc11 reference manual). Es más, la única forma de conocer realmente el conjunto de instrucciones es… programando, no obstante seguidamente ofrecemos algunos aspectos de carácter general que resultan de interés

Existen seis tipos diferentes de instrucciones en una máquina de tipo Von Neumann. Las tablas que presentamos en cada se han obtenido de los manuales de referencia del 68HC11.

Instrucciones de movimiento

Permite transferir información de un registro a la memoria, o entre registros, etc. Típicamente una tercera parte de las instrucciones de un programa suelen ser de movimiento. Seguidamente se presentan las instrucciones de movimiento del HC11 junto con el tipo de direccionamiento que soportan

Función	Mnemonic	IMM	DIR	EXT	INDX	INDY	INH
Clear Memory Byte	CLR			X	X	X	
Clear Accumulator A	CLRA						X
Clear Accumulator B	CLRB X						X
Load Accumulator A	LDAA	X	X	X	X	X	
Load Accumulator B	LDAB	X	X	X	X	X	
Load Double Accumulator D	LDD	X	X	X	X	X	
Pull A from Stack	PULA						X
Pull B from Stack	PULB						X
Push A onto Stack	PSHA						X
Push B onto Stack	PSHB						X
Store Accumulator A	STAA	X	X	X	X	X	
Store Accumulator B	STAB	X	X	X	X	X	
Store Double Accumulator D	STD	X	X	X	X	X	
Transfer A to B	TAB						X
Transfer A to CCR	TAP						X
Transfer B to A	TBA						X
Transfer CCR to A	TPA						X
Exchange D with X	XGDX						X
Exchange D with Y	EGDY						X

Operaciones Aritméticas.

Permiten sumar, restar multiplicar o dividir valores del acumulador con una palabra tomada de la memoria. En el caso que nos ocupa existe la posibilidad de realizar operaciones con palabras de 8 y 16 bits. El conjunto de operaciones es

Función	Mnemo.	IMM	DIR	EXT	INDX	INDY	INH
Add Accumulators	ABA						X
Add Accumulator B to X	ABX						X
Add Accumulator B to Y	ABY						X
Add with Carry to A	ADCA	X	X	X	X	X	
Add with Carry to B	ADCB	X	X	X	X	X	
Add Memory to A	ADDA	X	X	X	X	X	
Add Memory to B	ADDB	X	X	X	X	X	
Add Memory to D (16 Bit)	ADDD	X	X	X	X	X	
Compare A to B	CBA						X
Compare A to Memory	CMPA	X	X	X	X	X	
Compare B to Memory	CMPB	X	X	X	X	X	
Compare D to Memory (16 Bit)	CPD	X	X	X	X	X	
Decimal Adjust A (for BCD)	DAA						X
Decrement Memory Byte	DEC				X	X	X
Decrement Accumulator A	DECA						X
Decrement Accumulator B	DECB						X
Increment Memory Byte	INC				X	X	X
Increment Accumulator B	INCB						X
Two's Complement Memory Byte	NEG				X	X	X
Two's Complement Accumulator A	NEGA						
Two's Complement Accumulator B	NEGB						
Subtract with Carry from A	SBCA	X	X	X	X	X	
Subtract with Carry from B	SBCB	X	X	X	X	X	
Subtract Memory from A	SUBA	X	X	X	X	X	
Subtract Memory from B	SUBB	X	X	X	X	X	
Subtract Memory from D (16 Bit)	SUBD	X	X	X	X	X	
Test for Zero or Minus	TST				X	X	X
Test for Zero or Minus A	TSTA						X
Test for Zero or Minus B	TSTB						X
Multiply (A X B $\Rightarrow$ D)	MUL						X
Fractional Divide (D $\div$ X $\Rightarrow$ X; r $\Rightarrow$ D)	FDIV						X
Integer Divide (D $\div$ X $\Rightarrow$ X; r $\Rightarrow$ D)	IDIV						X

Operaciones lógicas

Son instrucciones muy útiles en el control de registros, tanto de la unidad de proceso como de periféricos e implementan las operaciones booleanas básicas (AND, OR, EXOR, NOR) y complementación de números (complemento a dos)

