UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DESARROLLO DE UN TRADUCTOR DE LENGUAJE ENSAMBLADOR A CODIGO DE MAQUINA PARA UN MICROPROCESADOR DE 16 BITS.

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

ALFREDO MANZANO CUEVAS

DIRECTOR DE TESIS: ING. LUIS G. CORDERO BORBOA

MEXICO, D. F. 1996

DESARROLLO DE UN TRADUCTOR DE LENGUAJE ENSAMBLADOR A CODIGO DE MAQUINA PARA UN MICROPROCESADOR DE 16 BITS

DIRECTOR DE TESIS

ING. LUIS G. CORDERO BORBOA

Quiero expresar mi agradecimiento,
a todas aquellas personas que de
alguna manera me ayudaron, y
supieron esperar hasta la culminación
de este trabajo.

A mis Padres:

A quienes debo todo lo que soy,
gracias a su inmenso apoyo,
amor y confianza.

A Lendis, mi esposa,
y a Alfredo y Andrea, mis hijos:

Porque en ellos he tenido un gran
apoyo y motivación para seguir
adelante y alcanzar mis metas.

DESARROLLO DE UN TRADUCTOR DE LENGUAJE ENSAMBLADOR A CODIGO DE MAQUINA PARA UN MICROPROCESADOR DE 16 BITS

INTRODUCCION

El presente trabajo tiene como objetivo facilitar el manejo de la programación del microprocesador MC 68000, através de un paquete que traduce un programa escrito en lenguaje ensamblador a su correspondiente programa escrito en lenguaje de máquina para posteriormente introducirlo dentro del microcomputador basado en el microprocesador MC 68000.

Actualmente se cuenta con microcomputadores de este tipo en la facultad de ingeniería, por lo que se plantea este proyecto como una herramienta más para agilizar la elaboración y el procesamiento de programas en lenguaje ensamblador.

El capítulo 1 resume la estructura interna desde el punto de vista de programación (software), tales como componentes de un sistema de programación, macros y definiciones importantes como que es un ensamblador, su elaboración y diseño. El cápitulo 2 incluye conceptos básicos del microprocesador MC 68000 tales como su arquitectura, contenido interno, ejemplo de un sistema mínimo y su funcionamiento. En este capítulo se incluyen también conceptos de programación, principalmente modos de direccionamiento.

El capítulo 3 contiene las características particulares a emplear en el desarrollo del traductor y análisis del sistema.

El capítulo 4 contiene un desarrollo sistemático del diseño del sistema, basandonos en los conceptos manejados en los cápitulos anteriores, carta de estructura y diagrama de flujo de datos.

El capítulo 5 contiene el desarrollo del proyecto basados en el cápitulo anterior y el seudocódigo del diagrama de flujo de datos y la programación estructurada.

El capítulo 6 contiene ejemplos de traduccionas ejecutadas con el sistema, manejo del mismo, así como las acotaciones que se deben tomar en cuenta para el empleo del traductor.

CAPITULO I

DESCRIPCION DE ENSAMBLADORES

I DESCRIPCION DE ENSAMBLADORES

En un principio, el programador de computadoras tenía a su disposición una máquina que interpretaba, a través de hardware, cierto tipo de instrucciones fundamentales. La programación de la computadora se hacía escribiendo una serie de unos y ceros (lenguaje de máquina) colocándolos en la memoria de la máquina, posteriormente presionaba un boton de comienzo y de esta forma la máquina empezaba a ejecutarlos como instrucciones.

Los programadores encontraron difícil escribir y leer programas en lenguaje de máquina. En el intento por encontrar un lenguaje más conveniante comenzaron a usar nemónicos (símbolos) para cada instrucción de máquina, los cuales serían convertidos posteriormente a lenguaje de máquina. Tal lenguaje de nemónicos es llamado lenguaje ensamblador. Los programas conocidos como ensambladoras fueron escritos para automatizar la traslación de lenguaje ensamblador a lenguaje de máquina. La entrada a un ensamblador se conoce como programa fuente, la salida en lenguaje de máquina es el programa objeto.

Hay cuatro ventajas para usar el lenguaje ensamblador en lugar del lenguaje de máquina:

1. Es nemónico, esto es, escribimos ST en lugar de la configuración de bit 01010000.
2. Las direcciones son simbolicas, no absolutas.
3. La lectura del programa es más fácil.
4. El mantenimiento del programa es más fácil.

Una desventaja del lenguaje ensamblador es que requiere del uso de un ensamblador para traducir el programa fuente en programa objeto.

Con respecto al uso del computador la siguiente relación nos indica algunas caracteríaticas de los diferentes niveles de lenguaje.

BASIC, FORTRAN, PASCAL ;

Ventajas: Requiere de relativamente poco tiempo de programación, usa expresiones en inglés, cada instrucción individual es traducida en muchas instrucciones de máquina, es exportable, puede facilmente adaptarse a diferentes computadores.

Desventajas: Ineficiente uso de memoria comparado con lenguajes de más bajo nivel, dificultad de manejar bits, especialmente para operaciones de entrada salida, no puede ser optimizado respecto a su velocidad de ejecución.

MACROENSAMBLADORES :

Ventajas: Combina la velocidad de los ensambladores con cierta eficiencia de programación de los lenguajes de alto nivel.

Desventajas: Es más difícil de aprender y de usar que los lenguajes de alto nivel.

ENSAMBLADORES:

Ventajas: Corren mucho más rápido que los lenguajes de alto nivel, usa menos memoria que ellos y permite el control de cada bit y dirección.

Desventajas: Es más difícil de aprender y de usar que los lenguajes de alto nivel, requiere de más tiempo de programación, no es exportable, cada microprocesador usa diferente ensamblador.

LENGUAJE DE MAQUINA:

Ventajas: Trabaja con cada bit dentro y fuera del microprocesador, es el de mayor velocidad al ejecutarse.

Desventajas: Muy difícil de trabajar, requiere de mucho tiempo de programación, no es exportable, diferentes lenguajes para cada sistema y microprocesador.

El lenguaje ensamblador es un lenguaje de símbolos. Para usar un lenguaje ensamblador es necesario aprender los símbolos o instrucciones que reconoce. Estas instrucciones incluyen instrucciones para el procesador, instrucciones para el ensamblador y macroinstrucciones; todo esto difiere de ensamblador a ensamblador. Antes de pasar a describir cada una de ellas, platiquemos brevemente sobre las reglas de sintaxis y formateo de las instrucciones en forma general.

La SINTAXIS es un conjunto de reglas para colocar correctamente símbolos conjuntamente de manera que un ensamblador los pueda reconocer, trabajar y traducir apropiadamente. El punto que se debe recordar es que la sintaxis varía de ensamblador a ensamblador. Generalmente las instrucciones básicas tienen la misma representación, las instrucciones más avanzadas pueden no ser las mismas. Algunas reglas generales de sintaxis son :

1. Uso de símbolos para indicar en que sistema número se trabaje :

B ó % = Binario.
Q ó @ = Octal.
D = Decimal.
H ó $ = Hexadecimal.

2. Escribir el código de operación de la instrucción al principio, posteriormente escribir los operandos.

3. Separa el operando fuente del operando destino por una coma.

4. Uso de una letra extensiva para indicar el tamaño del operando.

B = Byte.
W = Word.
L = Long-word.

5. Uso de paréntesis en los operandos para indicar modo indirecto. Un par de perentesis alrededor de un operando muestra que el valor del operando será usado como una dirección indirecta.

6. Uso de signos para indicar direccionamiento post incrementado o predecrementado.

\+ Después del paréntesis para indicar postincremento.

\- Antes del paréntesis para indicar predecremento.

7. Uso de dos puntos después de una etiqueta.

8. Uso de un espacio después del código de operación.

Respecto al FORMATO de las instrucciones, los ensambladores organizan los programas en lenguaje ensamblador en divisiones llamadas campos. No son en realidad parte del código; Son estructuras que el ensamblador usa para simplificar la comunicación con el programador. Cuando se está programando se tiene que introducir la información en los campos correspondientes o el ensamblador no lo entendera. Existen tres campos principales: Etiqueta, Instrucción y Comentario.

El siguiente es un ejemplo en un programa ensamblador cualquiera :

```
Etiqueta        Cod. Op.        Operandos       Comentarios
DATA            EQU             $0010000        Almacena DATA
PROGRAM         EQU             $0012000        Almacena PROGRAM
                ORG             DATA
TEMP            DS.W            1               Valor a girar
COUNT           DS.W            1               Posiciones a
                                                girar.
                ORG             PROGRAM
SHIFTER         MOVE.W          TEMP,D0         D0 tiene el valor
                                                a girar.
                MOVE.W          COUNT,D1        Contador giros
                ROR.B           D1,D0           Giro
                MOVE.W          D0,VALUE
                RTS
                END             SHIFTER
```

Esta es la organización del código fuente que el ensamblador traducira en código objeto. El campo de la etiquete contiene etiquetas simbolicas o direcciones de varias instrucciones. El ensamblador puede utilizar estas referencias simbolicas para especificar saltos o regresos. El siguiente campo es realmente el más importante. El campo de las instrucciones se divide en dos partes, el primero es el código de operación, el siguiente son los operandos. Finalmente se tiene el campo de los comentarios. Los comentarios son opcionales, y no cambian la operación del programa, pero deben ser usados.

Hay varios tipos de ensambladores. Un poderoso ensamblador conocido como MACROENSAMBLADOR permite definir macroinstrucciones y usarlas. Estas macros son instrucciones para el procesador o para el macroensamblador agrupadas en bloques a las cuales se les da un nombre. Una vez que se ha definido la macroinstrucción, lo único que se tiene que hacer es llamarla para que el macroensamblador sustituya el bloque de instrucciones por el nombre de la macroinstrucción, se hablará de esta macroinstrucción más adelante.

Otro importante tipo de ensamblador es el que se conoce como CROSS ASSEMBLER. Dado que la creación, edición y ensamble de un programa fuente en código objeto es abstracto (no requiere del microprocesador real), los programadores frecuentemente usan ensambladores que estan escritos y corren en otras computadoras que no tienen el microprocesador para el cual el programa esta dirigido, a tales ensambladores se les conoce como CROSS ASSEMBLERS.

I.1 Tipos de instrucciones

El siguiente subtema contempla los dos tipos de instrucciones contemplados para el sistema que son: instrucciones para el procesador e instrucciones para el ensamblador.

I.1.1 Instrucciones para el procesador.

Las operaciones fundamentales que un microprocesador puede ejecutar se conocen como instrucciones para el procesador, son las palabras en el lenguaje del microprocesador. El conjunto de instrucciones del microprocesador MC68000 está dado en el cápitulo II, todo ese conjunto forman las instrucciones para el procesador MC68000.

Las instrucciones fundamentales de cualquier microprocesador pueden ser divididas en grupos funcionales. Estos grupos son muy similares de circuito integrado a circuito integrado.

Instrucciones de movimiento de datos.

Muchas personas piensan en el computador como máquinas matemáticas, cuando en realidad su mayor función es la de almacenamiento y manipulación de datos. Cualquier programa tiene que mover BITS y BYTES entre la entrada y la salida, entre la memoria y el CPU.

Las instrucciones de movimientos de datos permiten mover información de registro a registro, registro a memoria o de memoria a memoria, la información que se mueve puede ser dirección o dato. El movimiento de los datos puede ser BYTE, palabra o palabra completa.

Instrucciones aritméticas.

En terminos generales la mayoría de los microprocesadores pueden efectuar operaciones de sume y resta, los microprocesadores de 16 bits, tienen tambien multiplicación y división. Dentro de las operaciones aritméticas podemos mencionar, además, comparación, negación e inicialización de registros.

Instrucciones lógicas.

La mayoría de los computadores hacen uso extendido de las operaciones lógicas. Las operaciones generalmente manejadas son AND, OR, EOR, NOT.

Instrucciones de corrimiento y giro.

Son instrucciones que permiten cambiar o mover el contenido de un operando. Las instrucciones de giro y corrimiento mueven BITS en forma adyacente de una posición a la siguiente en un registro o localidad de memoria.

Instrucciones de salto condicional e incondicional.

Cualquiera de las instrucciones de salto condicional e incondicional obliga al programa a continuar la ejecución del programa en una nueva posición. Las instrucciones de salto incondicional ponen un nuevo valor en el CONTADOR DE PROGRAMA, de esta forma el procesamiento continúa en una nueva dirección.

I.1.2 Instrucciones para el ensamblador

Las DIRECTIVAS son instrucciones para el ensamblador, no son parte del conjunto de instrucciones para el procesador. Las directivas son usadas para especificar el comienzo de un programa, inicializar variables, reservar espacio de memoria para ciertas estructuras de datos o definir MACROS (son una secuencia comprimida de instrucciones usadas para salvar tiempo en la programación), la siguiente es una lista de las principales directivas que usan la mayoría de los ensambladores:

DATA	Introduce datos en una memoria.
EQUATE	Relaciona nombres simbólicos con direcciones o datos.
END	Marca el fin del programa.
ENTRY	Muestra que nombre está disponible para usarse.
EXTERNAL	Indica que el nombre está definido en otro programa.
LIST	Imprime programa fuente.
NAME	Imprime el nombre del programa en la cabezera de cada página.
ORIGIN	Especifica la localidad de memoria donde el programa o los datos quedara almacenado.
PAGE	Salta el listado a la siguiente página.
RESERVE	Asigna memoria.

Existen, también, directivas condicionales, es decir, directivas que seleccionan, mediante el valor de algún parámetro que parte del programa fuente será ensamblado. Generalmente estas directivas se encuentran asociadas a MACROS a las cuales nos referiremos más adelante.

I.2 Macroinstrucciones

El programador de lenguaje ensamblador suele encontrar necesario repetir bloques de instrucciones en un programa. El bloque puede consistir de instrucciones para salvar o intercambiar conjuntos de registros, por ejemplo, o una serie de operaciones aritméticas. Realmente el bloque puede agrupar conjunto de instrucciones para el procesador, directivas o una combinación de ambas. Las macroinstrucciones (generalmente llamadas MACROS) permiten referenciar a través de una sola instrucción un conjunto de instrucciones. Por cada referencia de la macro el ensamblador sustituirá todo el bloque de instrucciones.

Mediante la definición apropiada de macros, un programador en lenguaje ensamblador puede optimizar su uso. Puede hacer semejante su codificación a los lenguajes de alto nivel sin perder las ventajas básicas de la programación en lenguaje ensamblador. Integrar instrucciones en macros simplifica el desarrollo y mantenimiento de los programas, así como facilita la estandarización.

Para generar el programa objeto de un programa en lenguaje ensamblador es necesario usar un ensamblador, un programa en lenguaje ensamblador que use macros deberá ser traducido por un MACROENSAMBLADOR. Generalmente se considera a las macroinstrucciones como una extensión del lenguaje ensamblador básico, el macroensamblador es visto como extensión del ensamblador.

Características de las macroinstrucciones.

En su forma simple, una MACRO es una abreviatura para una secuencia de operaciones. Consideremos el siguiente programa (no se trata de algún programa escrito en cierto tipo de lenguaje ensamblador, véase solo para ilustrar la forma de las MACROS):

Ejemplo 1.

```
DATA        DS          2;
              .
              .
              .
ADD             1, DATA;
ADD             2, DATA;
ADD             3, DATA;
              .
              .
              .
ADD             1, DATA;
ADD             2, DATA;
ADD             3, DATA;

              .
              .
              .
```

En el ejemplo 1, la secuencia :

```
ADD             1, DATA;
ADD             2, DATA;
ADD             3, DATA;
```

ocurre dos veces. Con la facilidad de las macroinstrucciones es posible asignar un nombre a esta secuencia y usar este nombre en lugar de ella.

La asignación del nombre a la secuencia se hace a través de la definición de macroinstrucciones, y que en términos generales se forma de la siguiente manera :

```
MACRO   :  Palabra reservada.
[   ]   :  Nombre de la macro.
 ---    :  Secuencia de instrucciones.
 ---
 ---
MEND    :  Fin de la definición de macro.
```

La instrucción para el ensamblador MACRO es la primer línea de la definición e identifica a la siguiente línea como el nombre de la macroinstrucción. Siguiendo al nombre esta la secuencia de instrucciones que serán agrupadas en la macro. La definición es terminada por una línea con la instrucción para el ensamblador MEND.

Una vez que la macro ha sido definida, el uso del nombre de la macro como el nemónico de una operación es equivalente al uso de su correspondiente secuencia de instrucciones. Nuestro ejemplo 1 queda escrito de la siguiente forma :

```
DATA    DB    2;
        .
        .
        .
MACRO;
SUMA;
ADD     1, DATA;
ADD     2, DATA;
ADD     3, DATA;
MEND;
        .
        .
        .
SUMA;



        .
        .
        .
SUMA;
        .
        .
        .
```

En este caso el macroensamblador reemplaza cada llamada a la macro por las líneas :

```
ADD     1, DATA;
ADD     2, DATA;
ADD     3, DATA;
```

Este proceso de reemplazo es llamado expansión de la macro. La ocurrencia en el programa fuente del nombre de la macro se conoce como llamada a la macro.

La estructura de la macro presentada hasta el momento es capaz de insertar bloques de instrucciones en el lugar de las llamadas de las macros. Todas las llamadas a una macro dada son reemplazadas por un bloque idéntico. Este tipo de estructura de las macros no es flexible : no hay forma de modificar el código que reemplaza a la llamada de la macro.

Como resultado de esta restricción fué implementada la posibilidad de pasar PARAMETROS o ARGUMENTOS en la llamada a una macro. Correspondientes argumentos mudos de la macro aparecen en su definición.

Ejemplo 2.

```
DATA1     DS   2;
DATA2     DS   2;
      .
      .
      .
ADD     1, DATA1;
ADD     2, DATA1;
ADD     3, DATA1;
      .
      .
      .
ADD     1, DATA2;
ADD     2, DATA2;
ADD     3, DATA2;
      .
      .
      .
```

En este caso la secuencia de instrucciones son muy similares pero no idénticas. La primer secuencia ejecuta opereciones usando DATA1 como operando; La segunda, use DATA2. Realmente la secuencia difiere en un argumento variable. Una macro puede ser definida para permitir argumentos de la siguiente forma :

```
MACRO;
SUMA    @ ARG;
ADD     1, @ ARG;
ADD     2, @ ARG;
ADD     3, @ ARG;
MEND;
      .
      .
      .
SUMA    DATO1;
      .
      .
      .
 SUMA    DATO2;
      .
      .
      .
```

Es posible proporcionar más de un argumento en una llamada a macro. Cada argumento debe corresponder con los argumentos definidos en el nombre de la macro. Cuando la macro es ensamblada, los argumentos dados son sustituidos por sus respectivos argumentos mudos en la definición de la macro.

Ejemplo 3.

```
            DATA1     DS     2;
            DATA2     DS     2;
            DATA3     DS     2;
                  .
                  .
                  .
CICLO1:     ADD         1, DATA1;
            ADD         2, DATA2;
            ADD         3, DATA3;
                  .
                  .
                  .
CICLO2:     ADD         1, DATA1;
            ADD         2, DATA2;
            ADD         3, DATA3;
                  .
                  .
                  .
```

En este caso, los operandos en el bloque de instrucciones son diferentes, así como las etiquetas. Con la facilidad de manejo de argumentos en las llamadas a macros, el ejemplo 3 sería escrito como :

```
                  .
                  .
                  .
            MACRO;
            SUMA      @ARG1, @ARG2, @ARG3;
            ADD       1, @ARG1;
            ADD       2, @ARG2;
            ADD       3, @ARG3;
            MEND;
                  .
                  .
                  .
CICLO1:     SUMA       DATA1, DATA2, DATA3;
                  .
                  .
```

```
CICLO2:    SUMA      DATA3, DATA2, DATA1;
                       .
                       .
                       .
```

Cuando se habló de instrucciones para el ensamblador se mencionarón las directivas condicionales, su uso en la definición de macroinstrucciones es el mismo que en el programa en lenguaje ensamblador sin el manejo de macroinstrucciones, es decir, proporcionar un mecanismo para decidir que parte del bloque de instrucciones que abrevia la macro formará parte del programa ensamblado.

Consideremos el siguiente ejemplo:

Ejemplo 4.

```
            DATA1     DS   2;
            DATA2     DS   2;
            DATA3     DS   2;
                       .
                       .
                       .
CICLO1:     ADD       1, DATA1;
            ADD       2, DATA2;
            ADD       3, DATA3;
                       .
                       .
                       .
CICLO2:     ADD       1, DATA3;
            ADD       2, DATA2;


                       .
                       .
                       .
 CICLO3:    ADD       1, DATA1;
                       .
                       .
                       .
```

El programa puede ser escrito usando condicionales, de la siguiente forma :

```
            MACRO;
            VARIA       @BANDERA, @ARG1, @ARG2, @ARG3;
            ADD         1, @ARG1;
            .IF         (@BANDERA EQ 1)  .FINI
            ADD         2,@ARG2;
            .IF         (@BANDERA EQ 2)  .FINI
            ADD         3,@ARG3;
.FINI       MEND


                    .
                    .
                    .
CICLO1:     VARIA       3,DATA1,DATA2,DATA3;
                    .
                    .
                    .
CICLO2:     VARIA       2,DATA1,DATA2;
                    .
                    .
                    .
CICLO3:     VARIA       1,DATA1;
                    .
                    .
                    .
```

Las instrucciones que comienzan con un punto son directivas, esto es, no aparecen en la salida del macroensamblador. La instrucción al macroensamblador .IF (@BANDERA EQ 1) .FINI lo dirige para saltar a la etiqueta .FINI si el valor correspondiente a BANDERA es 1, de otra forma el macroensamblador continúa con la siguiente instrucción al .IF.

I.3 Proceso de ensamble

Cualquier ensamblador debe ejecutar los siguientes pasos :

1. Generar instrucciones :

   1.1 Evaluar el nemónico en el campo de operación para producir su código de máquina.

   1.2 Evaluar los operandos : definir el valor de cada símbolo, procesar literales y asignar direcciones.

2. Ejecutar directivas.

Con esta información, las siguientes son las etapas en el proceso de ensamble.

1. Mantener un contador de localidades.
2. Almacenar variables y sus direcciones.
3. Ejecutar directivas.
4. Almacenar literales y sus valores.
5. Determinar la longitud de las instrucciones para el procesador.
6. Evaluar las instrucciones para el procesador.
7. Generar el código de la instrucción.

CAPITULO II

PRESENTACION DEL MC 68000

II. Presentación del MC 68000

Tomando en cuenta que existen varias versiones del MC 68000 a continuación se comenta un poco de esta familia.

La primera presentación de motorola en el ámbito de los 16 bits fué el MC 68000, contiene 17 registros de datos y direcciones de 32 bits, 14 tipos de direccionamiento, capacidad para manejar 16 megabytes de memoria, 56 tipos de instrucción, utiliza segmentación y admite cinco tipos de datos. El bus de datos es de 16 bits y el bus de direcciones de 24 bits.

El microprocesador 68008, ofrece las caracteristicas básicas del 68000, sumando efectividad de costo en su encapsulado, utiliza un bus de datos 8 bits que permite acoplarlo y acceder a memorias económicas organizadas en bytes y a periféricos más sencillos. El microprocesador 68010 añade el soporte de hardware para memoria virtual y máquinas virtuales tablas de vectores múltiples e instrucciones para bucles de altas prestaciones, utiliza un bus de datos de 16 bits y un bus de direcciones de 24 bits. El microprocesador 68012 varía del 68010 en que su bus de direcciones es de 30 bits. El microprocesedor 68020 contiene buses completos de 32 bits, capaz de direccionar 4 gigabytes, interfaz a coprocesador, 7 tipos de datos, 18 modos de direccionamiento y un acelerador (cache) de altas prestaciones para instruciones en una pastilla.

II.1 Arquitectura del microprocesador

La arquitectura del microprocesador se entiende como su estructura interna, está incluye elementos como caracterización de las líneas de dirección, datos y control, así como su repercución en los formatos de instrucción y los modos de direccionamiento.

Alineamiento

En los procesadores 68000, 68008, 68010, 68012, las palabras y palabras largas (incluyendo palabras instrucción) deben comenzar en una dirección par. Este es un requerimiento impuesto por la naturaleza del bus. Si se intenta acceder a una palabra o palabra larga en una dirección impar, el procesador inicia un procesamiento excepcional.

El 68020 solventa esta limitación comprobando los accesos de palabras impares; si así ocurre, hace dos o más accesos parciales a la palabra o palabra larga. Naturalmente, esto añade una sobrecarga a la ejecución, por lo que se debera alinear, cuando sea posible, las palabras en fronteras pares (bit 0 igual a cero) y las palabras largas en fronteras de palabras largas (bit 0 y 1 iguales a cero). Para mejor funcionamiento, la pila debería ubicarse en una frontera de palabra larga.

Unidad aritmética y lógica

Datos de memoria o registros son ruteados através de la ALU, donde operaciones matemáticas y lógicas son ejecutadas. para después, los datos procesados sean enviados a su destino final.

El MC68000 tiene tres unidades aritméticas lógicas: una para procesar datos y dos para procesar direcciones. La ALU para datos tiene 16 bits para cálculos de datos y evaluación en un solo paso a los 16 bits de datos. Operaciones de 32 bits de datos son hechas en dos pasos, primero la parte baja de la palabra y después la parte alta de la palabra.

Otras dos ALU's son usadas conjuntamente para calcular direcciones. La dirección efectiva (EA) es el resultado final generado desde la instrucción de datos y modo de direccionamiento, para esto es necesario usar dos ALU's ya que las direcciones estan formadas de 32 bits.

Decodificador

El decodificador es la parte del microprocesador que interpreta las instruciones, este traduce las instrucciones en patrones de 1's y 0's para formar el lenguaje de máquina y decir al resto del microprocesador que hacer.

Los más avanzados microprocesadores como el 68000 están frecuentemente en microcódigo, tienen en efecto, un muy pequeño microprocesador que corre rutinas del microprocesador mayor.

Existe una memoria ROM que es programada con instrucciones "tiny" que dicen al procesador que hacer. El microprocesador "tiny" es llamado como microsecuenciador. Estas instruciones son más simples que el lenguaje de máquina. Ellas envuelven acciones elementales (llamadas micropalabras) tales como enviar señales a ciertas compuertas o retener un bit particular del registro de estado.

Micropalabras (microwords) están construídas dentro de microrutinas que vienen a ser instrucciones en lenguaje ensamblador.

Existen dos tipos de microprogramación: horizontal y vertical. Horizontal es más directa; un solo bit de una micropalabra puede habilitar un registro. Micropalabras horizontales son más largas y requieren microbuses más anchos y facilidades de almacenamiento.

