FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

A LOS ASISTENTES A LOS CURSOS

Las autoridades de la Facultad de Ingeniería, por conducto del jefe de la División de Educación Continua, otorgan una constancia de asistencia a quienes cumplan con los requisitos establecidos para cada curso.

El control de asistencia se llevará a cabo a través de la persona que le entregó las notas. Las inasistencias serán computadas por las autoridades de la División, con el fin de entregarle constancia solamente a los alumnos que tengan un mínimo de 80% de asistencias.

Pedimos a los asistentes recoger su constancia el día de la clausura. Estas se retendrán por el periodo de un año, pasado este tiempo la DECFI no se hará responsable de este documento.

Se recomienda a los asistentes participar activamente con sus ideas y experiencias, pues los cursos que ofrece la División están planeados para que los profesores expongan una tesis, pero sobre todo, para que coordinen las opiniones de todos los interesados, constituyendo verdaderos seminarios.

Es muy importante que todos los asistentes llenen y entreguen su hoja de inscripción al inicio del curso, información que servirá para integrar un directorio de asistentes, que se entregará oportunamente.

Con el objeto de mejorar los servicios que la División de Educación Continua ofrece, al final del curso deberán entregar la evaluación a través de un cuestionario diseñado para emitir juicios anónimos.

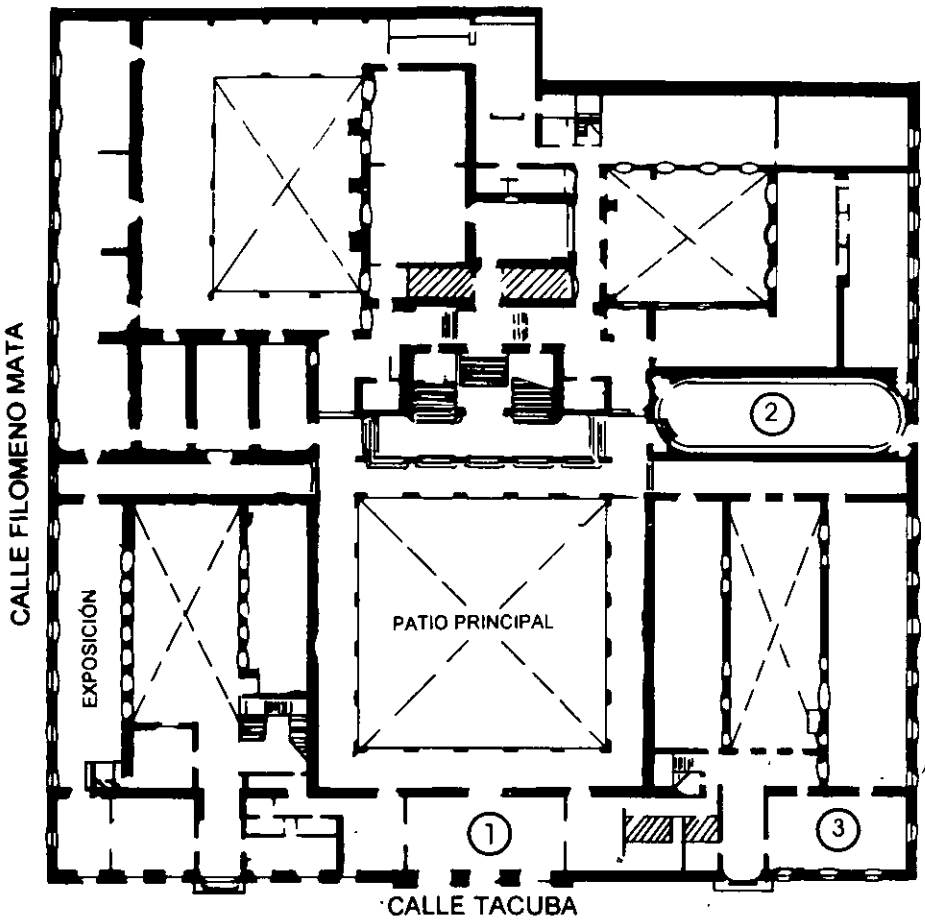
Se recomienda llenar dicha evaluación conforme los profesores impartan sus clases, a efecto de no llenar en la última sesión las evaluaciones y con esto sean más fehacientes sus apreciaciones.

Atentamente

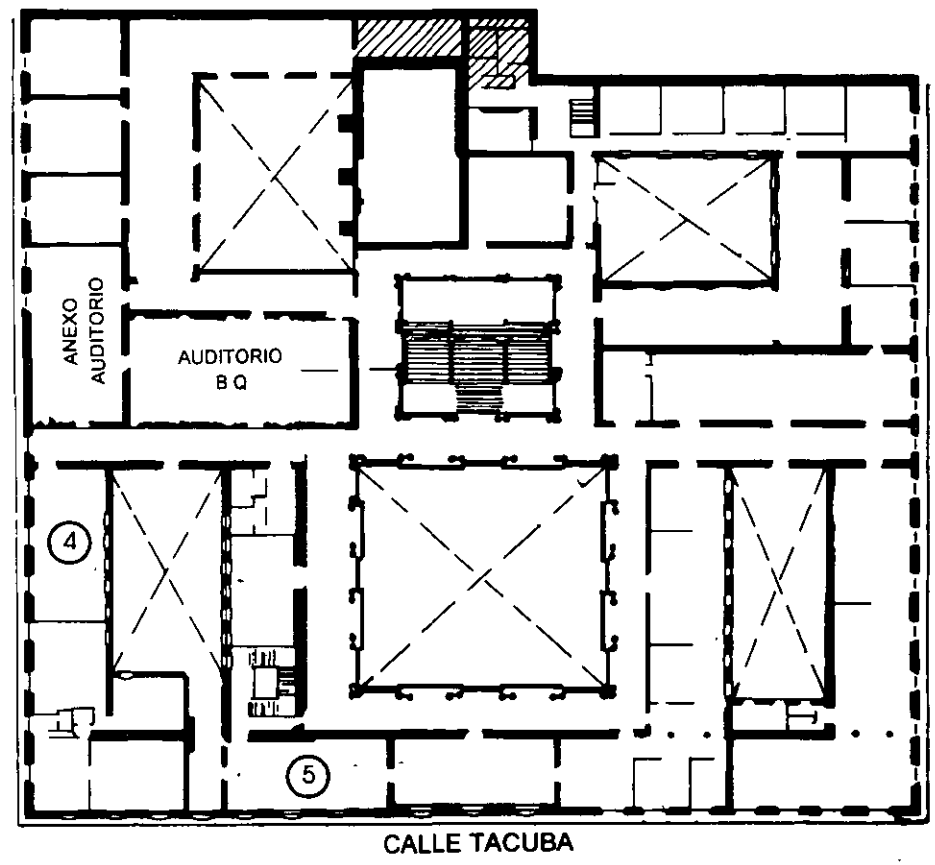
División de Educación Continua.

Palacio de Minería Calle de Tacuba 5 Primer piso Deleg. Cuauhtémoc 06000 México, D F APDO. Postal M-2285
Teléfonos: 5512-8955 5512-5121 5521-7335 5521-1987 Fax 5510-0573 5521-4021 AL 25

PALACIO DE MINERIA

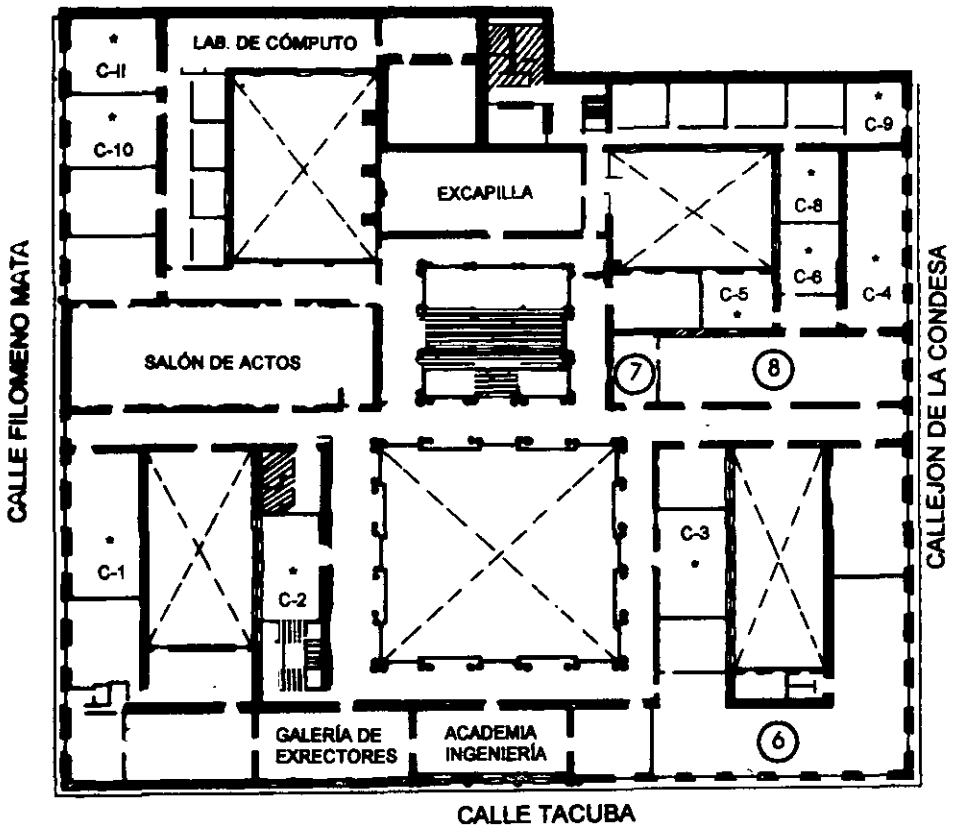


PLANTA BAJA



MEZZANINNE

PALACIO DE MINERIA



1er. PISO

GUÍA DE LOCALIZACIÓN

1. ACCESO
2. BIBLIOTECA HISTÓRICA
3. LIBRERÍA UNAM
4. CENTRO DE INFORMACIÓN Y DOCUMENTACIÓN "ING. BRUNO MASCANZONI"
5. PROGRAMA DE APOYO A LA TITULACIÓN
6. OFICINAS GENERALES
7. ENTREGA DE MATERIAL Y CONTROL DE ASISTENCIA
8. SALA DE DESCANSO

SANITARIOS

* AULAS

DIVISIÓN DE EDUCACIÓN CONTINUA
FACULTAD DE INGENIERÍA U.N.A.M.
CURSOS ABIERTOS

DIVISIÓN DE EDUCACIÓN CONTINUA
DEC

FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

"Tres décadas de orgullosa excelencia" 1971 - 2001

CURSOS ABIERTOS

DIPLOMADO EN MICROCONTROLADORES

(SISTEMAS EMBEBIDOS)

MODULO II: MICROCONTROLADORES DE 16 BITS Y SUS APLICACIONES A PROCESAMIENTO DIGITAL DE SEÑALES Y CONTROL DIFUSO

TEMA

EL MICROCONTROLADOR MC68HC12

EXPOSITOR: ING. BEATRIZ ESLAVA ARELLANES
PALACIO DE MINERIA
MARZO DEL 2001

EL MICROCONTROLADOR MC68HC12

ING. BEATRIZ ESLAVA ARELLANES

Introducción a la arquitectura del microcontrolador MC68HC12

El primer microcontrolador con el CP12 fue el MC68HC812A4 que tiene 112 pines, desarrollado para sistemas en modo expandido. El segundo dispositivo tiene 80 pines el MC68HC912B32, desarrollado para aplicaciones en modo mono-chip o single-chip. Ambos dispositivos se basan en arreglos modulares, este procesador es completamente compatible con los programas del MC68HC11 pero es mayor, mas grande. Está diseñado para que su bus opere a una velocidad de 8MHz.
La diferencia mas significativa entre los dos dispositivos es la expansión de la estructura de bus y la memoria en el Chip. El MC68HC812A4 tiene 1K byte de RAM estática y 4K bytes de EEPROM, pero no tiene memoria de programa. Los sistemas basados en este microcontrolador usan un bus no multiplexado para accesar memoria de programa externa. 22 líneas de dirección mas 6 selectores programables que permiten accesar más de 4 megabytes de memoria de programa y mas de 1 megabyte de memoria de datos. Asimismo el MC68HC812 tiene completos los 16 bits de datos para poder operar externamente con 16 bits o con 8 bits y el modo de 8 bits puede interconectarse con EPROMs y RAMs de bajo costo.
El MC68HC912B32 tiene una memoria EEPROM flash de 32k de memoria de programa, 1K byte de memoria RAM , y 768 bytes de EEPROM borrable. Este es el primer microcontrolador que incluye los dos tipos de memoria EEPROM en el mismo chip. Se necesita una fuente externa de 12 volts para poder borrar y programar la memoria flash, la EEPROM borrable usa la fuente de voltaje normal (2.7 a 5.5 Volts) para las operaciones de borrado y programado. El MC68HC912B32 puede ser usado en modo expandido pero su bus de datos/direcciones multiplexado fue desarrollado para prueba de fabrica.

En la siguiente tabla se observan las diferencias entre los dos microcontroladores

MC68HC812A4	MC68HC912B32
CPU de 16 bits	CPU de 16 bits
Voltaje de Operación 2.7 – 5.5 Volts	Voltaje de Operación 2.7 – 5.5 Volts
PLL programable o Xtal=2x velocidad de bus	Xtal=2x velocidad de bus
112 pines (95 pines de propósito general E/S) Key Wakeup on tres puertos de 8 bits	80 pines (64 pines de propósito general E/S)
Bus expandido no multiplexado o Modo mono-chip de 16 bits o 8 bits	Modo mono-chip y Modo Expandido con bus multiplexado de 16/16 y 16/8
Expansión de memoria a mas de 5 megabytes	
6 selectores de chip programables	
4K EEPROM 1K SRAM 256 Bytes de espacio de registros	32K EEPROM Flash 718 EEPROM 1K SRAM 256 Bytes de espacio de registros

8 canales de 8 bits de convertidor A/D	8 canales de 8 bits de convertidor A/D
8 canales de timer	8 canales de timer
Acumulador de pulsos de 16 bits	Acumulador de pulsos de 16 bits
	4 canales PWM
2 canales de SCI Asíncrono	1 canales de SCI Asíncrono
1Canal de SPI Síncrono	1Canal de SPI Síncrono
	1 canal digital serial J1850
Single-Wire BDM	Single-Wire BDM con hardware breakpoints
COP Watchdog timer y Clock Monitor	COP Watchdog timer y Clock Monitor
Timer de Interruptor periódico	Timer de Interruptor periódico

Arquitectura del microcontrolador MC68HC12

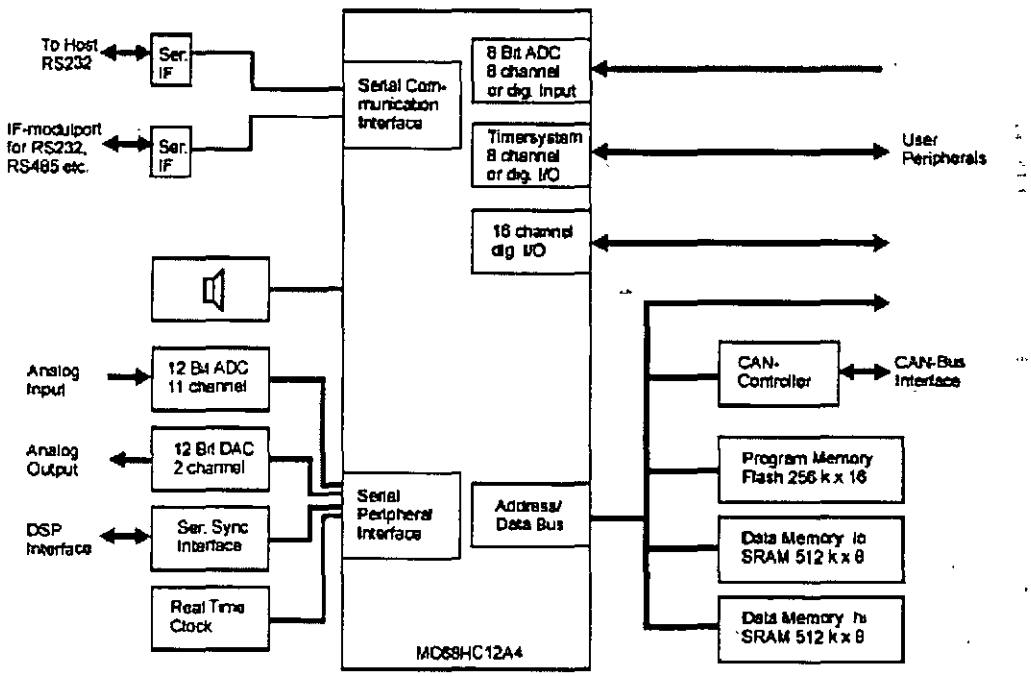


Fig. 1 Arquitectura del CPU12

Usando el mismo cristal de 16 MHz el HC12 obtiene un bus de 8 MHz comparado con los 4 MHz del HC11. Los dos , el A4 y el B32 pueden usar los modos de WAIT Y STOP para reducir la potencia y el A4 tiene adicionalmente un camino para "save power". Un PLL puede ser usado como un cristal de baja frecuencia (por ejemplo 32 KHz) y múltiplos para una frecuencia de operación deseada. Cuando el procesamiento demandado sea menor, el PLL puede ser usado para reducir la velocidad del reloj y esto reduce el consumo de potencia. Tiene adicionalmente divisores de reloj del sistema y módulos periféricos para

programas la velocidad hacia arriba o hacia debajo de el CPU y de la velocidad de bus. Mantiene una velocidad de reloj para los timers y periféricos seriales.
Los dos dispositivos permiten que la mayoría de los pines puedan ser usados para propósito general cuando no son necesitados para otras funciones. Los pines de control de bus tales como E y R/W pueden ser usados como pines de propósito general si no se tienen dispositivos de memoria externa. Esto hace que se tenga como máximo 95 pines de E/S de propósito general en el M68HC812A4 y en el M68HC912B32 se tenga 64. El sistema key wake up en el A4 permite seleccionar flancos en cualquier combinación de pines de los tres puertos de 8 bits. Esto permite que el programa controle el compromiso entre desempeño y ahorro de energía.

Los dos ofrecen un timer de 8 canales similar el timer de propósito general de 68HC11 excepto que todos lo 8 canales pueden ser configurados como entrada o salida de captura. Y tiene nuevas posibilidades de reloj. El reloj de acumulador de pulsos pude ser ruteado a el timer de 16 bits, y el acumulador de puso pude tener su reloj de un pin externo y el mismo pin externo se pude usar para el timer principal. Colocando en cascada el contador del acumulador de pulsos y el contador de timer principal se puede contar periodos muy largos. Otra configuración de reloj permite que una de las funciones de salida de comparación sea un reset para el contador principal de 16 bits, de tal manera que se pude tener un módulo de conteo. Tienen un acumulador de pulsos de 16 bits para contar eventos de un pin externo.

El B32 tiene 4 canales PWM con registros de 8 bits de control separados para cada ancho de pulso y ciclo de trabajo. El bloque de selección de reloj para cada canal permiten operar cada canal a diferente frecuencia. La velocidad se deriva de los selectores apagados de el divisor de 8 bits, seguido por el modulo de divisores de 8 bits que dan la posibilidad de tener un rango de frecuencia de entrada de 8 MHz a 125 HZ. La salida del PWM puede ser alineada a la izquierda o al centro. La alineación centrada resulta en menos flancos simultáneos que generan menos ruido.

Los 8 canales de convertidor A/D son similares a los del 68HC11 . Ambos dispositivos tienen 8 registros de 8 bits para el convertidor A/D, es decir se pueden conectar independientemente 8 señales analógicas al HC12 y no es necesario hacer 2 lecturas (primero 4 canales y después otros 4). Tiene 2 pines de polarización para el convertidor V_{RH} y V_{RL}, estos voltajes de referencia determinan el rango del convertidor A/D. Esto permite usar un voltaje de precisión en el dispositivo para asegurar las lecturas exactas. Comúnmente se conecta V_{RH} a +5v, y V_{RL} a GND.

El sistema de A/D es capaz de generar dos tipos de resultados. El primero es una serie de ocho lecturas secuenciales , se guardan los resultados de estas 8 muestras en el resultado los registros ATD0 - ATD7. El segundo modo se llama modo multicanal, es decir, se selecciona un canal y se guarda en el registro correspondiente el resultado. El resultado del canal 0 a ATD0, canal 1 a ATD1, y así sucesivamente a hasta el canal 7, en ATD7.

El COP (timer watchdog), sistema de reloj, y las interrupciones periódicas se encuentran en todos los MC68HC12. El disparo del timer del watchdog (cuando esta habilitado) es un reset si el software de aplicación falla al completar el servicio de secuencia dentro del periodo de tiempo El reloj monitor (cuando esta habilitado) dispara el sistema de reset si la

frecuencia de reloj falla. La velocidad de interrupción periódica (llamada algunas veces interrupción en tiempo real), puede generar interrupciones a una velocidad seleccionada por el usuario. La velocidad de interrupción periódica se obtiene dividiendo hacia abajo la velocidad del módulo de reloj (M). En el B32 el reloj M es el mismo que el reloj E, pero en el A4, el módulo de frecuencia puede ser E, E/2, E/4 o E/8. La velocidad de interrupción periódica se selecciona dividiendo M entre 2^{13} hasta M dividido por 2^{19} o aproximadamente 1 ms a 65 ms asumiendo una velocidad de bus de 8MHz sin divisiones de E.

El modelo de programación y el orden de las interrupciones es igual al del MC68HC11. Además todos los programas que usan el código para el HC11 son aceptados por los ensambladores del CPU12 sin ninguna modificación. La mayoría de las instrucciones del HC11 conservan el mismo código objeto en el CPU12.

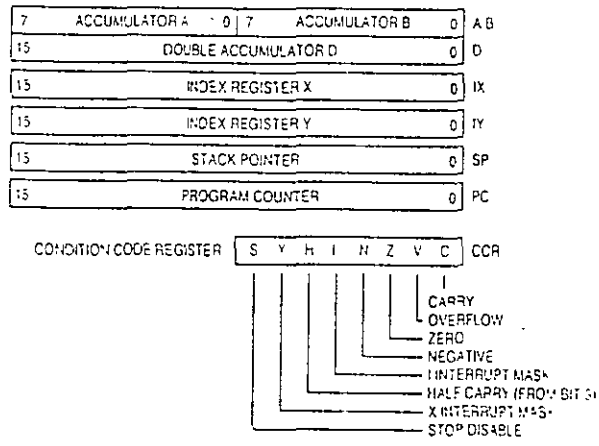


Fig. 2 Modelo de programación

La mejora mas obvia es que el CPU12 es un procesador de 16 bits con una ALU con capacidad de 20 bits para algunas operaciones . Todos los bus de datos en el MC68HC12 son de 16 bits y el bus de interface externa es normalmente de 16 bits pero tiene la opción de 8 bits. Esta opción permite tener sistemas mas baratos con memorias externas de 8 bits. Cuando se reduce el bus, el CPU12 ve al bus como de 16 bits. Un bus de interface inteligente detiene temporalmente los relojes del CPU cuando necesita dividir acceso de 16 bits en 2 piezas de 8 bits.

Los 68HC11 tienen un solo puerto de serie, y un solo puerto de SCI. La velocidad máxima disponible es de 9600 baudio, a menos que se seleccione una velocidad rara como 31.250K, o 41.666K.

Los 68HC12 tienen dos puertos de serie y un solo puerto de SCI. El los UART en la tarjeta trabajan independientemente, y puede manejarse a las velocidades normales de 38400 es decir, su puerto de serie es mas rápido.

Puerto de SPI es comparable a los 68HC11, pero es mucho más rápido (4 Mbit / S).

Uno de las calidades de los 68HC11 en el modo single-chip es el número de pines de I/O con que se cuenta. Tiene el PORTA, PORTB, PORTC, PORTD, y PORTE de decir 40 pines. de I/O
El 68HC12 tiene los puertos anteriores mas PORTF, PORTG, PORTH, PORTJ, PORTT, y finalmente PORTAD. Las asignaciones de puertos son las siguientes:

Port A	In/Out	General-purpose I/O in single-chip modes. External address bus ADDR[15:8] in expanded modes.
Port B	In/Out	General-purpose I/O in single-chip modes. External address bus ADDR[7:0] in expanded modes.
Port C	In/Out	General-purpose I/O in single-chip modes. External data bus DA-TA[15:8] in expanded wide modes; external data bus DATA[15:8]/ DATA[7:0] in expanded narrow modes.
Port D	In/Out	General-purpose I/O in single-chip modes and expanded narrow modes. External data bus DATA[7:0] in expanded wide mode. As key wakeup can cause an interrupt when an input transitions from high to low
Port E	PE[1:0] In PE[7:2] In/Out	Mode selection, bus control signals and interrupt service request signals; or general-purpose I/O.
Port F	In/Out	Chip select and general-purpose I/O.
Port G	In/Out	Memory expansion and general-purpose I/O.
Port H	In/Out	Key wakeup and general-purpose I/O, can cause an interrupt when an input transitions from high to low.
Port J	In/Out	Key wakeup and general-purpose I/O, can cause an interrupt when an input transitions from high to low or from low to high.
Port S	In/Out	Serial communications interface and serial peripheral interface subsystems and general-purpose I/O.
Port T	In/Out	Timer system and general-purpose I/O.
Port AD	In	Analog-to-digital converter and general-purpose input.

Tipos de datos

El CPU12 usa los siguientes tipos de datos:
Bits
Enteros signados de 5 bits
Enteros signados y no signados de 8 bits
Números decimales codificados en binario de 2 dígitos (8 bits)
Enteros signados de 9 bits
Enteros signados y no signados de 16 bits
Direccionamiento efectivo de 16 bits
Enteros signados y no signados de 32 bits

Los enteros negativos están en forma de complemento a dos.
Los enteros signados de 5 y 9 bits son usados solamente como offset en los modos de direccionamiento indexado.
La dirección efectiva de 16 bits se forma durante el cálculo del modo de direccionamiento.

Las divisiones enteras de 32 bits usan las instrucciones de división extendida. La multiplicación extendida y multiplicación extendida con acumulación entrega 32 bits.

Modos de direccionamiento.

Los modos de direccionamiento determinan como el CPU accesará a la localidades de memoria que se operarán .
Todos los modos de direccionamiento del CPU12 se muestran en la tabla siguiente:

Modo de Direccionamiento	Formato	Abreviatura	Descripción
Inherente	INST (no necesita operadores adicionales externos)	INH	Operandos en los registros del CPU
Inmediato	INST *#opr8i* O INST *#opr16i*	IMM	Los operandos estan incluidos en la instrucción con un tamaño de 8 o 16 bits
Directo	INST *opr8a*	DIR	El operando es la parte baja (8bits) de la direccion en el rango de $0000 a $00FF
Extendido	INT *opr16a*	EXT	El operando es una dirección de 16 bits
Relativo	INST *rel8* O INST *rel16*	REL	Un offset relativo de 8 o 16 bits se suma a valor de pc actual
Indexado (offset de 5 bits)	INT *oprx5,xysp*	IDX	Offset constante de 5 bits signado para x,y,sp o pc
Indexado (con pre-decremento)	INT *oprx3,-xys*	IDX	Auto pre-decremento en x, y o sp en 1~8
Indexado (con pre-incremento)	INT *oprx3,+xys*	IDX	Auto pre-incremento en x, y o sp en 1~8
Indexado (con post-decremento)	INT *oprx3,xys-*	IDX	Auto post-decremento en x, y o sp de 1~8
Indexado (con post-incremento)	INT *oprx3,xys+*	IDX	Auto post-incremento en x, y o sp de 1~8
Indexado (acumlador offset)	INT *adb,xysp*	IDX	Indexado con un offset de 8 bits (acc – A o B) o 16 bits (D) d x, y , sp o pc

Indexado (offset de 9 bits)	INT *oprx9,xysp*	IDX1	Offset de 9 bits contaante a x, y, sp o pc
Indexado (offset de 16 bits)	INT *oprx16,xysp*	IDX2	Offset de 16 bits contante para x, y , sp o pc (16 bits de offset con extension de 2 bytes)
Indexado Indirecto (offset de 16 bits)	INT *[oprx16,xysp]*	[IDX2]	El apuntador del operando se encuenta en... Offset de 16 bits contantes para x, y , sp o pc
Indexado Indirecto (offset del acumulador doble D)	INT *[D,xysp]*	[D,IDX]	El apuntador del operando se encuentra en ... x, y sp o pc mas el valor en D

Modos de direccionamiento

Dirección efectiva

Cada modo de direccionamiento excepto el inherente genera una dirección efectiva de 16 bits, que es usada durante la referencia a memoria por parte de la instrucción. El cálculo de la dirección efectiva no requiere ciclos extras de ejecución.

Modo de direccionamiento Inherente

La instrucciones que usan este modo de direccionamiento no usan operandos o todos los operandos están en los registros internos del CPU. En ambos casos el CPU no necesita accesar localidades de memoria para completar la instrucción.
Ejemplos

```
NOP     ;Esta instrucción no requiere operandos
INX     ;El operando es un registro del CPU
```

Modo de direccionamiento Inmediato

Los operandos de las instrucciones están incluidos en la instrucción, y se coloca en la cola de instrucción una palabra de 16 bits durante un tiempo durante ciclo normal de ejecución. Dado que el programa es leído en la cola de instrucción muchos ciclos antes de que se

necesite, cuando el operando del modo de direccionamiento inmediato es llamado por la instrucción, este ya está presente en la cola de instrucción.
El símbolo # es usado para indicar el operando del modo de direccionamiento inmediato. Muchos programadores cometen el error de omitir el símbolo #. Esto causa que el ensamblador interprete erróneamente la expresión.
Por ejemplo LDAA #$55 significa que carga el valor $55 en el acumulador A. Mientras que LDAA $55 significa que carga en el acumulador A el valor que tenga la dirección $0055. Sin el símbolo # se interpreta como una instrucción usando el modo de direccionamiento directo.

Ejemplos

```
LDAA    #$55
LDX     #$1234
LDY     #$67
```

Estos son ejemplos de direccionamiento inmediato de 6 y 16 bits. El tamaño de el operando inmediato esta implícito en el contexto de la instrucción. En el tercer ejemplo la instrucción implica un valor de 16 bits, pero solo hay 8 bits. En este caso el ensamblador genera un valor de 16 bits, debido a que el CPU espera un valor de 16 bits.

```
BRSET    FOO,#$03,THERE
```

En este ejemplo, se usa el modo de direccionamiento extendido para accesar el operando F00, el modo de direccionamiento inmediato es usado para accesar el valor de la mascara $03, y el modo de direccionamiento relativo es usado para identificar la dirección de destino del brindo en caso de que se presente la condición de brinco. BRSET, esta listado como una instrucción que usa el modo de direccionamiento extendido sin embargo usa los modos inmediato y relativo también.

Modo de direccionamiento Directo

Este modo a veces es llamado direccionamiento de página-cero debido a que el operando está en el rango de memoria de $0000 a $00FF. Esta direccionamiento siempre empieza con $00, solo la parte baja de la dirección se incluye en la instrucción, reduce el espacio de programa y el tiempo de ejecución. El sistema pude optimizarse colocando los datos de acceso frecuente en esta área de memoria. La parte alta de la dirección, se asume como $00
Ejemplo

```
LDAA    $55
LDX     $20
```

Modo de direccionamiento Extendido

En este modo de direccionamiento, la dirección completa de 16 bits que va a ser operada debe ser parte de la instrucción. Este modo de direccionamiento pude ser usado para accesar cualquier localidad en el mapa de memoria de 64K

Ejemplo

LDAA $F03B

Modo de direccionamiento Relativo

El modo de direccionamiento relativo es usado solo por instrucciones de brinco. Brincos condicionados largos y cortos usan este modo de direccionamiento exclusivamente, pero versiones de brinco con manipulación de bit (BRSET y BRCLR) usan múltiples modos de direccionamiento.
Instrucciones de brinco corto están formados por 8 bits de código de operación y 8 bits de offset contenido en el byte que sigue al código de operación. Las instrucciones de brinco largo están formadas por 8 bits de pre-byte, 8 bits de código de operación y un offset de 16 bits signado que son los 2 bytes que siguen al código de operación.
Cada instrucción de brinco condicional verifica el estado de bits en el registro de código de condición (CCR). Si los bits están en un estado específico, el offset se suma a la dirección de la localidad de memoria siguiente, después el offset forma la dirección efectiva, y continua la ejecución en esa dirección, si los bits no están en el estado específico, ejecuta la instrucción que sigue a la instrucción de brinco.
Varios modos de direccionamiento pueden ser usados para accesar la localidad de memoria, un operando de 8 bits de mascara se usa para verificar los bits. Si cada bit en memoria que corresponda a un 1 en la mascara es 1 (BRSET) o 0 (BCLR), un offset de 8 bits se suma a la dirección siguiente de memoria para formar la dirección efectiva, y la ejecución del programa continua en esta dirección, Si todos los bits en la memoria que corresponden a uno en la mascara no están en el estado específico, la ejecución continua con la instrucción que sigue a la instrucción de brinco.
Los offset de 8 y 16 bits son números en complemento a 2 signados para soportar brincos hacia arriba y hacia abajo en la memoria. El rango numérico de los brincos cortos es $80 (-128) a $7f (128). . El rango numérico de los brincos largos es $8000 (-32768) a $7FFF (32767). Si el offset es cero, el CPU ejecuta la instrucción siguiente a la instrucción de brinco.
Debido a que el offset es la parte final de la instrucción de brinco, usar un offset negativo puede causar que el PC apunte a un código de operación e inicie un loop. Para esto, la instrucción de brinco (BRA) siempre debe estar compuesta de 2 bytes, si se usa un offset de $FE se pone en un loop infinito; en la misma lógica un brinco largo (LBRA) siempre deberá tener un offset de $FFFC
Un offset que apunte a un código de operación puede causar que una condición de brinco condicional de bit, repita la ejecución hasta que una condición específica de bit sea satisfecha. Esta condición de brinco pude consistir de 4, 5 o 6 bits dependiendo el modo de direccionamiento usado para accesar el byte en memoria. El valor de offset para un loop puede variar. Usar un offset de $FC con BRCLR que accese a memoria usando 8 bits de post-byte indexado- pone a ejecutar un loop hasta que todos los bits en la memoria especificada que correspondan a uno en la máscara estén en cero.

Modos de direccionamiento Indexado.

El CPU12 usa una versión refinada del modo de direccionamiento indexado del MC68HC11 que reduce el tiempo de ejecución y elimina fallas de tamaño de código por el uso del registro de índice Y. En la mayoría de los casos, el tamaño del código en el CPU12 para operaciones idexadas es el mismo o menor que el del HC11. El tiempo de ejecución es menor en todos los casos. Las mejoras en el tiempo de ejecución, reducen el números de ciclos para todas las instrucciones indexadas y la velocidad de reloj es mayor.

El esquema de direccionamiento indexada tiene la posibilidad de sumar 0, 1 o 2 extensiones de byte después de código de operación. El postbyte y las extensiones realizan las siguientes tareas:

Especifican que registro de índice se esta usando

Determinan si el valor en es acumulador usa offset

1. Habilita automáticamente el pre o post decremento o incremento
2. Especifica el tamaño del incremento o decremento
3. Especifica el uso de 5-,9- o 16 bits de offset signado.

Esta aproximación elimina la diferencia ente el uso de los registros X y Y e incrementa enormemente las capacidades del direccionamiento indexado.

Las mayores ventajas del modo de direccionamiento indexado son:

- El Stack Pointer (SP) pude ser usado como un registro de índice en todas las operaciones indexadas
- El Program Counter (PC) puede ser usado como un registro de índice y se autoincrementado y autodecrementado
- Los acumuladores A, B y D pueden ser usados como offset
- Pre o Post decremento o incremento automático de –8 a +8
- Selección de 5, 9 o 16 bits de offset signado
- Uso de dos modos de direccionamiento indexado indirecto
  Modo indexado indirecto con offset de 16 bits
  Modo indexado indirecto con offset en el acumulador D

Todos los modos de direccionamiento usan un registro de 16 bits del CPU e información adicional para crear la dirección efectiva. En la mayoría de los casos la dirección efectiva especifica la localidad de memoria que va a ser afectada por la operación. En algunas variaciones del direccionamiento indexado, la dirección efectiva especifica la localidad que apunta a la localidad de memoria que va ser afectada por la operación.
Las instrucciones del modo de direccionamiento indexado usan un postbyte para especificar X, Y, SP, o PC como el registro de índice base.
Un grupo especial de instrucciones (LEAS, LEAX y LEAY) ocasionan que el cálculo de la dirección efectiva sea guardada en un registro de índice para futuros cálculos

Código Postbyte (xb)	Sintaxis Operando	Comentarios
rr0nnnn	,r n,r -n,r	Offset constante de 5 bits n=-16 a +15 rr puede especificar X, Y, SP o PC
111rr0zs	n,r -n,r	Offset constante (9- o16 bits signados) z- 0=9bits con signo en el postbyte (s) LSB 1= 16 bits si z=s=1, offset indexado indirecto rr puede especificar X, Y, SP o PC
111rr011	[n,r]	Offset de 16 bits indexado indirecto rr puede especificar X, Y, SP o PC
rr1pnnnn	n,-r n,+r n,r- n,r+	Auto pre-decremento/incremento o Auto post-decremento/incremento p =pre-(0) o post-(1), n=-8 a –1,+1 a +8 rr puede especificar X, Y o SP (PC no es una elección válida)
111rr1aa	A,r B,r D,r	Offset Acumulador (8 o 16 bits no signados) aa –00 =A 01 =B 10 =D (16 bits) 11= Ver offset acumulador indexado indirecto rr puede especificar X, Y, SP o PC
111rr111	[D,r]	Offset Acumulador indexado indirecto rr puede especificar X, Y, SP o PC

Resumen de Direccionamiento Indexado

Direccionamiento indexado offset constante de 5 bits

Este modo de direccionamiento usa 5 bits signados de offset que esta incluido en el postbyte de la instrucción. Este offset corto es sumado al registro de índice base (X,Y, SP O PC) para formar la dirección efectiva de la localidad de memoria que será afectada por la instrucción. Esta da un rango de –16 a +15 del valor en el registro de índice base. Existen sin embargo modos de direccionamiento de 9 o 16 bits, estos modos requieren adicionalmente bytes de extensión en la instrucción para esta información extra. En la mayoría de las instrucciones indexadas en programas reales usan offset que se ajusten a la forma corta de 5 bits in direccionamiento indexado
Ejemplos

```
LDAA    0,X
STAB    -8,Y
```

Direccionamiento Indexado con offset de 9 bits constante

Este modo de direccionamiento usa 9 bits de offset signado que es sumado al registro de índice base (X, Y, SP, o PC) para formar la dirección efectiva de la localidad de memoria que va a ser afectada por la instrucción. Esto da un rango de –156 a +255 del valor base en el registro de índice. El bit mas significativo (bit de signo) del offset esta incluido en el postbyte de la instrucción y los restantes 8 bits son parte del byte de extensión después de postbyte de la instrucción, en el flujo de la instrucción.
Ejemplos

```
LDAA        $FF,X
LDAB        -20,Y
```

Direccionamiento Indexado con offset de 16 bits constante

Este modo de direccionamiento usa 16 bits de offset que se suma la valor del registro de índice base (X,Y, SP o PC) para formar la dirección efectiva que será afectada por la instrucción. Esto permite el acceso de cualquier dirección en los 64K del espacio de dirección. Ya que el bus de direcciones y el offset son de 16 bits, no importe que es valor del offset sea considerado como un valor signado o no signado ($FFFF puede ser +65,535 o como –1). Los 16 bits de offset son para dar los 2 bytes de extensión después de postbyte de la instrucción.
Direccionamiento Indexado Indirecto de 16 bits constante
Este modo de direccionamiento suma 16 bits a la instrucción, supkendo el offset del registro de índice base para formar la dirección de memoria que contendrá el apuntador de la localidad de memoria que va a ser afectada por la instrucción. Esta instrucción por si misma no apunta a la dirección de memoria sobre la que se va a actuar, mas bien la localidad de apuntador a la dirección que se va a actuar. Los corchetes cuadradas distinguen este modo de direccionamiento del indexado con offset de 16 bits constante.
Ejemplo

```
LDAA [10,X]
```

Direccionamiento Indexado con auto pre/post decremento/incremento

Este modo de direccionamiento tiene cuatro caminos para cambiar el valore en el registro de índice base como parte de la ejecución de la instrucción. El registro de índice pude ser incrementado o decremento por un valor entero antes o después de que el indexamiento se lleve a cabo. El registro de índice base puede ser X, Y o SP.
Las versiones de pre-decremento y pre-incremento de este modo de direccionamiento ajustan el valor en el registro de índice antes de accesar la localidad de memoria que va a ser afectada por la instrucción, - el registro de índice retiene el valor cambiado después de que la instrucción se ejecute..
Las versiones de post-decremento y post-incremento de este modo de direccionamiento usan el valor inicial en el registro de índice para accesar la localidad de memoria que va a ser afectada por la instrucción, después cambia el valor del registro de índice.