Función	Mnemonic	IMM	DIR	EXT	INDX	INDY	INH
AND A with Memory	ANDA	X	X	X	X	X	
AND B with Memory	ANDB	X	X	X	X	X	
Bit(s) Test A with Memory	BITA	X	X	X	X	X	
Bit(s) Test B with Memory	BITB	X	X	X	X	X	
One's Complement Memory Byte	COM	X	X	X			
One's Complement A	COMA						X
One's Complement B	COMB						X
OR A with Memory (Exclusive)	EORA	X	X	X	X	X	
OR B with Memory (Exclusive)	EORB	X	X	X	X	X	
OR A with Memory (Inclusive)	ORAA	X	X	X	X	X	
OR B with Memory (Inclusive)	ORAB	X	X	X	X	X	

Operaciones de edición

Permiten reordenar los bits; esto es, permite desplazar o rotar los bits del acumulador o de una palabra de memoria

Función	Mnemonic	IMM	DM	EXT	INDX	INDY	INH
Arithmetic Shift Left Memory	ASL			X	X	X	
Arithmetic Shift Left A	ASLA						X
Arithmetic Shift Left B	ASLB						X
Arithmetic Shift Left Double	ASLD						X
Arithmetic Shift Right Memory	ASR			X	X	X	
Arithmetic Shift Right A	ASRA						X
Arithmetic Shift Right B	ASRB						X
(Logical Shift Left Memory)	(LSL)			X	X	X	
(Logical Shift Left A)	(LSLA)						X
(Logical Shift Left B)	(LSLB)						X
(Logical Shift Left Double)	(LSLD)						X
Logical Shift Right Memory	LSR			X	X	X	
Logical Shift Right A	LSRA						X
Logical Shift Right B	LSRB						X
Logical Shift Right D	LSRD						X
Rotate Left Memory	ROL			X	X	X	
Rotate Left A	ROLA						X
Rotate Left B	ROLB						X
Rotate Right Memory	ROR			X	X	X	
Rotate Right A	RORA						X
Rotate Right B	RORB						X

Instrucciones de control.

Son aquellas que permiten modificar el flujo de programa; esto es, afectan, de alguna forma al contador de programa Existen varias clasificaciones. Una de ellas contempla cinco subgrupos; a saber:

1.-**Saltos condicionales** (Branch): permiten ir a posiciones de memoria que se encuentran a una distancia comprendida entre 127/-128 posiciones de la que tiene el contador de programa. En la mayoría de las ocasiones es más que suficiente, pero si se desea saltar a una posición de memoria no comprendida entre estos valores se debe utilizar las siguientes instrucciones
2.-**Saltos** (jump): Permite ir a cualquiera de las 64 K posiciones de memoria
3.-**Llamadas a subrutinas y retorno**: Las subrutinas permite dividir el programa de forma que facilita el desarrollo del mismo, aumenta su fiabilidad, etc.
4.-**Manipulación de interrupciones.**
5.-**Otras instrucciones**.

Función	Mnemonic	REL	DIR	EXT	IND	INDY	INH	Comentario
Bifurcaciones								
Branch if Carry Clear	BCC	X						C = 0 ?
Branch if Carry Set	BCS	X						C = 1 ?
Branch if Equal Zero	BEQ	X						Z = 1 ?
Branch if Greater Than or Equal	BGE	X						Signed ≥
Branch if Greater Than	BGT	X						Signed >
Branch if Higher	BHI	X						Unsigned >
Branch if Higher or Same (same as BCC)	BHS	X						Unsigned ≥
Branch if Less Than or Equal	BLE	X						Signed ≤
Branch if Lower (same as BCS)	BLO	X						Unsigned <
Branch if Lower or Same	BLS	X						Unsigned ≤
Branch if Less Than	BLT	X						Signed <
Branch if Minus	BMI	X						N= 1 ?
Branch if Not Equal	BNE	X						Z = 0 ?
Branch if Plus	BPL	X						N = 0 ?
Branch if Bit(s) Clear in Memory Byte	BRCLR		X		X	X		Bit Manip.
Branch Never	BRN	X						3-cycle NOP
Branch if Bit(s) Set in Memory Byte	BRSET		X		X	X		
Bit Manipulation Branch if Overflow Clear	BVC	X						V = 0 ?
Branch if Overflow Set	BVS	X						V = 1 ?
Salto								
Jump	JMP		X	X	X	X		
Llamadas a subrutinas								
Branch to Subroutine	BSR	X						
Jump to Subroutine	JSR		X	X	X	X		
Return from Subroutine	RTS						X	
Gestión de interrupciones								
Return from Interrupt	RTI						X	
Software Interrupt	SWI						X	
Wait for Interrupt	WAI						X	