Micropalabras verticales codifican información para 16 registros de 4 bits. Este formato es mucho más lento, debido a que las micropalabras tienen que ser decodificadas por si mismas lo cual requiere más áreas de circuito integrado.

Ambas formas de microcódigo son usadas en el procesador 68000, esto quiere decir un doble uso para lo que el procesador tiene un nanocódigo.

La información en microcódigo es puesta en microsubrutinas en nanocódigo, la cual actualmente hace ruteo, selección y direccionamiento.

Líneas de direcciones

El MC 68000 tiene un bus de direcciones de 23 bits (A1-A23) y es capaz de direccionar una memoria de 8 mega-words (16 MBytes), este bus de direcciones es de tres estados, unidireccional y asincrono.

Líneas de datos

Es de 16 bits (D0-D15) permite la transferencia de datos con longitud de byte, palabra ó palabra larga (2 accesos). En el ciclo de reconocimiento de interrupción en D0-D7 está el vector de transferencia. El bus es de tres estados, bidireccional, asincrono y no es multiplexado.

II.2 Señalización

Señales del MC 68000.

Control asincrono del bus.
Controla la comunicación asincrona con memorias y dispositivos periféricos.

Dirección válida en el bus, AS (por sus siglas en inglés).

Sentido de transferencia en el bus, R/W (por sus siglas en inglés).

Dato superior habilitado, UDS (por sus siglas en inglés).

Dato inferior habilitado, LSD (por sus siglas en inglés).

Reconocimiento de transferencia de datos, DTACK (por sus siglas en inglés).

Señales de arbitraje de bus.

Petición de bus, BR (por sus siglas en inglés).

Cesión de bus, BG (por sus siglas en inglés).

Reconocimiento de cesión de bus, BG ACK (por sus siglas en inglés.)

Señales de interrupción.

IPL0,IPL1,IPL2 : Tienen definido el nivel de interrupción o prioridad para indicar al procesador que dispositivo interrumpe.

Señales de control del sistema.

Señal de entrada de error en el bus, BERR (por sus siglas en inglés).
Señal bidireccional de reinicio, RESET (por su significado en inglés).
Señal bidireccional de parada, HALT (por su significado en inglés).

Señales de la familia 6800.

Señal disponible, E (por su sigla en inglés).

Señal de periférico válida, VPA (por sus siglas en inglés).

Señal de dirección de memoria válida, VMA (por sus siglas en inglés).

Códigos de función.

FC0,FC1 Y FC2 (FUNCTION CODE): Indican el estado normal o supervisor y el ciclo que se esta ejecutando en el procesador.

Modos de ejecución.

El MC 68000 puede operar en uno de dos modos, el modo "usuario" o el modo "supervisor". Un bit del registro de estado determina el modo de funcionamiento del procesador.

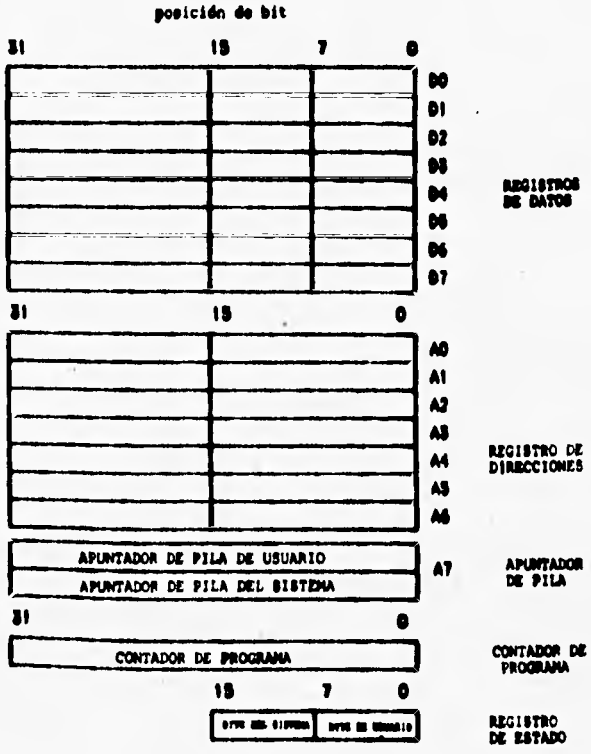
El modo usuario es para cuando el procesador ejecuta operaciones a nivel de aplicaciones, mientras que el modo supervisor está proyectado para programas a nivel de sistema operativo.

El modo supervisor tiene su propio puntero de pila, así como instrucciones adicionales, privilegiadas, que un programa ejecutándose en modo usuario no puede utilizar.

En base a los modos de ejecución definiremos los registros y su operación.

II.3 Registros

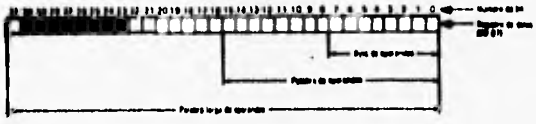
Los registros son localidades de memoria dentro del CPU, debido a que son direccionados fácilmente, leídos o escritos. Los registros pueden ser de propósito general o de propósito especial.



Registros en modo usuario

En modo usuario el MC 68000 tiene 8 registros de datos de 32 bits, 7 registros de dirección de 32 bits. Un apuntador(puntero) de pila (stacker pointer) de 32 bits, un contador de programa de 32 bits y un registro de código de condición de 8 bits.

Los registros de datos pueden usarse como datos de un solo bit, datos de 8 bits (byte), datos de 16 bits (palabra) y datos de 32 bits (palabra larga) como se muestra en la siguiente ilustración.



Cuando una instrucción utiliza un registro de datos como operando fuente o destino, sólo la porción correspondiente del registro es alterada los bits de mayor orden del registro permanecen sin cambio.

Además de su función como fuente o destino de operandos en una operación, los registros de datos pueden funcionar como registros índices o contadores en bucles.

Registros de direcciones

El MC 68000 tiene 7 registros de direcciones de uso general, dependiendo del modo de direccionamiento estos registros pueden mantener direcciones de punteros a operandos, direcciones base e índices.

Los registros de dirección pueden mantener datos de 16 bits o de 32 bits. No hay operaciones directas con bytes en los registros de dirección.

Cuando se utiliza directamente como operando fuente, los valores de 16 bits son ampliados en signo antes de usarlos (la palabra de mayor orden no afecta ni es afectada por la operación). Cuando se usan directamente como destino de operandos, las fuentes de 16 bits están ampliadas en el signo y rellenan completamente los 32 bits del registro de dirección de destino.

El procesador utiliza un octavo registro de direcciones A7, como puntero de pila, el procesador tiene dos punteros de pila distintos, dependiendo del modo de procesamiento en curso. Estos registros (A7' y A7') son comunmente más conocidos como puntero de pila del usuario (USP) y puntero de pila del sistema (SSP).

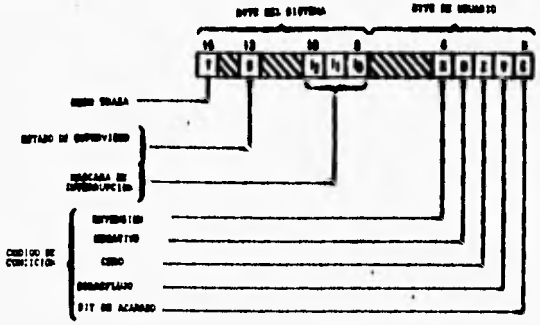
El procesador rellena una pila de posiciones altas a bajas de memoria predecrementando o post-incrementando los modos de direccionamiento indirectos para apilar (insertar) y desapilar (extraer), respectivamente.

Contador de programa

El contador de programa es un registro de uso especial de 32 bits. Apunta a la primera palabra de la siguiente instrucción a ser ejecutada (entre instrucciones) o a la siguiente palabra de una instrucción (durante la captación de instrucción). Como las direcciones deben estar alineadas en direcciones frontera pares(palabras), el contador de programa debe contener siempre un valor par.

Registro de estado

En modo usuario, el procesador tiene acceso al byte de menor orden del registro de estado. Este byte (el registro de código de condición o CCR) mantiene bits indicadores cuyos valores dependen del resultado de ciertas operaciones, tales como suma, resta, desplazamiento y otras.



Registros para el modo sistema

Adicionalmente a los registros de datos direcciones y de código de condición disponibles en el modo usuario, los elementos de la familia MC 68000 tienen varios registro adicionales utilizables en el modo supervisor. El tipo y número de registros supervisor varía entre miembros individuales de la familia MC 68000; por lo que haremos un enfoque hacia el MC 68000L4 debido a que el kit educativo esta basado en este procesador. Contiene dos registros adicionales, el puntero de pila supervisor (A7'o SSP) y el byte superior del registro de estado. El SSP tiene una longitud de 32 bits y apila descendentemente en memoria; las referencias a apilar deben estar alineadas a palabra (direcciones pares).

El byte del sistema del registro de estado junto con el registro de código de condición forman el registro de estado de 16 bits. El byte del sistema contiene dos bits indicadores y tres bits de máscara de interrupción.

El bit superior (S) especifica el modo de ejecución del procesador. Si es uno, el procesador está en modo supervisor; si es cero, el procesador está en modo usuario.

El bit de modo traza (T), cuando es uno especifica que el procesador se encuentra en modo traza, este modo conduce a un modo paso a paso de ejecución. Esto permite a un programa depurador monitorizar los resultados de un programa de aplicación ejecutándolo instrucción a instrución.

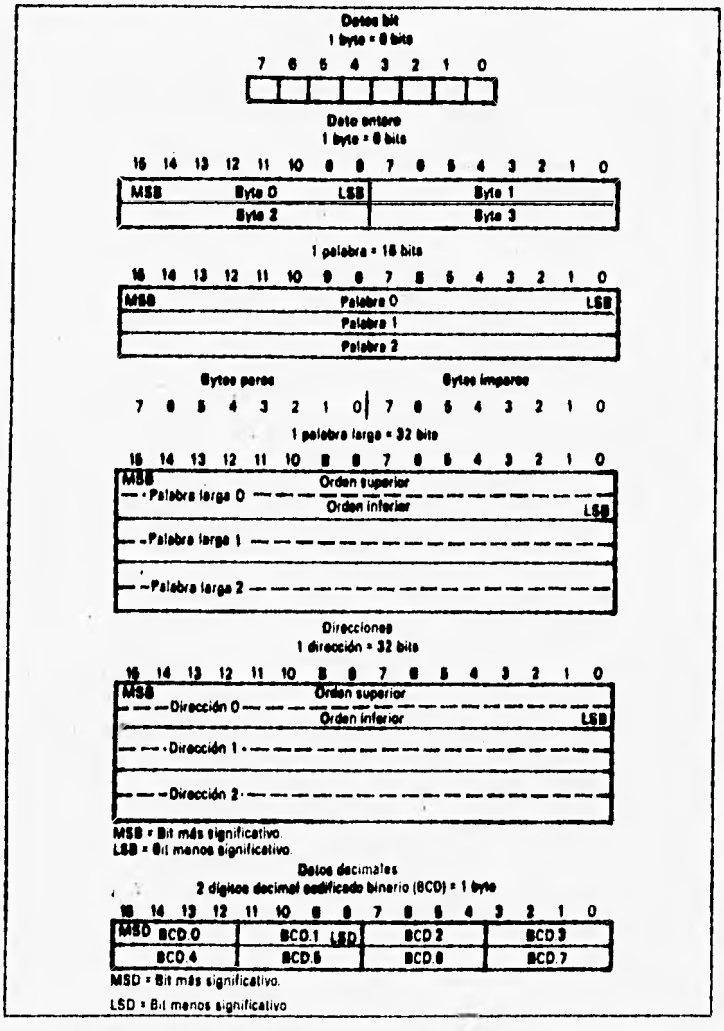
Los procesadores 68000 pueden operar en cualquiera de ocho niveles o prioridades de interrupción. Los dispositivos externos pueden intentar interrumpir al procesador con señales de validación según cualquier combinación en las líneas de petición de interrupción IPL0-IPL2. El valor binario de las líneas de interrupción representa la importancia del dispositivo que interrupe; un dispositivo de prioridad alta (tal como el reloj) tiene un nivel de interrupción de valor auperior al de un dispositivo de prioridad baja, tal como un controlador de terminal. La prioridad mayor es de 7 (en binario 111) y el nivel inferior es cero (en binario 000).

La máscara de interrupción en el registro de estado define el nivel de operación en curso del procesador. El código del modo normal del usuario, normalmente cero. Los controladores de dispositivos operan a niveles superiores. Cuando un dispositivo interrupe al procesador, la lógica interna compara el valor de la máscara de interrupción con el valor del dispositivo contenido en IPL0-IPL2.

La máscara de interrupción, en conjunción con las líneas de interrupción suministran un modelo utilísimo de jerarquizar las prioridades de ejecución.

II.4 Organización de la memoria

La Familia del MC68000 permite acceder a la memoria en grupos de bits constituyendo un byte, palabra o palabra larga. Una dirección especifica la posición del byte o, para datos de varios bytes, la posición del byte más significativo. El byte menos significativo de una palabra está en esa dirección más 1. El byte menos significativo de una palabra larga está en esa dirección más 3. La siguiente figura muestra como están alineados en memoria los bytes, palabras y palabras largas.



Organización de datos en memoria

Memoria virtual

En muchos sistemas basados en MC68000 como procesador central, solo una parte de los 16 megabyte (2 gigabytes para el MC68020) de espacio de direccionamiento contendrán memoria físicamente, sin embargo, por medio del uso de técnicas de memoria virtual, el sistema puede aparentar al usuario que dispone de 16 megsbytes de memoria físicamente. Estas técnicas han sido usadas por varios años en sistemas principales de gran capacidad (large main frame systems) y más recientemente en minicomputadores y ahora con el MC68000 pueden ser tolalmente soportados por sistemas basados en microprocesadores.

II.5 Configuración de un sistema mínimo

Para fines prácticos y con el enfoque de el proyecto que este trabajo persigue, se tomará referencia del computador educacional. El cual es un sistema completo de microcomputadora con procesador de 4 Mhz (MC68000L4).

Se muestra a continuación el diagrama a bloques del sistema basado en MC68000 para el computador educativo.

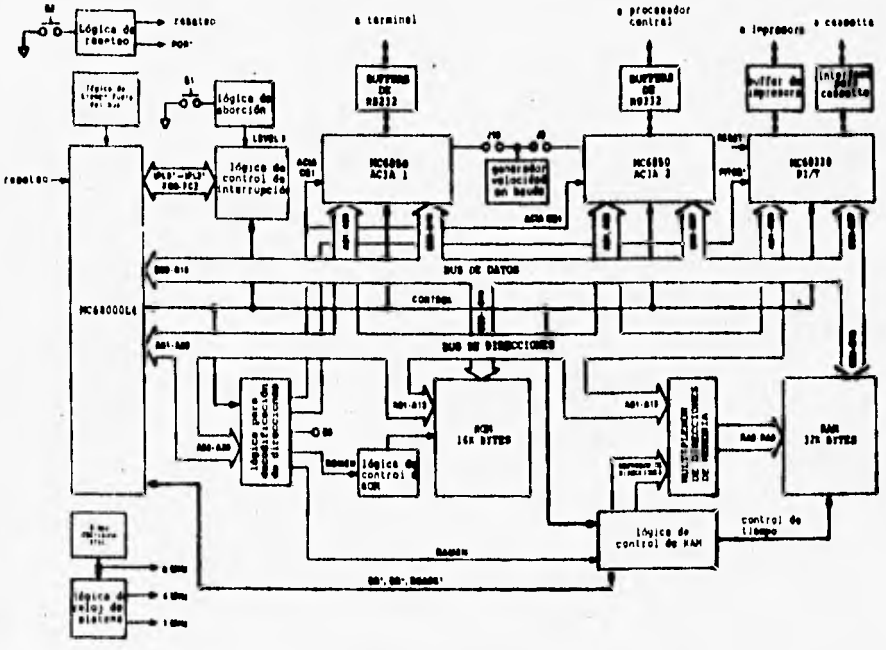


DIAGRAMA BLOQUES- COMPUTADOR EDUCACIONAL

Las áreas funcionales se describen a continuación.

MC68000L4 Microprocesador

Este procesador trabaja a razón de 4 Mhz como se comentó anteriormente. La ejecución de instrucciones consiste en una combinación de ciclos internos y ciclos de acceso al bus.

El microprocesador utiliza transferencias asincrona para llevar datos fuera del procesador. Esto significa que el procesador y los periféricos utilizan líneas de conformidad para coordinar la transferencia de datos.
Usando transferencia asincrona, el procesador permite transferencia de y hacia los periféricos de velocidad diversa.

Decodificador de direcciones

Para entrar a fondo con este elemento, nos basaremos en el mapa de memoria asignado al computador.

MAPA DE MEMORIA

función			dirección
memoria del sistema	tabla de vector excepción	ROM/EPROM ROM	$000000-$000007 (1) $000008-$0003FF
	memoria del tutor	RAM	$000400-$0008FF
memoria para usuario		RAM	$000900-$007FFF (1)
firmware del tutor		ROM/PROM	$008000-$00BFFF
no usada			$00C000-$00FFFF
dispositivos I/O	P1/T (solo parte baja)		$010000-$01003F
	AC1A2 (parte baja) & AC1A1 (parte alta)		$010040-$010043
	mapeo redundante		$010044-$01FFFF
no usada			$020000-$02FFFF
M6800 página (E6)			$030000-$03FFFF
no usada			$040000-$FFFFFF

nota: (1) denota solo lectura

La lógica de decodificación de direcciones habilita la generación de las señales RAMEN, ROMEN, ACIA CS1, y PITCS de acuerdo con el mapa de memoria. Las líneas de dirección A03-A23 son decodificadas y usadas con señales propias de control para generar estas señales.

La memoria RAM es direccionada en la parte inferior o más baja del mapa, excluyendo las ocho primeras localidades, las cuales contienen el apuntador de pila inicial y el contenido del contador de programa grabado en memoria PROM

La RAM es dividida dentro de dos áreas, esto es de $000008 a $0003FF es una área reservada para el sistema básico (firmware). El cual contiene instrucciones inalterables básicas para la operación del equipo.

La memoria RAM comprendida dentro de las direcciones $000900 a $007FFF es área disponible para el usuario.

Direcciones de $000400 a $0008FF son usadas como memoria borrable (scratchpad memory) para el firmware TUTOR incluyendo secuenciador de datos(buffer), apuntadores, memoria temporal, etc.

Todos los dispositivos de entrada/salida son mapeados dentro de la misma página de 64 Kbyte en las diracciones $010000 a $01FFFF.

El arreglo de memoria dinámica esta constituido por 63 chips de memoria MCM4116B (16K x 1). Estos pueden ser accesados en cualquiera de dos formas; por byte o por palabra y la transferencia de datos hacia o desde el MC68000 es a través del bus de transferencia asincrono.

El tiempo de acceso a memoria es aproximadamente de 450 nanosegundos y la señal RAM DTACK (reconocimiento de transferencia de datos) es generada en un tiempo de 500 a 625 nanosegundos después de un comienzo de ciclo de lectura o escritura.

Control lógico de RAM

Este controla las operaciones de memoria RAM. Genera un control temporizado (timing system) para los dispositivos RAM así como señales de control para el multiplexor de direcciones de memoria

El multiplexor genera direcciones en renglón y columna hacia las líneas A01-A14 durante ciclos de lectura y escritura.

Direcciones de refresco (refresh address) son también ruteadas a memoria por el multiplexor durante el refresco de memoria (refresh memory).

La memoria RAM es completamente refrescada cada 1.5 milisegundos en promedio (en el peor de los casos 1.9 milisegundos).

16KBYTE ROM

El sistema firmware (TUTOR) esta grabado en dos ROMs de 64kbits. El tiempo de acceso para las ROMs varía entre 350 a 400 nanosegundos dependiendo del dispositivo usado.

Puerto de comunicación serie

Está formado por dos MC6850 ACIA's que proveen un bus de interfase para puertos serie. ACIA1 es usado para comunicación con una terminal y esta conectado a los bits D08-D15 del bus de datos.

ACIA2 es usado para comunicación con un procesador huésped (host processor).

Generador de velocidad en bauds

Los relojes de transmisión y recepción son suministrados por el generador de velocidad en bauds. La velocidad puede fijarse desde 110 a 9600 baud por medio de la combinación de los puentes J9 y J10. Ambos puertos son RS-232c compatibles.

3. Indirecto con memoria:

   - Indirecto con memoria y post-índice*
   - Indirecto con memoria y pre-índice*

4. Indirecto con contador de programa:

   - Indirecto con PC y desplazamiento de 16 bits
   - Indirecto con PC e índice y desplazamiento de 8 bits*
   - Indirecto con PC e índice y desplazamiento de 16 ó 32* bits

5. Indirecto con memoria y con el contador de programa:

   - Indirecto a memoria con PC y post-índice*
   - Indirecto a memoria con PC y pre-índice*

6. Absoluto:

   - Absoluto corto
   - Absoluto amplio

7. Inmediato

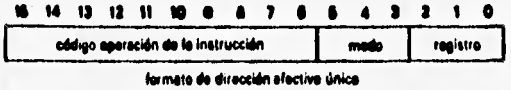
Los modos marcados con un asterisco (*) o bien tienen funciones ampliadas en el 68020 o bien existen sólo en el 68020.

Codificación de direcciones

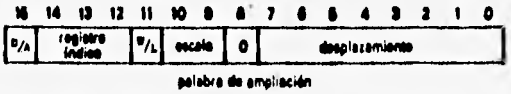
Las instrucciones del 68K codifican las direcciones de operandos de una manera específica. La primera palabra especifica la operación y, en muchos casos, la dirección de un operando. Las especificaciones de operandos adicionales van a continuación de esta primera palabra. En los primeros cuatro miembros de la familia 68K hay, como mucho, cuatro palabras de "ampliación". El 68020 permite hasta 10 palabras de ampliación (más que en los otros a causa de los modos adicionales de direccionamiento).

Las direcciones efectivas se codifican con ayuda de un campo modo y un campo registro. Cada uno de esos campos tiene una longitud de 3 bits y están embebidos en la primera palabra de instrucción. Cualquiera de las palabras de ampliación que sigan pueden ser sencillamente o bien direcciones de operandos, en forma de direcciones absolutas, o pueden ser formas bastante complejas con campos de direcciones embebidos o con modos de direccionamiento indirecto a memoria. Los campos de ampliación más complejos incluyen valores para indexación y direccionamiento indirecto, además de para la presencia o no y en su caso, tamaño de desplazamiento.

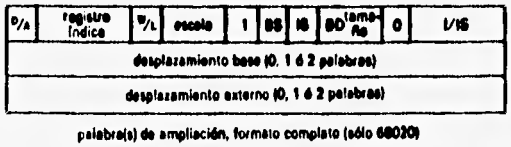
La siguiente tabla resume los campos del modo de direccionamiento tanto en el interior de la palabra instrucción como en el interior de las palabras de ampliación.



formato de dirección efectiva única



palabra de ampliación



palabra(s) de ampliación, formato completo (sólo 68020)

registro	registros de datos o direcciones
modo	modo de direccionamiento
código de operación	instrucción y posible información sobre modo/registro para el segundo operando
desplazamiento	valor con signo de 8 bits
escala	índice de factor de escala (sólo en 68020) 00=1X 01=2X 10=4X 11=8X

I/IS	Selección indexado/indirecto (sólo 68020,
BD tamaño	Tamaño de desplazamiento base (sólo 68020) 00 reservado 01 desplazamiento nulo 10 desplazamiento de palabra 11 desplazamiento largo
IS	Supresión de índice (sólo 68020,
BS	Supresión de base (sólo 68020) 0 evaluar y añadir registro base 1 suprimir registro base
W/L	Tamaño de registro índice 0 palabra extendida en signo 1 palabra larga con signo
registro índice	registro de datos o de direcciones (000-111)
D/A	Tipo de registro índice 0 registro de datos 1 [illegible] de dirección

La siguiente tabla resume los valores modo/registro y sus funciones.

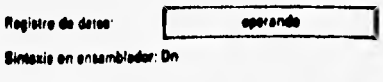
Codificación modo/registro.

Modo	Registro	Operación de direccionamiento
000	reg #	Directo a registro de datos
001	reg #	Directo a registro de dirección
010	reg #	Indirecto a registro de dirección
011	reg #	Indirecto a registro de dirección con post-incremento
100	reg #	Indirecto a registro de dirección con predecremento
101	reg #	Indirecto a registro de dirección con desplazamiento
110	reg #	Indirecto a memoria con registro de dirección con índice*
111	000	Absoluto corto
111	001	Absoluto largo
111	010	Indirecto con el contador de programa y desplazamiento
111	011	Indirecto a memoria con el contador de programa con índice*
111	100	Dato inmediato
111	101-111	Reservado
		* solo el 68020

II.7 Modos de direccionamiento

Directo a registro de datos.

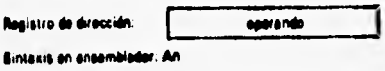
En el modo directo a registro de datos, un registro de datos contiene el operando.



Sintaxis en ensamblador: Dn

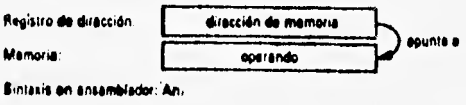
Directo a registro de direcciones.

En el modo directo a registro de direcciones, un registro de dirección contiene el operando.



Indirecto con registro de dirección.

En el modo de direccionamiento indirecto con registro, un registro de dirección contiene la dirección del operando (esto es, "apunta" al operando).

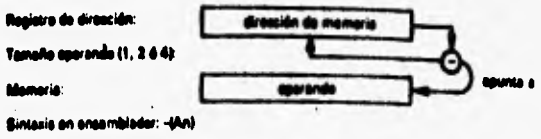


Indirecto con registro de dirección y post-incremento.

Como en el modo sencillo indirecto con registro de dirección, un registro de dirección, en el modo indirecto con post-incremento, contiene la dirección del operando. Sin embargo, después de usar el operando, el procesador suma el registro de dirección, la longitud del operando en bytes (esto es, 1 para una operación con 1 byte, 2 para una operación con palabra o 4 para una operación con palabra larga).

Indirecto con registro de dirección y predecremento.

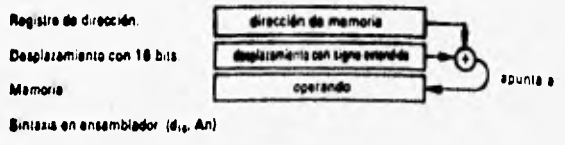
Como en el caso sencillo del modo indirecto con registro de dirección, en el modo indirecto con pre-decremento, un registro de dirección contiene la dirección del operando.



