UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

"SIMULADOR DEL 68000 EN PC"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

PRESENTA:

SILVIA OCADIZ GARCIA E

INGENIERO EN MECANICA ELECTRICA

JOERN RALPH KARSTEN GIHP

DIRECTOR DE TESIS: ING. ABEL CLEMENTE REYES

México, D.F. 1991

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

INDICE TEMATICO

INTRODUCCION

El microprocesador 68000, es un caso especial en el mundo de los microprocesadores. Puede ser considerado como un microprocesador de 32 bits aunque se encuentre en la categoria de los microprocesadores de 16 bits. La primera version que salio al mercado fué precisamente el MCM68000 de Motorola Inc., en 1979. Este microprocesador, tiene un bus de datos de 16 bits y puede manejar direcciones de 24 bits. El hecho de tener 24 bits de direccionamiento permite inicialmente trabajar con una memoria de 16 Mega Bytes. Sus buses son sincronos y sin multiplexar. Las frecuencias de trabajo que soportan las distintas versiones del 68000 son: 4, 6, 8, 10 y 12.5 MHz.

Cuenta con 54 instrucciones muy potentes. Con 14 modos de direccionamiento y la combinación de distintas longitudes de datos con las cuales se originan más de 1000 codigos de instrucción.

El 68000 posee en total 17 registros internos de 32 bits, puede realizar operaciones directamente en memoria y trabaja 5 tipos de datos: Bit, Digito BCD (4 bits), Byte (8 bits), Palabra (16 bits) y Palabra Larga (32 bits).

Tiene una gran flexibilidad para la realización de sistemas de multiprocesamiento y sobretodo tiene una gran compatibilidad con los productos de hardware desarrollados para los microprocesadores de 8 bits como son memorias, módulos de entrada/salida, con formatos y velocidades variables, cuenta con un bus sincrono/asincrono para dispositivos como memorias y puertos asincronos/sincronos; esta caracteristica reduce su costo de implantación y aumenta sus posibilidades de utilización en sistemas basados en microprocesadores.

EL HARDWARE DEL 68000

En la sección anterior se desarrolló una breve introducción de lo que es el microprocesador 68000 y sus virtudes o características más relevantes. Ahora toca enmarcar dichas características en una forma no tan general sino a manera de lista. Para que de pie a explicaciones posteriores.

Caracteristicas fundamentales.

A) Estructura interna orientada especialmente hacia lenguajes de alto nivel. Esta fabricado con tecnología HMOS-I y también con tecnología HMOS-II.

B) Diseño para soportar una "programación estructurada" simplificada.

C) Arquitectura modular basada en un conjunto de registros de 32 bits.

D) La capacidad de direccionamiento es de 16 mega bytes en su direccionamiento más simple pero se puede tener la capacidad para direcionar 64 mega bytes.

E) Diferentes versiones para soportar frecuencias de 4, 6, 8,10,12.5 y 25 MHz.

F) 5 tipos de datos:
   1) Bit
   2) Digito BCD (4 bits).
   3) Byte (8 bits).
   4) Palabra (16 bits).
   5) Palabra larga (32 bits).

G) Necesita alimentarse de +5 V_{cc}.

H) Buses sincronos sin multiplexar y asincronos.

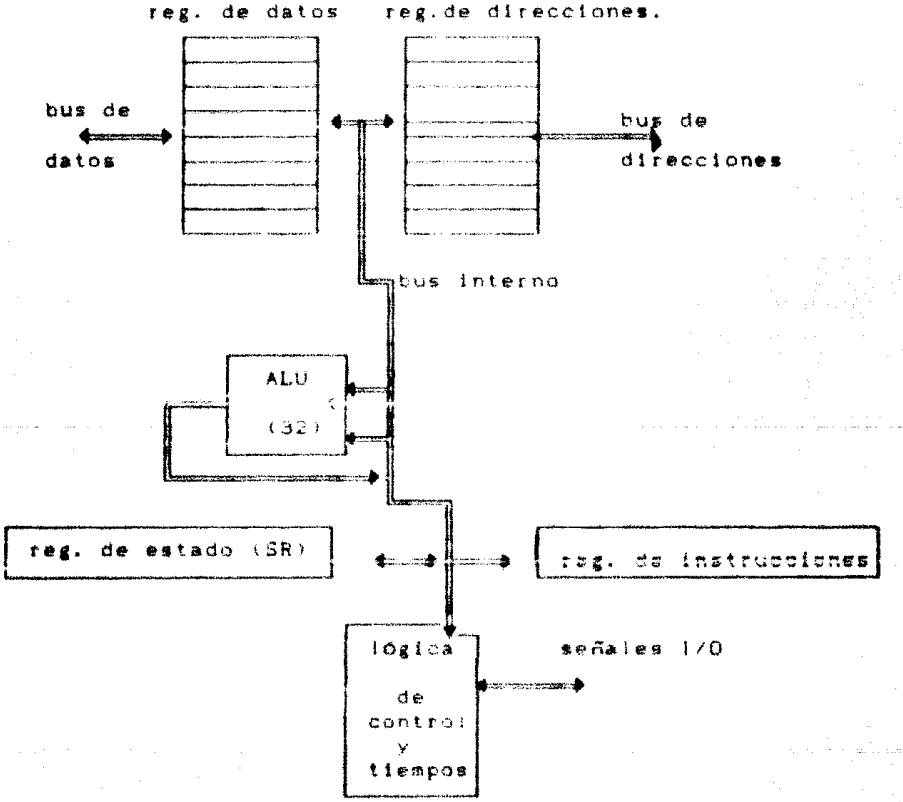
I) Cuenta con un hardware auxiliar para la detección de diferentes errores como:
   1) Instrucción ilegal
   2) Acceso incorrecto a memoria
   3) División por cero
   4) Interrupción ilegal

LA ARQUITECTURA DEL 68000

La arquitectura del microprocesador 68000 tiene los siguientes componentes:

Dos conjuntos de registros de 32 bits, un conjunto destinado a datos y el otro destinado a direcciones, cuenta con una Unidad Aritmetica Lógica (ALU), un registro de estado y el registro de instrucciones junto con la lógica de Control y Tiempos.

La ALU incluye operaciones de multiplicación, división, suma y resta con y sin signo en los diferentes longitudes de palabra que maneja el microprocesador.

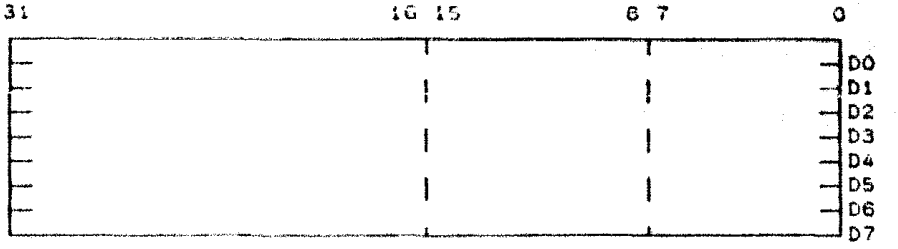


LA ARQUITECTURA DEL 68000

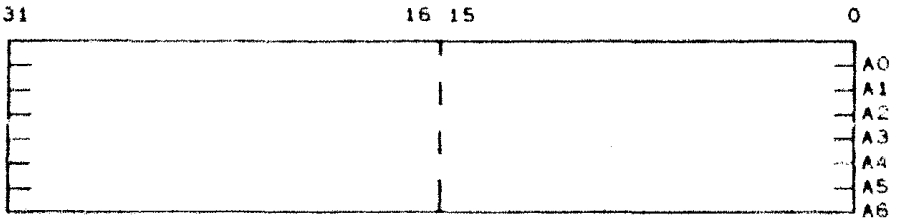
Registros del 68000.

Como se podrá observar en la figura, existen varios tipos de registros con los que cuenta el 68000, y son de datos, de direcciones, apuntadores, de status y de contador de programa.

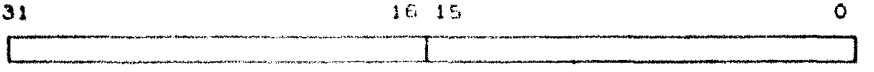
Registros de datos:



Registros de direcciones:



Registro Stack pointer (SSP o USP):



Contador de programa (PC):

Registro de Estado (Status register):

Los registros de datos son de 32 bits cada uno,estos registros se pueden utilizar como datos de un solo bit, datos de un byte, datos de una palabra y de una palabra larga. El bit menos significativo se le denomina bit 0 y el más significativo es entonces el bit 31. Con los registros de datos se pueden realizar operaciones sobre cualquiera de los 32 bits.

Las operaciones de un byte (B) se realizan sobre los 8 bits menos significativos (b7-b0), no afectando el contenido de los 24 bits restantes. En caso de que una operación B tuviera un resultado detamaño W ocurre un sobreflujo, que afectará al registro de estado (se explica más adelante este registro), pero no afecta a los bits (b8-b31). Las operaciones de una palabra (W) actuan sobre los 16 bits menos significativos. Al igual que en el caso anterior, una operacion W con resultado L provoca un desbordamiento, pero no afecta a los bits (b16-b31)

Por ultimo, se pueden realizar operaciones de palabra larga (L) ocupando los 32 bits del registro.

Los registros de direcciones son de 32 bits tambien y son 7, los registros A7 y A7' son registros que se utilizan como apuntadores de stack (usuario y supervisión respectivamente). Sin embargo es posible almacenar en ellos datos. Cuando actúan como operandos fuente pueden usarse únicamente con 16 o 32 bits. Cuando actuan como operando destino, se afectan los 32 bits del registro, independientemente del tamaño del operando de la instrucción. Es decir, si se trabaja con un operando de 16 bits, los otros 16 del registro de direcciones destino quedan afectados mediante la extensión del signo (bit más significativo).

Existe otro registro muy importante que es el contador del programa, el cual es de 32 bits y su función es apuntar a la siguiente dirección que va a ser accesada. De los 32 bits, sólo son utilizados 24 bits que es la longitud del bus de direcciones, de los cuales 23 son la dirección efectiva y el bit menos significativo, se utiliza para activar o desactivar las señales UDS (activa baja) y LDS (activa baja) e indican si se refiere al byte impar o par respectivamente. Cabe aclarar que esto sucede con la familia del 68000 (68008 y 68010).

El registro de estado, es de 16 bits y está dividido en dos partes: una es para el **sistema** y otro para el **usuario**, esto debido a que el 68000 puede operar de estos dos modos.

El modo usuario está previsto para ejecutar operaciones a nivel de aplicaciones. El modo sistema está proyectado para programas del sistema operativo. El modo sistema es el de más alta prioridad; todas las instrucciones del set son ejecutables en este modo.

Cuando el procesador trabaja en modo usuario, están activos todos los registros de datos, los registros A0 al A6, el apuntador de stack del usuario (USP) y los primeros 8 bits del registro de condición (CCR). Este byte contiene bits indicadores que dependen del resultado de ciertas, operaciones tales como: suma, resta, desplazamiento etc.

En la parte de **usuario** se tienen las siguientes banderas:

C- Acarreo (**Carry**). Pasa al nivel uno cuando hay acarreo en los bits mas significativos en las operaciones de suma o resta y tambien se afecta cuando se ejecutan instrucciones como las de desplazamiento, rotación, etc.

V- Sobreflujo (**Overflow**). Cuando en las operaciones con signo el resultado sobrepasa a la representación en complemento a dos, esta bandera se prende (se activa en 1).

Z- Cero. Cuando el resultado de una operación ha sido cero, esta bandera se prende (se activa en 1).

N- Signo. Se enciende cuando el resultado de una operación es de signo negativo.

X- Acarreo Extendido. Es una bandera de acarreo para operaciones de multiple presición y sirve para encadenar operaciones en las que intervienen datos de mayor longitud, es transparente al movimiento de datos y, en las instrucciones donde actua, toma el mismo valor que la bandera C.

La parte de supervisor, consta de las siguientes banderas:

T- Modo Trazado (Trace). Se activa en uno durante el estado supervisor, cuando se encuentra en uno, el usuario puede conocer el estado del sistema y poder ejecutar paso a paso un programa.

S- Supervisor. Se activa en uno cuando el microprocesador se encuentra en estado supervisor, donde se puede modificar la màscara de interrupciones.

I_0-I_2-Màscara de interrupciones. Su combinación lógica refleja el nivel de prioridad a partir del cual se atiende a las interrupciones.

Las terminales de comunicación del 68000.

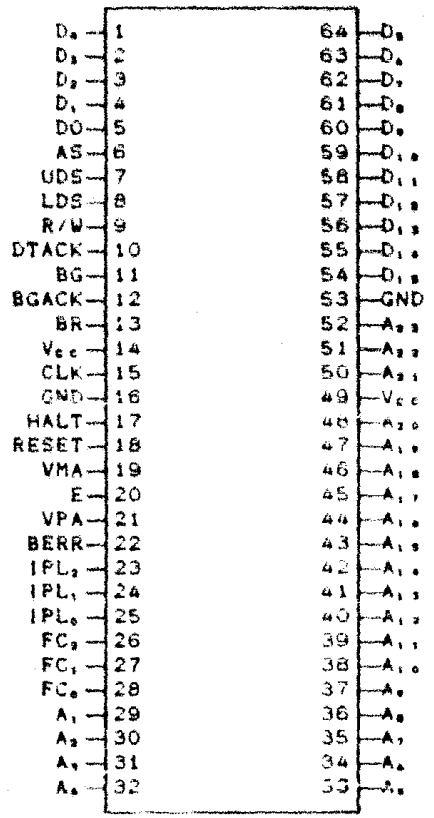
El 68000 consta de 64 terminales mediante las cuales se puede desarrollar un potente sistema de procesamiento.

A este sistema se le puede acoplar el mayor tipo de memorias y módulos de entrada y salida estándar. Tiene dos buses de control: uno asincrono y otro sincrono que amplia la posibilidad de conexión de periféricos.

Ademas del bus de datos de 16 líneas (D_0-D_{15}) y el bus de direcciones de 23 (A_1-A_{23}), existen otras señales, que se agrupan formando los conjuntos siguientes:

1) Bus de control sincrono.
2) Bus de control asincrono.
3) Control de DMA o gestión del bus.
4) Control de la función o estado de la CPU.
5) Control del sistema.

La siguiente figura muestra la disposición de las terminales de este microprocesador.



Lineas de alimentación.

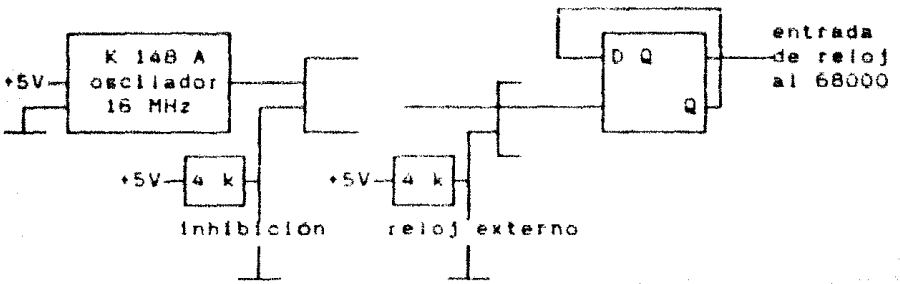
Las terminales 14 y 19 son para conectar la alimentación positiva de la fuente de poder de 5 V_{cc} con una tolerancia de 5%. Las terminales 16 y 53 son para la referencia (tierra) del sistema.

El consumo del 68000 varia entre 1.50 y 1.75 Watts, dependiendo de la frecuencia a la que opera y al tipo de encapsulado (cerámico o plástico).

Señal de reloj.

El reloj corresponde a la terminal 15 del microprocesador e introduce una frecuencia estable a niveles TTL que sirve para desarrollar internamente, los pulsos de reloj de sincronia.

En la siguiente figura se muestra un circuito electrónico tipico, encargado de generar la señal CLK para el 68000.



Nota: En lugar del oscilador mostrado, se puede utilizar uno de cuarzo.

Lineas de datos.

Son 16 lineas bidireccionales identificadas de la D_0 a la D_{15} que se encuentran en la parte superior de la cápsula; el tamaño del bus implica que para transmisión de datos de mayor longitud, sea necesario más de un ciclo de bus.

En el caso de las interrupciones, cuando se ha reconocido una de ellas, el periférico exterior debe proporcionar sobre las lineas D_0 a D_7, el numero del vector de interrupción. Sólo los periféricos asincronos de la familia 68000 están preparados para esta forma de actuación.

Lineas de direcciones y señales UDS y LDS.

Las lineas de direcciones se componen de 23 lineas unidireccionales denominadas por la nomenclatura de la A_1 a la A_{23} y las cuales permiten accesar 2^{23} direccioes de 16 bits (8 Mbytes).

Junto a este conjunto de lineas de direcciones se encuentran las señales UDS y LDS (activas bajas) las cuales seleccionan el byte par o impar de una dirección o palabra, con esto se puede controlar hasta 16 Mbytes (8 mega bytes para las pares y 8 mega bytes para las impares). El estado de A_0 del registro contador de programa determina el nivel de estas señales. Si A_0=0, se activa UDS, accesando a los bytes pares por las lineas D_8-D_{15}. Si A_0=1, se activa LDS y se accesa a los bytes impares a través de las lineas D_0-D_7. La activación simultánea de las señales UDS y LDS permite el acceso de una palabra.

Cuando se produce un ciclo de interrupción, las lineas A_1, A_2 y A_3 reflejan el nivel de prioridad de la interrupción reconocida.

Lineas de control asincrono.

Se habia mencionado en la introducción, que el microprocesador 68000 contaba con un bus sincrono/asincrono para periféricos sincronos/asincronos y dispositivos como memorias; en esta parte se tratarán las lineas del microprocesador que hacen posible la comunicación sincrona de éste con los dispositivos externos.

Las lineas fundamentales son: R/W (señal de salida), AS (señal de salida), DTACK (señal de entrada), UDS (señal de salida) y LDS (señal de salida) de las cuales, las lineas UDS y LDS ya han sido tratadas.

La señal R/W determina si el ciclo es de lectura (activa en 1 fisico) o si es de escritura (activa en cero) y controla la actuación del bus de datos en combinación con las señales UDS y LDS y se pueden presentar los casos que se muestran en la siguiente tabla.

Ciclo	R/W	UDS	LDS	BUS DE DATOS D_8 - D_{15}	D_0 - D_7
-	-	1	1	NO VALIDOS	NO VALIDOS
lectura	1	0	0	SI	SI
lectura	1	1	0	NO	SI
lectura	1	0	1	SI	NO
escritura	0	0	0	SI	SI
escritura	0	1	0	NO	SI
escritura	0	0	1	SI	NO

AS (address strobe) cuando está activa esta señal (nivel bajo), indica la presencia de una dirección válida en el bus de direcciones para periféricos asincronos.

DTACK (Data Transfer Acknowledge) es una señal activa baja es de entrada al microprocesador y le indica el fin de transmisión de datos; cuando el CPU reconoce esta señal, recoje la información del bus de datos y termina el ciclo de lectura.

La combinación de estas dos señales últimas, da lugar a una transferencia asincrona; esquemáticamente, se dice que el 68000 activa la señal AS en la presencia de una dirección válida para acceder a algún periférico y no la retira hasta recibir la señal DTACK.

Lineas de control Sincrono.

Con el objeto de poder hacer compatibles los periféricos con que cuentan los predecesores de este microprocesador, se creó una serie de señales que permiten una comunicación sincrona con dichos periféricos.

Las lineas que hacen posible este tipo de comunicación son: **E** (señal de salida), **VMA** (señal de salida), **VPA** (señal de entrada) y se explican a continuación.

Por la terminal **E** (enable), se genera una señal de reloj con una frecuencia fija, cuyo periodo se compone de 10 periodos del reloj del microprocesador, 6 de nivel bajo y 4 de nivel alto.

La señal **VPA** (Valid Peripherial Address) es activa baja y con esta se le informa al microprocesador que el elemento direccionado es un dispositivo de la familia 6800, la transferencia de la información va a estar sincronizada de acuerdo a los pulsos de reloj procedente de la línea E; también esta señal indica que la interrupción que se está recibiendo va a tener una vectorización automática para su atención.

La señal **VMA** (Valid Memory Adress) es activa baja e indica que la dirección presente en el bus de direcciones es válida y que el CPU está sincronizado para su activación. VMA es una señal de respuesta a VPA.

Líneas de control de la función o estado del procesador.

Son tres señales las que reflejan la actividad del microprocesador a saber: FC_0, FC_1 y FC_2 (señales de salida) y que informan del modo de trabajo (supervisor o usuario) y de si la zona accedida por el bus de direcciones corresponde a programa o a datos. El uso eficiente de estas líneas permiten controlar 4 mapas de memoria de 16 Mbytes cada uno. La información que proporcionan estas líneas, llamadas FC_0, FC_1, FC_2, es válida siempre que la señal AS (activa baja) se halle activada.

En la tabla siguiente se presenta el ciclo correspondiente a cada combinación de estados.

FC_2	FC_1	FC_0	CICLO
0	0	0	Reservado sin definir
0	0	1	Datos usuario.
0	1	0	Programa usuario.
0	1	1	Reservado.
1	0	1	Datos supervisor.
1	1	0	Programa supervisor.
1	1	1	Reconocimiento interrupción

La decodificación de los estados de estas líneas sirven para controlar las cuatro zonas de memoria de 16 Mbytes; además de usarse en la manipulación de memoria del sistema sirven para proteger zonas especiales de la misma. Requieren una lógica auxiliar precisa a la salida de estas líneas.

Lineas de control de interrupciones.

Son las lineas IPL_0, IPL_1 e IPL_2, las cuales tienen el objetivo de recibir el nivel de prioridad del dispositivo que solicita la interrupción.

La mayor prioridad es la del nivel 7 (en binario: 111), que es una interrupción no mascarable y la minima prioridad es 0 (en binario:000).

Si el código del periférico es de mayor prioridad que la establecida en el registro de estado (I_2, I_1, e I_0), se atiende y en caso contrario se ignora. Si la interrupción se acepta, el CPU presenta el nivel de la interrupción aceptada por medio de las lineas A_1, A_2 y A_3, lo que sirve como reconocimiento de la misma e inmediatamente procede a su ejecución como se explicará más adelante. El reconocimiento de una interrupción pone a las lineas FC_0, FC_1 y FC_2 en nivel alto.

Cuando se acepta una interrrupción, el SR y su máscara se guardan en el stack y en la máscara se carga el nivel de prioridad del periférico atendido, esto se hace para evitar que pueda aceptar interrupciones de menor prioridad. Estos detalles se explicaran en la sección de interrupciones.

Lineas de control de DMA o de arbitraje de bus.

Las lineas que comprenden este sistema son: **BR Petición de bus (Bus Request)**, **BG Cesión de bus (Bus Grant)** y **BGACK Control de bus (Bus Grant Acknowledge)**.

Dichas lineas sirven para conceder el control de los buses del sistema a los periféricos que lo solicitan para hacer un acceso directo a memoria. A continuación se explican las funciones de cada una de las lineas mencionadas.

BR Petición del bus (Bus Request).

Es una linea de entrada al 68000, que sirve para comunicarle que un dispositivo exterior desea tomar el control de los buses del sistema.

Para establecer una prioridad en los elementos que pueden ser los peticionarios de bus, se les suele colocar en serie como se muestra en la siguiente figura:

BG Cesión de bus (Bus Grant).

Esta linea sale del microprocesador y notifica la recepción de una petición de sección de bus. Se activa un ciclo de reloj despues de que se activa BR.

La cesión de los buses ocurre cuando finaliza el ciclo de bus en curso. En otras palabras, el bus se entrega cuando las señales AS y DTACK están en nivel alto (desactivadas), lo que significa que el CPU deja en estado de alta impedancia al bus de datos, al de direcciones, a las lineas FC_0, FC_1 y FC_2 y a las lineas de control del bus, AS, UDS, LDS y R/W.

BGACK Control de bus (Bus Grant Acknowledge).

Esta señal procede del peticionario de bus y confirma a el CPU que ya ha tomado su control. A partir de ese momento, el nuevo maestro desactiva BR y gobierna a los buses mientras mantenga activa esta señal. Al desactivarse BR, el 68000 desactiva BG.

Lineas de control del microprocesador.

Son tres lineas y son **BERR** Error de bus (Bus error), **RESET** (Reestablecimiento) y **HALT**. La descripción de estas señales es la siguiente:

RESET Reestablecimiento.

Es una línea bidireccional y puede ser activada de dos formas: desde software y desde hardware, cuando el CPU activa esta línea, se debe a la ejecución de la instrucción RESET. Durante 124 ciclos de reloj mantiene la línea en nivel 0, para dar el suficiente tiempo de respuesta a los elementos exteriores. En esta situación el estado del CPU no se afecta, pasando a continuación a la siguiente instrucción.

Cuando esta señal viene del exterior del microprocesador y permanece durante más de 4 ciclos de reloj, en ese instante se procede a un proceso de excepción.

HALT.

Cuando funciona como entrada y es activada desde fuera de el CPU, permite ejecutar un programa paso a paso con el fin de depurarlo. En el estado HALT, después de completarse el ciclo de bus en curso, las líneas de los buses de datos y direcciones, así como las de código de función pasan a alta impedancia quedando también desactivadas las líneas de control de bus o DMA.

El CPU activa la señal HALT, cuando se detiene a consecuencia de un error en el bus y da lugar a una excepción que será comentada en la sección de excepciones.

Cuando se activan simultáneamente las señales de RESET y HALT, se produce una inicialización que afecta a todo el sistema.

BERR Error de bus (Bus Error).

Es una señal de entrada al procesador, misma que le informa sobre la ocurrencia de un error en el sistema, como podria ser un acceso ilegal a memoria o el fallo en la respuesta de algun periférico. Al activarse BERR, el microprocesador procede a un proceso de excepción de error de bus o nuevamente, repite el ciclo de bus. Este último caso sucede cuando BERR=HALT=0, actuando HALT como entrada.

Ciclo de lectura.

I Direccionamiento del periférico.
- Pone la señal R/W en 1.
- Pone el código de la función en FC_0-FC_2.
- Coloca la dirección en A_1 - A_{23}.
- Activa la señal AS.
- Activa las lineas UDS y LDS.

II Salida de datos del periféricos.
- Decodifica la dirección.
- Coloca el dato sobre D_0 - D_{15}.
- Activa la señal DTACK.

III Final de la transferencia.
- Se refistra el dato.
- Se desactivan UDS y LDS.
- Se desactiva AS.
- El periférico retira el dato.
- Desactiva DTACK el periférico.

Ciclo de escritura

I Direccionamiento del periférico.
- Código de función sobre FC_0 - FC_2.
- Coloca la dirección en A_1 - A_{23}.
- Activa la señal AS.
- Pone a 0 la señal R/W.
- Coloca el dato en D_0 - D_{15}.
- Activa las lineas UDS y LDS.

II Entrada de datos al periférico.
- Decodifica la dirección.
- Almacena el dato.
- Activa la señal DTACK.

III Final de la transferencia.
- Desactiva UDS y LDS.
- Desactiva AS.
- Quita el dato de D_0 - D_{15}
- Pone a 1 la señal R/W.
- Desactiva DTACK el periférico.

Organización de la Memoria

El 68000 trabaja con memorias de 8 bits conectadas en paralelo. Esto permite al programador de software usar bytes, palabras y palabras largas.

Estas memorias conectadas en paralelo se conocen como memoria par y memoria non. La que contiene los bits más significativos se le llama memoria par y a la que contiene los bits menos significativos se le llama memoria non.

La memoria par esta conectada a las lineas más significativas del bus de datos (D8-D15) y la memoria non esta conectada a las lineas menos significativas del bus de datos (D0-D7).

En páginas anteriores se mencionó que el 68000 maneja direcciones de 24 bits, sin embargo el bus de direcciones sólo cuenta con 23 lineas.

Estas lineas son de la A23 a la A1. Con estas 23 lineas el 68000 puede accesar 8,388,608 localidades de 16 bits (8 Mega Palabras). El bit A0 no está conectado directamente al "mundo exterior". En lugar de esta linea se tienen dos lineas adicionales llamadas UDS y LDS. El valor de este bit A0 y alguna información adicional sobre el tamaño de la instrucción o dato generan valores para las lineas UDS y LDS.

Para poder seleccionar una localidad de 8 bits, la linea UDS o LDS se activa para activar una memoria par o non. Si se desea accesar el byte menos significativo de alguna localidad, la linea UDS es deshabilitada y la linea LDS es habilitada, habilitando de esta manera a la memoria non. El dato viajará por los 8 bits menos significativos del bus de datos. Si se desea accesar el byte más significativo de alguna localidad, la linea UDS es habilitada y la linea LDS es deshabilitada, es decir, es habilitada la memoria par.

El dato viajará por los 8 bits más significativos del bus de datos.

Si se desea accesar una localidad de 16 bits las líneas UDS y LDS son habilitadas, habilitando de esta manera las dos memorias; el dato viaja por los 16 bits del bus de datos. La dirección de esta localidad debe ser necesariamente par.

Tambien es posible trabajar datos de 32 bits. En este caso el 68000 accesa 2 localidades de 16 bits contiguas. La dirección de esta localidad de 32 bits tambien debe ser par.

Si se pudiera hacer referencia a una localidad non de 16 bits el 68000 deberá accesar primero al byte menos significativo de dicha localidad (este byte viajará por las líneas menos significativas del bus de datos). Posteriormente se deberá accesar al byte más significativo de la localidad siguiente (este byte viajara por las líneas más significativas del bus de datos). Finalmente se tendrá en el bus de datos que el byte más significativo se encuentra en los bits D0-D7 y el byte menos significativo en los bits D8-D15. Es decir, se invertira la posicion de los bytes cambiando radicalmente el valor de la palabra. Para evitar este problema el 68000 incurre en un error al detectar una localidad non de 16 o 32 bits y ejecuta un proceso de excepción. Es por esta misma razón que las instrucciones del 68000 son de frontera par.

Instruociones del 68000

Como ya se mencionó, el 68000 cuenta con 56 instrucciones; las cuales, combinadas con la enorme variedad de modos de direccionamiento, ofrecen una muy potente herramienta para el desarrollo de software para los sistemas basados en este microprocesador.

Las instrucciones pueden agruparse en ocho tipos distintos:

* Transferencias (o movimiento) de datos.
* Aritmética entera.
* Aritmética booleana
* Desplazamientos y rotaciones.
* Manipulación de bits individuales.
* Manipulación en decimal codificado binario (BCD).

* Control de flujo de programa.
* Control del sistema.