El CPU12 permite que el registro de índice puede ser incrementado o decrementado por cualquier valor entero en el rango de –8 a –1, o 1 a 8. El valor no necesita estar relacionado con el tamaño del operando de la instrucción que se esta ejecutando. Estas instrucciones pueden ser usadas para incorporar un ajuste en el registro de índice mas bien que usar una instrucción adicional e incrementar el tiempo de ejecución. Este modo de direccionamiento también es usado para realizar operaciones en estructuras de series de datos en memoria.
Cuando una instrucción LEAS, LEX o LEY es ejecutada usando este modo de direccionamiento, y la operación modifica el valor del registro de índice, el valor final del registro ese valor que va a ser usado para las operaciones de acceso a memoria.
Ejemplos

```
STAA    1,-SP       ;equivalente a PSHA
STX     2,-SP       ;equivalente a PSHX
LDX     2,SP+       ;equivalente a PULX
LDAA    1,SP+       ;equivalente a PULA
MOVW    2,X+,4,+Y
```

Este ejemplo demuestra como se trabaja con estructuras de datos mayores en bytes y words. Con esta instrucción en un loop, es posible mover palabras de datos de una lista teniendo una palabra por entrada en una segunda tabla que tiene 4 bytes por elemento de la taba. En este ejemplo el apuntador es actualizado después de que el dato es leído de memoria (post-incremento) mientras que el apuntador destino es actualizado después de que es usado para accesar memoria (pre-incremento)

Direccionamiento Indexado con offset acumulador

En este modo de direccionamiento, la dirección efectiva es la suma de los valores en el registro de índice base y una de los acumuladores. El valor en el registro de índice no se cambie. El registro de índice puede ser X, Y, SP o PC y el acumulador pude ser uno de los acumuladores de 8 bits (A o B) o el acumulador D de 16 bits
Ejemplo

```
LDAA    B,X
```

Esta instrucción internamente suma B con X para formar la dirección en donde A va ser puesto. B y X no son cambiados por esta instrucción. La operación es similar a la siguiente combinación en el HC11
Ejemplo

```
ABX
LDAA    0,X
```

Sin embargo la secuencia de estas 2 instrucciones alteran el registro de índice. Si esta secuencia es parte de un loop donde B cambie en cada pasada, el registro de índice tiene que ser reubicado con el valor de referencia en cada pasada del loop.

Direccionamiento Indexado Indirecto usando el Acumulador D

Este modo de direccionamiento suma el valor del acumulador D al valor que tenga el registro de índice base, para formar la dirección de memoria que contenga el apuntodar de la localidad de memoria que va a ser afectada por la instrucción. El operando de la instrucción no apunta a la dirección de la localidad de memoria sobre la que se va a actuar, apunta a la localidad del apuntador de la dirección de memoria sobre la que se va a actuar. Los corchetes cuadrados distinguen este modo de direccionamiento del modo de idesado usando offset acumulador.
Ejemplos

```
JMP     [D,PC]
GO1     DC.W    LOCALIDAD1
GO2     DC.W    LOCALIDAD2
GO3     DC.W    LOCALIDAD3
```

Este ejemplo es un cálculo GOTO. El valor empieza en GO1 son las direcciones de destinos potenciales de las instrucciones de brinco. En el tiempo en el que la instrucción JMP [D,P,C] SE EJECUTA, PC apunta a la dirección G01, y D tiene uno de los valores $0000, $0002 o $0004 el cual es determinado por el programa antes de llegar a la instrucción JMP. Asumiendo que el valor en D sea $0002 la instrucción JMP suma el valor de D y PC para formar la dirección GO2 y brinca ahí. La localidades LOCALIDAD1 a LOCALIDAD3 son conocidas cuando se ensambla el programa pero el destino de JMP depende del valor que se calcule en D durante la ejecución del programa

Conjunto de Instrucciones

El set de instrucciones del CPU12 es mayor que el set de instrucciones del H68HC11. Códigos escritos para el H68HC11 pueden ser reensamblados y ejecutados en el CPU12 sin ningún cambio.
En la arquitectura del M68HC12 toda la memoria, y I/O están mapeados en 64K de espacio de memoria. Esto permite que el mimo set de instrucciones pueda ser usado para accesar memoria, I/O y registros de control. Las instrucciones carga, almacenamiento, transferencia, intercambio y movimiento facilitan el movimiento de datos a y de memoria a periféricos.

Instrucciones de carga y almacenamiento.

Las instrucciones de carga (load) copian el contenido de memoria en el acumulador o registro. El contenido de memoria no cambia con la operación. Las instrucciones de carga afectan los bits de código de condición pero no se necesitan instrucciones de test para revisar los valores cargados para condiciones de negativo o cero.
Las intrusiones de almacenamiento (store) copian el contenido del registro del CPU en memoria. El contenido de registro/acumulador no cambian con la operación. Estas instrucciones automáticamente actualizan los bits N y Z del código de condición, lo cual elimina la necesitas de separar instrucciones de test en algunos programas

Table 5-1 Load and Store Instructions

Load Instructions		
Mnemonic	Function	Operation
LDAA	Load A	(M) ⇒ A
LDAB	Load B	(M) ⇒ B
LDD	Load D	(M : M + 1) ⇒ (A B)
LDS	Load SP	(M : M + 1) ⇒ SP
LDX	Load Index Register X	(M : M + 1) ⇒ X
LDY	Load Index Register Y	(M : M + 1) ⇒ Y
LEAS	Load Effective Address Into SP	Effective Address ⇒ SP
LEAX	Load Effective Address Into X	Effective Address ⇒ X
LEAY	Load Effective AddRess Into Y	Effective Address ⇒ Y
Store Instructions		
Mnemonic	Function	Operation
STAA	Store A	(A) ⇒ M
STAB	Store B	(B) ⇒ M
STD	Store D	(A) ⇒ M, (B) ⇒ M + 1
STS	Store SP	(SP) ⇒ M M + 1
STX	Store X	(X) ⇒ M , M + 1
STY	Store Y	(Y) ⇒ M M + 1

Instrucciones de transferencia e intercambio

Las instrucciones de transferencia copian el contenido de un registro o acumulador en otro registro o acumulador. El contenido de la fuente no se cambia con la operación. TFR es una instrucción de transferencia universal, pero otros nemónicos son aceptados por compatibilidad con el M68HC11.

Las instrucciones de intercambio intercambian el contenido de un par de registros o acumuladores.

Table 5-2 Transfer and Exchange Instructions

Transfer Instructions		
Mnemonic	Function	Operation
TAB	Transfer A To B	(A) ⇒ B
TAP	Transfer A To CCR	(A) ⇒ CCR
TBA	Transfer B To A	(B) ⇒ A
TFR	Transfer Register To Register	(A, B, CCR, D, X, Y, or SP) ⇒ A, B, CCR, D, X, Y, or SP
TPA	Transfer CCR To A	(CCR) ⇒ A
TSX	Transfer SP To X	(SP) ⇒ X
TSY	Transfer SP To Y	(SP) ⇒ Y
TXS	Transfer X To SP	(X) ⇒ SP
TYS	Transfer Y To SP	(Y) ⇒ SP
Exchange Instructions		
Mnemonic	Function	Operation
EXG	Exchange Register To Register	(A, B, CCR, D, X, Y, or SP) ⇔ (A, B, CCR, D, X, Y, or SP)
XGDX	Exchange D With X	(D) ⇔ (X)
XGDY	Exchange D With Y	(D) ⇔ (Y)
Sign Extension Instruction		
Mnemonic	Function	Operation
SEX	Sign Extend 8-bit Operand	(A, B, CCR) ⇒ X, Y, or SP

Instrucciones de Movimiento

Estas instrucciones mueven bytes de dato o palabras de una fuente (M_1,M:M+1_1) a un destino en memoria (M_2,M:M+1_2). Seis combinaciones de direccionamiento para especificar la fuente y destino son permitidos, inmediato, extendido e indexado (IMM⇒EXT, IMM⇒IDX, EXT⇒EXT, EXT⇒IDX, IDX⇒EXT, IDX⇒IDX)

Table 5-3 Move Instructions

Mnemonic	Function	Operation
MOVB	Move Byte (8-bit)	(M_1) ⇒ M_2
MOVW	Move Word (16-bit)	(M M + 1_1) ⇒ M · M + 1_2

Instrucciones de suma y resta

Sumas signadas y no signadas de 8 y 16 bits pueden se llevadas a cabo entre registros o entre registros y memoria. Instrucciones especiales soportan cálculos indexados. Instrucciones que suman el carry del CCR facilitan el cálculo con precisión múltiple
Restas de 8 o 16 bits signados y no signados pueden ser llevadas a cabo entre registros o entre registro y memoria. Instrucciones especiales soportan el cálculos indexado. Instrucciones que restan el carry del CCR facilitan al cálculo con precisión múltiple.

Addition Instructions		
Mnemonic	Function	Operation
ABA	Add A To B	(A) + (B) ⇒ A
ABX	Add B To X	(B) + (X) ⇒ X
ABY	Add B To Y	(B) + (Y) ⇒ Y
ADCA	Add With Carry To A	(A) + (M) + C ⇒ A
ADCB	Add With Carry To B	(B) + (M) + C ⇒ B
ADDA	Add Without Carry To A	(A) + (M) ⇒ A
ADDB	Add Without Carry To B	(B) + (M) ⇒ B
ADDD	Add To D	(A B) + (M · M + 1) ⇒ A B
Subtraction Instructions		
Mnemonic	Function	Operation
SBA	Subtract B From A	(A) − (B) ⇒ A
SBCA	Subtract With Borrow From A	(A) − (M) − C ⇒ A
SBCB	Subtract With Borrow From B	(B) − (M) − C ⇒ B
SUBA	Subtract Memory From A	(A) − (M) ⇒ A
SUBB	Subtract Memory From B	(B) − (M) ⇒ B
SUBD	Subtract Memory From D (A B)	(D) − (M M + 1) ⇒ D

Instrucciones binarias de código decimal

Para sumar operando en código decimal, se usan las instrucciones de suma que ponen el bit de half-carry en el CCR, y ajustan el resultado con la instrucción DDA.

Table 5-5 BCD Instructions

Mnemonic	Function	Operation
ABA	Add B To A	(A) + (B) ⇒ A
ADCA	Add With Carry To A	(A) + (M) + C ⇒ A
ADCB	Add With Carry To B	(B) + (M) + C ⇒ B
ADDA	Add Memory To A	(A) + (M) ⇒ A
ADDB	Add Memory To B	(B) + (M) ⇒ B
DAA	Decimal Adjust A	(A)10

Instrucciones de decremento e incremento

Estas instrucciones optimizan la operaciones de suma y resta de 8 y 16 bits. Estas son generalmente usadas para implementar contadores. Debido a que no afectan el bit de carry en el CCR, son muy usadas para loops de contadores en rutinas de múltiple precisión.

Table 5-6 Decrement and Increment Instructions

Decrement Instructions		
Mnemonic	Function	Operation
DEC	Decrement Memory	(M) – $01 ⇒ M
DECA	Decrement A	(A) – $01 ⇒ A
DECB	Decrement B	(B) – $01 ⇒ B
DES	Decrement SP	(SP) – $0001 ⇒ SP
DEX	Decrement X	(X) – $0001 ⇒ X
DEY	Decrement Y	(Y) – $0001 ⇒ Y
Increment Instructions		
Mnemonic	Function	Operation
INC	Increment Memory	(M) + $01 ⇒ M
INCA	Increment A	(A) + $01 ⇒ A
INCB	Increment B	(B) + $01 ⇒ B
INS	Increment SP	(SP) + $0001 ⇒ SP
INX	Increment X	(X) + $0001 ⇒ X
INY	Increment Y	(Y) + $0001 ⇒ Y

Instrucciones de comparación y test

Instrucciones de comparación y test realizan una resta entre un par de registros o entres registro y memoria. El resultado no se almacena, pero los códigos de condición se actualizan con esta operación. Estas instrucciones son usadas generalmente para establecer condiciones para instrucciones de brinco. En esta arquitectura, la mayoría de las instrucciones actualizan los bits de código de condición automáticamente, por lo que a veces es innecesario separar las instrucciones de test y comparación

Table 5-7 Compare and Test Instructions

Compare Instructions		
Mnemonic	Function	Operation
CBA	Compare A To B	(A) – (B)
CMPA	Compare A To Memory	(A) – (M)
CMPB	Compare B To Memory	(B) – (M)
CPD	Compare D To Memory (16-bit)	(A · B) – (M · M + 1)
CPS	Compare SP To Memory (16-bit)	(SP) – (M M + 1)
CPX	Compare X To Memory (16-bit)	(X) – (M M + 1)
CPY	Compare Y To Memory (16-bit)	(Y) – (M . M + 1)
Test Instructions		
Mnemonic	Function	Operation
TST	Test Memory For Zero Or Minus	(M) – $00
TSTA	Test A For Zero Or Minus	(A) – $00
TSTB	Test B For Zero Or Minus	(B) – $00

Instrucciones de lógica booleana

Estas instrucciones llevan a cabo las operaciones lógicas entres los acumuladores de 8 bits o el CCR y el valor en memoria. Realiza las funciones AND, OR y OR Exclusiva.

Instrucciones de complemento, limpia y negado
Cada una de estas operaciones realiza una operación binaria específica en un valor en el acumulador o en memoria. Las operaciones de limpia ponen un cero, las operaciones de complemento reemplazan el valor con el complemento a uno y la operación de negado reemplazan el valor con el valor de complemento a 2.

Table 5-8 Boolean Logic Instructions

Mnemonic	Function	Operation
ANDA	AND A With Memory	(A) • (M) ⇒ A
ANDB	AND B With Memory	(B) • (M) ⇒ B
ANDCC,	AND CCR With Memory (Clear CCR bits)	(CCR) • (M) ⇒ CCR
EORA	Exclusive OR A With Memory	(A) ⊕ (M) ⇒ A
EORB	Exclusive OR B With Memory	(B) ⊕ (M) ⇒ B
ORAA	OR A With Memory	(A) + (M) ⇒ A
ORAB	OR B With Memory	(B) + (M) ⇒ B
ORCC	OR CCR With Memory (Set CCR bits)	(CCR) + (M) ⇒ CCR

Instrucciones de complemento, limpia y negado

Cada una de estas instrucciones llevan a cabo una operación binaria específica sobre un valor en el acumulador o memoria. Operaciones de limpia ponen el valor en cero, la operación de complemento reemplaza el valor por su complemento a uno, y la operación de negado reemplaza el valor por su complemento a dos.

Table 5-9 Clear, Complement, and Negate Instructions

Mnemonic	Function	Operation
CLC	Clear C Bit In CCR	0 ⇒ C
CLI	Clear I Bit In CCR	0 ⇒ I
CLR	Clear Memory	$00 ⇒ M
CLRA	Clear A	$00 ⇒ A
CLRB	Clear B	$00 ⇒ B
CLV	Clear V bit in CCR	0 ⇒ V
COM	One's Complement Memory	$FF – (M) ⇒ M or $(\overline{M})$ ⇒ M
COMA	One's Complement A	$FF – (A) ⇒ M or $(\overline{A})$ ⇒ A
COMB	One's Complement B	$FF – (B) ⇒ M or $(\overline{B})$ ⇒ B
NEG	Two's Complement Memory	$00 – (M) ⇒ M or $(\overline{M}) + 1$ ⇒ M
NEGA	Two's Complement A	$00 – (A) ⇒ A or $(\overline{A}) + 1$ ⇒ A
NEGB	Two's Complement B	$00 – (B) ⇒ B or $(\overline{B}) + 1$ ⇒ B

Instrucciones de multiplicación y División

Estas son instrucciones de multiplicación de 8 o 16 bits signados y no signadas. La multiplicación de 8 bits obtiene un resultado de 16 bits. Y la operación de 16 bits obtiene un resultado de 32 bits.

Las instrucciones enteras y fraccionarias tienen un dividendo, divisor y resultado y residuo de 16 bits. La división extendida usa un dividendo de 32 bits y un divisor de 16 para entregar un resultado de 16 bits y un residuo de 16 bits.

Table 5-10 Multiplication and Division Instructions

Multiplication Instructions		
Mnemonic	Function	Operation
EMUL	16 By 16 Multiply (Unsigned)	(D) × (Y) ⇒ Y : D
EMULS	16 By 16 Multiply (Signed)	(D) × (Y) ⇒ Y : D
MUL	8 By 8 Multiply (Unsigned)	(A) × (B) ⇒ A : B
Division Instructions		
Mnemonic	**Function**	**Operation**
EDIV	32 By 16 Divide (Unsigned)	(Y : D) ÷ (X) Quotient ⇒ Y Remainder ⇒ D
EDIVS	32 By 16 Divide (Signed)	(Y : D) ÷ (X) Quotient ⇒ Y Remainder ⇒ D
FDIV	16 By 16 Fractional Divide	(D) ÷ (X) ⇒ X remainder ⇒ D
IDIV	16 By 16 Integer Divide (Unsigned)	(D) ÷ (X) ⇒ X remainder ⇒ D
IDIVS	16 By 16 Integer Divide (Signed)	(D) ÷ (X) ⇒ X remainder ⇒ D

Instrucciones de manipulación y test de bit

Estas instrucciones usan un valor de mascara para verificar o cambiar el valor de un bit individual en el acumulador o en memoria. BITA y BITB revisan los bits sin alterar el valor al efectuar su operación.

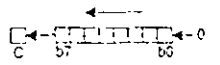
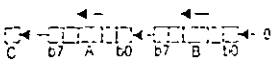
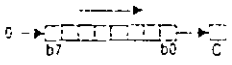
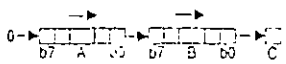
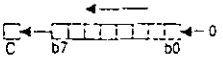
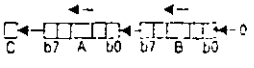
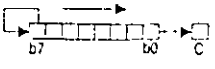
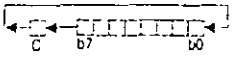
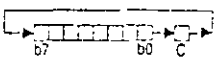
Table 5-11 Bit Test and Manipulation Instructions

Mnemonic	Function	Operation
BCLR	Clear Bits in Memory	(M) • ($\overline{mm}$) ⇒ M
BITA	Bit Test A	(A) • (M)
BITB	Bit Test B	(B) • (M)
BSET	Set Bits in Memory	(M) + (mm) ⇒ M

Instrucciones de corrimiento y rotación

Hay instrucciones de corrimiento y rotación para todos los acumuladores y bytes de memoria. Todas las de corrimiento usan el bit C para facilitar las operaciones de múltiples bytes.

Table 5-12 Shift and Rotate Instructions

Logical Shifts		
Mnemonic	Function	Operation
LSL LSLA LSLB	Logic Shift Left Memory Logic Shift Left A Logic Shift Left B	
LSLD	Logic Shift Left D	
LSR LSRA LSRB	Logic Shift Right Memory Logic Shift Right A Logic Shift Right B	
LSRD	Logic Shift Right D	
Arithmetic Shifts		
Mnemonic	Function	Operation
ASL ASLA ASLB	Arithmetic Shift Left Memory Arithmetic Shift Left A Arithmetic Shift Left B	
ASLD	Arithmetic Shift Left D	
ASR ASRA ASRB	Arithmetic Shift Right Memory Arithmetic Shift Right A Arithmetic Shift Right B	
Rotates		
Mnemonic	Function	Operation
ROL ROLA ROLB	Rotate Left Memory Through Carry Rotate Left A Through Carry Rotate Left B Through Carry	
ROR RORA RORB	Rotate Right Memory Through Carry Rotate Right A Through Carry Rotate Right B Through Carry	

Instrucciones de Máximo y Mínimo

Estas instrucciones son usadas para hacer comparaciones entre el acumulador y una localidad de memoria. Estas instrucciones pueden ser usadas para operaciones de programación lineal.
Las instrucciones MAX y MIN utilizan al acumulador A para llevar a cabo comparaciones de 8 bits mientras que las instrucciones EMAX y EMIN usan el acumulador D para llevar a cabo comparaciones de 16 bits. El resultado (máximo o mínimo) pude ser colocado en el acumulador (EMAXD, EMIND, MAXA, MINA) en dirección de memoria (EMAXM, EMINM, MAXAM, MINM)

Table 5-14 Minimum and Maximum Instructions

Minimum Instructions		
Mnemonic	Function	Operation
EMIND	MIN Of 2 Unsigned 16-bit Values Result to Accumulator	MIN ((D), (M · M + 1)) ⇒ D
EMINM	MIN Of 2 Unsigned 16-bit Values Result to Memory	MIN ((D) (M M + 1)) ⇒ M M+1
MINA	MIN Of 2 Unsigned 8-bit Values Result to Accumulator	MIN ((A), (M)) ⇒ A
MINM	MIN Of 2 Unsigned 8-bit Values Result to Memory	MIN ((A), (M)) ⇒ M
Maximum Instructions		
Mnemonic	Function	Operation
EMAXD	MAX Of 2 Unsigned 16-bit Values Result to Accumulator	MAX ((D), (M M + 1)) ⇒ D
EMAXM	MAX Of 2 Unsigned 16-bit Values Result to Memory	MAX ((D), (M : M + 1)) ⇒ M M + 1
MAXA	MAX Of 2 Unsigned 8-bit Values Result to Accumulator	MAX ((A), (M)) ⇒ A
MAXM	MAX Of 2 Unsigned 8-bit Values Result to Memory	MAX((A), (M)) ⇒ M

Instrucciones de multiplicar y acumular

La instrucción EMACS multiplica 2 operandos de 16 bits de memoria y acumula el resultado en 32 bits en una tercera localidad de memoria. EMACS puede ser usada para implementar filtros digitales simples o rutinas de fuzificación que usen operandos de 16 bits. Las instrucciones WAV incorpora operaciones de 8 o 16 bits de multiplica y acumula para obtener el numerador de un calculo de peso promedio. La instrucción EMACS pude automatizar esta porción de promedio cuando los operandos usados son de 16 bits.

Table 5-15 Multiply and Accumulate Instructions

Mnemonic	Function	Operation
EMACS	Multiply And Accumulate (Signed) 16 × 16 Bit ⇒ 32 Bit	$((M_{(X)}\ M_{(X+1)}) \times (M_{(Y)}\ M_{(Y+1)})) + (M \sim M + 3) \Rightarrow M \sim M + 3$

Instrucciones de interpolación de tabla

Las instrucciones TBL y ETBL interpolan valores de tablas almacenadas en memoria. Cualquier función que pueda ser representada como una serie de ecuaciones lineales pude ser representada por una tabla de tamaño apropiado. La interpolación pude ser usada para muchos propósitos, incluyendo funciones de membresía tabulares. TBL usa tablas de entrada de 8 bits y entrega resultados de 8 bits. ETBL usa tablas de entrada de 16 bits y entrega resultados de 16 bits. Usa el modo de direccionamiento indexado da gran flexibilidad a la estructura de tablas.

Table 5-16 Table Interpolation Instructions

Mnemonic	Function	Operation
ETBL	16-bit Table Lookup And Interpolate (no indirect addressing modes allowed)	(M : M + 1) + [(B) × ((M + 2 : M + 3) – (M : M + 1))] => D Initialize B, and index before ETBL <ea> points to the first table entry (M : M + 1) B is fractional part of lookup value
TBL	8-bit Table Lookup And Interpolate (no indirect addressing modes allowed)	(M) + [(B) × ((M + 1) – (M))] => A Initialize B, and index before TBL <ea> points to the first 8-bit table entry (M) B is fractional part of lookup value.

Instrucciones de brinco

Las instrucciones de brinco ocasionan que la secuencia cambie cuando se presenten condiciones específicas. El CPU12 utiliza tres tipos de intrusiones de brinco: Brinco corto, brinco largo, y brincos condicionales.

Instrucciones de brinco corto

Cuando se alcanza una condición específica, un offset de 8 bits se suma al valor del contador de programa, y la ejecución del programa continua en esta nueva dirección.

En rango numérico par un brinco corto de $80 (-128) a $7F (127) de la dirección de la siguiente localidad de memoria después del valor del offset

Instrucciones de brinco largo

Cuando se alcanza una condición específica, un offset de 16 bits se suma al valor del contador de programa, y la ejecución del programa continua en esta nueva dirección.

En rango numérico par un brinco corto de $8000 (-32768) a $7FFF (32767) de la dirección de la siguiente localidad de memoria después del valor del offset

Table 5-17 Short Branch Instructions

Unary Branches		
Mnemonic	**Function**	**Equation or Operation**
BRA	Branch Always	1 = 1
BRN	Branch Never	1 = 0
Simple Branches		
Mnemonic	**Function**	**Equation or Operation**
BCC	Branch if Carry Clear	C = 0
BCS	Branch if Carry Set	C = 1
BEQ	Branch if Equal	Z = 1
BMI	Branch if Minus	N = 1
BNE	Branch if Not Equal	Z = 0
BPL	Branch if Plus	N = 0
BVC	Branch if Overflow Clear	V = 0
BVS	Branch if Overflow Set	V = 1

Unsigned Branches			
Mnemonic	**Function**	**Relation**	**Equation or Operation**
BHI	Branch if Higher	R > M	C + Z = 0
BHS	Branch if Higher or Same	R ≥ M	C = 0
BLO	Branch if Lower	R < M	C = 1
BLS	Branch if Lower or Same	R ≤ M	C + Z = 1

Signed Branches			
Mnemonic	**Function**	**Relation**	**Equation or Operation**
BGE	Branch if Greater than or Equal	R ≥ M	N ⊕ V = 0
BGT	Branch if Greater Than	R > M	Z + (N ⊕ V) = 0
BLE	Branch if Less than or Equal	R ≤ M	Z + (N ⊕ V) = 1
BLT	Branch if Less Than	R < M	N ⊕ V = 1

Table 5-18 Long Branch Instructions

Unary Branches		
Mnemonic	**Function**	**Equation or Operation**
LBRA	Long Branch Always	1 = 1
LBRN	Long Branch Never	1 = 0
Simple Branches		
Mnemonic	**Function**	**Equation or Operation**
LBCC	Long Branch If Carry Clear	C = 0
LBCS	Long Branch If Carry Set	C = 1
LBEQ	Long Branch If Equal	Z = 1
LBMI	Long Branch If Minus	N = 1
LBNE	Long Branch If Not Equal	Z = 0
LBPL	Long Branch If Plus	N = 0
LBVC	Long Branch If Overflow Clear	V = 0
LBVS	Long Branch If Overflow Set	V = 1
Unsigned Branches		
Mnemonic	**Function**	**Equation or Operation**
LBHI	Long Branch If Higher	C + Z = 0
LBHS	Long Branch If Higher Or Same	C = 0
LBLO	Long Branch If Lower	Z = 1
LBLS	Long Branch If Lower Or Same	C + Z = 1
Signed Branches		
Mnemonic	**Function**	**Equation or Operation**
LBGE	Long Branch If Greater Than Or Equal	N ⊕ V = 0
LBGT	Long Branch If Greater Than	Z + (N ⊕ V) = 0
LBLE	Long Branch If Less Than Or Equal	Z + (N ⊕ V) = 1
LBLT	Long Branch If Less Than	N ⊕ V = 1

Instrucciones de brincos condicionales.

Estos brincos se llevan a cabo cuando los bits en memoria están en un estado específico. La mascara del operando es usado para verificar la localidad. Si todos los bits de la localidad que corresponden a unos en la mascara están en 1 (BRSET) o en 0 (BRCLER) el brinco se lleva a cabo.

En rango numérico par un brinco corto de $80 (-128) a $7F (127) de la dirección de la siguiente localidad de memoria después del valor del offset

Table 5-19 Bit Condition Branch Instructions

Mnemonic	Function	Equation or Operation
BRCLR	Branch if Selected Bits Clear	$(M) \bullet (mm) = 0$
BRSET	Branch if Selected Bits Set	$(\overline{M}) \bullet (mm) = 0$

Instrucciones de loop rudimentario

Pueden ser llavados a cabo través de contadores. Las instrucciones verifican que el valor del contador en el registro o acumulador (A, B, D, X, Y o SP) sea cero o un valor diferente de cero como condición de brinco. Se usan versiones de predecremento, pre-incremento y solo-verificar.

En rango numérico par un brinco corto de $80 (-128) a $7F (127) de la dirección de la siguiente localidad de memoria después del valor del offset

Table 5-20 Loop Primitive Instructions

Mnemonic	Function	Equation or Operation
DBEQ	Decrement Counter and Branch if = 0 (counter = A, B, D, X, Y, or SP)	(counter) – 1 ⇒ counter If (counter) = 0, then Branch else Continue to next instruction
DBNE	Decrement Counter and Branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	(counter) – 1 ⇒ counter If (counter) not = 0, then Branch else Continue to next instruction
IBEQ	Increment Counter and Branch if = 0 (counter = A, B, D, X, Y, or SP)	(counter) + 1 ⇒ counter If (counter) = 0, then Branch else Continue to next instruction
IBNE	Increment Counter and Branch if ≠ 0 (counter = A, B, D, X, Y, or SP)	(counter) + 1 ⇒ counter If (counter) not = 0, then Branch else Continue to next instruction
TBEQ	Test Counter and Branch if = 0 (counter = A, B, D, X,Y, or SP)	If (counter) = 0, then Branch else Continue to next instruction
TBNE	Test Counter and Branch if ≠ 0 (counter = A, B, D, X,Y, or SP)	If (counter) not = 0, then Branch else Continue to next instruction

Instrucciones de brinco y subrutinas

Las instrucciones de brinco cambian inmediatamente la secuencia. La instrucción JMP carga al PC con el dirección dentro de los 64 K de memoria y la ejecución del programa continua en esta dirección. La dirección puede estar dada en forma absoluta 16 bits o determinada de varias formas por el direccionamiento indexado.

Las instrucciones de subrutina optimizan el proceso de transferencia de control a un segmento de código que lleve a cabo un tarea particular. Se pueden utilizar para iniciar una

subrutina llamadas cortas (BSR) llamadas largas (JSR) o llamadas a memoria expandida (CALL).
La dirección de retorno se almacena en el stack, cuando se inicia la ejecución de la subrutina. Las subrutinas en el espacio normal de memoria (64K) son terminadas con la instrucción RTS . RTS saca del stack el valor de la dirección de retorno.
La instrucción CALL es usada cuando se trabaja con memoria expandida. CALL coloca en el stack el valor del registro PPAGE y la dirección de retorno, entonces escribe un nuevo valor en PPAGE para seleccionar la página de memoria donde esta la subrutina. El valor de la página es un operando inmediato y todos los modos de direccionamiento excepto el indexado indirecto, en estos modos, el operando apunta a localidades de memoria en donde la esta el nuevo valor de pagina y la dirección de subrutina son colocados. La instrucción RTC es usada para terminar las subrutinas en memoria expandida. RTC saca del stack el valor de PPAGE y regresa la dirección para continuar con instrucción siguiente a CALL.

Table 5-21 Jump and Subroutine Instructions

Mnemonic	Function	Operation
BSR	Branch to Subroutine	$SP - 2 \Rightarrow SP$ $RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ Subroutine address $\Rightarrow$ PC
CALL	Call Subroutine in Expanded Memory	$SP - 2 \Rightarrow SP$ $RTN_H\ RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP$ $(PPAGE) \Rightarrow M_{(SP)}$ Page $\Rightarrow$ PPAGE Subroutine address $\Rightarrow$ PC
JMP	Jump	Subroutine Address $\Rightarrow$ PC
JSR	Jump to Subroutine	$SP - 2 \Rightarrow SP$ $RTN_H\ \ RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ Subroutine address $\Rightarrow$ PC
RTC	Return from Call	$M_{(SP)}\ \ M_{(SP+1)} \Rightarrow PC_H\ \ PC_L$ $SP + 2 \Rightarrow SP$
RTS	Return from Subroutine	$M_{(SP)} \Rightarrow$ PPAGE $SP + 1 \Rightarrow SP$ $M_{(SP)}\ \ M_{(SP+1)} \Rightarrow PC_H\ \ PC_L$ $SP + 2 \Rightarrow SP$

Instrucciones de interrupción.
Las instrucciones de interrupción manejan la transferencia de control a una rutina que lleva a cabo tareas críticas. Las interrupciones por software son un tipo de excepción.
La instrucción SWI inicializa el proceso de excepción sincrono. Primero, el valor del PC de retorno es almacenado en el stack. Después el contexto de CPU es colocado en el stack, continua la ejecución en la dirección apuntada por el vector SWI.
La ejecución de la instrucción SWI causa una interrupción sin un requerimiento de servicio de interrupción. SWI no es inhibida por las banderas globales I y X del CCR, y la ejecución de un conjunto de SWI pone el bit de mascara I del CCR en cero. Cuando se ejecuta la instrucción RTI al final del servicio de rutina SWI se restablece el contexto del CPU.
La instrucción RTI es usada para terminar cualquier tipo de interrupción. incluyendo las rutinas de servicio de interrupción. RTI primero restablece el CCR, B:A, X y Y, después regresa a la dirección guardada en el stack. Si no hay otra interrupción pendiente, se continua con la ejecución normal.

Table 5-22 Interrupt Instructions

Mnemonic	Function	Operation
RTI	Return from Interrupt	$M_{(SP)} \Rightarrow CCR$, $SP + 1 \Rightarrow SP$ $M_{(SP)} : M_{(SP+1)} \Rightarrow B : A$, $SP + 2 \Rightarrow SP$ $M_{(SP)} \; M_{(SP+1)} \Rightarrow X_H : X_L$, $SP + 2 \Rightarrow SP$ $M_{(SP)} : M_{(SP+1)} \Rightarrow Y_H \; Y_L$, $SP + 2 \Rightarrow SP$ $M_{(SP)} \; M_{(SP+1)} \Rightarrow PC_H \; PC_L$; $SP + 2 \Rightarrow SP$
SWI	Software Interrupt	$SP - 2 \Rightarrow SP$, $RTN_H \; RTN_L \Rightarrow M_{(SP)} \; M_{(SP+1)}$ $SP - 2 \Rightarrow SP$, $Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$, $X_H \; X_L \Rightarrow M_{(SP)} \; M_{(SP+1)}$ $SP - 2 \Rightarrow SP$, $B \; A \Rightarrow M_{(SP)} \; M_{(SP+1)}$ $SP - 1 \Rightarrow SP$, $CCR \Rightarrow M_{(SP)}$
TRAP	Software Interrupt	$SP - 2 \Rightarrow SP$; $RTN_H \; RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP$, $Y_H \; Y_L \Rightarrow M_{(SP)} \; M_{(SP+1)}$ $SP - 2 \Rightarrow SP$; $X_H : X_L \Rightarrow M_{(SP)} \; M_{(SP+1)}$ $SP - 2 \Rightarrow SP$, $B \; A \Rightarrow M_{(SP)} \; M_{(SP+1)}$ $SP - 1 \Rightarrow SP$, $CCR \Rightarrow M_{(SP)}$

Instrucciones de manipulación de índices

Estas instrucciones llevan a cabo operaciones de 8 o 16 bits en los tres registros de índice y los acumuladores, otros registros o memoria.

Table 5-23 Index Manipulation Instructions

Addition Instructions		
Mnemonic	Function	Operation
ABX	Add B to X	(B) + (X) ⇒ X
ABY	Add B to Y	(B) + (Y) ⇒ Y
Compare Instructions		
Mnemonic	Function	Operation
CPS	Compare SP to Memory	(SP) – (M M + 1)
CPX	Compare X to Memory	(X) – (M M + 1)
CPY	Compare Y to Memory	(Y) – (M M + 1)
Load Instructions		
Mnemonic	Function	Operation
LDS	Load SP from Memory	M M+1 ⇒ SP
LDX	Load X from Memory	(M M + 1) ⇒ X
LDY	Load Y from Memory	(M M + 1) ⇒ Y
LEAS	Load Effective Address into SP	Effective Address ⇒ SP
LEAX	Load Effective Address into X	Effective Address ⇒ X
LEAY	Load Effective Address into Y	Effective Address ⇒ Y
Store Instructions		
Mnemonic	Function	Operation
STS	Store SP in Memory	(SP) ⇒ M M+1
STX	Store X in Memory	(X) ⇒ M M + 1
STY	Store Y in Memory	(Y) ⇒ M M + 1
Transfer Instructions		
Mnemonic	Function	Operation
TFR	Transfer Register to Register	(A, B, CCR, D, X, Y, or SP) ⇒ A, B, CCR, D, X, Y, or SP
TSX	Transfer SP to X	(SP) ⇒ X
TSY	Transfer SP to Y	(SP) ⇒ Y
TXS	Transfer X to SP	(X) ⇒ SP
TYS	Transfer Y to SP	(Y) ⇒ SP
Exchange Instructions		
Mnemonic	Function	Operation
EXG	Exchange Register to Register	(A, B, CCR, D, X, Y, or SP) ⇔ (A, B, CCR, D, X, Y, or SP)
XGDX	EXchange D with X	(D) ⇔ (X)
XGDY	EXchange D with Y	(D) ⇔ (Y)

Instrucciones de stack

Hay dos tipos de instrucciones con el stack. Las instrucciones con el stack pointer usan una forma especial de instrucciones matemáticas y de transferencia de datos para llevar a cabo la manipulación del stack pointer. Las operaciones de stack salvan información y regresan información del sistema de stack.

Table 5-24 Stacking Instructions

Stack Pointer Instructions		
Mnemonic	Function	Operation
CPS	Compare SP to Memory	(SP) – (M : M + 1)
DES	Decrement SP	(SP) – 1 ⇒ SP
INS	Increment SP	(SP) + 1 ⇒ SP
LDS	Load SP	(M : M + 1) ⇒ SP
LEAS	Load Effective Address into SP	Effective Address ⇒ SP
STS	Store SP	(SP) ⇒ M : M + 1
TSX	Transfer SP to X	(SP) ⇒ X
TSY	Transfer SP to Y	(SP) ⇒ Y
TXS	Transfer X to SP	(X) ⇒ SP
TYS	Transfer Y to SP	(Y) ⇒ SP
Stack Operation Instructions		
Mnemonic	Function	Operation
PSHA	Push A	(SP) – 1 ⇒ SP; (A) ⇒ $M_{(SP)}$
PSHB	Push B	(SP) – 1 ⇒ SP; (B) ⇒ $M_{(SP)}$
PSHC	Push CCR	(SP) – 1 ⇒ SP; (A) ⇒ $M_{(SP)}$
PSHD	Push D	(SP) – 2 ⇒ SP, (A : B) ⇒ $M_{(SP)}$: $M_{(SP+1)}$
PSHX	Push X	(SP) – 2 ⇒ SP, (X) ⇒ $M_{(SP)}$: $M_{(SP+1)}$
PSHY	Push Y	(SP) – 2 ⇒ SP, (Y) ⇒ $M_{(SP)}$: $M_{(SP+1)}$
PULA	Pull A	($M_{(SP)}$) ⇒ A, (SP) + 1 ⇒ SP
PULB	Pull B	($M_{(SP)}$) ⇒ B; (SP) + 1 ⇒ SP
PULC	Pull CCR	($M_{(SP)}$) ⇒ CCR, (SP) + 1 ⇒ SP
PULD	Pull D	($M_{(SP)}$: $M_{(SP+1)}$) ⇒ A : B; (SP) + 2 ⇒ SP
PULX	Pull X	($M_{(SP)}$: $M_{(SP+1)}$) ⇒ X, (SP) + 2 ⇒ SP
PULY	Pull Y	($M_{(SP)}$: $M_{(SP+1)}$) ⇒ Y; (SP) + 2 ⇒ SP

Instrucciones de calculo de apuntador e índice.

