ESCUELA DE PREPARACIÓN DE OPOSITORES
E. P. O.
C/. La Merced, 8 – Bajo A Telf.: 968 24 85 54
30001 MURCIA

INF33 – SAI33

Programación en lenguaje ensamblador. Instrucciones básicas. Formatos. Direccionamientos.

Esquema.

1 Introducción.

Este tema presentamos las ideas básicas sobre la programación en lenguaje ensamblador. Este lenguaje está totalmente ligado a la estructura del computador, aunque nosotros abordaremos el estudio del tema sin hacer referencia directa a una máquina específica.

Aunque los lenguajes ensambladores son distintos de un computador a otro, la mayor parte de ellos sigue una filosofía concreta, por lo que es de una gran utilidad seguir una terminología común; esto es lo que pretende el estándar IEEE 694 de ensamblador para microprocesadores. Por tanto, para el desarrollo del tema vamos a seguir dicho estándar.

En el desarrollo de este tema, presentaremos el formato de las instrucciones en ensamblador, así como las instrucciones básicas, para concluir nuestro estudio con los diferentes modos de direccionamiento, que indican las formas en las que la CPU capta los operandos o almacena los resultados.

2 Programación en lenguaje ensamblador.

2.1 Programación en lenguaje máquina.

La programación en lenguaje máquina consiste en definir, en forma binaria, octal o hexadecimal, los códigos y direcciones de las instrucciones necesarias para resolver el problema propuesto.

Veamos con un ejemplo, empleando los códigos del Z-80, cómo se desarrolla este tipo de programación. Supongamos que se quieren sumar tres números de 1 octeto cada uno, almacenados en las posiciones $4F35_{16}$, $4F36_{16}$ y $4F37_{16}$ de la memoria. El resultado deberá dejarse en la posición $4FFC_{16}$. Supondremos, además, que los registros de la máquina están libres y se pueden usar sin preservar su contenido.

De entre las posibles soluciones, se selecciona la compuesta por los siguientes pasos:

- Inicializar el doble registro HL con $4F35_{16}$. (HL $\leftarrow$ $4F35_{16}$)
- Cargar el primer dato en el acumulador. (A $\leftarrow$ M(HL))
- Incrementar HL. (HL $\leftarrow$ HL + 1)
- Sumar el segundo dato con el acumulador. (A $\leftarrow$ A + M(HL))
- Incrementar HL. (HL $\leftarrow$ HL + 1)
- Sumar el tercer dato con el acumulador. (A $\leftarrow$ A + M(HL))
- Transferir el acumulador (que contiene la suma deseada) a la posición de memoria $4FFC_{16}$. ($4FFC_{16}$ $\leftarrow$ A)

De las tablas de instrucciones del Z-80 se pueden obtener los códigos en hexadecimal de las instrucciones deseadas. El resultado es el siguiente:

HL ← 4F3516	21354F
A ← M(HL)	7E
HL ← HL+1	23
A ← A+M(HL)	86
HL ← HL+1	23
A ← A+M(HL)	86
$M(4FFC_{16})$ ← A	32FC4F

Ahora bien, observemos que, en dicho programa, se ha supuesto que las operaciones se hacen correctamente sin desbordamiento. Ello no siempre será así, por lo que se deben establecer los medios para detectar esta situación, y poder avisar al usuario o tomar la decisión pertinente. Para ello, se supondrá que los números están representados en complemento a dos y que existe un programa para tratar el error, programa que empieza en la posición de memoria $0CA3_{16}$. Se deberá intercalar, después de cada suma, una bifurcación condicional a la posición $0CA3_{16}$. Dado que el Z80 indica la condición de desbordamiento en complemento a 2 ó 1 poniendo a 1 el bit de estado P/V, esta bifurcación deberá hacerse empleando una instrucción de bifurcación condicional sobre el bit mencionado. Si PV = 1, se hace PC ← $0CA3_{16}$, por lo que se cumple el objetivo deseado.

El programa contenido a partir de la dirección $0CA3_{16}$, deberá tratar el desbordamiento, no entrando nosotros aquí en su desarrollo.

El ejemplo, por tanto, quedaría como sigue:

HL ← $4F35_{16}$	21354F
A ← M(HL)	7E
HL ← HL+1	23
A ← A+M(HL)	86
Si PV=1; PC ← $0CA3_{16}$	EAA30C
HL ← HL+1	23
A ← A+M(HL)	86
Si PV=1; PC ← $OCA3_{16}$	EAA30C
$M(4FFC_{16})$ ← A	32FC4F

Se observa lo incómodo de trabajar así. Por un lado, hay que saberse los códigos de las instrucciones (o andar mirando en las tablas), siendo muy fácil equivocarse en algún número, con lo que se puede cambiar totalmente la operación deseada. Por otro lado, hay que tener en cuenta la posición exacta de las instrucciones y de los datos.