Estas instrucciones de decisión se pueden agrupar según el bit del registro de control (flag) afectado

No Flag	Carry	Zero	Sign	Overflow	Arithmetic	Logical
BRA	BCC, BCS	BEQ, BNE	BMI, BPL	BVS, BVC	BGE, BGT	BHI, BHS
JMP					BLE, BLT	BLO, BLS

Otros comandos

Son funciones de difícil clasificación, que se suelen incluir en un grupo denominado "miscelanea"

Función	Mnemonic	INH
No Operation (2-cycle delay)	NOP	X
Stop Clocks	STOP	X
Test	TEST	X

Desarrollo de programas.

Para poder implementar aplicaciones específicas es necesario utilizar herramientas que permitan traducir los diagramas de flujo que diseñamos a un lenguaje que sea almacenado y entendido por el microprocesador. Existen varias técnicas que hay que trasladar de la asignatura de "arquitectura de computadores" y adaptarlas a la familia de Motorola ® y al entorno de programación de la misma.

Estructuras de datos

Las estructuras de datos son una parte tan importante en el desarrollo como el propio programa. De la forma de ordenar los mismos dependen factores como la capacidad de almacenamiento, la eficiencia estática y dinámica (velocidad de acceso). La información que gestiona un sistema programado se puede organizar en **tres estructuras diferentes**, en función del tipo de usuario que la analiza

- **Estructura de información**: Es la que se debe presentar al usuario final para que la Operación del sistema sea óptimo.
- **Estructura de datos**: Es de la que dispone el programador para desarrollar el Software del sistema e indica la forma de acceder a la misma.
- **Estructura de almacenamiento**: Es el modo en el que se ordena la información en la memoria del sistema.

Una forma de gestionar de forma eficaz las estructuras de datos se basa en el uso de directivas de ensamblador. Son instrucciones que se parecen a las del propio microcontrolador, pero que generan código máquina para realizar tareas de asignación de zonas de memoria, indicación de comienzo y fin de programa, etc… Las directivas dependen del compilador utilizado, algunas son las siguientes.

Directiva	Significado
NAM N	Declara que el nombre del programa es N
END	Termina el programa
ORG N	Marca el comienzo del programa en la posición N
L: RMB N	(Reserve memory byte) Coloca N palabras y designa L como etiqueta de la primera palabra
L: EQU N	Asigna a la etiqueta L el valor N
L: FCB alfa, beta	(Form constant byte)Reserva un byte a cada una de las variables, etiquetas o números que aparecen separadas por comas. Asigna la etiqueta L al primer dato
L: FDB alfa, beta	(Form double byte) Reserva dos bytes a cada una de las estiquetas, constantes, caracteres o números que aparecen separados por comas. Asigna la etiqueta L al primer valor.

Estructuras de datos indexados.

Estas estructuras incluyen vectores, listas, matrices y tablas.

Un vector es una secuencia de elementos (números de la misma precisión), cada uno de ellos asociado a un índice *i* (habitualmente el primer elemento se asocia con el valor *cero*)

Una lista tiene la misma estructura que un vector, pero sus elementos pueden ser de diferente precisión, caracteres, palabras de código, etc.

```
.org     $100
v:      fcb     31,17,10
u:      fdb     31,17,10
;para colocar el i-esimo elemento del vector v en el acumulador B,
;suponiendo que i se encuentra en el acumulador B es necesario realizar
        ldx     #v
        abx
        ldaa    0,x
;y para el i-esimo elemento del vector u en el acumulador
        ldx     #u
        aslb
        abx
        ldd     0,x
```