Las instrucciones de búsqueda de dirección efectiva permiten 5-, 8-, o 16 bits constantes, o el contenido de los acumuladores de 8 bits A y B o el acumulador D para ser sumado al contenido de los registros de índice X y Y, el SP o el PC

Table 5-25 Pointer and Index Calculation Instructions

Mnemonic	Function	Operation
LEAS	Load Result Of Indexed Addressing Mode Effective Address Calculation Into Stack Pointer	r ± Constant ⇒ SP or (r) + (Accumulator) ⇒ SP r = X, Y, SP, or PC
LEAX	Load Result Of Indexed Addressing Mode Effective Address Calculation Into X Index Register	r ± Constant ⇒X or (r) + (Accumulator) ⇒X r = X, Y, SP, or PC
LEAY	Load Result Of Indexed Addressing Mode Effective Address Calculation Into Y Index Register	r ± Constant ⇒Y or (r) + (Accumulator) ⇒ Y r = X, Y, SP, or PC

Instrucciones de los códigos de condición

Las instrucciones de los códigos de condición son formas especiales instrucciones de transferencia de datos o matemáticas que pueden ser usadas para cambiar las condiciones del registro de código de condición

Table 5-26 Condition Codes Instructions

Mnemonic	Function	Operation
ANDCC	Logical AND CCR with Memory	(CCR) • (M) ⇒ CCR
CLC	Clear C bit	0 ⇒ C
CLI	Clear I bit	0 ⇒ I
CLV	Clear V bit	0 ⇒ V
ORCC	Logical OR CCR with Memory	(CCR) + (M) ⇒ CCR
PSHC	Push CCR onto Stack	(SP) – 1 ⇒ SP ; (CCR) ⇒ $M_{(SP)}$
PULC	Pull CCR from Stack	($M_{(SP)}$) ⇒ CCR; (SP) + 1 ⇒ SP
SEC	Set C bit	1 ⇒ C
SEI	Set I bit	1 ⇒ I
SEV	Set V bit	1 ⇒ V
TAP	Transfer A to CCR	(A) ⇒ CCR
TPA	Transfer CCR to A	(CCR) ⇒ A

Instrucciones de stop y wait

Hay dos instrucciones que ponen al CPU en un estado inactivo y reduce el consumo de potencia.

La instrucción STOP pone en el stack la dirección de retorno y el contenido de los registros y acumuladores del CPU , entonces detiene todos los relojes del sistema

La instrucción WAIT coloca en el stack la dirección de retorno y el contenido de los registros y acumuladores del CPU , entonces espera algún servicio de interrupción: sin embargo, las señales de reloj continúan funcionando.

Ambas instrucciones necesitan que ocurra una interrupción o un reset antes de que se continúe con la ejecución normal. Requieren el mismo número de ciclos de reloj para continuar la ejecución normal después de haberse llevado a cabo el servicio de interrupción, restablecer después de STOP requiere un tiempo extra para que el oscilador alcance la velocidad de operación.

Table 5-27 Stop and Wait Instructions

Mnemonic	Function	Operation
STOP	Stop	SP – 2 ⇒ SP, RTN_H : RTN_L ⇒ $M_{(SP)}$ $M_{(SP+1)}$ SP – 2 ⇒ SP, Y_H : Y_L ⇒ $M_{(SP)}$ $M_{(SP+1)}$ SP – 2 ⇒ SP, X_H : X_L ⇒ $M_{(SP)}$ $M_{(SP+1)}$ SP – 2 ⇒ SP; B : A ⇒ $M_{(SP)}$: $M_{(SP+1)}$ SP – 1 ⇒ SP; CCR ⇒ $M_{(SP)}$ STOP CPU Clocks
WAI	Wait for Interrupt	SP – 2 ⇒ SP, RTN_H : RTN_L ⇒ $M_{(SP)}$ $M_{(SP+1)}$ SP – 2 ⇒ SP, Y_H : Y_L ⇒ $M_{(SP)}$ $M_{(SP+1)}$ SP – 2 ⇒ SP; X_H : X_L ⇒ $M_{(SP)}$ $M_{(SP+1)}$ SP – 2 ⇒ SP, B : A ⇒ $M_{(SP)}$ $M_{(SP+1)}$ SP – 1 ⇒ SP, CCR ⇒ $M_{(SP)}$

Instrucciones Difusas

El CPU12 incluye cuatro instrucciones que efectúan tareas específicas de lógica difusa. Además muchas mas instrucciones son muy usadas un programas de lógica difusa.
Las cuatro instrucciones son MEM que evalúa las funciones de membresía trapezoidales, REV y REVW, que efectúan el la evaluación pesos MIN-MAX de la regla, y WAV que efectúa el promedio de pesos en funciones de membresía singleton..

Table 5-13 Fuzzy Logic Instructions

Mnemonic	Function	Operation
MEM	Membership Function	μ (grade) $\Rightarrow M_{(Y)}$ (X) + 4 $\Rightarrow$ X; (Y) + 1 $\Rightarrow$ Y, A unchanged if (A) < P1 or (A) > P2, then μ = 0, else μ = MIN [((A) – P1) × S1, (P2 – (A)) × S2, $FF] where: A = current crisp input value X points to a four byte data structure that describes a trapezoidal membership function as base intercept points and slopes (P1, P2, S1, S2) Y points at fuzzy input (RAM location) See instruction details for special cases
REV	MIN-MAX Rule Evaluation	Find smallest rule input (MIN) Store to rule outputs unless fuzzy output is larger (MAX) Rules are unweighted Each rule input is an 8-bit offset from a base address in Y Each rule output is an 8-bit offset from a base address in Y $FE separates rule inputs from rule outputs $FF terminates the rule list REV can be interrupted
REVW	MIN-MAX Rule Evaluation	Find smallest rule input (MIN) Multiply by a rule weighting factor (optional) Store to rule outputs unless fuzzy output is larger (MAX) Each rule input is the 16-bit address of a fuzzy input Each rule output is the 16-bit address of a fuzzy output Address $FFFE separates rule inputs from rule outputs $FFFF terminates the rule list Weights are 8-bit values in a separate table REVW can be interrupted
WAV	Calculates Numerator (Sum of Products) and Denominator (Sum of Weights) for Weighted Average Calculation Results Are Placed In Correct Registers For EDIV immediately After WAV	$\sum_{i=1}^{B} S_i F_i \Rightarrow Y.D$ $\sum_{i=1}^{B} F_i \Rightarrow X$
wavr	Resumes Execution Of Interrupted WAV Instruction	Recover immediate results from stack rather than initializing them to 0.

Otras instrucciones que son muy usadas en las programas de lógica difusa con MINA, EMIND, MASM, EMAXM, TBL, ETBL Y EMACS. Para mayor resolución en los programas difusos, están las instrucciones matemáticas de precisión extendida.

Los requerimientos para implementar un soporte difuso en el CPU12 son pequeños y no requieren incrementar apreciablemente el soto para un usuario típico. El kernel de inferencia difuso en el CPU12 requiere unas quince líneas como mucho espacio de código y ejecuta 15 veces más rápido en comparación a un kernel implementado en con microcontrolador de propósito general.

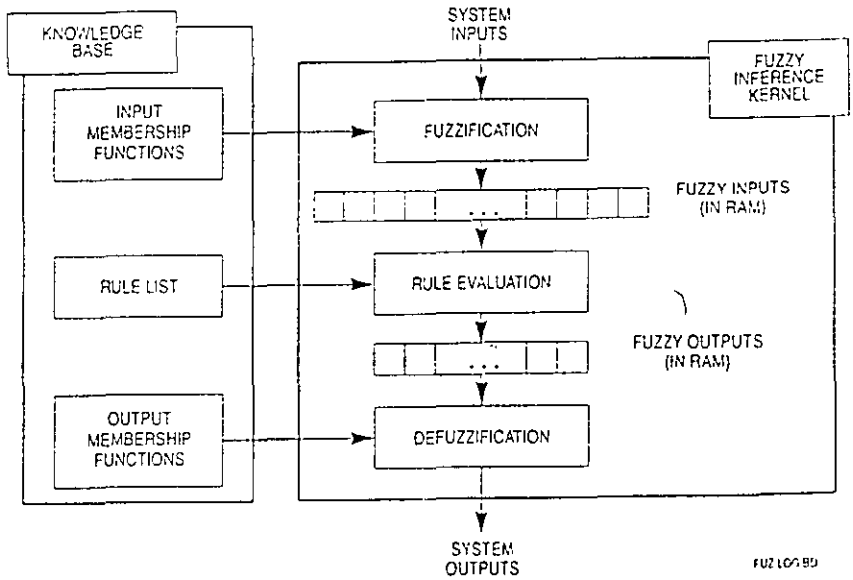


Fig.3 Diagrama bloques de un sistema de lógica difusa

Fuzificación (MEM)

Durante el paso de la fuzificación, el valor de la entrada corriente (actual) es comparada con las funciones de membresía de entrada almacenadas para determinar el grado en el cual cada etiqueta del sistema de entrada es verdadero. Esto esta acompañado de encontrar el valor de y para la entrada actual en la función de membresía trapezoidal para cada etiqueta del sistema de entrada. La instrucción MEM realiza este cálculo para una etiqueta de un sistema de entrada. Para el desarrollo completo de la tarea de fuzificación para un sistema, se ejecutan muchas instrucciones MEM, usualmente en una estructura de loop en el programa.

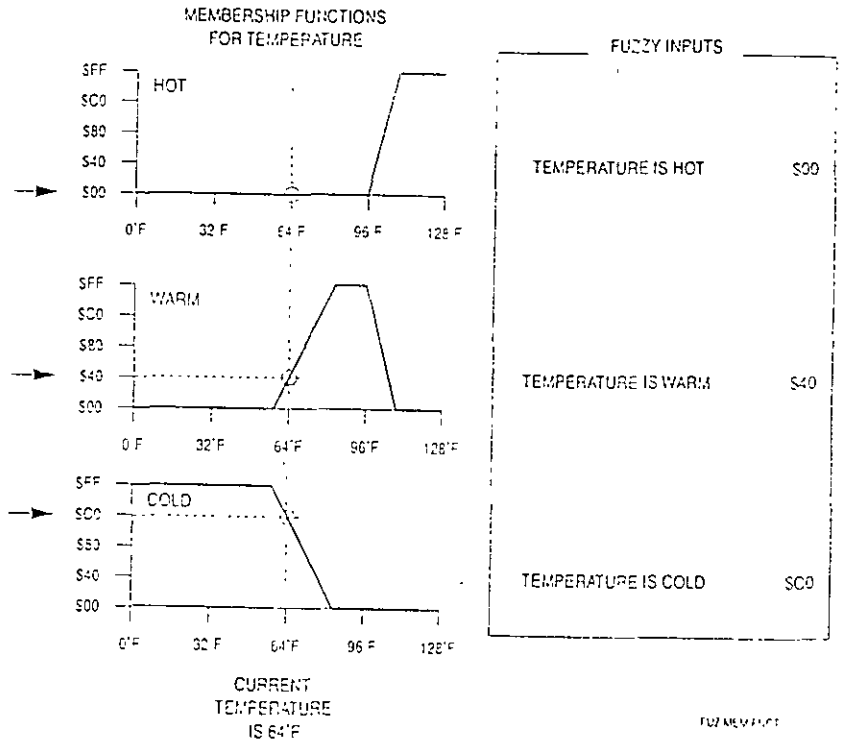


Fig. 4 Fuzificación usando funciones de membresía

La figura 4 muestra un sistema de tres funciones de membresía de entrada, una para cada etiqueta del sistema de entrada. El eje x de las tres funciones de membresía representan el rango posible de valores del la entrada del sistema. La línea vertical que cruza las tres funciones de membresía representa una entrada específica. El eje y representa el grado de verdad y varia de la completa falsedad ($00 o 0%) hasta la completa verdad ($FF o 100%). El valor y correspondiente a punto en el cual se intercepta la entrada en cada función de membresía, es el grado en el cual el valor de entrada corresponde a la etiqueta asociada para esta entrada del sistema. Por ejemplo, la expresión "temperatura caliente" es 25% verdadera ($40). El valor $40 es almacenado en alguna localidad de RAM y es llamado entrada difusa (en este caso, la entrada difusa para "temperatura caliente") Hay una localidad de RAM para cada entrada difusa (por cada etiqueta de cada sistema de entrada). Cuando la fuzificación empieza, el valor actual a la entrada está en un acumulador del CPU12, un registro de índice apunta a la primera función de membresía definida en la base de conocimientos, y el segundo registro de índice apunto a la primera entrada difusa en RAM. Por cada entrada difusa que es calculada ejecutando la instrucción MEM, el resultado se almacena como entrada difusa y ambos apuntadores se actualizan automáticamente para apuntar a los localidades asociadas con la siguiente entrada difusa. La instrucción MEM tiene cuidada para ir contando el numero de etiquetas por entrada al sistema y buscar el valor actual para cualquier subsecuente de las entradas al sistema.

El resultado final del paso de fuzificación es la tabla de entradas difusas que representa las condiciones presentes del sistema

Evaluación de reglas (REV y REVW)

La evaluación de reglas es el elemento central del programa de inferencia. Este paso del proceso es una lista de reglas de la base de conocimiento usando los valores actuales de la entrada difusa de TRAM para producir una lista de salidas difusas en RAM. Estas salidas difusas pueden ser pensadas como sugerencias previas o sin tratamiento para que la salida del sistema pueda responder la condición de entrada actual. Antes de que los resultados puedan ser aplicados, las salidas difusas deben ser procesadas, o defuzificadas, para producir un solo valor de salida tal que represente el efecto combinado de todas las salidas difusas.
El CPU12 ofrece 2 variaciones para la evaluación de reglas. La instrucción REV entrega una regla no valuada (todas las reglas son consideradas igualmente importantes). La instrucción REVW es similar pero permite que cada regla tenga un factor de peso separado el cual es almacenado en una estructura de datos paralela (por separado) en la base de conocimientos. Además de los pesos, las instrucciones de evaluación de reglas difieren, ya que las reglas están codificados en la base de conocimiento.
Suponga el ejemplo de la siguiente regla.

Si la temperatura es caliente y la presión es alta, entonces el calor (debería ser) apagado.

A primera vista, se ve como si esta regla estuviera en una forma compacta difícilmente entendible por un microcontrolador, pero es simple reducir esta regla en una lista pequeña de apuntadores de memoria. La porción izquierda de la regla es la declaración de una condición de entrada y la porción derecha es la declaración de la acción de salida
La porción izquierda está fingido para uno o mas antecedentes (en este caso dos) conectados por el operador difuso *and*. Cada expresión de antecedente consiste de nombre del sistema de entrada seguido por "*es*" seguido por el nombre de la etiqueta. La etiqueta debe ser definida por la función de membresía en la base de conocimientos. Cada expresión de antecedente corresponde a una entrada difusa en RAM. El conector "*y*" es el operador que permite conectar las expresiones antecedentes, no es necesario incluir esto en la codificación de reglas. Los antecedentes pueden ser codificados como una lista de apuntadores a (o direccionamiento de) la entradas difusas a las cuales se refiera.
La porción derecha es la regla esta fingida para una o mas (en este caso uno) consecuente. Cada expresión de consecuente consiste del nombre de la salidas, seguido por "*es*" , seguido por el nombre de la etiqueta. Cada expresión de consecuente corresponde a una específica salida en RAM. Los consecuentes de una regla pueden ser codificado como una lista de apuntadores (o direccionamiento a) de las salidas difusas a las que haga referencia.
Las reglas completas están en la base de conocimientos como una lista de apuntadores o direccionamiento de entradas difusas y salidas difusas. Para que la lógica de evaluación de reglas trabaje, necesita saber cuales apuntadores se refieren a una entrada y cuales se refieren a que salida, Esto es necesario para saber cuándo se ha alcanzado la ultima regla en el sistema.

Un método de organización es tener fijo el número de reglas con un numero específico de antecedentes y consecuentes. El segundo método utilizado en los kernel libres de motorola, es marcar el fin de la lista de reglas con un valor reservado, y usar un bit en el apuntador para distinguir el antecedente del consecuente. Un tercer método de organización, es marcar en fin de la lista de reglas con un valor reservado y separar antecedentes y consecuentes con otro valor reservado. Esto permite cualquier número de reglas y permite que cada regla tenga cualquier número de antecedentes y consecuentes, sujeto al límite impuesto por la disponibilidad de memoria del sistema.

Cada regla es evaluada secuencialmente. Pero las reglas como grupo son tratadas como si todas ellas fueran evaluadas simultáneamente. Dos operaciones matemáticas se llevan a cabo durante la evaluación de reglas. El operador difuso "Y" corresponde a la operación matemática de mínimo y la operación difusa *"o"* corresponde a una operación matemática de máximo. El conector *"y"* es usado para conectar el antecedente con la regla. El operador *"o"* esta implícito entre reglas sucesivas. Antes de evaluar cualquier regla, todas las salidas difusas son puestas en cero. Para cada regla evaluada, el valor mínimo (menor) del antecedente es tomado como el verdadero de la regla. Este valor verdadero es aplicado a cada consecuente de la regla (almacenando este valor en la correspondiente salida difusa) a menos que la salida difusa sea mayor (máximo). Si dos reglas afectan a la misma salida difusa, la regla que tiene mayor grado de verdad es la salida difusa, debido a que las reglas están conectadas por la implicación *"o"*.

En el caso del peso de la regla, el valor de verdad de la regla es determinado encontrando la menor regla antecedente.

Defuzificación (WAV)

El paso final en los programas difusos combina la lista de salidas difusas en una salida difusa compuesta. Así como se usan formas trapezoidales para las entradas, el CPU12 usa singletons o singularidades como funciones de membresía de salida. El eje X representa el rango posible de valores de salida del sistema. Las funciones singlenton consisten de la posición en el eje X de la etiqueta de salida. Las salidas difusas corresponden al peso en el eje Y correspondiente a la función de membresía de salida.

La instrucción WAV calcula la suma del numerador y denominador para el calculo del peso promedio de las salidas difusas de acuerdo a la fórmula:

$$Salida\ del\ sistema = \frac{\sum_{i=1}^{n} SiFi}{\sum_{i=1}^{n} Fi}$$

Donde n es el número de etiquetas de salida del sistema de salida, Si son las posiciones de los singlentons en la base de conocimientos, y Fi son las salidas difusas en RAM. Para un programa difuso común en el CPU, n es 8 o menor (pueden estar dentro del rango 0 a

255), *Si* y *Fi* son valores de 8 bits. La división final debe hacerse por separado con la instrucción EDIV, inmediatamente después de la instrucción WAV.
Antes de ejecutar la instrucción WAV, un acumulador debe ser cargado con el número de iteraciones (*n*) un registro de índice debe apuntar a la lista de posiciones de los singletons en la base de conocimientos, y un segundo registro de índice debe apuntar a la lista de salidas difusas en RAM. Si el sistema tiene mas de una salida, la instrucción WAV debe ejecutarse para cada salida del sistema.

PROGRAMACIÓN

Escribir software en lenguaje ensamblador es similar a escribir software para procesos. En lenguaje ensamblador se tienen dos tipos de comandos para escribir el código fuente. Los códigos de operación que son instrucciones que se traducirán en lenguaje de maquina para ser ejecutados por el microcontrolador cuando se ejecute el programa. Se usan pseudo-códigos para dar las instrucciones al ensamblador.
Para ilustrar el proceso de desarrollo, se implementara un sistema de cerradura digital. El sistema tiene siete switches. Si el patrón binario de 7 bits 0100011 aparece en el puerto A (bit 6-0) durante 10ms, entonces se abrirá la chapa. Los bits 6-0 del puerto a son señales que entrarán al microcontrolador y el bit 7 del puerto a es la señal de salida.
El proceso de desarrollo de software es:

1. Listar las entradas y las salidas. Especificando el rango de valores y su significado. En este ejemplo se usa el puerto A. Los bits 6-0 serán entradas digitales. El bit 7 del puerto A será de salida. Un 1 en este bit indicara que la chapa estará abierta. En pseudocódigo se asignan los nombres simbólicos, PORTA Y DDRA a las direcciones correspondientes $0000 y $0002

```
PORTA       equ   $0000  ;   PA6-PA0   switches,   PA7
cerradura
DDRA        equ   $0002  ;   especifica   entradas   y
salidas
```

2. Se necesita hacer una lista de la estructura de datos necesarios. Estos son usados generalmente para salvar información. Si se necesita el dato sea permanente, entonces se coloca en un espacio global. Si el software cambia este valor entonces deberá ser ubicado en RAM. En este ejemplo se necesita un contador de 16 bits no signado. Se usa la directiva ORG para colocar la estructura de datos en RAM. RMB reserva bytes para esta estructura. Estas líneas asignan al nombre simbólico CNT la información que tenga la dirección $0800

```
            ORG   $F000
CNT         rmb   2       ;Contador de 16-bits
```

Si es necesario se pude definir estructuras fijas, que estarán localizadas en EEPROM. Para el ejemplo se definirán 8 bits constantes que tendrán la clave para abrir la puerta. Estas líneas estarán despues del programa, por lo que es necesario definirla en ROM o EEPROM. FCB define 8 bits constantes.

```
                ORG     $F000          ;EEPROM
LLAVE           fcb     %00100011      ;CODIGO
```

3. El siguiente paso es desarrollar el algoritmo, que tendrá la secuencia de operaciones que serán ejecutadas. En este paso se recomienda hacer algún diagrama de flujo que tenga la secuencia de pasos de que se desean programar, antes de programar en lenguaje ensamblador. Normalmente los programas se colocan en ROM o EEPROM, para el M68HC812A4 la EEPROM empieza en la dirección $F000.
   La primera instrucción que debe llevar el programa es la inicialización del stack.

```
                lds     #$0C00
```

   Posteriormente se escribe el algoritmo a implementar. Para este ejemplo:
   a) Se definen las entradas y salidas del puerto A (PA6-PA0 entradas y PA7 salida)

```
                movb    #$80,DDRA          ;PA6-PA0 ENTRADA, PA7
        SALIDA
```

   b) Se cierra la chapa

```
                bclr    PORTA,#$80                ;
```

   c) Se inicializa el contador con 4000, para hacer un loop que tenga un retardo de 10 ms.

```
                movw    #4000,CNT      ; 10,000,000ns/(125*20)
```

   d) Se implementa un loop para esperar la clave correcta

```
loop            ldaa    PORTA   ;[3]  input  from  7  switches
                anda    #$7F    ;[1]
                cmpa    LLAVE   ;[3]     es     el     código?
                bne     off     ;[3]
```

   Si es el código, entonces del contador de 16 bits de decrementa.

```
                ldx     CNT;                                    [3]
                dex             ;                               [1]
                stx     CNT     ; [3]
```

   Si el contador llega a cero entonces la puerta se abre, es decir en PA7 habrá un 1

```
                bne     loop           ;[2]=20cycles/loop
                bset    PORTA,#$80     ;se   habre   la   chapa
                bra     loop
```

   Si los switches no corresponden a la clave, entonces se cierra la chapa y CNT se reinicializa

```
off             movw    #4000,CNT         ;10,000,000ns/(125*20)
                bclr    PORTA,#$80          ;chapa        cerrada
                bra     loop
```

El programa final es el siguiente

```
; Se habre la chapa (PA7=1) si los switches coresponden a la
clave

PORTA           equ $0000     ;    PA6-PA0    switches,    PA7
cerradura
DDRA            equ $0002     ;    especifica    entradas    y
salidas

          ORG     $F000
CNT               rmb   2       ;Contador de 16-bits
          ORG     $F000         ;EEPROM
LLAVE                   fcb     %00100011       ;CODIGO

                  lds   #$0C00
          movb    #$80,DDRA     ;PA6-PA0 ENTRADA, PA7 SALIDA
          bclr    PORTA,#$80    ;
          movw    #4000,CNT     ; 10,000,000ns/(125*20)
loop              ldaa  PORTA   ;  [3]  input  from  7  switches
          anda    #$7F          ;                              [1]
          cmpa    LLAVE         ;     [3]     es     el     código?
          bne     off           ; [3]
          ldx     CNT           ;                              [3]
          dex                   ;                              [1]
          stx     CNT           ; [3]
          bne     loop          ;       [2]=20       cycles/loop
          bset    PORTA,#$80    ;    se    habre    la    chapa
          bra     loop
off       movw    #4000,CNT     ;          10,000,000ns/(125*20)
          bclr    PORTA,#$80    ; chapa cerrada
          bra     loop
LLAVE                   fcb     %00100011       ; código
```

4. El siguiente paso es ensamblar el programa.

El ensamblador generara código ejecutable por el microcontrolador. En este caso genera un archivo con formato S19. Este archivo

INTRODUCCIÓN AL CONTROL DIFUSO

En la década de los 60's después de que apareció el *paper* originario del Dr. Zadeh (1965), muchos desarrollos teóricos en lógica difusa han aparecido en Estados Unidos, Europa y Japón. Desde mediados de los 70's a la fecha, los investigadores japoneses han sido los que mas han contribuido en el avance de la implementación práctica de la teoría, han tenido un excelente trabajo de comercialización de esta tecnología y ahora tienen más de 2000

patentes en el área. Sin embargo, muchos de los resultados asociados con la tecnología difusa son debidos a la lógica difusa, y algunos debidos a los avances en los sensores usados en estos productos.

La teoría de los conjuntos difusos dan una forma de representar incertidumbre. aunque históricamente, la teoría de probabilidad ha sido la principal herramienta para representar la incertidumbre en modelos matemáticos, debido a esto, toda la incertidumbre era asumida como una caracterización de incertidumbre aleatoria. La teoría de conjuntos difusos es una gran herramienta para el modelado de un tipo de incertidumbre asociada con vaguedad, con imprecisión y con la carencia de información con respecto a un elemento particular de problema tratado. Otra potencialidad de la teoría de conjuntos difusos es la posibilidad de utilizar variables lingüísticas en lugar de variables cuantitativas para representar conceptos imprecisos.

La incorporación de la teoría de los conjuntos difusos y la lógica difusa en los modelos computacionales, ha demostrado una gran utilidad en áreas en donde la intuición y el criterio todavía juegan un rol importante en el modelo. Aportaciones a control, como es el control de temperatura, de trafico o el control de procesos, es en donde se han tenido las últimas aplicaciones de lógica difusa.

La lógica difusa es mas usada en dos tipos de situaciones. (i) Modelos muy complejos en donde en entendimiento es muy limitado, de hecho es intuitivo. y (ii) procesos en donde se involure el razonamientos humano, la percepción y la toma de decisiones humanas

El controlador "perse" es una abstracción matemática y como tal no puede incorporar un conocimiento completo. Todas sus estructuras clásicas coinciden en dos puntos básicos que se deben saber para su funcionamiento:

- El sistema que va a ser controlado se tiene que conocer, para predecir una respuesta a una entrada conocida (implica conocimiento).
- La meta de control ha de expresarse en términos concisos (fórmulas matemáticas que involucran índices de comportamiento)

Desafortunadamente la elegante herramienta matemática aumenta en complejidad al especificar más un modelo. No puede ser construido con toda precisión, debido a las no linealidades, el carácter no estacionario del sistema y el conjunto de datos representativos, por tanto se recurre a métodos estadísticos y algoritmos o funciones multivariables de optimización. En esos casos se tiende a estimar determinados parámetros que se desconocen para minimizar las tareas de control

Lo anterior inclina a una nueva lógica Heurística que permite el diseño de simples reguladores que se comporten igual o mejor que las sofisticadas leyes de los controladores multivariables.

Lo heurístico juega un papel importante para escoger criterios apropiados. En este caso los CONJUNTOS DIFUSOS vislumbran un gran panorama dentro del Control Experto, caracterizados por expresar la vaguedad de los juicios humanos, los cuales son expresados de una manera verbal más que mediante modelos modelos matemáticos.

LÓGICA DIFUSA.

Aleatoriedad VS Ambigüedad.

Lo *"Difuso"* es una alternativa para describir con precisión lo vago o no plenamente definido, difiere de lo aleatorio conceptual y teóricamente, aunque no por esto dejan de existir similitudes tales como operar en un intervalo unitario (0 1) o asociar proposiciones y conjuntos. La distinción radica en como tratan a un conjunto y su complemento en el conjunto mismo ya que en los conjuntos difusos hay grados de membresía en tanto que en los conjuntos de Cantor solo pertenecen o no. Fundamentalmente la opción difusa extiende a la probabilidad el ignorar el principio de la *"No Contradicción"* y el principio de *"Exclusión"*.

Podemos concluir que *"Difuso"* describe un concepto o conjunto ambiguo, mide en que grado es o pertenece un elemento a un conjunto, en cambio lo aleatorio describe la incertidumbre de que un evento se dé.

Suponga que esta en el desierto y encuentra 2 vasos con líquido. El líquido del primero vaso es descrito como si tuviera el 95% de posibilidad que sea salubre y buena. El líquido del segundo vaso es descrito como si tuviera el .95 de pertenencia a la clase de líquidos "saludables y buenos". ¿Qué vaso escogería?

Cual es la filosofía de distinción entre estas dos formas de información? En el primer caso se tiene la probabilidad de que el .95 de 1.0 o 0 de que el liquido sea o no potable. El valor de pertenencia del .95 indica la potabilidad del liquido "saludable y bueno" es del 0.95 después de la medición y prueba.

Lo difuso describe lo ambiguo de un evento, mientras que lo aleatorio describe la incertidumbre en la ocurrencia del evento.

Conjuntos difusos

En la teoría clásica de conjuntos, la transición entre membresía y no membresía (pertenencia o no pertenencia) de un elemento del Universo del Discurso es abrupta y bien definida. En el caso difuso esa transición es gradual, debido a que sus fronteras son vagas o ambiguas.

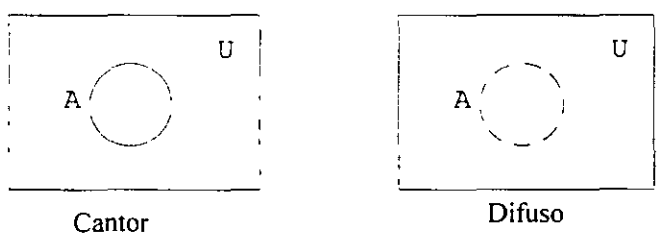


Fig. 4
Representación en diagramas de Venn
de un Conjunto Difuso y Conjunto de Cantor

Un conjunto difuso, entonces, es un conjunto que contiene los elementos que tiene varios grados de pertenencia en el conjunto, Esta idea es contrastante con los conjuntos clásicos debido a que la membresia en los conjuntos clásicos puede ser de completa pertenencia y

de completa no pertenencia. Y los elementos de los conjuntos difusos pueden no tener membresía completa, y pueden ser además miembros de otro conjunto difuso en el mismo universo.
Elementos de un conjunto difuso pueden ser mapeados en el universo de valores de membresia usando una función teórica. La función mapea los elementos de un conjunto difuso A en valores numéricos reales en el intervalo [0,1]. Una convención de notación para conjuntos difusos cuando el universo de discurso, X, es discreto y finita, es como se muestra a continuación para un conjunto difuso A

$$A = \left\{ \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + ... \right\} = \left\{ \sum_i \frac{\mu_A(x_i)}{x_i} \right\}$$

Cuando el universo, X, es continuo e infinito, un conjunto difuso A está dado por:

$$A = \left\{ \int \frac{\mu_A(x}{x} \right\}$$

Operaciones con conjuntos difusos.

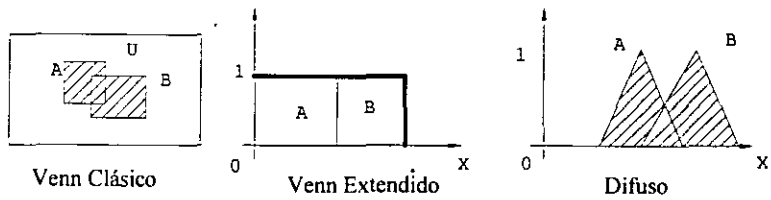
Definiendo tres conjuntos difusos A, B, C, en el universo X. Para un elemento, x, dado del universo, se tienen las siguientes operaciones :

nion	$\mu_{A\cup B}(x) = \mu_A(x) \vee \mu_B(x)$
Inter sec ción	$\mu_{A\cap B}(x) = \mu_A(x) \wedge \mu_B(x)$
Complemento	$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$

Cualquier conjunto difuso A definido en un universo X es un subconjunto del ese universo. También por definición, solo con conjuntos clásicos, el valor de membresia de cualquier elemento x en el conjunto nulo $\varnothing$ es cero, y el valor de membresía de un elemento x en el conjunto completo X es 1.
La colección de todos los conjuntos difusos y los subconjuntos de esta denotados como la potencia del conjunto difuso P(X). Como es obvio, basados en el hecho de que todos los conjuntos difusos se pueden traslapar, la cardinalidad $n_{p(x)}$, de un conjunto de potencia es infinito $n_{p(x)} = \infty$

Los conjuntos difusos se combinan en forma parecida a los que no lo son, a través de operaciones de máximos y mínimos como se puede observar en las siguientes comparaciones, donde se muestran los diagramas de Venn normales y extendidos para los casos de conjuntos de Cantor y difusos.

Unión



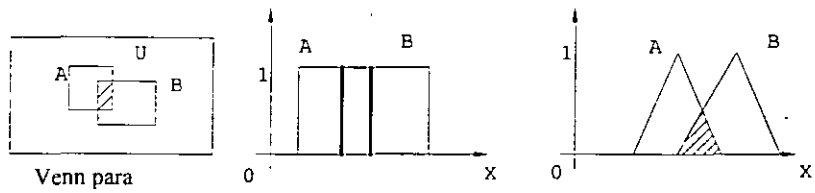
Venn Clásico Venn Extendido Difuso

$$A \cup B = \{ x | x \in A \; o \; x \in B \} \quad \textit{Tradicional}$$

$$A \cup B = MAX(A, B) \quad \textit{Difuso}$$

Fig. 5
Unión

Intersección



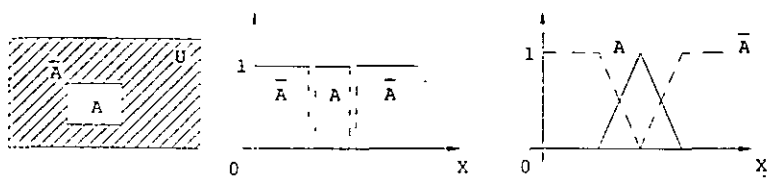
Venn para Cantor | Venn extendido tradicional | Venn extendido difuso

$$A \cap B = \{ x | x \in A \; y \; x \in B \} \quad \textit{Tradicional}$$

$$A \cap B = MIN(A, B) \quad \textit{Difuso}$$

Fig. 6
Intersección

Complemento



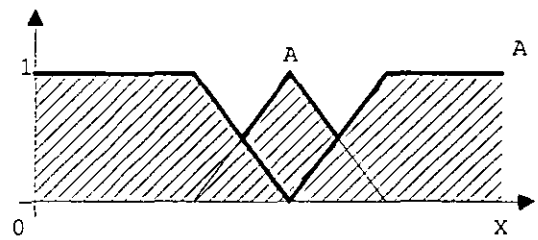
Venn para cantor | Venn extendido tradicional | Venn extendido difuso

$$\overline{A} = 1 - A \quad \textit{difuso}$$

$$\overline{A} = \{ x | x \in A, x \in U \} \quad \textit{tradicional}$$

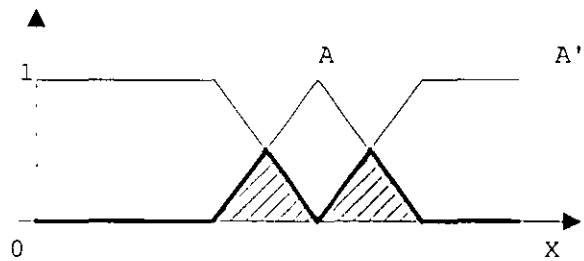
Fig. 7
Complemento

Como se menciono anteriormente, existen 2 operaciones que son diferentes entre conjuntos clásicos y conjuntos difusos. Estas son las dos leyes del medio excluido (Ley del medio excluido y la Ley o Principio de Contradicción)



$$X = A \cup \overline{A}$$

Fig. 8
Principio del Medio Excluido
En el principio de exclusión se observa que una función no puede tener todo el valor de verdad, es decir no todos sus valores son uno. A esta situación se le conoce como bajo traslape
A∪A'≠X bajo traslape (no todos los valores de la función A∪A son unos)



$$A \cap \overline{A} \neq 0$$

Fig. 9
Principio de Contradicción
Se observa como puede existir valores medios diferente de cero.
Es conocido también como sobre traslape.
A∩A'≠0 sobretraslape (no todos los valores de la función A∩A son ceros)

Funciones de Membresía

Las funciones de membresía caracterizan el aspecto borroso(difuso) de los conjuntos difusos, en una forma gráfica para el uso conceptual del formalismo matemático de la teoría de conjuntos difusos.

Debido a que las funciones de membresía comprenden la esencia de todo el aspecto difuso de un conjunto difuso particular, esta descripción es la esencia de la propiedad difusa u operación. Dada la importancia de la gráfica de la función de membresía es necesario que se le dedique una especial atención a su construcción.

Aspectos de la función de membresía.

Debido a que toda la información esta contenida en un conjunto difuso descrito por sus funciones de membresía, es necesario desarrollar terminología para describir varios aspectos de estas funciones.

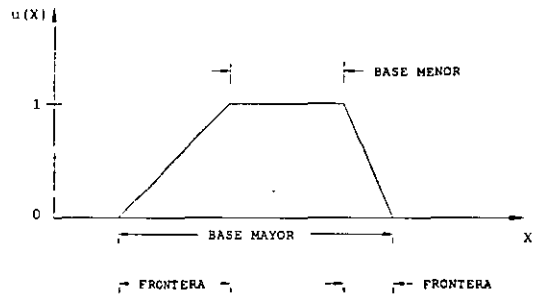


Fig. 10
Función de Membresia

La base menor (parte central) de la función de membresía de un conjunto difuso A es definida como la región del universo que esta caracterizado completamente y tiene una completa membresía dentro del conjunto A. La base menor comprende los elementos X del universo tal que $\mu_A(x) = 1$

La base mayor (Soporte) de la función de membresía esta definido como la región del universo que esta caracterizado por un membresía diferente de cero en el conjunto A. Esto es que la base mayor abarca los elementos X del universo tales que $\mu_A(x) > 0$.

Los limites (fronteras) de la función de membresía están definidos como la región del universo que contiene a los elementos que tienen membresía diferente de cero pero no tienen completa membresía. Estos bordes comprenden los elementos x del universo tal que $0 < \mu_A(x) < 1$. Estos elementos del universo con los que tienen un grado de vaguedad. o membresía parcial en el conjunto difuso.

Un conjunto difuso *normal* es aquel cuya función de membresía tiene al menos un elemento x en el universo cuyo valor de membresía es unitario.

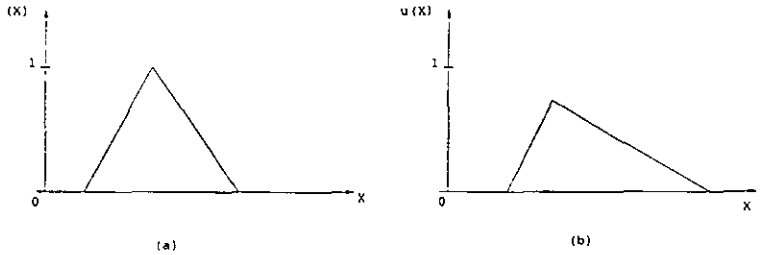


Fig. 11

Conjuntos difusos (a) normal y (b) subnormal

Un conjunto difuso convexo es descrito por una función de membresía cuyos valores de membresía están incrementándose monoticamente, o cuyos valores de membresía están decrementandose monoticamente, o cuyos valores de membresía están incrementandose monoticamente y luego decrementandose monoticamente con valores crecientes de los elementos del universo. Para cualquier elemento *x*, *y* y *z* en un conjunto difuso la relación $x<y<z$ implica que

$$\mu_{\underline{A}}(y) \geq min\left[\mu_{\underline{A}}(x)\mu_{\underline{A}}(z)\right]$$

Entonces se puede decir que $\underline{A}$ es un conjunto convexo

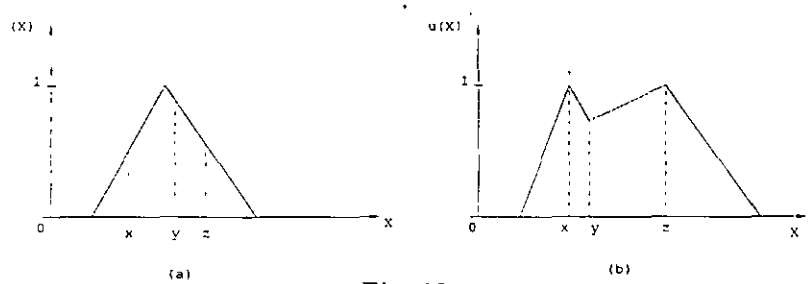


Fig. 12

Conjunto difuso (a) normal y convexo (b) normal y no convexo

Una propiedad importante de dos conjuntos convexos, es que dado A y B, la intersección de dos conjuntos convexos da como resultado un conjunto convexo. Es decir, para A *and* B, con los dos convexos $A \cap B$ es también convexa.

El *peso* de un conjunto difuso A es el valor máximo de la función de membresía, $\max\{\mu_{\underline{A}}(x)\}$. Si el peso de un conjunto difuso es menor que la unidad, entonces se dice que el conjunto difuso es subnormal.

Si $\underline{A}$ es convexo, de un solo punto y normal, el conjunto difuso esta definido en una linea rea, entonces A es llamado *numero difuso*.