Es importante mencionar que el 68000 no cuenta con instrucciones de Entrada/Salida. A diferencia de algunos otros microprocesadores, el 68000 se comunica con los puertos exactamente de la misma manera que lo hace con la memoria.

Las instrucciones de transferencia de datos permiten el desplazamiento de datos entre registros, entre un registro y una posición de memoria y directamente entre dos posiciones de memoria. A continuación se enumera cada una de las instrucciones de este tipo:

EXG	intercambio de registros
LEA	cargar dirección efectiva
LINK	enlazar ubicar stack
MOVE	mover fuente a destino
MOVEA	mover fuente a registro de dirección
MOVEC	transferir registro de control
MOVEM	transferencia múltiple de registros
MOVEP	transferencia a periférico
MOVEQ	mover datos cortos a destino
MOVES	mover espacio de direcciones
PEA	insertar en stack dirección efectiva
UNLK	desenlazar stack
SWAP	intercambiar palabras de una palabra larga

Operaciones aritméticas. El 68000 cuenta con cuatro funciones aritméticas básicas de punto fijo: suma, resta, multiplicación y divison. Se tienen además instrucciones de comperación entre dos enteros, extensión de signo y complemento, así como para borrar (poner a cero). Las instrucciones que pertenecen a este tipo son:

ADD	sumar fuente a destino
ADDA	sumar fuente a registro de dirección
ADDI	sumar dato inmediato a destino
ADDQ	sumar dato corto a destino
ADDX	sumar con bit extendido a destino
SUB	restar fuente al destino
SUBA	restar fuente al registro de dirección
SUBI	restar inmediato al destino

SUBQ	resta corta al destino
SUBX	resta con bit extendido al destino
NEG	negar (complementar fuente)
NEGX	negar con extensión de signo
MULU	multiplicación sin signo
MULS	multiplicación con signo
DIVU	dividir sin signo
DIVS	dividir con signo
EXT	extensión del signo
EXTB	byte extensión del signo
CLR	borrar el operando (poner a cero)
CMP	comparar fuente con destino
CMPA	comparar fuente con registro de dirección
CMPI	comparar dato inmediato con destino
CMPM	comparar memoria

Las operaciones de multiplicación y división se pueden realizar con o sin signo. Se pueden multiplicar dos numeros de 16 bits obteniendo un resultado de 32 bits. Se puede realizar la división de un dividendo de 32 bits con un divisor de 16, obteniendo un resultado de 16 bits.

Las instrucciones de aritmética booleana llevan a cabo las operaciones lógicas fundamentales, típicas de todo microprocesador. Las instrucciones AND, OR y EOR pueden realizar operaciones con dato inmediato. A continuación se enumeran las instrucciones booleanas:

AND	AND entre fuente y destino
ANDI	AND entre dato inmediato y destino
OR	OR entre fuente y destino
ORI	OR entre dato inmediato y destino
EOR	exclusive OR entre fuente y destino
EORI	exclusive OR dato inmediato y destino
NOT	NOT del destino
Scc	comprobar los códigos de condición y poner a uno el operando
TST	comprobar operando y poner a uno los códigos de operación

Instrucciones de desplazamiento y rotación. El 68000 permite hacer desplazamientos aritméticos y lógicos y rotaciones con o sin signo. A continuación se enumera cada una da las instrucciones de este tipo:

ASL	desplazamiento aritmetico a la izquierda
ASR	desplazamiento aritmetico a la derecha
LSL	desplazamiento logico a la izquierda
LSR	desplazamiento logico a la derecha
ROL	rotación a la izquierda
ROR	rotación a la derecha
ROXL	rotación a la izquierda con bit extendido
ROXR	rotación a la derecha con bit extendido

Las instrucciones de manejo de bits son utilizadas para probar el estado de un bit especifico, ya sea de un registro de datos o de un byte de una localidad de memoria. Son 4 instrucciones que se enumeran a continuación:

BTST	revisar bit
BSET	poner bit a uno
BCLR	borrar bit (poner a cero)
BCHG	cargar bit

Las instrucciones decimal codificado realizan operaciones en aritmética BCD. El 68000 cuenta con las siguientes instrucciones:

ABCD	sumar fuente a destino
NBCD	negar destino
SBCD	restar fuente al destino

Las instrucciones de flujo del programa permiten realizar bifurcaciones condicionales e incondicionales a cualquier punto del programa. Es posible realizar bifurcaciones referidas al PC asi como bifurcaciones absolutas. Además se cuenta con instrucciones que permiten llamadas a subrutinas y retornos utilizando el STACK. A continuación se presentan las instrucciones pertenecientes a este tipo:

Bcc	ramificación condicional
BRA	ramificación incondicional
BSR	ramificación a subrutina
DBcc	comprobar, decrementar y ramificar
JMP	saltar a dirección
JSR	saltar a subrutina
NOP	no opera
RTE	retornar de excepción (instr. privilegiada)
RTR	retornar y restaurar códigos de condición
RTS	retornar de subrutina

Instrucciones de control del sistema incluyen operaciones privilegiadas (solo realizables estando en modo supervisor) e instrucciones que permiten la interconexión entre programas en modo usuario y en modo supervisor. Las instrucciones de este tipo son las siguientes:

ANDI	AND inmediato con el reg. estado/condiciones
BKPT	punto de ruptura
CHK	intercep. de operando con frontera sup. errónea
EORI	exclusive OR inmediato con estado
ILLEGAL	intercepción de instrucción ilegal
MOVE	transf. de/hacia reg. de estado/condiciones
MOVEC *	transferencia de/hacia registro de control
MOVES *	transferencia de/hacia espacio de direcciones
RESET *	reiniciar periféricos conectados a línea RESET
STOP *	detener procesador
TRAP	intercepción incondicional
TRAPV	intercepcion si hay desbordamiento

* instrucciones privilegiadas

En el apéndice se pueden consultar todas las instrucciones en forma más extensa.

MODOS DE DIRECCIONAMIENTO

Los **modos de direccionamiento** indican la forma en la cual el CPU debe adquirir un operando para ejecutar la instruccion presente.

En el 68000 existen seis clases principales de direccionamientos con sus respectivas variantes, generando asi 14 modos diferentes para obtener una dirección o un dato. A continuación se les presenta.

1. **Direccionamiento directo de registro:**
   a) registro directo de datos
   b) registro directo de direcciones

2. **Direccionamiento indirecto de registro:**
   a) registro indirecto
   b) registro indirecto con postincremento
   c) registro indirecto con predecremento
   d) registro indirecto con desplazamiento
   e) registro indirecto con desplazamiento e indice.

3. **Direccionamiento absoluto de datos:**
   a) corto absoluto
   b) largo absoluto

4. **Direccionamiento relativo del PC (Contador de Programa):**
   a) contador de programa con desplazamiento
   b) contador de programa con indice

5. **Direccionamiento inmediato de datos:**
   a) inmediato
   b) inmediato rapido

6. **Direccionamiento implicito:**
   a) registro implicito

Dos caracteristicas importantes en el direccionamiento del 68000 son: que cualquier registro puede ser utilizado para direccionamiento directo o indirecto, y que cualquier registro puede ocuparse como un registro indice.

Los ocho registros de datos manejan 1, 4, 8, 16 o 32 bits. Los registros de direcciones, junto con el SP (apuntador al stack), manejan direcciones de operandos de 32 bits.

MODOS DE DIRECCIONAMIENTO

Si los operandos son de 24 bits, los 8 bits restantes no se toman en cuenta. Al utilizar un registro de datos como un operando fuente o destino, solo se modifican los bits que le correspondan: 8 bits (byte), 16 bits (word o palabra), o 32 bits (long word o palabra larga). Los bits restantes del registro de datos permanecen sin alteracion.

Para el caso de los registros de direcciones y el SP (stack pointer), se modifican los 16 bits (palabra) o los 32 bits (palabra larga), segun sea la longitud de su registro.

El formato de las instrucciones es el siguiente. Las instrucciones tienen una longitud de una a cuatro palabras. En la primera (palabra de operacion) se define la longitud de la instruccion y de la operacion a efectuarse. Las palabras restantes (palabras de extension) especifican los operandos, los cuales pueden ser inmediatos o extensiones al modo de direccionamiento efectivo, indicados en la palabra de operacion.

A continuacion se explican los modos de direccionamiento:

1. DIRECCIONAMIENTO DIRECTO DE REGISTRO

Este direccionamiento efectivo indica que el operando está en uno de los 16 registros de uso multiple.

a) Registro Directo de Datos.

El operando está en el registro de datos especificado por el campo del registro de la dirección efectiva.

Creación: EA = Dn
Sintáxis: Dn
Modo: 000 reg. de datos Dn
Registro: n

b) **Registro Directo de Direcciones.**

El operando se encuentra en el registro de dirección indicado por el campo del registro de la dirección efectiva.

Creación: EA = An
Sintáxis: An
Modo: 001 reg. de dir. An
Registro: n

2. DIRECCIONAMIENTO INDIRECTO DE REGISTRO

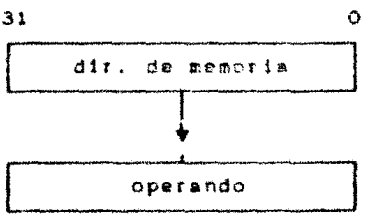
Estos indican que el operando está en alguna localidad de memoria y proveen la dirección especifica del operando. La "Palabra de Extensión" se emplea cuando se le aneva a la palabra de instrucción, una o dos palabras en donde se especifica completamente al operando.

a) **Registro Indirecto.**

La dirección del operando está en el registro de dirección indicado en el campo del registro. La referencia se clasifica como de datos, con la excepción del salto (jump) y del salto a una subrutina.

Creación: EA = (An)
Sintáxis: (An)
Modo: 010 reg. de dir. An
Registro: n

dir. de memoria

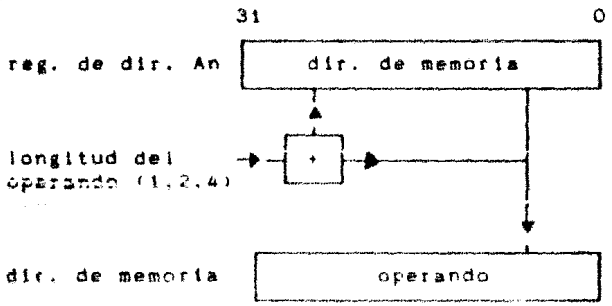


b) **Registro Indirecto con Postincremento.**

La dirección del operando está en el registro de dirección señalado por el campo del registro. Después de utilizar la dirección del operando, ésta se incrementa en uno, dos o cuatro, dependiendo si el tamaño del operando es de byte, palabra o palabra larga.

Si el registro de dirección es el apuntador a la pila (SP) y el tamaño del operando es de un byte entonces la dirección se incrementará en 2, y no en 1, para mantener al apuntador a la pila dentro de un formato de palabra. La referencia se clasifica como de datos.

```
Creación: An = An + N
Sintaxis: (An)+
Modo:     011
Registro: n
```

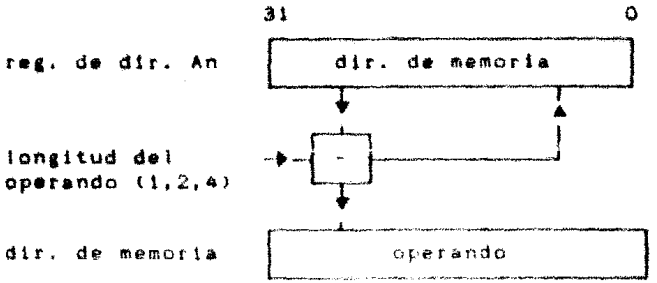


c) **Registro Indirecto con Predecremento.**

La dirección del operando se encuentra en el registro de dirección indicado por el campo del registro. Antes de utilizar la dirección del operando se decrementa en uno, dos o cuatro, dependiendo si la longitud del operando es de byte, palabra o palabra larga.

Si el registro de dirección es el SP y el operando es de un byte, la dirección se decrementa en dos, en lugar de uno, para mantener al SP dentro de un formato de palabra. La referencia se clasifica como de datos.

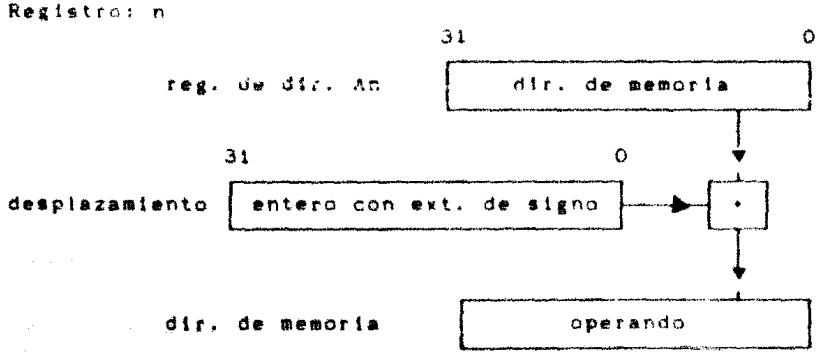
```
Creación: An = An - N
          EA = (An)
Sintáxis: -(An)
Modo:     100
Registro: n
```



d) **Registro con Desplazamiento.**

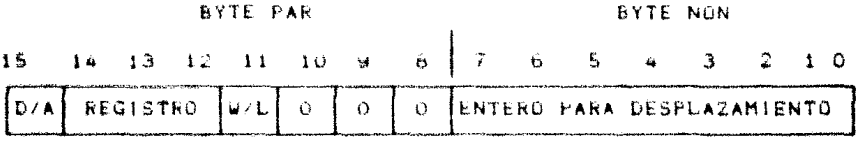
Este modo de direccionamiento necesita una palabra de extensión. La dirección del operando es la suma de la dirección, en el registro de dirección, más un entero con extensión de signo y 16 bits de desplazamiento en la palabra de extensión. La referencia se clasifica como de datos con la excepción del salto (jump) y del salto a una subrutina.

```
Creación: EA = (An) + d16
Sintáxis: d1C (An)  o  (d16,An)
Modo:     101
Registro: n
```



e) **Registro Indirecto con Desplazamiento e Indice.**

Este modo de direccionamiento requiere de una palabra de extensión que cumpla con el siguiente formato:



bits 15 : indicador del tipo de registro indice
0 = registro de datos (D)
1 = registro de dirección (A)

bits 14 al 12 : numero del registro indice

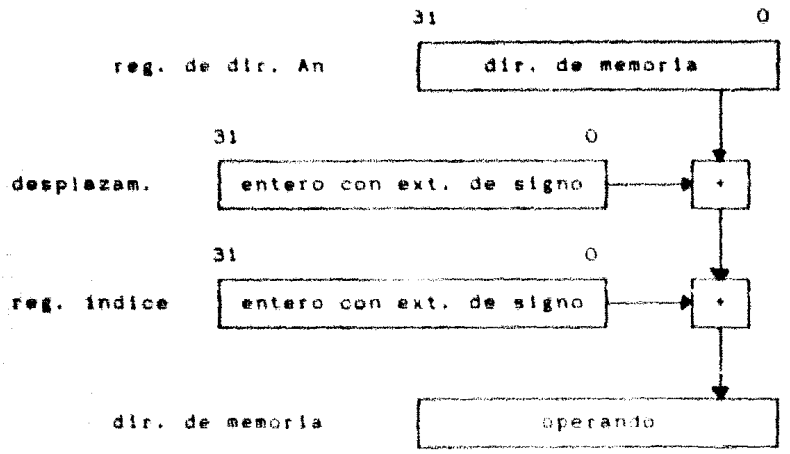
bit 11 : tamaño del indice
0 = en el registro indice hay un entero de orden inferior y con extensión de signo (W).
1 = en el registro indice hay un valor con formato de palabra larga (L).

La dirección del operando se forma sumando los tres siguientes elementos: la dirección contenida en el registro de dirección, el desplazamiento entero con extensión de signo (situado en los 8 bits de menor orden de la palabra de extensión), y el contenido del registro indice.

La referencia se clasifica como de datos, con excepción del salto (jump) y el salto a subrutinas. El tamaño del registro indice no afecta el tiempo de ejecución de las instrucciones.

Creación: EA = (An) + Xn + d8
Sintáxis: d8 (An, Xn.W) o (d8, An, Xn.W) indice de palabra
d8 (An, Xn.L) o (d8, An, Xn.L) indice de palabra larga
Modo: 110
Registro: n

MODOS DE DIRECCIONAMIENTO



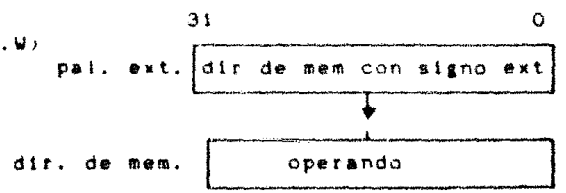
3. MODOS DE DIRECCIONAMIENTO ESPECIALES

Estos modos utilizan el campo del registro de dirección efectiva para especificar el modo de direccionamiento especial.

a) **Corto Absoluto.**

Este modo de direccionamiento necesita una palabra de extensión. La dirección del operando está en la palabra de extensión. A la dirección de 16 bits, se le extiende el signo antes de utilizarla. La referencia se clasifica como de datos, con la excepción del salto (jump) y el salto a una subrutina.

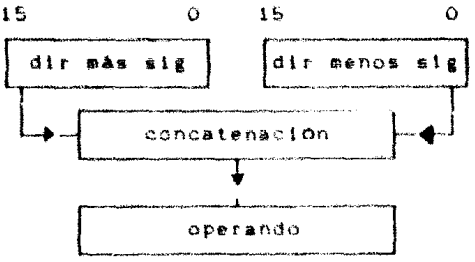
Creación: EA dado
Sintáxis: xxx.W o (xxx.W)
Modo: 111
Registro: 000



b) **Dirección Larga Absoluta.**

Este modo de direccionamiento necesita dos palabras de extensión. La dirección del operando se obtiene concatenando las palabras de extensión. La parte más significativa de la dirección es la primera palabra de extensión, la parte menos significativa es la segunda palabra de extensión. La referencia se clasifica como de datos, con la excepción del salto (jump) y del salto a una subrutina.

Creación: EA dado
Sintáxis: xxx.L o (xxx.L)
Modo: 111
Registro: 001



4. Direccionamiento Relativo del PC (Contador de Programa).

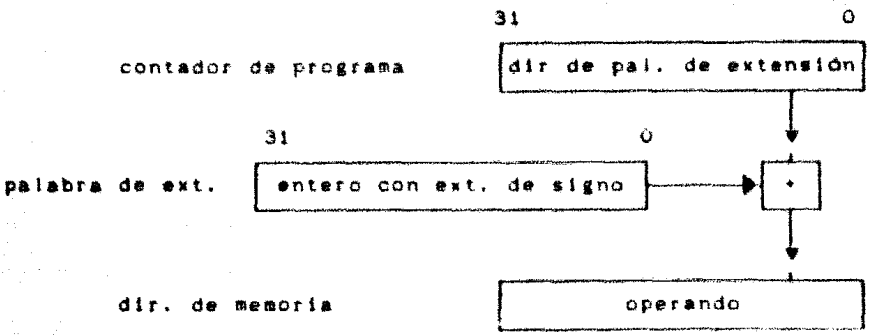
Estos direccionamientos toman como base el contenido del PC para poder crear la dirección de memoria.

a) **Contador de Programa con Desplazamiento.**

Utiliza una palabra de extensión. La dirección del operando es la suma de la dirección contenida en el PC y el desplazamiento entero de 16 bits con extensión de signo que se encuentra en la palabra de extensión. El valor contenido en el PC es la dirección de la palabra de extensión. La referencia se clasifica como de programa.

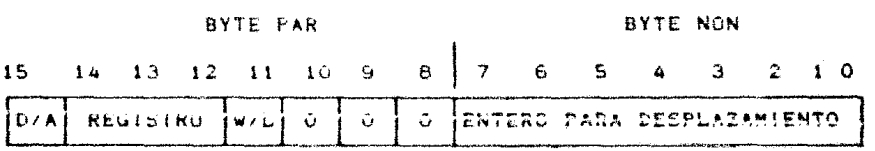
Creación: EA = (PC) + d16
Sintáxis: d16(PC) o (d16,PC)
Modo: 111
Registro: 010

MODOS DE DIRECCIONAMIENTO



b) Contador de Programa con Indice.

Este modo de direccionamiento necesita una palabra de extension, que cumpla con el siguiente formato:



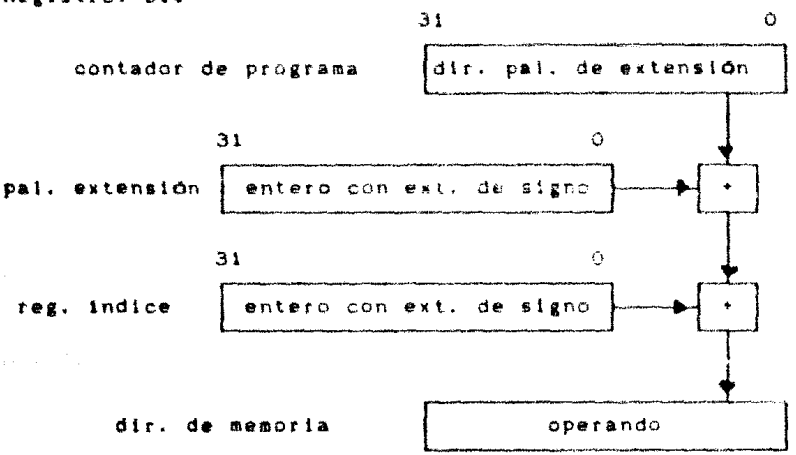
bits 15 : indicador del tipo de registro indice
0 = registro de datos (D)
1 = registro de direccion (A)

bits 14 al 12 : numero del registro indice

bit 11 : tamaño del indice
0 = en el registro indice hay un entero de orden inferior y con extensión de signo (W).
1 = en el registro indice hay un valor con formato de palabra larga (L).

La dirección es la suma de la dirección contenida en el contador de programa, más el numero entero del desplazamiento con extensión de signo que se encuentra en los ocho bits menos significativos de la palabra de extensión, más el contenido del registro índice. El valor que reside en el contador de programa es la dirección de la palabra de extensión. Esta referencia se clasifica como de programa. El tamaño del registro índice no afecta el tiempo de ejecución de la instrucción.

Creación: EA = (PC) + (Xn) + d8
Sintáxis: d8(PC,Xn.W) o (d8,PC,Xn.W) índice de palabra
d8(PC,Xn.L) o (d8,PC,Xn.L) índice pal. larga
Modo: 111
Registro: 011



5. Direccionamiento Inmediato de Datos.

Este tipo de direccionamiento permite que los datos sigan inmediatamente a la palabra de instrucción.

MODOS DE DIRECCIONAMIENTO

a. Inmediato.

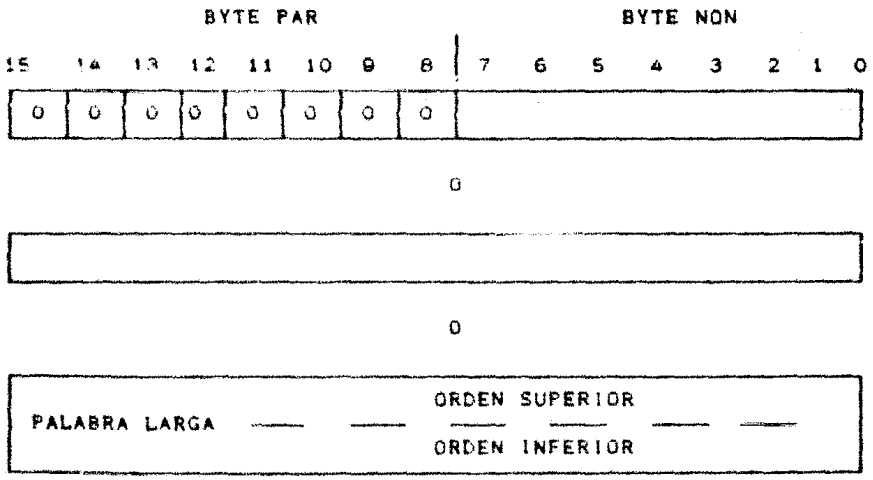
El operando es el valor que sigue después de la palabra de la instrucción. Dependiendo del tamaño de la operación, se necesitarán una o dos palabras de extensión.

Operación de 1 byte - el operando es el byte menos significativo de la palabra de extensión.

Operación de 1 palabra - el operando es la palabra de extensión.

Operación de 1 palabra larga - el operando ocupa las dos palabras de extensión, los 16 bits más significativos están en la primera palabra de extensión, y los 16 bits menos significativos están en la segunda.

Creación: operando dado
Sintáxis: # xxxx o # <dato>
Modo: 111
Registro: 100



b) **Inmediato Rápido.**

Este modo sólo es una pequeña variante del modo inmediato, consiste en que el dato está en la misma palabra de la instrucción. Por esta razón ya no se necesitarán palabras de extensión para el dato. La creación, la sintaxis, el modo y el registro son los mismos que para el direccionamiento inmediato.

6. Direccionamiento Implícito.

Hay algunas instrucciones que hacen referencia en forma implícita a algunos registros específicos. Estos registros son los siguientes: contador de programa (PC), apuntador a la pila (SSP o USP) y el registro de estados (SR).

a) **Registro Implícito.**

A continuación se presenta una tabla con las instrucciones en las cuales se hace referencia a un registro que contiene al operando:

Instrucción	Registro(s) Referido(s)
Bcc, BRA	PC
BSR	PC, SP
CHK	SSP, SR
DBcc	PC
DIVS, DIVU	SSP, SR
JMP	PC
JSR, LINK	PC, SP
MOVE CCR	SR
MOVEC	VBR, SFC, DFC
MOVES	SFC, DFC

Instrucción	Registro(s) Referido(s)
MOVE SR	SR
MOVE USP	USP
PEA	SP
RTD, RTS	PC, SP
RTE, RTR	PC, SP, SR
TRAP, TRAPV	SSP, SR
UNLK	SP
lógica inmediata a CCR, a SR	SR

EXCEPCIONES

El **"procesamiento de excepciones"** en el 68000 se refiere al sistema de interrupción de otros microprocesadores. Se le denomina "excepcion" porque los eventos que generan interrupciones en el CPU, están originados por estados no comunes (excepcionales) en el procesador.

El estado de procesamiento de excepciones está relacionado con interrupciones, instrucciones de "trap", rastreo, y otras condiciones. Por esta razon, las excepciones abarcan un rango más amplio de eventos.

La lógica de procesamiento de excepciones utiliza una tabla de vectores de salto para transferir el control del programa al programa manejador apropiado cuando ocurra una excepción.

La excepción puede ser **generada internamente** por una instrucción o por una condición inusual surgida durante la ejecución de una instruccion.

El **procesamiento externo** de la excepción puede ser forzado por una interrupción, por un error del canal (bus error), o por un reinicio (RESET). El procesamiento de excepciones está diseñado para proveer un cambio de contexto eficiente para que el procesador pueda manejar condiciones inusuales.

El procesador trabaja en **dos estados : usuario y supervisor**. En cada estado se determina cuales operaciones son legales, es decir, con cuales privilegios pueden contar. El modo de operación se define en el bit S (estado supervisor) del registro de estados, si S=1 se está trabajando en modo supervisor, pero si S=0 entonces el 68000 está en modo usuario. El dispositivo externo de manejo de memoria utiliza estos estados para controlar y trasladar accesos a memoria. Estos estados también definen cual apuntador a la pila se utilizará, (usuario o supervisor), en cualquier instrucción que se utilice este tipo de apuntador.

Cuando se reinicia al 68000 con RESET (de software), el procesador comienza a funcionar en modo supervisor. El CPU permanece en este modo hasta que se ejecuta una de las siguientes instrucciones:

RTE - retorno de la excepción
MOVE - transferir hacia el registro de estado
ANDI - "and" inmediato con el registro de estado

EORI - "or exclusivo" inmediato con el registro de estado

Con estas instrucciones el 68000 no se cambia automáticamente al modo usuario. Primero se cambia el estado del bit S (estado supervisor) de 1 a 0 del registro de estados, y después el 68000 ya comienza a operar en modo usuario.

Durante el modo usuario, sólo el procesamiento de excepciones puede hacer que se cambie al modo supervisor. En este procedimiento, el estado actual se guarda en el bit S, del registro de estados y el procesador cambia al modo supervisor. Cuando termina la ejecución de la instrucción, en la dirección especificada por el procesamiento de la excepción, el procesador permanece en modo supervisor.

El **procesamiento de una excepción** ocurre en **cuatro etapas**, con variantes dependiendo de las causas que originen a las excepciones.

En la **primera**, se hace una copia temporal del registro de estados para que éste registro, esté preparado para el procesamiento de la excepción. El registro de estados es el siguiente:

BYTE DEL SISTEMA							BYTE DEL USUARIO					
15		13		10	9	8		4	3	2	1	0
T		S		I2	I1	I0		X	N	Z	V	C

T - modo de rastreo (traza)
S - modo supervisor
I - mascara de interrupción

X - extensión
N - negativo
Z - cero
V - sobreflujo
C - acarreo

En la **segunda** etapa, se determina el vector de excepciones, y durante la **tercera** se almacena el contenido actual del procesador.

En la **cuarta** y ultima etapa, se obtiene un nuevo contexto y el procesador se cambia al procesamiento de instrucciones.

EXCEPCIONES

Los **vectores de excepciones** son localidades de memoria en las cuales el procesador busca la dirección de la rutina que pueda manejar cierta excepción. Estos vectores tienen dos palabras de longitud, excepto por el del reinicio (reset) que tiene cuatro. Los vectores de excepciones permanecen en el área de datos del supervisor, pero la excepción de reinicio permanece en el modo de programación (RESET).

Existe un **identificador numérico para cada vector de excepciones**. Este numero tiene 8 bits, que al multiplicarlos por cuatro dan el desplazamiento de la excepción dentro del vector. Estos números se crean externa o internamente dependiendo de la causa de la excepción.

El procesador forma el **desplazamiento del vector**, moviendo el numero binario del vector dos bits a la izquierda, y llenando con ceros los bits de orden mayor hasta que el vector tenga 32 bits. Este desplazamiento se utiliza como la dirección absoluta para obtener el vector de excepción.

La memoria tiene 512 palabras de longitud y hay 255 vectores únicos, algunos están reservados para TRAPS y otras funciones del sistema. Del total de vectores, hay 192 reservados para interrupciones del usuario. Sin embargo, no hay protección para las primeras 64 entradas, por lo tanto los vectores de interrupciones del usuario pueden traslaparse. A continuación se muestra la tabla de vectores de interrupción.