Una matriz es un vector cuyos elementos son vectores de la misma longitud, por lo que se necesitan dos índices para direccionar adecuadamente a un elemento de la misma.

```
      org     $100
;Definimos dos matrices donde se ordenan varios los datos dando preferencia
;a las filas (matriz a1)
a1:     fcb     1,2,3
        fcb     4,5,6
        fcb     7,8,9
        fcb     10,11,12
;o a las columnas (matriz a2)
a2:     fcb     1,4,7,10
        fcb     2,5,8,11
        fcb     3,6,9,12
;para extraer un dato de una de estas matrices se utiliza dos contadores
;(variables i, j), por ejemplo para las matrices ordenadas según el
;criterio de a1 el algoritmo de extraccion de un dato es
;       dirección=(i x n)+ j + dirección de a1(0,0)
;donde n es el número de filas que tiene la matriz. En el siguiente ejemplo
;suponemos que i esta almacenado en el acumulador A y j en el B, entonces,
;la rutina para obtener el valor a(i,j) en el acumulador A es
        pshb            ;guarda el valor de B (j) en el Stack
        ldab    #3      ;pone "n" en el acumulador B
        mul             ;multiplica i por n en A
        tsx             ;toma el último valor introducido en el Stack (j)
        addb    0,x
        adca    #0      ;propaga el acarreo
        addd    #a1     ;suma la dirección de a1 en X
        xgdx            ;apunta x a a(i,j)
        ldaa     0,x    ;coloca a(i,j) en A
        ins             ;incrementa el valor del SP
.end
```

Una tabla es a una matriz lo que una lista es a un vector. En la misma se pueden almacenar datos o caracteres.

Para acceder a los datos almacenados en estas estructuras es aconsejable utilizar el modo de direccionamiento indexado (instrucción INX)

Estructuras de datos secuenciales

En este caso a los datos se acceden por su posición relativa. En vez de utilizar un índice *i* sólo se puede acceder al dato anterior o posterior: Cadenas (strings), pilas (stacks)

Una cadena es una secuencia de elementos tales que desde el i-ésimo elemento sólo se puede acceder al (i+1)-ésimo o al (i-1)-ésimo. Para manejar estas estructuras se recomienda las instrucciones INX o/y DEC.

```
;Ejercicio:
;completa las lineas de comentario de esta rutina, que compara una cadena
;almacenada en memoria con la cadena 'start'
;analiza que función tiene cada una de las instrucciones

.org    $100
alfa:   rmb     2       ;reserva espacio en memoria RAM
beta:   fcb     'start' ;caracteres a comparar
cadena: equ     $40
.org    $120            ;empieza el programa
srch:   ldx     alfa    ;
        ldy     #beta
        ldaa    #5      ;el bucle tendra 5 pasos
bucle:  ldab    0,x
        cmpb    0,y
        bne     incorrt
        inx
        iny
        deca
        bne     bucle
        jmp     cadena
incorrt rmb     0
.end

;suponemos que cadena es una subrutina a la ira el programa si la
;secuencia es correcta (coincide con start)
```

Una pila es una estructura de que almacena información de forma LIFO (Last input, Firts Output) esto es, el primero en entrar es el último en salir

Un “queue” es una estructura FIFO (first input, first output). Su forma de operación equivale a un registro de desplazamiento, y en ocasiones se denominan registros de desplazamiento elásticos. Se pueden utilizar para almacenamiento temporal de datos, siempre que estos se utilicen en el mismo orden en el que se han almacenado.

Escribiendo programas claros

- Documentación

Consiste en un conjunto de comentarios y caracteres de flujo que describen a un programa. Se deben incluir al final de las líneas del programa en ensamblador para explicar, de forma resumida la acción de cada operando. También es aconsejable introducir líneas específicas de documentación (utilizando el signo “;” el compilador no tiene en cuenta estas líneas para generar el código de programa)

Para completar la documentación se debe incluir un diagrama de flujo de programa, que desde el punto de vista de un sistema secuencial, no es más que un grafo de estado que lo desribe.