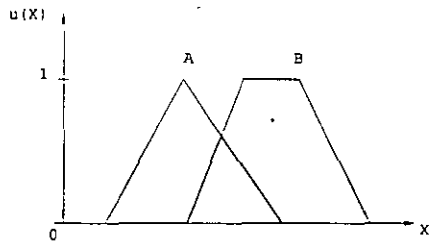


Fig. 13
La intersección de dos conjuntos difusos convexos produce un conjunto convexo

Formas estándar y fronteras

Una de las formas más comunes de las funcionas de membresía son aquellas que son normales y convexas, sin embargo muchas operaciones con conjuntos difusos, es decir operaciones con funciones de membresía, resultan en un conjunto difuso subnormal y convexo (Principio de extensión, y operación unión).
Las funciones de membresía pueden ser simétricos o asimétricos. Son típicamente definidos en un universo, pero se pueden describir en universo multidimensional (o n-dimensonal). Estas hipersuperficies o curvas, son mapeadas por combinación de sus parámetros en el espacion n-dimensional de los valores de membresia en el intervalo [0,1]. Estos valores de membresia expresan el grado de membresia que una combinación especifica de parámetros en el espacio n-dimensioal tiene un conjunto difuso particular definido en el. universo u-dimensional de discurso.

Fuzificación.

El proceso de fuzificación es una conversión, el hacer de una cantidad claramente definida (crisp) un valor o cantidad difusa. Esto se puede hacer por el simple reconocimiento de muchas de las cantidades que puede ser consideradas claramente definidas y deterministas volverlas no deterministas completamente, para que puedan ser consideradas inciertas.
Si la forma de incertidumbre aparece debido a imprecisiones, ambigüedad o vaguedad, entonces es probablemente difusa y puede ser representada por una función de membresía.
En el mundo real, el hardware de un volmetro digital genera salidas claramente definidas pero estos datos están sujetos a errores experimentales. La información que se ve en la figura 5 muestra un posible rango de error para valores típicos leídos y la función de membresia asociada puede ser tal que represente su imprecisión.

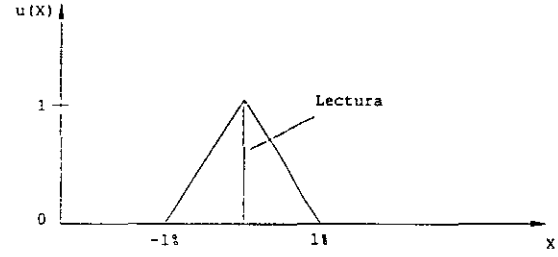


Fig. 14
Función de membresía que representa la imprecisión en
la lectura de un voltaje bien definido

La representación de datos imprecisos como un conjunto difuso es usada pero no es un paso obligatorio cuando estos datos son usados en sistemas difusos. Esta idea es mostrada en la fig. 14 donde se considera que un dato bien definido es elegido (fig. 15.a), o leído como un dato difuso (fig. 15.b).

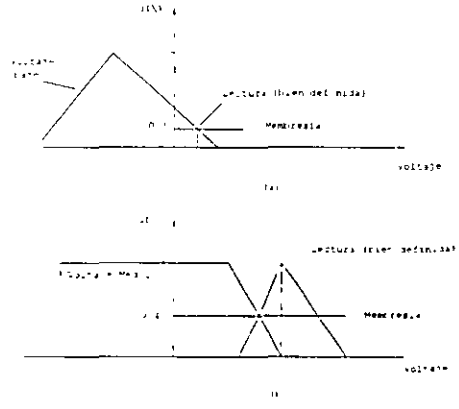


Fig. 16
Comparación de lecturas de conjuntos difusos y claramente definidos
(a) conjunto difuso y lectura claramente definida
(b) conjunto difuso y conjunto difuso leído

En la figura 15.a podemos comparar el valor del voltaje claramente definido con un conjunto difuso, decir "voltaje bajo". En la fig. podemos ver que la entrada definida se intersecta con el conjunto difuso "voltaje Bajo" con un valor de membresia de 0.3. El valor leído y el conjunto difuso puede decirse que están de acuerdo en el valor de 0.3.

En la fig. 15.b la intersección la intersección de conjunto difuso "medio voltaje" y el valor fuzzificado (leído) ocurre en un valor de membresía de 0.4. Podemos ver que la intersección de dos conjuntos difusos es el triángulo pequeño, donde la mayor membresía ocurre en el valor de 0.4.

Asignación de los valores de membresía

Hay más de un camino para asignar valores de membresía o funciones a variables difusas, como asignar funciones de densidad de probabilidad a variables aleatorias. El proceso de asignación pude ser intuitivo o puede ser basado en algún algoritmo o operaciones lógicas. Se listan algunos de los métodos descritos en la literatura para la asignación de valores de membresía o funciones de variables difusas.

Intuición
Inferencia
Conjuntos difusos angulares
Redes neuronales
Algoritmos genéticos
Razonamientos inductivo

Conversiones difusas-a-claramente definidas.(Defuzificación)

Existe un problema en la aplicación de resultados en forma difusa, en el mundo que nos rodea. De hecho la información la asimilamos todos los dias en forma difusa, sin embargo, la implementación de resultados o desiciones humanas o de máquinas esta en forma de binaria o claramente definida. Las máquinas no entienden las instrucciones dadas en lenguaje natural del humano, por lo que se necesita "defuzificar" el resultado difuso. Necesitamos encontrar la conversión de resultados difusos en salidas claramente definidas. Matemáticamente la defuzificación de un conjunto difuso es un proceso de redondeo de salida, de la localización en un hipercubo unitario del vértice mas cercano.
Hay varias formas para convertir conjuntos difusos a una valor escalar.

Corte lambda de conjuntos difusos.

Primero se considera un conjunto difuso A, después de define el conjunto de corte lambda A_λ, donde $0 \le \lambda \le 1$. El conjunto A_λ es un conjunto claramente definido llamado corte lambda del conjunto difuso $\underline{A}$ donde $A_\lambda = \{x|\mu_A(x) \ge \lambda\}$. Vea que el conjunto de los cortes lambda , es un conjunto definido derivado de los conjuntos difusos. En conjunto difuso va a ser transformado en un numero finito de conjuntos de números de cortes lambda debido a que hay un infinito numero de valores de λ de en el intervalo [0,1].
Cualquier elemento $x \in A_\lambda$ pertenece a $\underline{A}$ con un grado de membresía la que es mayor o igual al valor de λ.
Los cortes landa deber cumplir con las siguientes propiedades

$$(\underline{A} \cup \underline{B})_\lambda = A_\lambda \cup B_\lambda$$

$$(\underline{A} \cap \underline{B})_\lambda = A_\lambda \cap B_\lambda$$

$$\overline{(A)}_\lambda \neq (\overline{A}_{\lambda\lambda}) \quad \text{excepto para el valor de } \lambda = 0.5$$

Para cualquier $A_\lambda \leq \alpha$, donde $0 \leq \alpha \leq 1$, es verdad para $A\alpha \subseteq A_\lambda$ donde A_0=X.
Estas propiedades que se muestran como operaciones standard en conjuntos difusos es equivalente con las operaciones de los conjuntos λ.

Métodos de *defuzificacion.*

Defuzificación es la conversión de una cantidad difusa a una cantidad precisa, y la *fuzificación* es la conversión de las cantidades precisas a cantidades difusas. La salida de un proceso puede ser la unión lógica o dos o mas funciones de membresía definidas en el universo de discurso de la variable de salida.
En general el proceso de salida pude involucrar muchas partes de salida, y las funciones que representan capa parte de la salida pude ser dibujada como triángulo o trapecio. En la figura 5.4 las funciones de membresía no son siempre normales. En general se tiene

$$\underline{C}_k = \bigcup_{i=1}^{k} \underline{C}_i = \underline{C}$$

Se muestran a continuación algunos métodos de defuzificación:

1. Principio de Máxima Membresía. Se conoce también como método de pesos, en este esquema esta limitado a funciones puntiaguda de salida. Este método esta dado por la expresión algebraica

$$\mu_{\underline{C}}(z^*) \geq \mu_{\underline{C}}(z) \quad para \quad toda \quad z \in Z$$

gráficamente:

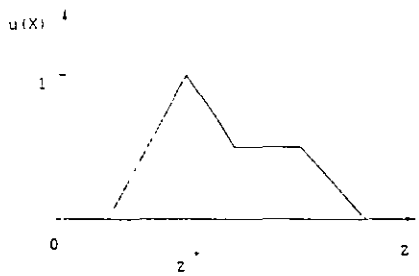


Fig. 17
Método de máxima membresía

2, Método del centroide. Este procedimiento (llamado centro de área, centro de gravedad) es de los que mas han prevalecido y físicamente atrayente método de defuzificación, y esta dado por la expresión

$$z^* = \frac{\int \mu_{\underline{C}}(z) \cdot z \, dz}{\int \mu_{\underline{C}}(z) \, dz} \tag{30}$$

donde $\int$ indica una integración algebraica

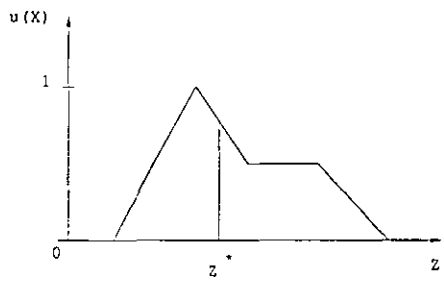


Fig. 18
Método del centroide

3. Método del peso promedio. Este método solo es valida para funciones de salidas simétricas y esta dado por la expresión

$$z^* = \frac{\sum \mu_{\underline{C}}(\bar{z}) \cdot z}{\sum \mu_{\underline{C}}(\bar{z}) z} \tag{31}$$

donde $\sum$ indica la suma algebraica. fig. El método del peso promedio esta formado por los pesos de cada función de membresía de salida por sus respectivos valores máximos de la función.

Fig. 19
Método del peso promedio

4. Max-Min Este método es cercano al primer método, excepto que la localización del valor máximo de membresía pude ser no único. (la máxima membresía puede ser estabilizarse, nivelarse preferentemente que un solo punto). Esta dado por

$$z^* = \frac{a+b}{2}$$

donde a y b están definidos en la figura 10

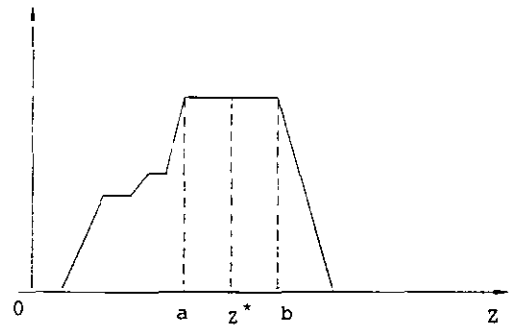


fig. 20

5. Centro de sumas: Este es el método mas rápido de defusificación, que esta actualmente en uso. Este proceso involucra la suma algebraica de los conjuntos de salida, C1 y C2 en lugar de su unión.
El valor defusificado z* esta dado por la siguiente ecuación:

$$z^* = \frac{\int z \sum_{k=1}^{n} \mu c\ (z) dz}{\int \sum_{k=1}^{n} \mu c_k (z) dz}$$

Este método es similar al método de peso promedio, excepto que en el método de cetro de sumas los pesos son las áreas de las respectivas funciones de membresia, mientras que el método de pesos promedio los pesos son valores de membresia.

6. Centro de mayor área. Si la entrada del conjunto difuso esta dentro de dos subregiones convexas, entonces el centro de gravedad de la subregión convexa con mayor área, se usa para obtener le valor defuzificado z* de salida. Algebraicamente:

$$z^* = \frac{\int \mu c_m (z) z dz}{\int \mu c_m (z) dz}$$

donde C_m es la subregión convexa, que tiene mayor área .

7. Primero de máxima. Este método usa el traslape de las salidas o la unión de todos los conjuntos difusos de salida C_k para determinar el menor valor con máximo valor de membresía .

Sistemas basados en reglas

En el campo de la inteligencia artificial, (maquinas inteligentes) hay varios caminos para representar el conocimiento. Tal vez es mas común el camino de representar el conocimientos humanos en forma de expresiones naturales del lenguaje del tipo.

IF premisa (antecedente), THEN conclusión (consecuente)
(Si premisa (antecedente), ENTONCES conclusión (consecuente)).

Esta forma es comúnmente conocida como reglas IF-THEN. Es una expresión típica de inferencia tal que si se conoce el hecho (premisa, hipótesis, antecedente) se puede inferir o derivar otro hecho llamado conclusión (consecuente). Esta forma de representación del conocimiento, es muy apropiada en el contexto lingüístico porque expresa el conocimiento empírico y heurístico en nuestro lenguaje de comunicación.
Este tipo de reglas permiten a los algoritmos describir cual acción es la que deba tomarse basados en observaciones actuales. (en la teoría difusa, nada es hecho aleatoriamente, la información que contiene cierto grado de vaguedad es expresada lo mas fiel posible, sin la distorsión producida por una salida bien definida, y es entonces procesada de una manera apropiada).
Para describir el conocimiento experto se tiene un conjunto de reglas difusas. En general hay dos o mas antecedentes y conclusiones. Sin embargo considerando el caso de un antecedente y una conclusión se escribe de la siguiente manera.

$\{\text{IF es } A_i \text{ , THEN B es } B_i\}_{i=1}^{N}$

donde N es el número total de reglas, En general se tiene dos o mas antecedentes y conclusiones.

Los sistemas basados en reglas pueden componerse de un conjunto de reglas condicionales

Forma canónica de los sistemas basados en reglas:

Regla 1: IF condición C^1;THEN restricción R^1
Regla 2: IF condición C^2;THEN restricción R^2

Regla r: IF condición C^r;THEN restricción R^r

En general, oraciones condicionales e incondicionales ponen determinadas restricciones en el consecuente el proceso de evaluación debido a condiciones inmediatas o anteriores. Estas restricciones en general se manifiestan en forma de vaguedad en el lenguaje natural .

Descomposición de reglas

Expresiones lingüísticas expresados por el humano pueden involucrar una estructura de reglas , y usando las propiedades y operadores definidos para conjuntos difusos cualquier estructura de reglas puede ser descompuesto y reducido a un numero de reglas canónicas

simples, estas reglas están basadas la representación del lenguaje natural y sus modelos que a su ves están basados en conjuntos difusos .

Múltiples antecedentes conjuntivos

IF x es A^1 and A^2.....and A^L THEN y es B^S

Asumiendo un nuevo conjunto A^S como:
$A^S = A^1 \cap A^2 \cap \ldots \cap A^L$

Expresada en términos de funciones de membresia

$$\mu_{A^S}(x) = \min\left[\mu_{A^1}(x), \mu_{A^2}(x), \ldots \mu_{A^L}(x)\right]$$

Basados en la definición de la operación de intersección, la regla puede ser descrita como:
IF A^S THEN B^S

Multiples antecedentes disjuntivos

IF x es A^1 or A^2.....or A^L THEN y es B^S

Puede ser reescrito como:

IF x es A^S THEN y es B^S

Donde el conjunto difuso A^S esa definido como:

$A^S = A^1 \cup A \cup^2 \ldots \cup A^L$

$$\mu_{A^S}(x) = \max\left[\mu_{A^1}(x), \mu_{A^2}(x), \ldots \mu_{A^L}(x)\right]$$

Que esta basado en la operación unión difusa.

Condiciones condicionales con ELSE y UNLESS

a) IF A^1 THEN (B^1 ELSE B^2)

Puede ser descompuesta en dos formas canónicas conectadas por OR

IF A^1 THEN B^1

OR

IF NOT A^1 THEN B^2

b) IF A^1 (THEN B^1) UNLESS A^2

Puede ser descompuesta en:

IF A^1 THEN B^1

OR

IF A^2 THEN NOT B^1

c) IF A^1 (THEN B^1 ELSE IF A^2 THEN (B^2))

Puede se puesta como:
IF A^1 THEN B^1

OR

IF NOT A^1 AND A^2 THEN B^2

Reglas anidadas IF-THEN

IF A^1 THEN (IF A^2 THEN (B^1))

Que puede ser puesto como:

IF A^1 AND A^2 THEN B^1

Cuando las reglas son descompuestas en una serie de formas canónicas, cada una de estas formases una implicación y podemos reducir las reglas a una serie de relaciones.

Posibilidad y cuantificación de verdad

Los términos primarios y composición usan limites lingüísticos que puedes ser seguidos por variables lingüísticas denotando similidad, tales como "probable", "muy probable", "altamente probable", "no probable", pueden ser modificados semánticamente por oraciones que verdad, tales como "verdadero", "bastante verdadero", "poco verdadero", "falso", "mas o menos falso" y "muy falso". Estas etiquetas de probabilidad están basadas en nociones de probabilidad. Los términos primarios, tales como las reglas, pueden ser muy restrictivos

Los términos primarios y composición usan limites lingüísticos y que pueden ser dados por variables lingüísticas que denotan probabilidad. Los términos primarios, así como las reglas, pueden ser restringidas por variables lingüísticas asociadas con certeza, tales como " indefinido", "desconocido" y "definitivo".

Los términos primarios "si", "a la mejor", y "no" pueden asignárseles significados en las funciones de membresia dados por "muy, muy probable", "probable" y "muy, muy improbable". Observe que el termino "cualquier" es equivalente al universo de discurso y esta dado por

$$\mu_{cualquiera}(x) = 1 \qquad \text{para todo } x \in X \tag{35}$$

Suponiendo que se este interesado en la cuantificación del valor de verdad para un antecedente o consecuente en una regla. Dado τ el valor de verdad difuso, por ejemplo, "muy verdadero", "verdadero" y "poco verdadero", "poco falso", "falso", etc. El valor de verdad pude ser considerado un elemento difuso en el intervalo unitario caracterizado por su propia función de membresia. La cuantificación de la proposición de verdad puede ser expresado como "x es A es τ". La transformación de la proposición es:

$$x \text{ es } A \text{ es } \tau = \mu_A(x_\tau) \qquad (36)$$

La ecuación anterior tiene el efecto de reducir a valores de membresia del antecedente calificado como valor de verdad τ.

Agregación de reglas difusas.

La mayoría de los sistemas basados en reglas involucran mas de una regla. El proceso de obtener consecuentes (conclusiones) de los consecuentes individuales por cada regla contribuye a la base de reglas y se conoce como agregación de reglas . Para la estrategia de agregación, existen dos casos extremos:
a) Sistemas de reglas conjuntivas. En el caso de sistemas en las cuales se deba cumplir la conjunción, las reglas están conectadas por el conectivo "and". En este caso la agregación de salida (consecuente), y , se encuentra por la intersección de la contribución de todas las reglas consecuentes, y^i, donde i = 1,2,..,r, como

$$y = y^1 \ and \ y^2 \ and.....and \ y^r$$

o

$$y = y^1 \cap y^2 \cap ..... \cap y^r$$

que esta definido por la función de membresia

$$\mu_y(y) = \min(\mu_{y^1}(y), \mu_{y^2}(y),...., \mu_{y^r}(y)) \quad para \ y \in Y$$

b) Sistemas de reglas disjuntivas. En el caso de sistemas de reglas en donde se deba satisfacer una de las reglas, las reglas se interconectan por conectivas "or". En este caso la agregación de salidas se encuentra por la unión de las contribuciones de cada regla.

$$y = y^1 \ or \ y^2 \ or.....or \ y^r$$

O

$$y = y^1 \cup y^2 \cup ..... \cup y^r$$

Que esta definido por sus funciones de membresia como:

$$\mu_v(y) = \max(\mu_{y^1}(y), \mu_{y^2}(y), ..., \mu_{y^r}(y)) \quad para \ y \in Y$$

Técnicas de inferencia

Los procedimientos gráficos se pueden extender fácilmente y pueden ser usados para sistemas difusos basados en reglas . con cualquier numero de antecedentes(entradas) y consecuentes(salidas).
Un sistema difuso con 2 entradas x_1 y x_2 (antecedentes) y una sola salida y (consecuente) se describe por una colección de r proposiciones lingüísticas IF-THEN

$$IF \ x_1 \ es \ A_1^k \ y \ x_2 \ es \ A_2^k \ entonces \ y^k \ es \ B^k \quad para \ k = 1,2,....r$$

Donde A_1^k y A_2^k son conjuntos difusos que representan los pares del k-esimo antecedente, y B^k representa a los conjuntos difusos del k-esimo conscuente.

Por ejemplo:

Para las entradas x_1 y x_2 claramente definidas, con funciones delta. El sistema basado en reglas esta descrito por la ecuación anterior, entonces, la membresia para las entradas x_1 y x_2 se puede describir como

$$\mu(x_1) = \delta(x_1 - entrada(i)) = \begin{cases} 1, & x_1 = entrada(i) \\ 0, & en \ cualquier \ otro \ caso \end{cases}$$

Y

$$\mu(x_2) = \delta(x_2 - entrada(j)) = \begin{cases} 1, & x_2 = entrada(j) \\ 0, & en \ cualquier \ otro \ caso \end{cases}$$

Basados en el método de inferencia de implicación de Mandami (max-min) y para el conjunto de reglas disjuntivas, la agregación de reglas de salida r esta dado por

$$\mu_{B'}(y) = \max_{k}[\min[(\mu_{A_1^k}(entrada(i)), \mu_{A_2^k}(entrada(j))]] \quad k = 1,2,...r$$

Que tiene la siguiente interpretación gráfica. Se ilustra el ejemplo para dos reglas, los símbolo A_{11} y A_{12} se refieren al primer y segundo antecedente de la primera regla respectivamente, y el símbolo B_1 se refiere al consecuente difuso de la primera regla. Los símbolos A_{21} y A_{22} se refieren al primer y segundo antecedente de la segunda regla respectivamente, y el símbolo B_2 se refiere al consecuente difuso de la segunda regla.
La función mínima de la ecuación anterior es el valor mínimo de entre los dos valores de membresía de entrada ya que se conectan por la conectiva "and" (operación min). Estos valores se propagan a través del consecuente y truncan las funciones de membresia del consecuente en cada regla. Esta inferencia gráfica se hace para cada regla. Entonces las funciones de membresia truncadas forman una gráfica equivalente para la operación conjunción (max) que resulta en la agregación de las funciones de cada una de las formas de membresia de cada regla. Si se desea encontrar el valor de salida claramente definida se usa el método de defuzificación para evaluar y*.

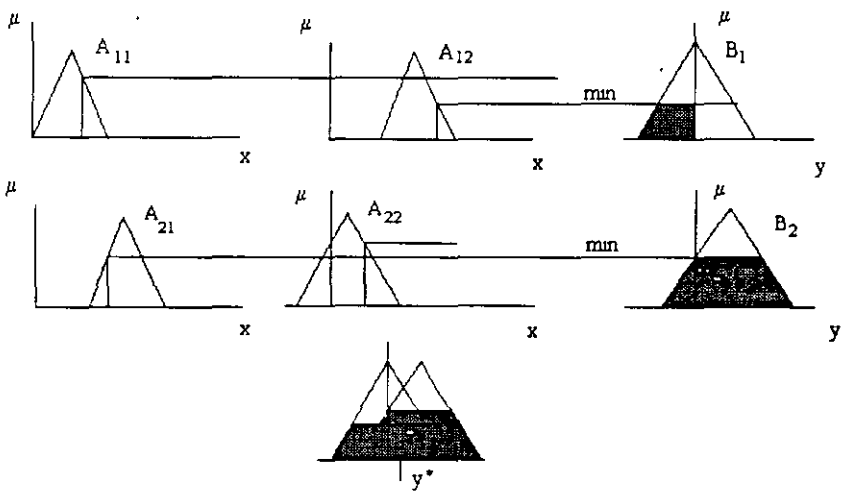


Fig. 21

Los Sistemas Difusos son modelos de *"Estimadores Libres"* que no requieren de una descripción Matemática ó Protocolo de Control, para saber como una salida depende de la entrada, representan ambigüedad, controlan sistemas con descripciones parciales o muestras de comportamiento .Es una forma de inteligencia que generaliza acciones cuando se ha excedido su experiencia, siendo capaz de describir procesos no lineales numéricamente en términos lingüísticos comunes. Se aproxima a ello con memorias de Asociación Difusa FAM que conlleva a un razonamiento de conjuntos de naturaleza Multidimensional y Matricial mucho más complicado que uno con proposiciones

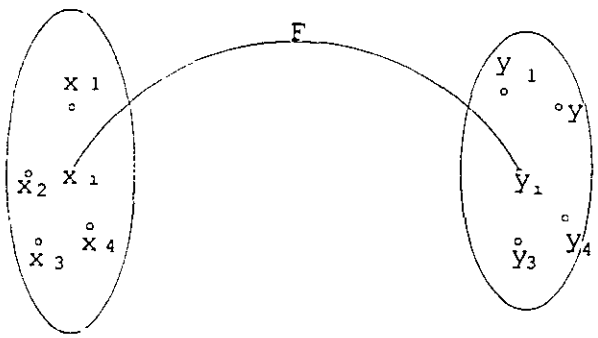


Fig. 22
Razonamiento con proposiciones.

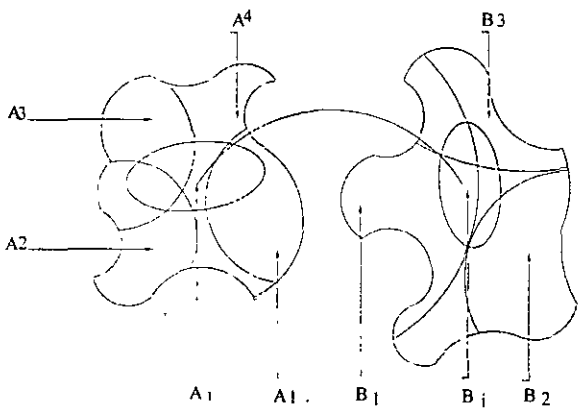


Fig. 23
Razonamiento con proposiciones difusas.

Los Controladores Difusos establecen transformaciones entre Hipercubos F: $I^n \rightarrow I^p$

Donde:

- I^n contiene todos los Subconjuntos Difusos.

$X^{-}\{X_1 \ldots X_n\}$ alberga el universo del discurso de entrada.

I^p alberga el universo del discurso de entrada del rango del espacio $y = \{y_1 \ldots\ldots y_n\}$ universo del discurso de salida.

Lo anterior se aprovecha para codificar memorias de Asociación Difusa FAM por medio de las reglas más simples (si A es *"antecedente 1"* entonces B es *"consecuente"*). (A_i, B_i) asociadas a los hipercubos.

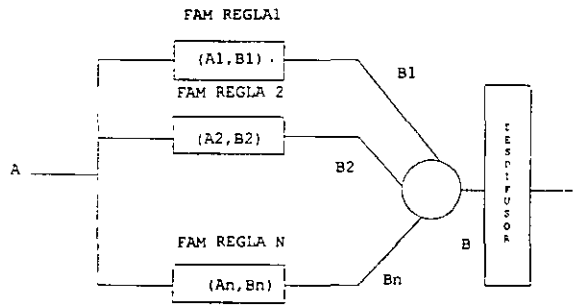


Fig. 24
Arquitectura básica de un Controlador Difuso

El mapeo es en la siguiente forma F(A)=B. F define la transformación del sistema
$F:I^n \rightarrow I^p$. La desdifusión produce la salida claramente definida. Si la entrada fue claramente definida *(crisp)* también la transformación se reduce al mapeo entre cubos booleanos $F:\{0,1\}^n \rightarrow \{0,1\}^p$. Tales sistemas se conocen como BIOFAM (binary input output). Por la razón antes expuesta su banco lingüístico es más pequeño, procesando más fáciles pares de números redundando en una mayor fluidez en el control. Este tipo de sistemas transduce el comportamiento de datos a comportamiento de reglas por lo que son muy populares en las aplicaciones.

Dos variables se relacionan dentro de los controladores difusos. el error "E" y el cambio en el error "CE". Se codifican lingüísticamente en conjuntos difusos de referencia formando el esquema básico de conocimiento. Una estructura detallada también contiene factores de escala, además de la interfaz de salida y un mecanismo de inferencia.

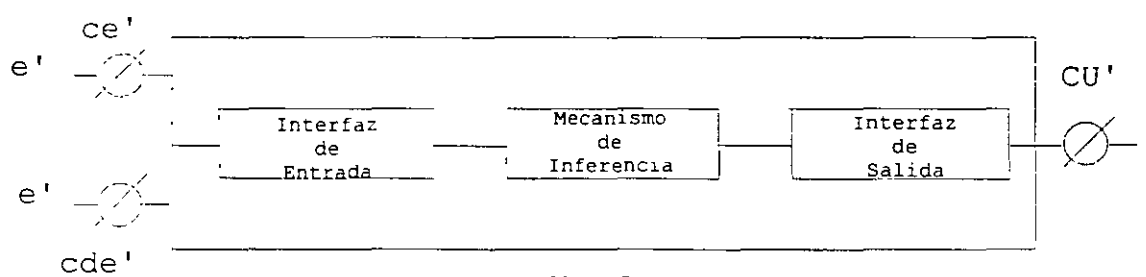


Fig. 25
Esquema básico de un Controlador Difuso.

Modos de Operación del Controlador Difuso

Información de Entrada	Información de Salida	
	Numérica	Difusa
Numérica	Control de Sistemas de Malla Cerrada	Control de Malla Abierta
Difusa	Control de Sistemas de Malla Cerrada	Control de Malla Abierta

Fig 26
Tipos de Controladores Difusos

1.- Entrada Numérica-Salida Numérica: Son los controladores que más se implementan en nuestros días. Se aplican en controles de malla cerrada no lineales. La información numérica es provista por sensores. Su implementación es funcional computacionalmente.
2.- Entrada Difusa- Salida Numérica: Específico para malla cerrada con acción humana que provee información lingüística adicional. No permite implementación compacta.
3.- Entrada Numérica Salida Difusa: La información de entrada es numérica su salida es un conjunto difuso que el operador observa y toma una decisión. Se usa en tutoriales.
4.- Entrada Difusa Salida Difusa. Es ampliamente explotado y puede funcionar como el modo genérico en el futuro (soporte en sistemas inteligentes).El control de la información es utilizada por el usuario quien es responsable de la acción final de control.

Los cuatro aspectos más importantes en el diseño de controladores difusos son:

-Integridad de las Reglas de Control.
-Consistencia de las Reglas de Control.
-Interacción de las Reglas de Control.
-Robustez del Controlador.

-Integridad de las Reglas de Control - Se da cuando el controlador puede generar la salida para cualquier entrada difusa. Todos los controladores difusos experimentales lo cumplen.
-Interacción de las Reglas de Control. - Todas afectan en cada salida, derivado propiamente de su construcción lógica de procesamiento en paralelo.
-Consistencia de las Reglas de Control. - Puede existir inconsistencia cuando existe un grado excesivo de información contradictoria o con criterios opuestos (bajo costo-alta eficiencia).
-Robustez del Controlador.- La diferencia entre el valor de control U obtenido por una entrada de información exacta (e y de) y U_{error} generada por una versión de ruido

(Gaussiano) es visto como un indicador de tolerancia al error. Los resultados obtenidos por COG determinan un valor sensato del control difuso absorbiendo la mayor cantidad de ruido.

METODOLOGIA DE DISEÑO DE UN CONTROLADOR DIFUSO

Es conveniente establecer una metodología en el diseño de un Controlador Difuso que permita abordar en una forma coherente la solución del problema. Se distinguen cinco pasos fundamentales que son:

Análisis y Partición del Sistema: Se aborda de manera general el problema, prácticamente se define. Para ello debemos:
Identificar Entradas y Salidas con el fin de obtener nuestras variables de control.
Analizar y Simplificar el Problema para poder particionar y resolver de manera modular aminorando el trabajo.

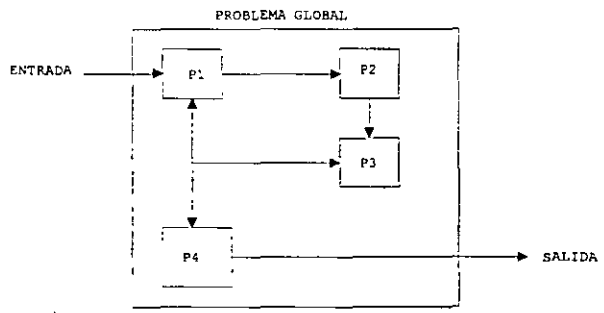


Fig. 27
Simplificación del Problema

Identificar las Unidades Difusas Hay que observar cuales son las entradas que manejan mayor ambigüedad para usar un controlador difuso o en caso contrario buscar métodos de control tradicional.

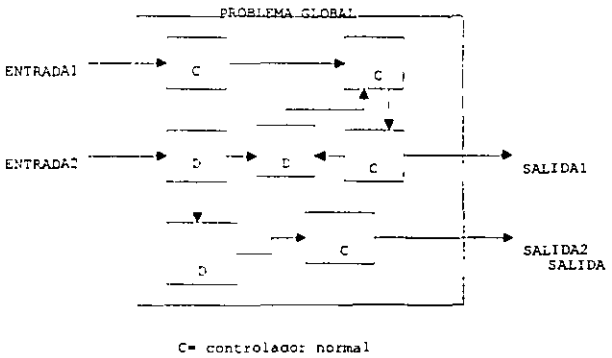


Fig. 28
Identificación de Unidades Difusas

Definición de Funciones de Membresía de Entrada y Salida El problema se decide resolver de manera difusa para ello es necesario:
Especificar el Universo para saber cual es el rango de variación de nuestras entradas y salidas y su comportamiento para establecer de manera adecuada los conjuntos difusos.
Escalamiento del Universo Dependiendo del comportamiento de las entradas y salidas se debe ajustar su escala, lineal, logarítmica o cualquier otra. Prácticamente depende de los sensores de las variables (potenciómetros, termopares, etc.).

Escritura de Reglas. En este paso se determinan las acciones de control de manera lingüística. En el proceso se puede distinguir:
Escritura de reglas obvias que implican el determinar a groso modo el control.
Determinar casos especiales que no resultan tan obvios al implicar criterios contradictorios como pudiera ser un bajo costo de una máquina y una alta eficiencia.
Sintonizar las reglas. Con el objeto de eliminar las contradictorias y llegar a una adecuada acción de control

Observación del Comportamiento del Modelo. Para verificar sus resultados y sintonizar bajo resultados, ello se puede hacer mediante:
Examen de resultados de salida. Ver si logra compensar el controlador para alcanzar los valores esperados (puntos de referencia).
Examen de las superficies de control. Siendo éstas la representación de las entradas con respecto a las salidas es necesario observar si no existen cambios demasiado bruscos o puntos que formen vacíos por contradicciones para responder de manera adecuada.

Optimización del Sistema Es el último paso dentro del diseño del controlador Se lleva a cabo en su entorno real para evaluar su comportamiento, volviendo a sintonizar y corregir si es necesario.

El control difuso y el HC12

Como ya se vio este microcontrolador incluye cuatro instrucciones para lógica difusa, además de mas memoria y funciones periféricas que un microcontrolador de propósito general. Las instrucciones difusas utilizan la lógica del CPU para realizar sus cálculos incluyendo suma, resta, multiplicación, multiplica-y-acumula, y en comparación, con la velocidad y eficiencia de los programas difusos se incrementa el improved sin incrementar el costo del microcontrolador. Un kernel de inferencia difusa en el M68HC12 es mucho mas pequeño y se ejecuta mas rápido que en el microcontrolador de propósito general M68HC11 .

KERNEL DIFUSO

A continuación se muestra una lista del kernel de inferencia difusa escrito en lenguaje ensamblador para el CP12. Los números en los paréntesis cuadrados son los contadores de ciclos. El Kernel utiliza dos entradas con 7 etiquetas para cada entrada y una salida con 7 etiquetas. El programa se ensambla en 57 bytes y se ejecuta aproximadamente en 54 µs a una velocidad de bus de 8 MHZ. La estructura se puede extender fácilmente a un sistema de propósito general con mas entradas y salidas

```
01 [2]   FUZZIFY     LDX   #INPUT_MFS      ;Point at MF definitions
02 [2]                LDY   #FUZ_INS        ;Point at fuzzy input area
03 [3]                LDAA  CURRENT_INS     ;Get first input value
04 [1]                LDAB  #7              ;7 labels per input
05 [5]   GRAD_LOOP    MEM                   ;Evaluate one MF
06 [3]                DBNE  B,GRAD_LOOP     ;For 7 labels of 1 input
07 [3]                LDAA  CURRENT_INS+1   ;Get second input value
08 [1]                LDAB  #7              ;7 labels per input
09 [5]   GRAD_LOOP1   MEM                   ;Evaluate one MF
10 [3]                DBNE  B,GRAD_LOOP1    ;For 7 labels of 1 input

11 [1]                LDAB  #7              ;Loop count
12 [2]   RULE_EVAL    CLR   1,Y+            ;Clr a fuzzy out & inc ptr
13 [3]                DBNE  b,RULE_EVAL     ;Loop to clr all fuzzy outs
14 [2]                LDX   #RULE_START     ;Point at first rule element
15 [2]                LDY   #FUZ_INS        ;Point at fuzzy ins and outs
16 [1]                LDAA  #$FF            ;Init A (and clears V-bit)
17 [3n+4]             REV                   ;Process rule list

18 [2]   DEFUZ        LDY   #FUZ_OUT        ;Point at fuzzy outputs
19 [1]                LDX   #SGLTN_POS      ;Point at singleton positions
20 [1]                LDAB  #7              ;7 fuzzy outs per COG output
21 [8n-9]             WAV                   ;Calculate sums for wtd av
22 [11]               EDIV                  ;Final divide for wtd av
23 [1]                TFR   Y,D             ;Move result to A:B
24 [3]                STAB  COG_OUT         ;Store system output

      ***** End
```

Figure 9-3 Fuzzy Inference Engine

Las líneas 1 a 3 inicializan los apuntadores y carga el valor de entrada al sistema en el acumulador A

La línea 4 inicializa el contador para el loop de las líneas 5 a 6.

Las líneas 5 a 6 realizan el loop para el proceso de fuzificación con las sete etiquetas para una entrada. La instrucción MEM encuentra el valor en el eje Y de la función de membresía para el valor de entrada presente, para una etiqueta de la entrada presente , y después coloca el resultado en su correspondiente entrada difusa. Los apuntadores X y Y son actualizados automáticamente por las líneas 4 y 1 y apuntan a la siguiente función de membresía y entrada difusa respectivamente.

La línea 7 carga el valor actual de la siguiente entrada al sistema. Los apuntadores X y Y nuevamente apuntan las localidades correctas como resultado de la actualización automática llevada a cabo por la instrucción MEM en la línea 5.
La línea 8 carga el contador nuevamente.
Las líneas 9 y 10 forman un loop para fuzificar las siete etiquetas de la segunda entrada al sistema. Cuando el programa llega a la línea 11, el registro de índice y apunta a la siguiente localidad después de la última entrada difusa, que es la primera salida difusa en el sistema.
La línea 11 pone contador del loop para limpiar las siete salidas difusas.
Las líneas 12 y 13 forman un loop para limpiar todas las salidas difusas antes de que la evaluación de reglas se inicie.
La línea 14 inicializa el registro de índice X para apuntar al primer elemento en la lista de reglas para la instrucción REV.
La línea 15 inicializa el registro de índice Y para apuntar a las entradas difusas y las salidas en el sistema. La lista de reglas (para REV) tiene un offset de 8 bits a partir de la dirección base para una entrada difusa particular o salidas difusa. El valor $FE es interpretado por REV como la marca entre el antecedente y consecuente.
La línea 16 inicializa el acumulador A en el valor mas alto de 8 bits como preparación para encontrar la entrada mas chica referenciada por el antecedente de la regla. La instrucción LDAA #$FF además limpia el bit V en el registro de código de condición de tal manera que la instrucción REV la procesa como antecedente. Durante el procesamiento de la lista de reglas, se estará revisando la bandera V, hasta que se detecte $FE en la lista. El bit V indicara cuando REV procese antecedente o consecuente.
La línea 17 es la instrucción REV, que lleva a cabo un loop para procesar los elementos en la lista de reglas hasta que encuentra el carácter $FF. Para el sistema de 17 reglas con dos antecedentes y un consecuente cada uno, la instrucción REV necesita 259 ciclos.
De la línea 18 a la 20 se inicializa los apuntadores y los contadores para la instrucción WAV, la línea 21 inicializa la defuzificación. La instrucción WAV calcula la suma de productos y la suma de pesos.
La línea 22 completa la defuzificación . La instrucción EDIV lleva a cabo una división de 32 bits entre16 bits de los resultados obtenidos por la instrucción WAV.
La línea 23 mueve el resultado de EDIV a el acumulador doble.
La línea 24 almacena los 8bits del resultados de la defuzificación.