Num. de Vector	Dirección Dec	Dirección Hex	Area	Asignación
0	0	000	SP	reinicio: SSP inicial
1	4	004	SP	reinicio: PC inicial
2	8	008	SD	error de canal
3	12	00C	SD	error de instrucción
4	16	010	SD	instrucción ilegal
5	20	014	SD	división entre cero
6	24	018	SD	instrucción CHK
7	28	01C	SD	instrucción TRAPV
8	32	020	SD	violación de privilegio

NOTA: SP=área del programa supervisor, SD=área de datos del supervisor.

EXCEPCIONES

Num. de Vector	Dirección Dec	Hex	Area	Asignación
9	36	024	SD	traza
10	40	028	SD	emulador linea 1010
11	44	02C	SD	emulador linea 1111
12-14	48-56	030-038	SD	reservado para Motorola
15	60	03C	SD	vec int no inicializado
16-23	64-92	040-05C	SD	reservado para Motorola
24	96	060	SD	interrupción falsa
25	100	064	SD	autovector de- nivel 1
26	104	068	SD	interrupción nivel 2
27	108	06C	SD	" nivel 3
28	112	070	SD	" nivel 4
29	116	074	SD	" nivel 5
30	120	078	SD	" nivel 6
31	124	07C	SD	" nivel 7
32-47	128-188	080-0BC	SD	vectores de TRAP
48-63	192-255	0C0-0FF	SD	reservado para Motorola
64-255	256-1020	100-3FC	SD	vec de interrup usuario

NOTA: SP=área del programa supervisor, SD=Área de datos del supervisor.

Las **excepciones generadas externamente** son las interrupciones, el error en el canal (BERR) y el reinicio (RESET). Las interrupciones son acciones que los dispositivos periféricos piden al procesador. El BERR y el RESET son utilizados para control de acceso y reinicio del procesador.

Las **excepciones internas** provienen de instrucciones, de errores de direcciones o de rastreo. Las siguientes instrucciones generan excepciones, como parte de la ejecución: **TRAP**, **TRAPV** (trapoverflow), **CHK** (compara al registro con bounds). Las instrucciones ilegales, la busqueda de palabras en direcciones nones y las violaciones de los dos estados (usuario y supervisor) causan excepciones.

El **TRACE** se comporta, al final de la ejecución de una instrucción, como una interrupción de alta prioridad generada internamente.

EXCEPCIONES

Las **excepciones** se clasifican en **tres grupos**:

El grupo 0 es el de **mayor prioridad**, con: **RESET, BERR** y error de dirección. En estos casos se aborta el ciclo actual y después se procesa la excepción.

El grupo 1 tiene **prioridad media** con: **trace** (rastreo), interrupción externa, instrucción ilegal y violación de privilegio, siguiendo este orden de jerarquía. Se termina la instrucción actual y se procesa la excepción.

El grupo 2 con la **menor prioridad**, tiene las excepciones internas que se presentan durante la ejecución normal de una instrucción. La ejecución de la instrucción inicializa el procesamiento de la excepción.

La **relación entre las prioridades** determina el orden en el cual se efectua las excepciones, en caso de presentarse **simultáneamente**.

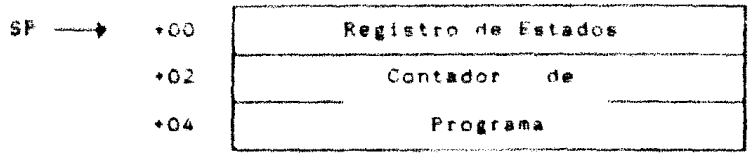
Dependiendo del procesador y del tipo de excepción, el procesador puede insertar de 0 a 44 palabras de datos en la pila (stack) del supervisor, como parte de la secuencia del procesamiento de excepciones. Este bloque de datos se llama **estructura pila**.

A continuación se presenta un formato general de la estructura pila de excepciones:

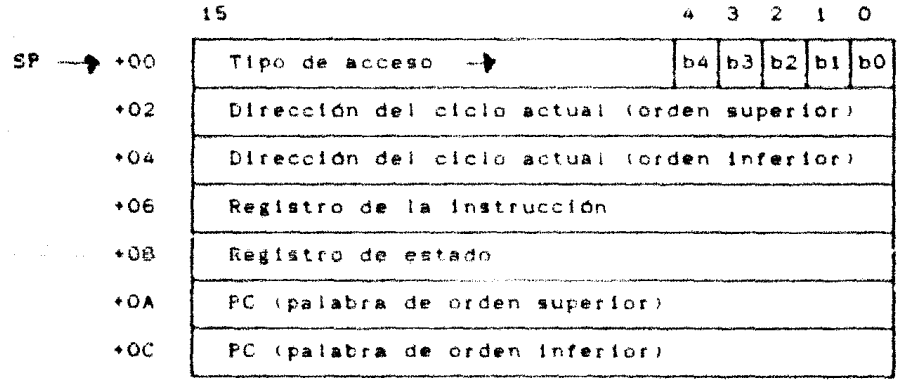
FORMATO	TIPO
0000	Formato corto (4 palabras)
0001	Relleno (4 p.)
0010	Excepción de instrucción (6 p.)
0011-0111	Reservado
1001	Instr. de mediación con procesador (10)
1100-1111	Reservado

EXCEPCIONES

A continuación se muestra la **estructura pila** del 68000 creada por: TRACE, TRAP, instrucción ilegal o no existente y violación de los privilegios o peticiones de interrupción.



En la siguiente figura se muestra la **estructura pila** del 68000, creada por una excepción del canal o por un error de dirección:



Los valores para los bits menos significativos, de esta esctructura pila, son:

b4 : 0= ciclo de escritura cancelado
1= ciclo de lectura cancelado

b3 : 0= instrucción en progreso
1= procesamiento de la excepción

b2 - b0 : código de la función

EXCEPCIONES

Cada una de las excepciones sigue la misma **secuencia general de eventos** cuando comienza. A continuación se presenta esta secuencia:

1. La excepción hace una copia interna del registro de estado y prende el bit del supervisor del registro de estados, inicializa en cero los bits del TRACE, y para el caso de las interrupciones modifica la máscara de éstas.

2. Determina la entrada a la tabla de vectores, ya sea para leer de ella un dispositivo de interrupción o para utilizar los números de entrada fijos asociados con otros tipos de procesamiento de excepciones.

3. Inserta el dato correspondiente en el stack del supervisor. El tipo de datos apilados depende tanto del tipo de excepción como de la clase de procesador de la familia del 68000.

4. Carga el PC (contador del programa) con la dirección procedente de la tabla de vectores y comienza la ejecución.

ARQUITECTURA DE LA PC

INTRODUCCION

Una computadora personal o PC (Personal Computer) es una computadora de bajo costo, en comparación a los demas sistemas de cómputo. La PC tiene gran demanda porque su software (programas del sistema y de aplicación) es compatible, en un alto grado, entre la mayoria de las computadoras personales. Es decir, el mismo software se puede utilizar en varias PCs sin ningun problema de formato, lenguaje, etc.

Desde el punto de vista de la arquitectura de la PC, los componentes que la forman, son circuitos integradados fabricados a gran escala, lo cual permite que la PC tenga dimensiones relativamente pequeñas. Esta caracteristica da origen a que la computadora personal sea portátil, controlable por el usuario y de fácil manejo. A continuación se tratara algunos de los componentes fisicos mas importantes de la PC.

MICROPROCESADOR 8088

El microprocesador es la unidad central de procesamiento (CPU) y uno de los principales componentes de la computadora. En el se controla y monitorea todos los procesos de la PC:

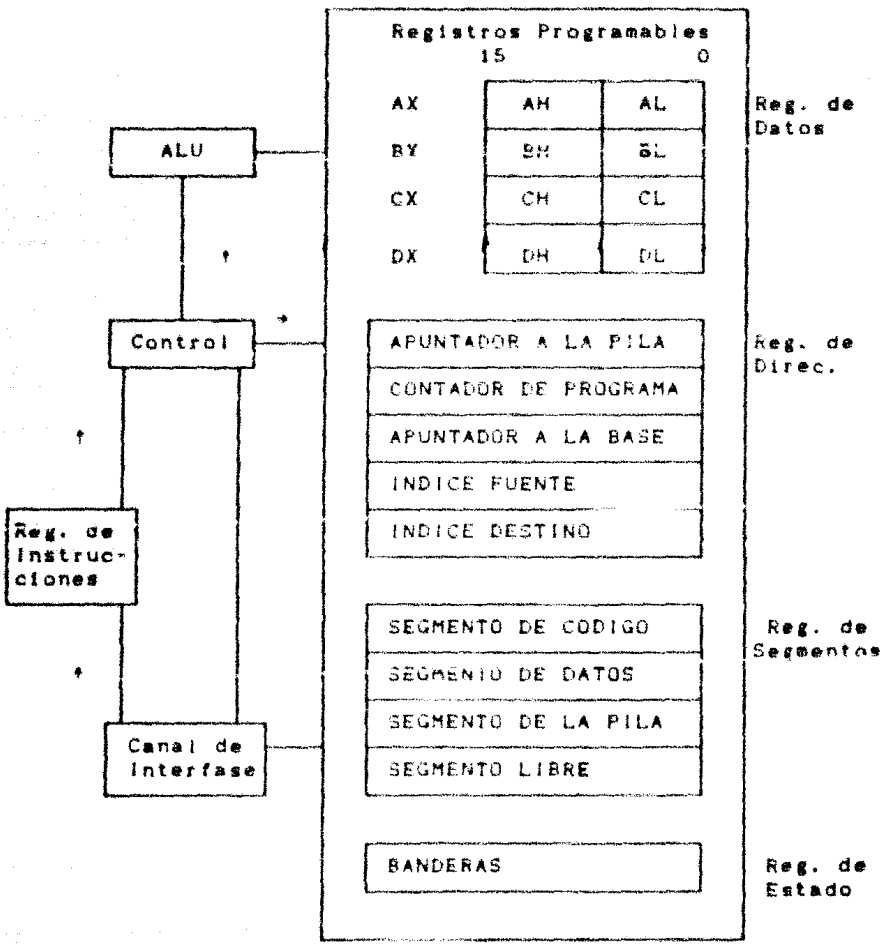
- registro de instrucciones
- decodificación de instrucciones
- interpretación y ejecución de instrucciones
- temporización y control de actividades
- realización de operaciones aritmético lógicas
- manejo de memoria
- manejo del canal de E/S
- control de dispositivos periféricos.

El microprocesador 8088 está diseñado con una arquitectura interna de 16 bits y una via externa de datos de 8 bits, que va a la memoria y al canal de entrada y salida.

El 8088 tiene un complemento de ocho registros generales de 16 bits cada uno. Los registros generales están subdivididos en dos grupos de cuatro registros: registros de datos y registros indice y apuntadores. En cada uno de los registros de datos se puede accesar, en forma independiente, la mitad superior o la inferior. Por lo tanto, cada registro de datos se puede utilizar como uno de 16 bits o dos registros de 8 b.

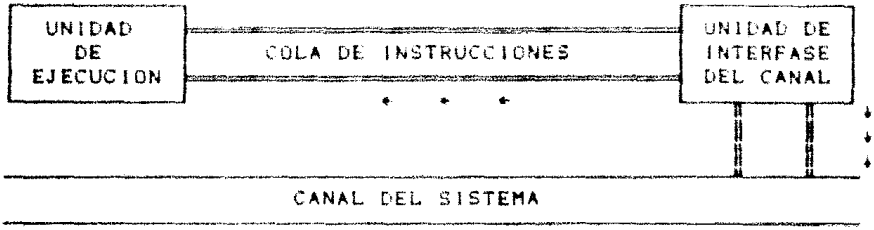
ARQUITECTURA DE LA PC

El 8088 accesa a la memoria a través de arreglos de bytes. Así, se puede almacenar instrucciones, bytes de datos y palabras de datos en cualquier dirección de memoria.
A continuación se muestra un diagrama a bloques del microprocesador 8088.



El CPU del 8088 utiliza una arquitectura de "pipeline" o de cola de espera. Esta consiste en dos unidades, la del canal de interfase (Bus Interface Unit, BIU) y la unidad de ejecución (Execution Unit, EU).
El BIU captura instrucciones adicionales, mientras que el EU ejecuta las instrucciones que ya se habían obtenido anteriormente.

Esto crea un efecto de cola entre las dos unidades. El BIU llena esta cola con las cuatro siguientes instrucciones secuenciales, en espera de ser llevadas a cabo. Al realizar esto, se decrementa el tiempo perdido en el canal, ya que las instrucciones están listas para su ejecución inmediata sin retrasos causados por la búsqueda de las instrucciones.



COPROCESADOR ARITMETICO

El coprocesador aritmético 8087 es opcional. Es un circuito integrado que tiene capacidad para manejar registros, tipos de datos, control e instrucciones a nivel de hardware. El 8087 también ejecuta funciones e instrucciones intrínsecas.

El coprocesador aritmetico está conectado en paralelo al CPU. Las líneas de estado del CPU habilitan al 8087 para monitorear y decodificar instrucciones en sincronía con el CPU, ahorrándole así, tiempo y trabajo al procesador. El 8087 puede procesar en paralelo con el CPU, o independientemente de éste.

El 8087, está subdividido en dos elementos procesadores, la Unidad de Control (UC) y la Unidad de Ejecución Numérica (UEN). La UEN ejecuta todas las instrucciones numéricas, y la UC recibe y decodifica instrucciones, lee y escribe operandos en memoria, y ejecuta instrucciones de control. Ambos elementos operan independientemente uno del otro, así la UC mantiene sincronía con el CPU 8088, mientras que la UEN procesa instrucciones numéricas.

CONTROLADOR DE ACCESO DIRECTO A MEMORIA (DMA)

El controlador de DMA se localiza en la tarjeta del sistema, permitiendo así que haya transferencia a gran velocidad de bytes, de programas y datos, entre la memoria y los periféricos de entrada y salida. El controlador de DMA tiene 4 canales independientes, por lo tanto el sistema puede realizar hasta 4 transferencias de bloques al mismo tiempo. El sistema utiliza el canal 0 para refrescar (refresh) la RAM dinamicamente. Los canales 1 y 3 están a disposición del sistema, y se encuentran en el canal de expansión para conexión de tarjetas. El canal 2 se utiliza para transferencias de acceso directo a memoria de los discos flexibles.

El circuito del DMA se puede operar en tres modos:

1) byte por byte - el control se regresa al procesador después de cada ciclo de un byte.

2) explosión (burst) - la operación continua mientras que los puertos esten listos.

3) continuo - el procesador no recobra el control hasta que la operación de transferencia de datos haya concluido.

CONTROLADOR PROGRAMABLE DE INTERRUPCIONES (PIC)

El controlador programable de interrupciones (8259A) puede manejar hasta 8 interrupciones con prioridad vectorizada, dirigidas al CPU. Está diseñado para minimizar

el software y el costo del tiempo real (overhead), al aplicar interrupciones con prioridad de multinivel.

El PIC funciona como un administrador general en un ambiente, cuyo sistema trabaja a base de interrupciones. El PIC acepta peticiones de los dispositivos periféricos, determina cual de las solicitudes que recibe tiene más importancia, evalua si alguna petición tiene mayor prioridad que el nivel que se está atendiendo en ese momento, y manda una interrupción al CPU, basándose en esa determinación.

Cada dispositivo periférico tiene comúnmente un programa especial o rutina que está asociada a sus requerimientos funcionales u operacionales específicos. A esto se le llama "rutina de servicio". Después de mandar una interrupción al CPU, el PIC debe darle información al CPU para que el Contador de Programa apunte a la rutina de servicio, asociada con el dispositivo que hizo la petición.

El 8259A es programado como periférico de E/S, por la rutina de inicialización del sistema.

Existe una selección de modos de prioridad, disponible para el usuario, con el fin de poder adaptar la forma en la que se procesan las peticiones, con los requerimientos del sistema. Se puede reconfigurar los modos de prioridad dinámicamente, durante la operación. Esto significa que la estructura completa de interrupciones se puede definir como se necesite, basándose en los requisitos del sistema.

A continuación se muestra una lista de líneas de petición de interrupción (IRQ), conectadas a dispositivos periféricos, en la tarjeta principal del sistema:

- IRQ0 sistema
- IRQ1 interfase del teclado
- IRQ2 disponible
- IRQ3 interfase de comunicación, COM2
- IRQ4 interfase de comunicación, COM1
- IRQ5 disponible
- IRQ6 interfase del drive de disco flexible
- IRQ7 interfase de la impresora

RELOJ / CONTADOR

El reloj/contador 8253 resuelve el problema de generar retrasos exactos de tiempo, bajo el control de software. El 8253 tiene 3 contadores independientes, de 16 bits cada uno.

Estos contadores son programados individualmente para que operen en uno de seis modos disponibles.

El canal (contador) 0 es programado como generador de tasa (rate) para temporizar el refresco (refreshing) de la RAM dinámica.

El canal 1 provee una interrupción para funciones de tiempo en el sistema (reloj de tiempo real, "timeout", etc.).

El canal 2 es programado como un generador de tonos. El valor de la cuenta determina la frecuencia de la señal de salida de la bocina.

INTERFASE PROGRAMABLE DE PERIFERICOS (PPI)

Hay una interfase programable de periféricos 8255A, que sirve para implantar puertos paralelos de propósito general, para uso del sistema. El 8255A consta de 24 terminales de E/S, los cuales pueden ser programados individualmente, en 2 grupos de 12, y ser operados en 3 modos principales.

El PPI contiene 3 puertos de 8 bits (A, B y C), se pueden configurar bajo diversas caracteristicas funcionales, con el software del sistema.

Puerto A - enclavamiento/buffer de salida de 8 bits enclavamiento de entrada de datos de 8 bits.

Puerto B - enclavamiento/buffer de E/S de datos de 8 bits buffer de entrada de datos de 8 bits.

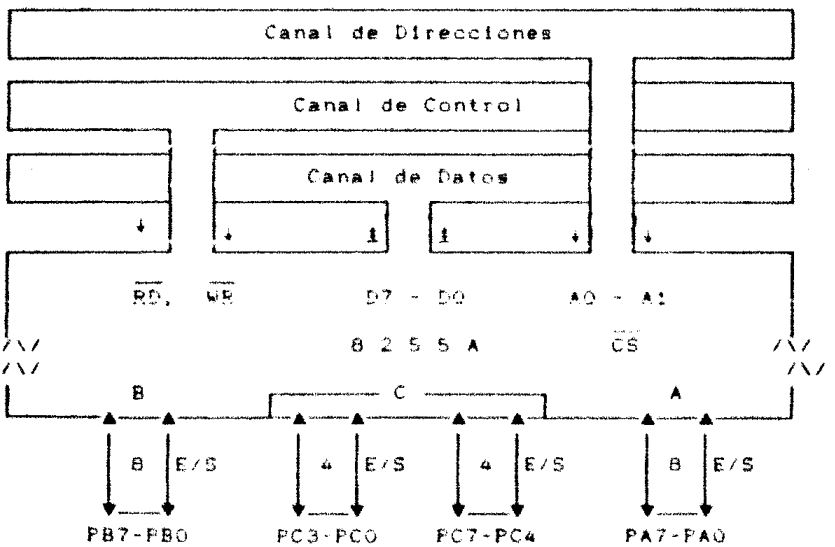
Puerto C - enclavamiento/buffer de salida de datos de 8 bits buffer de salida de datos de 8 bits.

A continuacion se va a describir brevemente los 3 modos de operación. También se ilustrará cada uno de estos modos para los puertos A, B y C.

Modo 0 (Entrada/Salida Básica).

Cada grupo de 12 terminales de E/S puede programarse en grupos de 4, para funcionar como entrada o salida. Esta configuración provee un medio de transferencia de datos de E/S de un puerto especifico, o hacia él.
Estas transferencias están controladas por medio de señales de selección estroboscópica (strobe) o de intercomunicación (handshake).

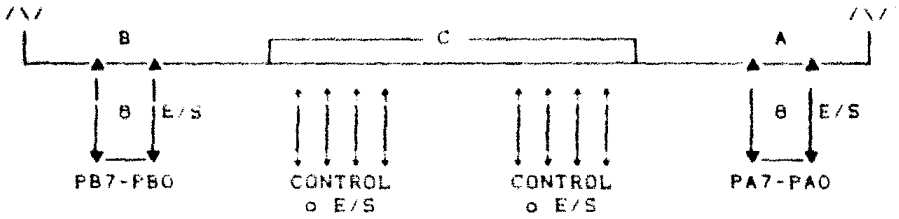
Su descripción funcional es la siguiente:
- hay dos puertos de 8 bits y dos puertos de 4 bits
- cualquier puerto puede ser configurado para entrada o salida
- las salidas se enclavan, pero las entradas no.



Modo 1 (Entrada/Salida con señal de seleccion estroboscópica, Strobe).

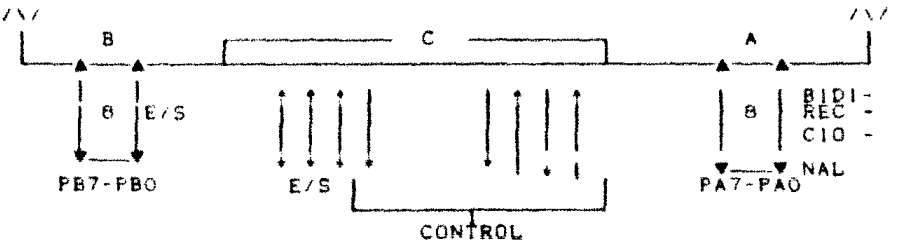
Cada grupo de 12 terminales puede ser programado quedando 8 líneas de entrada o de salida. El puerto A y el B ocupan las líneas del puerto C para generar o aceptar señales de control de interrupciones o de enlace (handshaking).
Descripción funcional:

- hay dos grupos de terminales (A y B)
- cada grupo contiene un puerto de datos de 8 bits, y un puerto de control y datos de 4 bits
- el puerto de 8 bits puede ser de entrada o salida; las entradas y salidas se enclavan
- los puertos de 4 bits se usan para control y estado.



Modo 2 (E/S con Canal Bidireccional y Señal Estroboscópica).

El puerto A y una sección del puerto C se programan como un canal bidireccional de 8 bits y 5 líneas de enlace (handshake). Las 11 líneas restantes pueden ser programadas como Modo 1 o Modo 0.



SEÑAL	DESCRIPCION	SEÑAL	DESCRIPCION
RESET	inicializa la entrada	PA7-0	puerto A
A0-A1	direccion del puerto	PB7-0	puerto B
D7-D8	canal bidirec. de datos	PC7-0	puerto C
*CS	selecc. circuito (chip select)		
*WR	escribe entrada		
*RD	lee entrada		

* : se habilita en bajo

CONTROLADOR DE COMUNICACIONES

Dos elementos de comunicación asincrona (ACE) INS8250, funcionan como interfases de E/S de datos en serie del sistema. La configuración funcional del ACE se programa por medio de software del sistema.
Los INS8250 realizan conversiones serie a paralelo en caracteres de datos recibidos de dispositivos periféricos o de MODEMs. Tambien pueden convertir de paralelo a serie caracteres recibidos del CPU.

El CPU puede leer el estado de los INS8250 en cualquier momento durante la operación funcional. La información de su estado, que se reporta, incluye el tipo y la condicion de las operaciones de transferencia que esté realizando el ACE, así como cualquier error de condición (paridad, punto de ruptura, desbordamiento, o enmarcamiento).

Los INS8250s se conectan al dispositivo serie, por medio de conectores DB25S, que se encuentran atras de la computadora.

Ademas de controlar la comunicacion asincrona de datos, el INS8250 incluye un Generador de Bauds programable, que puede dividir las entradas de referencia del reloj entre 1 hasta 65,535, para generar un reloj de 16x, el cual se utiliza para dirigir la lógica de transmisión interna. También se utiliza este reloj de 16x para dirigir la lógica del receptor. El INS8250 puede llevar a cabo el control de MODEMs, y cuenta con un sistema de interrupciones al procesador, el cual se puede modificar por medio de software, según los requerimientos del usuario, para minimizar el tiempo de cómputo que se necesite para manejar la liga de comunicaciones.

DESCRIPCION FUNCIONAL DE LA UNIDAD DEL DISCO FLEXIBLE

La unidad de disco no necesita de ninguna intervención externa de operación para funcionar adecuadamente bajo condiciones normales. La unidad de disco consiste de un sistema de unidad giratoria, un sistema de posicionamiento de cabeza (lectora), y un sistema para borrar, escribir y leer.

Cuando se sube la manija de la escotilla (parte frontal de la computadora), se puede introducir el disco flexible. El disco se introduce en la unidad, por medio de guias y de la manija. Al bajar la manija se activa el sistema de centro y prensa, el cual centra el disco y lo sujeta al eje de la unidad. El eje rota a velocidad constante de 300 rpm, por medio de un motor DC de servo-control. En seguida, la cabeza magnética se posiciona sobre el disco (sin tocarlo).

Una pista es una división concéntrica del disco; un sector es un area del disco que está delimitada por dos fronteras que irradian del centro a la orilla del disco. La cabeza se coloca sobre la pista deseada por medio de un motor/banda paso a paso. Este motor utiliza una rotación de un paso para generar un movimiento lineal de una pista.

La unidad de disco incluye los siguientes sistemas sensoriales:

1. Un interruptor de la pista 00, el cual sensa cuando la cabeza se posiciona en esta pista.

2. Un sensor de indice, este consiste en un diodo de luz y un fototransistor, estos elementos generan una señal cuando detectan el agujero indice del disco. Esta perforación hace referencia al inicio del disco.

3. Un sensor de protección contra escritura deshabilita los dispositivos correspondientes, cuando se le aplica al disco una marca contra escritura.

La unidad de disco utiliza discos de doble densidad, si se utiliza de densidad sencilla, se puede almacenar información con error.

CONTROLADOR DE DISCO FLEXIBLE

El formateador y controlador de disco flexible, UPD765, es el enlace entre el procesador y la unidad de disco flexible. Este controlador realiza todas las funciones necesarias para leer o escribir datos en el disco flexible. Entre otras actividades, realiza: lectura de sectores múltiple o individual, con búsqueda automática, y lectura completa de pistas; escritura múltiple o individual de sectores con busqueda de sectores; búsqueda automática de pistas con verificación; y selección de tiempo, por medio de programa, para controlar el movimiento de pista a pista de la cabeza.

UNIDAD DE DISCO DURO TIPO WINCHESTER

El sistema es configurado opcionalmente con una unidad de disco duro de 5 1/4 " . Se le llama Winchester a un tipo de memoria de acceso aleatorio, de bajo costo, el cual utiliza una cabeza movible, con técnica de grabación sin contacto directo con el disco. La unidad consiste de un medio de almacenamiento fijo, dispositivos electrónicos para lectura, escritura y control, una cabeza de lectura y escritura, un implemento para controlar la posición de la cabeza, y un sistema de filtración de aire.

CONTROLADOR DE DISCO DURO

El controlador de disco duro puede controlar la operación de hasta dos discos duros tipo Winchester. El controlador emplea la interfase de disco duro ST506. La interfase con el sistema se implanta a través de una de las 62 terminales de la ranura de expansión de la computadora. Todas las transferencias de datos se manejan a través del DMA del sistema y se inicializan directamente por medio del controlador de disco duro.

SIMULACION DE SISTEMAS

Introducción

Actualmente la simulación de sistemas cobra gran importancia en la ingeniería dado que influye, en buena medida, en diferentes aspectos de la vida diaria. Este tema es extenso, porque mediante la simulación se pueden analizar diferentes fenómenos que van desde el comportamiento de un átomo hasta el vuelo de una nave espacial. Con motivo de dar a entender mejor las ideas principales de este capítulo se estudiarán antes conceptos relacionados con la simulación, agrupándolos en tres partes fundamentales:

1) conceptos acerca de sistemas,

2) conceptos sobre modelos,

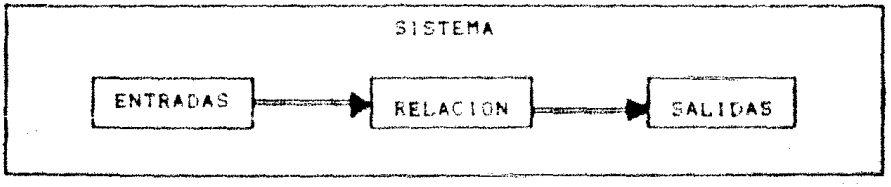
3) conceptos propiamente de la simulación de sistemas.

Se hace énfasis en el último tema, porque es el más importante, para los fines de este estudio.

Conceptos Generales Acerca de los Sistemas

A un conjunto de componentes reunidos bajo alguna interacción, con el fin de cumplir un determinado objetivo, se le denomina sistema. Cabe señalar que esta definición de sistema no se limita a los objetivos físicos; el concepto de sistema puede ser aplicado a fenómenos abstractos y dinámicos.

También se considera que un sistema es un ente formado por un grupo de entradas (insumos) y otro de salidas (producto final), con una relación que une a ambos. Lo cual conduce a entender que un sistema es un ente que relaciona o mapea a un conjunto de entradas sobre un conjunto de salidas, mediante una función bien definida.



Con base en lo anterior, Geoffrey Gordon clasificó los elementos que conforman a un sistema cualquiera, obteniendo las siguientes definiciones:

Entidad - es un objeto de interés dentro del sistema.

Atributo - es una propiedad de una entidad.

Actividad - es cualquier proceso que realice cambios en el sistema.

Estado del sistema - describe la situación de las entidades, atributos y actividades del sistema, en cierto tiempo requerido. Estudiando los cambios continuos en el estado del sistema se podrá observar el comportamiento del sistema bajo cualquier situacion.

En lo que concierne al microprocesador 68000, se puede decir que se le denomina sistema porque es un ente que está formado por un conjunto de rutinas. Este grupo de rutinas de servicio secuencia programas a través de una microcomputadora, también proporciona subrutinas de conversión, entrada-salida y depuración, y hace observaciones de utilidad para el operador. Si se aplican los conceptos de G. Gordon al sistema 68000, se tendría entonces que algunos de los elementos que lo forman son los siguientes:

Entidades: el CPU, los registros, los canales de datos y de control.

Atributos: la velocidad de respuesta, el tamaño de los registros.

Actividades: mover un dato de un registro a otro, realizar alguna operación lógica o aritmética.

Estado del sistema: generalmente se conoce al verificar el contenido de los registros de interés.

Despues de clasificar a los elementos de un sistema se procede a clasificar a los sistemas mismos, en forma global. Primero se les definira segun el tipo de actividades que realicen:

Un sistema puede llevar a cabo solo actividades internas o endogenas, sin tener relación alguna con su exterior; pero si opera con su medio ambiente, entonces sus actividades se denominan externas o exógenas.