- Programación en alto y bajo nivel

Además de los compiladores propios existe la posibilidad de generar código con compiladores de alto nivel (ej lenguaje C). De esta forma se reduce el tiempo de desarrollo de

un producto aunque, de forma general, se aumentan los recursos necesarios para implementarlo (más memoria, etc) ya que el compilación suele ser menos eficiente.

Subrutinas y argumentos

Recordemos que es adecuado segmentar grandes desarrollos en un programa principal y en varias subrutinas que ejecutan pequeñas partes del mismo. Es necesario efectuar una adecuada transferencia de valores y argumentos entre el programa principal y las subrutinas. Para poder realizarlo se utilizan dos técnicas diferentes

- Crear una lista de datos temporales: En este caso es aconsejable utilizar un direccionamiento indexado a estos datos (utilizando un de los dos registros de la CPU) tanto para crear o leer la lista
- Utilizando una pila (Stack) y el registro de pila (SR) junto con las instrucciones relacionadas con el mismo.

El 68HC11 tiene instrucciones específicas de saltos y retornos hacia y desde las subrutinas. Entonces, cuando se escriba una llamada a una subrutina se deben realizar los siguientes pasos

- Poner cualquier argumento que se desee pasar al Stack y los espacios necesarios para retornar información al programa principal.
- Ejecutar el salto a la subrutina
- Escribir la lista de argumentos (se aconseja utilizar la directiva FDB para direcciones y operandos de 2 bytes y la directiva FCB para operandos de 1 byte)

En la subrutina se deben realizar las siguientes operaciones

- Se deben crear variables locales y asociarlas a las variables transferidas desde el programa principal
- Ejecutar la rutina
- Salir de la misma con una instrucción adecuada (RTS, RTI) para recuperar información almacenada en el registro de Stack

```
;Subrutina de conversion de una cadena de caracteres en valores numericos del 0 al 19 y
;este numero es utilizado como una direccion.
;Es necesaria una cadena de caracteres. La forma mas eficaz es pasar la cadena por
;referencia (la dirección del primer carácter) al registro de índice y la longitud
;de la cadena al acumulador
;la llamada a la subrutina se realizara utilizando la cadena 'alpha'
bsr    hash
fcb    10
fdb    'alpha'
;esta rutina afecta a los registros CCR, A, X, Y
;retorna la direccion en B

hash   puly            ;toma la dirección de retorno
        ldaa    0,y     ;pone el contador de la cadena en A
        ldx     1,y     ;pone la dirección de la cadena en X
        iny             ;mueve la dirección de retorno
        iny
        iny
        pshy            ;pone la dirección de retorno en el Stack
        psha            ;guarda el contador en el Stack
        pshx            ;guarda el puntero de la cadena en el Stack
        clrb            ;borra el lugar donde se pondrá el resultado
        tsy             ;Y llega a la parte superior del puntero de Stack
bucle:  pshb            ;guarda el resultado y el CCR en el Stack
        tpa
        psha            ;guarda el CCR
        ldx     0,y     ;toma el puntero de la cadena
        ldab    0,x     ;toma un caracter
        inx             ;mueve el puntero
        stx     0,y     ;lo vuelve a colocar en el Stack
        subb    #$25    ;crea un caracter
        bpl     noclr   ;si es negativo
        clrb            ;poner el acumulador a cero
```

```
noclr:  pula            ;reestablece el CCR
        tap
        tsx
        adcb    2,x     ;añadir al caracter modificado
        ins
        rolb
        dec     2,y     ;decrementar el contador
        bne     bucle   ;hasta llegar a cero repetir el bucle
        rorb            ;hasta el último rolb
        ldaa    #$11    ;multiplicar por 11
        mul
        andb    #$1f    ;borra los tres bits mas significativos
        cmpb    #19     ;¿nos encontramos en el rango deseado?
        bls     exit    ;si es así, el resultado está en B
        subb    #20     ;
        cmpb    #9      ;si ahora es menor que 9
        bhi     exit    ;entonces debe ser el doble
        aslb            ;y en este caso
        bitb    #2      ;extendemos el bit menos significativo
        beq     exit    ;
        orab    #1
exit:   ins             ;borra el contador del Stack
        ins             ;borra el puntero del Stack
        ins
        rts             ;actualizamos la dirección de retorno que se
                        ;encuentra en el Stack
```