Notemos además, que aparte de la incomodidad de tener que llevar esta cuenta, cualquier modificación en el programa o los datos puede obligar a tener que correr de sitio datos o programas. En este caso, habrá que rehacer todas las direcciones de operandos y de bifurcaciones.

2.2 Lenguaje ensamblador.

El lenguaje ensamblador surge como una herramienta para simplificar la programación en instrucciones de máquina. Esta simplificación tiene dos aspectos fundamentales:

- El empleo de códigos mnemónicos, para representar las instrucciones.
- El empleo de nombres simbólicos, para designar a los datos y a las referencias.

De esta forma, se libera al programador de tener que especificar posiciones de memoria y de recordar los códigos de cada instrucción.

Por desgracia, todos los códigos mnemónicos empleados en los ensambladores se refieren a palabras inglesas. A pesar de ello, resultan más cómodos que los códigos máquina correspondientes. A título de ejemplo, la sentencia `ADD .1,.3` indica sumar el registro 1 con el 3, dejando el resultado en el registro 1. Evidentemente, es más fácil recordar `ADD` que su código máquina equivalente, que en el caso del Z-80 es 8A, considerando que el registro 1 es el A y que el registro 3 es el D.

Por otro lado, es más simple y menos propenso a errores referirse a una variable por un nombre que por su dirección en memoria. Así se podrá escribir `LD .5, #PESO` para indicar que se desea cargar (*load*) la dirección de la variable PESO en el registro 5. En código máquina del Z-80, la carga del valor $4A35_{16}$ en el doble registro HL se debería haber escrito, en hexadecimal, como: 21354A.

2.3 Programación versus codificación.

Es importante diferenciar entre el proceso de programación y el de codificación.

El proceso de programación debe ser, en principio, independiente del lenguaje empleado para codificar, aunque en la realidad se ve influido por las técnicas de programación que dicho lenguaje propicie. En concreto, para al lenguaje ensamblador, el juego de instrucciones permitido depende del computador empleado, por lo que, las operaciones posibles así como los modos de direccionamiento disponibles, favorecerán ciertas estructuras de datos y de control.

El proceso de programación consiste en convertir las especificaciones o descripción del problema, redactados en lenguaje natural, en un algoritmo específico para resolver el problema. En este proceso deberán establecerse las estructuras de datos, las informaciones de entrada y de salida empleadas, así como el flujo de control dentro del programa.

Por su parte, la codificación consiste en la conversión de estos pasos en sentencias del lenguaje empleado. La codificación es una actividad más bien monótona pero imprescindible. Evidentemente, el lenguaje empleado impondrá un refinado determinado. El ensamblador, al ser un lenguaje al nivel de máquina, requerirá el mayor nivel de detalle en el refinado.

Finalmente, hay que destacar que el proceso de codificación en ensamblador conlleva dos partes principales. Una consiste en la asignación de posiciones específicas a los datos. La otra es la traducción propiamente dicha de los pasos del problema en instrucciones elementales del computador. Es corriente ir redactando ambas partes en documentos separados, pues la primera regla de programación es no mezclar datos con instrucciones. Cuando se finaliza el proceso, se juntan ambas partes para obtener el programa completo. Algunos programadores tienen la costumbre de poner primero el código y luego los datos, mientras que otros lo hacen al revés. En algunos casos puede ser más eficaz poner primero los datos, puesto que así se simplifican las tablas de referencia que ha de generar el ensamblador al ensamblar el programa.

2.4 Programación en ensamblador.

Las técnicas de desarrollo de software tienen su óptima aplicación con ciertos lenguajes de alto nivel, como puede ser el Pascal, el Modula o el ADA. El lenguaje ensamblador es demasiado permisivo, en el sentido que deja salirse fácilmente de las

normas aplicables en los métodos anteriores, por lo que es muy frecuente que los programadores no se restrinjan a ellas.

Conviene, sin embargo, resaltar que una técnica muy útil a la hora de programar en ensamblador es la siguiente:

- Primero se programa el problema en un lenguaje de alto nivel, que se adapte bien a las técnicas modernas de desarrollo de programas, usando, evidentemente, dichas técnicas.
- Seguidamente, se traduce este programa, escrito en lenguaje de alto nivel, a ensamblador.

Esta técnica permite aplicar fácilmente las metodologías modernas de desarrollo de programas y, una vez comprobado el programa, traducir a ensamblador todo el problema o solamente aquellas partes que, por su repetición o extensión, así lo justifiquen.

En todo caso, conviene seguir las fases de todo desarrollo software y que son las siguientes:

- Definición del problema.
- Diseño del programa.
- Codificación.
- Corrección y verificación.
- Prueba y validación.
- Documentación.