Un sistema cerrado únicamente ejecuta actividades endógenas, en cambio, un sistema abierto tambien realiza actividades exogenas.

Cuando la salida del sistema, es decir su producto final, depende directamente de las entradas del sistema, entonces se dice que la actividad es determinista. Sin embargo, si una entrada origina salidas aleatorias, o salidas que dependen del tiempo, entonces su actividad es estocàstica.

Si se describe al 68000 empleando las caracteristicas antes mencionadas, entonces se puede decir que este microprocesador funge como un sistema abierto, porque puede interactuar con el usuario (medio ambiente), y que realiza actividad de tipo deterministico, porque para un cierto grupo de entradas invariables siempre se obtendrán las mismas salidas. Es decir si el 68000 ejecuta un mismo grupo de instrucciones varias veces, y maneja los mismos datos, entonces cada vez generara respuestas iguales sin que éstas cambien en forma aleatoria o que esten en función del tiempo en el cual se ejecutaron.

Después de describir a un sistema por el tipo de actividad que realiza, también es posible identificarlo por el tipo de salida que genera. A continuación se presenta una clasificación de los sistemas basándose en sus salidas en general:

Causal - este tipo de sistema se basa en el principio de que todo efecto siempre es el resultado de una causa. Esto significa que para que la salida producida por un sistema cambie de un estado a otro se requiere que la entrada aplicada a éste cambie con anterioridad. Por lo tanto las diferentes salidas obtenidas son reflejo de las entradas al sistema, como en el caso del 68000.

No Causal - las salidas varian, no importando cuales hayan sido las entradas o cuando se hayan presentado.

Dinámico - la salida en cierto tiempo depende de la entrada aplicada en ese mismo momento y de todas aquellas que se hayan presentado anteriormente. Tal es el caso del 68000, puesto que la salida final depende de todas las entradas que generen una secuencia de cambios de estado en sus registros o en las localidades de memoria que accese.

Estático - la salida en algún tiempo sólo depende de la entrada que se presente en ese preciso momento, sin tomar en cuenta las entradas que se hayan presentado anteriormente.

Determinástico - a una misma entrada le corresponde unicamente una salida; no importa si la misma entrada se repite varias veces, la salida seguirá siendo la misma, ya que no está en función del tiempo. Esto sucede en el 68000 porque si le aplicamos un mismo grupo de instrucciones y de datos, entonces dara lugar a una misma salida, sin depender del tiempo en el cuál se haya ejecutado.

Estocástico - una misma entrada puede originar diferentes salidas aleatorias, o las salidas pueden estar en función del tiempo.

Parámetros Concentrados - existe un numero finito de variables que intervienen en el sistema, asi es en el 68000.

Parametros Distribuidos - el comportamiento de un sistema está regido por un numero infinito de variables.

Lineal - los sistemas son descritos por medio de modelos matemáticos lineales, y generalmente (no siempre) se emplean ecuaciones en derivadas parciales para representarlos.

No Lineal - estos sistemas no cumplen con el principio de superposición, el cual define que la salida producida por varias entradas es igual a la suma de las salidas cuando se presentan las entradas individualmente. El sistema se representa por medio de ecuaciones en diferencias. También se puede represetar por medio de series, tal seria el caso para representar a un sistema digital.

Continuos - los estados del sistema varian en cualquier instante de tiempo, considerando que los valores del parámetro tiempo son numeros reales.

Discretos - en este caso el factor tiempo toma valores discontinuos o discretos, es decir, la variable tiempo tiene como dominio cualquier numero entero.

Invariantes en el Tiempo - los modelos de estos sistemas tienen parametros que son fijos, no dependen del tiempo. Es decir, si se aplica una misma entrada en dos instantes separados por un intervalo T de tiempo, se obtendrá dos salidas iguales, pero presentes en instantes defazados el mismo lapso T de tiempo. Si se explica por medio de la función de transferencia se puede decir que la función de transferencia de un sistema lineal invariante en el tiempo es la relación de la transformada de Laplace de la salida (función respuesta) a la transformada de Laplace de la entrada (función excitadora) bajo la suposición de que todas las condiciones iniciales son cero. Se puede decir que el 68000 es invariante en el tiempo, se obtendrá la misma salida corriendo un mismo programa en un instante u otro.

Variantes con el Tiempo - existen parametros que son funciones del tiempo, presentan caracteristicas dinámicas. Una misma entrada aplicada en dos tiempos diferentes generará dos salidas distintas. La función de transferencia está en función de variables que están relacionadas con el tiempo.

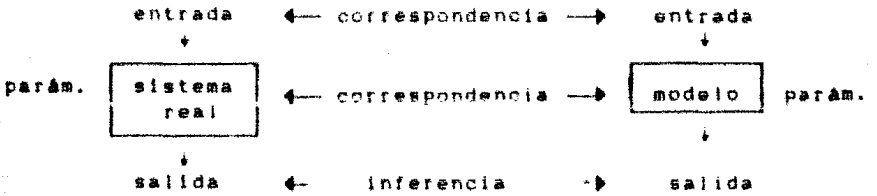
Concepto de Modelo

Para estudiar un sistema real a veces es posible experimentar con el mismo sin tener que utilizar un modelo. Sin embargo, no siempre es recomendable hacer esto porque puede ser impráctico, posiblemente las condiciones sean dificiles de controlar, tal vez sea casi imposible realizar algun experimento sobre el sistema real, seguramente será muy costoso, en ocasiones el sistema se puede dañar, las condiciones del medio ambiente pueden cambiar continuamente y sin poder ser controladas. Por lo tanto se considera un modelo para estudiar al sistema en cuestion.

Un modelo se define como la información, relacionada a un sistema, reunida para fines de estudio. Ampliando el concepto, un modelo es una representación de las principales caracteristicas y propiedades de un sistema dado. Por este motivo el modelo se emplea para describir, estudiar y predecir el comportamiento de un sistema.

Cabe señalar que el modelo debe describir al sistema en una forma suficientemente detallada, para que su comportamiento provea predicciones válidas del comportamiento del sistema.

A continuación se representa gráficamente un modelo como la simplificación de un sistema real.



Las funciones básicas de un modelo son las siguientes:

- predecir las posibles actividades de un sistema,
- comparar las salidas obtenidas, tomando en cuenta el conjunto de entradas que las originó,
- modificar las entradas hasta generar las salidas deseadas.

Dado que las funciones que realiza un modelo son muy amplias, cabe notar que un modelo se puede emplear en varios campos. Elmaghraby hizo un estudio acerca de las aplicaciones más generales que pueden tener los modelos y las clasificó en cinco tipos:

1) ayuda para el pensamiento

2) apoyo en la comunicación

3) entrenamiento e instrucción

4) herramienta para predecir

5) asistencia en la experimentación

Los modelos son una herramienta para el proceso del pensamiento porque se utilizan para organizar y clasificar conceptos abstractos o confusos. Al tratar de representar ideas o teorias, por medio de un modelo, las inconsistencias de éstas se notan y esto obliga a organizar, examinar y evaluar la validez de los pensamientos.

Un modelo se emplea en la comunicación, porque representa en forma más concisa (gráfica, fisica, etc.), algo que en varias palabras no seria fácil dar a entender. Un modelo evita confusiones originadas por una descripción ambigua y proporciona una comunicación más efectiva y eficiente. También permite una plena comprensión de la estructura general y de las relaciones de causa y efecto.

Un modelo es una excelente herramienta en el entrenamiento y la instrucción, porque se pueden manejar las condiciones para motivar a la persona a aprender sin ninguna presión, repitiendo el proceso hasta que lo haya entendido bien. Esto es especialmente importante en los entrenamientos de vuelo, porque se pueden representar condiciones muy dificiles, en una cabina de práctica, a las cuales el piloto se va a enfrentar en un vuelo real. Sin embargo, él tendrá la oportunidad de comportarse y maniobrar como si estuviera en condiciones reales, y en caso de fallar no habra daño alguno en él o en la nave.

En el SIM68K (Simulador del 68000) se utiliza un modelo para instruir al estudiante en el uso del sistema 68000. El usuario podrá escribir sus programas en el lenguaje ensamblador del 68000 y obtener resultados, pero desde una PC. En caso de requerir conocimientos básicos en el 68000, el alumno podrá aprenderlos en el TUT68K (Tutor del 68000). La ventaja del SIM68K y del TUT68K es que el usuario podrá aprender a su propio ritmo, a base de ensayo y error, no tendrá limitante de tiempo. Se ha demostrado que un aprendizaje de este tipo lleva a mejores resultados, porque se aprende por interés propio, y es un buen complemento práctico para una clase teórica.

Es importante un modelo para predecir las caracteristicas del comportamiento de una entidad modelada. Se emplea para poder prever posibles problemas que se tengan y evitar pérdidas materiales y personales. Por ejemplo, para implantar un sistema de emergencia en un barco, antes de que ocurra una falla en una travesia real.

SIMULACION DE SISTEMAS

Cuando es muy costoso, impráctico o casi imposible experimentar en un sistema real, se emplea un modelo del sistema. Así, se podrá controlar el embiente, las entradas y las salidas del experimento, con relativa facilidad y a menor costo, para obtener resultados más exactos y practicos.

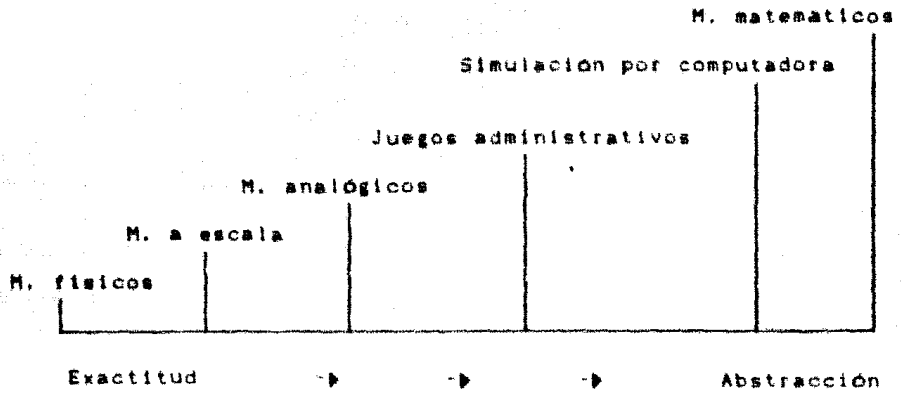
Por otra parte, un modelo debe ser sencillo, útil y representativo del sistema, o una parte de éste, que se desee modelar. En un mismo modelo no se requiere tomar en cuenta todos los detalles de un sistema, porque la información recibida podria ser confusa y tal vez no se cubrirían todos los objetivos solicitados.

Dado que un sistema puede ser complejo, o simplemente contar con varios elementos de interés, se puede aplicar el principio de modularidad y dividir el problema para entonces poder realizar varios modelos que cubran los diferentes aspectos de interés del sistema.

Una vez que se haya determinado que se requiere un modelo para la simulación de un sistema y tomando en cuenta todas las características con las que debe cumplir un modelo óptimo, es necesario definir el tipo de modelo que se requiere para que se puedan cubrir las necesidades de la simulación.

Existen diversos esquemas de clasificación de modelos, cada autor puede presentar una idea propia que varía con las demás. A continuación se muestra cómo Rowe emplea un esquema en forma de un espectro continuo.

La representación inicia en el lado izquierdo de la gráfica, con los modelos exactos o modelos reales a escala, y conforme se desplaza a la derecha, disminuye la exactitud en los modelos y aumenta el nivel de abstracción de éstos, finalizando así con los modelos matemáticos, los cuales son los más abstractos.



Los **modelos fisicos o icónicos** semejan al sistema en estudio fisicamente. Se emplean en la ingenieria como por ejemplo, los tuneles de viento o modelos de tamaño natural tales como una cabina para la simulación de vuelo.

Los **modelos a escala** son una copia del sistema real, pero varian en el tamaño del sistema que representan. Ejemplos pueden ser un globo terráqueo o una cadena de DNA. Este tipo de modelo se emplea para demostración o experimentación indirecta.

En los **modelos analógicos** se representa un atributo del sistema real con otro atributo del modelo. Un ejemplo es la relación de velocidad contra tiempo de un vehiculo, representada gráficamente por la longitud a lo largo de dos rectas.

A partir de los **juegos administrativos** aumenta la interacción hombre-computadora. En estos modelos, el hombre trabaja directamente con la salida de la computadora; él toma decisiones con base en la información recibida y estas decisiones retroalimentan al modelo.

SIMULACION DE SISTEMAS

La **simulación por computadora** permite planear, diseñar e implantar un modelo por medio de un programa de computadora. Dependiendo del grado de aproximación del modelo al sistema real simulado, se podrá obtener resultados apegados a la realidad y entonces implantar sin mayor dificultad, una solución óptima para el objetivo requerido.

En los **modelos matemáticos** se emplean símbolos para representar un atributo de un sistema. Por ejemplo un conjunto de ecuaciones pueden representar el volumen, presión o velocidad de un sistema.

Aunque un modelo puede ser muy complicado en sus fases matemática y física, su estructura fundamental debe ser muy sencilla. En términos generales se puede afirmar que un modelo consiste de alguna combinación de los siguientes elementos:

1) componentes

2) variables

3) parámetros

4) relaciones funcionales

5) restricciones

6) funciones de objetivo

Se entiende por **componentes**, elementos o subsistemas, las partes constitutivas, que en conjunto, conforman al sistema.

Las **variables** de entrada son aquellas que se producen fuera del sistema. Las variables creadas dentro del sistema se conocen con el nombre de:

- variables de estado, cuando indican una condición dentro del sistema,

- variables de salida, cuando salen del sistema.

Los **parámetros** son cantidades a las cuales el operador del sistema puede asignar valores arbitrarios, a diferencia de las variables, mismas que sólo se les puede asignar

valores que permitan la debida operación del sistema. Es importante señalar que los parametros, una vez establecidos, ya no se deben variar.

Las **relaciones funcionales** describen a las variables y a los parámetros de modo que muestran su comportamiento dentro del sistema. Estas relaciones pueden ser de naturaleza estocástica o determinística.

Las **restricciones** corresponden a limitaciones impuestas a los valores de las variables de entrada. Estas restricciones pueden ser impuestas por el diseñador, o por el sistema, según se requiera en el sistema mismo.

Por ultimo, la **función de objetivo** es una definición explicita de los objetivos o metas del sistema y de su forma de evaluación. Los objetivos pueden ser retentivos, tratan acerca de la conservación de los recursos o estados; o bien pueden ser adquisitivos, es decir, que conciernen con la adquisición de recursos o la obtención de estados.

Simulación de Sistemas

La simulación es la experimentación con un modelo de un sistema real, para investigar las propiedades del sistema, y consiste de las siguientes etapas:

1) Definicion del sistema - determinación de los limites o fronteras, restricciones y medidas de efectividad que se usarán para definir el sistema que se estudiará.

2) Formulación del modelo - reducción o abstracción del sistema real a un diagrama de flujo lógico.

3) Preparación de datos - identificación de los datos que el modelo requiere, y reducción de éstos a una forma adecuada.

4) Traslación del modelo - descripción del modelo en un lenguaje aceptable para la computadora que se empleará.

5) Validación - incremento a un nivel aceptable de confianza de modo que la inferencia obtenida del modelo respecto al sistema real sea correcta.

6) Planeacion estratégica - diseño de un experimento que producirá la información deseada.

7) Planeación táctica - determinar la manera en la cual se realizará cada una de las corridas de prueba especificadas en el diseño experimental.

8) Experimentación - corrida de la simulación para generar los datos deseados y efectuar el análisis de sensibilidad.

9) Interpretación - obtención de inferencias con base en los datos generados por la simulación.

10) Implantación - empleo del modelo y/o de los resultados.

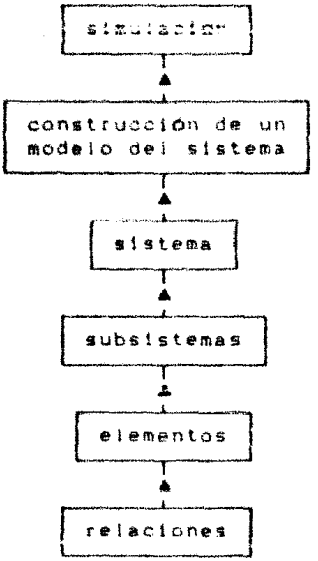
11) Documentación - registro de las actividades,del proyecto y los resultados, así como de la documentación del modelo y su empleo.

Tomando en cuenta cuáles son las etapas más generales de la simulación, se ocupa un diagrama de flujo para mostrar en una forma más dinámica los pasos para estudiar un sistema por medio de la simulación.

Este esquema se muestra en la página siguiente.

SIMULACION DE SISTEMAS

Cuando se desarrolla un modelo para implantarlo en un experimento de simulacion, se divide el sistema en estudio en un numero finito de subsistemas interconectados. Cada subsistema esta compuesto de elementos interconectados, cada uno de los cuales presenta un comportamiento propio. Las relaciones entre subsistemas debe definirse primero y despues el comportamiento dinámico del sistema. El comportamiento del sistema en general, depende del comportamiento de los subsistemas y del ambiente en el cual se desenvuelva el sistema. Estos conceptos se representan gráficamente a continuación:



En la simulacion existen problemas y limitaciones que se deben tomar en cuenta antes de hacer un proyecto de simulación. La principal dificultad de la simulación radica en su uso. Cualquier problema que se pueda resolver en forma sencilla con un método analítico no debe caer dentro de la simulacion, porque no es necesario y esto representaría una pérdida de recursos. La simulación se debe utilizar sólo cuando otros métodos no proveen una solución óptima.

Se puede presentar el caso en que la simulación no genere la solución optima al problema presentado, porque quizá necesite de la interrelación de varios factores que posiblemente no presenten condiciones optimas de trabajo, por ejemplo los recursos con los que se cuente o la experiencia del analista.

La exactitud de los resultados de la simulación es un tanto impredecible, aún para los modelos bien definidos. Esto se origina porque se manejan variables aleatorias y se experimenta con ejemplos limitados. Los resultados dependen de la calidad de los datos de entrada.

La validación de modelos complicados, especialmente de sistemas en la etapa de planificación, puede ser muy difícil. Esto se presenta cuando los datos de entrada son invalidos o la lógica del planteamiento es incorrecta.

La simulación puede ser muy cara en términos de tiempo de computadora. Se debe tratar de reducir el tiempo que se consuma pero sin aumentar la complejidad de la simulación.

La aparente simplicidad y el realismo de los modelos de simulación pueden confundir especialmente a las personas que no estén muy bien informadas respecto al tema o que no tengan la suficiente experiencia en este campo. Cuando se presenta un problema nuevo y difícil una persona puede inclinarse a emplear la simulación por la correspondencia inmediata que se puede establecer entre el problema y el modelo. Los modelos no se deben hacer exactamente iguales al sistema que representan, porque se busca simplificar el estudio y no aumentar la complejidad de este. El analista experimentado reproduce la estructura del sistema pero no su forma.

La simulación es una excelente herramienta pero hay que seguir los pasos planteados en este capitulo para darse cuenta si es necesario utilizarla o no.

ENSEÑANZA ASISTIDA POR COMPUTADORA

INTRODUCCION

La ciencia ha introducido la computación en varios campos, y el de la enseñanza no es la excepción. Se vió como una buena alternativa la aplicación de la computadora en la educación porque las computadoras son equipos de trabajo con los cuales se cuenta cada vez más en casi cualquier institución educativa, su manejo es relativamente sencillo, es importante estar al tanto de la tecnología, y además, actualmente se cuenta con varios paquetes de computación aplicados a la educación o al aprendizaje de algún tema en especial.

La importancia que se le ha reconocido a la educación por computadora es tal, que en la Facultad de Ingeniería se están realizando programas de computadora con este fin. Podemos citar el TUT68K, el cual es un programa tutor para aprender a manejar el sistema 68000 de Motorola, está diseñado para correrse en cualquier equipo PC compatible con IBM. También es el caso del SIM68K, programa simulador del 68000. Ambos programas son parte del trabajo de tesis que nosotros realizamos, al igual que un desensamblador.

A continuación trataremos temas generales acerca de la enseñanza asistida por computadora.

INTEGRACION DE INFORMACION, TECNOLOGIA Y EDUCACION

Hace casi veinte años se pensó en combinar información específica relacionada con algún tema en especial, tecnología que para el caso es la computadora, y métodos didácticos que llevarán a una buena enseñanza y por lo tanto a un mejor aprendizaje. A la conjugación de estos elementos se le llamó Enseñanza Asistida por Computadora (EAC).

A partir de ese tiempo, surgieron varios programas tutores. Se notó que los estudiantes que también se apoyaban en programas de EAC mejoraban su aprendizaje en un alto porcentaje, a comparación de aquellos que sólo asistían a clases guiadas por profesores.

Las ventajas que presenta la EAC son las siguientes:

El alumno decide qué temas va a estudiar, y qué tiempo le va a dedicar a cada uno, no importando que un tema en especial lo revise una y otra vez (el programa tutor siempre le responderá al alumno amablemente y no se va a cansar de sus preguntas), el estudiante decide su avance.

El profesor es un guía y no un proveedor de conocimiento. Esto facilita que el conocimiento no se le "inyecte" al alumno sino que sea el resultado natural de un proceso de aprendizaje. "Natural" significa que no sea estresante, al contrario, que estimule al alumno y que por interés propio aprenda.

La relación maestro-alumno va a ser directa y el alumno tendrá más confianza de preguntarle y de aclarar sus dudas. El maestro estará en mejor disposición de ayudarlo, dedicándole más tiempo y de la mejor manera, lo cual sería un poco difícil si el profesor estuviera impartiendo clase a todo un grupo.

Lo que el estudiante aprenda será a través de experiencia directa, por sí mismo podrá llegar a los puntos importantes del tema, y será capaz de formular sus conclusiones.

El alumno aprenderá por el método de ensayo y error, el error servirá como retroalimentación para que el estudiante mejore sus conocimientos del tema que se esté tratando.

El programa tutor puede ser un compendio de bibliografía y del conocimiento y experiencia de varios profesores. En este caso el tutor podrá servir como guía a otros profesores, cuya experiencia sea menor, para que puedan aplicar la estrategia de enseñanza del tutor a sus clases.

El profesor va a ser un buen observador, podrá tomar en consideración las partes del tema que para los alumnos sean más difíciles de entender, y distinguirá las técnicas de enseñanza del tutor, por medio de cuales el alumno aprende más eficientemente.

LAS COMPUTADORAS Y LA EDUCACION

Para poder lograr que un programa tutor sea lo que necesiten los alumnos, es necesario diseñarlo con cuidado y adoptar losmetodos pedagógicos más adecuados para el caso. Un tutor correctamente diseñado servirá en gran medida al alumno, pero si no fué realizado cuidadosamente, entonces puede causarle al estudiante dudas y conducirlo a un aprendizaje deficiente.

Se podrían aplicar las siguientes tres recomendaciones para que el tutor sea efectivo:

1) Utilizar cuestionarios, porque así el alumno aplicará lo que aprendió, y en todo caso podrá regresar a la parte del tema que no haya entedido bien.

2) La evaluación del desempeño del alumno durante el tutor, le servirá para tener una noción cuantitativa y cualitativa de su aprendizaje.

3) El material tratado en el tutor no deberá necesitar de la explicación continua de un profesor, porque estará planeado para que el alumno lo pueda utilizar por si solo, y conforme avance pueda captar todo o lo más relevante del tema.

La aplicación de la computadora en la educación tiene gran auge porque el método de aprendizaje es activo y no pasivo, como generalmente resulta en una clase con profesor y alumnos. En este último caso la información es mandada por el maestro o por los libros, y el alumno es un receptor pasivo o un espectador.

El aprendizaje debe ser activo si se desea que el estudiante capte ideas, conceptos, metodologías, etc. Esto se puede realizar cuando el cupo de alumnos es reducido, pero por diversas causas en los salones de clase se sobrepasa el numero máximo de estudiantes y casi se limita al profesor a utilizar métodos pasivos de enseñanza.

La computadora nos permite que el alumno no sea un espectador y que su aprendizaje sea activo, a un precio razonable. Con esto no se quiere decir que los profesores tienen que ser reemplazados por computadoras, pero en todo caso el papel principal de los maestros podría ser el de

reforzar los conocimientos del alumno o introducirlo en un tema nuevo, y relativamente complejo. Como el tutor puede examinar al alumno constantemente, el profesor sabrá el nivel de conocimiento que tenga el alumno del tema tratado. El profesor decidirá entonces qué hacer al respecto, si trabajar directamente con el alumno o aumentarle conocimientos básicos al tutor. También, el maestro se percatará del material que no sea de mucho interés a los estudiantes, y así poder modificar el tutor, dándole un enfoque diferente.

En un futuro, no muy lejano, la aplicación de la computación en la educación va a ser muy importante dado el alto nivel de interacción del tutor con el estudiante, la individualización de su enseñanza y aprendizaje y también la tendencia de reducir los costos de equipo de cómputo.

Sin embargo, la clave para una buena educación en el futuro radica en un sistema de producción de software efectivo, éste deberá ser diseñado, creado y revisado con detenimiento, para generar interés en el alumnado, pero sobre todo para que se mejore en gran medida sus conocimientos.

En el diseño del TUT68K nos dedicamos a trabajar con varios tutores para obtener ideas, nos dimos cuenta de las técnicas utilizadas, y si llevaban a un buen aprendizaje. Después con ideas propias y amplia bibliografía realizamos el TUT68K. En una primera etapa de realización, no cuenta con evaluaciones al alumno, pero si lo tratamos de hacer en una forma explícita, sencilla, pero con un buen contenido, para que el alumno lo entendiera fácilmente. En subsecuentes revisiones del tutor se le podrá complementar con una sección de evaluación al alumno.

FACTORES HUMANOS EN EL DISEÑO DE EAC

Para que los sistemas de EAC sean efectivos y tengan éxito, además de necesitar equipo de cómputo en buenas condiciones, contenido adecuado, y módulos de evaluación para el estudiante, es muy importante que cuente con interfases, correctamente diseñadas, entre computadora y estudiante.

Es imperativo que los diseñadores de sistemas de EAC, sistemáticamente evaluen todos los factores humanos que utilicen en los tutores, antes de entregar el programa a los

usuarios (alumnos).
Para mejorar la interfase computadora-estudiante se recomienda que el tutor sea probado por una persona, que tenga los conocimientos necesarios, que no sea ni diseñador ni estudiante, y que verifique que el tutor cumpla con eficiencia, claridad y simplicidad su objetivo. Dentro de un buen diseño de sistemas de EAC, se debe tomar en cuenta el comportamiento y la actitud del estudiante hacia el tutor, como la capacidad del sistema para ser interesante a los fines del alumno.

Un factor fundamental para que el sistema sea atractivo e interesante al alumno es la presentación de las pantallas del tutor. Se requiere que estén organizadas claramente, que su estructura sea apropiada a la tarea a realizar, que sean fáciles de entender y que sean consistentes con el nivel de conocimiento del estudiante.

Si solamente se le da al alumno un contenido valioso pero una presentación deficiente, el estudiante cometerá errores al leerlo y al interpretarlo, y no habrá interés por parte del estudiante de aprender por medio de programas tutores.

Dentro de un buen diseño de pantallas se debe tomar en cuenta los siguientes parámetros:

1) Simplicidad - al estudiante se le debe presentar con la cantidad y el nivel adecuado de material en la forma más sencilla posible. No se recomienda el uso de pantallas muy detalladas, pero si son necesarias, entonces deberán estar bien organizadas para evitar confusiones.

2) Diálogo Entre Computadora y Estudiante - el uso de menús permite una comunicación clara y evita errores por parte del alumno. Se recomienda que el alumno tenga acceso al menu principal en todo momento (esto se aplica en el TUTOR). El menu desplegado deberá mostrar sólo las opciones apropiadas del nivel actual. El número de opciones no deberá ser excesivo.

3) Localización de información importante - la información clave deberá estar en un lugar fijo para que el alumno se familiarice con esa localización y en caso de tener dudas pueda acudir al nivel de ayuda fácilmente.

4) Instrucciones - al dar instrucciones al alumno, es muy importante que sean claras y muy concisas, de otra forma el alumno se confundirá.

5) Espacio - cuando se presenta la información aglomerada, el tiempo de busqueda y la confusion aumentan bastante, y por otra parte el alumno se cansa rápidamente. Cuando se presenta la cantidad adecuada de información en la pantalla y con el espacio necesario, entonces el estudiante no se estresará al leerla y además la entenderá mejor.

6) Estandarización - se debe estandarizar terminología, abreviaturas y cualquier otra información que pueda causar confusión. Estas frases deberán ser consistentes en todo el programa.

7) Despliegado de Páginas - no toda la información se puede presentar en una sola página, por lo tanto el tutor deberá permitir que el alumno se mueva a través de las pantallas del tema con facilidad. En el TUT68K el usuario puede ver pantallas anteriores o posteriores a la actual, y la información está fragmentada de tal manera que se mantenga una secuencia lógica.

8) Uso del Color - se ha encontrado que el color da más información que las palabras, el estudiante responde rápidamente y comete menos errores. Estéticamente le atrae más al alumno y lo motiva a que trabaje con el tutor.

En el caso del TUT68K, decidimos utilizar pantallas monocromáticas en una primera versión, porque el uso de color reduce a la mitad la memoria disponible y como el tutor está sólo en un diskette quisimos ocupar la memoria eficientemente. Para cumplir con factores estéticos las pantallas son muy sencillas, la interacción del alumno no implica errores, se le da información importante, la lectura es sencilla y amena.

Siempre se le indica en qué parte del tutor se encuentra, el numero de página que está leyendo, y se indica si llegó a la ultima página.

Los factores humanos no son los unicos de importancia, pero pueden marcar una gran diferencia para que el tutor sea aceptado por el alumno, contribuyendo a una enseñanza amena, sencilla y completa; o no ser aceptado, aunque el contenido sea valioso.

METODOLOGIA PARA DESARROLLAR PROGRAMAS DE EAC

En el departamento de Ingenieria de la Universidad de Cambridge, Inglaterra, se diseñó una metodologia para desarrollar programas de enseñanza asistida por computadora. Esta metodologia cubre desde la especificación de los requerimientos hasta la revisión posterior a la implantación del tutor. A continuación se describe el proceso, las técnicas utilizadas para completar el tutor, y las responsabilidades de los participantes.

Objetivos de la Metodologia

Participación del usuario - el personal académico debe participar en las etapas iniciales de desarrollo del programa.