Una buena documentación de cada subrutina debe incluir la siguiente documentación

- Una breve descripción del significado de la misma
- Un listado de todos los argumentos, lo que significan y si han sido llamados por valor, por referencia o nombre
- Un ejemplo de una llamada clara a la subrutina, indicando que argumentos han pasado al Stack o si se han utilizado instrucciones del tipo BSR o SWI
- Un listado de los registros utilizados o modificados
- Un listado de cualquier error y como se solucionan posibles salidas no deseadas de la rutina, que pueden reportar errores

Es interesante que cualquier rutina sea independiente de la posición que ocupa, entonces los comentarios deben evitar información específica al respecto.

Operaciones aritméticas

El bus de datos 8 bits de este tipo de microcontroladores se queda, a todas luces, pequeño cuando se desean realizar operaciones aritméticas con un mínimo de precisión. Como dato de interés cualquier cálculo científico necesita con facilidad palabras de 40 bits y para entornos de prgramación con bus de datos de 32 bits es habitual utilizar doble precisión (64 bits) en este tipo de cálculos. Por otra parte datos de 16 bits son adecuados para calcular posiciones de memoria y para cálculos en procesos de control pero de nuevo se quedan cortos en cálculos aritméticos. Es interesante ver cómo manipular las instrucciones aritméticas de 16 bits del hc11 para conseguir cálculos aritméticos eficaces.

Otro problema que viene relacionado es la densidad de almacenamiento de datos. A mayor precisión es obvio que se gastan más recursos de memoria, tanto en datos como en longitud de programa (no tiene mucho sentido tener una rutina para sumar números de 16 bits, otra para números de 24 bits, etc…). Es interesante realizar un esfuerzo de diseño para conseguir rutinas modulares en cuanto al tamaño de los operandos se refiere.

La forma en la que conocemos cómo se manipula una operación con dos números (de tamaño arbitrario) es colocando uno de ellos en el Stack.

Consideremos la siguiente operación

(A+B)/(C-D)

La siguiente rutina permite implementar este algoritmo; para seguirla aconsejamos escribir en un papel aparte cada uno de los pasos que realiza.

```
      bsr      push
        fdb    A
        bsr    push
        fdb    B
        bsr    add
        bsr    push
        fcb    C
        bsr    push
        fcb    D
        bsr    sub
      bsr      div
```

El primer problema vemos que consiste en como construir el programa para llamar a cada una de las subrutinas de cálculo para implementar de forma correcta la fórmula algebraica que se desea calcular (una posible técnica se denomina “formual tree”).

El segundo problema consiste en escribir cada una de las subrutinas que realizan las operaciones básicas en las que se descompone la fórmula. Existen muchas rutinas para cada caso en particular, dejamos al estudiante el análisis de las que, en cada momento, necesiten para su desarrollo, no obstante mostramos una rutina típica de traslación de información para que sea analizada

```
;Esta rutina pone un numero de longitud N desde la posicion L en el Stack de operandos
;La secuencia de llamada es
ldaa    #N
ldx     #L
jsr     push
;
;utiliza A, B, X
push    psha            ; guarda la información de A
bucle:  ldab    0,X     ;obtiene una palabra desde el numero que se va a trasladar
        inx
        stab    0,Y     ;al Stack
        iny             ;mueve el puntero
        deca            ;cuenta el numero de palabras a mover
        bne     bucle
        pula            ;recupera la longitud
        staa    0,Y     ;pone la longitud al final
        iny             ;mueve el puntero
        rts
```

Bibliografía.

Manuales técnicos de Motorola: Se pueden obtener en el servidor de Motorola

"68HC11 Reference Manual"

- http://www.mcu.motsps.com/freeweb/amcu_ndx.html#mcu11

"HC11 E Serie Reference Manual".

- http://ebus.mot-sps.com/ProdCat/psp/0,1250,68HC11E9~98635,00.html

"Using the M68HC11 microcontroller. A guide to interfacing and programing the M68HC11 microcontroler". Skroder J. C. Ed. Prentice Hall

"Microcontroller Technology. The 68HC11. 2º Edition". Spasov. P., Ed Prentice Hall