Sin embargo, antes de usar el operando, el procesador resta la longitud del operando al registro de dirección (esto es, 1 para una operación de un byte, 2 para una operación con palabra o 4 para una operación con palabra larga).

Indirecto con registro de dirección y desplazamiento.

En el modo indirecto con desplazamiento, el operando reside en la dirección dada por la suma de los contenidos de un registro de dirección y un desplazamiento de 16 bits. El valor del desplazamiento es trasladado en signo a 32 bits para permitir desplazamientos negativos.

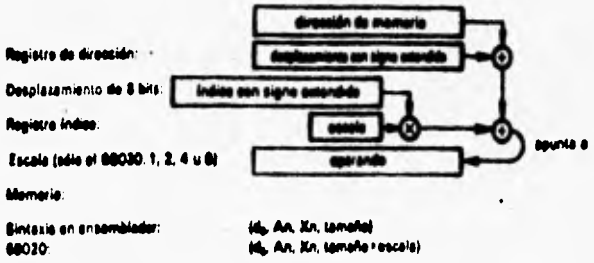


Indirecto con registro de dirección e índice y desplazamiento.

En el modo indirecto con índice y desplazamiento el operando reside en una dirección dada por la suma de los contenidos de un registro de dirección, un desplazamiento y el producto del registro índice por un factor de escala.

Si la base del desplazamiento es de 8 ó 16 bits, está el signo trasladado a 32 bits. (Hay que hacer notar que la versión de 8 bits de este tipo de direccionamiento sólo existe en el 68020.)

El valor del registro índice sólo puede ser un valor de 16 ó 32 bits; si es 16 bits, el procesador traslada el signo a 32 bits antes de añadirlo. El factor de escala puede ser 2E0(1), 2E1(2),2E2(4) ó 2E3(8), Sólo el 68020 admite índice escalado.

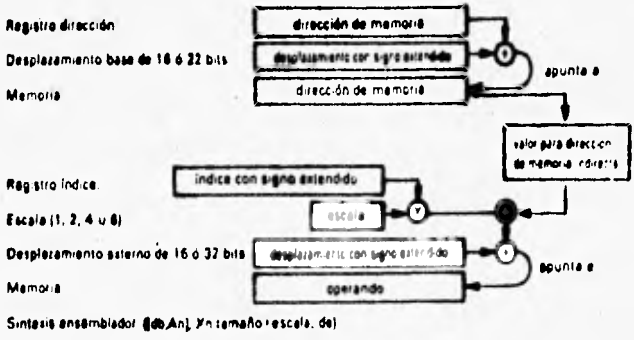


Indirecto con memoria y post-índice.

En el modo indirecto con memoria y post-índice, la dirección del operando se obtiene a partir de cuatro valores: el contenido de un registro de dirección, un desplazamiento base de 16 ó 32 bits, el producto del valor en un registro índice por la escala y un segundo desplazamiento (externo).

La base y el desplazamiento externo puede tener una longitud de 16 ó 32 bits, el procesador traslada el signo de los valores antes de añadirlos. El registro índice es también ampliado y trasladado el signo (cuando se requiera) antes de añadirlo a la dirección efectiva.

Los cuatro valores son opcionales; si se omiten, el procesador suma un valor cero de contribución al cálculo de la dirección efectiva. Al calcular la dirección efectiva, el procesador añade el valor del registro de dirección al desplazamiento base. Después utliza este valor para obtener un segundo valor de dirección. Para este segundo valor, el procesador añade la escala del registro índice. Finalmente, añade el desplazamiento externo para obtener la dirección del operando.

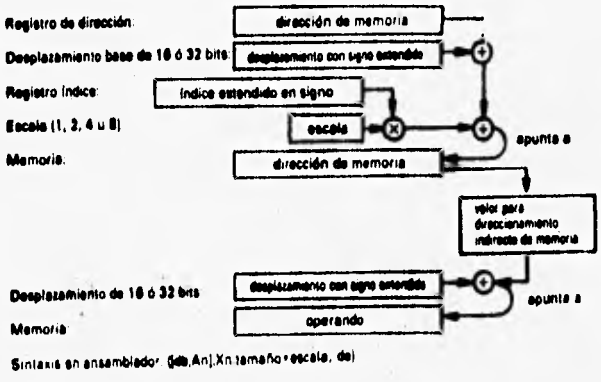


Indirecto con memoria y pre-índice.

En el modo indirecto con memoria y pre-índice (exclusivo del 68020), la dirección del operando se obtiene a partir de cuatro valores: el contenido de un registro de dirección, un desplazamiento base de 16 ó 32 bits, el producto de valor de un registro índice y la escala y un segundo desplazamiento externo.

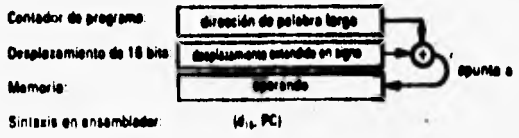
La base y el desplazamiento externo puede ser de 16 ó 32 bits; si son de 16 bits el procesador traslada el signo antes de sumarlos. El registro índice es también ampliado y su signo trasladado (cuando sea necesario) antes de ser sumado a la dirección efectiva. los cuatro valores son opcionales; si se omiten, el procesador supone un valor caro para esa contribución a la dirección efectiva.

Al evaluar la dirección efectiva, el procesador añade el valor del registro de dirección al desplazamiento base y el valor escala del registro índice. Después utiliza este valor para obtener una segunda dirección. Para calcular esta segunda dirección, el procesador suma el desplazamiento externo consiguiendo así la dirección del operando.



Indirecto con PC y desplazamiento.

En el modo indirecto con PC y desplazamiento, el procesador genera una dirección efectiva sumando el valor del contador de programa y el valor con el signo trasladado de esa palabra de ampliación de 16 bits.

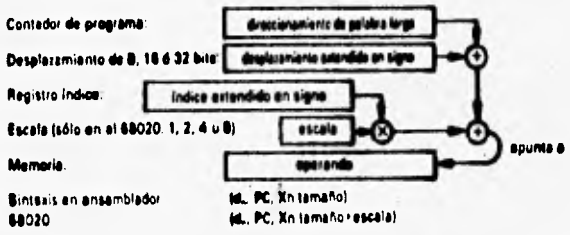


Indirecto con PC e índice y desplazamiento.

En el modo indirecto con contador de programa (PC) y desplazamiento, la dirección efectiva es la suma de la dirección en el contador de programa el valor en la palabra de ampliación y el valor en el registro índice (multiplicando por la escala, en el 68020).

El valor en el contador de programa en el momento de la evaluación será la dirección de la palabra de ampliación. El procesador trasladará el signo del desplazamiento si es de 8 ó 16 bits (la versión de ocho bits es sólo en el 68020).

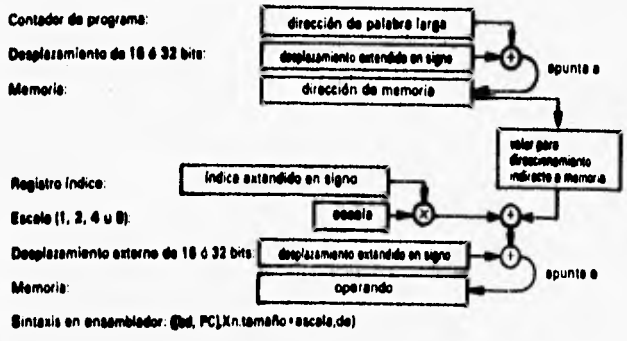
El índice del valor en registro índice puede ser trasladado, si es necesario, así como el del valor escalado, en el procesador 68020.



Indirecto a memoria con PC post-índice.

En el modo indirecto a memoria con PC y post-índice, la dirección efectiva se calcula a partir de cuatro valores. La suma del PC y un desplazamiento base se utiliza como una dirección. El procesador suma el valor de dicha dirección con una escala en el registro índice y con el desplazamiento externo, obteniendo así la dirección del operando.

El valor del contador de programa es el de la dirección de la palabra de ampliación. La base y el desplazamiento externo pueden ser de 16 ó 32 bits. El valor en el registro índice puede ser 1, 2, 4 u 8. Todos los componentes utilizados en el cálculo de la dirección del operando son opcionales; si se omiten, el procesador asume para ellos el valor cero.

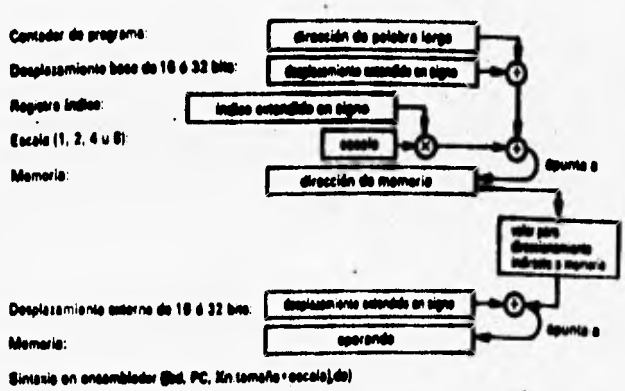


Indirecto a memoria con PC y pre-índice.

En el modo indirecto a memoria con PC y pre-índice, el cálculo de la dirección efectiva se hace con cuatro valores. La suma del PC, con el desplazamiento base y con la escala en el registro índice, se usa como una dirección de partida. El procesador suma el valor de esa dirección con el desplazamiento externo para dar la dirección del operando.

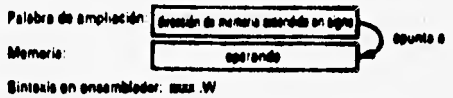
El valor del contador de programa es la dirección de la palabra de ampliación. Los desplazamientos base y externo pueden ser de 16 ó 32 bits; si es de una longitud de 16 bits, el procesador traslada el signo antes de usar el valor.

El valor en el registro índice puede estar en un byte, una palabra o una palabra larga y su signo se traslada si es necesario. El valor de escala pude ser 1, 2, 4 u 8. Todos los componentes utilizados en el cálculo de la dirección del operando son opcionales; si se omiten, el procesador supone un valor cero.



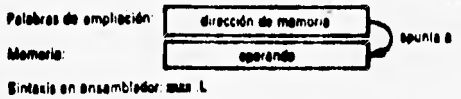
Absoluto corto.

En el direccionamiento absoluto corto la dirección del operando es el valor, con el signo trasladado, de la palabra de 16 bits de ampliación que sigue a la instrucción.



Absoluto largo.

En el direccionamiento absoluto largo, la dirección del operando está en la palabra de ampliación (32 bits) que sigue a la palabra instrucción.



Dato inmediato.

En el direccionamiento inmediato, el propio operando sigue a la propia instrucción. Dependiendo del tamaño del operando, la instrucción necesita una o dos palabras de ampliación.

II.8 Conjunto de instrucciones.

EL MC68000 cuenta con un set de instruciones largo y ortogonal, se dice que son ortogonales debido a que diferentes instrucciones de adición tienen el mismo modo de direccionamiento.

Captura anticipada de línea de espera

Algo especial en el diseño del procesador MC68000 es una instrucción de captura anticipada de línea de espera (prefetch queue). Mientras una instrucción comienza a ejecutarse, otra puede ser decodificada y otra puede ser capturada.

Tipo de datos

El MC68000 puede trabajar con bits,cuarteos (nibbles), bytes, palabra y palabra larga. Estos tipos de datos trabajan con bastantes instruciones lo que provee una gran flexibilidad de programación.

Definiremos las instrucciones del MC 68000 en forma de grupos de instrucciones, de modo que sea más clara la función de cada instrucción y más concreta. Los grupos de instrucciones son los siguientes:

Movimiento de datos
Aritméticas enteras
Decimales
Lógicas
Corrimiento y rotación
Manipulación de bit
Control de programa
Control de sistema
Instrucciones nulas

Instrucciones de movimiento de datos.

Estas instrucciones se describen primero debido a que se usan al principio de programas. Cualquier programa necesita mover varios bits y bytes hacia fuera o dentro del CPU y memoria. Las siguientes son las instrucciones de movimiento.

EXG,LEA,LINK,MOVE,MOVEM,MOVEP,MOVEQ,PEA,SWAP,UNLK.

EXG es una instrucción que intercambia el contenido del operando destino con el contenido del operando fuente.

LEA es una instrucción de carga de dirección efectiva, la dirección efectiva es información de dirección calculada desde el modo de direccionamiento y especifica un dato. Este es puesto en un registro de dirección y puede usarse para accesar una tabla en memoria.

LINK (enlace) y UNLINK (des-enlace) son instrucciones avanzadas, sirven para manipulación de pila en cuanto a organización de áreas de memoria, para subrutinas, funciones y módulos de programa.

MOVE cuando combinamos con el modo de direccionamiento , byte, palabra y palabra larga, esta simple instrucción es capaz de mover cualquier pedazo de información donde quiera que las líneas del bus del microprocesador lleguen. MOVE puede mover información de un registro a otro registro, de un registro a memoria, o de memoria a memoria.

MOVEM esta instrucción mueve los datos en grupos de registros hacia o desde memoria. Es muy usada ya que ahorra trabajo de programación y en lenguajes de alto nivel donde es necesario salvar y restaurar información.

MOVEP esta instrucción sirve para mover datos con los periféricos, facilita la transferencia de información enviada o recibida por el CPU con los periféricos.

MOVEQ es una instrucción de movimiento rápido debido a esto solo puede mover bytes de datos inmediatos a registros de datos.

PEA sirve para empujar la dirección efectiva dentro de la pila, esta instrucción decrementa el apuntador de pila por dos (hay dos bytes en una palabra) y pone la palabra baja de la dirección efectiva en la pila, entonces el apuntador de pila es decrementado otra vez por dos y la palabra alta de la dirección efectiva es puesta o empujada dentro de la pila.

SWAP esta instrucción sirve para mover palabras, intercambiando palabra baja y alta de un registro de datos.

Instrucciones aritméticas.

Las instruciones aritméticas son las siguientes

ADD, ADDX,CLR,CMP,DIVS,DIVU,EXT,MULS,MULU,NEG,NEGX,SUB,SUBX, TAS,TST.

Mientras el MC68000 puede ejecutar las mismas operaciones de adición y sustración que los otros microprocesadores, además puede ejecutar instrucciones de multiplicación y división.

ADD es una instrucción de adición aritmética binaria, suma el operando fuente y operando destino, el resultado es almacenado en el operando destino , borrando el valor anterior de este operando.

ADDX es una instrucción que ejecuta suma extendida ya que puede sumar bytes, palabras y palabras largas. Hay dos formas de addx: registro a registro y memoria a memoria.

CLR es la instrucción más simple, pone cero en el destino. El operando destino puede ser un registro de datos o una localidad de memoria y es llenada con tantos ceros como el tamaño de su especificación (byte,palabra o palabra larga).

CMP compara un operando fuente con un operando destino (substrayendo el contenido de la fuente con el contenido del destino).

DIVS y DIVU son instrucciones de división signada y no signada respectivasmente. DIVU divide un operando destino de 32 bits entre un operando fuente de 16 bits usando números binarios no signados. El resultado es de 32 bits es almacenado en el destino (el cual debe ser un registro de datos). DIVS hace exactamente lo mismo pero usando números signados.

EXT extiende el bit de signo de un número binario que puede ser interpretado como un número signado o no signado. Cuando se interpreta como signado el bit más significativo es leído como el signo del número : un cero significa positivo y un uno significa negativo.

MULS y MULU son instrucciones de multiplicación, una de las ventajas de los 16 bits del 68000 es que tiene multiplicación y división, circuito integrado de 8 bits tienen que ejecutar varias operaciones con rutinas de movimiento, corrimiento y comparación. Para obtener una división o una multiplicación.

MULU multiplica no signados y MULS multiplica signados, MULU multiplica dos operandos de 16 bits no signados produciendo un resultado de 32 bits no signado. MULS multiplica dos operandos de 16 bits signados y produce un resultado de 32 bits signado.

NEG es una instrucción de negación. El valor del operando destino es sustraido de cero usando complemento binario a dos, el resultado es almacenado en el destino.

NEGX es una instrucción de negación extendida y es una forma especial de negar que afecta la bandera X. El operando y la bandera X son sustraidos de cero.NEGX intenta simplificar trabajo aritmético de multiple-presición.

SUB es una instrucción de sustracción binaria, el operando fuente es sustraido del operando destino.

SUBX es una instrucción de sustración extendida.

TAS y TST son operaciones especiales de comparación, TST sustrae cero del contenido del destino y manipula las condiciones de código de banderas de acuerdo al resultado.TAS es más compleja, puede trabajar con un solo byte y puede direccionarse de ocho diferentes maneras.

Instrucciones decimales.

Aunque las computadoras trabajan con 1´s y 0´s binarios no toda la aritmética esta hecha usando las reglas básicas.

ABCD es una instrucción de suma usando código BCD con extensión, puede sumar dos operandos que esten en memoria.

SBCD es una instrucción que sustrae datos en BCD y actúa de la misma forma que ABCD.

NBCD es la negación BCD con extensión, el operando es negado por la sustracción de este y de la bandera X con cero.

Existen instrucciones avanzadas dentro de este grupo que son:

ABCD,BCHG,BCLR,BSET,CHK,DBcc,ILLEGAL,LEA,LINK,NBCD,PEA,RESET
RTE,RTR,SBCD,Scc,SWAP,TAS,TRAP,TRAPV,TST,UNLK.

Instrucciones lógicas.

Todas las computadoras digitales hacen uso pesado de operaciones lógicas, el MC68000 provee las siguientes instrucciones lógicas:

AND, OR, EOR, NOT.

Operaciones lógicas son usadas para limpiar, encender y provar posiciones específicas de bits dentro de bytes, palabra y palabra larga.

AND compara el operando fuente con el operando destino, bit por bit en cada posición siguiendo la regla de la operación AND. El resultado es almacenado dentro del operando destino. AND es usada generalmente para crear máscaras.

OR compara el operando fuente con el operando destino bit por bit de acuerdo a la operación lógica OR, el resultado es almacenado en el operando destino. OR también es usado para crear máscaras.

EOR compara el operando fuente con el operando destino bit por bit de acuerdo a la operación lógica EXOR. El resultado es almacenado en el operando destino. Esta instrucción es usada para invertir bits.

NOT completa el valor del operando destino y almacena el nuevo valor en el operando destino. Esta instrucción hace el complemento a uno.

Instrucciones de corrimiento y rotación.

Estas instrucciones no corresponden con lenguajes de alto nivel. La habilidad de mover o cambiar el contenido de un operando con localidades de operandos no usadas.
Las instrucciones de corrimiento y rotación son:

ASL, ASR, LSL, LSR, ROL, ROR, ROXL, ROXR.

ALS es una instrucción que hace un corrimiento aritmético a la izquierda, el término aritmético es para diferenciarlo del lógico.

ASR es una instrucción que hace un corrimiento aritmético a la derecha , mueve el valor de bits hacia la derecha.

LSL es una instrucción que hace un corrimiento lógico a la izquierda poniendo un cero al bit menos significativo. El bit más significartivo que sale es almacenado en las banderas C y X.

LSR es una instrucción que hace un corrimiento lógico a la derecha .

ROL es una instrucción de rotación a la izquierda, el bit menos sigificativo es copiado en la bandera C y copiado también en la posición del bit más significativo.

ROR es una instrucción de rotación hacia la derecha.

ROXL es una instrucción de rotación hacia la izquierda extendida, el bit más significativo es copiado a la bandera X y el contenido de esta bandera es copiado en la posición del bit menos significativo.

ROXR es una instrucción de rotación hacia la derecha extendida similar a ROXL.

Instrucciones de manipulación de bit.

Estas instrucciones pueden trabajar en bits individuales en lugar de bytes, palabras o palabras largas. Existen 4 tipos de manipulación de bit en el MC68000:

BTST, BSET, BCLR, BCHG.

BTST es una instrucción de prueba de bit, simplemente prueba el valor de una localidad particular de bit y usa la bandera Z para comunicar el valor. Si la bandera Z esta encebdida, el bit es 1, si esta apagada el bit es cero.

BSET es una instrucción de prueba y encendido de bit, después de probar un bit particular, BTST solo trabaja sobre la bandera Z. Bset prueba un bit particular, trabaja también sobre la bandera Z y después enciende el bit probado poniendolo en el valor 1.

CAPITULO III

ANALISIS DEL SISTEMA

III ANALISIS DEL SISTEMA

El análisis es un proceso lógico, el objetivo de esta fase no es resolver el problema inmediatamente, pero si determinar exactamente qué es lo que debe hacerse para resolverlo. Una metodología estructurada es muy útil ya que con una aproximación estructurada, el criterio de salida específico debe ser cumplido para cada paso del proceso. El objetivo básico de la etapa de análisis es desarrollar un modelo lógico de el sistema, usando herramientas tales como diagramas de flujo de datos , un diccionario elemental de datos, descripciones esquemáticas de los algoritmos relevantes. Este modelo lógico es sujeto de revisiones tanto del usuario como de los analistas, quienes deben concluir que el modelo refleja realmente lo que debe hacerse para resolver el problema.

El diagrama de flujo de datos tiene como objeto mostrar las transformaciones de los datos a medida que estos fluyen a través de los procesos del programa; es decir ayuda a analizar los cambios que ocurren a los datos de entrada a fin de lograr la salida deseada.

III.1 Conceptualización del problema

Definiremos primero el objetivo del sistema para hacerlo más claro, se quiere desarrollar un sistema que partiendo de un programa en lenguaje ensamblador genere su correspondiente lenguaje de máquina. Y esto que quiere decir ? , bueno comenzaremos por escribir lo que un programador necesita para ensamblar sus programas, el lenguaje ensamblador son un conjunto de instrucciones llamados mnemónicos, estos tienen un código equivalente en lenguaje máquina que se arma a través de tablas en las cuales se toman muchos aspectos en cuenta, tales como ; modo de direccionamiento, registros usados, tamaño de los operandos, cual de estos operandos es operando destino y cual es operando fuente, dirección efectiva, etc. Y ahora hablando del término macroensamblador, entran más conceptos como; macro definición, llamadas a macros, literales, símbolos, etc. De este modo tenemos que a parte de ser complejo programar en lenguaje ensamblador. es muy tardado ya que ensamblar las instrucciones a código máquina es una tarea muy laboriosa.

Por lo que el objetivo de este sistema es resolver la parte de ensamble y macrodefiniciones que existan dentro del programa en lenguaje mnemónico, para que de este modo, el usuario del sistema agilice su labor de prueba y ejecución de sus programas.

III.2 Metodología de análisis

Definiremos ahora una primera aproximación hacia la solución del problema, en la que se incluye todo el sistema de una forma general, y partiendo de esta empezar a modelar el sistema. Como entrada al sistema se tiene un programa en lenguaje ensamblador que se llamará LENGUAJE FUENTE, un proceso en el que se llevarán a cabo todas las transformaciones de la entrada, que se llamará MACROENSAMBLADOR , el cual tiene otras dos entradas que son DIRECTIVAS y CODIGO DE INSTRUCCION, la salida del proceso será un programa en lenguaje de máquina que se llamará PROGRAMA OBJETO. Empleando el método de análisis por diagramas de flujo de datos (DFD), la siguiente figura 1 muestra esta primer aproximación.

La siguiete etapa consiste en dar un paso hacia adentro del sistema, para definirlo más a detalle. El macroensamblador se puede separar en dos procesos que llamaremos: MACROPROCESADOR y ENSAMBLADOR , el primero tiene como entrada el código fuente y lo transforma en código expandido , el segundo toma como entrada el código expandido y lo transforma en código objeto, más adelante veremos como es que hacen estas acciones los procesos. La gráfica número 2 muestra esta segunda aproximación.

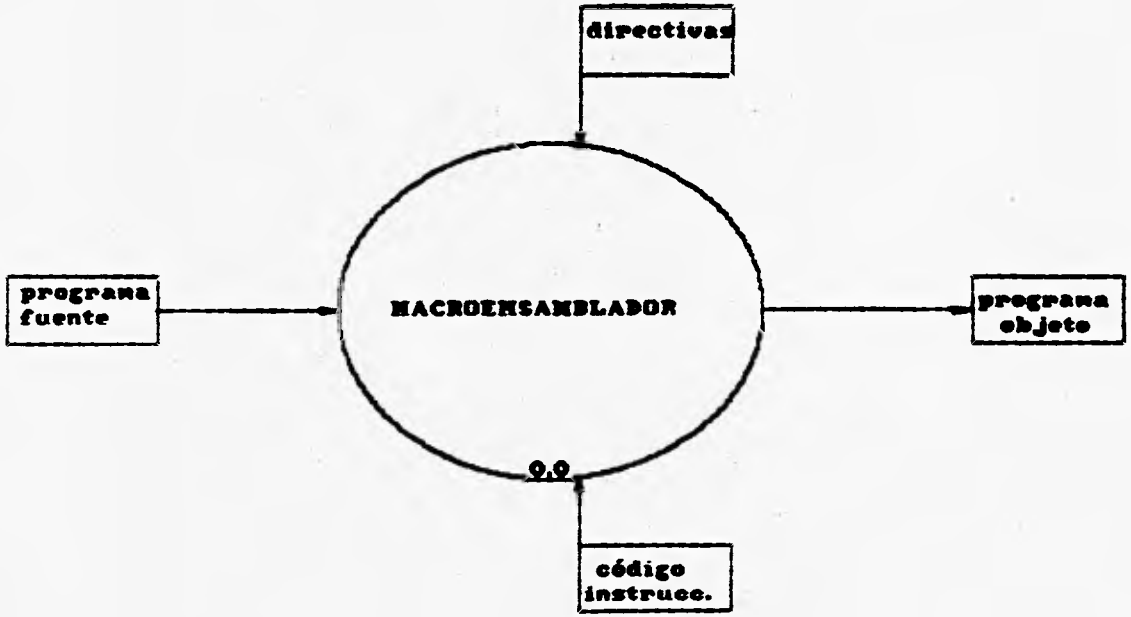

directivas
programa fuente
MACROENSAMBLADOR
programa objeto
código instruec.