Finalmente, queremos recordar los principios generales comunes a cualquier metodología de desarrollo de programas, cuya aplicación ayuda al buen diseño:

- Avanzar en pasos pequeños. No tratar de resolver mucho al mismo tiempo.
- Dividir los trabajos en partes lo más independientes posibles y que se puedan programar y verificar independientemente.
- Establecer un flujo de control lo más sencillo posible.
- Emplear esquemas gráficos. Son fáciles de visualizar y ayudan a detectar errores.
- Buscar las soluciones sencillas y claras. Si luego es necesario mejorar las prestaciones del sistema o reducir la memoria usada, cambiarlas por otras más complejas pero más eficaces.
- Ser lo más sistemático posible.
- Emplear métodos conocidos y probados.
- Establecer, desde el principio, los mecanismos para facilitar la verificación y mantenimiento del programa.
- Usar una terminología coherente y sistemática.
- No empezar a codificar hasta que el problema esté totalmente definido. De esta forma se pueden ahorrar modificaciones difíciles.
- Documentar el diseño a medida que se va realizando.

- Diseñar el programa de forma que los cambios más probables sean fáciles de introducir.

3 Formato de las instrucciones.

Existen dos clases de sentencias en ensamblador: las sentencias propiamente dichas, que corresponden a instrucciones máquina, y las pseudoinstrucciones, que son órdenes al programa ensamblador que se encarga de traducir el programa a código máquina. Las pseudoinstrucciones se ejecutan en tiempo de ensamblador, mientras que las instrucciones se traducen el código objeto.

El formato típico de ensamblador ocupa una línea por sentencia, incluyendo los siguientes campos:

```
Etiqueta Código Operandos Comentarios
```

Los caracteres que separan los distintos campos de la sentencia se denominan delimitadores, siendo éstos el blanco, la coma y el punto y coma. Normalmente, se emplean blancos para separar la etiqueta del código y para separar el código de los operandos. Los operandos se suelen separar mediante comas, mientras que el carácter punto y coma (;) se emplea para especificar el comienzo del campo de comentarios.

No suele estar permitido la inclusión de dos sentencias en una misma línea, ni que una sentencia exceda de una línea.

3.1.1 Etiqueta.

El campo de etiqueta es potestativo, y se empleará cuando se necesite. Sirve para identificar la instrucción o pseudoinstrucción especificada en esa línea. En realidad es equivalente a la dirección donde se ubique dicha instrucción o a la posición de memoria reservada por una pseudoinstrucción. Un caso típico de empleo de etiquetas es en los saltos o bifurcaciones, sirviendo para indicar a qué instrucción se quiere bifurcar. Otro caso típico es en la identificación de los operandos, expresados como posiciones de memoria reservadas por pseudoinstrucciones.

Generalmente, las etiquetas deben empezar por una letra y suelen estar limitadas a 8 ó 10 caracteres. Para que el ensamblador pueda reconocer la existencia de una etiqueta, ésta deberá comenzar en la primera posición de la línea o deberá terminar con el carácter dos puntos (:).

3.1.2 Código de la instrucción.

El segundo campo contiene el código mnemónico de la instrucción que se desea.

3.1.3 Operandos.

Después del código de la instrucción van los operandos, que deberán especificar la dirección física (esto es, el registro o la posición en memoria) o el dato propiamente dicho. El separador entre operandos suele ser la coma.

En las instrucciones con dos operandos, el operando destino suele ser el primero de los especificados (en el estándar IEEE 694 solamente se rompe esta regla con las instrucciones `MOVE`, `OUTPUT` y `STORE`, que llevan la dirección del destino como segundo operando).

Los operandos se pueden expresar mediante:

- Un nombre de un registro.
- Un nombre simbólico.
- Unos caracteres alfanuméricos.
- Una cantidad numérica.
- Una expresión.

En los dos últimos casos, se pueden expresar las cantidades numéricas en binario, octal, hexadecimal o decimal. Para diferenciar entre ellas se les añade la letra B para el binario, O ó Q para el octal, H para hexadecimal y D para decimal. En caso de no ponerse prefijo, se suele suponer que la cantidad está expresada en decimal.

Para los operandos expresados por cadenas de caracteres, éstos deberán encerrarse entre comillas dobles. Los nombres simbólicos pueden ser etiquetas, por lo que se refieren a la dirección de la instrucción o del dato especificado por la etiqueta.

Los operandos se pueden establecer mediante una expresión aritmético-lógica, con elementos numéricos o simbólicos, cuyo resultado sea el operando deseado. En general, estas expresiones deben ser estáticas, por lo que sus elementos simbólicos deben restringirse a etiquetas. Un posible error consistiría en tratar de utilizar, en estas expresiones, contenidos de registros y/o de posiciones de memoria, puesto que sus valores no están definidos en el momento en el que el ensamblador ha de calcular su valor.