El ejemplo anterior es un programa escrito en lenguaje ensamblador, sin embargo existe software de desarrollo que puede ser usado para el desarrollo de un sistema difuso.
Una herramienta de software de desarrollo par lógica difusa debería seguir el siguiente criterio:

Soportar todas la fases de desarrollo
(Diseño, simulación, Optimización, Verificación, Implementación)
Soportar todas las plataformas de hardware
(Microcontroladores, Controladores Lógicos Programables, Computadoras Personales, Estaciones de Trabajo, Sistemas de Control Distribuido para Procesos)
Soporte de interfaces para Estándares Industriales
(Interface para usuarios de Windows, DLL/DDE/OLE, interface para herramientas de simulación de software)

FuzzyTECH es un software de desarrollo completo que permite desarrollar sistemas basados en lógica difusa y redes neuronales. Para implementación en microcontroladores, fuzzyTECH ofrece generadores de código ensamblador que aseguran una desempeño computacional aprovechado al máximo.
El fuzzyTECH MCU-68HC12 Editions es una versión del fuzzyTECH dedicado a la familia de los microcontroladres M68HC12 de motorola. En cooperación con motorola, los expertos de lógica difusa de Inform desarrollaron un generador de código que optimiza las funciones de lógica difusa en el M68HC12.
En muchas aplicaciones, al implementar una estrategia de control esta pude ser optimizado en el proceso de ejecucción. FuzzyTECH soporta este tipo de optimización. Tiene un módulo RTRCD que utiliza el modo de debug del M68HC12 para:

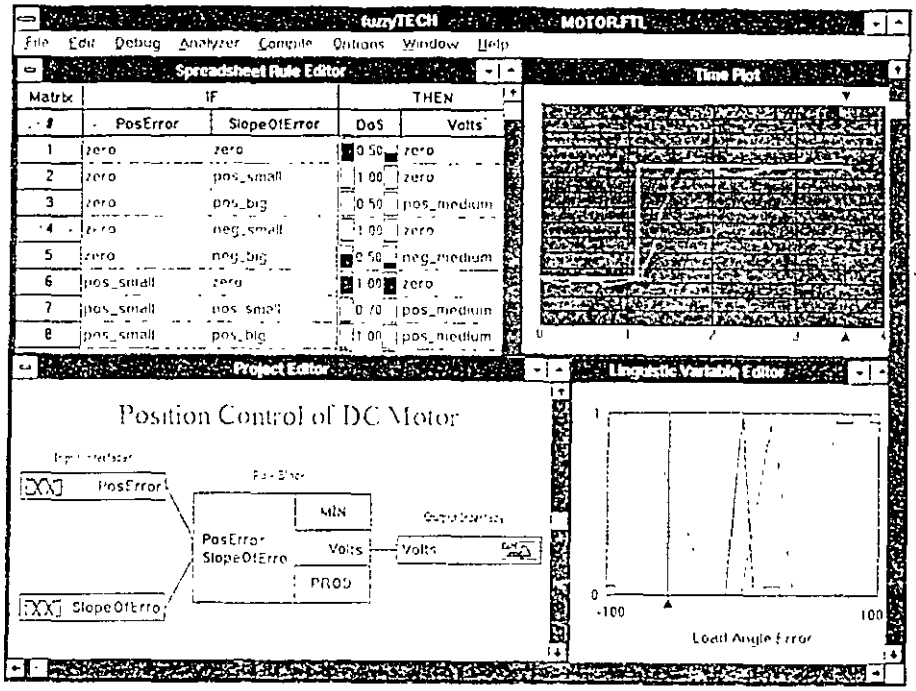
1. Visualizar la inferencia difusa completa en tiempo real usando los analizadores y editores dinámicos del fuzzyTECH.
2. Se puede hacer cualquier modificación en tiempo real sin interferir con el proceso ejecutandose.

Se presentara una pequeño resumen que muestra un diseño visual de un sistema completo. Tiene además varias técnicas de debug usadas para simulación, optimización y verificación.

Diseño Gráfico.

El camino difícil para implementar un sistema con lógica difusa es usar un lenguaje de programación y codificar todo el sistema difuso manualmente. Este es un buen ejercicio de programación, pero no es muy claro para entender cual es la lógica difusa. Además, cualquier modificación del sistema difuso pude causar una re-programación.

Precompilador Difuso
El segundo camino es usar un precompilador de lógica difusa. Estos precompiladores básicamente hacen una traducción de la descripción textual del sistema difuso en un lenguaje standard de programación, tal como el ANSI C.



Interfaces de diseño visual.

El camino mas eficiente para desarrollar un sistema difuso es usar una interface de diseño visual. Para almacenar el diseño del proyecto en disco, la herramienta usa un formato de descripción estándar, FTL. Además la herramienta lee en el sistema codificado-a mano en FTL y lo representa gráficamente. Integra además generadores de código para implementar el sistema difuso en C, ensamblador o PLC (Programmable Logic Controller) codificándolo inmediatamente de la interfase gráfica.

Para diseñar un sistema difuso gráficamente deben seguirse los siguientes pasos:

Paso 1 Estructura del Sistema

El primer paso en el diseño de sistemas difusos es la definición de la estructura del sistema. Se definen las entradas y salidas del sistema difuso y como interactúan entre ella. Los bloque pequeños de lado izquierdo son las interfaces de entrada. Las interfaces de entrada contienen además la fuzificación de los valores de entrada, el icono en la izquierda indica el método de fuzificación empleado. Los bloque pequeños en el lado derecho son las interfaces de salida que contienen el método de defuzificación. El bloque grande en la mitad de la pantalla es el bloque de reglas. Cada bloque de reglas contiene un conjunto de reglas difusas independientes. El testo remarca la estructura del sistema. El Editor, además permite el acceso a los componentes del sistema difuso. Por ejemplo, haciendo doble click en un bloque de reglas se abre el editor de reglas contenidas en él. Para crear un nuevo objeto en el Editor de Proyectos, se hace un click en un área vacía. Cuando está ejecutando un sistema difuso en un hardware distribuido, mas de un editor de proyecto existe.

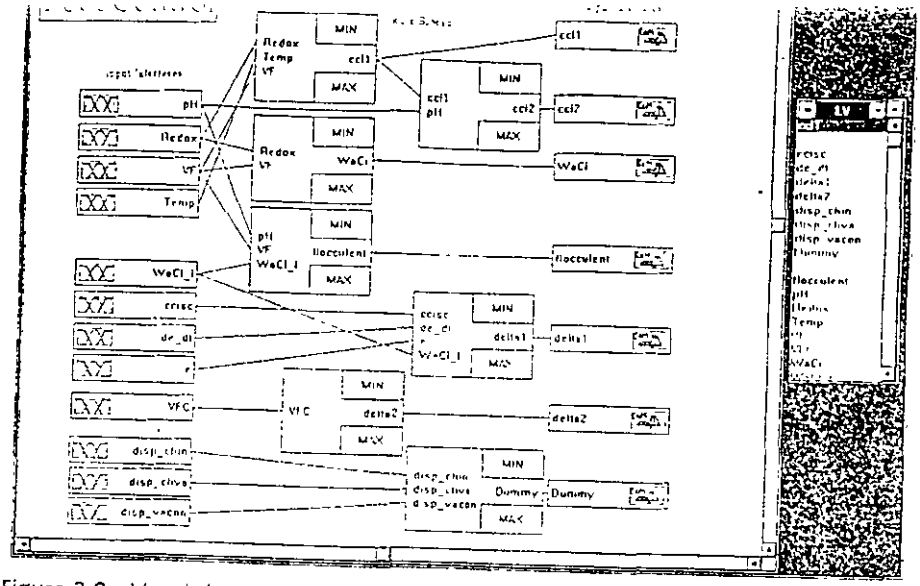


Figure 3 2 Visual design of the fuzzy logic system's structure in the Project Editor

Paso 2 Variables Lingüísticas y funciones de membresía

El siguiente paso en el diseño es definir el "vocabulario" del sistemas, esto es, las variables lingüísticas. Una variable lingüística es global en el sistema difuso, y la ventana LV , lista estas por nombre. Haciendo doble click en una nombre de variable existente o creando una nueva se invoca al editor gráfico para las funciones de membresía de la funciones de las variables respectivas.

El camino mas conveniente para diseñas las funciones de membresía es la definición del point-wise. Haciendo doble click en un espacio vacío se genera una nueva definición de punto. Haciendo click y moviendo el mouse se pude mover la definición del punto. Se pueden ligar puntos por lineas straight o funciones spline.

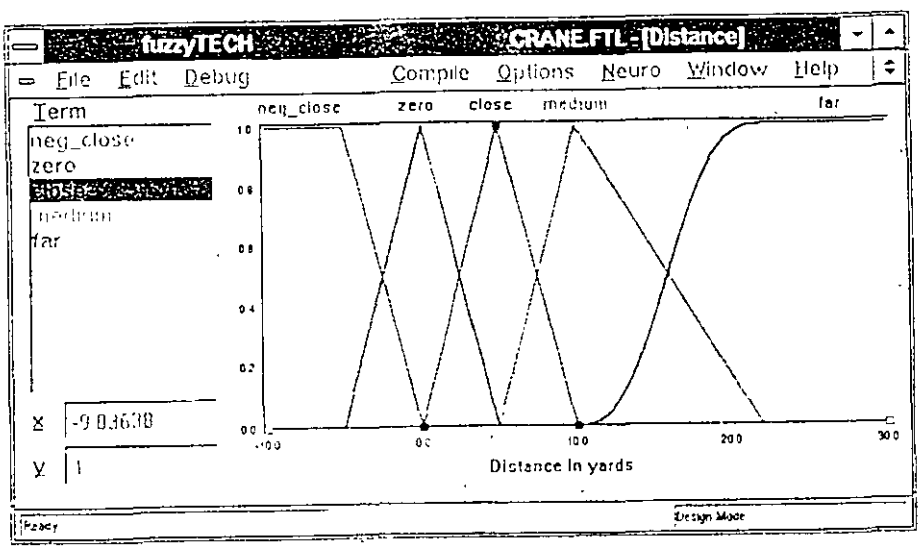


Figure 3 3 Visual design of membership functions of a linguistic variable in the Variable Editor

Paso 3 Bloque de reglas

La estrategia de control reside en la definiciópn de las reglas, la figura 3.4 muestra tres alternativas diferentes de definir y editar el bloque de reglas de un sistema difuso. Para pequeños blores de reglas el Spreadshet rule Editor es el mas conveniente. Cada columna representa una regla y el número en la columna izquierda es el número de regla. La primera columna tiene tres botones. El botón de Matriz abre el editor de la matriz de reglas y los otros dos botones, IF y THEN , títulos de las reglas parte if y parte Then. Haciendo click en estos botones se puede cambiar el método de inferencia. La segunda columna contiene

botones para cada variable. Haciendo click en los botones acomoda la base de reglas respecto a los términos de variable. El botón DoS representa el peso de cada regla, como el fuzzyTECH usa la extensión FAM en las reglas difusas. Para modificar cualquier parte de la regla, hay que hacer click en el elemento respectivo.
El editor de matriz de reglas entrega una manera mejor de ver el bloque de reglas. La matriz siempre representa la relación entre las reglas en la base. Estas siempre muestran como es la relación entre la definición de reglas entre dos variables. Para cada variable, el renglón o la columna en la matriz representa el término y los elementos de la matriz representan una regla. El tono de gris indica el peso de la regla. Si mas de dos variables se definen en el bloque de reglas, la matriz del editor de reglas lista estas en la parte baja de la ventana.

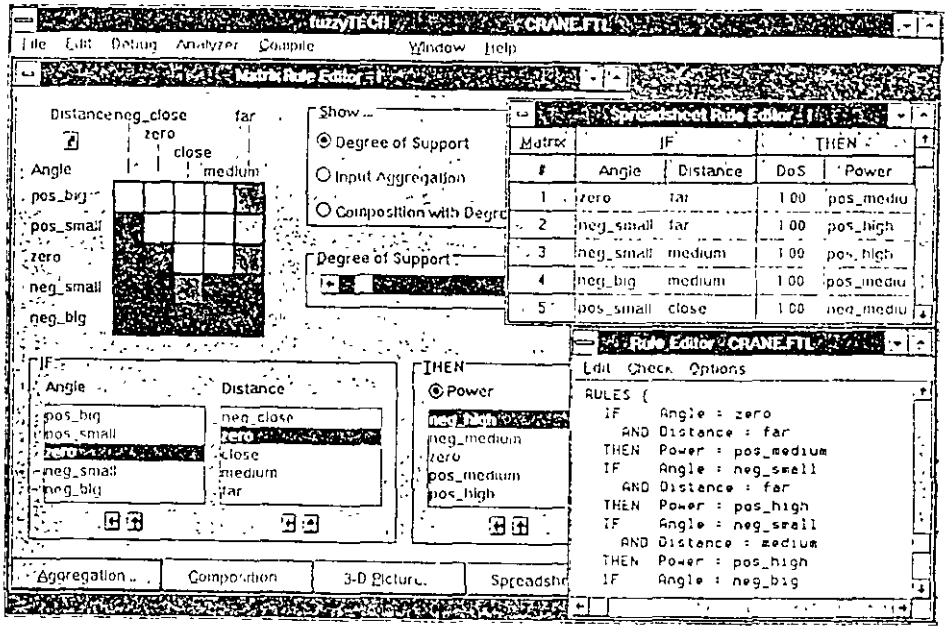


Figure 3.4 Alternative visual design of rules in the Matrix Rule Editor, the Spreadsheet Rule Editor, and in FTL text format

Modos de Debug para simulación, optimización y verificación.
El beneficio primario de usar un software de desarrollo es un soporte de debug compensivo. La experiencia muestra que mas del 90% del tiempo de desarrollado de sistemas difusos se pasa en el proceso de simulación, optimización y verificación.
Las herramientas que soportan este modo de debug son:

Interactivo (entrega una reacción del sistema a entradas)
fT-Link (liga a un proceso de simulación)
Serial Link (liga a una simulación o a un proceso por el puerto de comunicación COM)
DDE Link (liga con cualquier otro software que soporte DDE)
File Recorder (procesa datos de un archivo)
Connection (entrega su conección para correr un sistema difuso para traficar)
Monitor (tiene un monitor para un sistema difuso corriendo en tiempo real)
Online (se puede modificar un sistema difuso en tiempo real)

Debug Interactivo

El debug interactivo visualiza la inferencia difusa gráficamente. Todos los editores que son usados para diseñar sistema actualmente despliegan los cálculos. Se puede ver como las variables son fuzificadas, cuales reglas están activadas y el grado, y que defuzificación calculo. Fig. 5. El factor mas importante es que se pude modificar la mayoría de las partes del sistema mientras se observa como se afecta su desempeño. Si se mueve la definición del punto de una función de membresía, se puede ver todos los efectos de esta modificación en las reglas y la defuzificación.

Además, las reglas pueden ser modificadas mientras el modo debug esta activo, y se pueden agregar nuevas reglas. Las barras pequeñas a lado del valor del peso de la regla en la columna DoS indica el grado de firing de la entrada (verdad de la parte if), y el grado de firing de salida (verdad de la parte then)

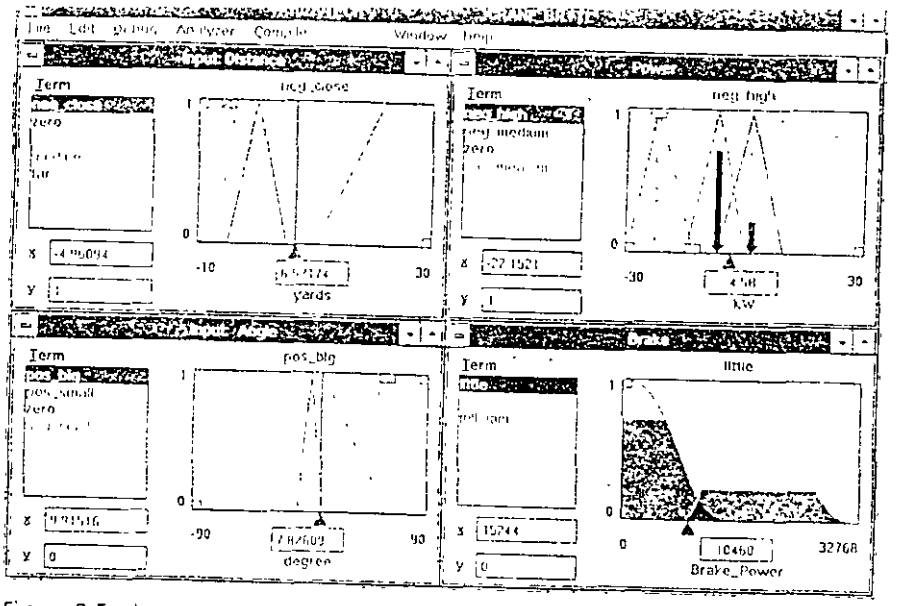


Figure 3 5 Interactive debugging lets you see the effects of all modifications immediately The red lines with the arrow indicate the non-fuzzy inputs and outputs. Drag the arrows of the input variable to change its value Move the definition points of the membership functions to optimize the system.

fT-Link y DDE-Link

Para algunas aplicaciones, el proceso matemático de simulación existe. Estos procesos pueden ser integrados un una herramienta de diseño difuso, tal como la inferencia difusa pude ser monitoreado y modificado en tiempo real. Link debuging soporta la mayoría de los lenguajes standard de programación como Mathlab/Simulink®, VisSim™, y Matrixx™.

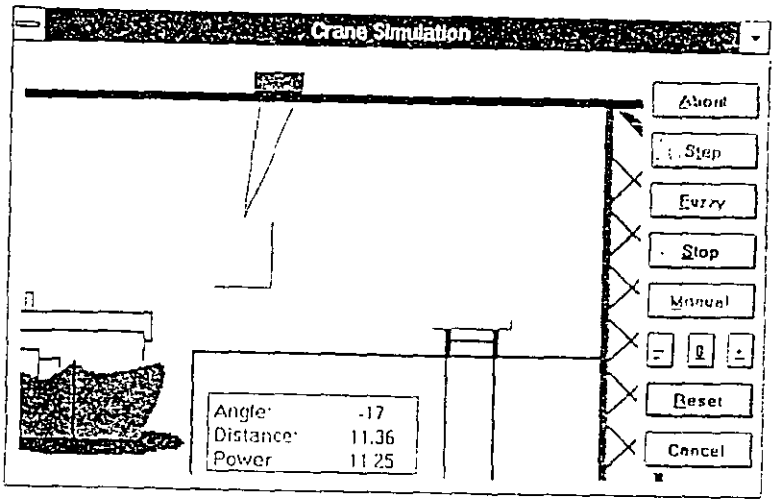


Figure 3.6 Process simulations can be written in any programming language and be linked to fuzzyTECH in Link Debug Mode

Serial Link

En algunos casos, la simulación no se pude llevar a cabo en el mismo hardware que el sistema de desarrollo difuso. Este modo de debug permite la entrada de variables al sistema difuso a través del puerto serial de la PC y regresa la variable de salida de la misma manera. En principio, se pude usar el modo debug serial para un control en tiempo real. Sin embargo, la respuesta en tiempo esta limitada y es indeterministica ya que depende de la "corrida" en la PC.

File Recorder

El Modo File Recorder permite usar datos de procesos. Como un control VCR, permite navegar a través de conjuntos de datos de entrada de un archivo. Este modo permite entender y analizar la reacción de un sistema difuso en una situación real de proceso. Además permite analizar el soporte "que-if"

El modo batch trabaja en forma similar a file recorder, pero solo escribe las salidas calculadas de un sistema difuso en un segundo archivo. Este modo es muy usado para verificar análisis de datos.

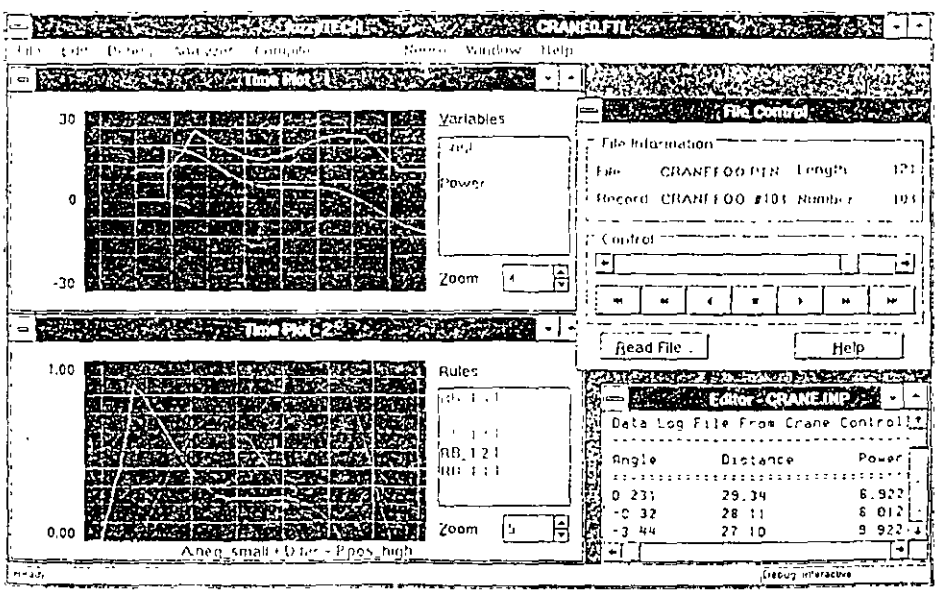


Figure 3.7 In file debug mode, a VCR-like control lets you navigate through pre-recorded process data

Modo Connection, Monitor y Online

Todos estos modos permiten ligar un controlador difuso ejecutándose en una tarjeta de hardware. En el modo connection, se pude ligar al sistema difuso ejecutándose para configurarlo, El control y upload trace se hacen en el controlador (hardware) El modo monitor permite visualizar todo el flujo de inferencia difusa y el modo online permite modificar el sistema difuso "on-the-fly".

Analizadores

Para optimizar, probar y verificar el diseño difuso, se utilizan los analizadores, Se pueden usar todos los analizadores en la mayoría de los modos debug. Los analizadores típicos son:

3D Plot
Transfer Plot
Time Plot
Statics

El 3D Plot visualiza la característica de transferencia de un sistema difuso. El color y la altura del contorno de la gráfica indica el valor de la variable de salida. La gráfica muestra regiones lineales, regiones no monótonas y regiones de instabilidad, donde no se activan reglas. Cuando se liga a un sistema ejecutándose (modo online) a simulación (fT-Link o DDE Link), o a través de archivos (file recorder), el punto de operación actual pude ser trazado. Esto permite detectar reglas superfluas o redundantes-

El Time Plot dibuja el valor de variables seleccionadas fuera de tiempo. Además dibuja grados de activación de reglas fuera de tiempo. Diferentes time plots pueden ver diferentes variables en diferentes escalas de tiempo.

Cundo se liga a un sistema ejecutándose (modo Online) a simulación (fT-Link o DDE Link), o a través de archivos (file recorder), el analizador estático recorre como cada regla es activada, y cual es el grado máximo y mínimo de activación . Esto permite analizar la importancia de varias reglas en la base de reglas.

Descripción de lenguajes difusos.

Para facilitar el diseño de sistemas difusos, se han creado lenguajes de programación difusos. Estos lenguajes contiene todos los elementos de un sistema difuso. Compiladores específicos y precompiladores convierten esta descripción en código ensamblador o algún lenguaje de programación como el C.

Actualmente, la mayoría de los diseñadores usan la herramienta de diseño visual en lugar de lenguajes de programación . Sin embargo, la mayoría del software de diseño usa lenguajes de programación como un formato de archivo debido a que entregan una descripción del sistema independiente del hardware.

Este simulador usa el FTL (Fuzzy Technology Language) el cual es soportado por compañías tales como Intel, Texas Instrument, Microchip, Allen-Bradley, Foxboro, Siemens-HG, SGS-Thomson, y Klockner-Moeller.

El fuzzyTECH MCU-68HC12 Editions genera además código para el M68HC12.

El microcontrolador MC68HC16

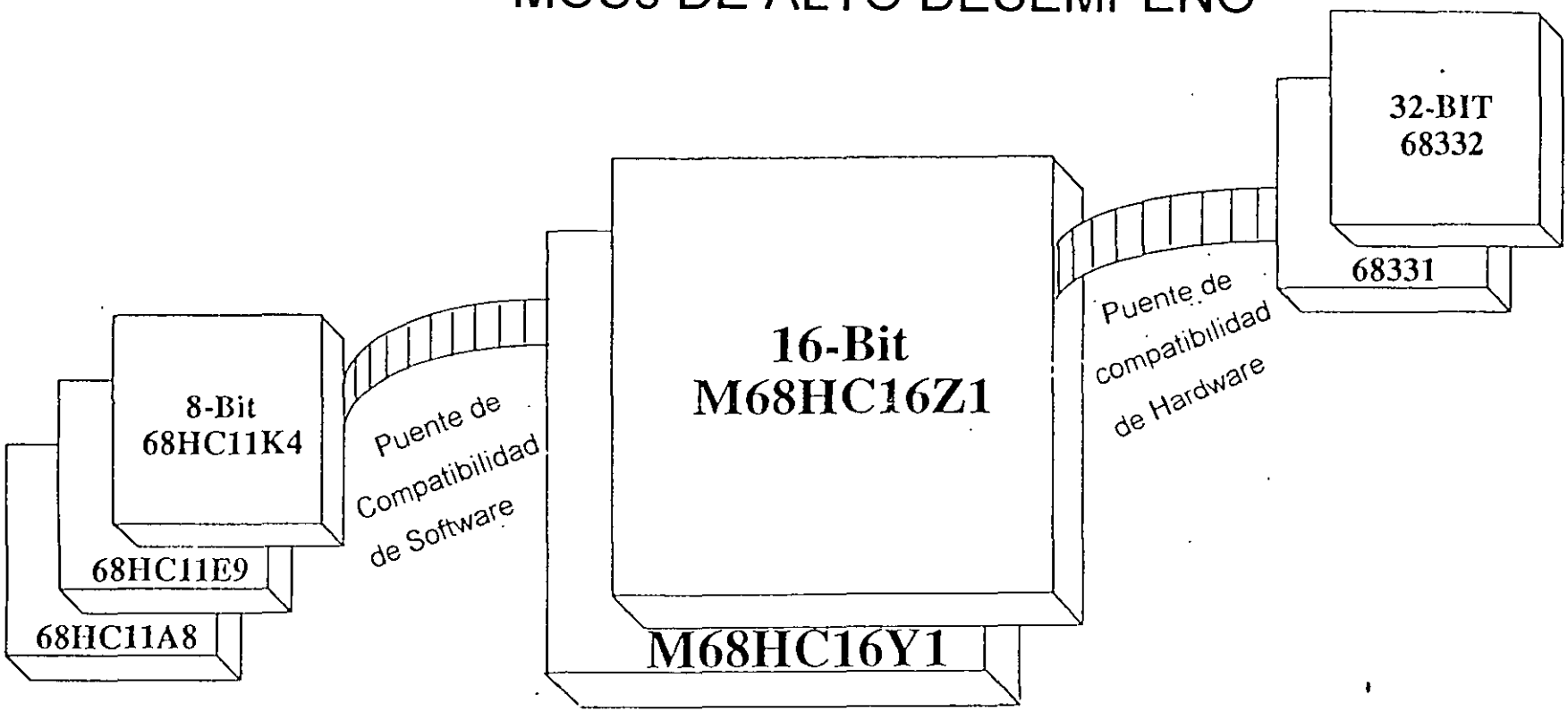
La arquitectura del 68HC16 es un cambio hacia 16-bit en la trayectoria de los microcontroladores de 8-bit de Motorola, especialmente es un paso hacia adelante en la familia 68HC11. Los miembros de la familia 68HC11 incluyen varios arreglos de memoria, convertidores analógico digital, dos tipos de puertos serie (asíncrono y asíncrono), temporizadores complejos, temporizadores con modulación de ancho de pulso, selectores de circuitos programables y sistema de protección de circuito. El microcontrolador MC68HC16Z1 es la versión original de la familia 68HC16 e incluye periféricos similares al 68HC11. El MC68HC16Z1, originalmente diseñado para control de discos duros, es un microcontrolador HCMOS con 1K byte de Memoria RAM (SRAM), un Temporizador de Propósito General (GPT), un Módulo Serie Encolado (QSM), un convertidor A/D de 8/10-bit (ADC), y un Módulo de Integración de Sistema (SIM). El diseño de microcontroladores modulares de la familia 68HC16 usa módulos estandarizados los cuales se conectan a través de un bus inter-módulos (IMB), parecido a la forma en que se conectan las tarjetas al bus ISA de una computador personal. El uso de módulos estandarizados permite un desarrollo rápido de nuevos microcontroladores los cuales cumplan las especificaciones de los clientes. El siguiente derivativo es el MC68HC16Y1, diseñado originalmente para el control de motores en los automóviles, cuatro nuevos módulos se ha añadido a la familia 68HC16, 48 Kbytes de ROM, Interfase de Comunicación Multicanal (MCCI), Módulo de Integración de Circuito (SCIM) y una Unidad de Procesamiento de Tiempo (TPU), y los módulos ya mencionados anteriormente ADC, GPT, y 2 Kbytes de RAM figura 1.

Familia de Microcontroladores 68HC16

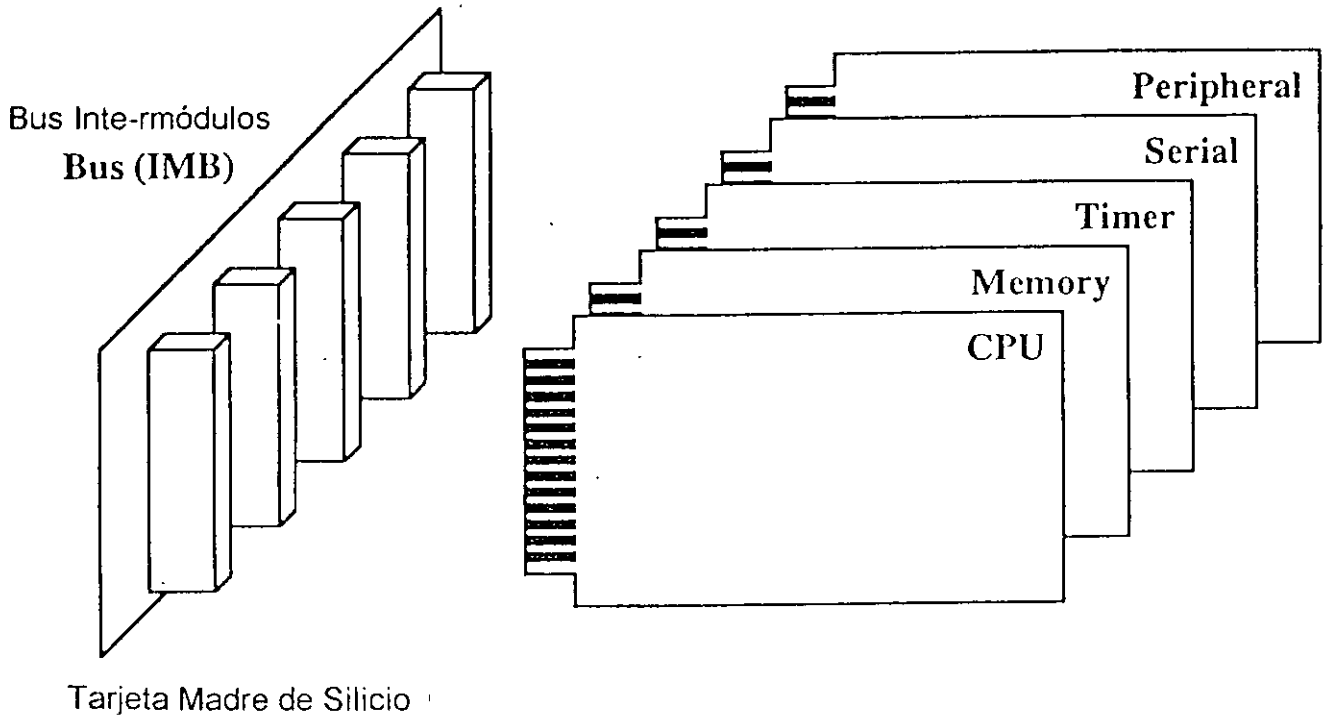
	Sistema	Temporizadores	Analógico	E/S serie	Ram	Rom
MC68HC16Z1	SIM	GPT	ADC	QSM	1K	---
MC68CH16Z2	SIM	GPT	ADC	QSM	2K	8K
MC68HC16Y1	SCIM	TPU,GPT	ADC	MCCI	2K	48K

El procesador de la familia 68CH16, conocido como el CPU16, tiene un código fuente compatible con la familia 68HC11. Esto permite el uso de software base existente para el 68HC11 en el 68HC16. sin embargo, este hecho no eclipsa las capacidades adicionales del CPU16 el cual adopta el uso eficiente de las técnicas modernas de programación. Esta arquitectura