FIG. 1

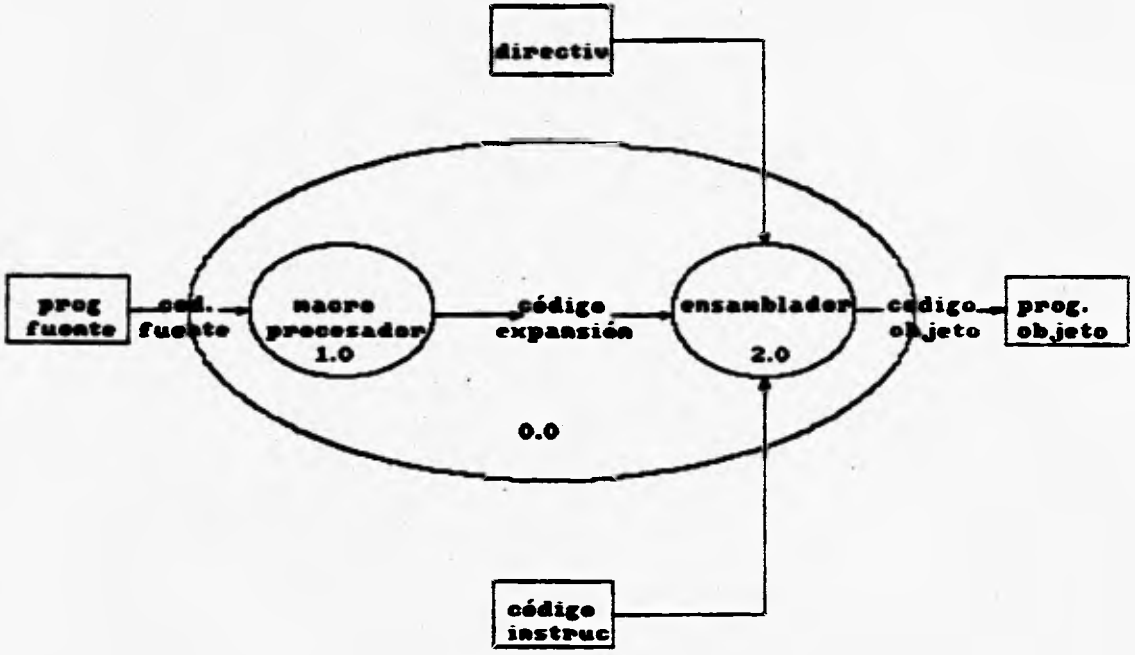

directiv
prog
fuente
cód.
fuente
macro
procesador
1.0
código
expansión
ensamblador
2.0
código
objeto
prog.
objeto
0.0
código
instruc

FIG. 2

La tercera aproximación del análisis del sistema es definir por separado el macroprocesador y el ensamblador.

Macroprocesador

El macroprocesador lo definiremos subdividiendolo en dos procesos que llamaremos: RECOPILAR MACRO DEFINICIONES y EXPANDIR MACRO LLAMADAS. En el primero se genera un contenedor de datos donde serán guardados los nombres y cuerpos de macros, en el segundo se sustituyen las llamadas a macros con los cuerpos correspondientes de estas macros, y de esta forma generar el código de expansión.

Recopilar Macro Definición

Definiremos ahora como estará formado el proceso de recopilar macro definiciones, lo podemos subdividir en dos procesos; IDENTIFICAR MACRO DEFINICION e IDENTIFICAR EXPANSION , el primero solo toma los nombres de las macros y los guarda en un contenedor, el segundo toma el cuerpo de la macro y lo guarda en otro contenedor de datos.

La gráfica 3 muestra el macroproceaador y la recopilación de macros.

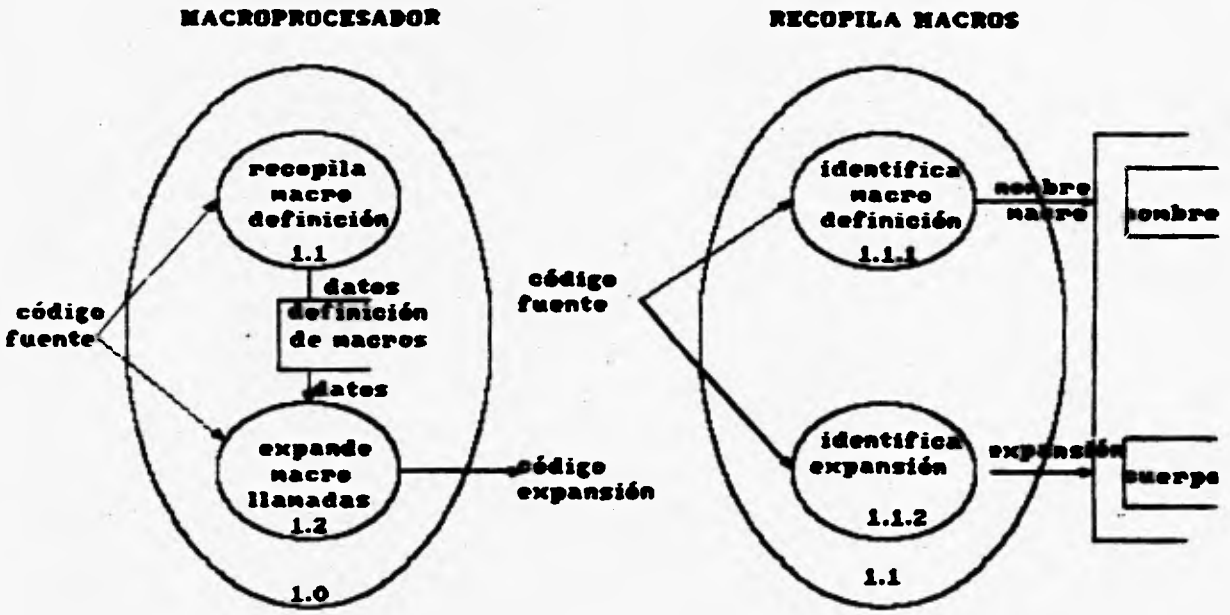

MACROPROCESADOR
RECOPILA MACROS
recopila macro definición
1.1
datos definición de macros
datos
expande macro llanadas
1.2
1.0
código fuente
código expansión
código fuente
identifica macro definición
1.1.1
identifica expansión
1.1.2
1.1
nombre macro
nombre
expansión
cuerpo

FIG. 3

Expandir macro llamadas.

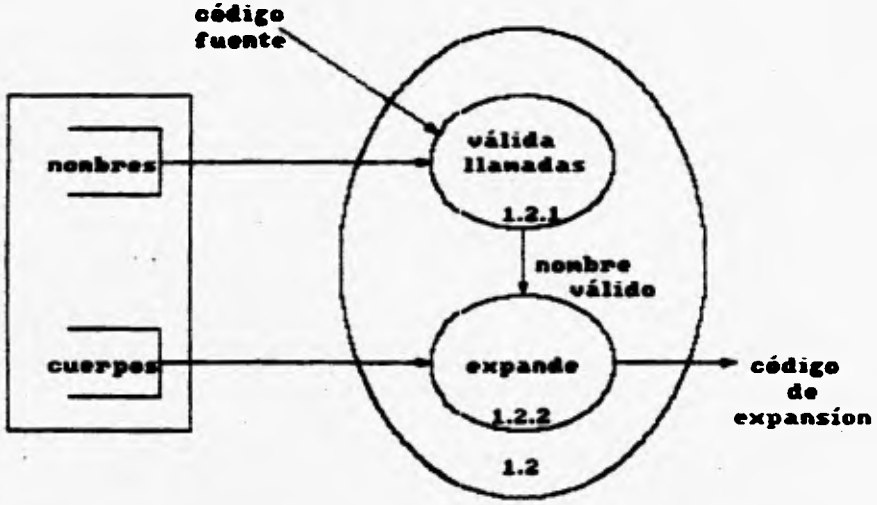
El proceso de expandir macro llamadas, lo definiremos como otros dos procesos; VALIDAR LLAMADA y EXPANSION. El proceso de validar llamada tiene como entrada los nombres de las macros y también el código fuente, y su función es verificar que el nombre de la macro recolectada existe dentro del código fuente y dar como salida los nombres de macros válidos.

El subproceso de expansión,tiene como entrada el cuerpo de la macro y el nombre válido, con este nombre válido localiza el cuerpo que le corresponde y lo sustituye en el lugar donde hizo la llamada a macro. Como salida del proceso de expansión tenemos el código de expansión. La gráfica 4 muestra este proceso.

Expandir macro llamadas.

El proceso de expandir macro llamadas, lo definiremos como otros dos procesos; VALIDAR LLAMADA y EXPANSION. El proceso de validar llamada tiene como entrada los nombres de las macros y también el código fuente, y su función es verificar que el nombre de la macro recolectada existe dentro del código fuente y dar como salida los nombres de macros válidos.

El subproceso de expansión,tiene como entrada el cuerpo de la macro y el nombre válido, con este nombre válido localiza el cuerpo que le corresponde y lo sustituye en el lugar donde hizo la llamada a macro. Como salida del proceso de expansión tenemos el código de expansión. La gráfica 4 muestra este proceso.



EXPANDE MACROLLAMADAS

FIG. 4

Ensamblador.

La otra parte o subproceso del macroensamblador que hace falta definir es el ensamblador. En este proceso la entrada son directivas, código fuente expandido y código de instrucción. La transformación de estos tres datos tiene como resultado o salida el código objeto. La gráfica 5 muestra este proceso.

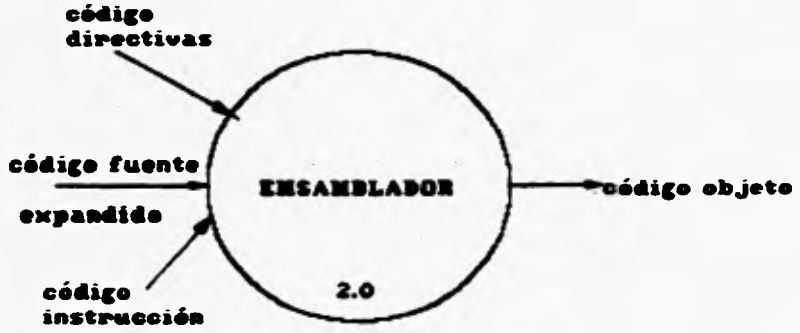

código
directivas
código fuente
expandido
código
instrucción
ENSAMBLADOR
2.0
código objeto

FIG. 5

El proceso de ensamble lo podemos definir o subdividir en dos procesos que llamaremos ; RECOPILAR SIMBOLOS y LITERALES y GENERADOR DE INSTRUCCIONES.

Recopilar símbolos y literales.

Este subproceso tiene como entradas el código de instrucción, el código fuente y las directivas. La función de este proceso es como su nombre lo indica, tomar solo los símbolos y las literales que serán la salida de este proceso.

Generador de instrucciones.

El generador de instrucciones tiene como entradas el código de instrucción, el código fuente , las directivas, literales y símbolos. Con todos estos elementos el generador de instrucción arma la instrucción en lenguaje máquina ya que la salida es el código objeto. La gráfica 6 muestra este proceso.

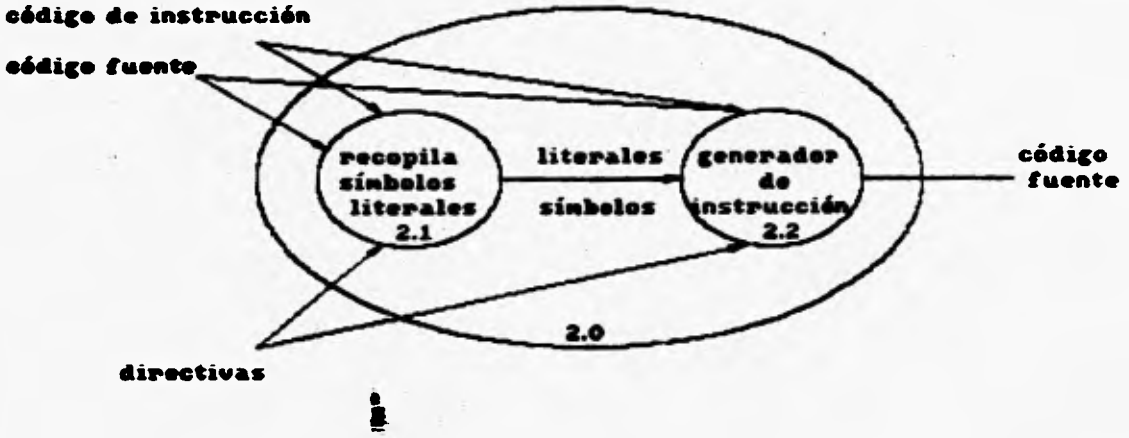

código de instrucción
código fuente
recopila
símbolos
literales
2.1
literales
símbolos
generador
de
instrucción
2.2
código
fuente
2.0
directivas

FIG. 6

El proceso de recolección de símbolos y literales, lo volvemos a subdividir en dos procesos, uno que IDENTIFIQUE SIMBOLOS y otro que IDENTIFIQUE LITERALES.

El proceso generador de instrucción también se puede subdividir en dos procesos más ; uno de VALIDACION DE INSTRUCCION y otro de ARMAR OBJETO. La gráfica 7 muestra los procesos descritos.

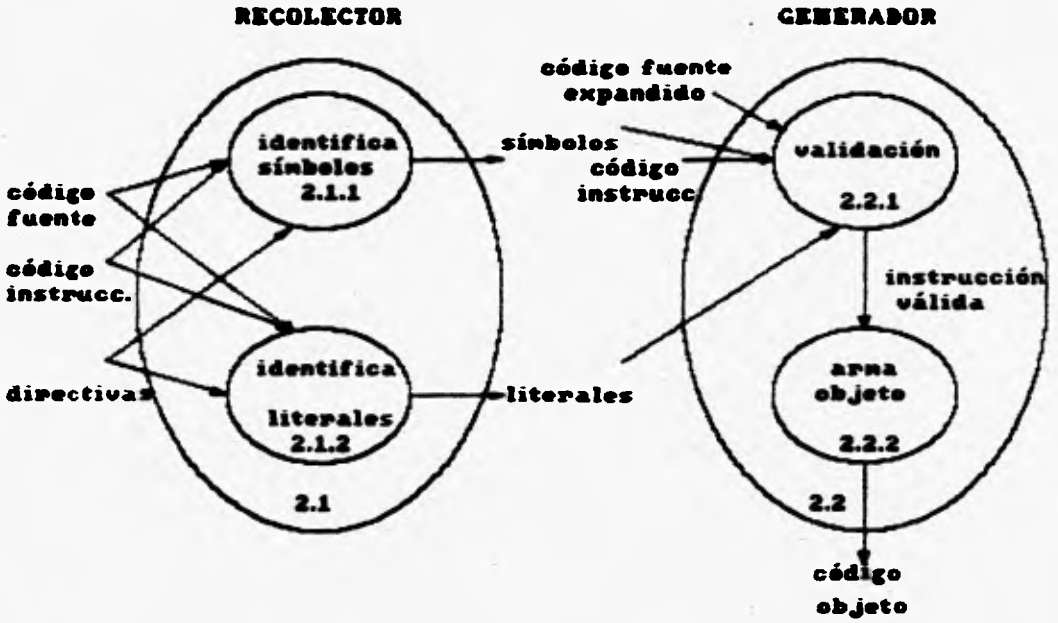

RECOLECTOR
GENERADOR
código fuente expandido
identifica símbolos 2.1.1
símbolos
código instrucc
validación 2.2.1
código fuente
código instrucc.
instrucción válida
identifica literales 2.1.2
directivas
literales
arma objeto 2.2.2
2.1
2.2
código objeto

FIG. 7

Con todos estos diagramas estructurados se tiene una aproximación definida de lo el sistema hace.

CAPITULO IV

DISEÑO DEL SISTEMA

IV DISEÑO DEL SISTEMA

Una vez que la etapa de análisis ha quedado terminada, el siguiente paso es determinar, en forma de un esbozo amplio, como el problema puede ser resuelto. Durante el diseño del sistema, el objetivo es moverse de lo lógico a lo físico.
Pueden ser consideradas distintas alternativas de solución, la cuestión a ser resuelta durante el diseño del sistema es : ¿como, en general, puede ser solucionado el problema ?.
Varias son las etapas que debe cubrir el diseño de sistemas:

1. Especifición del problema.
2. Identificación de módulos.
3. Identifición de entradas y salidas.
4. Especifición de estructuras de datos.
5. Repetir los pasos del 1 al 5 para cada módulo.

IV.1 Especifición del problema

Para el desarrollo del ensamblador es necesario separar la problemática en una serie de etapas lógicas que en forma posterior servirán como módulos "naturales" del sistema.

Acotemos primeramente el problema. Tratamos de desarrollar un programa traductor de instrucciones en mnemónicos a instrucciones de máquina, las instrucciones que vamos a traducir pueden ser macroinstrucciones, instrucciones con símbolos y literales, instrucciones simples y seudo instrucciones (directivas).

Existe una relación uno a uno entre las instruciones simples (formadas por mnemónicos y operandos que pueden ser constantes, registros de direccionamiento o registros de datos) y las instruciones en código máquina. Un primer intento de diseño nos permite reconocer 2 etapas principales del problema:

I Convertir las instrucciones "complejas" en instrucciones simples.

II Convertir las instruciones simples a su correspondiente código de máquina.

Por una instrucción compleja se entiende cualquier instrucción que agrupe bajo su nombre a un conjunto de instrucciones o bien una instrucción cuyos operandos son literales o símbolos sin un valor directo (no registros ni constantes), todas estas instrucciones son llamadas macroinstrucciones y seudoinstrucciones. Conforme a la definición dada de macroinstrucción en los cápitulos anteriores, los pasos a seguir para descomponerla en sus instrucciones simples serán:

- Recopilar las definiciones de Macros.

- Conservar el cuerpo de la Macro.

- Identificar la llamada de la Macro.

- Expandir o sustituir el cuerpo de la Macro.

Hasta este momento el conjunto de instrucciones que tiene nuestro programa "fuente" son instrucciones donde no aparecen llamadas a Macros. El siguiente paso es asignar valores a las literales y símbolos que puedan aparecer en el programa, es decir, procesar cierto tipo de seudoinstrucciones:

- Identificar símbolos y guardar su valor.

- Identificar literales y guardar su valor.

Ahora se está en condiciones de efectuar la conversión de las instrucciones en mnemónicos a las instrucciones de máquina, existen dos pasos que se deben efectuar para ello:

- Validar la sintaxis de la instrucción.

- Armar el código objeto.

IV.2 Identificación de los módulos principales.

Con las etapas obtenidas al especificar el problema parece natural que el sistema tenga tantos módulos como los pasos que se mencionaron, de manera que un diagrama como el de la figura 8 cubre todos los pasos. Esta figura se conoce como CARTA DE ESTRUCTURA.

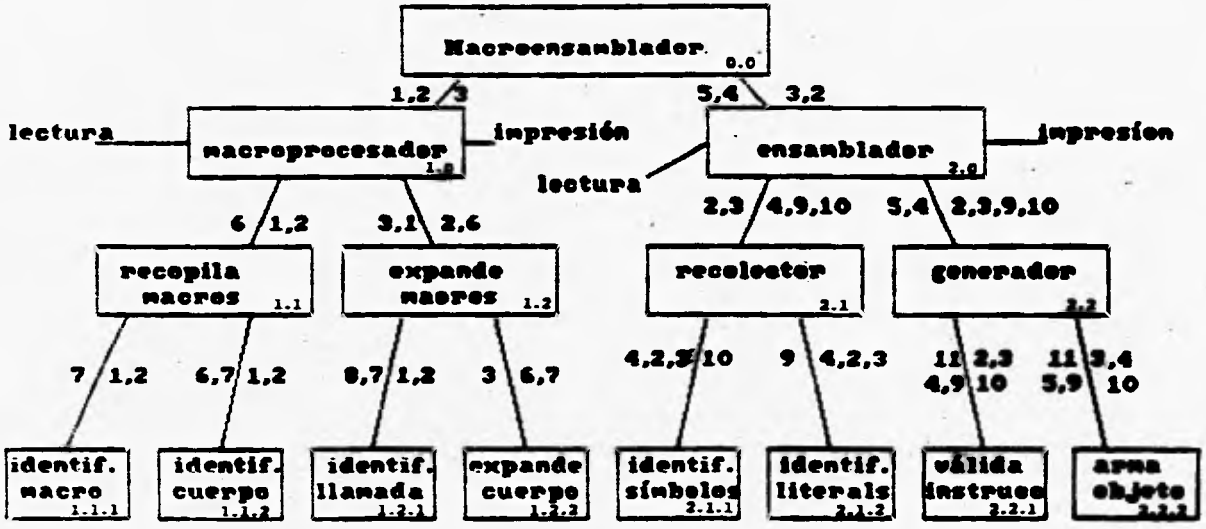

Macroensamblador 0.0
1,2
3
5,4
3,2
lectura
macroprocesador 1.0
impresión
ensamblador 2.0
impresion
lectura
6
1,2
3,1
2,6
2,3
4,9,10
5,4
2,3,9,10
recopila macros 1.1
expande macros 1.2
recolector 2.1
generador 2.2
7
1,2
6,7
1,2
8,7
1,2
3
6,7
4,2,3
10
9
4,2,3
11
4,9
2,3
10
11
5,9
3,4
10
identif. macro 1.1.1
identif. cuerpo 1.1.2
identif. llamada 1.2.1
expande cuerpo 1.2.2
identif. símbolos 2.1.1
identif. literals 2.1.2
válida instruc 2.2.1
arma objeto 2.2.2

FIG. 8

IV.3 Identifición entradas y salidas

Con la identificación de los módulos principales , el siguiente paso en el diseño del sistema consiste en reconocer las entradas y salidas de cada uno de esos módulos, siguiendo un proceso lógico como el llevado a cabo para el desgloce del sistema en sus módulos, (todos los números de entradas y salidas se refieren a la figura 8) tenemos:

-Módulo macroensamblador (0.0)

Este módulo representa al sistema como un todo, de manera que las entradas y salidas son funciones que como objetivo fueron planteadas para el sistema, es decir.

Entradas :
1 : Programa fuente.
2 : Directicas.
3 : Código de la instrucción.

Salidas :
5 : Código objeto.

El macroensamblador deberá efectuar dos funciones básicas:
Proceso de macroinstrucciones y ensamblado de instrucciones simples.

- Módulo de macroprocesador (1.0)

El objetivo del módulo es identificar las macroinstrucciones y sustituir su cuerpo en las llamadas a estas macros que aparescan en el programa.

Entradas:
1 : Programa fuente.
2 : Directivas.

Salidas:
3 : Código expandido.

- Módulo de ensamblador (2.0)

El objetivo de este módulo es traducir las instrucciones simples a instrucciones en lenguaje de máquina. Asignando valores a las literales y símbolos, y sustituyendo estos valores en las instrucciones del programa fuente para formar instrucciones simples.

Entradas:
3 : Programa expandido.
2 : Directivas.
4 : Código de la instrucción.

Salida:
5 : Código objeto.

El nivel siguiente de módulos separa los objetivos de su módulo predecesor conforme a funciones claramente definidas:

-Módulo que recopila macrodefinición (1.1)
El propósito de este módulo es obtener la definición de la macroinstrucción y almacenarla junto con su cuerpo para el caso en que sea llamada en el programa.

Entradas.
1 : programa fuente.
2 : directivas.

Salidas:
6 : Tabla de macroinstrucciones recopiladas.

- Módulo expande macrollamadas (1.2)
El propósito de este módulo es identificar las llamadas a macroinstrucciones, verificar que se encuentren en la tabla obtenida por el módulo recopila macrodefinición y sustituirla por el cuerpo de macroinstrucción en caso que sea válida.

Entradas:
1 : programa fuente.
2 : Directivas.
6 : Tabla de macroinstrucciones recopiladas.
Salidas:
3 : Código expandido.

-Módulo recolector (2.1)
El objetivo de este módulo es identificar símbolos y literales, asignarles valores y almacenarlos en una tabla para futuras referencias. El valor que tome una literal o símbolo puede quedar fijado por la seudoinstrucción que lo declare o por la posición del símbolo en el programa (aquí hablamos de etiquetas).
Entradas:
3 : Código expandido.
2 : Directivas.

Salidas.
9 : Tabla de literales.
10 : Tabla de símbolos.

- Módulo generador (2.2)

El objetivo de este módulo es generar la instrucción de máquina correspondiente a la instrucción del código expandido, para lograr generarla, verificará que la sintaxis de la instrucción sea válida.

Entradas:
2 : Directivas.
3 : Código expandido.
9 : Tabla de literales.
10 : Tabla de símbolos.
4 : Código de instrucción.

Salidas:
5 Código objeto.

El nivel más bajo de diseño de módulos, es digamos, el nivel de programación inmediata, cada uno de estos módulos realiza funciones simples y perfectamente definidas.

-Módulo de identificación de macrodefinición (1.1.1)

Esta módulo buscará en el programa fuente la declaración de alguna macrodefinición, validará su sintaxis y procesará directivas correspondientes.

Entradas:
1 : Programa fuente.
2 : Directivas.

Salidas:
7 : Nombre de la llamada.

-Módulo identifica cuerpo de la macroinstrucción (1.1.2)

con el nombre de la macro, el módulo de identificación del cuerpo de la macroinstrucción abrirá una entrada en la tabla de macros para almacenar el conjunto de instrucciones agrupadas por esta macro, verificará que el armado de macro sea correcto y procesará algunas directivas.

Entradas:
1 : Programa fuente.
2 : Directivas.
7 : Nombre de la macro.

Salida:
6 : Tabla de macroinstrucciones.

-Módulo que identifica llamada a macro (1.2.1)

El módulo tomará el campo de código de operación de la instrucción y buscará en la tabla de definición de macroinstrucciones, en caso de encontrarlo , le dirá a su módulo predecesor que existe una llamada válida y que deberá proseguir a expandirla; si no , guardará la instrucción en el programa expandido.

Entradas:
1 : Programa fuente.
2 : Directivas.

Salidas:
7 : Nombre de la macro.
8 : LLamada válida.

- Módulo que expande el cuerpo de la macro (1.2.2)

El módulo tomará cada instrucción que forma a la macroinstrucción y la almacenará en el código expandido, así como su nombre.

Entradas:
6 : Tabla de macroinstrucciones.
7 : Nombres de la macros.

Salidas:
3 : Código expandido.

- Módulo identifica símbolos (2.1.1)

El programa fuente expandido es leído instrucción por instrucción, se toma el campo de código de operación y se analiza si corresponde a un mnemónico válido de operación, en caso de no ser válido , se almacena en la tabla de símbolos junto con el valor que le corresponde, el valor será la localidad de memoria contenida en el contador de localidades hasta ese momento.

Entradas:
2 : Directivas.
3 : Código expandido.
4 : Código de instrucción.

Salidas:
10 : Tabla de símbolos.