Funcionalidad - el paquete final deberá contener el programa y documentación de apoyo que ofrezca el grupo de requisitos predefinidos.

Confiabilidad - esta metodologia debe producir programas muy completos y confiables.

Utilidad - el sistema o programa resultante deberá ser fácil de usar.

Capaz de Modificarse - debe existir amplia documentación acerca del paquete y de su estructura, para que se modifique fácilmente en caso de que lo requiera el hardware empleado o las necesidades del usuario.

Apoyo automatizado - los miembros del equipo de desarrollo deberán ayudarse de aparatos automáticos que les faciliten sus tareas.

Reutilidad - la metodología debe ser general y partes del diseño producido para una aplicación dada deberán servir para otros diseños similares.

Actividades de la Metodología

Las actividades que constituyen la metodología del desarrollo de programas da comienzo cuando un miembro del personal académico solicita un programa tutor para la enseñanza de algún tema, y termina cuando se presenta el programa documentado y probado.
A continuación se presentan estas actividades.

Especificación de requerimientos de enseñanza - el programador analista debe estar al tanto de los requisitos solicitados para el programa, de preferencia que sea por escrito.

Consolidación y desarrollo de los requerimientos - la formación del equipo de trabajo depende del tipo de programa que se solicite pero el núcleo del grupo va a estar formado por un miembro del personal académico y un analista/programador. El equipo deberá reunir la información necesaria para entender los requerimientos del programa. Se concentran en "lo que se quiere" y no en "cómo se va a desarrollar". Los requisitos se expresan como entradas y salidas de un diagrama de flujo muy general. El desarrollo del programa comenzará por considerar qué procesos se necesitan para producir las salidas deseadas con las entradas dadas.

Producción y revisión de proposiciones - la estructura gráfica del programa expresa formalmente el contenido del programa. Esta se revisa junto con el personal académico que lo solicitó.

Producción y revisión del diálogo - después de verificar la información de la propuesta, se revisa el diálogo o procesos de entrada y salida. Cada elemento del diálogo se analiza y se divide en sus componentes básicos, es decir un módulo que realice una entrada de datos o una salida de información. Esto sirve para ver el modelo dinámico de los diálogos. Es muy importante para reafirmar los requerimientos del paquete.

Diseño de elementos de cómputo - el producto final de esta etapa será una jerarquía de módulos, el nivel inferior consiste de bloques de cómputo. Estos módulos realizan funciones discretas, generalmente ya hay software que lo haga o en caso contrario la rutina es muy sencilla (ocupa una página de programación).

Producción y revisión del documento de requerimientos de programación - lo que queda por hacer es el diseño detallado del diálogo y de los elementos de programación para los cuales no existe software.

Implantación y prueba de módulos - el siguiente paso es llevar a cabo el diseño. Pero antes de esto se revisa cada módulo y se crea un plan de implantación de módulos. Se sigue una metodología descendente, pero los módulos complejos o muy importantes deberán ser implementados en una etapa temprana. Se incorpora cada módulo uno a la vez, probándolo con la parte del programa que ya funciona correctamente.

Prueba del sistema - se le realizan pruebas exhaustivas al sistema para asegurarse que no tenga fallas. El proceso de pruebas deberá incluir pruebas llevadas a cabo por programadores analistas, miembros del personal académico y usuarios (estudiantes) voluntarios.

Revisión posterior a la implantación - después de haber realizado pruebas satisfactoriamente, al administrador de cómputo se le da el código compilado y una descripción del programa, para que se introduzca el sistema a la biblioteca de programas. Se le realizan procesos de retroalimentación para mantenimiento y observaciones valiosas. El personal académico y el personal de desarrollo revisan el sistema cierto tiempo después de haber sido entregado a los estudiantes. Esto se hace con objeto de revisar el uso que se le ha dado al paquete, y discutir algunas experiencias obtenidas del paquete.

Esta metodología fomenta el trabajo en equipo y el personal académico interviene en todo el desarrollo, permitiendo así que el trabajo final cumpla con los objetivos requeridos.

ENSEÑANZA ASISTIDA POR COMPUTADORA

Este metodo es fácil de utilizar y es muy valioso para controlar un proyecto. Separar los procesos de entrada y salida con los de computación permite que los paquetes se puedan modificar fácilmente. La estructura del sistema es modular, por lo que se puede utilizar en proyectos similares.

En la Universidad de Cambridge es costoso el empleo de un analista programador, pero aun así, se ha mejorado y aumentado la producción de paquetes para la enseñanza.

ENSEÑANZA INTELIGENTE ASISTIDA POR COMPUTADORA

La enseñanza inteligente asistida por computadora (EIAC) es más poderosa que la EAC, esencialmente porque hay más flexibilidad para aprender, el alumno no está limitado a seguir una secuencia de instrucciones predeterminadas por el diseñador del sistema, en cambio está diseñada para seguir diferentes rutas de aprendizaje, dependiendo de los requerimientos del alumno o sus intereses.

Un sistema de EIAC es un ambiente exploratorio controlado totalmente por el estudiante, y no hace falta la asesoría de un profesor. La flexibilidad en el aprendizaje con EIAC se debe a la separación entre el dominio del conocimiento del sistema y sus instrucciones (comunicación, tutoría y modelos del estudiante).

Arquitectura del sistema de la EIAC

La arquitectura de la EIAC está organizada en cuatro tipos de conocimiento:

1) Dominio del conocimiento - el tema del área que se va a enseñar.

2) Entendimiento del dominio por parte del estudiante - evoluciona dinámicamente conforme progresa el aprendizaje, el alumno utiliza varias herramientas para aprender, y la parte tutorial del sistema se da cuenta de los conocimientos que el alumno necesita aprender, para reforzarlos después.

3) Conocimiento para dar tutoría - cómo enseñar, aplica principios pedagógicos para mantener el interés del alumno y aumentar su aprendizaje.

4) Conocimiento del discurso - cómo comunicarse con el estudiante.

Formas de Enseñar

Una función clave de un tutor es la de diagnosticar correctamente al alumno (y ajustar constantemente este diagnóstico conforme avance la sesión). Para esto examinará al alumno y estudiará sus respuestas. También puede utilizar la táctica de enseñar al estudiante y verificar si se mantiene interesado en el tema.

Enseñar al alumno es darle información (respondiendo una pregunta o corrigiendo un error) o también, guiándolo a través del razonamiento para llegar a una solución correcta, (o representación correcta). Esta última forma de enseñanza es el método Socrático, y ha sido adoptado por la EIAC porque se rige por una enseñanza basada en el descubrimiento, lo cual forma la parte medular de la EIAC.

Si el alumno da una explicación utilizando factores que no son suficientes, entonces el tutor escoje un contraejemplo y pregunta por qué la dependencia casual no se aplica en ese caso. Este método enseña utilizando casos que se generan por medio de las preguntas que se determinan por las respuestas del alumno. Así se llevará gradualmente al alumno a que razone por sí mismo y que descubra los principios básicos generales del modelo en estudio.

Las ideas básicas de la EIAC son las siguientes: facilitar aproximaciones sucesivas hacia el conocimiento del objetivo, minimizar la carga de trabajo de memorización, y proveer una retroalimentación inmediata de los errores.

La flexibilidad de interacción y aprendizaje realizados por el poder de técnicas de inteligencia artificial aplicados a tareas educativas ha marcado la diferencia entre EIAC y EAC.

El control del alumno y la intervención tutorial son dos formas opuestas de interacción entre alumno-computadora. Haciendo un balance de éstas, le da la razón a EIAC.

Posiblemente una siguiente versión del TUT68K o del SIM68K utilice elementos de inteligencia artificial, ya que los beneficios serían notables, pero por el momento seguirán funcionando con los elementos de EAC.

DESARROLLO DEL SIMULADOR 68000 EN PC

Para el desarrollo del SIM68K seguimos los pasos marcados en el capitulo de Simulacion de Sistemas.

La formulación del problema era la simulación del sistema 68000. Habia que encontrar la forma de simular un microprocesador de 32 bits con otro de 16 bits, como lo es el 8088. Necesitábamos hacer un sistema económico pero que diera buenos resultados.

Como contábamos con un microprocesador 8088 para hacer la simulación, definitivamente no se iba a alcanzar la misma velocidad de ejecución, a que si se estuviera corriendo un mismo programa, pero en un microprocesador 68000. Por lo tanto decidimos que seria un simulador en tiempo muerto. Nuestro enfoque primordial era que simulara satisfactoriamente un programa y no tanto que se minimizara el tiempo que tomaria la simulacion.

Sin embargo para facilitar y agilizar la simulacion vimos que era necesario crear primero un desensamblador, porque entonces se obtendrian las caracteristicas primordiales de cualquier instrucción ensamblada. Con esto disminuiria el tiempo de la simulacion. Y estos datos se podrian usar como paso de parametros al realizar la simulacion.

Todo el sistema esta basado entonces en el siguiente modelo general:

1) entrada del programa ensamblado

2) desensamblado del programa

3) adquisición de datos importantes, por instrucción para la simulación

4) simulación de las instrucciones.

na parte muy importante de cualquier sistema es el paso de parametros, porque es la forma en que se intercomunican dos o más sistemas. Para el caso del SIM68K se decidió que estos serian los siguientes:

1) mnemónico

2) operando 1

3) operando 2

4) longitud

5) modo de direccionamiento

6) registro

7) tamaño (byte, word, long word)

Todo el sistema se implantó en turbo pascal 5.0, se hizo uso de unita para agilizar el procesamiento de las instrucciones.

Las instrucciones se programaron en pequeños sistemas separados de los sistemas principales, y son llamados cuando el programa ensamblado así lo especifique.

La forma en que funciona todo el sistema es el siguiente:

1) Por medio de pantallas se presenta el sistema 68000.

2) Al usuario se le muestra cualquier directorio que desee accesar

3) Después de escoger el directorio que necesite, el usuario introduce el nombre del archivo ensamblado

4) Se verifica que se pueda accesar, en caso contrario se le marca error para que lo verifique y se pueda proceder

5) Se desensambla el archivo por grupos de 100 instrucciones a la vez, por lo tanto, se hacen pocos accesos al programa ensamblado y aumenta así la velocidad de ejecución

6) Al desensamblar el archivo, se van obteniendo valores importantes para realizar posteriormente la simulación, estos fungen como paso de parámetros: mnemónico, los dos operandos, la longitud de la instrucción, el modo de direccionamiento, el registro, el tamaño (byte, word, long word).

7) El simulador va leyendo cada instrucción desensamblada

8) Busca en una sección de memoria predefinida los operandos, basándose en el modo de direccionamiento en el cual se haya basado la instrucción

9) Analiza de qué instrucción se trata

10) Manda a la subrutina de la instrucción los datos necesarios para que se ejecute la instrucción

11) El resultado queda en los registros o localidades de memoria correspondientes, el PC se maneja desde afuera de las subrutinas de las instrucciones

12) Al terminar la ejecución de una instrucción se sigue a la siguiente, hasta llegar al final del programa.

Las limitaciones del diseño son: el numero de instrucciones que se pueden desensamblar a la vez. Como es simulación en tiempo muerto, no se llega a la velocidad que tardaría el sistema real. Toma en cuenta las instrucciones del 68000, pero no se hace una extensión de las demás características que tienen las versiones más avanzadas del 68000.

Las ventajas son: es un sistema confiable, económicamente bajo, ocupa sólo dos discos (5.25) para manejarlo, es muy amigable, la posibilidad de que el usuario introduzca datos erróneos por equivocación, es muy baja.

CONCLUSIONES

El sistema del simulador del 68000 fué un comienzo para simular un microprocesador. Puede servir como base para implantar los cambios o modificaciones pertinentes para que puedan correr las versiones más avanzadas del 68000.

Se puede utilizar como material didáctico.

Apendice 1
Instrucciones del 68000

En este apendice se describe a detalle cada una de las 56 instrucciones con que cuenta el 68000, incluyendo las variantes, la notacion de ensamblador y el tamaño del operando en bits. Tambien se mencionan las banderas de condicion del registro de estado que afecta cada operacion. Se encontraran ejemplos ilustrativos para cada instruccion.

Con el fin de poder definir las instrucciones y su notacion en ensamblador se presentan los siguientes simbolos:

An:	registro de direcciones (n señala el numero del registro).
Dn:	registro de datos (n señala el numero del registro).
Rn:	Cualquier registro de datos o direcciones (n señala el numero del registro).
SR:	registro de estado.
CCR:	codigo de condicion.
SP:	apuntador activo del stack.
USP:	apuntador del stack del usuario.
SSP:	apuntador de stack del sistema.
d:	valor de un desplazamiento (d8 indica un desplazamiento de 8 bits; d16 un desplazamiento de 16 bits).
(An):	contenido de la dirección cargada en el registro An.
N:	tamaño del operando, en bytes.
EA:	direccion efectiva del operando real.

Instrucciones de transferencia o movimiento de datos.

EXG: Exchange registers.

Función: intercambio de registros.
Variantes: ninguna.
Notación en ensamblador: EXG R., R.
Tamaño del operando: 32 bits
Observaciones: Esta operación no afecta las banderas.

Ejemplo: EXG D1,A0
El contenido de D1 se transfiere a A0 y el contenido de A0 se transfiere a D0.

LEA: Load Effective Address

Función: cargar dirección efectiva.
Variantes: ninguna.
Notación en ensamblador: LEA <ea> , An
Tamaño del operando: 32 bits.
Observaciones: Esta operación no afecta a las banderas

Ejemplo: LEA A1 ,A5
La dirección contenida en A1 se copia al registro A5.

LINK: Link and Allocate

Función: reservar en el stack una zona del número de bytes indicado, en el desplazamiento dejando el lugar del principio de dicha zona en un registro de direcciones.

Variantes: ninguna.

Notación en ensamblador: LINK An, #desplazamiento

Tamaño del operando: 32 bits

Observaciones: La ejecución de esta instrucción provoca las siguientes operaciones:

(SP) — An

An — SP

SP+d — SP

además de la extensión de signo a los 16 bits restantes.

Ejemplo: LINK A0,#50

Reserva lugar para 50 Bytes, la dirección del primero de los 50 bytes queda almacenado en el registro A0.

MOVE: (Move Data from Source to Destination)

Función: mover dato de registro fuente a registro destino.

Variantes: MOVEA: mover fuente a registro de dirección.

MOVEQ: mover datos cortos a destino.

MOVE: mover el apuntador de stack del usuario (USP).

Notación en ensamblador: MOVE ea , ea

Tamaño del operando: 8, 16 ó 32.

Observaciones: cuando el destino es un registro de direcciones las banderas no son afectadas, y ocurre extensión de signo a los bits restantes.

Ejemplos:

MOVE $1000, $2000

Transfiere 16 bits del registro $1000 al registro $2000.

MOVEI #$32,D5

El dato inmediato $32 se transfiere a la parte baja del

registro D5.

MOVE (A5)+,D5

El dato que apunta el registro A5 es transferido al registro D5 posteriormente se incrementa el contenido del registro A5.

MOVEA (5,A1

El dato que contiene D5 se copia al registro A1.

MOVEQ #$FA,D1

El numero $FA se carga en D1, extendiendo el signo a los 32 bits

MOVEM: (Move Multiple Registers)

Function: transferencia multiple de registros.
Variantes: ninguna.
Notacion en ensamblador: MOVEM LISTA DE REGISTROS , ea
MOVEM ea , LISTA DE REGISTROS
Tamaño del operando: 16 , 32 bits.
Observaciones: en el caso de operandos de 16 bits, ocurre extension de signo

Ejemplo: MOVEM $1000, A1-A4

Los registros A1 al A4 son almacenados a partir de la dirección $1000.

MOVEP: (Move Peripheral Data)

Function: esta instruccion se diseño para mover informacion entre los perifericos de 8 bits y la memoria. Controla la transferencia de 2 o 4 bytes para los perifericos, que estan conectados a las 8 lineas de mas peso del bus de

datos o a los de menos peso. En el primer caso, los bytes por se trasladan corresponden a direcciones pares, en el segundo a las direcciones nones.

Variantes: ninguna.

Notacion en ensamblador: MOVEP Dn, d(An)
MOVEP d(An), Dn

Tamaño del operando: en 32 bits.

Observaciones: esta operacion no afecta a las banderas

Ejemplo: MOVEP D2, $100(A1)

Si por ejemplo A1 contiene una direccion $1000, los 4 bytes del registro D2 se transfieren por las lineas de bus, quedando el byte mas significativo en la direccion $1002, y el menos significativo en la direccion $1008.

PEA: (Push Effective Address)

Funcion: empujar al stack la direccion efectiva.

Variantes: ninguna.

Notacion en ensamblador: PEA ea

Tamaño del operando: 32 bits.

Observaciones: esta operacion no afecta a las banderas.

Ejemplo: PEA (A0)

El contenido de la direccion especificada por A0 se carga en el stack

SWAP: (Swap Register Halves)

Funcion: intercambiar entre si las dos mitades de 16 bits de un registro de datos.

Variantes: ninguna.

Notacion en ensamblador: SWAP Dn

Tamaño del operando: 16 bits

Observaciones: No es necesario el intercambio, el bit de más peso es 1, pero si el resultado es cero, Z no es afectado. V y C son siempre cero.

Ejemplo: SWAP D2

Si D2 valía FF10, después de efectuar la operación SWAP, D2 vale 10FF.

UNLK: Unlink

Función: reinicializar el Stack constituido por el registro An, desenlazar stack.

Variables: ninguna.

Notación en ensamblador: UNLK An

Tamaño del Operando:

Observaciones: An se carga con la palabra sacada de SP y SP se carga con el contenido de An

An ← SP

(SP)+ ← An

Las banderas de condición no son afectadas

Ejemplo: UNLK A3

A3 recibe el valor de SP incrementado en 4, SP recibe el valor de A3.

ADD: (Add Binary)

Funcion: suma binaria entre el operando origen y el operando destino.

Variantes: ADDA: sumar fuente a registro de direccion.
ADDQ: sumar dato corto a destino.
ADDI: sumar dato inmediato a destino.
ADDX: sumar con bit de tendion a destino.

Notacion en ensamblador: ADD Dn, ea
ADD ea , Dn

Tamaño del operando: .b, .w, .l bits.

Observaciones: N=1 si el resultado es negativo, Z=1 si el resultado es cero, C=1 y X=1 si el resultado no cabe en el operando destino, V=1 si al sumar dos numeros de mismo signo el resultado sobrepasa el margen de complemento a 2, y en este caso el signo del resultado se cambia.

Ejemplos: ADD.L D2,D7

Los 32 bits de D2 son sumados a los 32 bits de D7, quedando el resultado de esta suma en el registro D7.

ADD.W D2,D7

Los 16 bits menos significativos de D2 son sumados a los 16 bits menos significativos de D7, quedando el resultado de esta suma en los 16 bits menos significativos del registro D2.

CLR: (Clear an Operand)

Funcion: borrar un operando (poner a cero).

Variantes: ninguna.

Notación en ensamblador: CLR ea

Tamaño del operando: 8, 16 y 32 bits.

Observaciones: las banderas N, V y C quedan a cero, Z no es afectado y Z pasa siempre a 1.

Ejemplo: CLR.B D5

Pone a cero los 8 bits menos significativos de D5.

CMF: (Comparar)

Función: compara fuente con destino.

Variantes: CMFR: comparar fuente con registro de dirección.

CMFM: comparar memoria.

CMFI: comparar dato inmediato con destino.

Notación en ensamblador: C F ea , Dn

Tamaño del operando: 8, 16 y 32 bits.

Observaciones: afecta a todas las banderas menos a X

Ejemplo: CMF.B D7,$0

Resta el contenido del registro D7 al contenido de la dirección $0, afectando a las banderas. El resultado de la resta se pierde.

DIVS: (Signed Division)

Función: dividir destino entre fuente con signo.

Variantes: ninguna.

Notación en ensamblador: DIVS ea , Dn

Tamaño del operando: 16 bits.

Observaciones: divide el operando destino, de 32 bits y situado en un registro de datos, entre el operando fuente, de 16 bits. El resultado se coloca en el registro

destino, el cuando el cociente en los 16 bits de menos significativos; el resto queda en los 16 bits más significativos.

En cuanto a las banderas C siempre 0 , no es afectado. Z=1 si el cociente es cero. N=1 si el cociente es negativo. V=1 cuando el cociente tiene más de 16 bits, al ser el dividendo mucho mayor que el divisor. El 68000 termina la operación sin modificar el dividendo ni el divisor.

Ejemplo: DIVS #100,D5

Los 32 bits de D5 - 2^0 (entendido a 16 bits), el resultado queda en D5

DIVU: (Unsigned Division)

Función: dividir destino entre fuente sin signo.

Variantes: ninguna.

Notación en ensamblador: DIVU <ea>, Dn

Tamaño del operando: 16 bits.

Observaciones: divide el operando destino, de 32 bits y situado en un registro de datos, entre el operando fuente, de 16 bits. El resultado se coloca en el registro destino, situando el cociente en los 16 bits de menos significativos; el resto queda en los 16 bits más significativos.

En cuanto a las banderas C siempre 0 , no es afectado. Z=1 si el cociente es cero. N=1 si el bit de más peso del cociente es 1 (no hay signo). V=1 cuando el cociente tiene más de 16 bits, al ser el dividendo mucho mayor que el divisor. El 68000 termina la operación sin modificar el dividendo ni el divisor.

Ejemplo: DIVS D2,D0

los 32 bits de D4 : los 16 bits menos significativos de D0. El resultado ocupa los 32 bits de D4.

EXT: (Sign Extend)

Función: extensión del signo.
Variantes: EXTB (extensión de signo para un Byte.
Notación en ensamblador: EXT Dn
Tamaño del operando: 16 y 32 bits.
Observaciones: las banderas V y C quedan a 0, mientras que X no se afecta. N=1 el resultado es negativo. Z=1 si el resultado es 0.

Ejemplo: EXT D5

MULS: (Signed Multiply)

Función: multiplicación con signo (fuente * destino) -> destino.
Variantes: ninguna.
Notación en ensamblador: MULS <ea>, Dn
Tamaño del operando: 16 bits.
Observaciones: multiplica dos operandos de 16 bits, con signo almacenando el resultado de 32 bits en el destino. Solo admite el tamaño de la palabra y el destino es siempre un registro de datos.
V y C están siempre a 0. X no se afecta. Z pasa a 1 si el resultado es 0.

Ejemplo: MULS #500,D0
El numero 500 extendido a 16 bits * los 16 bits menos significativos de D0. El resultado ocupa los 32 bits de D0.

MULU: (Unsigned Multiply)

Función: multiplicación sin signo.
Variantes: ninguna.
Notación en ensamblador: MULU ea , Dn
Tamaño del operando: 16 bits. INSTRUCCIONES DEL 68000
Observaciones: multiplica dos operandos de 16 bits, sin signo almacenando el resultado de 32 bits en el destino. Solo admite el tamaño de la palabra y el destino es siempre un registro de datos.
V y C están siempre a 0. X no se afecta. Z pasa a 1 si el resultado es 0.

Ejemplo: MULU #500,D1
El numero 500 extendido a 16 bits * los 16 bits menos significativos de D0. El resultado ocupa los 32 bits de D0.

NEG: (Negate)

Función: complementar, negar destino.
Variantes: NEGX: negar con extensión.
Notación en ensamblador: NEG ea
Tamaño del operando: 8, 16 y 32 bits.
Observaciones: resta el operando destino de cero, para obtener su complemento a 2. El resultado se almacena en el destino.
Afecta todas las banderas, sin embargo tanto C como V se ponen a cero si el resultado es cero; en los demás casos pasan a 1.

Ejemplo: NEG $FAFE0

Resta el contenido de $FAFB0 de cero y guarda el resultado en $FAFB0.

SUB: (Substract Binary)

Función: restar destino de fuente.

Variantes: SUBa: restar fuente al registro de dirección.

SUBi: restar dato inmediato al destino.

SUBq: restar dato corto al destino.

SUBx: restar con bit e tendido al destino.

Notación en ensamblador: SUB ea , Dn

SUB Dn, ea

Tamaño del operando: B, W y L bits.

Observaciones: afecta a todas las banderas. N=1 si el resultado es negativo. Z=1 si el resultado es cero. C=X=1 si se produce un carry. V=1 si el resultado de la resta de dos números con signo distinto excede el límite del complemento a 1.

Ejemplo: SUB.B $AB5C, D6

Los 8 bits menos significativos del registro D6 - contenido del registro $AB5C. El resultado ocupa los 8 bits menos significativos de D6.

AND: (And logical)

Funcion: AND logico entre fuente y destino.
Variantes: ANDI: AND entre dato inmediato y destino.
Notacion en ensamblador: AND ea , Dy
AND Dr, ea
Tamaño del operando: 8, 16, 32 bits.
Observaciones: uno de los operandos es siempre un registro de datos. C y V quedan siempre en 0. X no es afectado. Z pasa a 1 si el resultado es cero y N=1 si el bit de más peso del resultado es 1.

Ejemplo: AND #FF, D2
Se opera con los 16 bits del registro #FF y los 16 bits del registro D2, afectandose los 16 bits del registro D2.

EOR: (Exclusive OR logical)

Funcion: OR exclusiva entre fuente y destino.
Variantes: EORI: OR exclusiva entre dato inmediato y destino.
Notacion en ensamblador: EOR Dr, ea
Tamaño del operando: 8, 16 y 32 bits.
Observaciones: uno de los operandos es siempre un registro de datos. C y V quedan siempre en 0. X no es afectado. Z pasa a 1 si el resultado es cero y N=1 si el bit de mas peso del resultado es 1.

Ejemplo: EOR D2, $AB
Se opera con los 16 bits del registro D2 y los 16 bits del registro $AB, afectandose los 16 bits del registro $AB.

NOT: (Logical Complement)

Función: complemento a uno.
Variantes: ninguna.
Notación en ensamblador: NOT ea
Tamaño del operando: 8, 16 y 32 bits.
Observaciones: C y V quedan siempre en 0. X no es afectado. Z pasa a 1 si el resultado es cero y N=1 si el bit de mas peso del resultado es 1.

Ejemplo: NOT D0
Modifica los 16 bits menos significativos de D0, cambiando 1 por ceros y viceversa.

OR: (Inclusive OR Logical)

Función: función lógica OR entre fuente y destino.
Variantes: ORI: OR entre dato inmediato y destino.
Notación en ensamblaje: OR ea, Dn
OR Dn, ea
Tamaño del operando: 8, 16 y 32 bits.
Observaciones: uno de los operandos es siempre un registro de datos. C y V quedan siempre en 0, X no es afectado. Z pasa a 1 si el resultado es cero y N=1 si el bit de mas peso del resultado es 1.

Ejemplo: OR.L D1, D5
Se opera con los 32 bits del registro D1 y los 32 bits del registro D5, afectandose los 32 bits del registro D5.

Scc: (Set According to Condition)

Funcion: poner a un de acuerdo a condicion.
Variantes: ninguna.
Notacion en ensamblador: Scc ea
Tamano del operando: 8 bits.
Observaciones: no afecta a las banderas

Ejemplo: Scc (A2)
El byte definido por la direccion A2 se pone a 1 si C=1. En el caso contrario, se pone a 0.

TST: (Test an Operand)
Funcion: comparar con 0 el operando.
Variantes: ninguna.
Notacion en ensamblador: TST ea
Tamano del operando: 8, 16 o 32 bits.
Observaciones: no modifica al operando, el resultado solo actua sobre las banderas de condicion. X=C=0. X no se afecta. N=1 si el operando es negativo y Z=1 si el operando es cero.

Ejemplo: TST.B D2
Revisa si el contenido de D2 = cero y afecta las banderas de acuerdo al resultado.

TAS (Test and Set Operand)

Funcion: prueba y puesta a uno del operando.
Variantes: ninguna.
Notacion en ensamblador: TAS ea

Tamaño del operando: 8 bits.

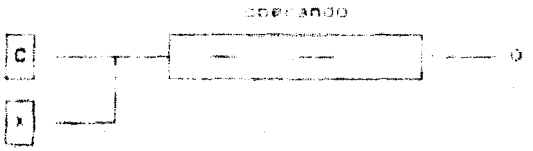
Observaciones: pone a 1 el bit 7 del operando. V=0. A no se afecta. N=1 si el operando es negativo y Z=1 si el operando es cero.

Ejemplo: TST $FF

Resultado: si el contenido de (FF) = 0 pone a 1 el byte 7 de (FF) y afecta las banderas de acuerdo al resultado.

ASL: (Arithmetic Shift Left)

Funcion: desplazamiento aritmetico a la izquierda.
Variantes: ninguna.
Notacion en ensamblador: ASL Dx, Dy
ASL #[1-8], Dy
ASL ea
Tamaño del operando: 8, 16 y 32 bits.
Observaciones: el bit de signo no se mantiene. N=1 si el bit de mas peso del resultado es 1. El numero de desplazamientos puede ser de entre 1 y 8. Z=0 si el resultado es cero. =1 cuando el bit de mas peso cambia de estado durante el desplazamiento. C=1 si el bit que sale del operando y se carga en esa bandera vale 1.

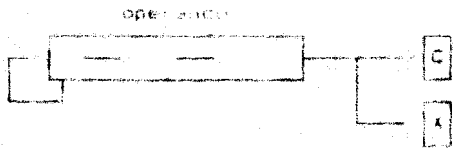


Ejemplo: ASL.L D1,D0
Los 32 bits de D0 se desplazan a la izquierda el numero de veces que indica D1

ASR: (Arithmetic Shift Right)

Funcion: desplazamiento aritmetico a la derecha.
Variantes: ninguna.
Notacion en ensamblador: ASR Dx, Dy
ASR #[Dato], Dy
ASR ea
Tamaño del operando: 8, 16 y 32 bits.

Observaciones: el bit de signo no se mantiene. N=1 si el bit de mas peso del resultado es 1. El numero de desplazamientos puede ser de entre 1 y 8. Z=1 si el resultado es cero. V=1 cuando el bit de mas peso cambia de estado durante el desplazamiento. C=1 si el bit que sale del operando y se carga en esa ultima vez 1.



Ejemplo: ASR.B D2,D6

Los 8 bits de D6 se desplazan a la izquierda el numero de veces que indica D2

LSL (Logical Shift Left)

Funcion: desplazamiento logico a la izquierda.

Variantes: ninguna.

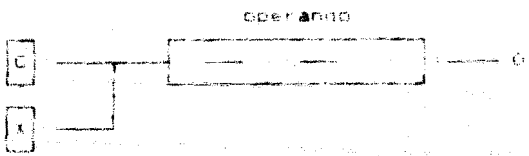
Notacion en ensamblador: LSL Dx, Dy

LSL #(DATO), Dy

LSL <ea>

Tamaño del operando: 8, 16 y 32 bits.