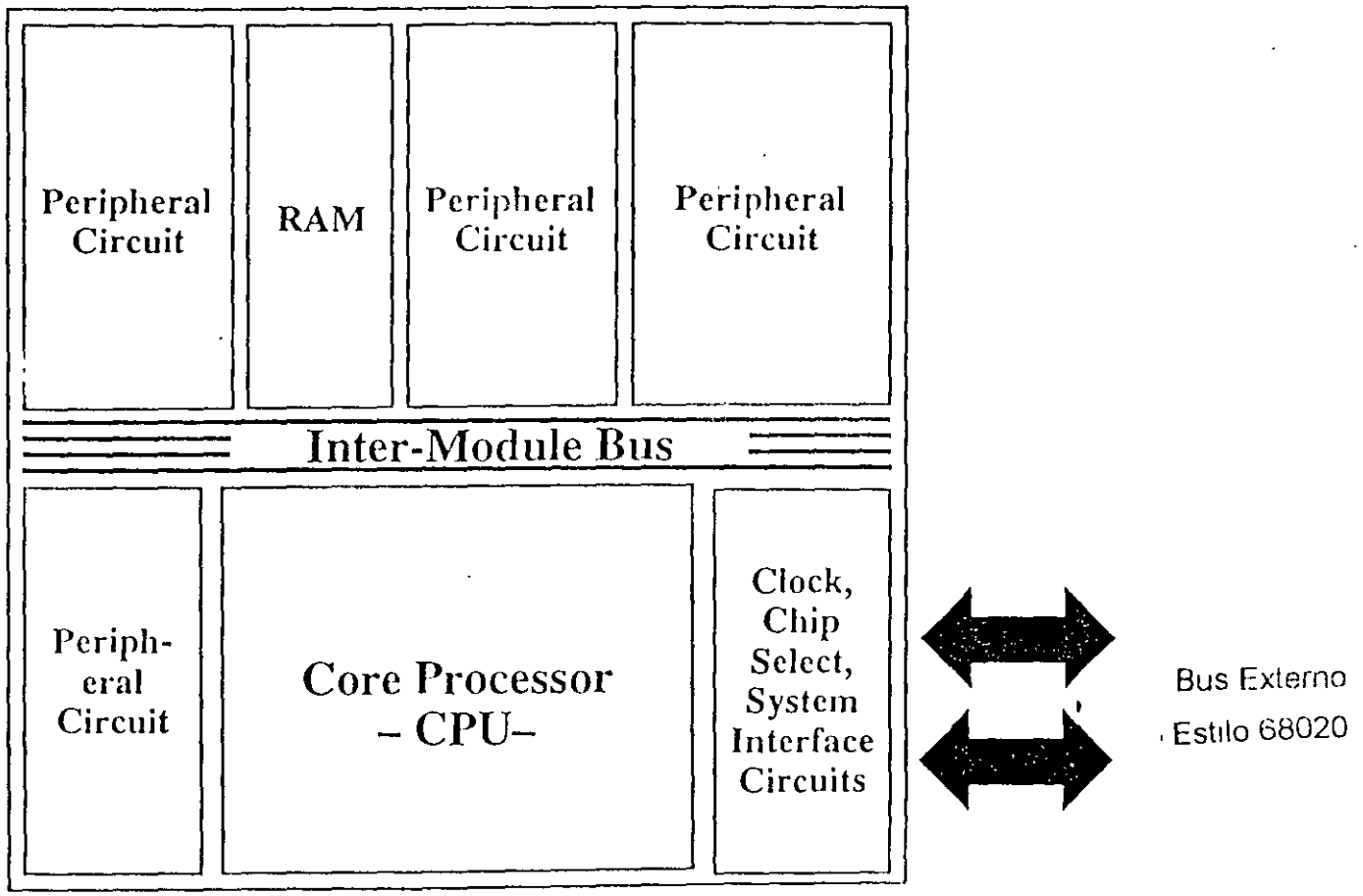
MCUs DE ALTO DESEMPEÑO



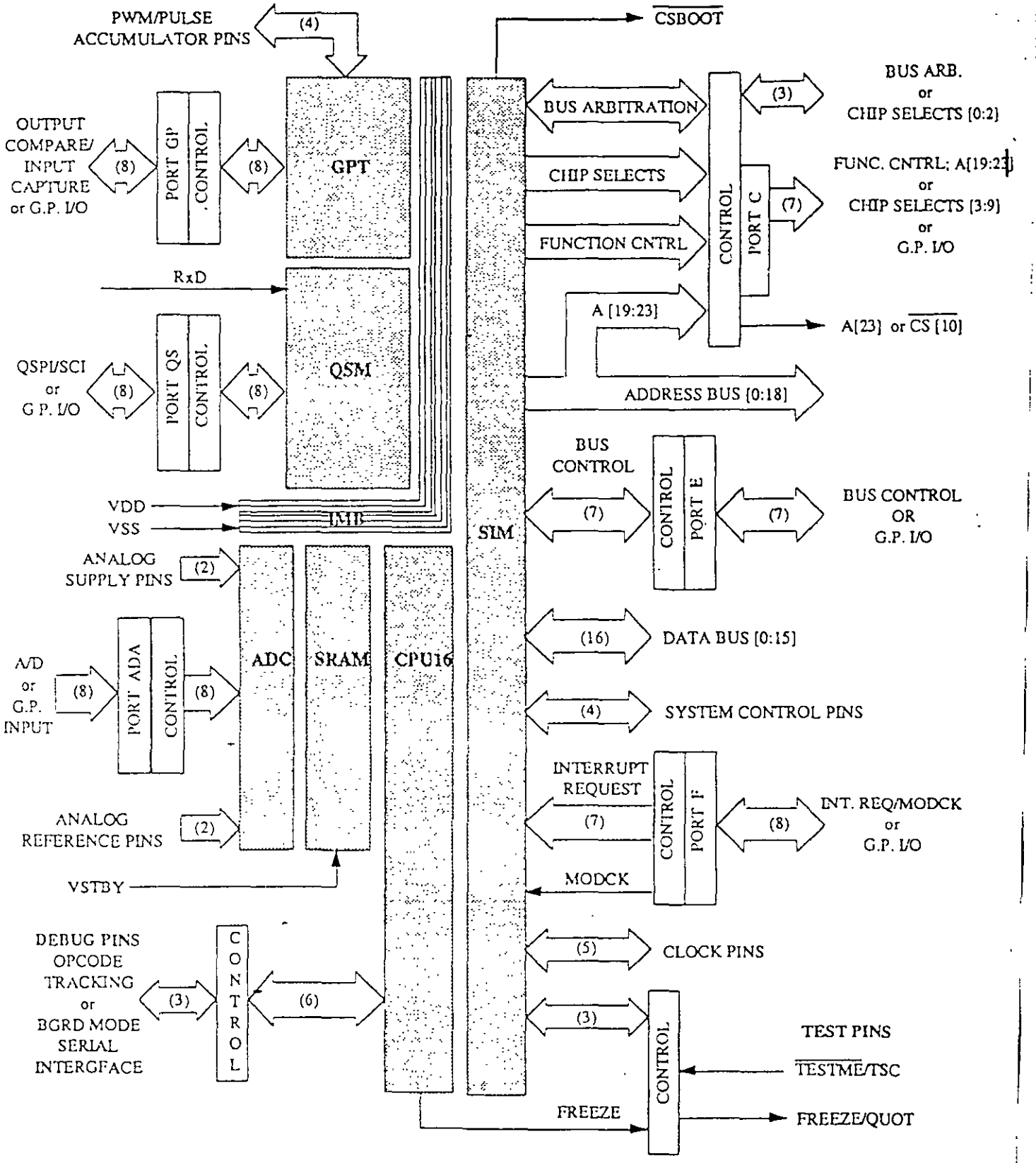
DISEÑO MODULAR



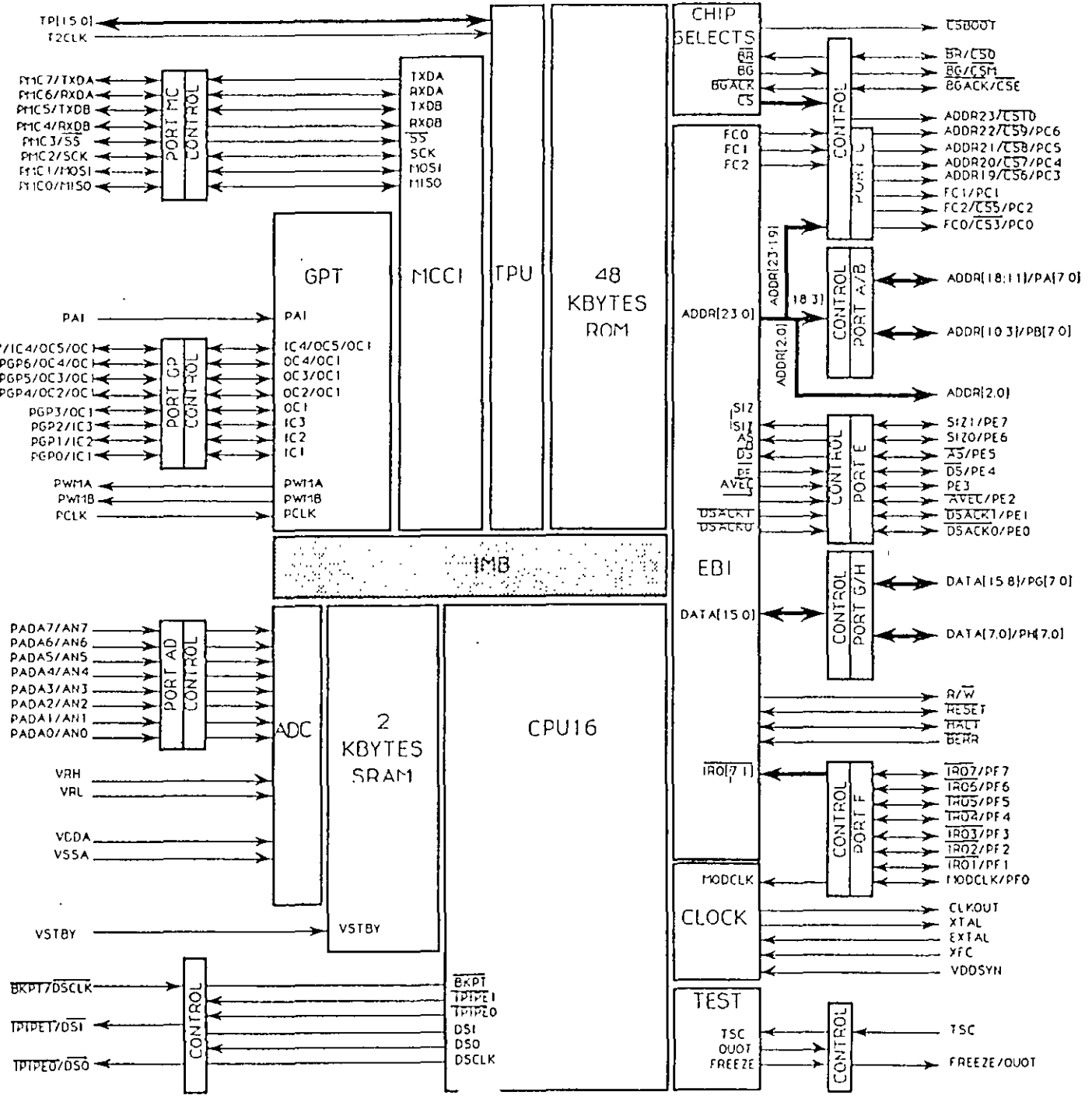
ARQUITECTURA MODULAR



68HC16Z1 DIAGRAMA A BLOQUES SIMPLIFICADO



HC16Y1
DIAGRAMA A BLOQUES

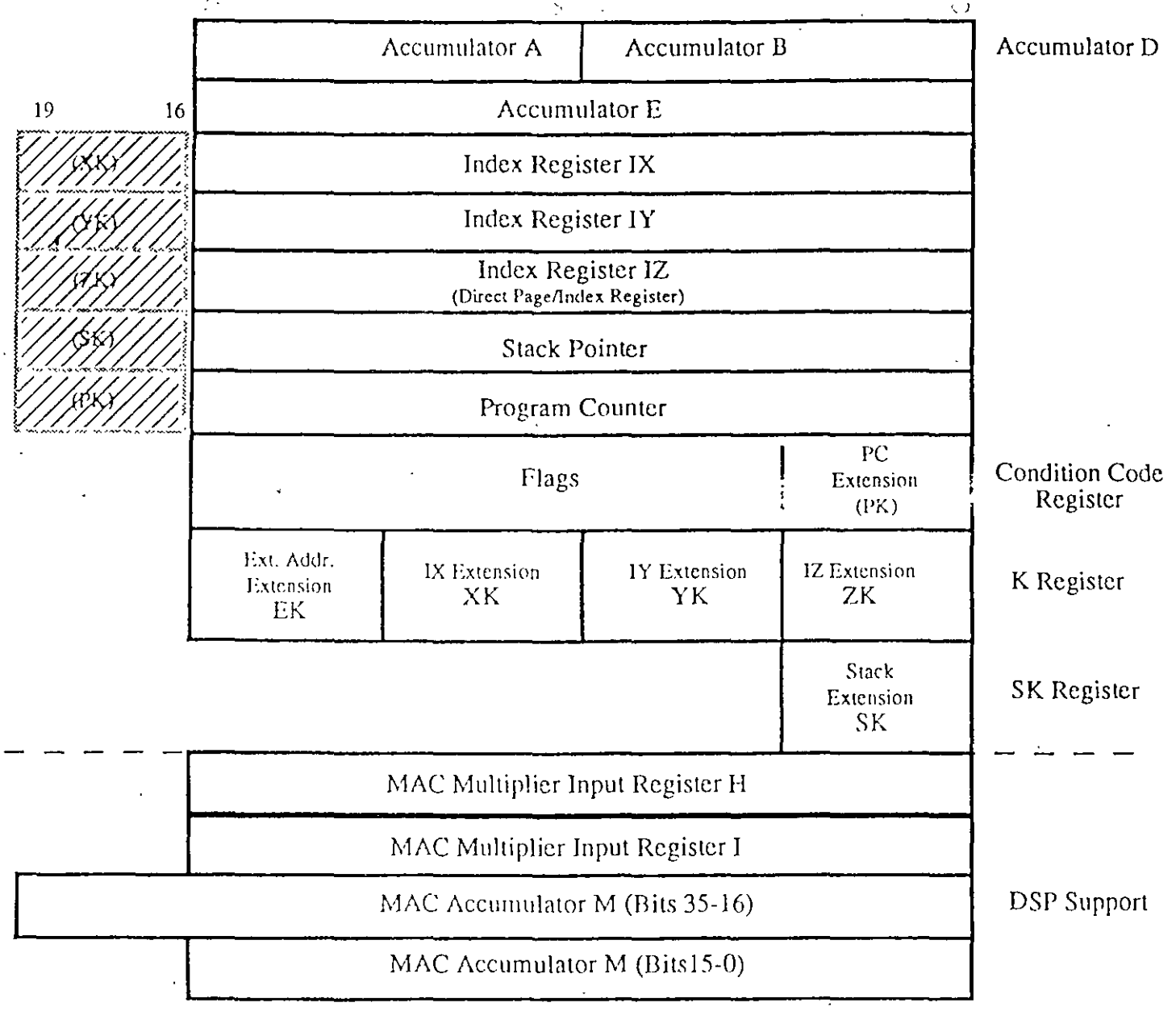


mejora grandemente los beneficios de los programadores que escriben programas en lenguaje assembly, haciendo la escritura de programas más fácil y también el código más eficiente para ser usado con las técnicas modernas de programación. Las mejoras también benefician a los programadores que usan el lenguaje C porque el CPU16 permite la implementación de compiladores más eficientes.

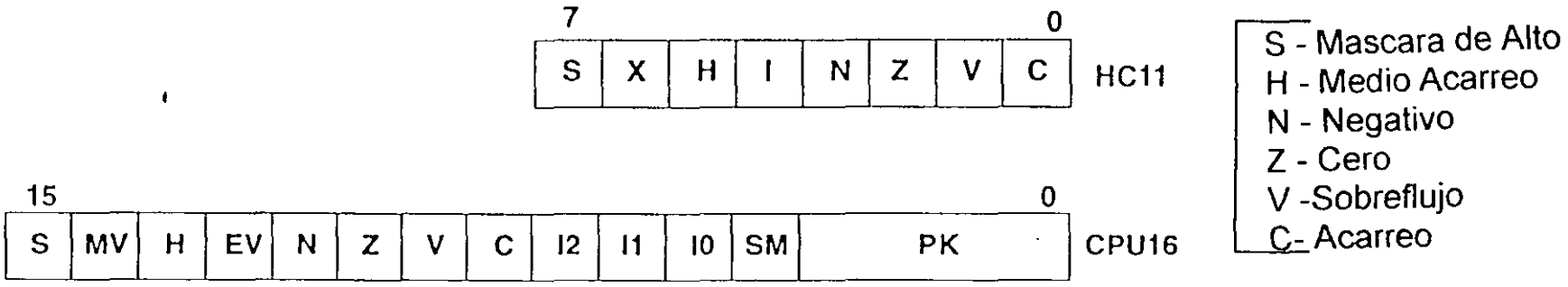
El CPU16

El CPU16 es una extensión completa a 16 bit del CPU11 usado en la familia de microcontroladores 68HC11. El CPU16 añade nuevos registros a su arquitectura, esto se muestra en la figura 2, añadiéndose con esto una gran funcionalidad. Ahora existen dos acumuladores de propósito general de 16-BIT (D y E), tres registros de índice de 16-BIT (X,Y, y Z), un registro de condición de código expandido (CCR) y registros de extensión de dirección (K). El acumulador adicional de 16 bit provee una mayor eficiencia y flexibilidad cuando se evalúan expresiones que tienen operandos de 16-bit. El tercer registro de índice provee flexibilidad en el acceso a datos globales y/o locales. La adición de los registros de extensión de dirección permite al CPU16 direccionar hasta 1M-byte de espacio de memoria figura 3,4,5.
El CPU16 también ofrece la funcionalidad integrada del procesamiento digital de señales (DSP) dentro del conjunto de instrucciones y un hardware para soportar algoritmos DSP de baja frecuencia en tiempo-real. Esto permite al microcontrolador servir como procesador digital de señales y como controlador. La función más comúnmente usada de procesamiento digital en sistemas embebidos de control es el FILTRADO, pero también se usan otras funciones de DSP. Están incluyen auto correlación (detección de una señal periódica en la presencia de ruido), correlación-cruzada (determinación de la presencia de una señal periódica definida), y rutinas de control de lazo cerrado (filtrado selectivo en una trayectoria de retroalimentación). Aunque la derivación de algoritmos de DSP es frecuentemente una tarea matemática compleja, los algoritmos por si mismos consisten típicamente de una serie de operaciones de multiplicación y acumulación (MAC). (El CPU16 dedica un conjunto de registros, colectivamente llamados la unidad MAC, que sirve para ejecutar operaciones MAC en tiempo-real). Para incrementar el throughput (la velocidad de datos de salida) el CPU16 ejecuta cálculos de dirección efectiva y prebúsquedas durante las operaciones MAC. En adición, la unidad MAC provee el modo de direccionamiento por módulo, para implementar eficientemente buffers circulares de DSP. La unidad MAC

CPU16 MODELO DE PROGRAMACION



REGISTRO DE CONDICION DE CODIGO



Los bits S,H,N,Z,V y C operan exactamente como en el M68hc11

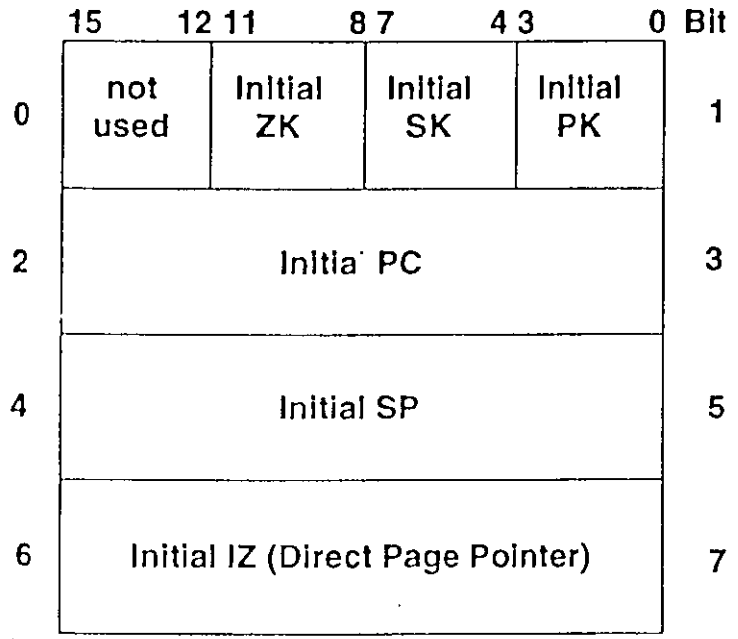
I2,I1,I0 - Mascara de prioridad de Interrupción

DSP

SM - Habilita Modo de Saturación de Dato
EV - Bit de Extensión de Sobreflujo
MV - Acumulador M de Sobreflujo
PK - Los 4 bits Superiores de una dirección de 20-bit

INICIALIZACION DE REGISTROS

PK:PC, SK:SP, ZK:IZ son inicializados con las primeras cuatro palabras en la Tabla de Vectores durante el RESET:



TERMINALES FC

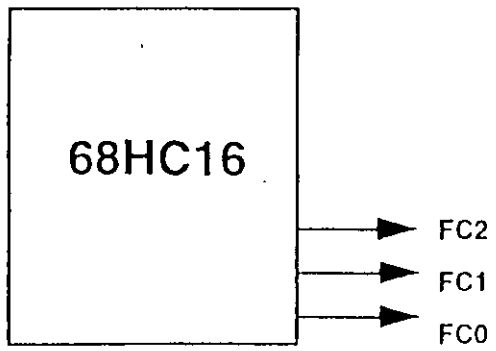
-Control de Función (FC)

-FC2 siempre en 1 en el HC16 (1=Supervisorio; 0= Usuario)

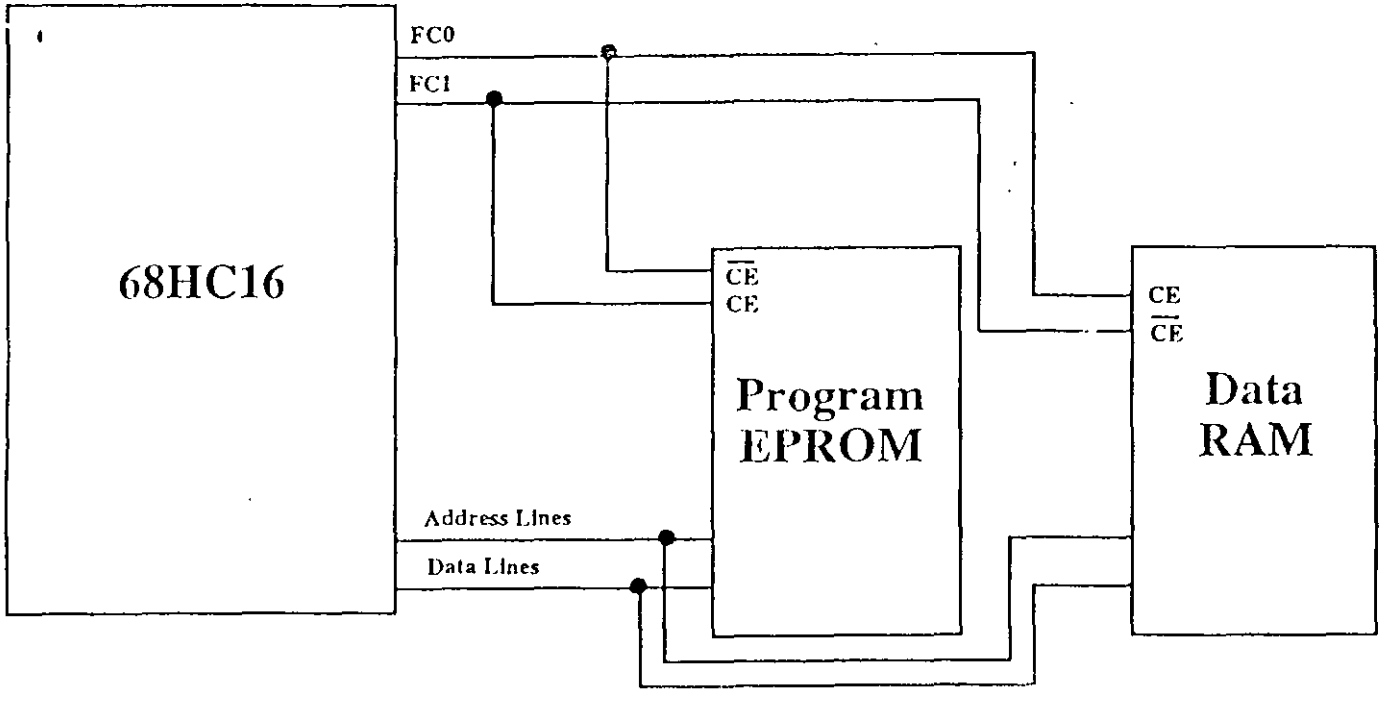
(1= Supervisor; 0= Usuario)

-FC2,FC1,FC0 terminales usadas alternativamente como selectores de circuito CS5,CS4,CS3

FC2	FC1	FC0	Espacio de Dirección
1	0	0	Reservado
1	0	1	Espacio de Dato
1	1	0	Espacio de Programa
1	1	1	Espacio de CPU (3 Tipos en el HC16: LSTOP, BREAKPOINT ACKNOWLEDGE e INTERRUPT ACKNOWLEGE; Líneas de Decodificación de dirección 19-16 para distinguir entre los tres)

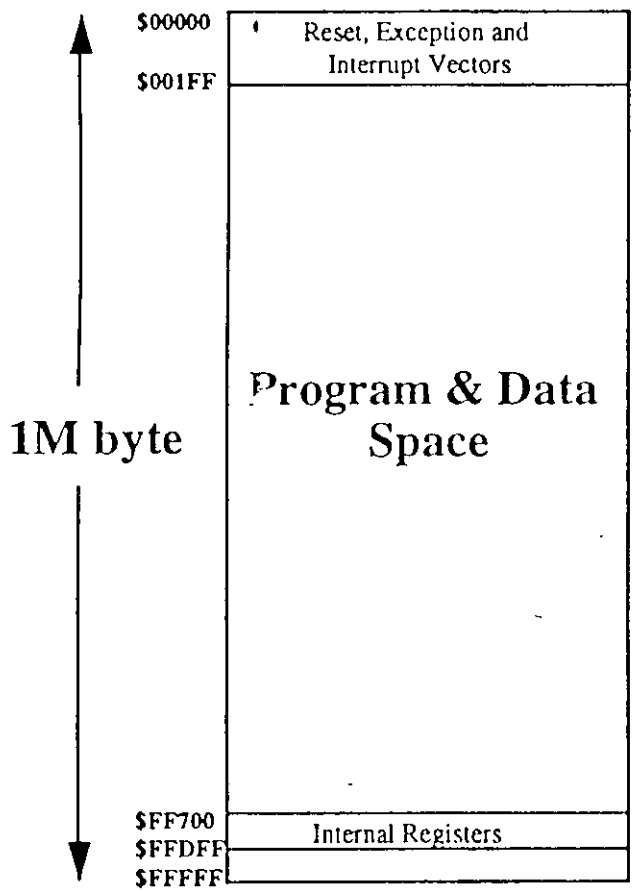


MAPAS SEPARADOS DE DATOS Y PROGRAMA

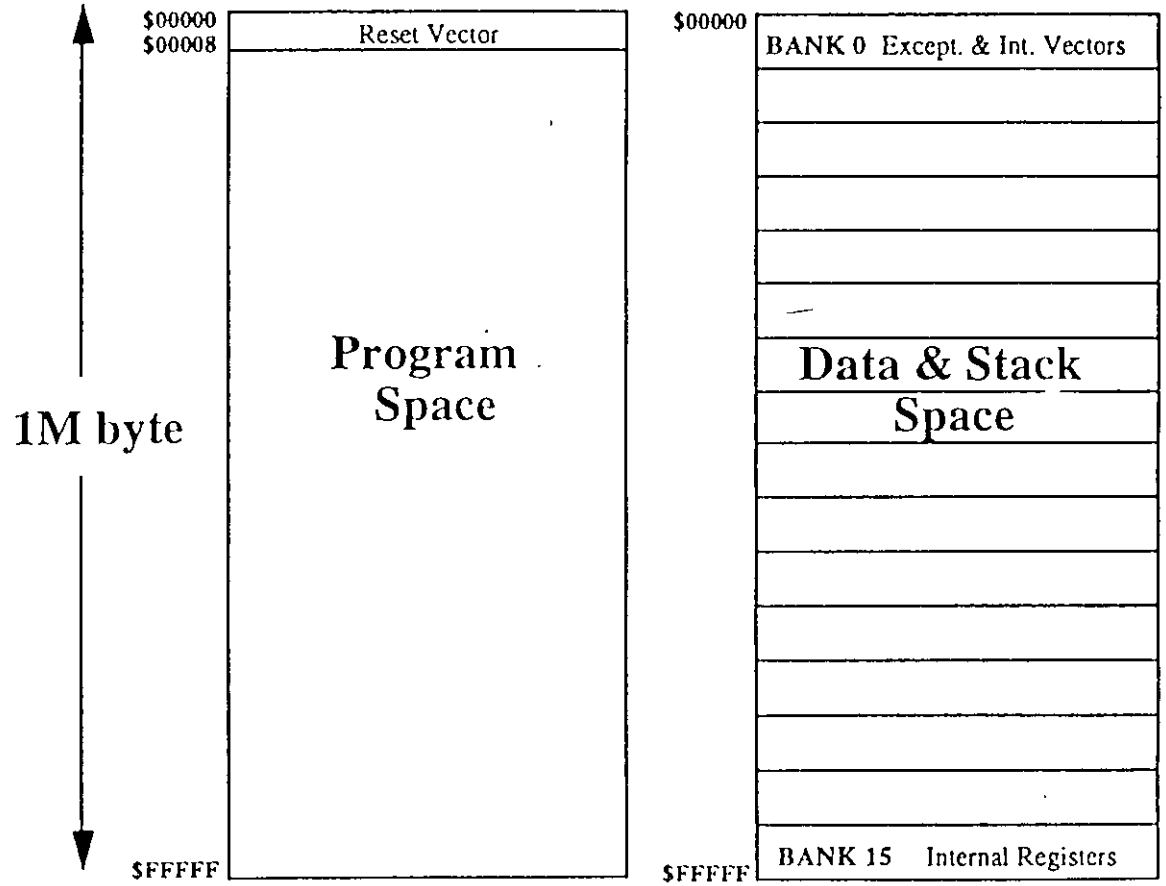


OPCIONES DE MAPA DE MEMORIA

ESPACIO DE PROGRAMA Y DATOS COMBINADOS
FC0 & FC1 no decodificadas



ESPACIO DE PROGRAMA Y DATOS SEPARADOS
FC0 & FC1 decodificadas



usa una forma simplificada del modo de direccionamiento por módulo para implementar filtros de respuesta finita a impulso (FIR) y buffers circulares durante la ejecución de las instrucciones MAC y MAC repetitiva. El CPU16 realza muchos de los modos de direccionamiento poseídos por el 68HC11 y añade varios modos de direccionamiento especializados.

Modos de Direccionamiento.

Inherente
Inmediato
Extendido
Indexado
Desplazamiento de acumulador
Relativo
Indexado postmodificado (MOVB/MOVW)
Modulo (MAC/RMAC).

Modo de Direccionamiento Inherente
Algunas instrucciones con este modo: CLRA,TSX, RTS

| PREBYTE | COD OP |

CLRA
TSX
RTS

La instrucción está completamente especificada por el código de operación y no requiere más operandos de información.

Modo de Direccionamiento Inmediato.
Algunas instrucciones con este modo: LDAA #$FF, LDX #$1250

| COD OP | DATO INM |

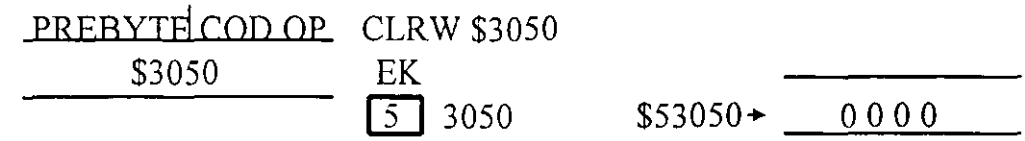
LDAA #$FF

PREBYTE	COD OP
DATO INM	

LDX #$1250

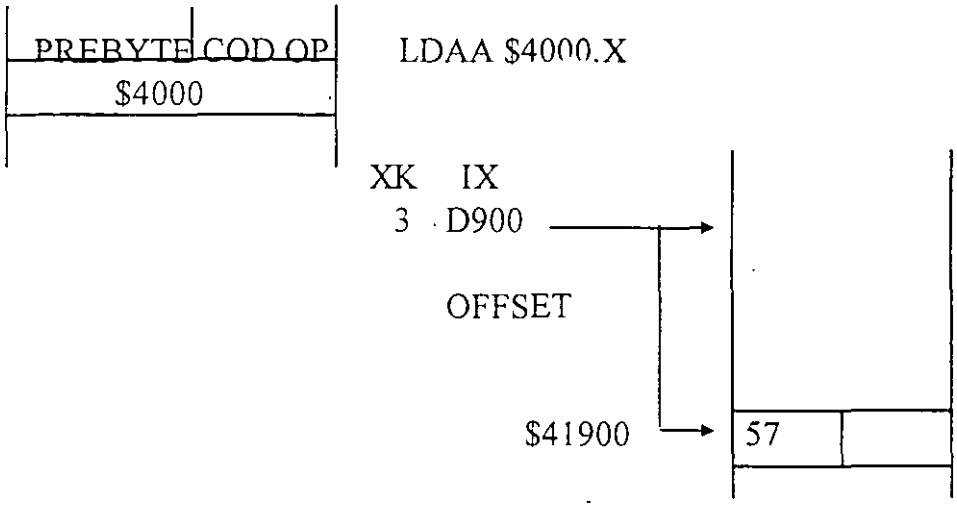
El operando del dato inmediato está dentro del flujo de la instrucción.

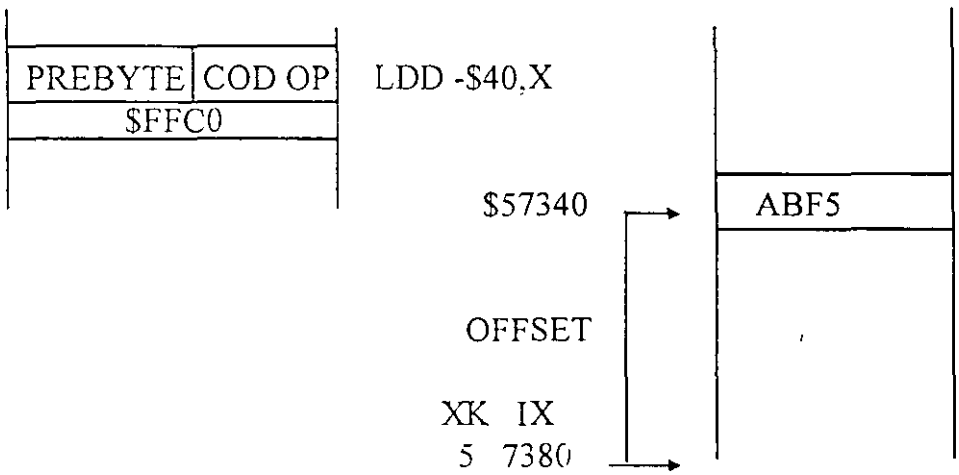
Modo de Direccionamiento Extendido.
Algunas instrucciones con este modo.



Los 16 bits menos significativos del operando de dirección están dentro del flujo de la instrucción; los 4 bits más significativos están en EK.

Modo de Direccionamiento Indexado (1/3)

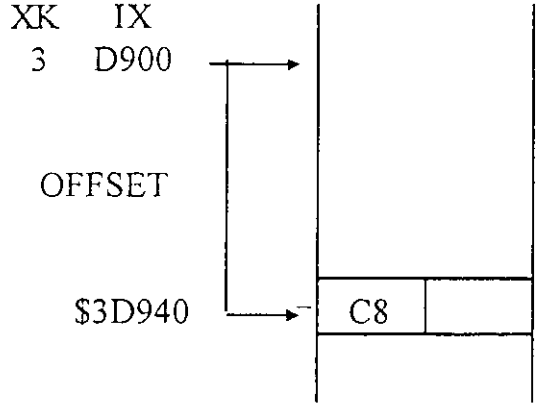




offset de 16-bit signado.
Los registros X,Y, o Z pueden ser usados para indexar.
Las fronteras de un banco pueden ser cruzadas, pero el registro K no es afectado.

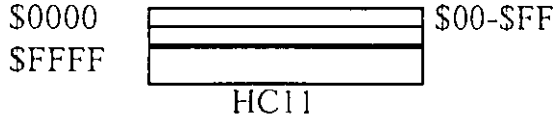
Modo de Direccionamiento indexado (2/3).

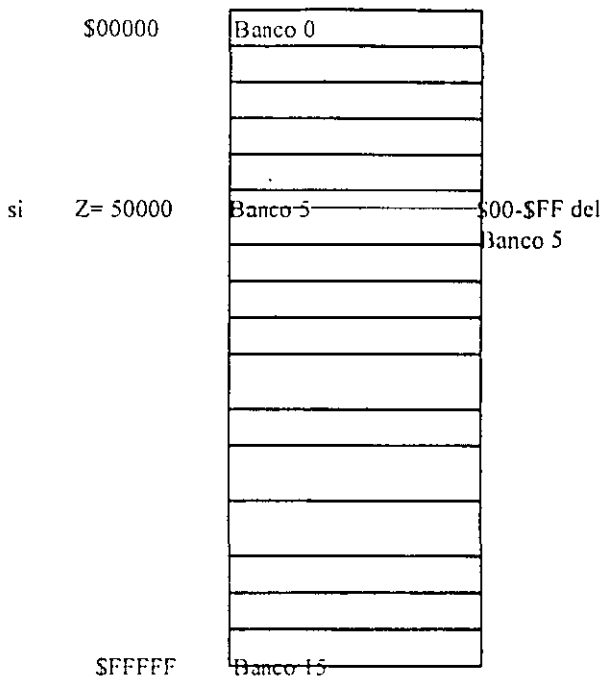
COD OP	$40	LDAA $40,X



Desplazamiento no signado de 8-bit (más eficiente que el offset de 16-bit)
Los registros X,Y, o Z pueden ser usados para indexar.
Las fronteras de un banco pueden ser cruzadas, pero el registro K no es afectado.

Modo de Direccionamiento Indexado (3).
68hc11: Modo de Direccionamiento Directo ($00-$FF) para variables frecuentemente usadas.
Ej.: LDAA $70
68HC16: Modo de Direccionamiento Directo, reemplazado con un modo de direccionamiento indexado en Z con un desplazamiento no signado de 8 BIT ($00-$FF rango del desplazamiento).
Ej.: LDAA $70,Z



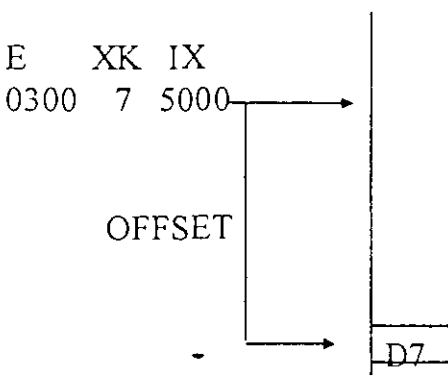


HC16

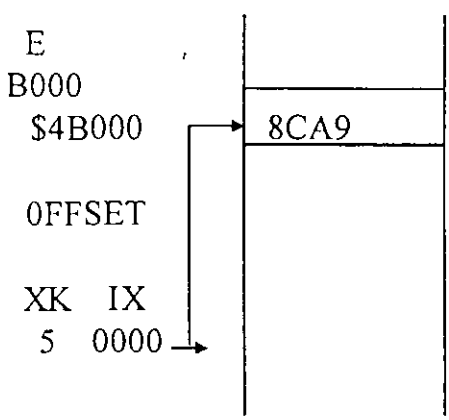
ZK:IZ inicializado durante el reset desde la tabla de vectores

Modo de Direccionamiento por Desplazamiento de Acumulador.

LDAA E,X



LDAA E,X



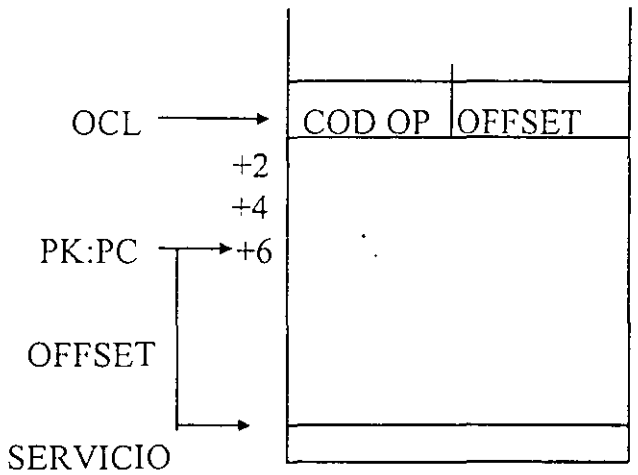
El desplazamiento en E es signado

Los registros X, Y, o Z pueden ser usados para indexar.

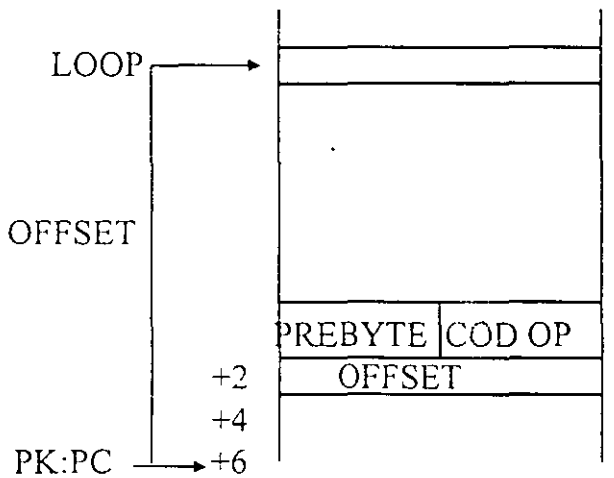
Las fronteras de un Banco pueden ser cruzadas, pero EK no es afectado.

Direccionamiento Relativo.
Únicamente para instrucciones de bifurcación.

BGT SERVICIO



LBEQ LOOP

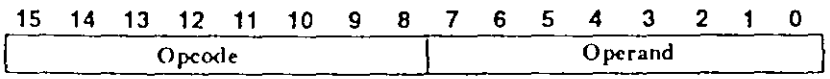


PK:PC esta +6 localidades de OCL (debido a la prebusqueda)
Bifurcaciones Cortas tienen un desplazamiento de 8-bit.
El rango es de -128 +126 desde PK:PC
Bifurcaciones Largas tienen un desplazamiento de 16-bit
El rango es de -32k +32k desde PK:PC

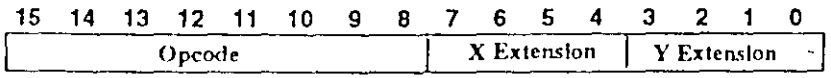
CONJUNTO DE INSTRUCCIONES

FORMATO DE INSTRUCCIONES

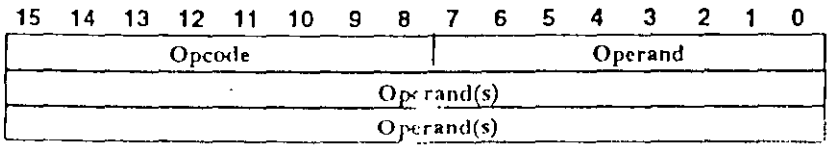
8-Bit Opcode with 8-Bit Operand



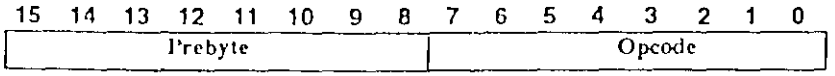
8-Bit Opcode with 4-Bit Index Extensions



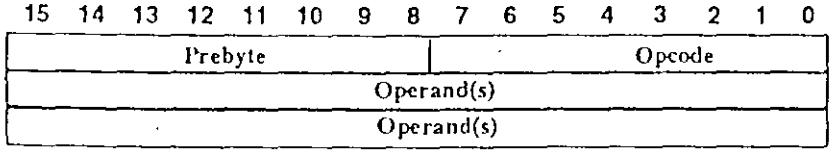
8-Bit Opcode, Arguments(s)



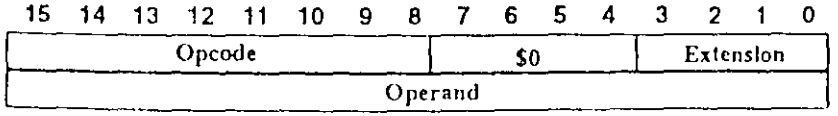
8-Bit Opcode with 8-Bit Prebyte, No Argument



8-Bit Opcode with 8-Bit Prebyte, Argument(s)



8-Bit Opcode with 20-Bit Argument



INSTRUCCIONES DE MANIPULACION DE DATOS (LOADS and STORES)

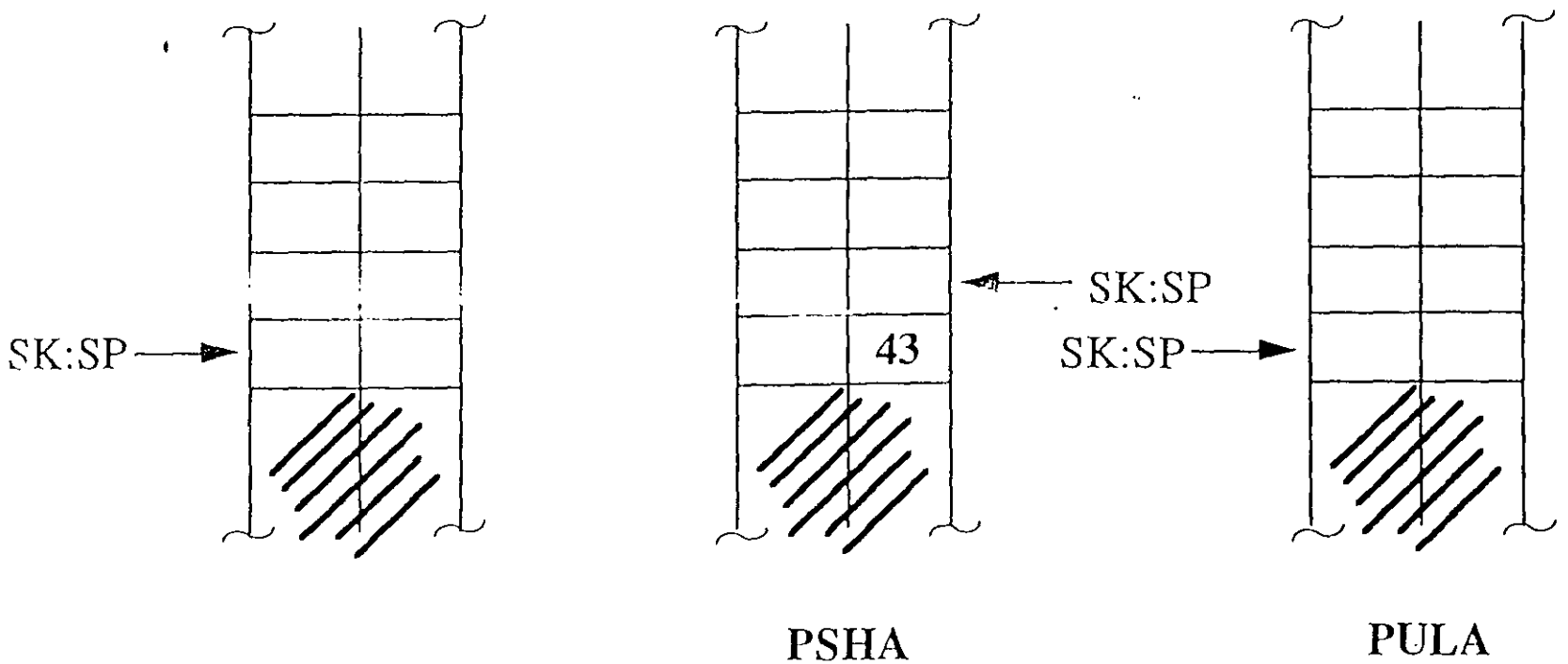
FUNCTION	MNEMONIC	OPERATION
LOAD 8-BIT REG	LDAA LDAB	(M) → A (M) → B
LOAD 16-BIT REG	LDD LDE LDS LDX LDY LDZ	(M: M+1) → R
LOAD 2 16-BIT REGS	LDED	(M: M+1) → E (M + 2: M + 3) → D
STORE 8-BIT REG	STAA STAB	A → (M) B → (M)
STORE 16-BIT REG	STD STE STS STX STY STZ	R → (M: M+1)
STORE 2 16-BIT REGS	STED	E → (M: M+1) D → (M + 2: M + 3)