- Módulo que identifica literales (2.1.2)

El programa fuente expandido es leído instrucción por instrucción, se toma el código de operación, si este código representa una instrucción para el macroensamblador, entonces, se toma el valor del contador de localidades y se almacena este valor junto con la identificación de la literal en la tabla de literales.

Entradas:

3 : Código expandido.
2 : Directivas.
4 : Código de la instrucción.

Salidas:
9 : Tabla de literales.

- Módulo de validar instrucción (2.2.1)

El propósito de este módulo es verificar la sintaxis de la instrucción en términos del código de operación y modos de direccionamiento.

Entradas:
2 : Directivas.
3 : Código expandido.
4 : Código de la instrucción.
9 : Tabla de símbolos.
10: Tabla de literales.

Salida:
11 : Instrucción válida.

- Módulo arma objeto (2.2.2)

Después de haber verificado la sintaxis de la instrucción, el siguiente paso es traducirla a lenguaje de máquina. El módulo arma objeto toma la instrucción del programa fuente expandido, la busca en la tabla de código de la instrucción , la ensambla y la almacena en el programa objeto.

Entradas:
3 : Código expandido.
4 : Código de instrucción.
9 : Tabla de literales.
10: Tabla de símbolos.
11: Instrucción válida.

Salida:
5 : Código objeto:

IV.4 Especificación de Estructuras de datos

1. Programa fuente.
Contiene el conjunto de instrucciones que serán ensambladas, las instrucciones pueden ser instrucciones para el microprocesador 68000 o directivas.

2. Tabla de nombres de macros.
Donde se almacenarán los nombres de las macros.

3. Tabla de argumentos de macros.
Donde se almacenarán los nombres de los argumentos de la macro. Consta de dos partes, la primera conserva el nombre del argumento y la segunda, contiene un número consecutivo para el argumento.

4. Tabla de cuerpo de macro.
Contiene cada una de las instrucciones que forman a la macroinstrucción, los argumentos en cada instrucción son sustituidos por números asignados en la tabla de argumentos de macros.

5. Programa expandido.
Es el programa fuente pero eliminadas las declaraciones de macroinstrucción y sustituidas las llamadas a macro por sus instrucciones correspondientes.

6. Tabla de símbolos.
Es usada para almacenar cada etiqueta y su valor correspondiente.

7. Tabla de literales.
Es usada para almacenar cada literal encontrada y su correspondiente localización.

8. Tabla de códigos de la instrucción.
Contiene información que identifica el código de operación de la instrucción, la sintaxis válida, la longitud de la instrucción y el código de máquina correspondiente.

9. Tablas de validación de instrucción.
Es usada para validar los modos de direccionamiento.

10. Tabla de dirección efectiva.
Es usada para obtener la dirección efectiva de cada uno de los modos de direccionamiento.

CAPITULO V

PROGRAMACION

V PROGRAMACION

El siguiente capítulo contiene lo que se denomina seudocódigo, que son los procedimientos que sigue el programa paso a pasó, escritos en lenguaje estructurado incluyen la referencia numérica que guardan con los diagramas de las etapas anteriores de análisis y diseño, posteriormente se encuentran los programas escritos en lenguaje Pascal.

'M A C R O E N S A M B L A D O R (0.0)

MENU: 1. Macroprocesador

2. Ensamblador

3. Salir del programa

Sí la opción es un 1 haz

Pasa al bloque de macroprocesador

Sí la opción es un 2 haz

Pasa al bloque de ensamblador

Fin

Fin

MACROPROCESADOR (1.0)

MENU? 1. Recopilar macrodefiniciones

2. Expandir macrollamadas

3. Regresar al menú anterior

Sí la opción es 1 ejecuta

Ve al bloque de identificar macrodefinición y ejecuta.

Ve al bloque de identificar cuerpo macro y ejecuta

Regresa al menú anterior

Sí no pregunta sí es un 2 y ejecuta

Ve al bloque identificar llamada y ejecuta

Va al bloque expande cuerpo y ejecuta

Pregunta si quiere ud. ensamblar

Ve al menú de ensamblador

Regresa al menú anterior

Sí no regresa al menú de macroensamblador.

IDENTIFICAR MACRODEFINICION (1.1.1)

```
Mientras haya instrucciones ejecuta
  Lee instrucción
  Busca primer carácter diferente a blanco
  Sí el primer carácter es punto
    Lee siguiente carácter hasta que sea diferente de blanco
    Sí cadena restante de la instrucción es igual a MACRO
      Guarda el nombre en la tabla de MACRO
      Guarda número de argumentos y argumentos
      LLama a identificar cuerpo MACRO
    Fin
  Fin
 Fin
```

IDENTIFICAR CUERPO DE MACRO (1.1.2)

```
Mientras haya instrucciones o no encuentres MEND
 Lee instrucción
 Busca primer carácter diferente de blanco
 Sí el primer carácter es diferente de punto
    Sí hay argumentos
      Reemplaza argumentos por índices
    Fin
    Guarda instrucción en tabla de cuerpos de MACRO
   Si no
    Busca primer carácter diferente a blanco
      Sí cadena restante es igual a MEND
        Indica que encontraste MEND
      Fin
   Fin
 Fin
```

IDENTIFICAR LLAMADAS (1.2.1)

```
Mientras haya instrucción ejecuta
Lee instrucción
Separa por campos y toma código de operación
Mientras haya elementos en la tabla de MACROS ejecuta
 Sí código de operación es igual a nombre de MACRO
     Marca encontré
     LLama a expandir cuerpo
 Si no
   Lee siguiente elemento de tabla de MACROS
 Fin
Si no lo encontré
   Graba la instrucción en el archivo expandído
Fin
```

EXPANDIR CUERPO (1.2.2)

```
Mientras haya instrucciones de la MACRO ejecuta
  Lee instrucción
  Sustituye argumentos de llamada
  Graba instrucción en código expandído
Fin
```

E N S A M B L A D O R (2.0)

MENU: 1. Recolector

2. Generador

3. Regresa al menú anterior(principal)

Sí le opción es un 1 y ejecuta

Ve al bloque identificar símbolos y ejecuta

Ve al bloque identificar literales y ejecuta

Regresa al menú anterior

Sí no pregunta si es un 2 y ejecuta

Ve al bloque válida instrucción y ejecuta

Ve al bloque arma directiva y ejecuta

Ve al bloque arma instrucción y ejecuta

Muestra al programa ya traducido a código máquina

Regresa al menú principal.

IDENTIFICAR SIMBOLOS Y LITERALES (2.1.1) y (2.1.2)

```
Mientras haya instrucciones en el código expandído
  Leer instrucción
  Separa por campos y obten código de operación
  Mientras haya mnemónicos de instrucción a comparar ejecuta
    Lee mnemónico
    Sí código de operación es igual a mnemónico
      Indica que se trata de una instrucción
      No busques más en table de mnemónicos
    Fin
  Fin
  Sí no es una instrucción entonces
    Mientras haya directivas a comparar haz
      Lee directiva
      Sí código de operación ss igual a directiva
        Indica que se trata de una directiva
        No busques más en tabla de directivas
      Fin
    Fin
  Fin
  Sí no es instrucción ni directiva
    Guarda código de operación encontrado en tabla de Símb.
  Fin
Fin
```

VALIDA INSTRUCCION (2.2.1)

```
Mientras haya instrucciones o errores menor al máximo
  Leer instrucción
  Separa por campos y obten el código de operación
  LLama a ARMAR DIRECTIVAS
  Sí no es directiva
     LLama a ARMAR INSTRUCCIONES
  Fin
Fin
```

ARMA DIRECTIVA (2.2.2)

```
Mientras haya directivas en la tabla de directivas haz
  Lee directiva
  Sí el código de operación es igual a la directiva
    Valida sintaxis de la directiva
    Sí hay error
      Guarda instrucción junto con su error en la salida
      Incrementa número de errores
    Si no
      Ejecuta directiva
      Incrementa contador de localidades
    Fin
  Fin
Fin
```

ARMA INSTRUCCION (2.2.2)

```
Mientras haya mnemónicos de instrucción ejecuta
   Lee mnemónico de instrucción
   Sí el código de operación es igual al mnemónico
     Valída sintaxis de la instrucción
     Sí hay error
       Guarda instrucción junto con su error en salida
       Incrementa número de errores
     Si no
       Genera código de la instrucción
       Incrementa contador de localidades
       Indica encontrado
     Fin
    Fin
Fin
```

PROGRAMA ENSAMBLE

El siguiente listado contiene la parte del sistema que ensambla las instrucciones para el microprocesador y que contiene los siguientes procedimientos :

a) Elimina exceso de blancos y compacta la instrucción.

b) Separa por palabras la instrucción actual analizada.

c) Guarda palabras que forman la instrucción.

d) Creación del diagrama de estados para los modos de direccionamiento.

e) Crea las tablas de dirección efectiva.

f) Crea la dirección efectiva dependiendo del modo de direccionamiento y del registro usado.

g) Se mueve a lo largo del autómata para identificar el modo de direccionamiento del operando.

h) Verifica la sintaxis de la instrucción, generando las posibles combinaciones de modos de direccionamiento.

i) Verifica que los modos de direccionamiento usados en la instrucción sean válidos.

j) Ensambla la instrucción a analizar.

k) Sustituye en el código el número de registro de datos usado.

l) Sustituye el registro de datos como destino.

m) Sustituye en el código el número de registro de direc-- ción usado.

n) Sustituye en el código el número de registro de direc-- ción usado como destino.

ñ) Sustituye el código de la dirección efectiva del operando fuente.

o) Sustituye el código de la dirección efectiva del operando destino.