Observaciones: el desplazamiento lo hace sin signo. El numero de posiciones que se desplaza es de 1 a 8. V siempre es 0. N=1 si el resultado es negativo y C=1 de acuerdo con el bit que le llega del operando



Ejemplo: LSL D2,D6

Los lo bits de D6 se desplazan a la izquierda el numero de veces que indica D2.

LSR: (Logical Shift Right)

Funcion: desplazamiento logico a la izquierda.

Variantes: ninguna.

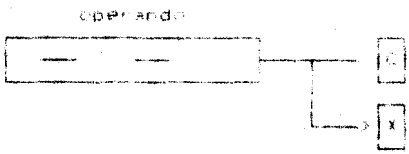
Notacion en ensamblador: LSR Dx, Dy

LSR #(DATO), Dy

LSR <ea>

Tamaño del operando: b, lo y 32 bits.

Observaciones: el desplazamiento lo hace sin signo. El numero de posiciones que se desplaza es de 1 a 8. C siempre es 0. N=1 si el resultado es negativo y Z=1 de acuerdo con el bit que se lleva del operando



Ejemplo: LSR.w D2,D6

Los lo bits de D6 se desplazan a la izquierda el numero de veces que indica D2

ROL: (Rotate Left without Extend)

Funcion: rotacion a la izquierda sin bit de extension.

Variantes: ninguna.

Notacion en ensamblador: ROL Dx, Dy

ROL #(DATO), Dy

ROL ea

Tamaño del operando: 8, 16 y 32 bits.

Observaciones: El bit que sale por el extremo izquierdo entra por el derecho, además de dirigirse al Carry. El número de rotaciones es de 1 a 8. Cuando el operando es una posición de memoria sólo se rota una posición y la longitud del operando es de 16 bits. N=1 si el bit más significativo del resultado de la rotación es 1. Z=1 si el resultado es cero. V, es siempre cero. C toma el valor del bit del extremo izquierdo. X no es afectado.

Ejemplo: ROL #3, D2

Los 16 bits de D6 se desplazan a la izquierda 3 veces.

ROR: (Rotate Right without Extend)

Función: rotación a la derecha sin bit de extensión.

Variantes: ninguna.

Notación en ensamblador: ROR Dx, Dy

ROR # DATO , Dy

ROR ea

Tamaño del operando: 8, 16 y 32 bits.

Observaciones: El bit que sale por el extremo derecho entra por el izquierdo, además de dirigirse al Carry. El número de rotaciones es de 1 a 8. Cuando el operando es una posición de memoria sólo se rota una posición y la longitud del operando es de 16 bits. N=1 si el bit más significativo del resultado de la rotación es 1. Z=1 si el resultado es cero. V, es siempre cero. C toma el valor del bit del extremo derecho. X no es afectado.

Ejemplo: ROR A5

Los 16 bits de A7 se desplazan a la derecha 1 vez.

ROXL: Rotate Left with Extend

Funcion: rotacion a la izquierda con bit de extension.
Variantes: ninguna.
Notacion en ensamblador: ROXL Dx , Dy
ROXL # DATO>, D.
ROXL ea>
Tamano del operando: B, lo ; W bits.
Observaciones: se trata de una rotacion en la que el bit que sale por la izquierda del operando se carga en las banderas C y X. El valor previo de X se introduce en el bit de la derecha del operando. El numero de rotaciones es de 1 a 8. Cuando el operando es una posicion de memoria solo se rota una posicion , la longitud del operando es de 16 bits. N=1 si el bit mas significativo del resultado de la rotacion es 1, Z=1 si el resultado es cero. V, es siempre cero. C toma el valor del bit del extremo izquierdo.

Ejemplo: ROXL.B D2,D7
El contenido de D7 se desplaza a la izquierda el numero de veces indicado por D2.

ROXR: Rotate Right with Extend

Funcion: rotacion a la derecha con bit de extension.
Variantes: ninguna.
Notacion en ensamblador: ROXR Dx, Dy
ROXR # DATO , Dy
ROXR ea

Tamaño del operando: 1, 16 ó 32 bits.

Observaciones: Se trata de una rotación en la que el bit que sale por la derecha del operando se carga en las banderas C y X. El valor previo de X se introduce en el bit de la izquierda del operando. El número de rotaciones es de 1 a 8. Cuando el operando es una posición de memoria sólo se rota una posición y la longitud del operando es de 16 bits. Tras el 8. bit más significativo del resultado de la rotación es 1, Z=1 si el resultado es cero. V, es siempre cero. C toma el valor del bit del extremo derecho.

Ejemplo: ROR.W #5,$FFA0

El contenido del registro $FFA0 se desplaza a la derecha 5 posiciones.

Instrucciones para Manipulacion de Bits Individuales.

BCHG: (Test a Bit and Change)

Funcion: probar un bit y cambiar.
Variantes: ninguna.
Notacion en ensamblador: BCHG Dn, ea
BCHG #(DATO), Dn
Tamaño del operando: 8 y 32 bits.
Observaciones: el objetivo de esta instruccion es probar un bit del operando reflejar su estado en la bandera Z cambiar el estado lógico del bit probado. Cuando el operando es un registro de datos se puede probar cualquiera de sus 32 bits. Si el operando es una posicion de memoria, solo se puede probar un byte (bits 0 al 7). La unica bandera que se afecta es la bandera Z.

Ejemplo: BCHG #4,D2
Prueba el bit 4 del registro D2 y, luego cambia su estado lógico.

BCLR: (Test a Bit and Clear)

Funcion: probar un bit y borrar (ponerlo a cero).
Variantes: ninguna.
Notacion en ensamblador: BCLR Dn, Dy
BCLR #(DATO), ea
Tamaño del operando: 8 y 32 bits.
Observaciones: el objetivo de esta instruccion es probar un bit del operando, reflejar su estado en la bandera Z poner a cero el bit probado. Cuando el operando es un registro de datos se puede probar cualquiera de sus 32 bits. Si el operando es una posicion de

memoria, solo se puede probar un byte (bits 0 al 7). La unica bandera que se afecta es la bandera Z.

Ejemplo: BCLR D7,14F00

Probar el bit indicado por D7 del registro $14F00.

BSET: (Test a Bit and Set)

Funcion: probar un bit y poner a uno.

Variantes: ninguna.

Notación en ensamblador: bSET Dn, <ea>

bSET #(DATO), <ea>

Tamaño del operando: 8 y 32 bits.

Observaciones: el objetivo de esta instrucción es probar un bit del operando reflejar su estado en la bandera Z y poner a 1 el bit probado. Cuando el operando es un registro de datos se puede probar cualquiera de sus 32 bits. Si el operando es una posición de memoria, solo se puede probar un byte (bits 0 al 7). La unica bandera que se afecta es la bandera Z.

Ejemplo: bSET #7,$4F00

Probar el bit indicado por #7 del registro 4F00 y ponerlo a 1.

BTST: (Test a Bit)

Funcion: prueba de un bit.

Variantes: ninguna.

Notación en ensamblador: bTST Dn, <ea>

BTST #(DATO), <ea>

Tamaño del operando: 8 y 32 bits.

Observaciones: El objetivo de esta instrucción es probar un bit del operando y reflejar su estado en la bandera Z. Cuando el operando es un registro de datos se puede probar cualquiera de sus 32 bits. Si el operando es una ubicación de memoria, sólo se puede probar un byte (bits 0 al 7). La única bandera que se afecta es la bandera Z.

Ejemplo: BCLR D1,$0F00

Probar el bit indicado por D1 del registro $AF00.

Instrucciones de Manipulación en Decimal Codificado en Binario (BCD).

ABCD: (Add Decimal with Extend)

Función: sumar fuente con el destino con el bit X en BCD.
Variantes: ninguna.
Notación en ensamblador: ABCD Dy , Dx
ABCD -(Ay) , -(Ax)
Tamaño del operando: 8 bits.
Observaciones: al no operar con signo N y V quedan indefinidos, mientras que C quedan siempre en 0. X no es afectado. Z pasa a 1 si el resultado es cero y N=1 si el bit de mas peso del resultado es 1.

Ejemplo: ABCD D1,D2
El Byte de menos peso de D1 se suma en BCD, con el correspondiente de D2 y a X. El resultado se almacena en D2

NBCD: (Negate Decimal with Extend)

Función: negar destino codificado BCD.
Variantes: ninguna.
Notación en ensamblador: NBCD ea
Tamaño del operando: 1a byte.
Observaciones: en aritmética BCD, el operando destino y el bit X se restan de cero, depositando el resultado en el destino. Con esta instrucción se calcula el complemento a 10 del destino (cuando X=0), así como el complemento a 9 (si X=1). Las banderas N, V y X quedan indefinidos. C=1 cuando se genera acarreo decimal. Z=0 si el resultado es distinto de cero; en caso contrario no cambia de estado.

Ejemplo: NBCD D2

Se realiza la resta 0-(D2)-X y se almacena en (D2).

SBCD: (Subtract Decimal with Extend)

Función: restar fuente a destino en BCD con bit de extensión.

Variantes: ninguna.

Notación en ensamblador: SBCD Dy, Dx

SBCD -(Ay), -(Ax)

Tamaño del operando: 8 bits.

Observaciones: al no operar con signo N y V quedan indefinidos, mientras que C queda siempre en 0. X no es afectado. Z pasa a 1 si el resultado es cero y N=1 si el bit de más peso del resultado es 1.

Bcc: (Branch conditionally)

Función: bifurcación condicional.
Variantes: ninguna.
Notación en ensamblador: Bcc etiqueta
Tamaño del operando: 8 y 16 bits.
Observaciones: si se cumple la condición especificada el contador de programa se desplaza al valor que se especifica. El desplazamiento es un entero en la forma de complemento a 2. No afecta las banderas. Cc presenta la condición, existen 14 condiciones que se enumeran al final de este grupo de instrucciones.

Ejemplo: B eq 05
Si el bit Z=1, como resultado de una comparación, el PC se incrementa en 26.

BRA: (Branch Always)

Función: bifurcación incondicional.
Variantes: ninguna.
Notación en ensamblador: BRA etiqueta
Tamaño del operando: 8 y 16 bits.
Observaciones: el contador de programa se desplaza al valor que se especifica. El desplazamiento es un entero en la forma de complemento a 2. No afecta las banderas.

Ejemplo: BRA 05
El PC se incrementa el valor del contenido de 15.

BSR (Branch to Subroutine)

Función: Bifurcación a subrutina.
Variantes: Ninguna.
Notación en ensamblador: BSR etiqueta
Tamaño de operando: 8 bits.
Observaciones: La dirección de la siguiente instrucción se guarda en el Stack, mientras que el PC se altera en el valor del desplazamiento, el cual es un entero en la forma de complemento a 2. Las banderas no son afectadas por esta instrucción.

Ejemplo: BSR 2C

El PC actual se almacena en el Stack y luego se incrementa en 2.

DBcc (Test Condition, Decrement and Branch)

Función: Comprobar condición, decrementar y ramificar.
Variantes: Ninguna.
Notación en ensamblador: DBcc Dn, ea
Tamaño del operando: 16 bits.
Observaciones: Si la condición se cumple, no se realiza ninguna operación y se pasa a la siguiente instrucción. Si la condición no se cumple, los 16 bits menos significativos del registro de datos que actúa como contador se decrementan en una unidad. Si el registro de datos toma el valor valor de -1, el PC se incrementa en 2, pasando a la siguiente instrucción. Si el registro de datos tiene un valor diferente a -1, el PC se altera en el valor del desplazamiento, que viene expresado en la forma de complemento a 2. Las banderas no se afectan.

Esta instruccion tiene semejanza con el FOR de un lenguaje de alta nivel, repitiendo una instruccion el numero de veces que le indica el contador (Dn) hasta que se cumpla cierta condicion.

Ejemplo: DB ne D5,50

Hasta que se cumpla la condicion o bien el valor de D5 llegue a -1 se repetira la siguiente instruccion. Despues de eso se continua con la subsiguiente instruccion.

JMP: (Jump)

Funcion: saltar a direccion.
Variantes: ninguna.
Notacion en ensamblador: JMP <ea>
Tamaño del operando:
Observaciones: Esta clase salto incondicional, no afecta las banderas.

Ejemplo: JMP A5
El PC toma el valor de A5.

JSR: (Jump to Subroutine)

Funcion: saltar a subrutina.
Variantes: ninguna.
Notacion en ensamblador: JSR <ea>
Tamaño del operando:
Observaciones: El programa continua en la direccion que se especifica en la instruccion, la direccion siguiente a la que contiene la instruccion JSR se guarda en el Stack. Las banderas no son afectadas.

Ejemplo: JSR $4AF2

El PC se incrementa y se guarda en el Stack, antes de tomar el valor de $4AF2.

NOP: (No Operation)

Funcion: no opera.

Variantes: ninguna.

Notacion en ensamblador: NOP

Tamaño del operando:

Observaciones: el PC se incrementa en 2.

Ejemplo: NOP

solo se incrementa en 2 el PC.

RTE: (Return from Exception) * instruccion privilegiada

Funcion: regresa de excepcion.

Variantes: ninguna.

Notacion en ensamblador: RTE

Tamaño del operando:

Observaciones: cuando ocurre una excepcion el SR y el PC se salvan en el Sbr. RTE regresa del Stack los valores de Sr y PC. Todas las banderas se afectan. Con esta instruccion se pasa del modo supervisor al modo usuario.

RTR: (Return and Restore Condition Codes)

Funcion: retorno y restauracion de codigos de condicion.

Variantes: ninguna.

Notación en ensamblador: RTR

Tamaño del operando:

Observaciones: el stack devuelve el valor del PC y las banderas de condición

RTS: (Return from subroutine)

Función: retorna de subrutina

Variantes: ninguna.

Notación en ensamblador: RTS

Tamaño del operando:

Observaciones: el stack devuelve el valor que el PC tenía cuando saltó a la subrutina.

MNEV.	TABLA DE CONDICIONES	EXPRESION
EQ	IGUAL A	Z=1
NE	DIFERENTE A	Z=0
MI	MENOR	N=1
PL	MAYOR	N=0
GT	MAYOR QUE (con signo)	Z and [N eor V]=0
LT	MENOR QUE (con signo)	[N eor V]=1
GE	MAYOR O IGUAL QUE (con signo)	[N eor V]=0
LE	MENOR O IGUAL QUE (con signo)	Z or [N eor V] = 1
HI	MAYOR QUE	C and Z = 0
LS	MENOR O IGUAL	C or Z=1
CS	CARRY A 1	C=1
CC	CARRY A 0	C=0
VS	OVERFLOW A 1	V=1
VC	OVERFLOW A 0	V=0

ANDI to CCR:

Función: AND inmediato al registro CCR
Variantes: ninguna.
Notación en ensamblador: ANDI # dato , CCR
Tamaño del operando: 8 bits
Observaciones: N=0 si bit 3 del dato inmediato es 0, Z=0 si el bit 2 del dato inmediato es 0, V=0 si el bit 1 del dato inmediato es 0, C=0 si el bit 0 del dato inmediato es 0 y X=0 si el bit 4 del dato inmediato es 0. En el caso de no ser 0 el bit que corresponde a cada bandera, este último no cambia.

Ejemplo: ANDI #192,CCR
AND inmediato entre el #192H y el reistro CCR.

ANDI to SR * Instruccion Privilegiada

Función: AND inmediato al registro SR
Variantes: ninguna.
Notación en ensamblador: ANDI # dato , SR
Tamaño del operando: 16 bits.
Observaciones: para ejecutar esta instruccion el sistema debe estar en modo supervisor, de lo contrario se produce una excepcion. N=0 si bit 3 del dato inmediato es 0, Z=0 si el bit 2 del dato inmediato es 0, V=0 si el bit 1 del dato inmediato es 0, C=0 si el bit 0 del dato inmediato es 0 y X=0 si el bit 4 del dato inmediato es 0. En el caso de no ser 0 el bit que corresponde a cada bandera, este último no cambia.

Ejemplo: ANDI #192,SR

AND inmediato mete el #192d y el registro SR.

EORI to CCR

Funcion: Or exclusivo entre el dato inmediato y CCR

Variantes: ninguna.

Notacion en ensamblador: EORI #xx, CCR

Tamaño del operando: 8 bits

Observaciones: el resultado de la operacion se almacena en el byte mas significativo del registro de condicion. N cambia si el bit 3 del dato inmediato es 1. Z cambia si el bit 2 del dato inmediato es 1. V cambia si el bit 1 del dato inmediato es 1. C cambia si el bit 0 del dato inmediato es 1. X cambia si el bit 4 del dato inmediato es 1.

Ejemplo: EORI #$13, CCR

en binario el numero 13H es 10011, las banderas que cambian son por lo tanto V, C y X

EORI to SR + Instruccion Privilegiada

Funcion: OR exclusivo entre dato inmediato y SR

Variantes: ninguna.

Notacion en ensamblador: EORI #xxxx, SR

Tamaño del operando: 16 bits.

Observaciones: para ejecutar esta instruccion el sistema debe estar en modo supervisor, de lo contrario se produce una excepcion. Todos los bits del registro de estado cambian. N cambia si el bit 3 del dato

inmediato es 1, Z cambia si el bit 2 del dato inmediato es 1, V cambia si el bit 1 del dato inmediato es 1, C cambia si el bit 0 del dato inmediato es 1, X cambia si el bit 4 del dato inmediato es 1.

Ejemplo: EORI #$2a, CCR

en binario el numero $2a es %00101010, las banderas que cambian son por lo tanto V, Z, C E X.

ORI to CCR

Funcion: Or entre el dato inmediato y CCR

Variantes: ninguna.

Notacion en ensamblador: ORI #xx, CCR

Tamaño del operando: 8 bits

Observaciones: el resultado de la operacion se almacena en el byte mas significativo del registro de condicion. N=1 si el bit 3 del dato inmediato es 1. Z=1 si el bit 2 del dato inmediato es 1. V=1 si el bit 1 del dato inmediato es 1. C=1 si el bit 0 del dato inmediato es 1. X=1 si el bit 4 del dato inmediato es 1.

Ejemplo: ORI #$22, CCR

en binario el numero $22 es 100010, las banderas que se encienden son por lo tanto V, N y Z

EORI to SR + Instruccion Privilegiada

Funcion: OR exclusivo entre dato inmediato y SR

Variantes: ninguna.

Notacion en ensamblador: EORI #xxxx, SR

Tamaño del operando: 16 bits.

Observaciones: para ejecutar esta instrucción el sistema debe estar en modo supervisor, de lo contrario se produce una excepción. Todos los bits del registro de estado cambian. N cambia si el bit 7 del dato inmediato es 1. Z cambia si el bit 6 del dato inmediato es 1. V cambia si el bit 5 del dato inmediato es 1. C cambia si el bit 4 del dato inmediato es 1. X cambia si el bit 4 del dato inmediato es 1.

Ejemplo: EORI #$02, CCR

en binario el numero $02H es 11000000010, las banderas que cambian son por lo tanto V, I_1 e I_0.

ILLEGAL: + Instruccion Privilegiada

Función: provoca una excepción por instrucción ilegal

Variantes: ninguna

Notación en ensamblador: ILLEGAL

Tamaño del operando:

Observaciones: no se afecta el registro de condición, el PC y SR se guardan en el stack, automáticamente se pasa a modo supervisor y el PC se carga con el vector de interrupción 4 (instrucción ilegal).

MOVE from/to CCR:

Función: mover datos entre el CCR y registro destino.

Variantes: ninguna

Notación en ensamblador: MOVE ea , CCR

MOVE CCR, ea

Tamaño del operando: 8 bits

Observaciones:

MOVE from to SR: + Instruccion Privilegiada

Funcion: mover datos entre el SR y registro destino.

Variantes: ninguna

Notacion en ensamblador: MOVE ea , SR

MOVE SR, ea

Tamaño del operando: 16 bits

Observaciones: para ejecutar esta instruccion el sistema debe estar en modo supervisor, de lo contrario se produce una excepcion.

MOVEC: + Instruccion Privilegiada

Funcion: mover de/hacia registro de control.

Variantes: ninguna

Notacion en ensamblador: MOVEC Rc, An

MOVEC Rn, Rc

Tamaño del operando: 32 bits.

Observaciones: no afecta a las banderas de condicion y la transferencia es siempre de 32 bits.

CHK: (Check Register Against Bounds)

Funcion: prueba de registro entre limites.

Variantes: ninguna.

Notacion en ensamblador: CHK <ea>, Dn

Tamaño del operando: 16 bits.

Observaciones: sirve para comprobar si un registro de datos tiene

su contenido comprendido entre unos dietos valores límites. Se comprueba si la palabra menos significativa del registro Dn es mayor que cero o menor que el límite superior, especificado en la instrucción bajo la forma de un entero en complemento a 2. Si no quedara el valor del registro Dn entre estos límites se produce un TRAP con excepción apropiada atendido por el vector número 6. N=1 si Dn < 0; N=0 si Dn > 0. Las demás banderas no se afectan.

Ejemplo: CHK (A2),D5
Se revisa que el valor de Dn sea mayor que cero y menor que el valor de (A2).

RESET: (Reset External Devices) + instrucción privilegiada

Función: reinicializar periféricos (instrucción privilegiada).
Variantes: ninguna.
Notación en ensamblador: RESET
Tamaño del operando:
Observaciones: El 68000 posee una terminal (RESET) que está conectada a todos los dispositivos externos del sistema. La instrucción RESET se ejecuta desde el modo supervisor, activa la terminal durante 124 ciclos de reloj. No afecta a las banderas.

STOP: (Load Status Register and Stop) + instrucción privilegiada
Función: cargar registro de estado y detener al procesador.
Variantes: ninguna.
Notación en ensamblador: STOP # n
Tamaño del operando:

Observaciones: el dato inmediato contenido en la instrucción se carga en SR, el PC avanza a la siguiente instrucción y el CPU detiene la busqueda y ejecución de instrucciones. La ejecución no se reanuda hasta que se produce una interrupción de mayor prioridad que la del proceso en curso, un RESET externo o un trazado. Los códigos de condición cambian según el valor del dato inmediato.

Ejemplo: STOP #0221

El valor 221H equivale a 1000100001, que es el valor con que se carga el SR. Esto equivale a poner una prioridad de interrupción de 100 binario. El procesador reanuda su operación si recibe una interrupción mayor que 100 binario.

TRAP: Trap + instrucción privilegiada

Función: intercepción incondicional.

Variantes: ninguna.

Notación en ensamblador: TRAP #vector

Tamaño del operando:

Observaciones: su funcionamiento es similar al de una interrupción, solo que esta se genera por software. Se pueden emplear 16 vectores, del 32 al 47. Las banderas no son afectadas

Ejemplo: TRAP #2

Se sigue el procedimiento que para una excepción generada externamente, solo que el vector que se carga es el 22H.

TRAPV: Trap on Overflow + instrucción privilegiada

Función: intercepción si hay sobreflujo.

Variantes: ninguna.

Retorna en [illegible] IRAF.

Tamaño del operando:

Observaciones: si V=1 se produce una operación similar a la de IRAF, solo que se carga el vector 7. Si V=0 no se realiza ninguna operación y se pasa a la siguiente instrucción.

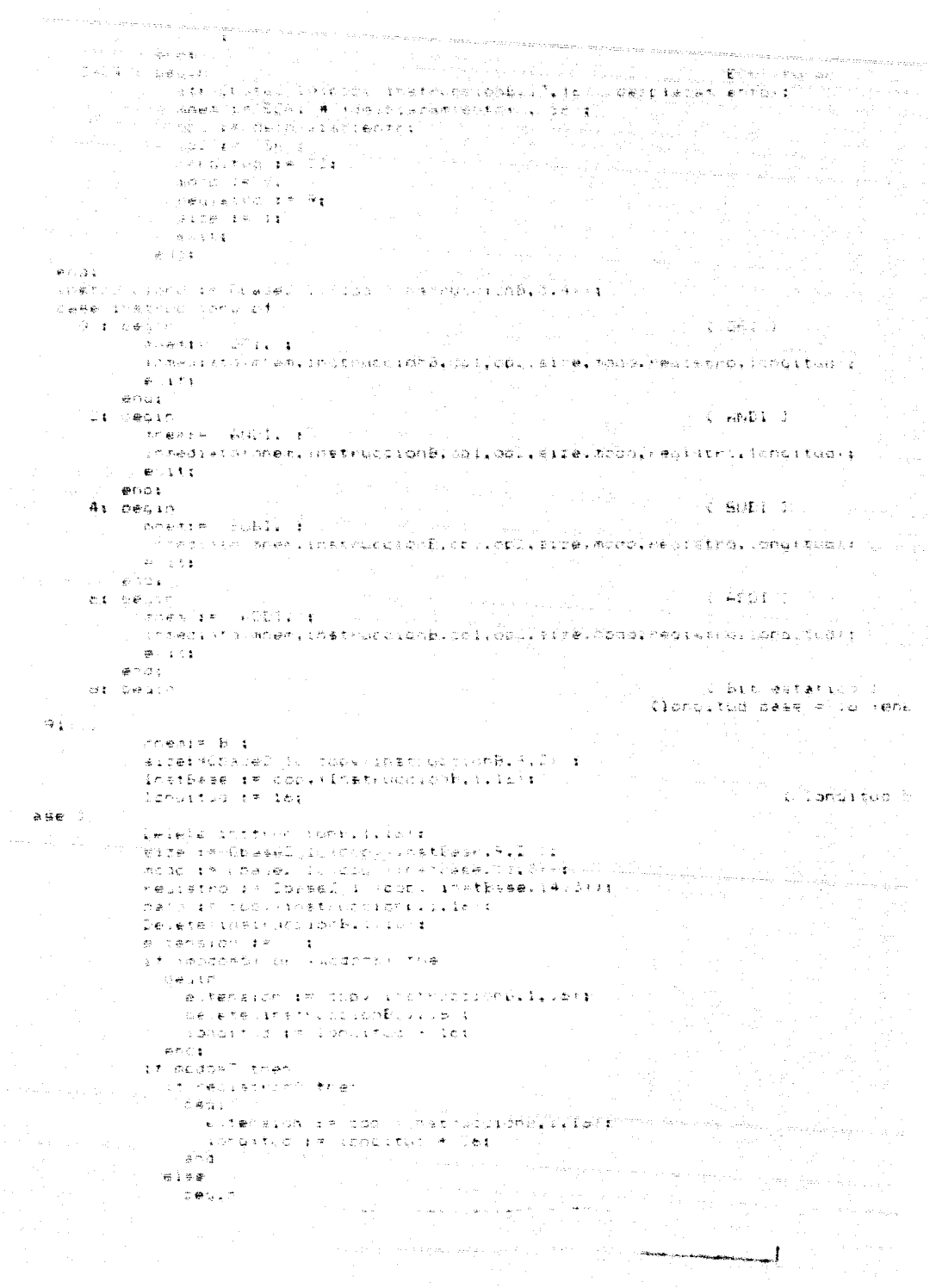
```
{ Programa ensamblador
  Autor: J. Ignacio Gallego Sagastume
  Fecha: 15/Jun/2000
  Version: 1.0 }

Program Ensamblador;

Uses Crt,Dos,Chasm;

Var
   Archivo : Text;
   Nombre  : String;
   Linea   : String;
   Codigo  : String;
   Error   : Integer;
   Dir     : Word;
   Inst    : Byte;
   Num     : Word;
   Cadena  : String;

Begin
   Clrscr;
   Write('Nombre del archivo a ensamblar: ');
   Readln(Nombre);
   Assign(Archivo,Nombre);
   Reset(Archivo);
   While not Eof(Archivo) do
   Begin
      Readln(Archivo,Linea);
      Codigo:=Copy(Linea,1,4);
      Val('$'+Codigo,Dir,Error);
      Codigo:=Copy(Linea,6,2);
      Val('$'+Codigo,Inst,Error);
      Num:=Ord(Inst);
      Cadena:=Mnem(Num);
      Writeln(Dir,' ',Cadena);
   End;
   Close(Archivo);
End.
```

```
                                  : string;
    directorio                    : searchRec;
    f, g                                                  : text;
    i, k,                                                 : word;
    ultimaLinea,
    lineaDeComandoActiva,
    totalLineas,                                          { total de lineas del archivo }
    x, y, z,                                                 { contadores }
    LOCAL_X,
    MODE,
    MCOLOR,
    j,c                           : integer;
    linea                         : string;
    tecla                         : char;
    archivo,                                              { archivo con copia de ed }
    archivo1                                              { archivo con programa desensamblado }
    temp                                                  { lo usa el rutina inicio }
                                  : string;
    fin,                                                  { verdadero si termina de leer archivo }
    ren                           : boolean;
    CodigoObjeto                  : array[1..60] of string;
    dirinfo                       : searchRec;  { record del DOS para directorio}
    col, reng                     : integer;
    mnuSel                        : string;
const lineasvuelta=20;


begin
  clrscr;
  clrscr;
  ventanaCris[1] := 2;          { columna izquierda }
  ventanaCris[2] := 2;          { renglon superior }
  ventanaCris[3] := 79;         { columna derecha }
  ventanaCris[4] := 23;         { renglon inferior }
  window(2,2,79,23);
  textbackground(blue);
  clrscr;
  writeln('Desensamblador de codigo objeto para 8086');
  normvideo;
  gotoxy(1,1+lineaMenu[1]);
  mode:=0;
  write('[',char(24),char(25),'] cambiar opcion [ ,ENTER],  selecciona  [ ',
                                                                  'F1 ayuda ]');
  gotoxy(2,1+lineaMenu[1]);
  write('Desensamblar');
  normvideo;
  highvideo;
  window(ventanaCris[1],ventanaCris[2],ventanaCris[3],ventanaCris[4]);
  selected:=mode in [1,2];{ si esta en ciclo, ventanaCris.caminar}
  repeat
  if mode=0 then          { no se elijio archivo }
    begin
      gotoxy(1,1+lineaMenu[1]);
      write('Desensamblar');
      clreol;
      exit;
    end;
  gotoxy(1,2);
  textbackground(blue);
  write('Archivo:',Archivo);
  writeln('  Inicio:',inicio);
  delete(mnuSel,length(mnuSel),1);
```

```
  writeln;
  write('Desea modificar el fichero del archivo ');
  textcolor(14);
  write(nombre);
  textcolor(7);
  writeln;

  {Verifica que no se haya modificado el fichero del archivo}
  {$I-} assign(textfile,nombre); reset(textfile); {$I+}

  if not (ioresult = 0) then
    begin
      textcolor(12);
      write('No existe el archivo ',nombre);
      textcolor(7);
      writeln('Desea crearlo? (s/n)');
      repeat
      until keypressed;
      respuesta:=readkey;
      if respuesta='s' then
      Clrscr
    end
  else
    begin
      window(1,1,80,25);
      assign(textfile,nombre);
      {$I-} rewrite(textfile); {$I+}
      if ioresult<>0 then
        begin
          Clrscr; writeln('Disco protegido. Coloque la palabra password en mayusculas y confirme escribiendo');
        end
      else
        begin
          close(textfile);
          window(1,1,80,25);
          clrscr;
          write('                                DESEA MODIFICAR ALGUNA LINEA
                ');
          window(1,3,80,25);
          linea := 0;
          finarchivo:=false;
          linealeida:='';
          lineadesencadenada:='';
          repeat
            {$I-} readln(textfile,linea); {$I+}
            if linea <> '' then
              begin
                for i := 1 to length(linea) do
                  lineadesencadenada:=lineadesencadenada+chr(ord(linea[i])-i);
                linea:=lineadesencadenada;
              end;
            inc(i);
            writeln(linea);
            gotoxy(1,1);
            write('Linea a modificar: ');
            readln(nlinea);
            while nlinea<>0 do
              begin
                inc(linea);
                inc(linealeida);
                readln(textfile,cambio[linealeida]);
                gotoxy(1,2);
                write('Linea a modificar: ');
                if linea=nlinea then
```