INSTRUCCIONES DE MANIPULACION DE DATOS (**PUSHES and PULLS**)

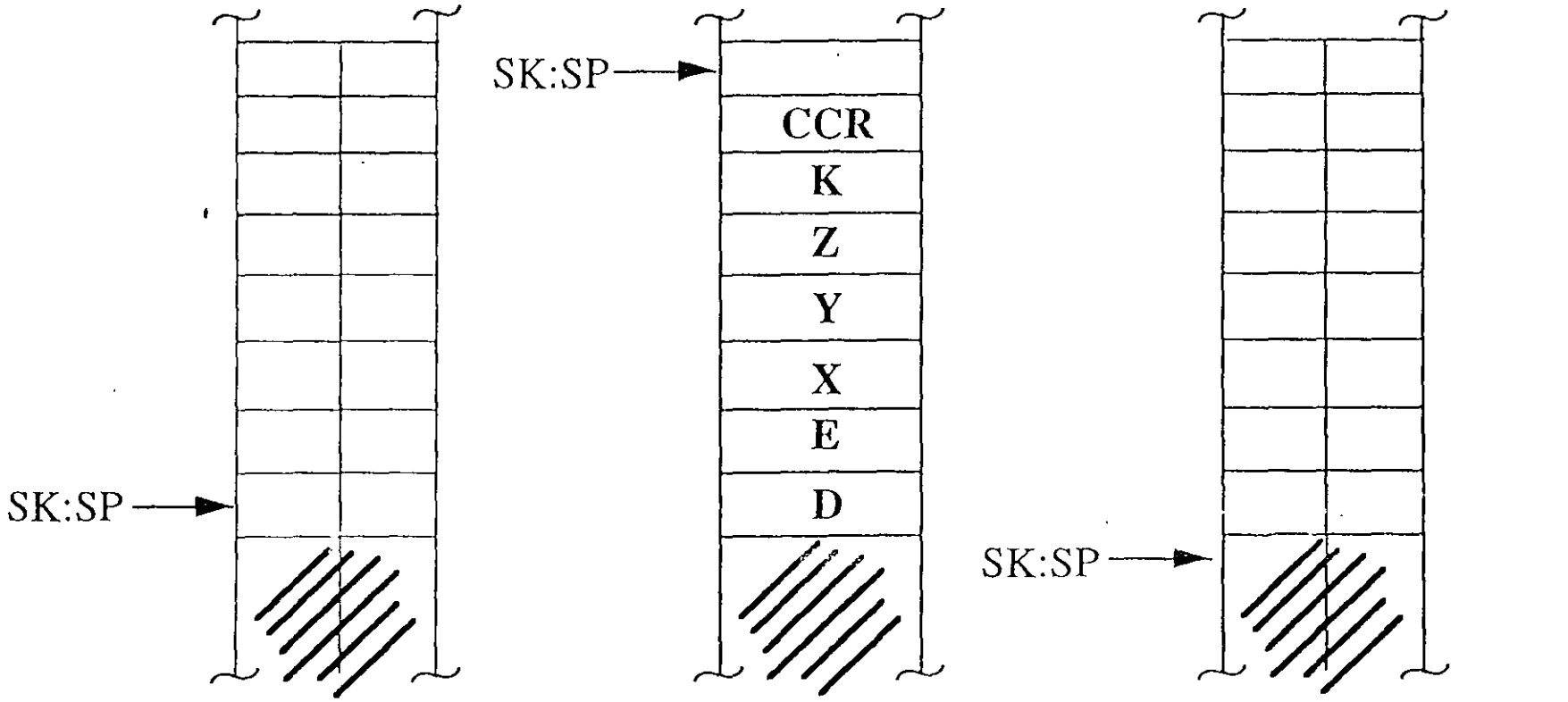
FUNCTION	MNEMONIC	OPERATION
PUSH 8 BIT REG	PSHA PSHB	(SK:SP) + 1 → SK:SP PUSH (R) (SK:SP) - 2 → SK:SP
PUSH MULTIPLE REGS.	PSHM <mask> 0=D 4=IZ 1=E 5=K 2=IX 6=CCR 3=IY 7=(reserved)	For mask bits 0 to 7: if mask bit set push register (SK:SP) - 2 SK:SP
PULL 8 BIT REG	PULA PULB	(SK:SP) + 2 → SK:SP PULL (R) (SK:SP) - 1 → SK:SP
PULL MULTIPLE REGS.	PULM <mask> 0=CCR 4=IX 1=K 5=E 2=IZ 6=D 3=IY 7=(reserved)	For mask bits 0 to 7: if mask bit set (SK:SP) + 2 SK:SP pull register

Nota: El meter (pushes) o sacar (pulls) 8 bits pueden dejar al apuntador de pila (stack pointer) en una dirección impar y dejar desalineado el acceso a datos

PSHA/B AND PULA/B



PUSH/ PULL MULTIPLE



PSHM D,E,X,Y,Z,K,CCR PULM D,E,X,Y,Z,K,CCR

Nota: Siempre meta (pushed) en este orden
Puede tener cualquier combinación de registros en máscara
El tiempo de instrucción depende en el número de registros en máscara

INSTRUCCIONES DE MANIPULACION DE DATOS (TRANSFERS)

FUNCTION	MNEMONIC	OPERATION
TRANSFER TO 8 BIT ACCUMULATOR	TBA TAB	B → A A → B
TRANSFER TO 16 BIT ACCUMULATOR	TDE TED	D → D E → D
TRANSFER B TO K REGISTER	TBXK TBYK TBZK TBSK TBEK	$B_{3\,0}$ → K reg
TRANSFER K REGISTER TO B	TXKB TYKB TZKB TSKB TEKB	K reg → $B_{3:0}$ 0000 → $B_{7:4}$

INSTRUCCIONES DE MANIPULACION DE DATOS (**TRANSFERS**)

FUNCTION	MNEMONIC	OPERATION
TRANSFER A / CCR	TAP TPA	A → CCR 15:8 CCR15:8 → A
TRANSFER D / CCR	TDP TPD	D → CCR 15:4 CCR → D
TRANSFER INDEX REGISTER TO INDEX REGISTER	TXY TXZ TYX TYZ TZX TZY	eg. TXY XK:IX → YK:IY
TRANSFER INDEX REGISTER TO STACK POINTER	TXS TYS TZS	eg. TXS (XK:IX) - 2 → SK:SP
TRANSFER STACK POINTER TO INDEX REGISTER	TSX TSY TSZ	eg. TSX (SK:SP) + 2 → XK:IX

INSTRUCCIONES DE MANIPULACION DE DATOS (EXCHANGES)

FUNCTION	MNEMONIC	OPERATION
EXCHANGE 8 BIT ACCUMULATORS	XGAB	A ◄► B
EXCHANGE 16 BIT ACCUMULATORS	XGDE	D ◄► E
EXCHANGE 16 BIT ACCUMULATOR WITH INDEX REGISTER	XGDX XGDY XGDZ XGEX XGEY XGEZ	D ◄► R E ◄► R

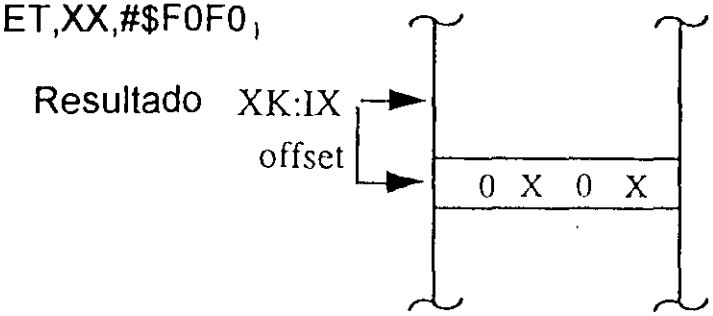
INSTRUCCIONES DE MANIPULACION DE DATOS (ALTER DATA)

FUNCTION	MNEMONIC	OPERATION
DECREMENT	DEC DECW DECA DECB	(M)-1 → (M) (M:M+1) -1→ (M:M+1) A-1 → A B-1 → B
INCREMENT	INC INCW INCA INCB	(M)+1 → (M) (M:M+1) +1→ (M:M+1) A+1 → A B+1 → B
COMPLEMENT,2'S (NEGATE)	NEG NEGW NEGA NEGB NEGD NEGE	0-(M) → (M) 0-(M:M+1)→ (M:M+1) 0-A → A 0-B → B 0-D → D 0-E → E
COMPLEMENT,1'S	COM COMW COMA COMB COMD COME	$(\overline{M})$ → (M) $(\overline{M:M+1})$ → (M:M+1) $\overline{A}$ → A $\overline{B}$ → B $\overline{D}$ → D $\overline{E}$ → E
SIGN EXTEND B into A	SXT	If $B_7 = 1$, A = $FF If $B_7 = 0$, A = $00

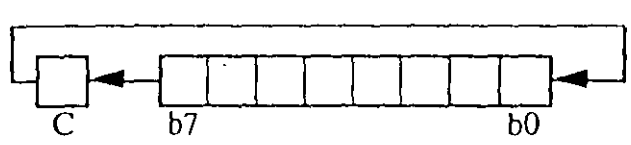
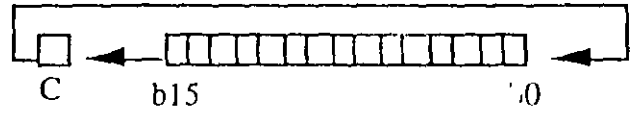
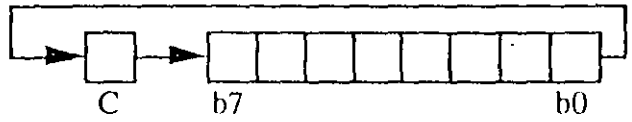
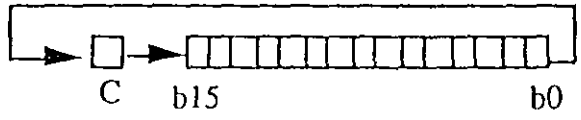
INSTRUCCIONES DE MANIPULACION DE DATOS (ALTER DATA)

FUNCTION	MNEMONIC	OPERATION
CLEAR	CLR CLRW CLRA CLRB CLRD CLRE	0 → (M) 0 → (M:M+1) 0 → A 0 → B 0 → D 0 → E
BIT(S) CLEAR	BCLR BCLRW	(M) • $\overline{\text{MASK}}$ → (M) (M:M+1) • $\overline{\text{MASK}}$ → (M:M+1)
BIT(S) SET	BSET BSETW	(M) + MASK → (M) (M:M+1) + MASK → (M:M+1)

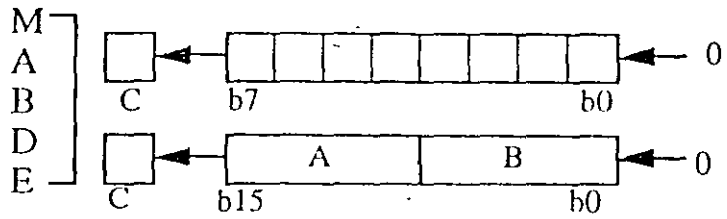
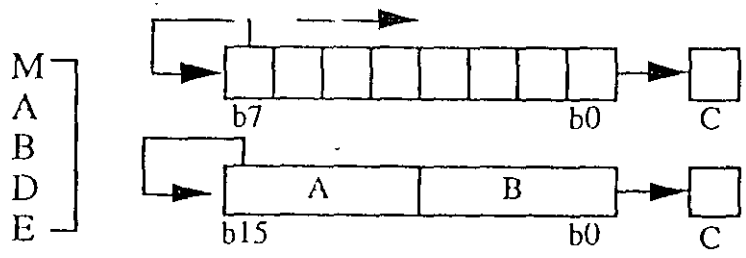
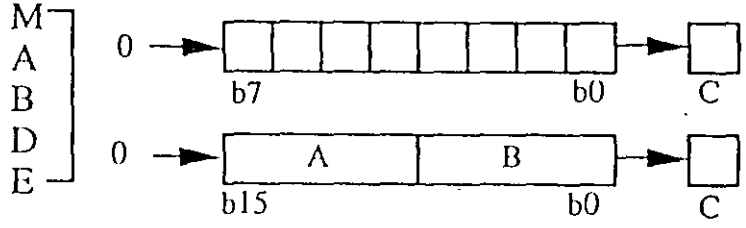
Ejemplo de Manipulación de Bit: BCLRW OFFSET,XX,#$F0F0



INSTRUCCIONES DE MANIPLACION DE DATOS (**ROTATES**)

FUNCTION	MNEMONIC	OPERATION
ROTATE BYTE LEFT	ROL ROLA ROLB	M A B 
ROTATE WORD LEFT	ROLW ROLD ROLE	M D E 
ROTATE BYTE RIGHT	ROR RORA RORB	M A B 
ROTATE WORD RIGHT	RORW RORD RORE	M D E 

INSTRUCCIONES DE MANIPULACION DE DATOS (SHIFTS)

FUNCTION	MNEMONIC	OPERATION
SHIFT LEFT, ARITHMETIC (LOGICAL)	ASL(LSL) ASLW(LSLW) ASLA(LSLA) ASLB(LSLB) ASLD(LSLD) ASLE(LSLE)	
SHIFT RIGHT, ARITHMETIC	ASR ASRW ASRA ASRB ASRD ASRE	
SHIFT RIGHT, LOGICAL	LSR LSRW LSRA LSRB LSRD LSRE	

INSTRUCCIONES ARITMETICAS
(ADD TO ACCUMULATORS)

FUNCTION	MNEMONIC	OPERATION
ADD	ADDA ADDB ADDD ADDE	A + (M) → A B + (M) → B D + (M:M+1) → D E + (M:M+1) → E
ADD WITH CARRY	ADCA ADCB ADCD ADCE	A + (M) + C → A B + (M) + C → B D + (M:M+1) + C → D E + (M:M+1) + C → E
ADD ACCUMULATORS	ABA ADE	A + B → A D + E → E
DECIMAL ADJUST A	DAA	Perform after ADDA, ADCA, or ABA: converts A to base 10

INSTRUCCIONES ARITMETICAS
(ADD TO INDEX REGISTERS)

FUNCTION	MNEMONIC	OPERATION
ADD B TO INDEX REGISTER	ABX ABY ABZ	000:B + XK:IX → XK:IX 000:B + YK:IY → YK:IY 000:B + ZK:IZ → ZK:IZ
ADD D/E TO INDEX REGISTER	ADX AEX ADY AEY ADZ AEZ	<<R + XK:IX → XK:IX <<R + YK:IY → YK:IY <<R + ZK:IZ → ZK:IZ
ADD IMMEDIATE TO INDEX REGISTER / STACK POINTER	AIX AIY AIZ AIS	<<IMM + XK:IX → XK:IX <<IMM + YK:IY → YK:IY <<IMM + ZK:IZ → ZK:IZ <<IMM + SK:IS → SK:SP

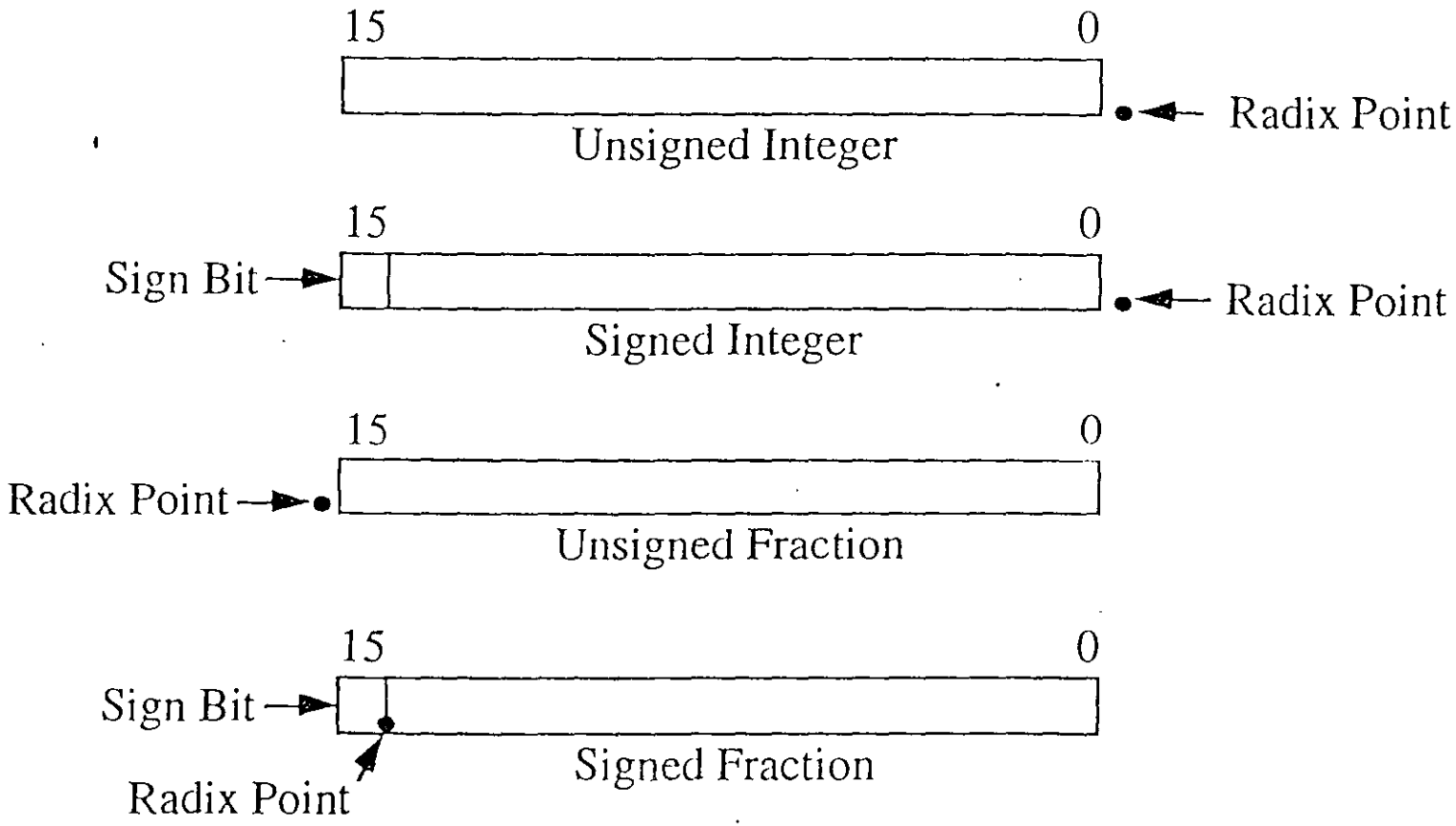
INSTRUCCIONES ARITMETICAS
(SUBTRACT FROM ACCUMULATORS)

FUNCTION	MNEMONIC	OPERATION
SUBTRACT	SUBA SUBB SUBD SUBE	A - (M) → A B - (M) → B D - (M:M+1) → D E - (M:M+1) → E
SUBTRACT WITH CARRY	SBCA SBCB SBCD SBCE	A - (M) - C → A B - (M) - C → B D - (M:M+1) - C → D E - (M:M+1) - C → E
SUBTRACT ACCUMULATORS	SBA SDE	A - B → A E - D → E

INSTRUCCIONES ARITMETICAS
(MULTIPLY & DIVIDE)

FUNCTION	MNEMONIC	OPERATION	OVERFLOW	DIV BY 0
INTEGER MULTIPLY	MUL (unsigned) EMUL (unsigned) EMULS (signed)	A * B → D E * D → E:D	N/A	N/A
FRACTIONAL MULTIPLY	FMULS (signed)	E * D → E:$D_{31:1}$ 0 → D_0		
INTEGER DIVIDE	IDIV (unsigned)	D/IX → IX Remainder → D	N/A	C=1 IX = $FFFF D = Undefined
	EDIV (unsigned) EDIVS (signed)	E:D/IX → IX Remainder → D	V=1 Accumulators left unchanged	Exception Processing; Accumulators left unchanged
FRACTIONAL DIVIDE	FDIV (unsigned)	D/IX → IX Remainder → D	V=1 IX = $FFFF D = Undefined	C=1 IX = $FFFF D = Undefined

TIPOS DE DATOS



INSTRUCCIONES LOGICAS

FUNCTION	MNEMONIC	OPERATION
AND mask to ACCUMULATOR	ANDA ANDB ANDD ANDE	A • (M) → A B • (M) → B D • (M:M+1) → D E • (M:M+1) → E
AND mask to CCR	ANDP	CCR • IMM16 → CCR*
INCLUSIVE OR mask to ACCUMULATOR	ORAA ORAB ORD ORE	A + (M) → A B + (M) → B D + (M:M+1) → D E + (M:M+1) → E
OR mask to CCR	ORP	CCR + IMM16 → CCR*
EXCLUSIVE OR mask to ACCUMULATOR	EORA EORB EORD EORE	A ⊕ (M) → A B ⊕ (M) → B E ⊕ (M) → D E ⊕ (M) → E

* PK field (CCR 3:0) not affected

INSTRUCCIONES DE PRUEBA DE DATO

FUNCTION	MNEMONIC	TEST
BIT TEST	BITA BITB	A • (M) B • (M)
COMPARE	CBA CMPA CMPB	A-B A-(M) B-(M)
	CPD CPE CPS CPX CPY CPZ	R - (M:M+1)
TEST for ZERO OR MINUS	TST TSTW TSTA TSTB TSTD TSTE	(M)-0 (M:M+1) -0 A-0 B-0 D-0 E-0

INSTRUCCIONES DE BIFURCACION

FUNCTION	MNEMONIC	OPERATION
NO OPERATION	NOP	PK:PC advances to next instruction
Non-conditional BRANCHES	BRA LBRA	PK:PC + offset → PK:PC
	BRN LBRN	PK:PC advances to next instruction

Conditional BRANCHES	B<cc>
	LB<cc>
		If <cc> then
		PK:PC + offset → PK:PC
		else
		PK:PC advances to next instruction

	BRSET
	
	eg.
	BRSET $10,X $40 SERVICE
		If $(\overline{M})$ • mask = 0 (ie. if bits specified by mask are set) then
		PK:PC + offset → PK:PC
		else
		PK:PC advances to next instruction

	BRCLR
	
	eg.
	WAIT BRCLR $2000 $40 WAIT
		If (M) • mask = 0 (ie. if bits specified by mask are clear) then
		PK:PC + offset → PK:PC
		else
		PK:PC advances to next instruction

Nota: Saltos Largos usan desplazamientos signados de 16-bit
otros usan desplazamientos signados de 8-bit.

INSTRUCCIONES DE BIFURCACIONES CONDICIONADAS

MNEMONIC	CONDITION	CCR TEST	INDICATION
(L)BMI	MINUS	N=1	r=NEGATIVE
(L)BPL	PLUS	N=0	r=POSITIVE
* (L)BVS	OVERFLOW	V=1	r=SIGN ERROR
* (L)BVC	NO OVERFLOW	V=0	r=SIGN OK
* (L)BLT	LESS	$[N \oplus V]=1$	A < M
* (L)BGE	GREATER OR EQUAL	$[N \oplus V]=0$	A >= M
* (L)BLE	LESS OR EQUAL	$[Z+(N \oplus V)]=1$	A <= M
* (L)BGT	GREATER	$[Z+(N \oplus V)]=0$	A > M
(L)BEQ	EQUAL	Z=1	A=M
(L)BNE	NOT EQUAL	Z=0	A <> M
(L)BHI	HIGHER	[C+Z]=0	A > M
(L)BLS	LOWER OR SAME	[C+Z]=1	A <= M
(L)BCC (BHS)	CARRY CLEAR	C=0	A >= M
(L)BCS (BLO)	CARRY SET	C=1	A < M

La indicación se refiere a el uso de una Instrucción CMPA M inmediatamente antes de la bifurcación

* Usadas en aritmética signada únicamente

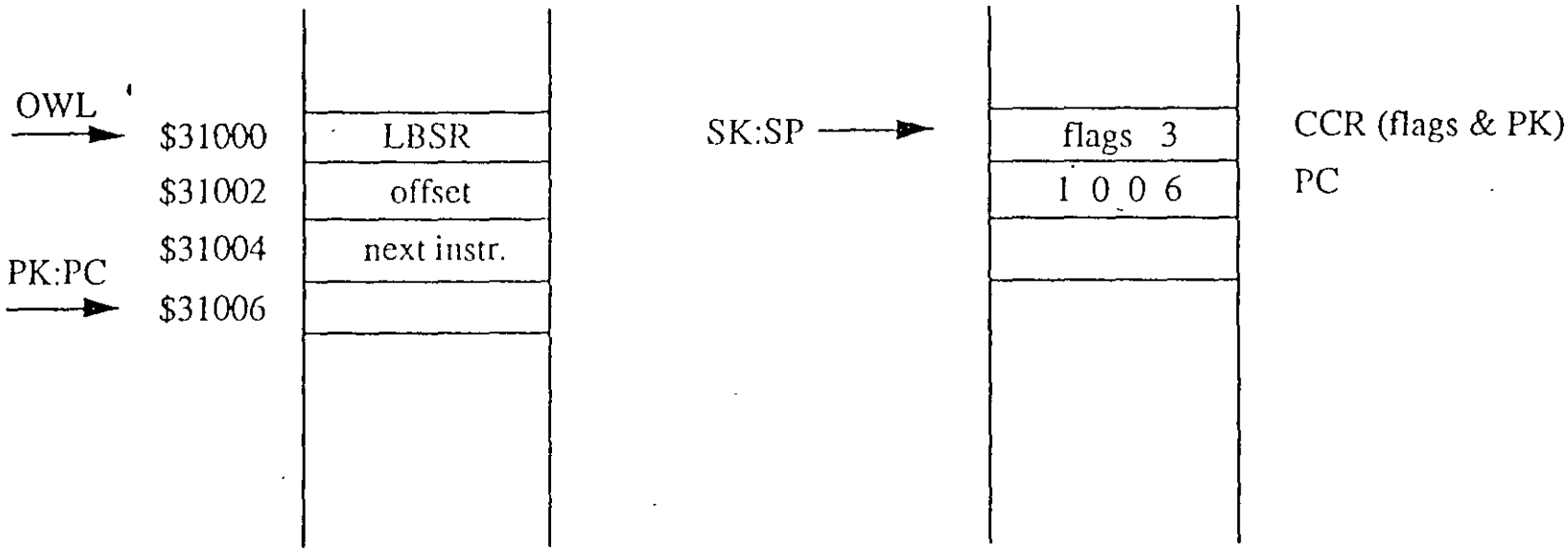
INSTRUCCIONES DE SALTO A SUBRUTINA

FUNCTION	MNEMONIC	OPERATION
BRANCH to SUBROUTINE	BSR	PK:PC - 2 → PK:PC PC, CCR pushed on stack PK:PC + offset → PK:PC
	LBSR	PC, CCR pushed on stack PK:PC + offset → PK:PC
RETURN from SUBROUTINE	RTS	Pull PK Pull PC PK:PC - 2 → PK:PC

Nota: Saltos Largos usan desplazamiento signado de 16-bit; otros usan desplazamiento signado de 8-bit

RTS

Debido al pipeline, PK:PC= Localidad Palabra de Operación (OWL)+6



RTS restaura PK:PC-2 para ejecutar la siguiente instrucción en $31004

Las banderas no son restauradas

Nota:LBSR JSR son instrucciones de 2 palabras, BSR es una instrucción de una palabra; que hace que RTS apunte a la siguiente instrucción después de BSR, BSR resta 2 de PK:PC antes de operar la pila

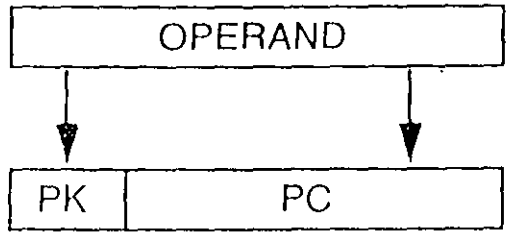
JMP / JSR

FUNCTION	MNEMONIC	OPERATION
'JUMP	JMP	<ea> ⟶ PK:PC
JUMP TO SUBROUTINE	JSR	Push CCR, PC <ea> ⟶ PK:PC

EXTENDED 20

JSR $35A02

EA = $35A02

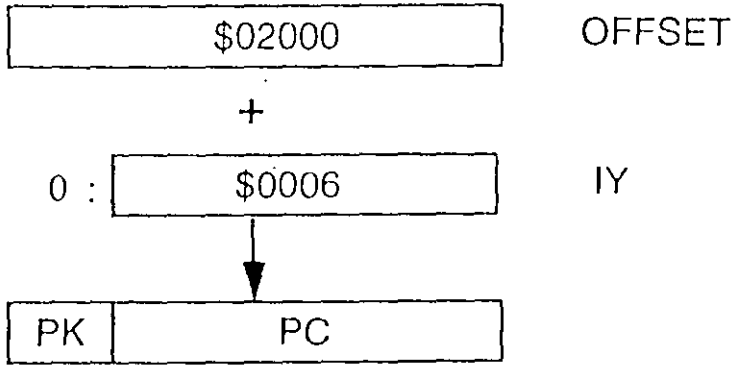


INDEXED with 20-bit Offset

YK:IY | $5 | $0006

JMP $02000,Y

PK:PC = $02006 (not $52006)



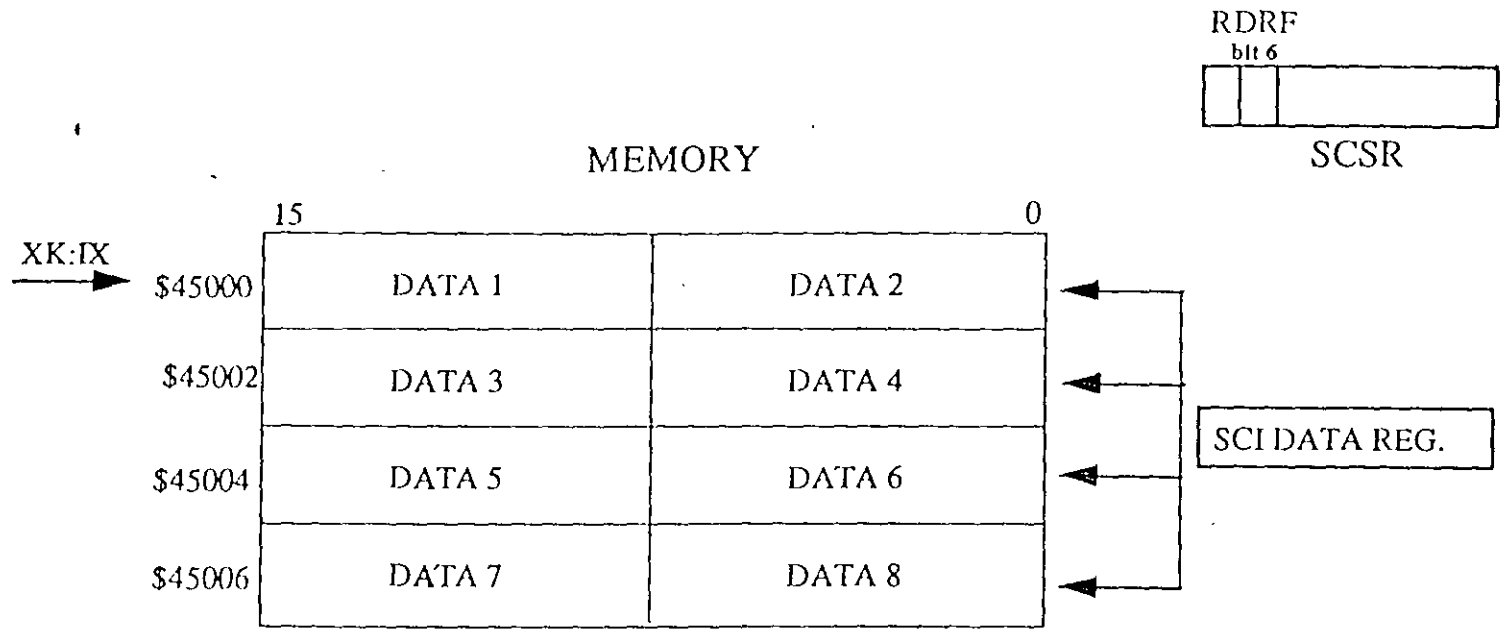
INSTRUCCIONES DE MOVIMIENTO

FUNCTION	MNEMONIC	OPERATION
MOVE BYTE	MOVB	<M> ⟶ <M>
MOVE WORD	MOVW	<M:M+1> ⟶ <M:M+1>

-MOVB/MOVW permiten PMI,EXT; EXT,PMI;EXT,EXT
-PMI siempre es usada en conjunto con el registro IX.
-El valor incrementado es cualquier desplazamiento signado de 8-bit.

eje. MOVB $1000,1,X
MOVB -2,X, LABEL
MOVW LABEL1,LABEL2

INSTRUCCION MOVB CON INDICE POST MODIFICADO (PMI)



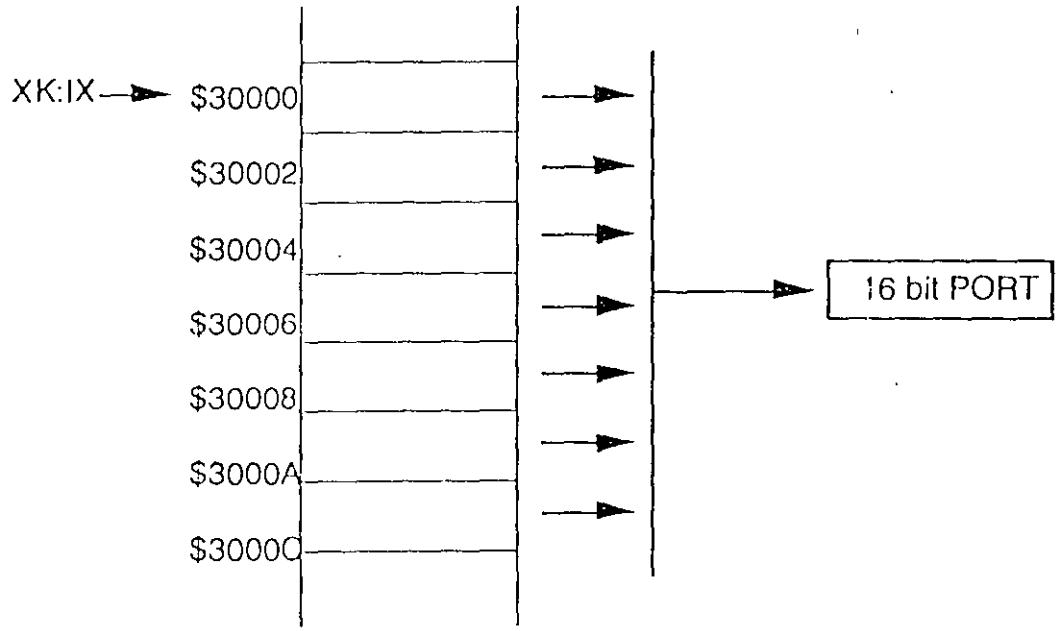
Nota: RDRF puede ser configurada para que cause una interrupción

```
Nextchar  BRCLR   SCSR,#$40,Nextchar   ; se checa la bandera RDRF para ver si el receptor está lleno
          MOVB    SCDR,1,X            ; mueve dato a donde apunta X, incrementa X en 1
          LDAA    SCDR                ;mueve dato al acumulador A
          CMPA    #End_Pattern        ; checa fin de dato desde el SCI
          BNE     Nextchar            ; regresa si hay más datos
```

Nota: RDRF puede ser configurada para que cause una interrupción

INSTRUCCION MOVW CON INDICE POST MODIFICADO (PMI)

-PMI siempre es usada en conjunto con el registro X
-El valor del incremento es cualquier desplazamiento de 8-bit signado




```
        LDAB    #3          ; inicializa XK:IX como $30000
        TBXK
        LDX     #0
Output  MOVW    2,X,PORT    ; mueve una palabra y post incrementa IX en dos bytes
        CPX     #EndAddr    ; checa dirección final
        BNE     Output      ; regresa si no es final
```

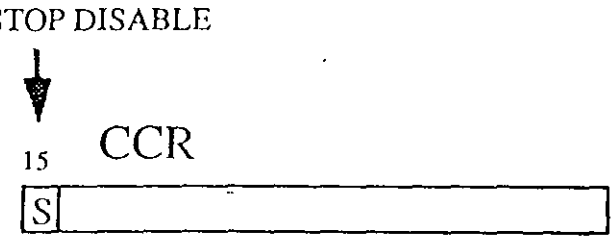
LPSTOP

LPSTOP DISABLE

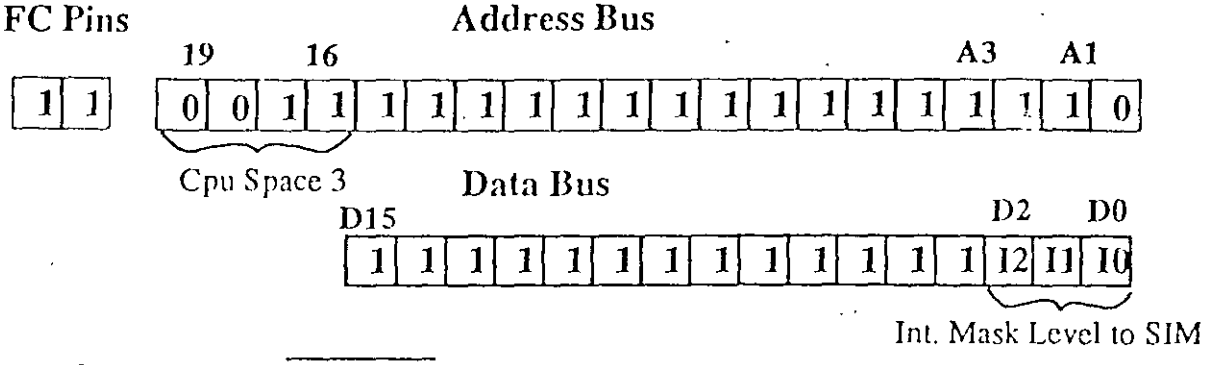
IF S= 1

LPSTOP = 4 cycle NOP

ELSE



A) Ciclo de emisión de LPSTOP



SIM provides $\overline{\text{DSACKx}}$ response

B) El reloj interno es detenido

RESET or INTERRUPT > IP Mask exits LPSTOP mode

Notas:

PTI no es automáticamente deshabilitada durante LPSTOP

Z1: Corriente de corrida= 100ma; LPSTOP: con/VCO apagado=350 con/VCO encendido=5 mA.

WAI

Reposa al CPU

Reloj interno del CPU es detenido; Los otros modulos continúan trabajando.
RESET o INTERRUPCION -- mascara IP saca del WAI

Comparación LPSTOP/WAIT:
-Reconocimiento de interrupción más rápido después de WAI
-LPSTOP minimiza consumo de potencia del MCU

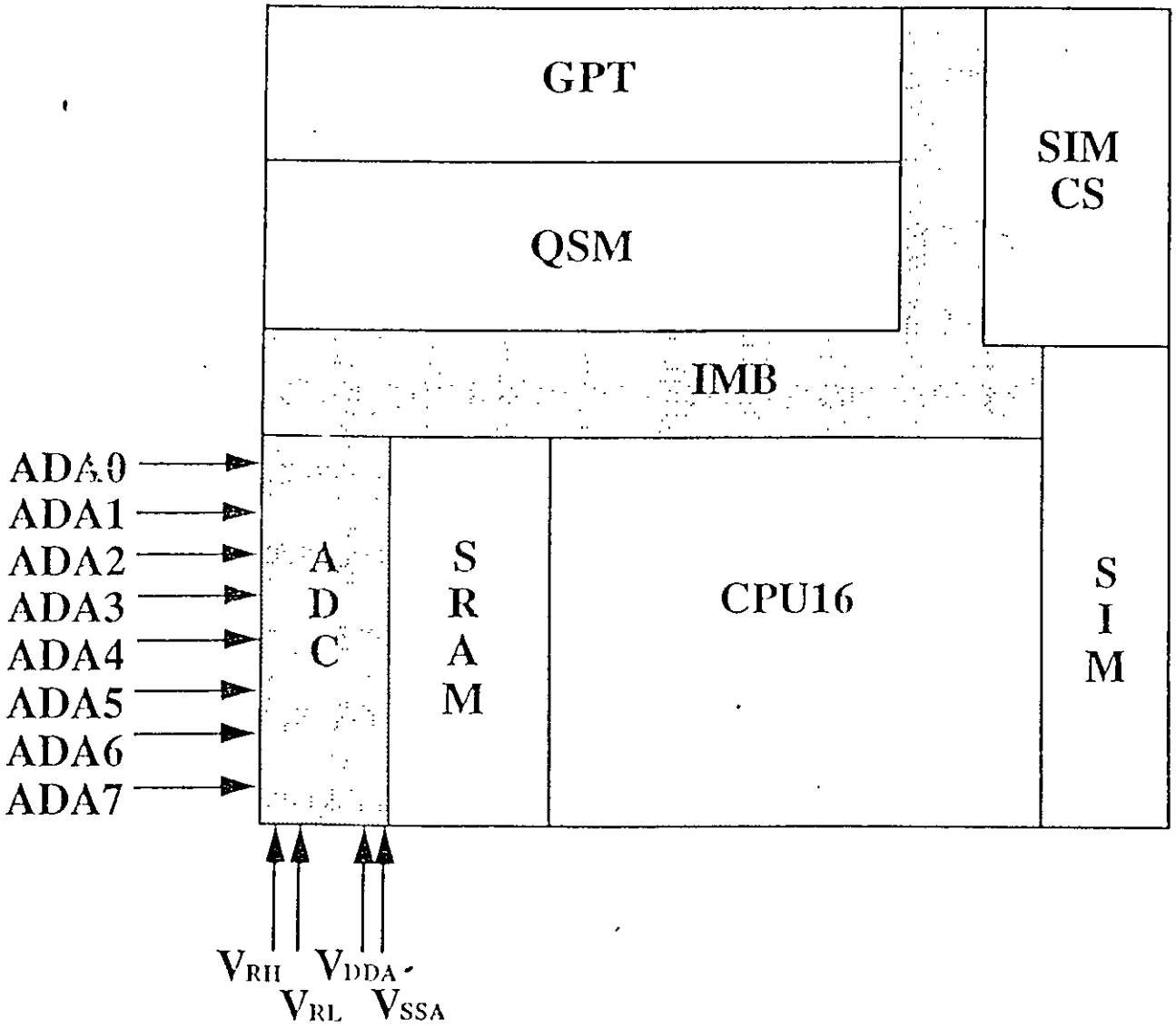
CONJUNTO DE INSTRUCCIONES.