p) Sustituye el código binario de la instrucción por código hexadecimal.

```
PROGRAM ENSAMBLE(INPUT,OUTPUT,FUENTE,CODIGO);
 CONST
  LONGBIN=        16;
  LONGMODO=       36;
  LEDOS=         ,35;
  LONGCOD=        55;
  LCARAC=         15;
  LONGNOM=        12;
  LIMINST=        55;
  LIMCTD=          7;
  LONGSEP=         3;
 TYPE
  WORDS=          PACKED ARRAY [1..LIMCTD, 1..LONGNOM ] OF CHAR;
  WORD=           PACKED ARRAY [1..LONGNOM] OF CHAR;
  INSTRUC=        PACKED ARRAY [1..LIMINST] OF CHAR;
  DIRECCION=      PACKED ARRAY [1..6] OF CHAR;
  ADIRECCION=     PACKED ARRAY [1..2, 1..6] OF CHAR;
  TMODO=          PACKED ARRAY [1..12,1..3] OF CHAR;
  TREGISTER=      PACKED ARRAY [1..13,1..3] OF CHAR;
  FORMATO=        PACKED ARRAY [1..2 , 1..LONGMODO] OF CHAR;
  TBINARIO=       PACKED ARRAY [1..2, 1..LONGBIN] OF CHAR;
  TCMAQUINA=      PACKED ARRAY [1..LONGBIN] OF CHAR;
  TINMEDIATO=     PACKED ARRAY [1..4,1..4] OF CHAR;
  TCMAQHEX=       PACKED ARRAY [1..4] OF CHAR;
 VAR
  FUENTE,
  CODIGO:       TEXT;
  LINEAIN,LINEAOUT:       INSTRUC;
  I,J,K,II,LONG:          INTEGER;
  NUMREG,NUMPBLE:         INTEGER;
  ERROR:                  INTEGER;
  MODO,NUMWORDS:          INTEGER;
  NUMBINARIO:             INTEGER;
  PALABRAS:               WORDS;
  PALABRA:                WORD;
  ADICIONAL:          PACKED ARRAY [1..LIMCTD, 1..LONGSEP] OF CHAR;
  TABAUTOM:           ARRAY [1..LEDOS, 1..LCARAC] OF INTEGER;
  TABCARAC:           ARRAY [1..LCARAC] OF CHAR;
  DIREFEC:            DIRECCION;
  ADIREFEC:           ADIRECCION;
  TABMODO:            TMODO;
  TABREG:             TREGISTER;
  POSIBILIDADES:      FORMATO;
  AMODO:              ARRAY[1..2] OF INTEGER;
  ABINARIO:           TBINARIO;
  CMAQUINA:           TCMAQUINA;
  CINMEDIATO:         TINMEDIATO;
  CMAQHEX:            TCMAQHEX;
  TIPOINSTRUC:        CHAR;
  APUNTEXT:           INTEGER;
  {*****************************************************}
  {*                                                   *}
  {*          ELIMINA EXCESO DE BLANCOS                *}
  {*          Y COMPACTA LA INSTRUCCION                *}
  {*                                                   *}
```

```
{*********************************************************}
PROCEDURE QUITABLANCOS (VAR LONG: INTEGER;VAR LINEA:INSTRUC;
                        VAR LINEA2:-INSTRUC );
CONST
 BLANCO=        ' ';
VAR
 SEPARADOR:  SET OF CHAR;
 I,J,K,L,H:    INTEGER;
 CARACTER:   CHAR;
BEGIN
 SEPARADOR:= [ ]; SEPARADOR:=SEPARADOR + ['#'];
 SEPARADOR:= SEPARADOR + ['(']; SEPARADOR:= SEPARADOR + ['$'];
 SEPARADOR:= SEPARADOR + [')']; SEPARADOR:= SEPARADOR + ['&'];
 SEPARADOR:= SEPARADOR + [','];
 J:=0; K:=0; L:=0; H:=0;
 FOR I:=1 TO LONG DO
   BEGIN
     CARACTER:= LINEA[I];
     IF CARACTER IN SEPARADOR THEN
      H:=1;
    IF CARACTER <> BLANCO THEN
     BEGIN
      J:= J+1;
      LINEA2[J]:=LINEA[I];
      K:=0; L:=1;
     END
    ELSE
     BEGIN
      IF (K = 0) AND (L = 1) AND (H = 0) THEN
       BEGIN
        J:=J+1;
        LINEA2[J]:=LINEA[I];
        K:=1;
       END;
      END;
    END;
    LONG:=J;
 END;{*QUITABLANCOS*}
{*****************************************************************}
{*                                                               *}
{* SEPARA POR PALABRAS LA INSTRUCCION ACTUAL ANALIZADA           *}
{*                                                               *}
{*****************************************************************}
  PROCEDURE PORWORDS( VAR PALABRAS: WORDS; LINEA: INSTRUC;
                      VAR NUMWORDS: INTEGER);
  CONST
  BLANCO= ' ';
  PUNTOCOMA= ';';
 VAR
  CARACTER:      CHAR;
  I,J,K,W,WW:    INTEGER;
  SEPARADOR:     SET OF CHAR;
{*                                                               *}
{*      GUARDA PALABRAS QUE FORMAN LA INSTRUCCION                *}
{*                                                               *}
```

```
  PROCEDURE GUARDAWORD;
  BEGIN
   K:= K+1;
   J:=0; WW:=0;
   REPEAT
    J:= J+1;
    PALABRAS[K,J]:= LINEA[I];
    I:= I+1
   UNTIL ( LINEA[I] IN SEPARADOR ) OR ( I > LONG );
   I:= I-1; (* PARA APUNTAR AL ULTIMO CARACTER VISTO *)
  END;
 {*                                                              *)
  BEGIN
    SEPARADOR:=[ ]; SEPARADOR:= SEPARADOR + [' '];
    SEPARADOR:=SEPARADOR + [','); SEPARADOR:= SEPARADOR + [';'];
    {*SEPARA POR PALABRAS*}
    FOR K:=1 TO LIMCTD DO
     BEGIN
      FOR I:=1 TO LONGSEP DO
        ADICIONAL[K,I]:= BLANCO;
      FOR I:=1 TO LONGMOM DO
        PALABRAS[K,I]:= BLANCO;
      END;
     K:=0 ; I:=0; WW:=0;
     WHILE (K <= LIMCTD) AND ( I < LONG ) DO
      BEGIN
       I:= I+1;
       CARACTER:= LINEA[I];
       IF  CARACTER IN SEPARADOR THEN
        BEGIN
         WW:= WW+1;
         ADICIONAL[K,WW]:= CARACTER;
        END
       ELSE
           GUARDAWORD;
      END;
      NUMWORDS:=K;
  END; {PORWORDS}
{*****************************************************************}
{*                                                               *}
{*   CREACION DEL DIAGRAMA DE ESTADOS PARA LOS MODOS DE          *}
{*   DE DIRECCIONAMIENTO. LOS MODOS CONTEMPLADOS SON :           *}
{*                                                               *}
{*          1. DIRECTO A REGISTRO DE DATOS.                      *}
{*          2. DIRECTO A REGISTRO DE DIRECCION.                  *}
{*          3. INDIRECTO A REGISTRO DE DIRECCION.                *}
{*          4. INDIRECTO A REGISTRO DE DIRECCION CON +           *}
{*          5. INDIRECTO A REGISTRO DE DIRECCION CON -           *}
{*          6. INDIRECTO CON DESPLAZAMIENTO.                     *}
{*          7. INDIRECTO CON EL PC Y DESPLAZAMIENTO              *}
{*         12. INMEDIATO.                                        *}
{*          9. ABSOLUTO CORTO.                                   *}
{*         10. ABSOLUTO LARGO.                                   *}
{*****************************************************************}
 PROCEDURE AUTOMATA;
```

```
VAR
 I,J:       INTEGER;
BEGIN
 TABCARAC[1]:= ' '; TABCARAC[2]:= 'D'; TABCARAC[3]:= 'A';
 TABCARAC[4]:= 'X'; TABCARAC[5]:= 'Y'; TABCARAC[6]:= ',';
 TABCARAC[7]:= '('; TABCARAC[8]:= ')'; TABCARAC[9]:= '+';
 TABCARAC[10]:= '-'; TABCARAC[11]:='@'; TABCARAC[12]:= '#';
 TABCARAC[13]:= '$';
{*                                                              *}
  FOR I:=1 TO LEDOS DO
   FOR J:=1 TO LCARAC DO
    TABAUTOM[I,J]:= 35;
{*                                                              *}
  TABAUTOM[1,1]:= 1; TABAUTOM[1,2]:= 2; TABAUTOM[1,3]:=5;
  TABAUTOM[1,4]:= 16; TABAUTOM[1,5]:= 16;
  TABAUTOM[1,7]:= 8; TABAUTOM[1,10]:=15; TABAUTOM[1,11]:=16;
  TABAUTOM[1,12]:=17; TABAUTOM[1,13]:=20; TABAUTOM[2,4]:=3;
  TABAUTOM[2,5]:=3; TABAUTOM[2,11]:=3;
  TABAUTOM[3,1]:= 4; TABAUTOM[3,6]:= 4; TABAUTOM[5,5]:=6;
  TABAUTOM[5,4]:= 6; TABAUTOM[5,11]:= 6;
  TABAUTOM[6,1]:= 7; TABAUTOM[6,6]:= 7; TABAUTOM[8,1]:=8;
  TABAUTOM[8,3]:= 9; TABAUTOM[9,5]:=10;
  TABAUTOM[9,4]:=10; TABAUTOM[9,11]:=10;TABAUTOM[10,1]:=10;
  TABAUTOM[10,8]:=11; TABAUTOM[11,1]:=12; TABAUTOM[11,6]:=12;
  TABAUTOM[11,9]:=13; TABAUTOM[13,1]:=14; TABAUTOM[13,6]:=14;
  TABAUTOM[15,7]:= 8; TABAUTOM[16,7]:= 8; TABAUTOM[16,11]:=16;
  TABAUTOM[16,4]:= 16; TABAUTOM[16,5]:=16;
  TABAUTOM[17,1]:=17; TABAUTOM[17,11]:=18;
  TABAUTOM[17,4]:=18; TABAUTOM[17,5]:=18; TABAUTOM[18,1]:=19;
  TABAUTOM[18,6]:=19; TABAUTOM[18,11]:=18;
  TABAUTOM[18,4]:=18; TABAUTOM[18,5]:= 18;TABAUTOM[20,1]:=20;
  TABAUTOM[20,4]:=21; TABAUTOM[20,5]:= 21;
  TABAUTOM[20,11]:=21; TABAUTOM[21,1]:=25; TABAUTOM[21,6]:=25;
  TABAUTOM[21,4]:= 22; TABAUTOM[21,5]:= 22;
  TABAUTOM[21,11]:=22; TABAUTOM[22,1]:=25; TABAUTOM[22,6]:=25;
  TABAUTOM[22,4]:=23; TABAUTOM[22,5]:= 23;
  TABAUTOM[22,11]:=23; TABAUTOM[23,1]:=25; TABAUTOM[23,6]:=25;
  TABAUTOM[23,4]:=24; TABAUTOM[23,5]:= 24;
  TABAUTOM[23,11]:=24; TABAUTOM[24,1]:=25; TABAUTOM[24,6]:=25;
  TABAUTOM[24,4]:=26; TABAUTOM[24,5]:= 26;
  TABAUTOM[24,11]:=26; TABAUTOM[26,1]:=30; TABAUTOM[26,6]:=30;
  TABAUTOM[26,4]:= 27; TABAUTOM[26,5]:= 27;
  TABAUTOM[26,11]:=27; TABAUTOM[27,1]:=30; TABAUTOM[27,6]:=30;
  TABAUTOM[27,4]:= 28; TABAUTOM[27,5]:= 28;
  TABAUTOM[27,11]:=28; TABAUTOM[28,1]:=30; TABAUTOM[28,6]:=30;
  TABAUTOM[28,4]:= 29; TABAUTOM[28,5]:= 29;
  TABAUTOM[28,11]:=29; TABAUTOM[29,1]:=30; TABAUTOM[29,6]:=30;
END; {* LLENA AUTOMATA *}
{*                                                              *}
{**************************************************************}
{*                                                              *}
{*   CREA LAS TABLAS DE DIRECCION EFECTIVA CONFORME AL LIBRO    *}
{*   MANUAL DEL MICROPROCESADOR 68000.                          *}
{*                                                              *}
{**************************************************************}
```

```
{*                                                                    *)
 PROCEDURE CREAEFEC;
 BEGIN
  TABMODO[1,1]:='0';TABMODO[1,2]:='0';TABMODO[1,3]:='0';
  TABMODO[2,1]:='0';TABMODO[2,2]:='0';TABMODO[2,3]:='1';
  TABMODO[3,1]:='0';TABMODO[3,2]:='1';TABMODO[3,3]:='0';
  TABMODO[4,1]:='0';TABMODO[4,2]:='1';TABMODO[4,3]:='1';
  TABMODO[5,1]:='1';TABMODO[5,2]:='0';TABMODO[5,3]:='0';
  TABMODO[6,1]:='1';TABMODO[6,2]:='0';TABMODO[6,3]:='1';
  TABMODO[7,1]:='1';TABMODO[7,2]:='1';TABMODO[7,3]:='0';
  TABMODO[8,1]:='1';TABMODO[8,2]:='1';TABMODO[8,3]:='1';
  TABMODO[9,1]:='1';TABMODO[9,2]:='1';TABMODO[9,3]:='1';
  TABMODO[10,1]:='1';TABMODO[10,2]:='1';TABMODO[10,3]:='1';
  TABMODO[11,1]:='1';TABMODO[11,2]:='1';TABMODO[11,3]:='1';
  TABMODO[12,1]:='1';TABMODO[12,2]:='1';TABMODO[12,3]:='1';
  TABREG[1,1]:='0';TABREG[1,2]:='0';TABREG[1,3]:='0';
  TABREG[2,1]:='0';TABREG[2,2]:='0';TABREG[2,3]:='1';
  TABREG[3,1]:='0';TABREG[3,2]:='1';TABREG[3,3]:='0';
  TABREG[4,1]:='0';TABREG[4,2]:='1';TABREG[4,3]:='1';
  TABREG[5,1]:='1';TABREG[5,2]:='0';TABREG[5,3]:='0';
  TABREG[6,1]:='1';TABREG[6,2]:='0';TABREG[6,3]:='1';
  TABREG[7,1]:='1';TABREG[7,2]:='1';TABREG[7,3]:='0';
  TABREG[8,1]:='1';TABREG[8,2]:='1';TABREG[8,3]:='1';
  TABREG[9,1]:='0';TABREG[9,2]:='0';TABREG[9,3]:='0';
  TABREG[10,1]:='0';TABREG[10,2]:='0';TABREG[10,3]:='1';
  TABREG[11,1]:='0';TABREG[11,2]:='1';TABREG[11,3]:='1';
  TABREG[12,1]:='0';TABREG[12,2]:='1';TABREG[12,3]:='1';
  TABREG[13,1]:='1';TABREG[13,2]:='0';TABREG[13,3]:='0';
 END; (* CREAEFEC *)
{*                                                                    *)
{**********************************************************************}
{*                                                                    *)
{*     CREA LA DIRECCION EFECTIVA DEPENDIENDO DEL MODO                *)
{*     DE DIRECCIONAMIENTO Y DEL REGISTRO USADO.                      *)
{*                                                                    *)
{**********************************************************************}
 PROCEDURE EFECTIVA(OPERANDO:WORD; VAR MODO: INTEGER;
           VAR DIREFEC: DIRECCION; NUMREG: INTEGER);
 VAR
  FLAG:   BOOLEAN;
  I:      INTEGER;
 BEGIN
  FLAG:= FALSE;
  IF NUMREG = 99 THEN (* MODO DE DIR NO USA REGISTROS *)
   FLAG:= TRUE;
  IF FLAG  THEN
   BEGIN
     FOR I:=1 TO 3 DO
      BEGIN
       DIREFEC[I]:= TABMODO[MODO,I];
       DIREFEC[I+3]:= TABREG[MODO+1,I];
      END;
   END
  ELSE
    BEGIN
```

```
    FOR I:=1 TO 3 DO
     BEGIN
      DIREFEC[I]:= TABMODO[MODO,I];
      DIREFEC[I+3]:= TABREG[NUMREG+1,I];
     END;
   END;
END; {* EFECTIVA *}
{*                                                                    *}
{**********************************************************************}
{*                                                                    *}
{*   SE MUEVE A LO LARGO DEL AUTOMATA PARA IDENTIFICAR                *}
{*   EL MODO DE DIRECCIONAMIENTO DEL OPERANDO.                        *}
{*                                                                    *}
{**********************************************************************}
{*                                                                    *}
PROCEDURE VIAJA(OPERANDO: WORD; VAR MODO,NUMREG: INTEGER;
                TIPO:CHAR;VAR APUNTEXT: INTEGER);
CONST
 EDOERROR=     35;
VAR
 I,INDICE,J,II,WW,JJ:  INTEGER;
 AUX:                  INTEGER;
 EDO,VEDO,SEGEDO:      INTEGER;
 CONTEDO:              INTEGER;
 DATO:                 CHAR;
 ERROR:                BOOLEAN;
BEGIN
 I:=0;VEDO:=1; ERROR:= FALSE;
 CONTEDO:=0;MODO:=0;NUMREG:=99;
 REPEAT
  I:=I+1; INDICE:= 0;
  EDO:= VEDO; CONTEDO:=CONTEDO+1;
  II:=0;
  DATO:= OPERANDO[I];
  FOR J:=1 TO LCARAC DO
   BEGIN
    IF TABCARAC[J] = DATO THEN
       INDICE:= J;
   END;
  IF INDICE = 0 THEN (* SE TRATA DE UN CARACTER NUMERICO *)
   BEGIN
    IF ( ORD(DATO) >= 48 ) AND ( ORD(DATO) <= 54) THEN
     BEGIN
      INDICE:= 5;
      NUMREG:= ORD(DATO)-48;
     END
    ELSE
     IF ( ORD(DATO) = 55 ) THEN
      BEGIN
       INDICE:= 4;
       NUMREG:= ORD(DATO)-48;
      END
     ELSE
      INDICE:= 11;
   END;
```

```
VEDO:= TABAUTOM[EDO,INDICE];
IF CONTEDO = 1 THEN
   SEGEDO:= VEDO;
IF VEDO = EDOERROR THEN
   . ERROR:= TRUE;
CASE VEDO OF
   4: MODO:= 1 ;
   7: MODO:= 2 ;
  12: CASE SEGEDO OF
         8: MODO:= 3;
        15: MODO:= 5;
        16: MODO:= 6;
       ELSE
         ERROR:= TRUE;
      END;
  14: MODO:= 4;
  19: BEGIN
       IF TIPO = '2' THEN (* INTRUCCION LARGA *)
          BEGIN
            II:=LONGNOM; AUX:=APUNTEXT+2; WW:=5;
            REPEAT
              IF OPERANDO[II] <> ' ' THEN
                 BEGIN
                   WW:=WW-1;
                   CINMEDIATO[AUX,WW]:= OPERANDO[II];
                 END;
                 II:= II-1
            UNTIL (WW = 1) OR (II = 1);
            IF II > 1 THEN
               BEGIN
                 AUX:=APUNTEXT+1; WW:=5;
                 REPEAT
                  IF OPERANDO[II] <> ' ' THEN
                   BEGIN
                    WW:=WW-1;
                    CINMEDIATO[AUX,WW]:= OPERANDO[II];
                   END;
                   II:= II-1;
                  UNTIL (WW = 1 ) OR (II = 1);
               END;
           APUNTEXT:= APUNTEXT+2;
          END;
        IF TIPO = '1' THEN
           BEGIN
             II:=LONGNOM;
             APUNTEXT:=APUNTEXT+1; WW:=5;
             REPEAT
               IF OPERANDO[II] <> ' ' THEN
                  BEGIN
                    WW:=WW-1;
                    CINMEDIATO[APUNTEXT,WW]:= OPERANDO[II];
                  END;
                  II:= II-1;
              UNTIL (WW = 1 ) OR (II = 1);
           END;
```

```
            MODO:=12;
            NUMREG:=99;
          END;
      25: BEGIN
           APUNTEXT:=APUNTEXT+1;
           WW:=5; II:= LONGNOM;
           REPEAT
            IF OPERANDO[II] <> ' ' THEN
               BEGIN
                 WW:=WW-1;
                 CINMEDIATO[APUNTEXT,WW]:= OPERANDO[II];
               END;
              II:= II-1;
           UNTIL (WW = 1 ) OR (II = 1);
           MODO:= 8;
           NUMREG:=99;
          END;
      30: BEGIN
           II:=LONGNOM; APUNTEXT:=APUNTEXT+2; WW:=5;
           REPEAT
             IF OPERANDO[II] <> ' ' THEN
                BEGIN
                  WW:=WW-1;
                  CINMEDIATO[APUNTEXT,WW]:= OPERANDO[II];
                END;
                II:= II-1
            UNTIL (WW = 1) OR (II = 1);
            IF II > 1 THEN
               BEGIN
                 APUNTEXT:=APUNTEXT+1; WW:=5;
                 REPEAT
                   IF OPERANDO[II] <> ' ' THEN
                      BEGIN
                         WW:=WW-1;
                         CINMEDIATO[APUNTEXT,WW]:= OPERANDO[II];
                       END;
                   II:= II-1;
                  UNTIL (WW = 1 ) OR (II = 1);
               END;
            MODO:=9;
            NUMREG:=99;
           END
       ELSE;
     END
   UNTIL (ERROR= TRUE) OR ( MODO <> 0 );
  END; {* VIAJA *}
{*****************************************************************}
{*                                                               *}
{*      VERIFICA SINTAXIS DE LA INSTRUCCION, GENERANDO LAS       *}
{*      POSIBLES COMBINACIONES DE LOS MODOS DE DIRECCIONAMIENTO. *}
{*                                                               *}
{*****************************************************************}
{*                                                               *}
 PROCEDURE OBTENCODIGO( INSTRUCCION:WORD;VAR POSIBILIDAD: FORMATO;
                        VAR ERROR: INTEGER; VAR NUMPBLE:INTEGER;
```

```
                        VAR ABINARIO: TBINARIO; VAR TIPO: CHAR);
  VAR
   H,I,W,WW:     INTEGER;
   LINEAIN:      INSTRUC;
  BEGIN
   ASSIGN( CODIGO, 'INSTRUC.DAT');
   RESET(CODIGO);
   W:=0;
   WHILE ( NOT EOF(CODIGO))  DO
    BEGIN
     I:=0; H:=0;WW:=0;
     REPEAT
      I:= I+1;
      READ(CODIGO,LINEAIN[I])
     UNTIL I=LONGCOD;
     FOR I:=1 TO LONGMOM DO
      BEGIN
       IF LINEAIN[I] = INSTRUCCION[I] THEN
          H:= H+1;
      END;
     IF H = LONGMOM THEN
      BEGIN
        W:=W+1;
        FOR J:=13 TO LONGMODO DO
         BEGIN
          WW:= WW+1;
          POSIBILIDAD[W,WW]:= LINEAIN[J];
         END;
        WW:=0;TIPO:= LINEAIN[38];
        FOR J:=40 TO 55 DO
         BEGIN
          WW:=WW+1;
          ABINARIO[W,WW]:= LINEAIN[J];
         END;
      END;
      READLN(CODIGO);
    END;
    CLOSE(CODIGO);
    IF W = 0 THEN {* NO EXISTE LA INSTRUCCION *}
     ERROR:= 1;
    NUMPBLE:= W;
  END;{* OBTENCODIGO *}
{*************************************************************************}
{*  VERIFICA QUE LOS MODOS DE DIRECCIONAMIENTO USADOS EN LA INSTRUC. *}
{*  SEAN VALIDOS.                                                    *}
{*                                                                   *}
{*************************************************************************}
 PROCEDURE VERMODOS( VAR NUMBIN: INTEGER);
 VAR
  I,J,K,H:         INTEGER;
  AAMODOS:         ARRAY[1..12,1..2] OF INTEGER;
  TIPO:            PACKED ARRAY[1..2] OF CHAR;
  MASOPERANDOS:    BOOLEAN;
  ERROR,VALIDO:    BOOLEAN;
  COLUMNA:         INTEGER;
```

```
 BEGIN
  AAMODOS[1,1]:=1;AAMODOS[1,2]:=2;
  AAMODOS[2,1]:=3;AAMODOS[2,2]:=4;
  AAMODOS[3,1]:=5;AAMODOS[3,2]:=6;
  AAMODOS[4,1]:=7;AAMODOS[4,2]:=8;
  AAMODOS[5,1]:=9;AAMODOS[5,2]:=10;
  AAMODOS[6,1]:=11;AAMODOS[6,2]:=12;
  AAMODOS[7,1]:=13;AAMODOS[7,2]:=14;
  AAMODOS[8,1]:=15;AAMODOS[8,2]:=16;
  AAMODOS[9,1]:=17;AAMODOS[9,2]:=18;
  AAMODOS[10,1]:=19;AAMODOS[10,2]:=20;
  AAMODOS[11,1]:=21;AAMODOS[11,2]:=22;
  AAMODOS[12,1]:=23;AAMODOS[12,2]:=24;
  TIPO[1]:='1';TIPO[2]:='2';
{*
  I:=1;MASOPERANDOS:=TRUE;K:=1;
  ERROR:= FALSE;H:=1;
  WHILE ( I <= 2 ) AND ( MASOPERANDOS = TRUE ) DO
   BEGIN
     J:= AMODO[I];
     IF J <> 99 THEN
     BEGIN
        COLUMNA:= AAMODOS[J,K];
        VALIDO:=FALSE;
        REPEAT
          IF POSIBILIDADES[H,COLUMNA] = TIPO[I] THEN
             BEGIN
               VALIDO:= TRUE;
               NUMBIN:=H;
             END
             ELSE
               H:=H+1;
        UNTIL ( H > NUMPBLE) OR (VALIDO = TRUE);
        IF VALIDO = FALSE THEN
          BEGIN
            ERROR:=TRUE;
            MASOPERANDOS:= FALSE;
          END;
     END
     ELSE
         MASOPERANDOS:= FALSE;
     I:=I+1; K:= K+1;
   END; {* WHILE *}
END; {* VERMODOS *}
{*                                                                          *}
{**************************************************************************}
{*                                                                          *}
{*  ENSAMBLA LA INSTRUCCION A ANALIZAR. TOMA EL FORMATO DE LA               *}
{*  INSTRUCCION DE LA TABLA DE CODIGOS Y EFECTUA CIERTA LOGICA              *}
{*  DEPENDIENDO DE LOS VALORES QUE SE ENCUENTREN EN ESE FORMATO.            *}
{*                                                                          *}
{**************************************************************************}
{*                                                                          *}
 PROCEDURE ENSAMBLA( NUMBINARIO: INTEGER; VAR CMAQUINA: TCMAQUINA;
```

```
                    VAR APUNTEXT: INTEGER);
 VAR
  POINTERCMAQ:     INTEGER;
  CARACTER:        CHAR;
  I,J:             INTEGER;
{*                                                                    *}
{**********************************************************************}
{**  SUSTITUYE EN EL CODIGO EL NUMERO DE REGISTRO DE DATOS USADO **}
{**                                                                  **}
{**********************************************************************}
PROCEDURE DATAREGISTER;
BEGIN
 IF AMODO[1] = 1 THEN
  BEGIN
   POINTERCMAQ:= POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[1,4];
   POINTERCMAQ:= POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[1,5];
   POINTERCMAQ:=POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[1,6];
  END
 ELSE
  BEGIN
   POINTERCMAQ:= POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,4];
   POINTERCMAQ:= POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,5];
   POINTERCMAQ:=POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,6];
  END;
END; {* DATA REGISTER *}
{**********************************************************************}
{***       SUSTITUYE EL DATA REGISTER COMO DESTINO              ***}
{*                                                                    *}
{**********************************************************************}
PROCEDURE DESTDATA;
  BEGIN
   POINTERCMAQ:= POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,4];
   POINTERCMAQ:= POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,5];
   POINTERCMAQ:=POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,6];
  END;{* DESTDATA *}

{*                                                                    *}
{**********************************************************************}
{**  SUSTITUYE EN EL CODIGO EL NUMERO DE REGISTRO DE DIRECCION    **}
{**  USADO.                                                          **}
{**********************************************************************}
PROCEDURE ADDRESSREGISTER;
BEGIN
   POINTERCMAQ:= POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,4];
   POINTERCMAQ:= POINTERCMAQ+1;
```

```
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,5];
   POINTERCMAQ:=POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,6];
END; {* ADDRES REGISTER *}
{*                                                                    *}
{**********************************************************************}
{**  SUSTITUYE EN EL CODIGO EL NUMERO DE REGISTRO DE DIRECCION       **}
{**  USADO COMO DESTINO.                                             **}
{**********************************************************************}
PROCEDURE DESTADDRESS;
BEGIN
   POINTERCMAQ:= POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[1,4];
   POINTERCMAQ:= POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[1,5];
   POINTERCMAQ:=POINTERCMAQ+1;
   CMAQUINA[POINTERCMAQ]:= ADIREFEC[1,6];
END; {* DESTADRESS *}
{*                                                                    *}
{**********************************************************************}
{**  SUSTITUYE EN EL CODIGO EL EFECTIVE ADDRESS DEL OPERANDO         **}
{**                                                                  **}
{**********************************************************************}
PROCEDURE EFECTIVEADDRESS;
VAR
 I: INTEGER;
BEGIN
 IF AMODO[1] = 1 THEN
  BEGIN
   FOR I:=1 TO 6 DO
    BEGIN
      POINTERCMAQ:= POINTERCMAQ+1;
      CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,I];
    END
  END
 ELSE
  BEGIN
   FOR I:=1 TO 6 DO
    BEGIN
      POINTERCMAQ:= POINTERCMAQ+1;
      CMAQUINA[POINTERCMAQ]:= ADIREFEC[1,I];
    END;
  END;
END; {* EFECTIVEADDRESS *}
{*                                                                    *}
{**********************************************************************}
{**  SUSTITUYE EN EL CODIGO EL EFECTIVE ADDRESS DEL OPERANDO         **}
{**  FUENTE.                                                         **}
{**********************************************************************}
PROCEDURE EFECTIVESOURCE;
VAR
 I: INTEGER;
BEGIN
   FOR I:=1 TO 6 DO
    BEGIN
```

```
      POINTERCMAQ:= POINTERCMAQ+1;
      CMAQUINA[POINTERCMAQ]:= ADIREFEC[1,I];
    END;
END; {* EFECTIVE *}
{*                                                                    *}
{**********************************************************************}
{**  SUSTITUYE EN EL CODIGO EL EFECTIVE-ADDRESS DEL OPERANDO         **}
{**  DESTINO.                                                        **}
{**********************************************************************}
PROCEDURE EFECTIVEDESTINO;
VAR
 I: INTEGER;
BEGIN
   FOR I:=1 TO 6 DO
    BEGIN
      POINTERCMAQ:= POINTERCMAQ+1;
      CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,I];
    END;
END; {* EFECTIVEDESTINO *}
{*                                                                    *}
{**********************************************************************}
{**  SUSTITUYE EN EL CODIGO EL EFECTIVE ADDRESS DEL OPERANDO         **}
{**  DESTINO EN FORMA INVERTIDA.                                     **}
{**********************************************************************}
PROCEDURE DESTINOEFECTIVE;
VAR
 I: INTEGER;
BEGIN
   FOR I:=4 TO 6 DO
    BEGIN
      POINTERCMAQ:= POINTERCMAQ+1;
      CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,I];
    END;
   FOR I:=1 TO 3 DO
    BEGIN
      POINTERCMAQ:= POINTERCMAQ+1;
      CMAQUINA[POINTERCMAQ]:= ADIREFEC[2,I];
    END;
END; {* DESTINOEFECTIVE *}
{*                                                                    *}
{**********************************************************************}
{**  SUSTITUYE EN EL CODIGO EL CORRIMIENTO.                          **}
{**                                                                  **}
{**********************************************************************}
PROCEDURE SHIFTCOUNT( VAR APUNTEXT: INTEGER);
VAR
 I:          INTEGER;
 CARACTER:   CHAR;
BEGIN
   CARACTER:= CINMEDIATO[1,4];
   CASE  CARACTER OF
      '1': BEGIN
             POINTERCMAQ:= POINTERCMAQ+1;
             CMAQUINA[POINTERCMAQ]:= '0';
             POINTERCMAQ:= POINTERCMAQ+1;
```

```
        CMAQUINA[POINTERCMAQ]:= '0';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
      END;
'2': BEGIN
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '0';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '0';
      END;
'3': BEGIN
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '0';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
      END;
'4': BEGIN
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '0';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '0';
      END;
'5': BEGIN
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '0';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
      END;
'6': BEGIN
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '0';
      END;
'7': BEGIN
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '1';
      END;
'8': BEGIN
        POINTERCMAQ:= POINTERCMAQ+1;
        CMAQUINA[POINTERCMAQ]:= '0';
```

```
              POINTERCMAQ:= POINTERCMAQ+1;
              CMAQUINA[POINTERCMAQ]:= '0';
              POINTERCMAQ:= POINTERCMAQ+1;
              CMAQUINA[POINTERCMAQ]:= '0';
            END;
   END; {* CASE *}
   APUNTEXT:=0;
END; {* SHIFTCOUNT*}
{*                                                    *}
 BEGIN
  I:=1;
  POINTERCMAQ:=0;
  WHILE ( I <= LONGBIN ) DO
   BEGIN
    CARACTER:= ABINARIO[NUMBINARIO,I];
    CASE CARACTER OF

     'A': BEGIN
            ADDRESSREGISTER;
            I:=I+3;
          END;

     'C': BEGIN
            DATAREGISTER;
            I:=I+3;
          END;
     'F': BEGIN
            EFECTIVEADDRESS;
            I:=I+6;
          END;
     'J': BEGIN
            EFECTIVESOURCE;
            I:=I+6;
          END;
     'L': BEGIN
            EFECTIVEDESTINO;
            I:=I+6;
          END;
     'N': BEGIN
            DESTDATA;
            I:=I+3;
          END;
      'P': BEGIN
            DESTADDRESS;
            I:=I+3;
          END;
      'O':  BEGIN
            DESTINOEFECTIVE;
            I:=I+6;
          END;
      'S': BEGIN
            SHIFTCOUNT(APUNTEXT);
            I:=I+3;
          END;
      ELSE;
```

```
            BEGIN
              POINTERCMAQ:= POINTERCMAQ+1;
              CMAQUINA[POINTERCMAQ]:= CARACTER;
              I:=I+1;
            END;
    END; {* CASE *}
   END; {* WHILE *}
END; {* ENSAMBLA *}
{*                                                                    *}
{**********************************************************************}
{**  SUSTITUYE EL CODIGO BINARIO DE LA INSTRUCCION                   **}
{**  POR UN CODIGO EN HEXADECIMAL.                                   **}
{**********************************************************************}
PROCEDURE HEXADECIMAL( CMAQUINA: TCMAQUINA; CINMEDIATO: TINMEDIATO;
                       APUNTEXT: INTEGER; VAR CMAQHEX: TCMAQHEX);
VAR
 BIT,I,J,POINTER:        INTEGER;
 L,SUMA,ABIT,EXP:        INTEGER;
BEGIN
 POINTER:= 1;
 WHILE ( POINTER <= 4 ) DO
  BEGIN
    I:=POINTER * 4;
    SUMA:=0;
    EXP:= 0;
    REPEAT
      ABIT:=1;
      BIT:= ORD(CMAQUINA[I]) - 48;
      IF EXP > 0 THEN
       BEGIN
        FOR J:=1 TO EXP DO
         ABIT:=2*ABIT ;
       END;
      SUMA:= ABIT*BIT + SUMA;
      I:=I-1;
      EXP:=EXP+1;
    UNTIL ( EXP > 3 );
   CASE  SUMA OF
      0:  BEGIN
            CMAQHEX[POINTER]:= '0';
            POINTER:= POINTER+1;
          END;
      1:  BEGIN
            CMAQHEX[POINTER]:= '1';
            POINTER:= POINTER+1;
          END;
      2:  BEGIN
            CMAQHEX[POINTER]:= '2';
            POINTER:= POINTER+1;
          END;
      3:  BEGIN
            CMAQHEX[POINTER]:= '3';
            POINTER:= POINTER+1;
          END;
      4:  BEGIN
```

```
                CMAQHEX[POINTER]:= '4';
                POINTER:= POINTER+1;
              END;
          5:  BEGIN
                CMAQHEX[POINTER]:= '5';
                POINTER:= POINTER+1;
              END;
          6:  BEGIN
                CMAQHEX[POINTER]:= '6';
                POINTER:= POINTER+1;
              END;
          7:  BEGIN
                CMAQHEX[POINTER]:= '7';
                POINTER:= POINTER+1;
              END;
          8:  BEGIN
                CMAQHEX[POINTER]:= '8';
                POINTER:= POINTER+1;
              END;
          9:  BEGIN
                CMAQHEX[POINTER]:= '9';
                POINTER:= POINTER+1;
              END;
         10:  BEGIN
                CMAQHEX[POINTER]:= 'A';
                POINTER:= POINTER+1;
              END;
         11:  BEGIN
                CMAQHEX[POINTER]:= 'B';
                POINTER:= POINTER+1;
              END;
         12:  BEGIN
                CMAQHEX[POINTER]:= 'C';
                POINTER:= POINTER+1;
              END;
         13:  BEGIN
                CMAQHEX[POINTER]:= 'D';
                POINTER:= POINTER+1;
              END;
         14:  BEGIN
                CMAQHEX[POINTER]:= 'E';
                POINTER:= POINTER+1;
              END;
         15:  BEGIN
                CMAQHEX[POINTER]:= 'F';
                POINTER:= POINTER+1;
              END;
      END; {* CASE *}
  END; {* WHILE *}
FOR I:= 1 TO 4 DO
 WRITE(CMAQHEX[I]);
FOR I:= 1 TO APUNTEXT DO
 FOR J:=1 TO 4 DO
  WRITE(CINMEDIATO[I,J]);
WRITELN;
```

```
END; {* HEXADECIMAL*}

 {*                              *}
    BEGIN
    ASSIGN (FUENTE,'FUENTE.DAT');
    RESET(FUENTE);
    AUTOMATA;
    CREAEFEC;
    WHILE NOT EOF(FUENTE) DO
     BEGIN
      I:=0;
      REPEAT
        I:= I+1;
        READ(FUENTE,LINEAIN[I]);
      UNTIL (LINEAIN[I] = ';') OR ( I = LIMINST);
      LONG:= I;
      QUITABLANCOS(LONG,LINEAIN,LINEAOUT);
      PORWORDS(PALABRAS,LINEAOUT,NUMWORDS);
      FOR J:=1 TO LONGNOM DO
       PALABRA[J]:= PALABRAS[1,J];
      OBTENCODIGO(PALABRA,POSIBILIDADES,ERROR,NUMPBLE,ABINARIO,TIPOINSTR
;
      FOR I:=1 TO 4 DO
       FOR J:=1 TO 4 DO
        CINMEDIATO[I,J]:='0';
      I:=2; AMODO[1]:=99; AMODO[2]:=99;APUNTEXT:=0;
      WHILE I <= NUMWORDS DO
       BEGIN
        FOR J:=1 TO LONGNOM DO
          PALABRA[J]:= PALABRAS[I,J];
        VIAJA(PALABRA,MODO,NUMREG,TIPOINSTRUC,APUNTEXT);
        AMODO[I-1]:=MODO;
        EFECTIVA(PALABRA,MODO,DIREFEC,NUMREG);
        FOR J:=1 TO 6 DO
         ADIREFEC[I-1,J]:= DIREFEC[J];
        I:=I+1;
       END;
       VERMODOS(NUMBINARIO);
       ENSAMBLA(NUMBINARIO,CMAQUINA,APUNTEXT);
       HEXADECIMAL(CMAQUINA,CINMEDIATO,APUNTEXT,CMAQHEX);
     READLN(FUENTE);
     END;
    CLOSE(FUENTE);
   END.
```

PROGRAMA MACROENSAMBLE

El siguiente listado contiene la parte del sistema que procesa MACROS o instruciones para el procesador y que contiene los siguientes procedimientos principales:

a) Elimina exceso de blancos y compacta la instrucción.

b) Separa por palabras la instrucción actual analizada.

c) Guarda el argumento de las macros.

d) Identifica la declaración de MACRO y guarda su nombre y el cuerpo de esta.

e) Reemplaza en el cuerpo de la macro los argumentos por índices.

f) Procedimiento para recolectar nombres de MACROS y argumentos.