Los operadores considerados por el estándar IEEE 694, para establecer las expresiones, son los siguientes:

operador	función
+	Suma
-	Resta
*	Multiplicación
/	División
&	AND lógico
~	OR exclusivo
|	OR lógico
-	Signo negativo
~	Inversor lógico

Sin embargo, la mayoría de los ensambladores disponen de más operandos. Una lista típica puede ser la siguiente:

operador	función	Prioridad
+	Signo positivo	1
-	Signo negativo	1
.NOT. o ~	Inversor lógico	1
.RES.	Resultado	1
**	Exponenciación	2
*	Multiplicación	3
/	División	3
.MOD.	Módulo	3
.SHR.	Desplazamiento derecha lógico	3
.SHL.	Desplazamiento izquierda lógico	3
+	Suma	4
-	Resta	4
.AND. o &	AND lógico	5
.OR. o |	OR lógico	6
.XOR. o ~	OR exclusivo	6
.EQ. o =	Igual	7
.GT. o >	Mayor que	7
.LT. o <	Menor que	7
.UGT.	Sin signo mayor que	7
.ULT.	Sin signo menor que	7

Los paréntesis se pueden incluir en las expresiones para especificar el orden en que se aplican los operadores. Sin embargo, es de destacar que, en muchos ensambladores, la expresión no debe tener blancos, pues se tomarían como delimitadores (este no es el caso del IEEE 694, puesto que utiliza siempre la coma como delimitador entre los operandos).

Suele existir un símbolo para designar la dirección de la instrucción en curso, En muchos ensambladores este símbolo es el `$`, pero el IEEE 694 emplea `*` para esta función. Por ejemplo, `*+10D` indica una dirección 10 posiciones mayor que la dirección de la instrucción donde se incluye esta expresión. Por su lado, la instrucción `BR *` supone un bucle infinito de ella misma.

Notemos que las expresiones van dirigidas al programa ensamblador. Este programa, tomando el valor numérico de los símbolos de la expresión (si es que hay símbolos), calcula su valor numérico en binario y la sustituye por éste.

3.1.4 Comentarios.

Finalmente, en el campo de comentarios se pueden (y deben) escribir los comentarios explicativos del proceso que se realiza. No es obligatorio, aunque sí muy recomendado, comentar los programas. En cualquier caso, el programa ensamblador los ignora. Los comentarios se suelen especificar mediante el punto y coma (;), pudiéndose emplear líneas que sólo contengan comentarios.

4 Instrucciones básicas.

Las instrucciones del ensamblador se refieren a los códigos mnemónicos de las instrucciones de máquina que tenga un computador. En general, cada fabricante tiene sus propios mnemónicos, adaptados a las instrucciones de máquina de sus computadores. Sin embargo, existe una tendencia a uniformar, por lo que los estándares son cada vez más importantes.

El juego de instrucciones de un computador debe cumplir dos condiciones: debe ser completo y debe ser eficaz. Que un juego de instrucciones sea completo significa

que con él se puede calcular, en un tiempo finito, cualquier tarea computable. Esta condición es, evidentemente, necesaria pero no suficiente, puesto que, además, el juego de instrucciones ha de ser eficaz, esto es, ha de permitir una alta velocidad de cálculo, sin exigir a cambio una excesiva complicación de la unidad de control ni de la unidad aritmética.

Una función f(x) se dice que es computable si puede ser calculada en un número finito de pasos por una máquina de Turing. Aunque un computador no tiene una memoria infinita, como la postulada en la máquina de Turing, puede emplearse para evaluar prácticamente cualquier función computable.

Desde este punto de vista, el juego de instrucciones puede ser realmente sencillo. La máquina de Turing utiliza solamente las cuatro instrucciones de escribir, mover a la izquierda una posición y leer, mover a la derecha una posición y leer, y parar.

Del mismo modo se pueden construir computadores con juegos de instrucciones muy reducidos. Por ejemplo, está demostrado que las dos instrucciones siguientes: decrementar y bifurcar si cero, e incrementar, forman un juego de instrucciones completo, con el que se puede resolver cualquier problema resoluble en otros computadores.

Aunque el estudio del juego mínimo completo tiene un interés teórico, no cabe duda que, desde el punto de vista de los computadores comerciales, lo que tiene interés es que sean rápidos y económicos, por lo que habrá que dotarles de un conjunto bien seleccionado de instrucciones. La selección del juego de instrucciones de un computadores, por tanto, uno de los puntos más críticos de su diseño.

A continuación, presentaremos las instrucciones consideradas por el estándar IEEE 694.

4.1 Instrucciones de transferencia de datos.

Las instrucciones de transferencia de datos permiten repetir, en el operando destino, la información almacenada en el operando origen, quedando este último sin modificar. Destino y origen pueden ser registros o posiciones de memoria. En general, no modifican los biestables de estado del computador.

Estas instrucciones suelen transferir una palabra aunque existen algunas que transfieren fracciones de palabras y otras que transfieren bloques completos.