Periféricos Inteligentes.
Algunos de los módulos periféricos inteligentes y característicos que existen en la familia 68HC16 incluyen:
1) Convertidos Analógico/Digital con ocho canales y 10-bit de resolución
2) Modulo Serial Encolado (QSM)
3) Unidad de Procesamiento de Tiempo (TPU)
4) Módulo de Integración de Sistema (SIM)

Interfase Analógica.
El Convertidor Analógico Digital (ADC) provee al 68HC16 con una interfase analógica. Este módulo es un convertidor de aproximaciones sucesivas con ocho modos de operación y resolución seleccionable de 8 o 10-bit proveyendo incrementos base de voltaje de 0.02 o 0.005 volts sobre un voltaje de referencia de 5 volts. El módulo consiste de dos subsistemas, uno analógico y otro digital figura 5. El subsistema analógico consiste de un multiplexor, un amplificador de muestreo de entrada, un convertidor digital-analógico de arreglo resistencia-capacitancia (DAC RC), y un comparador de alta-ganancia. Estos componentes juegan varios roles. El multiplexor permite al usuario seleccionar una de las ocho fuentes de señal internas o externas para la conversión. El amplificador de muestreo provee una fuente de alta impedancia externa desde el circuito interno. El arreglo DAC RC ejecuta dos funciones: actúa como un circuito de muestreo-reten, y provee la comparación de salida digital-analógica necesaria para la conversión de aproximaciones sucesivas. El comparador indica cuando cada salida sucesiva del arreglo DAC RC es mas alta o baja que la entrada muestreada.
El subsistema de control digital contiene registros y lógica para controlar el proceso de conversión. Los registros de control y la lógica asociada seleccionan la resolución de conversión, multiplexan la entrada, muestrean el tiempo y proveen los ciclos de reloj de ADC. Después de la conversión de cada entrada, el subsistema de control digital almacena el resultado, un bit a la vez, en el registro de aproximaciones sucesivas (SAR) y entonces transfiere los resultados a uno de los ocho registros de resultados figura 6.

Entrada/Salida Serie.

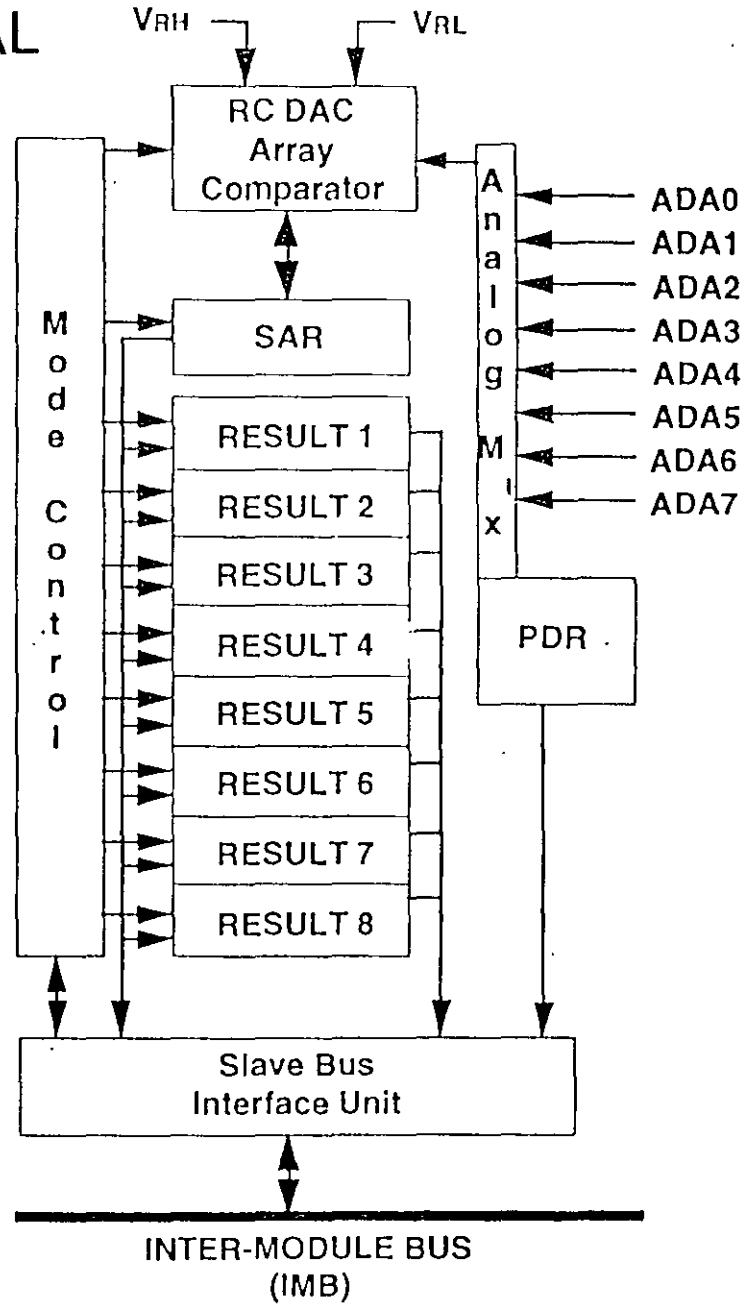
CONVERTIDOR ANALOGICO A DIGITAL ADC



CONVERTIDOR ANALOGICO A DIGITAL ADC

- 8 Canales/ Resolución de 10-Bit
- 8 Registros de Resultado
- 3 Formas de Alineamiento de Dato
  - -Justificado a la derecha
  - -Justificado a la izquierda
  - -Justificado a la izquierda con Signo (para soporte de DSP)
- 8 Modos de Conversión
  - 1 Canal, 4 o 8 veces y Alto
  - 4 o 8 Canales, Una vez y Alto
  - 1 Canal, Continuamente en grupos de 4 o 8
  - 4 o 8 Canales, Continuamente

- Tiempo de Muestreo Programable
  - 4,8,16 o 32 pulsos de reloj
- Tasa de Conversión Rápida
  Basada en un reloj máximo: 2Mz
  8-Bit en 8 seg
  10-Bit en 9 seg

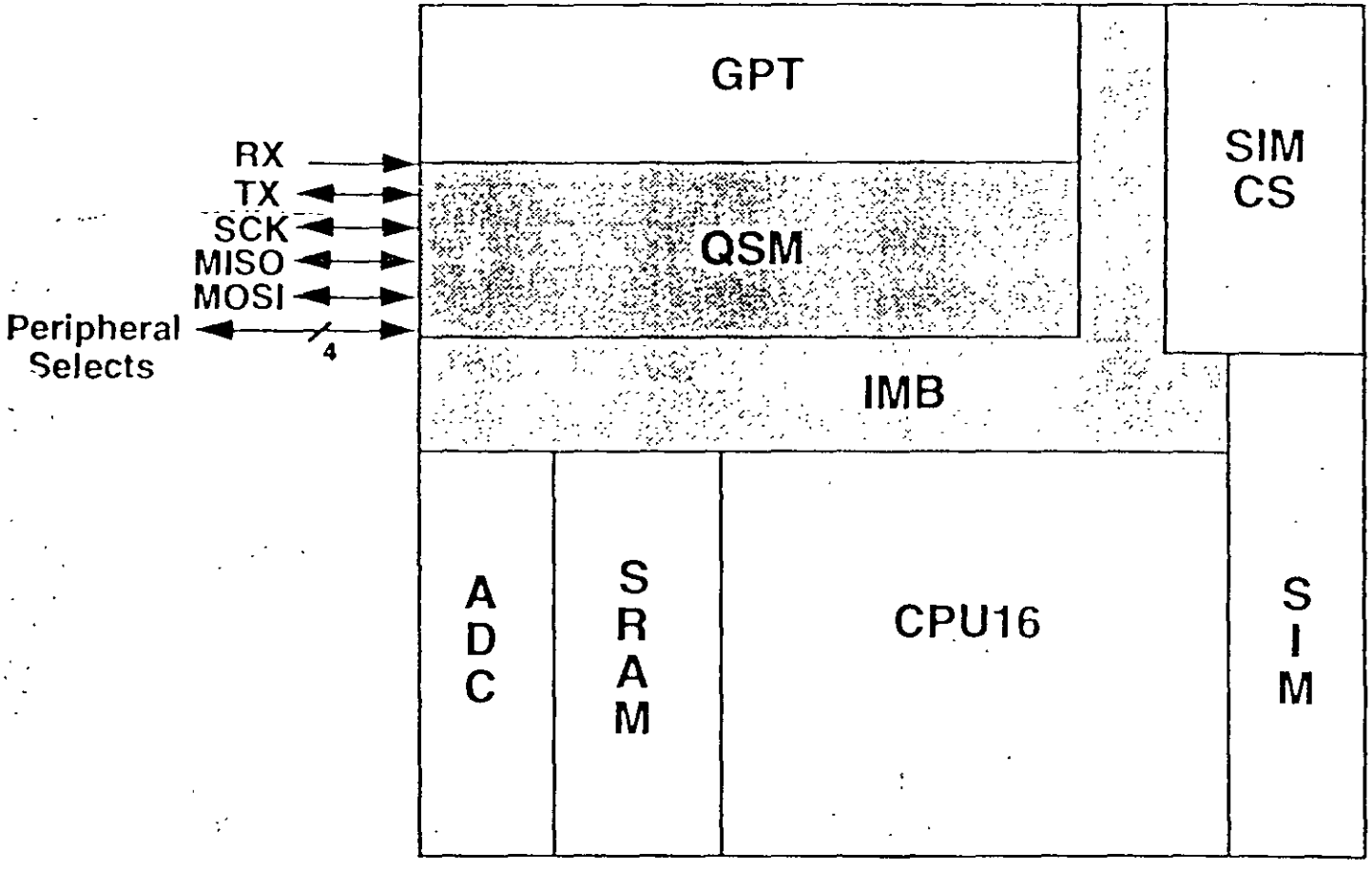


-Las entradas pueden ser usadas como de propósito general
(Port ADA)

El módulo serial encolado (QSM) contiene dos interfaces serie: la interface serie de comunicación (SCI) y la interface serie de periféricos encolada (QSPI) figura 7.
El SCI provee comunicación de entrada/salida a través de un Transmisor Receptor Universal Asíncrono (UART) Serie, comúnmente usado para comunicación RS-232 (RS-232 es una forma muy popular de conectar equipos de comunicación). Es capaz de operar ya sea en modo full o half-duplex con tasas de transferencia de 64 a 524k baud. Los niveles de voltaje de salida del SCI son cero y cinco volts para salidas en alto y bajo. Estos niveles no son los estándar para RS-323 pero son adecuados para comunicarse con otros puertos SCI. Para comunicación con RS-232 se requiere de circuitos de interfase para poder trasladar los niveles de voltaje y así poder comunicarse con dispositivos tales como terminales, plotters, analizadores lógicos, impresoras, módems y otros equipos de comunicación de datos.
Mientras el SCI es una interfase serial asíncrona, el QSPI provee una expansión rápida de periféricos o una comunicación inter-procesadores a través de un bus full-duplex, asíncrono, con cuatro terminales programables de selección de periféricos, lo cual permite direccionar hasta 16 dispositivos periféricos. El QSPI opera hasta 4Mbits/segundo, el cual es más rápido que el SCI debido a la sincronización de la transferencia de datos, la cual elimina bits de sobre encabezado (por eje. bits de dirección, de inicio, de alto los cuales son usados por el SCI). El QSPI también contiene una característica única, una RAM para formar una cola la cual permite una transferencia serie de 8 o 16 bits, o una transmisión de un flujo de datos de 256-bit sin la intervención del CPU16. Un modo especial de enrrollamiento soporta un muestreo continuo de un periférico serial, con actualización automática QSPI RAM, una interfase más eficiente con convertidores A/D, D/A, memorias serie EEPROM y Drivers de Despliegues.

Temporizador.
La unidad de Procesamiento de Tiempo (TPU) es semiautonoma, con 16 canales y módulo coprocesador inteligente. Este contiene un micro código de instrucciones el cual desempeña funciones simples y complejas de tiempo, tales como control de motor de pasos, tiempo de entradas y modulación por ancho de pulso. Puesto que el TPU opera desde su propio micro código este provee funciones de tiempo de alto desempeñó sin la asistencia de un núcleo procesador RISC. El TPU ejecuta su micro código especial tipo RISC para desempeñar funciones que planifiquen tareas y lleven a cabo entradas/salidas de alta velocidad. La figura 8 muestra un

MODULO SERIAL ENCOLADO (QSM)



QSM

SCI:
-Sistema Serial Asíncrono Mejorado
-Módulo Contador de Baud Rate
(64 Baud a 524K Baud con reloj de16.78 Mhz)
-Generación y Detección de Paridad
-Detección de Errores
-Full o Half Duplex

-Ideal para comunicaciones remotas

QSPI:
-Interfaz Serial Síncrona

-RAM de COLA para hasta 16 transferencias
  -Longitud de Transferencia Variable
  -Selección de Circuito Automático
  -Retrasos programables entre transferencias
  -Una vez y Alto o Modo Enrrollado

-Ideal para Comunicaciones entre procesadores en una misma tarjeta

-Las terminales del QSM pueden ser usados como E/S de propósito general

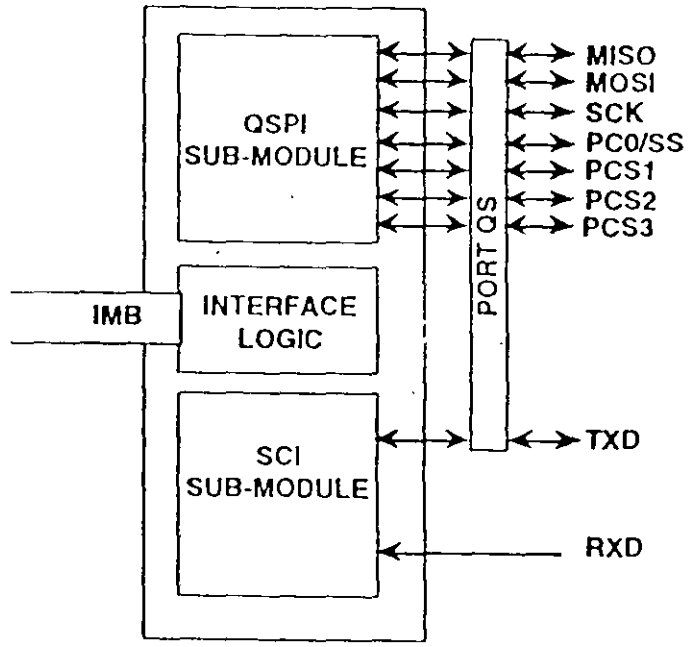


diagrama de bloques simplificado de su módulo de tiempo. La ventaja del TPU es la habilidad para operar independientemente del CPU16 usando el código microprogramado. Algunas de las funciones desempeñadas por el TPU son:

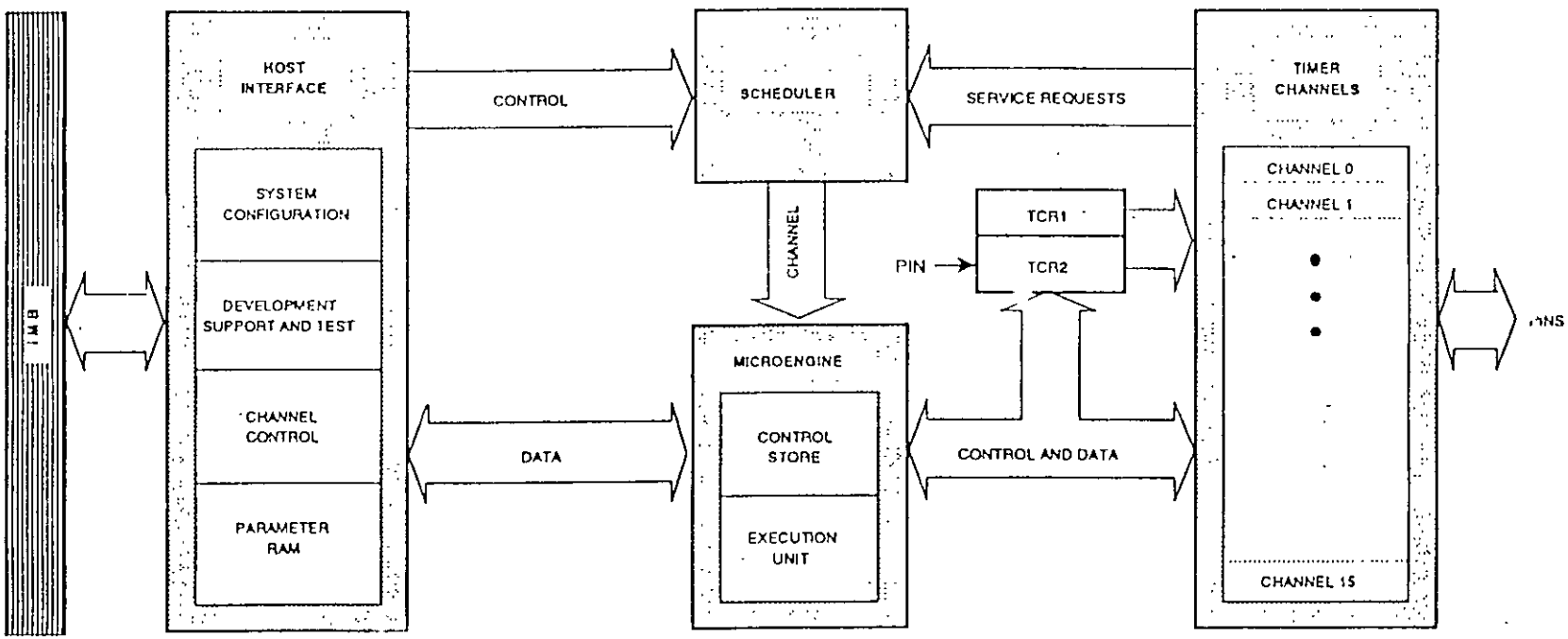
-Entradas/Salidas Discretas
-Entrada de Captura/Conteo de Transiciones de Entrada
-Salidas de Comparación
-Modulación por Ancho de Pulso
-Modulación por Ancho de Pulso Sincronizada
-Mediciones de Periodo con Perdida de Detección de Transición
-Generador de Pulsos con Sincronización de Posición
-Acumulador de Periodo/Ancho de Pulso
-Motor de Pasos

El TPU provee aceleración lineal y control de desaceleración de motores de pasos con un número de tasa de pasos programables. El programa del usuario pone el paso de la posición deseada, emitiendo una petición de paso y el TPU da los pasos al motor para la tasa deseada usando un perfil de aceleración/desaceleración. Este perfil actualiza mientras el TPU genera los pasos para el motor, el cual provee un algoritmo ideal para el control del servo motor.

Integración del Sistema.
El Módulo de Integración del Sistema (SIM) es el responsable de la interfase del bus, protección del sistema y provee interrupciones periódicas para soportar rutinas de control con tiempo-critico. El SIM genera una señal de reloj para el sistema de 16.78 MHZ, usando un cristal de 32.768 khz como referencia. Este usa el sistema de reloj para sincronizar las interfaces internas y externas al bus, manejando toda la comunicación y transferencia de datos desde y hacia todos los módulos IMB y localidades externas de dirección, actùando como controlador de bus. El SIM también asiste en la conexión de memoria y otros periféricos al microcontrolador sin necesidad de lógica de pegamento, mediante hasta 12 líneas programables de selección de circuito.

Protección del Sistema.
El SIM ofrece a los diseñadores una forma de monitorear la actividad del bus externo e igualmente las actividades de operación del software. Construidos dentro del SIM como varios monitores. El monitor del bus observa al bus para evitar ciclos excesivamente largos que se dan sobre el

DIAGRAMA A BLOQUES SIMPLIFICADO DEL TPU



SIM- MODULO DE INTEGRACION DEL SISTEMA

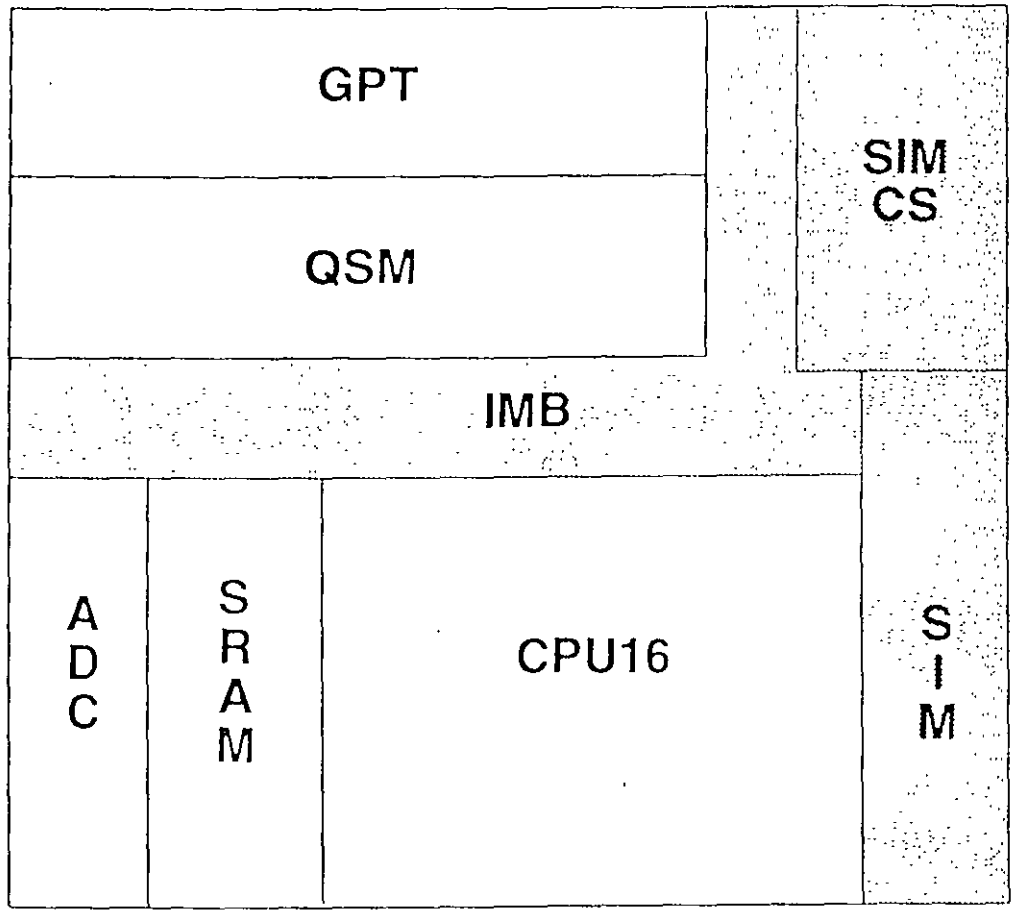
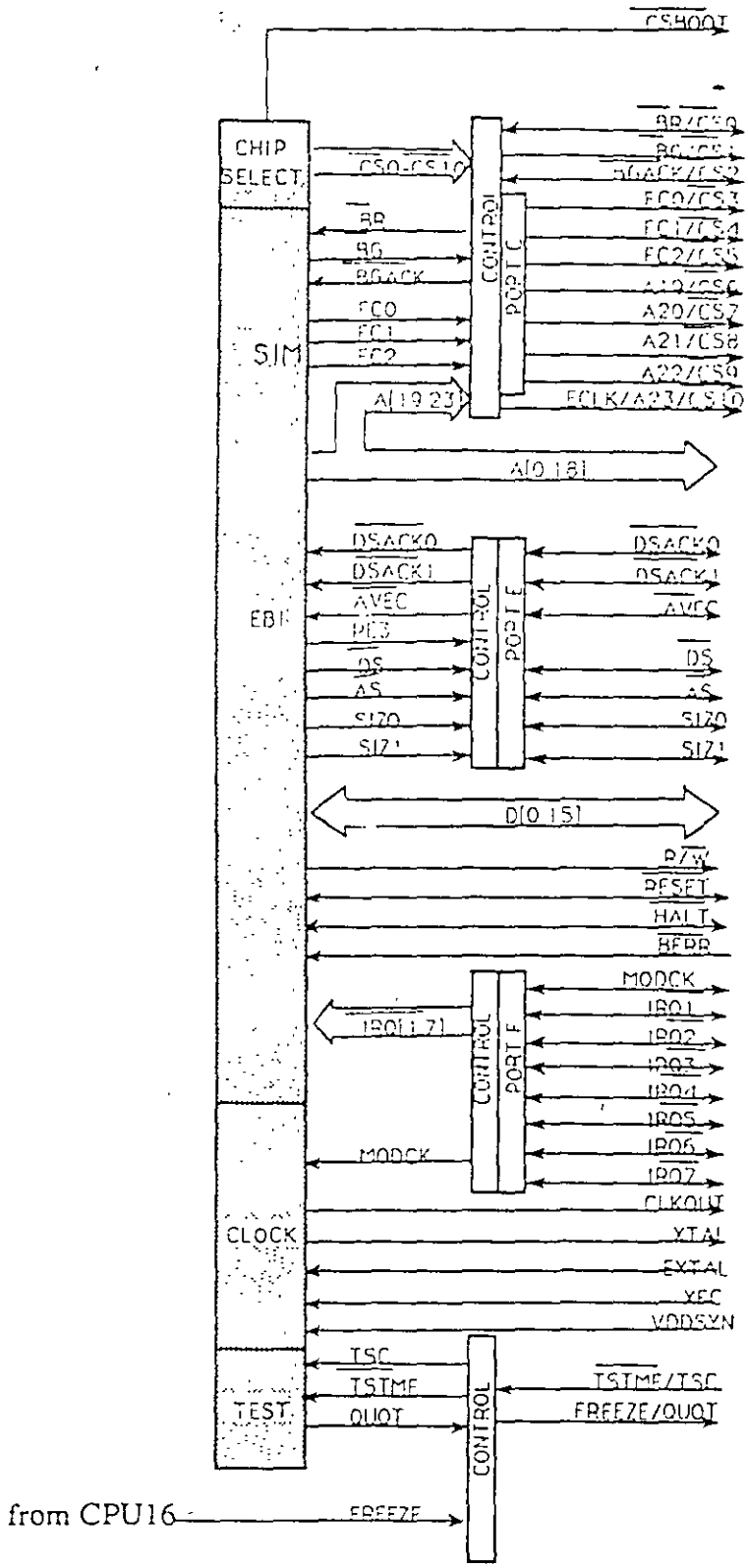


DIAGRAMA DE BLOQUE SIM



bus externo debido al acceso de memoria o periféricos externos inexistentes. El monitor del bus completa el ciclo activando un error de bus, el cual genera una señal de interrupción para el CPU16. El monitor de interrupción espuria supervisa el proceso de interrupción. El CPU16 reconoce la petición de interrupción, arbitra, reconoce el nivel más alto de interrupción y entonces busca el vector de interrupción pedido. El monitor de interrupción espuria genera un error de bus si no ocurre una fuente de arbitraje durante el ciclo de interrupción. El monitor del software llamado perro de guardia, requiere actualizaciones periódicas de su registro de servicio antes que el temporizador del perro de guardia expire, si el temporizador del perro de guardia expira el monitor activará una señal de reset.

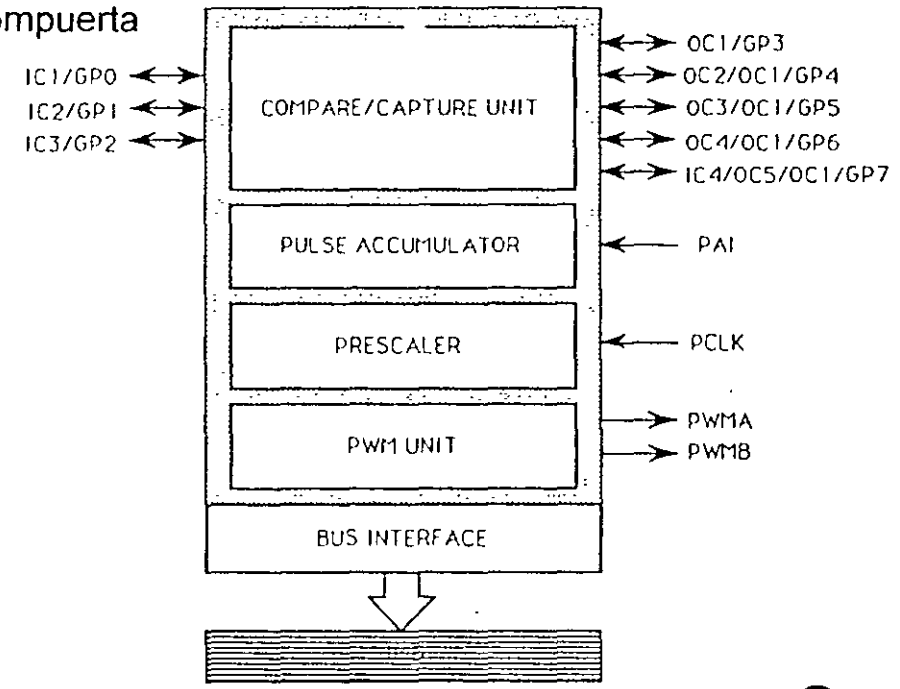
Selectores de Circuito Programables.

Un microprocesador típico requiere de hardware adicional para proveer señales externas de selección de circuitos. El SIM del 68HC16 resuelve este problema proveyendo hasta 12 líneas programables de selección de circuito de propósito general (ver figura 9). Cada línea de selección tiene asociada un registro base conteniendo la dirección base del circuito seleccionado. El registro de opciones provee la información del rango de direcciones con tamaño del bloque de dos byte hasta 512kbytes. El registro de opciones también configura la activación del selector de circuito y los estados de espera (cero a 13 ciclos) necesarios para memoria externa y periféricos. La activación de la línea de selección del circuito puede ser sincronizada con las señales del bus de control para proveer habilitaciones de salida, comandos de strobe o señales de reconocimiento de interrupción.

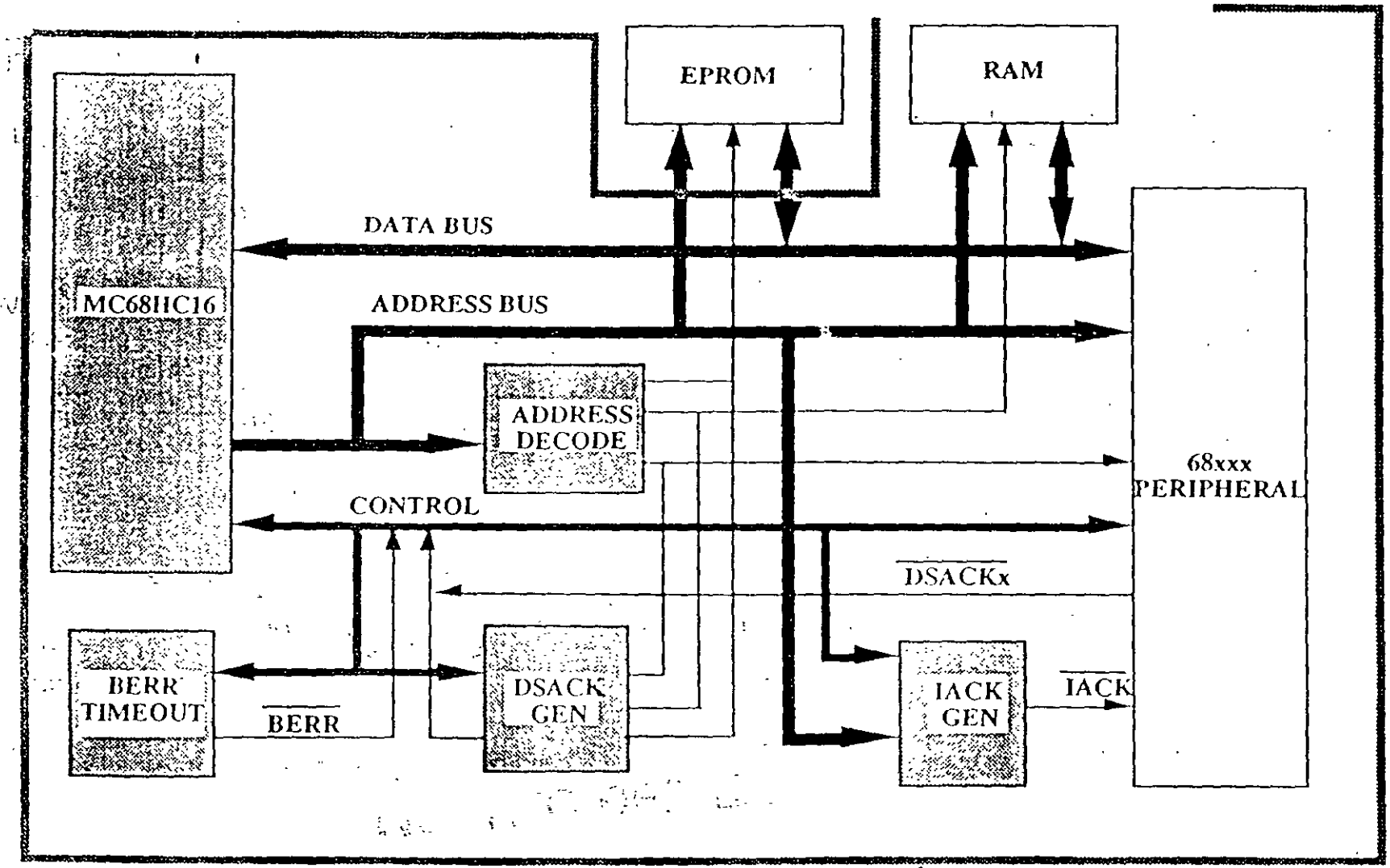
MODULO DEL TEMPORIZADOR DE PROPOSITO GENERAL- GPT

- Comparación/Captura unidad similar al del M68HC11F1
  - -Contador dedicado de 16-Bit (240ns de resolución)
  - -Tres canales de entrada de captura
  - -Cuatro canales de salida de comparación
  - -Un canal de propósito dual (E/captura o S/comparación)
  - -Una función de salida de comparación controladora de múltiples canales (OC1)
  - -Acumulador de Pulsos:
    - -Terminales Dedicadas
    - -Contador de lectura/escritura de 8-bit
    - -Contador de Eventos o Acumulación de Tiempo por compuerta
- PWM unidad similar al MC68HC05B6
  - -Dos salidas con modulación por ancho de pulso
  - -resolución de 8-bit
  - -Fuente de reloj independiente de la unidad Comparación/Captura
  - -Frecuencia desde 4 Hz a 32.8khz

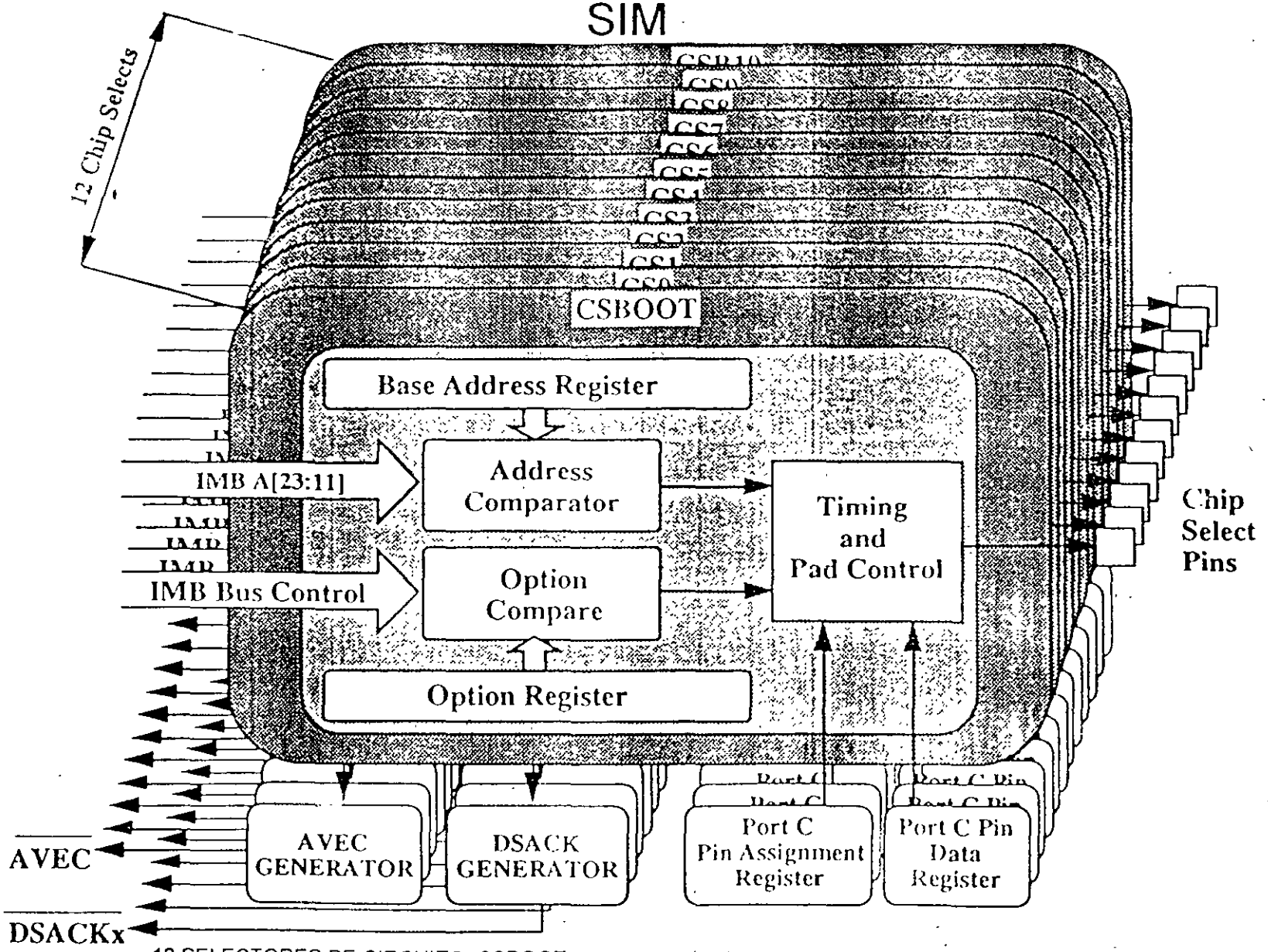
- Otras Características
  - Entradas dedicadas de reloj
- Las terminales pueden ser usadas como E/S de propósito general



LOGICA INTEGRACION EN ÉL CIRCUITO



MODULO DE SELECCION DE CIRCUITO SIM



-12 SELECTORES DE CIRCUITO CSBOOT selector de circuito dedicado, CS0-10
-Para Cada Selector de Circuito Seleccionar un selector de circuito en CSPAR0/1,activar el Registro de Dirección Base Registro de Opciones
-Los generadores AVEC y DSACK pueden ser u aún si la terminal no está configurada como Selector de Circuito