g) Procedimiento para expandir el cuerpo de la MACRO.

```
PROGRAM MACROENSAM(INPUT,OUTPUT,FUENTE,INTERMEDIO,EXPANDIDO);
 CONST
  LIMNOM=         15;
  LONGNOM=        12;
  LIMCUERPO=      80;
  LIMINST=        40;
  LIMCTD=          7;
  LIMARG=          5;
  LONARG=         15;
  LONGSEP=         3;
 TYPE
  WORDS=          PACKED ARRAY [1..LIMCTD, 1..LONGNOM ] OF CHAR;
  INSTRUC=        PACKED ARRAY [1..80 ] OF CHAR;
  APUNTADORES =   INTEGER;
  BANDERA=        BOOLEAN;
 VAR
  ERROR,CONTINST:      INTEGER;
  FUENTE,INTERMEDIO,
  EXPANDIDO:          TEXT;
  LINEAIN,LINEAOUT:  INSTRUC;
  APUNTNOM:           APUNTADORES;
  APUNTCUERPO:        APUNTADORES;
  APUNTINDICE:        APUNTADORES;
  APUNTARGMCR:        APUNTADORES;
  HAYMACRO:           BANDERA;
  HAYLLAMADA:         BANDERA;
  I,J, LONG:          INTEGER;
  PALABRAS:           WORDS;
  ADICIONAL:          PACKED ARRAY [1..LIMCTD, 1..LONGSEP] OF CHAR;
  NOMBRESMCR:         PACKED ARRAY [1..LIMNOM,1..LONGNOM] OF CHAR;
  INDICEMCR:          ARRAY [1..LIMNOM,1..2] OF INTEGER;
  CUERPOMCR:          PACKED ARRAY [1..LIMCUERPO,1..LIMINST] OF CHAR;
  ARGMCR:             PACKED ARRAY [1..LIMARG,1..LONARG] OF CHAR;
  {*****************************************************}
  {*                                                   *}
  {*          ELIMINA EXCESO DE BLANCOS                *}
  {*          Y COMPACTA LA INSTRUCCION                *}
  {*                                                   *}
  {*****************************************************}
  PROCEDURE QUITABLANCOS (VAR LONG: INTEGER; VAR LINEA: INSTRUC;
                          VAR LINEA2:-INSTRUC );
  CONST
   BLANCO=        ' ';
  VAR
   SEPARADOR:  SET OF CHAR;
   I,J,K,L,H:    INTEGER;
   CARACTER:   CHAR;
  BEGIN
   SEPARADOR:= [ ]; SEPARADOR:=SEPARADOR + ['@'];
   SEPARADOR:= SEPARADOR + ['('];  SEPARADOR:= SEPARADOR + ['$'];
   SEPARADOR:= SEPARADOR + [')'];  SEPARADOR:= SEPARADOR + ['&'];
   SEPARADOR:= SEPARADOR + [','];
   J:=0; K:=0; L:=0; H:=0;
   FOR I:=1 TO LONG DO
     BEGIN
```

```
      CARACTER:= LINEA[I];
      IF CARACTER IN SEPARADOR THEN
         H:=1;
      IF CARACTER <> BLANCO THEN
       BEGIN
        J:= J+1;
        LINEA2[J]:=LINEA[I];
        K:=0; L:=1;
       END
      ELSE
       BEGIN
        IF (K = 0) AND (L = 1) AND (H = 0) THEN
         BEGIN
          J:=J+1;
          LINEA2[J]:=LINEA[I];
          K:=1;
         END;
        END;
       LONG:=J;
      END;
   END;{*QUITABLANCOS*}
{*****************************************************************}
{*                                                               *}
{* SEPARA POR PALABRAS LA INSTRUCCION ACTUAL ANALIZADA           *}
{*                                                               *}
{*****************************************************************}
   PROCEDURE PORWORDS( VAR PALABRAS: WORDS; LINEA: INSTRUC;
                       VAR NUMWORDS: INTEGER);
   CONST
   BLANCO= ' ';
   PUNTOCOMA= ';';
  VAR
   CARACTER:     CHAR;
   I,J,K,W,WW:   INTEGER;
   SEPARADOR:    SET OF CHAR;
{*                                                               *}
{*     GUARDA PALABRAS QUE FORMAN LA INSTRUCCION                 *}
{*                                                               *}
  PROCEDURE GUARDAWORD;
  BEGIN
   K:= K+1; J:=0; WW:=0;
   REPEAT
    J:= J+1;
    PALABRAS[K,J]:= LINEA[I];
    I:= I+1
   UNTIL ( LINEA[I] IN SEPARADOR ) OR ( I > LONG );
   I:= I-1; {* PARA APUNTAR AL ULTIMO CARACTER VISTO *}
  END;
{*                                                               *}
  BEGIN
    SEPARADOR:=[ ]; SEPARADOR:= SEPARADOR + [' '];
    SEPARADOR:=SEPARADOR + [','']; SEPARADOR:= SEPARADOR + [';'];
    {*SEPARA POR PALABRAS*}
    FOR K:=1 TO LIMCTD DO
     BEGIN
```

```
    FOR I:=1 TO LONGSEP DO
      ADICIONAL[K,I]:= BLANCO;
    FOR I:=1 TO LONGNOM DO
      PALABRAS[K,I]:= BLANCO;
   END;
   K:=0 ; I:=0; WW:=0;
   WHILE (K <= LIMCTD) AND ( I < LONG ) DO
    BEGIN
     I:= I+1;
     CARACTER:= LINEA[I];
     IF  CARACTER IN SEPARADOR THEN
      BEGIN
       WW:= WW+1;
       ADICIONAL[K,WW]:= CARACTER;
      END
     ELSE
        GUARDAWORD;
    END;
    NUMWORDS:=K;
END; {PORWORDS}
{*************************************************************}
{*                                                           *}
{*  GUARDA EL ARGUMENTO DE LAS MACROS, RELACIONANDOLOS       *}
{*  CON UN INDICE.                                           *}
{*                                                           *}
{*************************************************************}
  PROCEDURE ARGUMENTOS( PALABRAS: WORDS; NUMWORDS: INTEGER);
   CONST
    AMPER=     '&';
    BLANCO=    ' ';
   VAR
    BLANCOS:    PACKED ARRAY [1..LONGNOM] OF CHAR;
    I,K,LL:     INTEGER;
    HAYARG:     BANDERA;
   BEGIN
    FOR I:=1 TO LONGNOM DO
     BLANCOS[I]:=BLANCO;
    HAYARG:= TRUE;
    J:=3;    { POSICION FIJA DESDE DONDE COMIENZAN LOS ARGUMENTOS }
    K:=0;
    LL:= 48; {* ascii del cero *}
    WHILE  ( HAYARG ) AND ( J <= NUMWORDS)  DO
     BEGIN
      IF PALABRAS[J,1] = AMPER THEN
       BEGIN
          K:=K+1;
          FOR I:=1 TO LONGNOM DO
            ARGMCR[K,I]:= PALABRAS[J,I];
          ARGMCR[K,13]:= AMPER;
          LL:=LL+1;
          ARGMCR[K,14]:= CHR(LL);
       END
       ELSE
        BEGIN
          WRITELN(' SIMBOLO NO ASOCIADO A NINGUNA VARIABLE ');
```

```
          HAYARG:= FALSE;
        END;
        J:=J+1;
      END; {WHILE}
      APUNTARGMCR:= K;
     END; {ARGUMENTOS}



{*************************************************************}
{*                                                           *}
{*  IDENTIFICA A LA DECLARACION DE MACRO Y GUARDA SU         *}
{*  NOMBRE EN LA TABLA DE NOMBRES DE MACRO, INDICANDO        *}
{*  QUE LAS SIGUIENTES LINEAS SERAN EL CUERPO DE LA MACRO    *}
{*                                                           *}
{*************************************************************}
  PROCEDURE MACRO (LONG: INTEGER; LINEA: INSTRUC);
   CONST
     M     =     'M';
     A     =     'A';
     C     =     'C';
     R     =     'R';
     O     =     'O';
     BLANCO =    ' ';
     SEIS   =     6;
   TYPE
     ALFA   =   'A'..'Z';
   VAR
    I,J,K:     INTEGER;
    NUMWORDS: INTEGER;
    PALABRAS: WORDS;
    CADENA:   PACKED ARRAY[1..SEIS] OF CHAR;
    ALFABETO: SET OF ALFA;
    CARAC:    ALFA;


BEGIN
ALFABETO:= [ ];
FOR CARAC:='A' TO 'Z' DO
 ALFABETO:=ALFABETO + [CARAC];
PORWORDS( PALABRAS,LINEA,NUMWORDS);
CADENA[1]:= M; CADENA[2]:=A; CADENA[3]:=C;
CADENA[4]:=R; CADENA[5]:=O; CADENA[6]:= BLANCO;
J:=0;
FOR I:=1 TO SEIS DO
  BEGIN
   IF PALABRAS[1,I] = CADENA[I] THEN
     J:= J+1;
   END;
  IF J = SEIS THEN {ENCONTRAMOS LA PALABRA MACRO *}
   BEGIN
    APUNTNOM:= APUNTNOM +1;
    APUNTINDICE:= APUNTINDICE +1;
    INDICEMCR[APUNTINDICE,1]:= APUNTCUERPO +1;
```

```
    IF PALABRAS[2,1] IN ALFABETO THEN
    ELSE
    ERROR:= 3;
    FOR I:= 1 TO LONGNOM DO
     BEGIN
          NOMBRESMCR [APUNTMOM,I]:= PALABRAS[2,I];
          HAYMACRO:= TRUE;
     END;
     ARGUMENTOS(PALABRAS,NUMWORDS);
    END;
   END;  {* MACRO *}
{*************************************************************}
{*                                                           *}
{*   REEMPLAZA  EN EL CUERPO DE LA MACRO LOS ARGUMENTOS      *}
{*   POR INDICES.                                            *}
{*                                                           *}
{*************************************************************}
 PROCEDURE REEMPLAZA( VAR LINEA: INSTRUC; VAR LONG: INTEGER;
                      BRECOLECTA: BANDERA);
 CONST
  BLANCO= ' ';
 VAR
  PALABRAS:      WORDS;
  NUMWORDS:.     INTEGER;
  I,J,H:         INTEGER;
  YAVIBLANCO:    BANDERA;
{*****************************************************************}
{*                                                               *}
{*  COMPARA PALABRAS PARA INDICAR EL ARGUMENTO A SUSTITUIR.      *}
{*                                                               *}
{*****************************************************************}
PROCEDURE COMPARANORDS;
 VAR
  I,H,J,WW,W:      INTEGER;
  ENCONTRADO:      BANDERA;
BEGIN
 I:=0;
 WHILE ( I < NUMWORDS ) DO
  BEGIN
   I:= I+1;H:=0; ENCONTRADO:= FALSE;
   REPEAT
    H:=H+1; W:=0;
    FOR J:=1 TO LONGNOM DO
     BEGIN
      IF PALABRAS[I,J] = ARGMCR[H,J] THEN
       W:=W+1;
     END;
     IF W = LONGNOM THEN
      BEGIN
        FOR WW:=1 TO LONGNOM DO
          PALABRAS[I,WW]:= BLANCO;
        PALABRAS[I,1]:= ARGMCR[H,13];
        PALABRAS[I,2]:= ARGMCR[H,14];
        ENCONTRADO:=TRUE;
      END
```

```
   UNTIL ( ENCONTRADO = TRUE ) OR ( H = APUNTARGMCR )
  END; {* WHILE *}
 END; {* COMPARAWORDS *}
{*****************************************************************}
{*                                                               *}
{*  COMPARA PALABRAS PARA INDICAR EL ARGUMENTO A SUSTITUIR.      *}
{*  PARA LA EXPANSION DE LA MACRO.                               *}
{*****************************************************************}
PROCEDURE EXPWORDS;
 VAR
  I,H,J,WW,W:      INTEGER;
  ENCONTRADO:         BANDERA;
BEGIN
 I:=0;
 WHILE ( I < NUMWORDS ) DO
  BEGIN
   I:= I+1;H:=0; ENCONTRADO:= FALSE;
   REPEAT
    H:=H+1; W:=0;
    IF PALABRAS[I,1] = ARGMCR[H,13] THEN
       W:=W+1;
    IF PALABRAS[I,2] = ARGMCR[H,14] THEN
       W:= W+1;
    IF W = 2 THEN
      BEGIN
        FOR WW:=1 TO LONGNOM DO
          PALABRAS[I,WW]:= BLANCO;
        FOR WW:=1 TO LONGNOM DO
         PALABRAS[I,WW]:= ARGMCR[H,WW];
        ENCONTRADO:=TRUE;
      END
   UNTIL ( ENCONTRADO = TRUE ) OR ( H = APUNTARGMCR )
  END; {* WHILE *}

 END; {* EXPWORDS *}  {*
  *}
{****                                                      *****}
{*                                                            *}

 BEGIN
  PORWORDS(PALABRAS,LINEA,NUMWORDS);
   IF BRECOLECTA THEN    {* RECOLECTANDO LA DEFINICION DE MACRO*}
     COMPARAWORDS
  ELSE
    EXPWORDS;          {* EXPANDIENDO CUERPO DE MACRO *}
  H:=0;
  FOR I:=1 TO NUMWORDS DO
   BEGIN
     YAVIBLANCO:=FALSE;
     FOR J:=1 TO LONGNOM DO
      BEGIN
       IF ( PALABRAS[I,J] <> BLANCO ) THEN
         BEGIN
          H:=H+1;
          LINEA[H]:= PALABRAS[I,J];
         END
```

```
        ELSE
          BEGIN
           IF YAVIBLANCO = FALSE THEN
             BEGIN
               H:=H+1;
               LINEA[H]:=BLANCO;
               YAVIBLANCO:=TRUE;
             END;
          END;
       END;
      FOR J:=1 TO LONGSEP DO
       BEGIN
        IF ADICIONAL[I,J] <> BLANCO THEN
         BEGIN
          H:=H+1;
          LINEA[H]:= ADICIONAL[I,J];
         END;
       END;
    END; {* FOR *}
    LONG:=H;

END; {* REEMPLAZA *}



{*************************************************************}
{*                                                           *}
{*   ALMACENA EL CUERPO DE LAS MACROS Y ACTUALIZA LOS        *}
{*   INDICES EN LA TABLA DE INDICES DE LAS MACROS.           *}
{*                                                           *}
{*************************************************************}
  PROCEDURE CUERPO ( VAR LONG: INTEGER; LINEA: INSTRUC );
   CONST
   BLANCO= ' ';
   CINCO= 5;
  VAR
   I,J,K,L:  INTEGER;
   NUMWORDS: INTEGER;
   CARACTER: CHAR;
   PALABRAS: WORDS;
   BRECOLECTA: BANDERA;
   CADENA:   PACKED ARRAY[1..CINCO] OF CHAR;
  BEGIN
   PORWORDS ( PALABRAS,LINEA,NUMWORDS);
   CADENA[1]:= 'M'; CADENA[2]:= 'E'; CADENA[3]:= 'N';
   CADENA[4]:= 'D'; CADENA[5]:= ' ';
   J:=0;
   FOR I:=1 TO CINCO DO
    BEGIN
     IF CADENA[I] = PALABRAS[1,I] THEN
      J:= J+1;
    END;
   IF J = CINCO THEN   {* HAY MEND*}
```

```
    BEGIN
     HAYMACRO:= FALSE;
     INDICEMCR[APUNTINDICE,2]:= APUNTCUERPO;
    END
   ELSE                    {*GUARDA CUERPO*}
     BEGIN
      APUNTCUERPO:= APUNTCUERPO +1;
      BRECOLECTA:= TRUE;
      REEMPLAZA(LINEA,LONG,BRECOLECTA);
      FOR I:=1 TO LONG DO
       CUERPOMCR[APUNTCUERPO,I]:= LINEA[I];
      FOR I:=LONG+1 TO LIMINST DO
       CUERPOMCR[APUNTCUERPO,I]:= BLANCO;
     END;
  END; {* CUERPO*}
{*                                                          *}
{*  PROCEDIMIENTO PARA RECOLECTAR NOMBRES DE MACROS Y       *}
{*  ARGUMENTOS.                                             *}
{*                                                          *}
{*  ENTRADA :     PROGRAMA FUENTE                           *}
{*                                                          *}
{*  SALIDA  :     TABLA DE NOMBRE DE MACROS                 *}
{*                TABLA DE INDICES DE MACROS                *}
{*                TABLA DE CUERPOS                          *}
{*                                                          *}
PROCEDURE RECOLECTA;
  CONST
   BLANCO= ' ';
  VAR
   ENDP,ENDPA: PACKED ARRAY [1..5] OF CHAR;
  BEGIN
  APUNTNOM:= 0; APUNTCUERPO:=0; APUNTINDICE:=0; HAYMACRO:= FALSE;
  ASSIGN (FUENTE,'FUENTE.DAT');
  ASSIGN (INTERMEDIO,'INTERMEDIO.DAT');
  REWRITE(INTERMEDIO);
  RESET(FUENTE);
  CONTINST:=0;
  WHILE (NOT EOF(FUENTE)) AND (ERROR = 0) DO
   BEGIN
    I:=0; CONTINST:=CONTINST + 1;
    REPEAT
      I:= I+1;
      READ(FUENTE,LINEAIN[I]);
    UNTIL (LINEAIN[I] = ';') OR ( I = LIMINST);
    IF I = LIMINST THEN
      ERROR:= 1
    ELSE
     BEGIN
       LONG:= I;
       QUITABLANCOS(LONG,LINEAIN,LINEAOUT);
       FOR J:=1 TO 5 DO
           ENDPA[J]:= LINEAOUT[J];
       IF HAYMACRO THEN
         CUERPO(LONG,LINEAOUT)
       ELSE
```

```
          BEGIN
            MACRO (LONG, LINEAOUT);
            IF HAYMACRO = FALSE THEN
              BEGIN
                FOR I:=1 TO LONG DO
                  WRITE(INTERMEDIO,LINEAOUT[I]);
                WRITELN(INTERMEDIO);
              END;
          END;
        READLN(FUENTE);
      END;
    END; {* WHILE *}
    IF ERROR = 0 THEN
    BEGIN
        ENDP[1]:='E';ENDP[2]:='N';ENDP[3]:='D';ENDP[4]:=';';
        J:=0;
        FOR I:=1 TO 5 DO
         BEGIN
           IF (ENDP[I] = ENDPA[I]) OR (ENDPA[I] = BLANCO ) THEN
              J:=J+1;
         END;
         IF J < 4 THEN
           ERROR:=2;
    END;
  CLOSE(INTERMEDIO);
  CLOSE(FUENTE);
 END;
{*                                                          *}
{*  PROCEDIMIENTO PARA EXPANDIR EL CUERPO DE LA MACRO       *}
{*                                                          *}
{*  ENTRADA :    PROGRAMA FUENTE SIN DECLARACION            *}
{*               DE MACROS.                                 *}
{*                                                          *}
{*  SALIDA  :    PROGRAMA FUENTE CON LAS LLAMADAS A         *}
{*               MACROS EXPANDIDAS.                         *}
{*                                                          *}
{*                                                          *}
{*                                                          *}
PROCEDURE LLAMADAMCR( PALABRAS: WORDS; VAR HAYLLAMADA: BANDERA;
                      NUMWORDS: INTEGER);
CONST
 PUNTOYCOMA=    ';';
 BLANCO=        ' ';
 AMPER=         '&';
VAR
 I,J,L,LUGAR,K,LL:  INTEGER;
 ENCONTRADO:        BANDERA;
 BRECOLECTA:        BANDERA;
 DIFERENTE:         BANDERA;
 LINEA:             INSTRUC;
BEGIN
 MAYLLAMADA:= FALSE;
 ENCONTRADO:= FALSE;
 I:=0;
 WHILE ( I <= APUNTNOM) AND ( ENCONTRADO = FALSE ) DO
```

```
BEGIN
 DIFERENTE:= FALSE;
 I:= I+1;
 J:=0;L:=0;
 REPEAT
  J:= J+1;
  IF PALABRAS[1,J] = NOMBRESMCR[I,J] THEN
     L:= L+1
  ELSE
   BEGIN
     IF PALABRAS[1,J] = PUNTOYCOMA THEN
      L:= L+1
     ELSE
      DIFERENTE:= TRUE
   END
 UNTIL  ( DIFERENTE = TRUE ) OR ( J = LONGNOM);
 IF L = LONGNOM THEN
    BEGIN
      ENCONTRADO:= TRUE;
      HAYLLAMADA:= TRUE;
      LUGAR:= I;
    END;
END; {* WHILE *}
IF ENCONTRADO THEN
   BEGIN
     LL:= 48; APUNTARGMCR:=0;
     FOR I:=2 TO NUMWORDS DO
      BEGIN
       FOR J:=1 TO LONGNOM DO
        ARGMCR[I-1,J]:= PALABRAS[I,J];
       LL:= LL+1;
       APUNTARGMCR:= APUNTARGMCR+1;
       ARGMCR[APUNTARGMCR,13]:= AMPER;
       ARGMCR[APUNTARGMCR,14]:= CHR(LL);
      END;
      J:= INDICEMCR[LUGAR,1];
      K:= INDICEMCR[LUGAR,2];
      FOR I:=J TO K DO
       BEGIN
        LL:=0;
        FOR L:=1 TO LIMINST DO
         BEGIN
          LL:=LL+1;
          LINEA[LL]:= CUERPOMCR[I,L];
          IF CUERPOMCR[I,L] = PUNTOYCOMA THEN
           LONG:=LL;
         END;
         BRECOLECTA:= FALSE;
         REEMPLAZA(LINEA,LL,BRECOLECTA);
         FOR L:=1 TO LL DO
          WRITE(EXPANDIDO,LINEA[L]);
       WRITELN(EXPANDIDO);
      END;{* IF *}
   END;
```

```
   END; {* LLAMADAMCR *)
{*                                                    *)
   PROCEDURE EXPANDE;
     VAR
      NUMWORDS: INTEGER;
     BEGIN
     HAYLLAMADA:= FALSE;
     ASSIGN (INTERMEDIO,'INTERMEDIO.DAT');
     ASSIGN (EXPANDIDO,'EXPANDIDO.DAT');
     REWRITE(EXPANDIDO);
     RESET(INTERMEDIO);
     WHILE NOT EOF(INTERMEDIO) DO
      BEGIN
       I:=0;
       REPEAT
         I:= I+1;
         READ(INTERMEDIO,LINEAIN[I])
       UNTIL LINEAIN[I] = ';';
       LONG:= I;
       PORWORDS(PALABRAS,LINEAIN,NUMWORDS);
       LLAMADAMCR(PALABRAS, HAYLLAMADA,NUMWORDS);
       IF HAYLLAMADA = FALSE THEN
        BEGIN
          FOR I:=1 TO LONG DO
             WRITE(EXPANDIDO,LINEAIN[I]);
          WRITELN(EXPANDIDO);
        END;
       READLN(INTERMEDIO);
     END;
     CLOSE(INTERMEDIO);
     CLOSE(EXPANDIDO);
    END;
  {*                                                    *)
  {*              PROGRAMA PRINCIPAL MACRO              *)
  {*                                                    *)
   BEGIN
FOR I:=1 TO 10 DO;
WRITELN;
WRITELN('************** PRINCIPIA PROGRAMA MACRO ***************');
     ERROR:= 0;
     RECOLECTA;
     IF ERROR <> 0 THEN
      BEGIN
       CASE ERROR OF
         1: WRITELN('*ERR* INSTRUCCION ',CONTINST,' FALTA ; ');
         2: WRITELN('*ERR* PROGRAMA SIN END FINAL ');
         3: WRITELN('*ERR* FALTA NOMBRE DE MACRO  EN ',CONTINST);
       END;
      END
     ELSE
        EXPANDE;
WRITELN('***************** TERMINO PROGRAMA MACRO ***************');
   END.
```