Las denominaciones más frecuentes son las que se detallan a continuación:

MOVE	Transfiere el contenido de un registro a otro, o de una posición de memoria a otra. A veces se denomina TRANSFER.
STORE	Transfiere el contenido de un registro a la memoria.
LOAD	Transfiere el contenido de una posición de memoria a un registro. Es la operación inversa al STORE.
MOVE BLOCK	Transfiere un bloque de datos.
MOVE MULTIPLE	Copia el contenido del origen en múltiples posiciones de memoria.
EXCHANGE	Intercambia el contenido de los operandos especificados.
CLEAR	Pone a "ceros" el destino. A veces se denomina RESET.
SET	Pone a "unos" el destino.
PUSH	Transfiere el origen a la cabecera de la pila.
POP	Transfiere la cabecera de la pila al destino.

4.2 Instrucciones que modifican la secuencia del programa.

Las instrucciones de modificación de secuencia de programa permiten alterar la secuencia normal de ejecución del mismo. De forma genérica se dice que son instrucciones de salto o bifurcación, puesto que, en vez de pasar a la instrucción que ocupa la posición siguiente, "saltan" a ejecutar las instrucciones que se encuentran en otra posición de memoria.

La secuencia normal de ejecución de un programa se basa en el contador de programa PC, que es incrementado para pasar de la dirección de una instrucción a la de la siguiente. En concreto, si la instrucción en curso tiene un tamaño de Z palabras, se debe hacer la operación: PC ← PC+Z, para pasar a la siguiente instrucción. Si se transfiere o "carga" en el registro PC una nueva dirección X (PC ← X); la siguiente instrucción que se ejecutará será la que ocupe esta posición X, por lo que se habrá bifurcado a la dirección X.

Aunque una instrucción de bifurcación no es más que una transferencia en la que el destino es el contador de programa, su trascendencia y alternativas hace que se deban considerar de forma independiente. Existen distintos tipos de bifurcaciones, que se detallarán a continuación.

Con mucha frecuencia la dirección de bifurcación es un valor absoluto, por lo que debe usarse la notación /DIR, de acuerdo al estándar IEEE 694.

4.2.1 Bifurcaciones condicionales.

Las bifurcaciones condicionales son instrucciones que tienen dos secuencias distintas: cuando no se cumple la condición de bifurcación, no hacen nada y PC←PC+Z; cuando sí se cumple la condición de bifurcación, modifican el PC, que recibe la dirección de bifurcación.

Las condiciones de bifurcación se establecen sobre los biestables de estado, biestables que almacenan ciertas condiciones sobre las operaciones realizadas con anterioridad. Estas condiciones pueden hacerse sobre un sólo biestable o sobre varios simultáneamente.

El estándar IEEE 694 considera las siguientes condiciones, que reproducimos en inglés con sus códigos mnemónicos. La instrucción de bifurcación se ejecutará si la condición especificada es cierta:

ZERO (Z)	Cero
NOT ZERO (NZ)	No cero
EQUAL (E)	Igual
NOT EQUAL (NE)	Desigual
CARRY (C)	Acarreo
NOT CARRY (NC)	Sin acarreo
POSITIVE (P)	Positivo
NOT POSITIVE (NP)	No positivo
NEGATIVE (N)	Negativo
NOT NEGATIVE (NN)	No negativo
OVERFLOW (V)	Desbordamiento
NO OVERFLOW (NV)	Sin desbordamiento
GREATER THAN (GT)	Mayor que
GREATER THAN OR EQUAL (GE)	Mayor o igual
LESS THAN (LT)	Menor que
LESS THAN OR EQUAL (LE)	Menor o igual
HIGHER (H)	Superior que (sin signo)
NOT HIGHER (NH)	No superior (sin signo)
LOWER (L)	Inferior que (sin signo)
NOT LOWER (NL)	No inferior (sin signo)
TRUE (T)	Verdadero
FALSE(F)	Falso
PARITY EVEN (PE)	Paridad par
PARITY ODD (PO)	Paridad impar

Aunque no es muy frecuente, puede construirse un computador sin biestables de estado. Evidentemente, en este caso las instrucciones de bifurcación condicional no se pueden hacer sobre condiciones anteriores. La propia instrucción de bifurcación ha de generar la condición, por lo que ha de estar compuesta por una comparación y la bifurcación condicional propiamente dicha.

4.2.2 Bifurcaciones con retorno.

Las instrucciones de bifurcación con retorno salvaguardan la dirección de la instrucción que ocupa la posición siguiente (esto es, salvaguardan el valor PC+Z). De esta forma, se puede retomar al punto donde se bifurcó y seguir ejecutando en la dirección siguiente a la que causó el salto.

El uso más frecuente de la bifurcación con retorno es para llamar a subrutinas (instrucción que suele llamarse CALL o BRANCH TO SUBROUTINE).

Muchas máquinas disponen de bifurcaciones con retornos condicionales, de forma que el salto a la subrutina sólo se hace si se cumple una cierta condición.