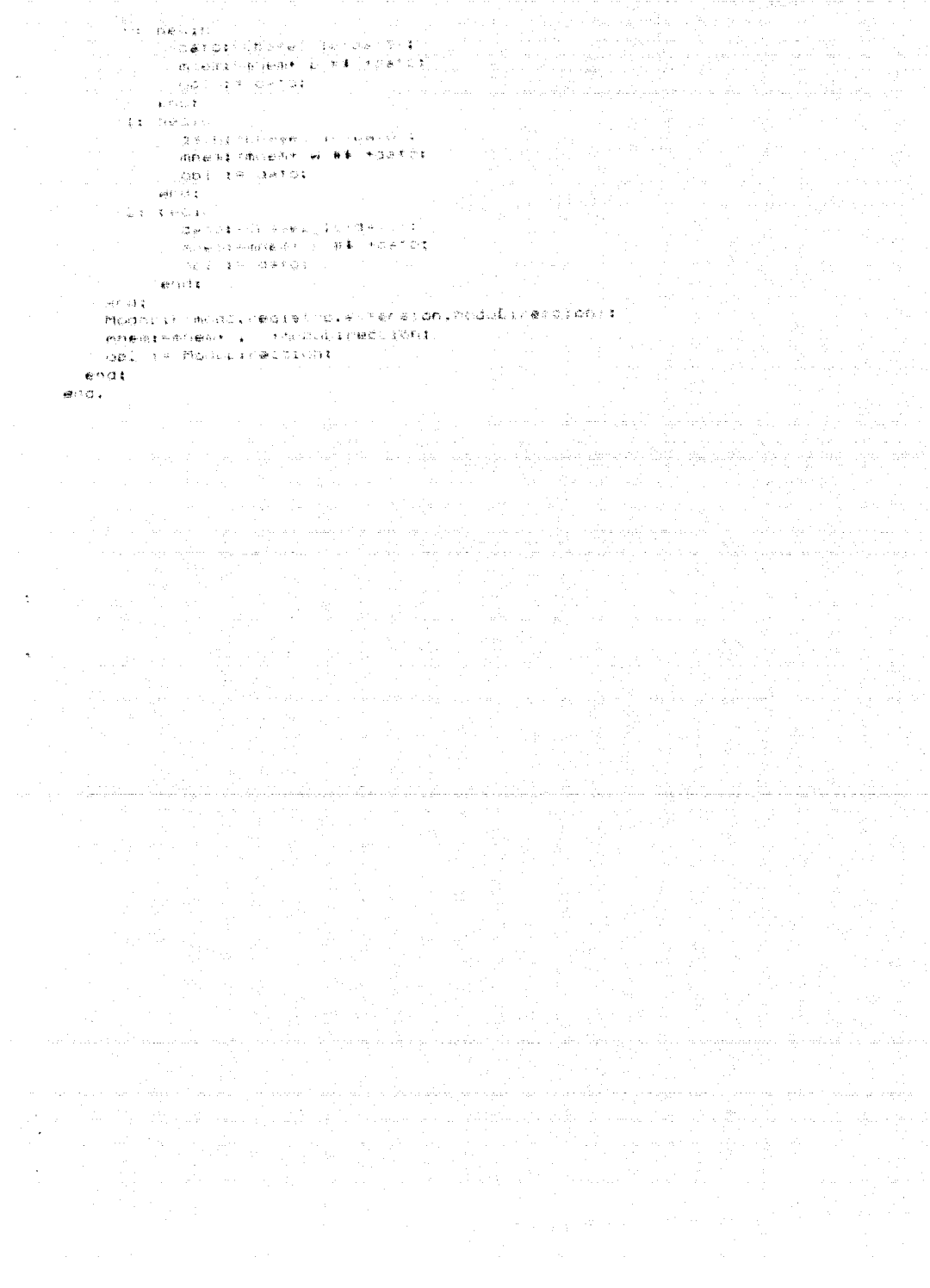
```
                WRITE([illegible]);
                WRITE(TEXTFILE, '[illegible]');
                WRITE(TEXTFILE,[illegible]);
                WRITE([illegible]);
                WRITE([illegible]);
                WRITE([illegible]);
                WRITE([illegible]);
                WRITE(TEXTFILE, ' Registro ');
                WRITE(TEXTFILE,Registro[illegible]);
                WRITE(TEXTFILE, ' Tamaño ');
                WRITE(TEXTFILE,[illegible]);
                if [illegible] then
                  begin
                    [illegible] se termino el desensamblado! [illegible]
                    [illegible];
                    exit
                  end;
              end;
            end;
          close(textfile);
        until fin;
        GotoXY([illegible]);
        HideCursor;
        [illegible];
        write('                    Oprima cualquier tecla para continuar
              ');
        NormVideo;
        inChr:=[illegible];
        inCtl:=[illegible];
        repeat
          Read([illegible],InCtl);
        until [illegible] or inCtl[illegible];
        [illegible];
      end;
    end;
  ClrScr;
end.
```

```
unit nivel1;

{ Nivel 1: Esta unidad incluye rutinas de bajo nivel para decodificar
  instrucciones de 16 bits. Incluye rutinas para determinar el modo
  de direccionamiento, el registro destino, el registro fuente,
  el desplazamiento, el modo de direccionamiento destino,
  el modo de direccionamiento fuente, el tamaño destino,
  el tamaño fuente y el tamaño de la instrucción en bytes.
  En todo caso se requiere la dirección según el modo de
  direccionamiento utilizado.

interface

uses dos,                              { esta unit se utiliza para convertir la
                                         cantidad a cadena de caracteres
                                         (o, viceversa) en el momento de
                                         leer el archivo de comandos }
     Crt;                              { funciones de pantalla de Turbo Pascal }

FUNCIONES QUE UTILIZAN:

procedure nivel1instruccion(           { instrucción en binario }
                    var mnem,          { mnemónico desensamblado }
                        op1,
                        op2            { string }
                    var
                        longitud,      { longitud en bits de la instrucción }
                        modo,          { modo de direccionamiento en decimal }
                        registro,      { registro del modo de direccionamiento en
 decimal }
                        size           { tamaño de la operación en decimal ( 1 o 2 ) }
 : integer);                           { integer }

procedure inmediatos(var mnem                                  : string;
                     instruccion,
                     op1,
                     op2               : string;
                     var size, modo, registro, longitud : integer);

implementation

FUNCTION nivel1;

{ Función para desensamblar codigos cuyos primeros 4 bits son 0000
  Funciones que utilizan: }

var
    fuente,                            { registro afectado }
    destino,                           { registro afectado }
    desplazamiento,                    { según el modo de direccionamiento
                                         se trata del desplazamiento
                                         o del registro }
    modofuente,                        { modo de direccionamiento fuente }
    tipopos,                           { para MOVEP, tipo de registro a
                                         transferir }
    n,                                 { número de registro datos o instrucción }
    dato,                              { dato inmediato }
    instbase,
    variacion
                        : string;
    instruccion0,                      { valor en decimal de alguna parte del
                                         código }
```



```
            end;
          case size of
            1: mnem := mnem + 'B' ;
            2: mnem := mnem + 'W' ;
            4: mnem := mnem + 'D' ;
            8: mnem := mnem + 'Q' ;
          end;
          str(ibase, longitud:0);
          mnem := mnem + ' #' + longitud;
          op1 := dato;
          inmediato:=mnem,instruccion8.op1.op,size,modo.registro.longitud;
          exit;
        end;
    10: begin                                                    { EOF }
          mnem := 'DW ' + longitud;
          inmediato:=mnem,instruccion8.op1.op,size,modo.registro.longitud;
          exit;
        end;
    11: begin
          mnem := 'DB ' + longitud;
          inmediato:=mnem,instruccion8.op1.op,size,modo.registro.longitud;
          exit;
        end;
    14: begin                                                    { MOVE }
          mnem := 'MOV ';
          size:=copy(instruccion,4,2);
          case size of
            '01': mnem := mnem + 'B ' ;
            '11': mnem := mnem + 'W ' ;
            '10': mnem := mnem + 'L ' ;
          end;
          val('$'+copy(instruccion,6,2),reg1,cod);
          if copy(instruccion,8,1) = '0' then
            reg1:=8 + reg1
          else
            reg1:=reg1 + 16;
          reg2:=strtoint('$'+copy(instruccion,7,1));
          if copy(instruccion,5,1) = '0' then
            begin
              mnem := mnem+'D'+inttostr(reg2)+','+inttostr(reg1);
              op1 := modo[reg2];
              op2 := reg1;
            end
          else
            begin
              mnem:= mnem+inttostr(reg1)+','+modo[reg2];
              op1 := reg1;
              op2 := modo[reg2];
            end;
          exit;
        end;
  end;
  if copy(instruccion,1,2)='00' then                             { MOVEP }
    begin
      mnem:= 'MOVEP ';
      str(copy(instruccion,5,3),reg1);
      reg1:= 8 + reg1;
      str(copy(instruccion,13,3),registro);
```

```
        ...
        area := (byte*)&list[0];
        ...
        ...
        ...
        ...
        ...
    end;
end;


{$O+,F+,S-,R-,I-,D-,V-,B-,N-,L-}
{
Algoritmo: Determinamos que tipo de instrucción es dependiendo de su contenido, es decir
           que tipo, modo e instrucción con modo de direccionamiento
           tiene.
}
unit Info_8;  {Version 1.1}

interface

var dat,                                { Indica el byte de datos }
    e_tension,                          { Indica el byte de extension }
    instruc,                            { Indica el byte de instruccion }
    ModoDireccion                       { }
                                : string;

function Direccionamiento(
{ PARAMETRO DE ENTRADA: instruccion  palabra de extension que contiene el
                        dato inmediato.
  PARAMETROS DE SALIDA: cada parte del mnemonico correspondiente al dato
                        ingresado.
}

implementation

  begin
    instruc := '';
    longitud := 0;                                          { longitud base
    delete(instruc,1,1);
    size := Lo(mem[seg(instruccion)^:ofs(instruccion)^]);
    mode := Lo(mem[seg(instruccion)^:ofs(instruccion)^+1]);
    register := Lo(mem[seg(instruccion)^:ofs(instruccion)^+2]);
    if size = 1 then
      begin
        dato := char(mem[seg(instruccion)^:ofs(instruccion)^+3]);
        delete(instruc,1,1);
        longitud := longitud + 1;                           { 1byte }
      end
    else
      begin
        dato := chr(mem[seg(instruccion)^:ofs(instruccion)^+3])+
        delete(instruccion,1,1);
        longitud := longitud + 1;                           { byte or word }
      end;
    e_tension := '';
    if (mode=5) or (mode=6) then
      begin
        e_tension := char(mem[seg(instruccion)^:ofs(instruccion)^+3]);
        delete(instruccion,1,1);
        longitud := longitud + 1;
      end;
    if mode=7 then
      if (register=3) then
        begin
          e_tension := char(mem[seg(instruccion)^:ofs(instruccion)^+3]);
          delete(instruccion,1,1);
          longitud := longitud + 1;
        end;
      else
        begin
          e_tension := char(mem[seg(instruccion)^:ofs(instruccion)^+3]);
          longitud := longitud + 2;
```



```
{ Unidad
  Esta unidad permite definir instrucciones de maquina de 16 bits con
  hasta tres operandos, uno de ellos opcional. Los operandos pueden
  ser: registros, constantes, direcciones de memoria, etiquetas.
  En todos ellos se permite el uso de la notacion de
  direccionamiento utilizada.
}

interface

uses crt,                              { esta unit se utiliza para el retardo, la
                                         condicion y el modo de linea(ansi.sys)
                                         lo, y el puerto en ensamblador }
     Dlaser;                           { funciones de cadir de base 2,10,16}

{
EJEMPLO QUE UTILIZA:
}
PROCEDURE Nivel1(instruccion : string;
                 var mnem,
                     op1,
                     op2
                                : string;
                 var constante,
                     modo,
                     registro,
                     size
                                : integer);
procedure MOVE(var mnem, i_instruccion, op1, op2 : string;
               var constante, modo, registro, size : integer);

implementation

PROCEDURE Nivel1;

{
SINOPSIS: funcion para desensamblar codigos cuyos primeros 4 bits son 0001
EJEMPLO QUE UTILIZA:
}

PARAMETROS DE ENTRADA: instruccion
PARAMETROS DE SALIDA: mnemonico

begin
  MOVE(mnem,instruccion,op1,op2,longitud,modo,registro,size);
end;

PROCEDURE MOVE;
{
SINOPSIS: desensambla unicamente la instruccion MOVE.S
VARIABLES QUE UTILIZA:
}
var
    modoDireccion,
    a_tension
                                : string;
    x,
    modo,
    registro
                                : integer;

PARAMETROS DE ENTRADA: instruccion.
```