ARCHIVO DE INSTRUCCIONES

El siguiente archivo contiene las instrucciones válidas para el microprocesador MC68000 junto con unas tablas que sirven para válidar el modo de direccionamiento del operando, por ejemplo si el operando es fuente tiene que encontrar un "uno" en la intercección de la columna con el renglón de la instrucción, la posición en las columnas están relacionadas directamente con los modos de direccionamiento. La columna 38 contiene un uno si la instrucción es de tamaño palabra y un dos si es de tamaño palabra larga. Las columnas de la 39 en adelante son los formatos de cada instrucción, que una vez procesados son divididos de cuatro en cuatro generando un número hexadecimal.

```
ADD.B       12020202020202020200000 1 1101CCC100FFFGGG
ADD.B       02101010101010101010101 1 1101CCC000FFFGGG
ADD.W       12020202020202020200000 1 1101CCC101FFFGGG
ADD.W       02101010101010101010101 1 1101CCC001FFFGGG
ADD.L       12020202020202020200000 2 1101CCC110FFFGGG
ADD.L       02101010101010101010101 2 1101CCC010FFFGGG
ADDA.W      10120010101010101010101 1 1101AAA011JJJKKK
ADDA.L      10120010101010101010101 2 1101AAA111JJJKKK
ADDI.B      02000202020202020200010 1 0000011000LLLMMM
ADDI.W      02000202020202020200010 1 0000011001LLLMMM
ADDI.L      02000202020202020200010 2 0000011010LLLMMM
ADDX.B      12000000000000000000000 1 1101NNN100000CCC
ADDX.B      00001200000000000000000 1 1101AAA100001PPP
ADDX.W      12000000000000000000000 1 1101NNN101000CCC
ADDX.W      00001200000000000000000 1 1101AAA101001PPP
ADDX.L      12000000000000000000000 2 1101NNN110000CCC
ADDX.L      00001200000000000000000 2 1101AAA110001PPP
CLR.B       10001010101010100000000 1 0100001000JJJKKK
CLR.W       10001010101010100000000 1 0100001001JJJKKK
CLR.L       10001010101010100000000 2 0100001010JJJKKK
CMP.B       12101010101010101010101 1 1011NNN000JJJKKK
CMP.W       12101010101010101010101 1 1011NNN001JJJKKK
CMP.L       12101010101010101010101 2 1011NNN010JJJKKK
CMPA.W      10121010101010101010101 1 1011NNN011JJJKKK
CMPA.L      10121010101010101010101 2 1011NNN111JJJKKK
CMPI.B      02000202020202020200010 1 0000110000LLLMMM
CMPI.W      02000202020202020200010 1 0000110001LLLMMM
CMPI.L      02000202020202020200010 2 0000110010LLLMMM
CMPM.B      00000012000000000000000 1 1011AAA100001PPP
CMPM.W      00000012000000000000000 1 1011AAA101001PPP
CMPM.L      00000012000000000000000 2 1011AAA110001PPP
DIVS.W      12001010101010101010101 1 1000NNN111JJJKKK
DIVU.W      12001010101010101010101 1 1000NNN011JJJKKK
EXT.W       10000000000000000000000 1 0100100010000CCC
EXT.L       10000000000000000000000 2 0100100011000CCC
MULS.W      12001010101010101010101 1 1100NNN011JJJKKK
MULU.W      12001010101010101010101 1 1100NNN010JJJKKK
NEG.B       10001010101010101000000 1 0100010000JJJKKK
NEG.W       10001010101010101000000 1 0100010001JJJKKK
NEG.L       10001010101010101000000 2 0100010010JJJKKK
NEGX.B      10001010101010101000000 1 0100000000JJJKKK
NEGX.W      10001010101010101000000 1 0100000001JJJKKK
NEGX.L      10001010101010101000000 2 0100000010JJJKKK
SUB.B       12101010101010101010101 1 1001NNN000JJJKKK
SUB.B       12020202020202020200000 1 1001CCC100LLLMMM
SUB.W       12101010101010101010101 1 1001NNN001JJJKKK
SUB.W       12020202020202020200000 1 1001CCC101LLLMMM
SUB.L       12101010101010101010101 2 1001NNN010JJJKKK
SUB.L       12020202020202020200000 2 1001CCC110LLLMMM
SUBA.W      10121010101010101010101 1 1001AAA011JJJKKK
SUBA.L      10121010101010101010101 2 1001AAA111JJJKKK
SUBI.B      02000202020202020200010 1 0000010000LLLMMM
SUBI.W      02000202020202020200010 1 0000010001LLLMMM
SUBI.L      02000202020202020200010 2 0000010010LLLMMM
SUBX.B      12000000000000000000000 1 1001NNN100000CCC
```

```
SUBX.B      00001200000000000000000  1  1001AAA100001PPP
SUBX.W      12000000000000000000000  1  1001NNN101000CCC
SUBX.W      00001200000000000000000  1  1001AAA101001PPP
SUBX.L      12000000000000000000000  2  1001NNN110000CCC
SUBX.L      00001200000000000000000  2  1001AAA110001PPP
AND.B       12001010101010101010101  1  1100NNN000JJJKKK
AND.B       12020202020202020200000  1  1100CCC100LLLMMM
AND.W       12001010101010101010101  1  1100NNN001JJJKKK
AND.W       12020202020202020200000  1  1100CCC101LLLMMM
AND.L       12001010101010101010101  2  1100NNN011JJJKKK
AND.L       12020202020202020200000  2  1100CCC111LLLMMM
ANDI.B      02000202020202020200010  1  0000001000LLLMMM
ANDI.W      02000202020202020200010  1  0000001001LLLMMM
ANDI.L      02000202020202020200010  2  0000001010LLLMMM
EOR.B       12000202020202020200000  1  1011CCC100LLLMMM
EOR.W       12000202020202020200000  1  1011CCC101LLLMMM
EOR.L       12000202020202020200000  2  1011CCC110LLLMMM
EORI.B      02000202020202020200010  1  0000101000LLLMMM
EORI.W      02000202020202020200010  1  0000101001LLLMMM
EORI.L      02000202020202020200010  2  0000101010LLLMMM
NOT.B       10001010101010101000000  1  0100011000JJJKKK
NOT.W       10001010101010101000000  1  0100011001JJJKKK
NOT.L       10001010101010101000000  2  0100011010JJJKKK
OR.B        12001010101010101010101  1  1000NNN000JJJKKK
OR.B        12020202020202020200000  1  1000CCC100LLLMMM
OR.W        12001010101010101010101  1  1000NNN001JJJKKK
OR.W        12020202020202020200000  1  1000CCC101LLLMMM
OR.L        12001010101010101010101  2  1000NNN010JJJKKK
OR.L        12020202020202020200000  2  1000CCC110LLLMMM
ORI.B       02000202020202020200010  1  0000000000LLLMMM
ORI.W       02000202020202020200010  1  0000000001LLLMMM
ORI.L       02000202020202020200010  2  0000000010LLLMMM
TST.B       10001010101010101000000  1  0100101000JJJKKK
TST.W       10001010101010101000000  1  0100101001JJJKKK
TST.L       10001010101010101000000  2  0100101010JJJKKK
EXG.L       12000000000000000000000  2  1100CCC101000NNN
EXG.L       00120000000000000000000  2  1100PPP101001AAA
LEA.L       00021000000000000000000  2  0100AAA111JJJKKK
MOVE.B      12101212121212121210101  1  0001000QQQJJJKKK
MOVE.W      12101212121212121210101  1  0011000QQQJJJKKK
MOVE.L      12101212121212121210101  2  0010000QQQJJJKKK
MOVEA.W     10121010101010101010101  1  0011AAA001JJJKKK
MOVEA.L     10121010101010101010101  2  0010AAA001JJJKKK
MOVEP.W     10000000000200000000000  1  0000CCC100001AAA
MOVEP.W     02000000001000000000000  1  0000CCC110001AAA
MOVEP.L     10000000000200000000000  2  0000CCC101001AAA
MOVEP.L     02000000001000000000000  2  0000CCC111001AAA
MOVEQ.L     02000000000000000000010  2  0111NNN0SSSSSSSS
PEA.L       00001000001010101010000  2  0100100001JJJKKK
ASL.B       12000000000000000000000  1  1110CCC100100NNN
ASL.B       02000000000000000000010  1  1110SSS100000NNN
ASL.W       12000000000000000000000  1  1110CCC101100NNN
ASL.W       02000000000000000000010  1  1110SSS101100NNN
ASLM.W      00001010101010101000000  1  1110000111JJJKKK
ASL.L       12000000000000000000000  2  1110CCC111100NNN
```

```
ASL.L       02000000000000000000000010 2 1110SSS111000NNN
ASR.B       12000000000000000000000000 1 1110CCC000100NNN
ASR.B       02000000000000000000000010 1 1110SSS000000NNN
ASR.W       12000000000000000000000000 1 1110CCC001100NNN
ASR.W       02000000000000000000000010 1 1110SSS001000NNN
ASRM.W      00001010101010101000000000 1 1110000111JJJKKK
ASR.L       12000000000000000000000000 2 1110CCC011100NNN
ASR.L       02000000000000000000000010 2 1110SSS011000NNN
LSL.B       12000000000000000000000000 1 1110CCC100101NNN
LSL.B       02000000000000000000000010 1 1110SSS100001NNN
LSL.W       12000000000000000000000000 1 1110CCC101101NNN
LSL.W       02000000000000000000000010 1 1110SSS101001NNN
LSLM.W      00001010101010101000000000 1 1110001111JJJKKK
LSL.L       12000000000000000000000000 2 1110CCC111101NNN
LSL.L       02000000000000000000000010 2 1110SSS111001NNN
LSR.B       12000000000000000000000000 1 1110CCC000101NNN
LSR.B       02000000000000000000000010 1 1110SSS000001NNN
LSR.W       12000000000000000000000000 1 1110CCC001101NNN
LSR.W       02000000000000000000000010 1 1110SSS001001NNN
LSRM.W      00001010101010101000000000 1 1110001011JJJKKK
LSR.L       12000000000000000000000000 2 1110CCC011101NNN
LSR.L       02000000000000000000000010 2 1110SSS011001NNN
ROL.B       12000000000000000000000000 1 1110CCC100111NNN
ROL.B       02000000000000000000000010 1 1110SSS100011NNN
ROL.W       12000000000000000000000000 1 1110CCC101111NNN
ROL.W       02000000000000000000000010 1 1110SSS101011NNN
ROLM.W      00001010101010101000000000 1 1110011111JJJKKK
ROL.L       12000000000000000000000000 2 1110CCC111111NNN
ROL.L       02000000000000000000000010 2 1110SSS111011NNN
ROR.B       12000000000000000000000000 1 1110CCC000111NNN
ROR.B       02000000000000000000000010 1 1110SSS000011NNN
ROR.W       12000000000000000000000000 1 1110CCC001111NNN
ROR.W       02000000000000000000000010 1 1110SSS001011NNN
RORM.W      00001010101010101000000000 1 1110011011JJJKKK
ROR.L       12000000000000000000000000 2 1110CCC011111NNN
ROR.L       02000000000000000000000010 2 1110SSS011011NNN
ROXL.B      12000000000000000000000000 1 1110CCC100110NNN
ROXL.B      02000000000000000000000010 1 1110SSS100010NNN
ROXL.W      12000000000000000000000000 1 1110CCC101110NNN
ROXL.W      02000000000000000000000010 1 1110SSS101010NNN
ROXLM.W     00001010101010101000000000 1 1110010111JJJKKK
ROXL.L      12000000000000000000000000 2 1110CCC111110NNN
ROXL.L      02000000000000000000000010 2 1110SSS111010NNN
ROXR.B      12000000000000000000000000 1 1110CCC000110NNN
ROXR.B      02000000000000000000000010 1 1110SSS000010NNN
ROXR.W      12000000000000000000000000 1 1110CCC001110NNN
ROXR.W      02000000000000000000000010 1 1110SSS001010NNN
ROXRM.W     00001010101010101000000000 1 1110010011JJJKKK
ROXR.L      12000000000000000000000000 2 1110CCC011110NNN
ROXR.L      02000000000000000000000010 2 1110SSS011010NNN
SWAP.W      10000000000000000000000000 1 0100100001000CCC
BCHG.B      12000202020202020200000000 1 0000CCC101LLLMMM
BCHG.B      02000202020202020200000010 1 0000100001LLLMMM
BCHG.L      12000202020202020200000000 2 0000CCC101LLLMMM
BCHG.L      02000202020202020200000010 2 0000100001LLLMMM
```

```
BCLR.B      12000202020202020200000 1 0000CCC110LLLMMM
BCLR.B      02000202020202020200010 1 0000100010LLLMMM
BCLR.L      12000202020202020200000 2 0000CCC110LLLMMM
BCLR.L      02000202020202020200010 2 0000100010LLLMMM
BSET.B      12000202020202020200000 1 0000CCC111LLLMMM
BSET.B      02000202020202020200010 1 0000100011LLLMMM
BSET.L      12000202020202020200000 2 0000CCC111LLLMMM
BSET.L      02000202020202020200010 2 0000100011LLLMMM
BTST.B      12000202020202020200000 1 0000CCC100LLLMMM
BTST.B      02000202020202020200010 1 0000100000LLLMMM
BTST.L      12000202020202020200000 2 0000CCC100LLLMMM
BTST.L      02000202020202020200010 2 0000100000LLLMMM
ABCD.B      12000000000000000000000 1 1100AAA100000CCC
ABCD.B      00001200000000000000000 1 1100AAA100001CCC
NBCD.B      02000202020202020200000 1 0100100000LLLMMM
SBCD.B      12000000000000000000000 1 1000AAA100000CCC
SBCD.B      00001200000000000000000 1 1000AAA100001CCC
BHI.S       00000000000000100000000 1 01100010YYYYYYYY
BLS.S       00000000000000100000000 1 01100011YYYYYYYY
BCC.S       00000000000000100000000 1 01100100YYYYYYYY
BCS.S       00000000000000100000000 1 01100101YYYYYYYY
BNE.S       00000000000000100000000 1 01100110YYYYYYYY
BEQ.S       00000000000000100000000 1 01100111YYYYYYYY
BVC.S       00000000000000100000000 1 01101000YYYYYYYY
BVS.S       00000000000000100000000 1 01101001YYYYYYYY
BPL.S       00000000000000100000000 1 01101010YYYYYYYY
BMI.S       00000000000000100000000 1 01101011YYYYYYYY
BGE.S       00000000000000100000000 1 01101100YYYYYYYY
BLT.S       00000000000000100000000 1 01101101YYYYYYYY
BGT.S       00000000000000100000000 1 01101110YYYYYYYY
BLE.S       00000000000000100000000 1 01101111YYYYYYYY
BHI.L       00000000000000000100000 2 01100010YYYYYYYY
BLS.L       00000000000000000100000 2 01100011YYYYYYYY
BCC.L       00000000000000000100000 2 01100100YYYYYYYY
BCS.L       00000000000000000100000 2 01100101YYYYYYYY
BNE.L       00000000000000000100000 2 01100110YYYYYYYY
BEQ.L       00000000000000000100000 2 01100111YYYYYYYY
BVC.L       00000000000000000100000 2 01101000YYYYYYYY
BVS.L       00000000000000000100000 2 01101001YYYYYYYY
BPL.L       00000000000000000100000 2 01101010YYYYYYYY
BMI.L       00000000000000000100000 2 01101011YYYYYYYY
BGE.L       00000000000000000100000 2 01101100YYYYYYYY
BLT.L       00000000000000000100000 2 01101101YYYYYYYY
BGT.L       00000000000000000100000 2 01101110YYYYYYYY
BLE.L       00000000000000000100000 2 01101111YYYYYYYY
BRA.S       00000000000000100000000 1 01100000YYYYYYYY
BRA.L       00000000000000000100000 2 01100000YYYYYYYY
BSR.S       00000000000000100000000 1 01100001YYYYYYYY
BSR.L       00000000000000000100000 2 01100001YYYYYYYY
DBHI.L      00000000000000000100000 2 0101001011001CCC
DBLS.L      00000000000000000100000 2 0101001111001CCC
DBCC.L      00000000000000000100000 2 0101010011001CCC
DBCS.L      00000000000000000100000 2 0101010111001CCC
DBNE.L      00000000000000000100000 2 0101011011001CCC
DBEQ.L      00000000000000000100000 2 0101011111001CCC
```

```
DBVC.L      0000000000000000010000000 2 0101100011001CCC
DBVS.L      0000000000000000010000000 2 0101100111001CCC
DBPL.L      0000000000000000010000000 2 0101101011001CCC
DBMI.L      0000000000000000010000000 2 0101101111001CCC
DBGE.L      0000000000000000010000000 2 0101110011001CCC
DBLT.L      0000000000000000010000000 2 0101110111001CCC
DBGT.L      0000000000000000010000000 2 0101111011001CCC
DBLE.L      0000000000000000010000000 2 0101111111001CCC
JMP.        0000100000101010101010000 0 0100111011JJJKKK
JSR.        0000100000101010101010000 0 0100111010JJJKKK
NOP.        0000000000000000000000000 0 0100111001110001
RTR.        0000000000000000000000000 0 0100111001110111
RTS.        0000000000000000000000000 0 0100111001110101
```

CAPITULO VI

DEMOSTRACION DE RESULTADOS

VI DEMOSTRACION DE RESULTADOS

Este capítulo contiene la prueba del sistema mostrando algunos ejemplos de programas en lenguaje ensamblador y su traducción a lenguaje máquina así como un tema relacionado con el manejo del sistema y sus acotaciones, como emplear la sintaxis y los requerimientos para obtener del sistema traductor los resultados satisfactorios.

Acotaciones para traducir programas con el sistema.

a) Todas las instrucciones deberán llevar una extensión de la siguiente forma : .B sí la instrucción es de tamaño byte .W sí la instrucción es de tamaño palabra y .L sí es de tamaño palabra larga. ejemplo :

```
MOVE.L (A0), D0
```

b) En el caso de que las instrucciones se refieran a movimientos entre registros de direcciones o instrucciones inmediatas deberán estar bien definidas, llevando la letra en la instrucción como el siguiente ejemplo :

```
CMPI.B  #0, D0      (en esta instrucción la I)

MOVEA.L  A0,A1      (en esta instrucción la A)
```

c) Cuando una instrucción incluya números decimales deberán ser convertidos previamente a números hexadecimales. ejemplo :

sí la instrucción es MOVE.L #65601, A1

deberá escribirse como MOVE.L #10041, A1

Manejo del sistema traductor

Una vez que se han hecho los cambios necesarios para que las instrucciones sean compatibles con las definidas en el sistema, se procederá a escribirlas en el archivo de datos llamado FUENTE.DAT , tenemos dos opciones para correr el traductor, una que nuestro programa incluya definiciones de MACROS , para lo cual se deberá correr el programa MACROENSAMBLE.EXE primero , posteriormente se deberá correr el programa ENSAMBLE.EXE. La segunda opción es que el programa a traducir no contenga definiciones de MACROS, con lo que bastará correr solamente el programa ENSAMBLE.EXE.

```
MACRO UNO;
MOVEA.B    #10,(A6);
ADDA.W     #1,A6;
MOVE.B     #E,(A6);
ADDA.W     #1,A6;
MOVE.W     #F8,D7;
MOVE.W     #1B,D0;
MEND ;
ADDA.W     #1,A6;
MOVEA.L    #10041,A1;
MOVE.W     #1500,D0;
MOVEA.L    #1000,A0;
UNO;
CMPI.B     #0,D0;
BEQ.S      $00002E;
ADDA.L     #1,A0;
BRA.S      $000010;
MOVE.W     #E4,D7;
TRAP       #E;
UNO ;
LEA.L      $00001500,A0;
MOVE.W     #FD,D7;
TRAP       #E;
NOP;
NOP;
MOVE.W     #8,D7;
TRAP       #E;
MOVE.W     #E4,D7;
TRAP       #E;
NOP;
END;
```

PROGRAMA CON DECLARACION DE MACRO Y SU CORRESPONDIENTE EXPANSION.

```
ADDA.W    #1,A6;
MOVEA.L   #10041,A1;
MOVE.W    #1500,D0;
MOVEA.L   #1000,A0;
MOVEA.B   #10,(A6);
ADDA.W    #1,A6;
MOVE.B    #E,(A6);
ADDA.W    #1,A6;
MOVE.W    #F8,D7;
MOVE.W    #1B,D0;
CMPI.B    #0,D0;
BEQ.S     $00002E;
ADDA.L    #1,A0;
BRA.S     $000010;
MOVE.W    #E4,D7;
TRAP      #E;
MOVEA.B   #10,(A6);
ADDA.W    #1,A6;
MOVE.B    #E,(A6);
ADDA.W    #1,A6;
MOVE.W    #F8,D7;
MOVE.W    #1B,D0;
LEA.L     $00001500,A0;
MOVE.W    #FD,D7;
TRAP      #E;
NOP;
NOP;
MOVE.W    #8,D7;
TRAP      #E;
MOVE.W    #E4,D7;
TRAP      #E;
NOP;
END;
```

EXPANSION.

EJEMPLO 1

```
MOVE.L    #10035,A0;
MOVE.W    #0100,D6;
MOVEP.W    D6,$0000(A0),
MOVE.L    $0504,A0;
ABCD.B    D4,D0;
CMPI.B    #60,D0;
BLTS.S    $000050;
CLR.W     D0;
ABCD.B    D4,D1;
CMPI.B    #60,D1;
BLT.S     $000003C;
CLR.W     D2;
MOVE.W    D2,D5;
LSL.W     #4,D5;
ROR.B     #4,D5;
ORI.W     #3030,D5;
MOVE.W    D5,(A0);
MOVE.W    D1,D5;
LSL.W     #4,D5;
ROR.B     #4,D5;
ORI.W     #3030,D5;
ROR.W     #8,D5;
MOVE.B    D5,3(A0);
MOVE.W    D0,D5;
LSL.W     #4,D5;
ROR.W     #4,D5;
ORI.W     #3030,D5;
MOVE.L    #2500,A5;
MOVE.W    #00F3,D7;
TRAP;
RTR;
NOP;
END;
```

```
A>ENSAMBLE.EXE
000000      207C00010035
000006      3C3C0100
00000A      0D880000
00000E      20780504
000012      C104
000014      0C000060
000018      6D36
00001A      4240
00001C      C304
00001E      0C010060
000022      6D18
000024      4241
000026      C504
000028      0C020018
00002C      6D02
00002E      4242
000030      3A02
000032      E94D
000034      E81D
000036      00453030
00003A      3085
00003C      3A01
00003E      E94D
000040      E81D
000042      00453030
000046      11450004
00004A      E05D
00004C      11450003
000050      3A00
000052      E94D
000054      E81D
000056      00453030
00005A      314500006
00005E      2A782500
000062      3E3C00F3
000066      4E4E
000068      4E77
00006A      4E71

A>
```

EJEMPLO 2

```
MOVEA.L   #100,A5;
MOVEA.W   #2001,A6;
MOVE.W    #F1,D7;
TRAP;
MOVEA.B   #10,(A6);
ADDA.W    #1,A6;
MOVE.B    #E,(A6);
ADDA.W    #1,A6;
MOVE.W    #F8,D7;
MOVE.W    #1B,D0;
TRAP      #E;
MOVE.W    #42,D0;
TRAP      #E;
MOVE.W    #1,D1;
MOVE.W    #2000,SR;
NOP;
NOP;
TST.W     D1;
BNE.S     $00003A;
NOP;
MOVE.W    #2700,SR;
MOVE.W    #E4,D7;
TRAP      #E;
NOP;
MOVE.W    #F3,D7;
TRAP      #E;
MOVE.W    #0,D1;
NOP;
END;
```

```
A>ENSAMBLE.EXE
000000      2A7C00001000
000006      2C7C00001000
00000C      3E3C00F1
000010      4E4E
000012      1CBC000A
000016      DCFC0001
00001A      1CBC000D
00001E      DCFC0001
000022      3E3C00F8
000026      303C001B
00002A      4E4E
00002C      303C002A
000030      4E4E
000032      323C0001
000036      46FC2000
00003A      4E71
00003C      4E71
00003E      4A41
000040      66F8
000042      4E71
000044      46FC2700
000048      3E3C00E4
00004C      4E4E
00004E      4E71
000050      3E3C00F3
000054      4E4E
000056      323C0000
00005A      4E71

A>
```

EJEMPLO 3

```
MOVEA.W   #2000,A5;
MOVEA.L   A5,A6;
MOVEA.L   A5,$2504;
MOVE.L    #0,$2508;
MOVEA.L   #1,D4;
MOVE.W    #F1,D7;
TRAP      #E;
MOVE.W    #71B,(A6)+;
MOVE.W    #2E00,(A6)+;
MOVE.L    #1B3D2B42,-(A5);
SUBI.L    #4,(A5);
MOVE.W    #1B2A,-(A5);
SUBA.L    #2,A5;
MOVE.L    A5,$2500;
MOVEA.L   $2504,A0;
MOVE.W    (A0),D2;
ANDI.W    #0F0F,D2;
LSL.B     #4,D2;
ROR.W     #4,D2;
MOVE.B    3(A0),D1;
LSL.W     #8,D1;
MOVE.B    4(A0),D1;
ANDI.W    #0F0F,D1;
LSL.B     #4,D1;
ROR.W     #4,D1;
MOVE.W    6(A0),D0;
ANDI.W    #0F0F,D0;
LSL.B     #4,D0;
ROR.W     #4,D0;
MOVEA.L   #10025,A0;
MOVE.L    #1E848,D3;
MOVEP.L   D3,$0000(A0);
MOVEQ.W   #-A140,D3;
MOVEA.L   #10021,A0;-
MOVEQ.W    #-A140,D3;
MOVEP.W    D3,$0000(A0);
MOVE.W     #2000,SR;
BRA.S      $00090A;
MOVE.W    #E4,D7;
TRAP      #E;
NOP;
NOP;
```

```
A>ENSAMBLE.EXE
000000      3A7C2000
000004      2C4D
000006      21CD2504
00000A      21FC0000000002508
000012      283C00000001
000018      3E3C00F1
00001C      4E4E
00001E      3CFC071B
000022      3CFC2E00
000026      2B3C1B3D2B42
00002C      049500000004
000032      3B3C1B2A
000036      9BFC00000002
00003C      21CD2500
000040      20782504
000044      3410
000046      02420F0F
00004A      E90A
00004C      E85A
00004E      12280003
000052      E149
000054      12280004
000058      02410F0F
00005C      E909
00005E      E859
000060      30280006
000064      02400F0F
000068      E908
00006A      E858
00006C      207C00010025
000072      263C0001E848
000078      07C80000
00007C      207C00010021
000082      363CA140
000086      07880000
00008A      46FC2000
00008E      60FA
000090      3E3C00E4
000094      4E4E
000096      4E71
000098      4E71

A>
```

EJEMPLO 4

```
LEA.L     $0500,A0;
MOVE.W    #FD,D7;
TRAP      #E;
MOVE.L    A0,$0504;
MOVE.W    #E4,D7;
TRAP      #E;
NOP;
NOP;
MOVE.W    #1,D7;
TRAP      #E;
MOVE.W    #E4,D7;
TRAP      #E;
NOP;
MOVEA.L   #1000,A0;
MOVEA.L   A0,A1;
ADDA.L    #4,A1;
MOVE.W    (A0),D2;
CMP.L     (A1)+,D0;
BLE.S     $00006C;
MOVE.L    -4(A1),(A0);
MOVE.L    D0,-4(A1);
CMPA.L    $2000,A1;
BNE.S     $00005E;
ADDA.L    #4,A0;
MOVEA.L   A0,A1;
CMPA.L    $2000,A0;
BNE.S     $00005E;
RTS;
NOP;
END;
```

```
A>ENSAMBLE.EXE
000000      41F80500
000004      3E3C00FD
000008      4E4E
00000A      21C80504
00000E      3E3C00E4
000012      4E4E
000014      4E71
000016      4E71
000018      3E3C0001
00001C      4E4E
00001E      3E3C00E4
000022      4E4E
000024      4E71
000050      207C00001000
000056      2248
000058      D3FC00000004
00005E      2010
000060      B099
000062      6F0B
000064      20A9FFFC
000068      2340FFFC
00006C      B3F82000
000070      66EC
000072      D1FC00000004
000078      2248
00007A      B1F82000
00007E      66DE
000080      4E75
000082      4E71


A>
```

EJEMPLO 5

```
MOVEA.L   #10001,A0;
MOVE.W    #0,D0;
MOVEP.W   D0,$0000(A0);
MOVEA.L   #10005,A0;
MOVE.W    #-1,D0;
MOVEP.W   D0,$0000(A0);
MOVEA.L   #10009,A0;
MOVE.W    #FF00,D0;
MOVEP.W   D0,$0000(A0);
MOVEA.L   #1000D,A0;
MOVE.W    #2020,D0;
MOVEP.W   D0,$0000(A0);
MOVEA.L   #10001,A0;
MOVE.W    #F00,D0;
MOVEP.W   D0,$0000(A0);
MOVEA.L   #10011,A0;
MOVEA.L   #1000,A1;
MOVE.L    #3,D4;
MOVEP.W   D0,$0000(A0);
JSR.S     $00000078;
NOP;
NOP;
DBLS.L    D4,$00000078;
JMP.S     $0000004C;
NOP;
NOP;
NOP;
NOP;
NOP;
MOVE.L    #3,D2;
MOVE.L    #A5,D3;
NOP;
DBLT.L    D3,$000084;
DBLT.L    D2,$00007E;
RTS;
NOP;
NOP;
END;
```

```
A>ENSAMBLE.EXE
000000      207C00010001
000006      303C0000
00000A      01880000
00000E      207C00010005
000014      303CFFFF
000018      01880000
00001C      207C00010009
000022      303CFF00
000026      01880000
00002A      207C0001000D
000030      303C2020
000034      01880000
000038      207C00010001
00003E      303C0F00
000042      01880000
000046      207C00010011
00004C      227C00001000
000052      283C00000003
000058      01880000
00005C      4EB80978
000060      4E71
000062      4E71
000064      53CCFFF0
000068      4EF8094C
00006C      4E71
00006D      4E71
000070      4E71
000072      4E71
000074      4E71
000076      243C00000003

A>
```

CONCLUSIONES

CONCLUSIONES

En el desarrollo del traductor se consideraron varias etapas, que comprenden desde como definir el marco teórico hasta la puesta en marcha de puntos técnicos en la programación. El marco teórico se logro a través de diferente bibliografía y la asesoría recibida por parte del director de tésis, enfocando la información hacia la arquitectura del microprocesador y conceptos de lenguajes ensambladores, análisis y diseño de sistemas.

En el diseño del traductor se siguieron dos procesos, el primero manejando la información de entrada como procesador de textos y el segundo usado como reconocimiento y conversión de textos a su correspondiente valor de máquina. Para ambos procesos se definieron algoritmos que tomaban su base en autómatas finitos y se genero una tabla con entradas para cada instrucción del microprocesador, que almacena el código objeto, modos de direccionamiento válidos y la longitud de cada instrucción.

Los programas fuentes que el sistema traduce tuvieron que ser acotados a una sintaxis para poderlos interpretar, estos es formateados para las macroinstrucciones, para las instrucciones del procesador y las directivas.

El programa objeto generado se escribe en un archivo, que puede posteriormente conectarse con un cargador para transferirlo directamente hacia el computador educacional.

BIBLIOGRAFIA

Autor:	Phillip R. Robinson.
Obra:	MASTERING THE 68000 MICROPROCESSOR.
Editorial:	Tab books inc. 1985.

Autor:	William Cramer/Gerry Kane.
Obra:	Manual del microprocesador 68000.
Editorial:	McGraw Hill. 1987

Autor:	Steve Wood.
Obra:	Turbo pascal.
Editorial:	McGraw Hill. 1987

Obra:	Introducción a los microprocesadores usando el MC 68000.
Editorial:	Editado por la Facultad de Ingeniería. 1989

Autor:	Lance A. Levental.
Obra:	6800 Assembly Language Programming.
Editorial:	Osborne/McGraw Hill. 1980

Autor:	John J. Donovan.
Obra:	System Programming.
Editorial:	McGraw Hill. 1987