Uno de los problemas clásicos de las bifurcaciones con retorno es la selección del lugar donde se salvaguarda la dirección de retorno. Esta dirección no es más que uno de los parámetros que se deben enviar a una subrutina:

1. Salvaguarda en un registro especial. La solución es sencilla, pero no permite llamadas anidadas, a menos que este registro sea a su vez salvaguardado, lo que complicaría el tratamiento.
2. Salvaguarda en un registro general. Es una solución similar a la anterior con el inconveniente adicional de ocupar uno de los pocos registros generales del computador.
3. Almacenamiento en la subrutina. Por ejemplo, se puede reservar la primera palabra de la subrutina para almacenar la dirección de retorno. El

procedimiento permite llamadas anidadas de cualquier nivel, pero no así las recursivas, pues se destruirían las posiciones de retorno anteriores.

4. Almacenamiento en una pila de control. El computador debe mantener una pila de control donde se almacenan las direcciones de retorno. Este método permite llamadas anidadas de cualquier nivel, así como llamadas recursivas. En efecto, cada nueva llamada va introduciendo en la pila su dirección de retorno, sin destruir las anteriores. Dado que los retornos se hacen en orden inverso a las llamadas, la pila siempre tiene en la cabecera la dirección de retorno adecuada.

4.2.3 Instrucciones de bifurcación más comunes.

BRANCH	Bifurcación que puede ser condicional o incondicional. También se suele llamar JUMP. A veces se combina con el incremento o decremento automático de un registro, lo que es muy conveniente para formar bucles, puesto que engloba las dos operaciones fundamentales para su construcción.
CALL	Bifurcación con retorno, que se emplea para llamar a una subrutina. Puede ser condicional o incondicional. También recibe el nombre de JUMP TO SUBROUTINE o de BRANCH AND LINK.
RETURN	Instrucción complementaria del CALL, restituye la dirección del programa llamante. En algunas máquinas puede ser también condicional. Suele existir un caso especial, el RETURN FROM INTERRUPT, que sirve para volver de las interrupciones.
SKIP	Es una bifurcación condicional especial muy compacta que sólo salta una instrucción, por lo que no necesita contener la dirección de bifurcación. Si la condición de la instrucción se cumple, el contador de programa se vuelve a incrementar, con lo que se salta la instrucción siguiente. Se emplea en unión a un BRANCH incondicional para construir bucles.
RETURN WITH SKIP	Esta instrucción incrementa la dirección de retorno antes de realizar éste. Por los tanto, no retorna a la instrucción siguiente a la llamada, sino a alguna instrucción posterior.

4.3 Instrucciones aritméticas.

Las instrucciones que encontramos en el grupo de las aritméticas afectan a los bits de estado. Las denominaciones más corrientes se listan a continuación.

ADD	Suma.
SUBTRACT	Resta.
INCREMENT	Incrementa.
DECREMENT	Decrementa.
MULTIPLY	Multiplica.
DIVIDE	Divide.
NEGATE	Cambia de signo.
ABSOLUTE	Valor absoluto.
ADD WITH CARRY	Suma, añadiendo el acarreo del resultado anterior. Es conveniente para operaciones en precisión múltiple.
SUBTRACT REVERSE	Resta en orden inverso.
SUBTRACT WITH BORROW	Resta, teniendo en cuenta el acarreo de resta de la operación anterior. Se usa en precisión múltiple.

4.4 Instrucciones de comparación.

La operación de comparación COMPARE consiste en restar o en hacer la operación XOR de cada bit de dos o más operandos. No se almacena el resultado, pero sí se modifican, según proceda, los biestables de estado.

Suele ir precediendo a una bifurcación condicional, que interrogará el valor de alguno de los biestables de estado que modifique la comparación. La operación TEST es una comparación con cero.

4.5 Instrucciones lógicas.

Las instrucciones lógicas, más corrientes, son AND, OR, NOT, y XOR. Conviene recordar que las operaciones lógicas se realizan en cada uno de los bits de los operandos de forma independiente y que modifican los bits de estado.

4.6 Instrucciones de desplazamiento.

Las operaciones de desplazamiento modifican los bits de estado, siendo las denominaciones más corrientes las que se detallan a continuación.

SHIFT	El SHIFT puede ser a derechas o izquierdas, lógico o aritmético.
ROTATE	La rotación puede ser a derechas o izquierdas, Además, puede incluirse al bit de acarreo en la rotación.

4.7 Instrucciones de bit.

El grupo de las instrucciones de bit incluye las tres instrucciones siguientes: BIT TEST, BIT SET, y BIT CLEAR.

4.8 Instrucciones de entrada/salida y misceláneas.

Las instrucciones de entrada/salida son en realidad instrucciones de transferencia, con la peculiaridad de que el destino o el origen es un registro de un periférico. Muchos computadores no tienen este tipo de instrucciones, realizándose estas funciones con las instrucciones de LOAD, STORE y/o MOVE. Las denominaciones más corrientes son las siguientes:

INPUT	Transfiere la información de un puerto de entrada a un registro o a memoria. A veces se denomina READ.
OUTPUT	Operación inversa de INPUT. A veces se denominan WRITE.