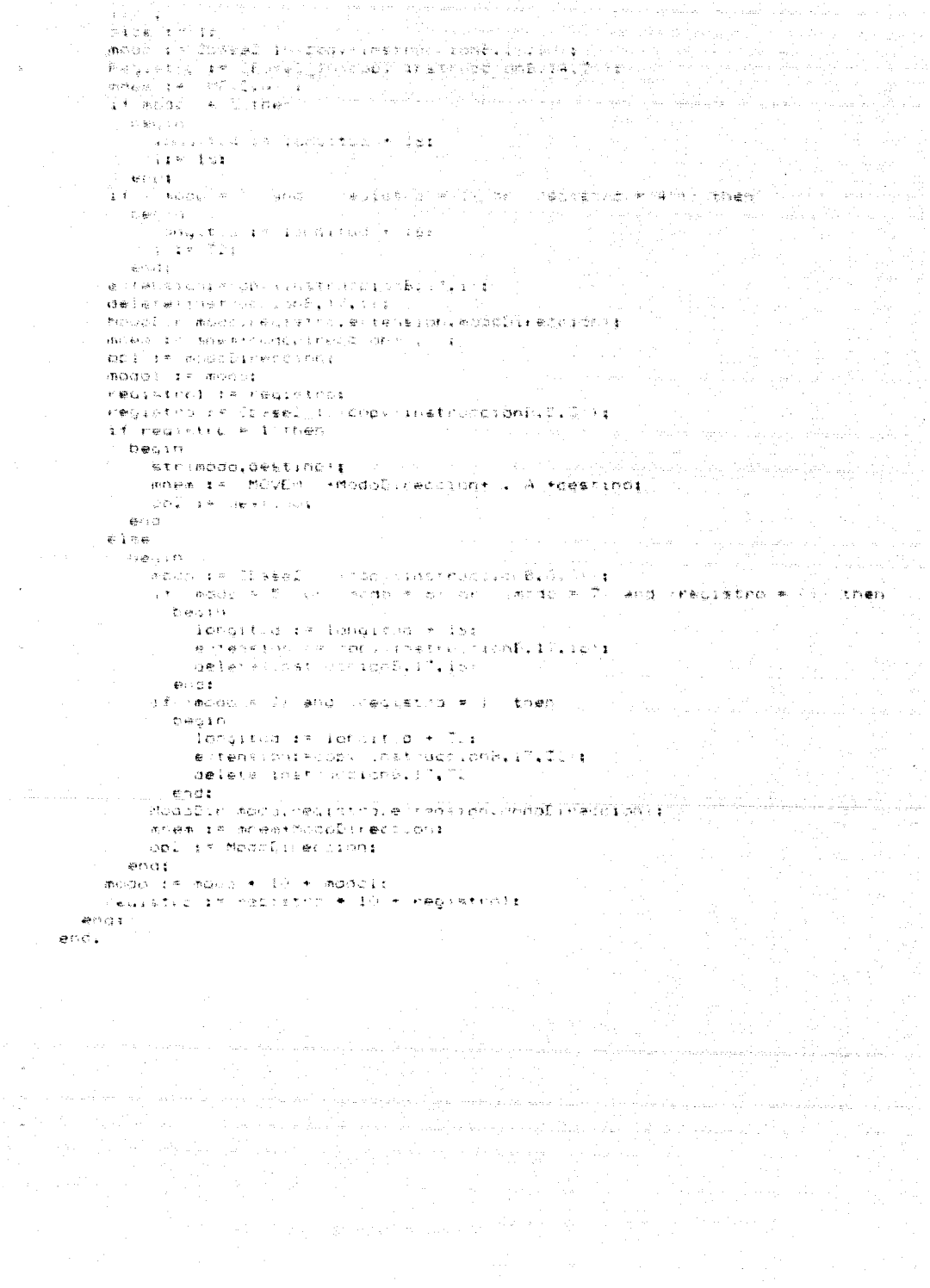
```
    end;
  end.
```

```
unit Unit2;

{ Esta unidade contém as instruções para a criação de uma janela e
  um botão, além de definir a função que será executada quando o botão
  for pressionado. Também contém uma função para mover a janela
  horizontalmente, usando as teclas de seta esquerda e direita.
  Finalmente, contém uma função para fechar a janela quando a tecla
  Esc for pressionada. Essas funções são usadas pela unidade principal
  para criar uma interface gráfica simples. }

interface

uses Unit1;

var
  Janela: TForm;
  Botao: TButton;
  Label1: TLabel;

procedure CriarJanela;
procedure MoverJanela(Direcao: Integer);
procedure FecharJanela;

implementation

procedure CriarJanela;
begin
  Janela := TForm.Create(nil);
  Janela.Caption := 'Minha Janela';
  Janela.Width := 300;
  Janela.Height := 200;

  Botao := TButton.Create(Janela);
  Botao.Parent := Janela;
  Botao.Caption := 'Clique aqui';
  Botao.Left := 100;
  Botao.Top := 100;
  Botao.OnClick := Unit1.Form1.Button1Click;

  Label1 := TLabel.Create(Janela);
  Label1.Parent := Janela;
  Label1.Caption := 'Use as setas para mover a janela';
  Label1.Left := 50;
  Label1.Top := 50;

  Janela.Show;
end;

procedure MoverJanela(Direcao: Integer);
begin
  case Direcao of
    VK_LEFT: Janela.Left := Janela.Left - 10;
    VK_RIGHT: Janela.Left := Janela.Left + 10;
  end;
end;

procedure FecharJanela;
begin
  Janela.Close;
end;

end.
```

```
unit Unit1;

{ Esta unidade contém as instruções para a criação de uma janela e
  um botão, além de definir a função que será executada quando o botão
  for pressionado. Também contém uma função para mover a janela
  horizontalmente, usando as teclas de seta esquerda e direita.
  Finalmente, contém uma função para fechar a janela quando a tecla
  Esc for pressionada. Essas funções são usadas pela unidade principal
  para criar uma interface gráfica simples. }

interface

uses
  Unit2;

var
  Form1: TForm;
  Button1: TButton;
  Label1: TLabel;

procedure Button1Click(Sender: TObject);

implementation

procedure Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Você clicou no botão!';
end;

end.
```

```
unit Unit2;

{ Esta unidade contém as instruções para a criação de uma janela e
  um botão, além de definir a função que será executada quando o botão
  for pressionado. Também contém uma função para mover a janela
  horizontalmente, usando as teclas de seta esquerda e direita.
  Finalmente, contém uma função para fechar a janela quando a tecla
  Esc for pressionada. Essas funções são usadas pela unidade principal
  para criar uma interface gráfica simples. }

interface

uses
  Unit1;

var
  Form2: TForm;
  Button2: TButton;
  Label2: TLabel;

procedure Button2Click(Sender: TObject);

implementation

procedure Button2Click(Sender: TObject);
begin
  Label2.Caption := 'Você clicou no botão!';
end;

end.
```

```
unit Unit3;

{ Esta unidade contém as instruções para a criação de uma janela e
  um botão, além de definir a função que será executada quando o botão
  for pressionado. Também contém uma função para mover a janela
  horizontalmente, usando as teclas de seta esquerda e direita.
  Finalmente, contém uma função para fechar a janela quando a tecla
  Esc for pressionada. Essas funções são usadas pela unidade principal
  para criar uma interface gráfica simples. }

interface

uses
  Unit2;

var
  Form3: TForm;
  Button3: TButton;
  Label3: TLabel;

procedure Button3Click(Sender: TObject);

implementation

procedure Button3Click(Sender: TObject);
begin
  Label3.Caption := 'Você clicou no botão!';
end;

end.
```



```
unit Unit1;

{ SINOPSIS: Esta unidad incluye rutinas que se pueden utilizar para
            convertir cadenas a enteros y viceversa, y en el tratamiento correcto
            de errores en estas conversiones.
            Además, se define una función, llamada, MENSAJE, que se encarga de
            la tarea de mostrar un mensaje de error en la línea de mensajes del
            procesador de órdenes. }

interface

USES DE UTILIDAD:

uses Crt,                               { para utilizar la función ReadKey }
     Dos;                               { funciones de tratamiento de fecha y hora }

{
RUTINAS QUE UTILIZA:
}
function Nivel1(mensaje,mensaje2 : string;
                var mens, ord, ord2 : string;
                var longitud,
                    modo,
                    resultado,
                    size
                      : integer);
procedure MOVE(var mens, instruccion, ord, ord2 : string;
               var longitud,modo,resultado,size : integer);
implementation

function Mensaje;
{
SINOPSIS: Función que se encarga de mostrar mensajes de error en el
VARIABLES QUE UTILIZA:
}
{
PARAMETROS DE ENTRADA: ninguno.
PARAMETROS DE SALIDA: mensaje.
}
begin
  MOVE(mensaje,instruccion,ord,ord2,longitud,modo,resultado,size);
end;

PROCEDURE MOVE;
{
SINOPSIS: descodifica únicamente la instrucción MOVE.W
VARIABLES QUE UTILIZA:
}
var
    modo,resultado,
    i,
    mode,
    resultado
                                : integer;
{
PARAMETROS DE ENTRADA: ninguno.
PARAMETROS DE SALIDA: mens,
```

```
(*$I- ,$V- *)

{ Initialize the virtual machine by setting the instruction counter to zero,
  and the stack pointer to the first location in the stack.  Also, set the
  memory size to the number of memory locations available to the program.
  Finally, set the number of command line arguments to the value passed
  in the parameter list. }

procedure InitializeVM;

begin { InitializeVM }
  ic := 0;
  sp := 0;
  memsize := MaxMem;
  argcount := argc;
end { InitializeVM };

{ Routines for initializing the instruction table }

procedure Define(mnemonic: string; opcode, op1, op2: integer);

begin { Define }
  with instructions[opcode] do
    begin
      name := mnemonic;
      op1 := op1;
      op2 := op2;
    end { with };
end { Define };

procedure InitializeInstructions;

begin { InitializeInstructions }
  Define('NOP', 0, 0, 0);
  Define('PUSH', 1, 1, 0);
  Define('POP', 2, 1, 0);
  Define('DUP', 3, 0, 0);
  Define('ADD', 4, 0, 0);
  Define('SUB', 5, 0, 0);
  Define('MUL', 6, 0, 0);
  Define('DIV', 7, 0, 0);
  Define('MOD', 8, 0, 0);
  Define('NEG', 9, 0, 0);
  Define('INC', 10, 1, 0);
  Define('DEC', 11, 1, 0);
  Define('EQ', 12, 0, 0);
  Define('NE', 13, 0, 0);
  Define('LT', 14, 0, 0);
  Define('LE', 15, 0, 0);
  Define('GT', 16, 0, 0);
  Define('GE', 17, 0, 0);
  Define('JMP', 18, 1, 0);
  Define('JMPZ', 19, 1, 0);
  Define('JMPNZ', 20, 1, 0);
  Define('CALL', 21, 1, 0);
  Define('RET', 22, 0, 0);
  Define('PUSHLOC', 23, 1, 0);
  Define('PUSHPAR', 24, 1, 0);
  Define('PUSHIND', 25, 0, 0);
  Define('POPIND', 26, 0, 0);
  Define('PUSHARG', 27, 1, 0);
  Define('HALT', 28, 0, 0);
end { InitializeInstructions };

{ Routines for initializing the memory }

procedure InitializeMemory;

var
  i: integer;

begin { InitializeMemory }
  for i := 0 to MaxMem do
    memory[i] := 0;
end { InitializeMemory };

{ Routines for initializing the stack }

procedure InitializeStack;

var
  i: integer;

begin { InitializeStack }
  for i := 0 to MaxStack do
    stack[i] := 0;
end { InitializeStack };

{ Routines for initializing the program }

procedure InitializeProgram;

begin { InitializeProgram }
  for ic := 0 to MaxProg do
    with program[ic] do
      begin
        opcode := 0;
        op1 := 0;
        op2 := 0;
      end { with };
  ic := 0;
end { InitializeProgram };

{ Routines for initializing the symbol table }

procedure InitializeSymbols;

var
  i: integer;

begin { InitializeSymbols }
  for i := 0 to MaxSymbol do
    with symbols[i] do
      begin
        name := '';
        value := 0;
      end { with };
  nextsymbol := 0;
end { InitializeSymbols };

{ Routines for initializing the fixup table }

procedure InitializeFixups;

var
  i: integer;

begin { InitializeFixups }
  for i := 0 to MaxFixup do
    with fixups[i] do
      begin
        name := '';
        location := 0;
      end { with };
  nextfixup := 0;
end { InitializeFixups };

{ Routines for initializing the virtual machine }

procedure InitializeVM;

begin { InitializeVM }
  InitializeInstructions;
  InitializeMemory;
  InitializeStack;
  InitializeProgram;
  InitializeSymbols;
  InitializeFixups;
end { InitializeVM };
```

```
                end;
              $F2:
3E3A : begin
                mnem := 'SED' ;                                 { SED }
                exit;
              end;
3E3F : begin
                mnem := 'SBC' ;                                 { SBC }
                exit;
              end;
3E44 : begin
                mnem := 'CLD' ;                                 { CLD }
                exit;
              end;
3E49 : begin
                mnem := 'PLX' ;                                 { PLX }
                exit;
              end;
3E4E : begin
                mnem := 'TCS' ;                                 { TCS }
                exit;
              end;
3E53 : begin
                mnem := 'PEA' ;                                 { PEA }
                exit;
              end;
3E58 : begin
                mnem := 'TAX' ;                                 { TAX }
                exit;
              end;
3E5D : begin
                mnem := 'PLB' ;                                 { PLB }
                exit;
              end;
3E62 : begin
                mnem := 'TXY' ;                                 { TXY }
                exit;
              end;
3E67 : begin                                                    { MOVE }
                str(Cbase( ... ).instruction^.i,s);
                regControl := cbase( ... ).instruction^.i;
                location := 2;
                size := 2;
                case regControl of
                  0: registroCon := 'C' ;
                  1: registroCon := 'DP' ;
                  2048: registroCon := 'AP' ;
                  2049: registroCon := 'DESTINO BASE' ;
                end;
                if cbase( ... ).instruction^.i = 0 then
                  begin
                    mnem := 'MVP' + s + ' ' + registroCon;
                    op1 := '$' + s;
                  end
                else
                  begin
                    mnem := 'MVN' + s + ' ' + registroCon;
                    op1 := '$' + s;
                  end;
                op2 := registroCon;
                exit;
              end;
3E7F : begin                                                    { MOVSC }
                str(Cbase( ... ).instruction^.i,s);
                regControl := cbase( ... ).instruction^.i;
                location := 2;
                size := 2;
                case regControl of
                  0: op1 := 'BIT' ;
```

```
                    Low : Word := .LOCATION.BASE;
                  end;
                if (Operation = Operation_Type'(Store)) then
                  begin
                    Opc := 0 + Register;
                    Write (Code, Opc, Low, High);
                  end
                else
                  begin
                    Opc := 8 + Register;
                    Write (Code, Opc, Low, High);
                  end;
              end;
          end;
      end;
    Instruction := Table(Instruction.Next);
  end Instruction;
  case Instruction.Op of
    2#0000# : begin                                          { SWAP }
                Str(Code^[0].Instruction.Length,Register);
                Longitud := 1;
                Size := 1;
                Mnem := 'SWAP ' + Register;
                Opc := 2 + Register;
                Exit;
              end;
    2#0001# : begin                                          { DEFT }
                Str(Code^[0].Instruction.Length,Register);
                Mnem := 'DEFT ' + Register;
                Opc := Register;
                Longitud := 1;
                Exit;
              end;
    2#0010# : begin                                          { EXT.W }
                Str(Code^[0].Instruction.Length,Register);
                Longitud := 1;
                Size := 1;
                Mnem := 'EXT.W ' + Register;
                Opc := 3 + Register;
                Exit;
              end;
    2#0011# : begin                                          { EXT.L }
                Str(Code^[0].Instruction.Length,Register);
                Longitud := 1;
                Size := 2;
                Mnem := 'EXT.L ' + Register;
                Opc := 4 + Register;
                Exit;
              end;
    4#0000# : begin                                          { LINK }
                Str(Code^[0].Instruction.Length,Register);
                Val := 4 + Register;
                Longitud := 2;
                Extension := Code^.Instruction.Length;
                If Code^.Instruction.Length < 0 then
                  Extension := Negative(Extension)
                Else
                  Str(Extension:Length,Extension);
                Opc := 8 + Register;
                Mnem := 'LINK ' + Opc + ' # ' + Val;
                Exit;
              end;
    4#0001# : begin                                          { UNLK }
                Str(Code^[0].Instruction.Length,Register);
                Longitud := 1;
```

```
```

```
    mem := [mode = DIR]  + MODE(dir) + reg;
    GET ( word );
    DEC := MODE(dir) ;
    opl := MODE(dir) ;
    emit;
  end;
1: begin                                                    { MOVE TO SR }
    mode := Decode( word ).instruction.ea.mode;
    register := Decode( word ).instruction.ea.reg;
    size := 2;
    longword := 1e;
    if mode = 7 then
      begin
        register := register * 16;
        if mode = 7 and register = 12 then
          longword := longword + 1e;
      end;
    i := longword;
    if i in [0..7] then
      emit(instruction.ea.instruction,i,'.w')
    MOVE( mode,register,emit,memory,MOVE(direction));
    mem := [MOVE  +MOVE(direction) , SR ];
    dec := [DEC ];
    opl := MOVE(direction);
    emit;
  end;
2: begin                                                    { MOVE TO SR }
    mode := Decode( word ).instruction.ea.mode;
    register := Decode( word ).instruction.ea.reg;
    size := 2;
    longword := 1e;
    if mode = 7 then
      begin
        register := register * 16;
        if mode = 7 and register = 12 then
          longword := longword + 1e;
      end;
    i := longword;
    if i in [0..7] then
      emit(instruction.ea.instruction,i,'.w');
    MOVE( mode,register,emit,memory,MOVE(direction));
    mem := [MOVE  +MOVE(direction) , SR ];
    dec := [DEC ];
    opl := MOVE(direction);
    emit;
  end;
```

```
        register := (base(instruction[11:7]));
        size := 2;
        longitud := 0;
        if mode = 0 then
          begin
            longitud := (base(instruction[14:12]));
            if (mode = 0 and register = 0) then
              longitud := longitud + 1;
          end;
        i := longitud;
        if i != 0 then
          extern/static procedure (size,i,mode);
        Modificador.registro := register.Modificador;
        mode := mode + (base(instruction[6:0]));
        ori := Modificador;
        exit;
      end;
    47: begin                                                    { lw }
        mode := (base(instruction[31:27]));
        register := (base(instruction[19:15]));
        size := 4;
        longitud := 0;
        if mode = 0 then
          begin
            longitud := longitud + 0;
            if (mode = 0 and register = 0) then
              longitud := longitud + 1;
          end;
        i := longitud;
        if i != 0 then
          extern/static procedure (size,i,mode);
        Modificador.registro := register.Modificador;
        mode := (base(instruction[6:0]));
        ori := Modificador;
        exit;
      end;
    50: begin                                                    { jsr }
        mode := (base(instruction[11:7]));
        register := (base(instruction[14:12]));
        size := 4;
        longitud := 0;
        if mode = 0 then
          begin
            longitud := longitud + 0;
            if (mode = 0 and register = 0) then
              longitud := longitud + 1;
          end;
        i := longitud;
        if i != 0 then
          extern/static procedure (size,i,mode);
        Modificador.registro := register.Modificador;
        mode := (jal - instruction);
        ori := Modificador;
        exit;
      end;
    51: begin                                                    { jmp }
        mode := (base(instruction[11:7]));
        register := (base(instruction[14:12]));
        size := 4;
        longitud := 0;
        if mode = 0 then
          begin
            longitud := longitud + 0;
            if (mode = 0 and register = 0) then
```

```
```

```
      if (sw) (registro ... [illegible] ... ) then
        begin
          ...[illegible];
          listaregs(listaregs, ...
        end;
      i := length(listaregs);
      delete(listaregs,i,1);
      listaregs := listaregs + ' ';
      if modo > 5 then
       begin
          longitud := longitud + 2;
          if (modo = 7) and (registro = 4) then
            longitud := longitud + 2;
       end;
      i := longitud;
      if i > 2 then
        extension(...(instruccion, ...,7),i+12);
      modoDir(modo,registro,extension,op2);
      op1 := listaregs;
      mnem := mnem + op1 + ',' + op2;
      exit;
     end;
end;
instruccionD := Cbase2_10(copy(instruccionB,5,4));
case instruccionD of
   0: begin                                                 { NEGX }
        longitud := 2;
        modo := Cbase2_10(copy(instruccionB,11,3));
        registro := Cbase2_10(copy(instruccionB,14,3));
        size := Cbase2_10(copy(instruccionB,9,2));
        mnem:= 'NEGX.';
        case size of
          0: mnem:=mnem+'B ';
          1: mnem:=mnem+'W ';
          2: mnem:=mnem+'L ';
        end;
        if modo > 5 then
          begin
            longitud := longitud + 2;
            if (modo = 7) and (registro = 4) then
              longitud := longitud + 2;
          end;
        i := longitud;
        if i>2 then
          extension:=copy(instruccionB,17,i*8-16);
        modoDir(modo,registro,extension,op1);
        mnem := mnem+op1;
        exit;
      end;
   2: begin                                                 { CLR }
        longitud := 2;
        modo := Cbase2_10(copy(instruccionB,11,3));
        registro := Cbase2_10(copy(instruccionB,14,3));
        size := Cbase2_10(copy(instruccionB,9,2));
        mnem:= 'CLR.';
        case size of
          0: mnem:=mnem+'B ';
          1: mnem:=mnem+'W ';
          2: mnem:=mnem+'L ';
        end;
        if modo > 5 then
          begin
            longitud := longitud + 2;
            if (modo = 7) and (registro = 4) then
```

```
        ...
        if ... then
          ...
        ...
        ...
        ...
      end;
    4: begin                                                    { MEM }
        location := lo;
        mode := byte(instruction^.instruction[1]);
        register := byte(instruction^.instruction[2]);
        size := byte(instruction^.instruction[3]);
        mregs := [ax, ...
        case size of
            1: mregs := mregs + [...
            2: mregs := mregs + [...
            4: mregs := mregs + [...
        end;
        if mode = 5 then
          begin
            longitud := longitud + lo;
            if (mode = 7) and (register = 0) then
              longitud := longitud + lo;
          end;
        i := longitud;
        if i in then
          ...
        ...
        mem := ...
        ...
      end;
    5: begin                                                    { MOV }
        location := ...;
        mode := byte(instruction^.instruction[1]);
        register := byte(instruction^.instruction[2]);
        size := byte(instruction^.instruction[3]);
        mregs := [ax, ...
        case size of
            1: mregs := mregs + [...
            2: mregs := mregs + [...
            4: mregs := mregs + [...
        end;
        i := length(instruction);
        if i in ...
          ...
        ...
        mem := mem + mode;
        ...
      end;
    6: begin                                                    { TAT }
        location := ...;
        mode := byte(instruction^.instruction[1]);
        register := byte(instruction^.instruction[2]);
        size := byte(instruction^.instruction[3]);
        mregs := [ax, ...
        case size of
            1: mregs := mregs + [...
            2: mregs := mregs + [...
            4: mregs := mregs + [...
        end;
        if mode = 5 then
          begin
            longitud := longitud + lo;
            if (mode = 7) and (register = 0) then
```

```
```

```
unit nivel2;

[illegible] : Esta unidad ... [illegible] ... [illegible]
        [illegible] ... [illegible]
        [illegible]
        [illegible], [illegible], [illegible] fuente, cuenta, [illegible]
        En todo caso se respeta siempre la sintaxis, segun el modo de
        direccionamiento utilizado.

interface

UNITS QUE UTILIZA:

uses crt,                                    este unidad se utiliza para obtener la
                                             condicion y el modo de direccionamien-
                                             to, [illegible]
                                             [illegible]
     ubase;                                  funciones de cambio de base [illegible]

RUTINAS QUE UTILIZA:

PROCEDURE nivel2(instruccion : string;
                 var mnem, op1, op2 : string;
                 var longitud,
                     modo,
                     registro,
                     size
                               : integer);

implementation

PROCEDURE nivel2;

SINOPSIS : funcion para desensamblar codigos cuyos primeros 4 bits son 0101
VARIABLES QUE UTILIZA:

var
    registro1,
    cond,
    extension,
    modoDireccion,
    dato
                          : string;
    instruccionD
                          : integer;

PARAMETROS DE ENTRADA: instruccionS,
PARAMETROS DE SALIDA: ninguno

begin
  op1 := NULL ;
  op2 := NULL ;
  if copy(instruccionS,9,[illegible]) = [illegible]  then              { DBcc }
    begin
      modo := [illegible];
      registro := [illegible];
      longitud := [illegible];
      size := [illegible];
      op1 := D [illegible](instruccionS,14,[illegible]);
      cond:=condicion(copy(instruccionS,5,4));
      mnem:= 'DB' +[illegible] ,  +op1;
      exit;
    end;
```

```
     modo := Cbase10(copy(instruccionB,11,3));
     registro := Cbase10(copy(instruccionB,14,3));
     longitud := 16;
     size := 0;
     if modo [illegible] 5 then
       begin
         longitud := longitud + 16;
         if (modo = 7) and (registro = 1) then
           longitud := longitud + 16
       end;
     l:=longitud;
     if l [illegible] 16 then
       extension:=copy(instruccionB,17,l-16);
     [illegible](modo,registro,extension,[illegible]);
     [illegible]
     mnem:= [illegible] + [illegible];
     exit;
   end;
if [illegible] then                                          { MODO 1 }
   begin
     modo := Cbase10(copy(instruccionB,11,3));
     registro := Cbase10(copy(instruccionB,14,3));
     longitud := 16;
     size := 0;
     if modo [illegible] 5 then
       begin
         longitud := longitud + 16;
         if (modo = 7) and (registro = 1) then
           longitud := longitud + 16
       end;
     l:=longitud;
     case size of
       0: mnem := 'ADD.B #';
       1: mnem := 'ADD.W #';
       2: mnem := 'ADD.L #';
     end;
     str(Cbase10(copy(instruccionB,5,3)),dato);
     if l>16 then
       extension:=copy(instruccionB,17,l-16);
     [illegible](modo,registro,extension,[illegible]);
     mnem:=mnem+dato+','+[illegible];
     exit;
   end
else
   begin
     modo := Cbase10(copy(instruccionB,11,3));
     registro := Cbase10(copy(instruccionB,14,3));
     longitud := 16;
     size := 0;
     if modo [illegible] 5 then
       begin
         longitud := longitud + 16;
         if (modo = 7) and (registro = 1) then
           longitud := longitud + 16
       end;
     l:=longitud;
     case size of
       0: mnem := 'SUB.B #';
       1: mnem := 'SUB.W #';
       2: mnem := 'SUB.L #';
     end;
     str(Cbase10(copy(instruccionB,5,3)),dato);
     if l>16 then
       extension:=copy(instruccionB,17,l-16);
```

```
unit Pascal;

{Importante: Esta unidad (pascal.tpu) no interactua con ninguna otra unidad
 con excepción de la unidad System y Crt(para el manejo de errores en
 ejecución). La declaración de procedimientos de
 interrupción, ejemplo, como el procedimiento Keyboard, deberán
 ir todos juntos en secuencia siempre la anterior sea, o si no hay una
 otra interrupción utilizada.}

interface

uses Crt;

var tra,                          {esta variable se emplea para saber la
                                   posición a la cual se tiene que
                                   ir. y en el caso de existir registro de un
                                   campo diferente lo indica}
    Clase1,                       {variables de cadena de tipo Char}

{Rutinas que utilizan}

PROCEDURE Nivelinstruccion5 : string;
                  var mem,obj,cc2      : string;
                  var longitud,
                      modo,
                      registro,
                      size
                                 : integer;


implementation

PROCEDURE Nivel5;
{Simplifica: función para concatenar codigos cuyos primeros 4 bits son uno}
{Variables que utiliza:}
var
    cc1,
    cc2,
    operacion
                                 : string;
    i                            : integer;

{RUTINAS QUE UTILIZA: cc2a1a
VARIABLES DE ENTRADA: instruccion5
VARIABLES DE SALIDA: ninguno.}

begin
  cc1 := 'NULL';
  cc2 := 'NULL';
  i:=Length(instruccion5.P.5);
  if i = 2 then
    begin
      longitud := 1;
      size := 0;
      operacion:= '.B' + Clase2[instruccion5.P.5];
    end
  else
    begin
      longitud := 2;
      size := 1;
      operacion:= '.W' + Clase2[instruccion5.P.5];
```

```
unit nivel1;

{ Esta unidad contiene todas las instrucciones de procesamiento de
  bajo nivel, es decir todas las instrucciones que manejan
  registros, las de entrada y salida de archivos en
  general, y las de control de flujo, como saltos, llamadas a
  rutinas, etc. Todas las rutinas de esta unidad se llaman
  directamente desde el intérprete. }

interface

uses CTE;

     Crt;                          { Rutinas de pantalla de Turbo 3.0 }

{
RUTINAS QUE UTILIZA:
}
PROCEDURE Nivel1(instruccion$ : string);
                 var anem,
                     op1,
                     op2      : string;
                 var longitud,
                     modo,
                     registro,
                     size
                              : integer );

procedure MOVEW(var anem, instruccion$ : string;
                var longitud, modo, registro, size : integer);
implementation

PROCEDURE Nivel1;
{
SINOPSIS: función para desensamblar codigos cuyos primeros 4 dígitos son 1111
VARIABLES QUE UTILIZA:

}
RUTINAS QUE UTILIZA: MOVEW
PARAMETROS DE ENTRADA: instruccion$
PARAMETROS DE SALIDA: ninguno
}
begin
  MOVEW(anem,instruccion$,longitud,modo,registro,size);
end;

PROCEDURE MOVEW;
{
SINOPSIS: funcion que desensambla la instruccion MOVEW
VARIABLES QUE UTILIZA:
}
var
    n,
    auto,
    op1,
    op2
                  : string;
    codigo,
    i
                  : integer;
```

```
unit rutinas3;

{
SINOPSIS: Esta unidad provee rutinas de interfaccion entre sistema y bits
          con los que trabaja el cpu. Decodifica el codigo de operacion
          ubicado en la estructura del contador de instrucciones (pc)
          determina, cuando fuere, codigo, registro, destino destino
          en todo caso se resuelve sintaxis segun el modo de
          direccionamiento utilizado.
}
interface

{
UNITS QUE UTILIZA:
}
uses crt,                               { esta unidad se utiliza para colocar la
                                          condicion y el modo de direccionamien
                                          to, y el numero decimal negativo de un
                                          numero binario negativo}
     dos;                               { funciones de cambio de base 2,10,16 }
{
RUTINAS QUE UTILIZA:
}
FUNCTION Nivel0instruccion0 : string;
                var mem,
                    opl,
                    opc       : string;

                var longitud,
                    modo,
                    registro,
                    size
                                : integer;
procedure DivSinVar(mem,instruccion0,opl,opc:string;
                var longitud, modo, registro,size : integer);
procedure DivConVar(mem, instruccion0,opl,opc:string;
                var longitud, modo, registro,size : integer);

procedure IdConVar(mem, instruccion0,opl,opc:string;
                var longitud, modo, registro,size : integer);
procedure IdSinVar(mem, instruccion0,opl,opc:string;
                var longitud, modo, registro,size : integer);
implementation

FUNCTION Nivel0:

{
SINOPSIS: Funcion Debe desensamblar codigos de operacion de 8 bits con 1 o
VARIABLES QUE UTILIZA:
}
var
    bit0
    : string;
{
RUTINAS QUE LLAMA: Dec, Bin, Hex, Decb:
ARCHIVOS DE ENTRADA: INSTRUCCIONO,
ARCHIVOS DE SALIDA: ninguno
}
begin
  bit0:=decb(instruccion0.0.3);
  if bit0= '111' then
    begin
      size := 1;
      modo := Dec(' ' + decb(instruccion0.11.7));
      registro := Dec(' ' + decb(instruccion0.11.9));
      longitud := 1;
```

```
            longitud := longitud * 16;
            if modo = 7 and registro = 0 then
              longitud := longitud * 16
          end;
        DirectMem.instruccionb.out.adc.longitud.modo.registro.size := 1
        exit
      end;
    if bit = nil  then
      begin
        size := 1;
        modo := CBase2_Intcopyinstruccionb.ii.2];
        registro := CBase2_Intcopyinstruccionb.ii,2];
        longitud := 16;
        if modo  = 0 then
          begin
            longitud := longitud * 16;
            if modo = 7  and registro = 0 then
              longitud := longitud * 16
          end;
        DirectMem.instruccionb.out.adc.longitud.modo.registro.size;
        exit;
      end;
    if copyinstruccionb.ii = 0  then
      begin
        modo := CBase2_Intcopyinstruccionb.ii,2];
        registro := CBase2_Intcopyinstruccionb.ii,2];
        longitud := 16;
        if modo  = 0 then
          begin
            longitud := longitud * 16;
            if modo = 7  and registro = 0 then
              longitud := longitud * 16
          end;
        IOMem.instruccionb.out.adc.longitud.modo.registro.size;
        exit;
      end
    else
      if cod_instruccionb.ii = 0  then
        begin
          DirectMem.instruccionb.out.adc.longitud.modo.registro.size;
          exit;
        end
      else
        begin
          IOMem.instruccionb.out.adc.longitud.modo.registro.size;
          exit;
        end;

end;

PROCEDURE Divb;
{
SINOPSIS: termina de desensamblar la instruccion Divb
VARIABLES QUE UTILIZA:
}
var n,
    extension,
    ModRM:integer
                    :string;
{
PARAMETROS DE ENTRADA: instruccionb
PARAMETROS DE SALIDA: MNEM
}
  begin
```

```
    end;

PROCEDURE LIW;

{
SINOPSIS: termina de desensamblar la instruccion LIW
VARIABLES QUE UTILIZA:
}
var n,
    extension,
    ModoDireccion
                    :string;
{
VARIABLES DE ENTRADA: instruccionB
VARIABLES DE SALIDA: mnem

  begin
    mnem := 'LIW  ';
    desplazamiento:= 1;
    str(instruccionB[desplazamiento+1],n);
    if modo = 5 then
      begin
        longitud := longitud + 1b;
        if (modo = '1') and (registro = '0') then
          longitud := longitud * 1b;
      end;
    if i = 1b then
      desplazamiento:=desplazamiento+1b;
    ModoDireccion:=modo(instruccionB[desplazamiento],extension);
    mnem := mnem+', ' + n;
    opc := 3 +n;
  end;

PROCEDURE LOR;
{
SINOPSIS: termina de desensamblar la instruccion LOR
VARIABLES QUE UTILIZA:
}
var n,
    extension,
    ModoDireccion,
    bit
                    :string;
{
VARIABLES DE ENTRADA: instruccionB
VARIABLES DE SALIDA: mnem
}
  begin
    mnem := 'LOR  ';
    bit := copy(instruccionB,7,2);
    if bit='01' then
      begin
```

```
        end;
      if bit2= 0 then
        begin
          mnem := mnem+ ',W' ;
          size := 1;
        end;
      if bit2= 1 then
        begin
          mnem := mnem+ ',L' ;
          size := 2;
        end;
      desplazamiento:= 0;
      if modo = 5 then
        begin
          longitud := copy(instruccion,3,1);
          if modo = 7 and registro = 1 or registro = 4 then
            longitud := longitud + 1;
        end;
      str(base, binario, instruccion,3, 7,n);
      i := longitud;
      if i > 1 then
        while i<>1 do
      Modo(instruccion,registro,e tension,ModoDireccion);
      n := 0 +n;
      if copy(instruccion,8,1) = 0 then  { (OR  Dn , <Ea> ) = (Dn)
        begin
          mnem := mnem+'B ,'+ModoDireccion;
          op1 := n;
          op2 := ModoDireccion;
        end
      else
        begin                                    { (OR  ea , Dn ) = ea )
          mnem := mnem+ModoDireccion+ ',Dn';
          op1 := ModoDireccion;
          op2 := n;
        end;
    end;

PROCEDURE SBCD;
{
SUBRUTINA QUE termina de desensamblar la instruccion SBCD
VARIABLES QUE UTILIZA:
}
var R ,
    Ry,
              :string;
{
VARIABLES DE ENTRADA: instruccion
VARIABLES DE SALIDA: mnem
}
  begin
    mnem := 'SBCD  ';
    str(base2 10:copy(instruccion,5,3),Ry);
    str(base2 10:copy(instruccion,14,3),Rx);
    if copy(instruccion,13,1)= 1  then
      begin
        mnem := mnem+ '-(A'+Ry+ '),-(A'+Rx+ ')';
        op1 := 'A'+Ry;
        op2 := 'A'+Rx;
      end
    else
      begin
        mnem := mnem+ '(A'+Ry+ '),(A'+Rx ;
        op1 := '(A'+Ry;
```

```
unit nivel4;
{
SINOPSIS: Esta unidad recibe codigos de instruccion cuyos primeros 4 bits
          son iguales, separa el registro y operando si es necesario,
          calcula la longitud de la instruccion en
          mnemonico, operando fuente, coma, espacio, operando destino.
          En todo caso se respeta siempre la sintaxis segun el modo de
          direccionamiento utilizado.
}
interface

UNITS QUE UTILIZA:
{
uses tts,                          { esta unidad se emplea para acceder a la
                                     condicion y el modo de direccionamien-
                                     to, y el numero decimal negativo de un
                                     numero binario negativo}
     Cbase;                        { funciones de cambio de base 2...16 }
{
RUTINAS QUE UTILIZA:
}
PROCEDURE Nivel4(instruccion : string;
                 var mnem,
                     op1,
                     op2       : string;
                 var longitud,
                     modo,
                     registro,
                     size
                               : integer);
procedure SUBnivel4(var mnem, instruccion8,op1,op2 : string;
                    var longitud,modo,registro,size : integer);
procedure SUBnivel4(var mnem, instruccion8,op1,op2 : string;
                    var longitud,modo,registro,size : integer);
procedure SUBnivel4(var mnem, instruccion1,op1,op2 : string;
                    var longitud,modo,registro,size : integer);
implementation

PROCEDURE Nivel4;
{
SINOPSIS:
var
    bit7 : string;
begin
  if copy(instruccion8,4,1) = '1' then
    begin
      SUBnivel4(mnem,instruccion8,op1,op2,longitud,modo,registro,size);
      exit;
    end;
  bit7:=copy(instruccion8,1,1);
  if (bit7='0') or (bit7='00') then
    begin
      SUBnivel4(mnem,instruccion8,op1,op2,longitud,modo,registro,size);
      exit;
    end
  else
    begin
      SUBnivel4(mnem,instruccion8,op1,op2,longitud,modo,registro,size);
      exit;
    end;
end;
```

```
    ...
      e_tension,
      ModoDireccion : string;
  begin
    modo := clase2_1(cop.instruccionB,11,3);
    registro := clase2_1(cop.instruccionB,14,3);
    mnem := 'SUB ';
    n:=cop.instruccion[B,5,2];
    if n= 00 then
      begin
        mnem := mnem+ 'B ';
        size := 0;
      end;
    if n= 01 then
      begin
        mnem := mnem+ 'W ';
        size := 1;
      end;
    if n= 10 then
      begin
        mnem := mnem+ 'L ';
        size := 2;
      end;
    str(clase2_1(cop.instruccionB,5,2),n);
    longitud := 1;
    if modo = 5 then
      begin
        longitud := longitud + 1;
        if (modo = 7) and (registro < 1) or (registro = 4) then
          longitud := longitud + 1;
      end;
      i := longitud;
    if i <> 1 then
      desplazamiento := cop. desplazamiento[7,i-1];
    modoDir(modo,registro,e_tension,ModoDireccion);
    if cop.instruccionB,8,1] = 0 then
      begin
        mnem := mnem+'modoDireccion+ ' + D +n;
        op1 := ModoDireccion;
        op2 := ' D +n;
      end
    else
      begin
        mnem := mnem+ D +n+ ' , '+modoDireccion;
        op1 := ' D +n;
        op2 := ModoDireccion;
      end;
end;

procedure SUBA;
var n,
    e_tension,
    ModoDireccion : string;
  begin
    modo := clase2_1(cop.instruccionB,11,3);
    registro := clase2_1(cop.instruccionB,14,3);
    mnem := 'SUBA ';
    if cop.instruccionE,8,1] = 0 then
      begin
        mnem := mnem+ 'W ';
        size := 1;
      end
    else
      begin
        mnem := mnem+ 'L ';
```

```
        end;
      str(cbase2 [i+1]^.instruction^.s,s);
      if mode = 2 then
        begin
          longitud := longitud + 1;
          if (mode = 7) and (registro = 1) or (registro = 4) then
            longitud := longitud + 1;
        end;
      i := i + longitud;
      if i > 15 then
        desensamblar := desensamblar + '...';
      modificacion(mode,registro,terminar,0);
      mnem := mnem + ' ' + s;
      op2 := s + ',';
    end;

procedure sub1;
var s,
    s1,
    s2,
    modificacion : string;

  begin
    mode := cbase2 [i+1]^.instruction[1].3;
    registro := cbase2 [i+1]^.instruction[1].4,3);
    size := cbase2 [i+1]^.instruction[1].6,2);
    mnem := 'SUB';
    case size of
      0: mnem := mnem + '.B';
      1: mnem := mnem + '.W';
      2: mnem := mnem + '.L';
    end;
    str(cbase2 [i+1]^.instruction[1].5,s1);
    str(cbase2 [i+1]^.instruction[1].14,s2);
    if copy(instruccion,12,1) = '1' then
      begin
        mnem := mnem + ' -(A' + s1 + '),-(A' + s2 + ')';
        op1 := '-(A' + s1 + ')';
        op2 := '-(A' + s2 + ')';
      end
    else
      begin
        mnem := mnem + ' D' + s1 + ',D' + s2;
        op1 := 'D' + s1;
        op2 := 'D' + s2;
      end;
  end;

end.
```

```
unit NivelB;

interface

uses Crt, Dos;

PROCEDURE NivelBInstruccion( var mnem,
                                 op1,
                                 op2      : string;
                             var longitud,
                                 modo,
                                 registro,
                                 size
                                          : integer );

implementation

PROCEDURE NivelB;

var
    registro1,
    Rv,
    cond,
    extension,
    modoDirecto,
    dato
                        : string;
    instruccionB
                        : integer;

begin
  modo := Copy(Hex(instruccionB),1,2);
  registro := Copy(Hex(instruccionB),4,2);
  instruccionB:= Copy(Hex(instruccionB),3,2);
  str(Copy(Hex(cmp,instruccionB,5,7),registro);
  case instruccionB of
    0: begin
         size := 0;
         longitud := le;
         if modo <> 5 then
           begin
             longitud := longitud + le;
             if (modo = 2) and ((registro = 1) or (registro = 4)) then
               longitud := longitud + le;
           end;
         if longitud <> le then
           extension := Copy(instruccionB,1,2*longitud-le);
         ModoDir(modo,registro,extension,op1);
         mnem := 'CMP.B ' + op1 + ', D' + registro1;
         op2 := 'D' + registro1;
         exit;
       end;
    1: begin
         size := 1;
         longitud := le;
         if modo <> 5 then
           begin
             longitud := longitud + le;
             if (modo = 2) and ((registro = 1) or (registro = 4)) then
               longitud := longitud + le;
```

```
        extension := cod.instrucciones[i].longitud-1;
      modo(1,modo,registro.extension,cod);
      mnem := CMP,W +cod+ ', D +registro1;
      cod := 1 +registro1;
      exit;
    end;
  2: begin
      size := 2;
      longitud := 2;
      if modo = 5 then
        begin
          longitud := longitud + 1;
          if (modo = 7) and (registro = 1) or (registro = 4)) then
            longitud := longitud + 1;
        end;
      if longitud = 1 then
        extension := cod.instrucciones[i].longitud-1;
      modo(1,modo,registro,extension,cod);
      mnem := CMP,W +cod+ ', D +registro1;
      cod := 1 +registro1;
      exit;
    end;
  3: begin
      longitud := 1;
      size := 1;
      if modo = 5 then
        begin
          longitud := longitud + 1;
          if (modo = 7) and (registro = 1) or (registro = 4)) then
            longitud := longitud + 1;
        end;
      if longitud = 1 then
        extension := cod.instrucciones[i].longitud-1;
      modo(1,modo,registro,extension,cod);
      mnem := CMP,W +cod+ ', D +registro1;
      cod := 1 +registro1;
      exit;
    end;
  4: begin
      if cod.instrucciones[i] = 0 then
        begin
          size := 0;
          longitud := 1;
          str(cod:1 (cod.instruccione[i],1)),r);
          cod := r +'F';
          cod := 0 +registro1;
          mnem := CMFM,B '# +hex+ '+, (4 +registro1+ )+ ;
          exit;
        end
      else
        begin
          longitud := 1;
          size := 0;
          if modo = 5 then
            begin
              longitud := longitud + 1;
              if (modo = 7) and (registro = 1) then
                longitud := longitud + 1;
            end;
          if longitud = 1 then
            extension := cod.instrucciones[i].longitud-1;
          modo(1,modo,registro,extension,cod);
          cod := 0 +registro1;
          mnem := CMFM,F + 0 +registro1+ ', +cod;
```

```
        end;
    2: begin
         if copy(instruccion8,7,2) = '11' then
           begin
             size := 1;
             longitud := 1b;
             str(base,copy(instruccion8,14,2),ry);
             mod := '00';
             op2 := '0' +registro1;
             mnem := 'MOV' + espacio + reg8[ry] + ',' + reg8[op2];
             exit;
           end
         else
           begin
             longitud := 1b;
             size := 1;
             if mod = '00' then
               begin
                 longitud := longitud + 1b;
                 if (mod = '00') and (reg.etro = '110') then
                   longitud := longitud + 1b;
               end;
             if longitud = 1b then
               extension := copy(instruccion8,17,longitud-1b+1);
             Mod2bin(mod,registro,extension,op2);
             op1 := '0' +registro1;
             mnem := 'MOV' + reg8[op1] + ',' + op2;
             exit;
           end;
       end;
    3: begin
         size := 2;
         longitud := 1b;
         str(base,copy(instruccion8,14,2),ry);
         mnem := 'MOV' + 'AX' + ',' + '[' + registro1 + ']';
         op1 := '0' + ry;
         op2 := '0' + registro1;
         exit;
       end;
    7: begin
         if copy(instruccion8,7,2) = '11' then
           begin
             size := 2;
             longitud := 1b;
             str(base,copy(instruccion8,14,2),ry);
             mnem := 'MOV' + reg16[ry] + ',' + reg16[registro1];
             op1 := '0' + ry;
             op2 := '0' + registro1;
             exit;
           end
         else
           begin
             longitud := 1b;
             size := 2;
             if mod = '00' then
               begin
                 longitud := longitud + 1b;
                 if (mod = '00') and (registro = '110') then
                   longitud := longitud + 1b;
               end;
             if longitud = 1b then
               extension := copy(instruccion8,17,longitud-1b+1);
             Mod2bin(mod,registro,extension,op2);
             op1 := '0' +registro1;
```