Existen una serie de instrucciones que no se pueden clasificar en ninguna de las categorías anteriores y que llamaremos misceláneas. Entre ellas se pueden destacar las siguientes:

WAIT	Esta instrucción para la ejecución del programa, hasta que se reciba una interrupción externa.
HALT	Esta instrucción para el procesador.
CONVERT	Tiene por objeto cambiar los formatos de las instrucciones, generalmente, para realizar operaciones de entrada/salida. También se utiliza para hacer extensión de signo.
NO OPERATION	No realiza ninguna operación. Se puede utilizar para llenar huecos en los programas o para temporizar esperas.

5 Direccionamientos.

Un modo de direccionamiento es un procedimiento que permite determinar un operando, o la ubicación de un operando o una instrucción. Dado que, generalmente, lo que se especifica es la dirección donde se almacena el dato o la instrucción, la denominación genérica de modo de direccionamiento queda justificada.

Llamaremos objeto a la instrucción, operando o resultado que se desea direccionar. El objeto puede residir en la propia instrucción, en un registro o en la

memoria principal, siendo el objetivo de los modos de direccionamiento especificar el lugar concreto donde se encuentra.

Aunque, a primera vista, parecería lo más conveniente que la instrucción incluyese directamente el objeto o su dirección real, ello puede no ser lo más indicado, como apuntan razones tales como:

1. Ahorro de espacio. Mientras más cortas sean las instrucciones, menos almacenamiento ocupan los programas y menos bits hay que leer de memoria principal para ejecutar un programa. En este sentido, serán convenientes direccionamientos que ocupen poco espacio.
2. Código reubicable y reentrante. El código reubicable (que se puede ejecutar en cualquier zona de memoria) y el código reentrante (que puede ser invocado desde varios puntos simultáneamente), exigen direccionamientos relativos.
3. Estructuras de datos. El manejo de las estructuras de datos, tales como tablas, matrices, colas, listas, ..., se simplifica con el empleo de algunos modos de direccionamiento relativos.

En la siguiente tabla vemos una posible clasificación de los modos de direccionamiento.

Inmediato			
Directo	Absoluto	De registro	
		De memoria	
		De página base	
	Relativo	Al contador de programa	
		A un registro	
		A un registro índice	Postautodecremento
			Preautodecremento
			Postautoincremento
			Preautodecremento
		A pila	
Indirecto			
Implícito			

5.1 Direccionamiento inmediato.

El direccionamiento se llama inmediato cuando el objeto, en este caso un operando, se encuentra contenido en la propia instrucción. Ejemplos:

162A D ← 2A es una instrucción del Z80 con direccionamiento inmediato, puesto que contiene el valor 2A del operando.

927C47B8 M(X) ← 7C es una instrucción del IBM 370 con direccionamiento inmediato, pues el operando que se transfiere a la posición de memoria X es 7C, que se encuentra en la propia instrucción (la dirección de memoria X se calcula, con la información 47B8, mediante un direccionamiento relativo con registro base, como se verá más adelante).

Hay máquinas que permiten distintos tamaños de operandos inmediatos. Por ejemplo, el VAX permite inmediatos de 6, 8, 16, 32 y 64 bits. Con ello se pretende reducir la memoria necesaria, adaptando la instrucción al tamaño de dato deseado.

5.2 Direccionamiento directo absoluto.

Un direccionamiento se llama directo, en contraposición con el indirecto, cuando expresa la dirección real del objeto. Por otro lado, el direccionamiento absoluto indica que la instrucción contiene una dirección efectiva sin compactar. Por tanto, el direccionamiento directo absoluto indica que la instrucción contiene la dirección real, sin compactar, del objeto.

Este tipo de direccionamiento presenta tres alternativas:

1. La información contenida en la instrucción puede ser el identificador de un registro, cuando el objeto deseado se encuentra almacenado en ese tipo de elemento. En algunos textos se considera este caso como otro tipo de direccionamiento que se denomina direccionamiento de registro.
2. La información contenida en la instrucción es una dirección completa de memoria principal. El número de bits necesarios para ello depende del mapa de direcciones del computador. Por ejemplo, el Z80, con un mapa de 64 Koctetos, requiere 16 bits, mientras que el VAX, con un mapa virtual de 4 Goctetos, requiere 32 bits.
3. La información contenida en la instrucción es una dirección limitada, que permite referirse solamente a una parte del mapa de memoria. De esta forma, se reduce el tamaño de la instrucción, pero con la mencionada limitación del mapa de memoria. Este tipo de direccionamiento suele llamarse direccionamiento de página base y es muy frecuente en los microprocesadores.

Ejemplos:

3A35F7 A ← M(F735). Instrucción del Z80 que transfiere al registro A el contenido de la posición F735.

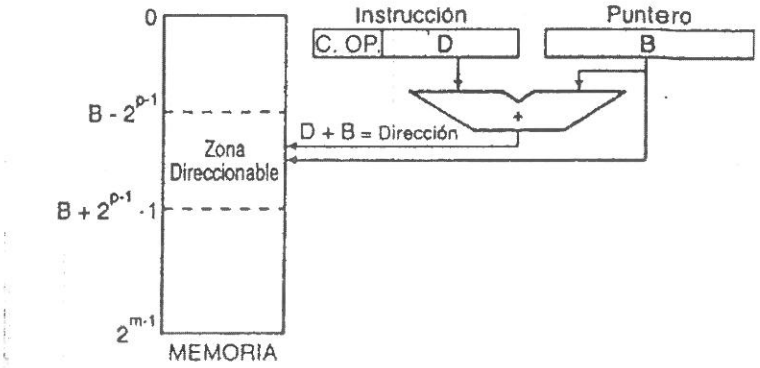
D08F B ← B - M(8F). Instrucción del MC6800 que resta al acumulador B el contenido de la posición 8F. Siendo 8F una dirección absoluta de página base, que en este microprocesador se reduce a las direcciones 0 a 255.

5.3 Direccionamiento directo relativo.

En el direccionamiento directo relativo la instrucción no contiene la dirección del objeto, sino un desplazamiento D sobre una dirección marcada por un puntero. La dirección se calcula sumando el desplazamiento D al puntero de referencia, que suele estar almacenado en un registro.

Este tipo de direccionamiento suele necesitar menos bits que el directo absoluto, puesto que el desplazamiento D puede tener bastantes menos bits que los que exige el mapa de direcciones. Notemos que el registro que sirve de puntero puede ser del tamaño deseado. Por ello, este direccionamiento es más compacto, pero requiere, en contrapartida, realizar la operación de suma del puntero más el desplazamiento.

La mayoría de los computadores permiten desplazamientos positivos y negativos. De esta forma, se puede alcanzar una zona de memoria principal alrededor de la posición marcada por el puntero. La siguiente figura representa la zona alcanzada para el caso de un desplazamiento de p bits representado en complemento a 2. En dicha figura se ha supuesto, como es habitual, que el mapa de direcciones tiene m bits y es mayor que el rango del desplazamiento, eso es, que $m > p$.



Evidentemente, mientras mayor sea la longitud p del desplazamiento D, mayor será el campo direccionable para un puntero determinado, pero más larga será la instrucción. Por ejemplo, el VAX permite desplazamientos de 8, 16 y 32 bits.

Aunque pueda parecer que la suma requerida por este direccionamiento debe retardar mucho la ejecución de la instrucción que lo emplea, lo cierto es que no es así.

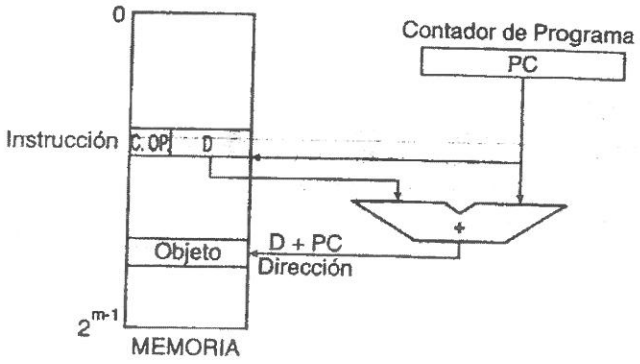
El interés de este tipo de direccionamiento se fundamenta en que es la base del código reentrante y reubicable, puesto que permite cambiar las direcciones de datos y de bifurcaciones sin más que cambiar el contenido de un registro. Además, algunas técnicas de protección de memoria también residen en este direccionamiento. Finalmente, también permite recorrer de forma eficaz las estructuras de datos.

Existen distintas posibilidades en cuanto al registro que se emplea como puntero y en cuanto al tratamiento que sufre este último. Veamos seguidamente estas alternativas.

5.3.1 Direccionamiento relativo al contador de programa PC.

En el direccionamiento relativo a contador de programa, como su nombre indica, el puntero empleado es el contador de programa PC, esto es, el registro que almacena la dirección de la instrucción que se va a ejecutar.

Puesto que, normalmente, el contador de programa se incrementa al tiempo que se lee cada instrucción, la posición de referencia es la de la instrucción siguiente, tal y como indica la siguiente figura. Por ejemplo, si la instrucción en curso con direccionamiento relativo a PC, está almacenada en la posición 1725 y contiene un desplazamiento de +52, la posición direccionada es PC + 52, pero PC = 1725 + 1 = 1726, luego la posición direccionada es 1726 + 52 = 1778.



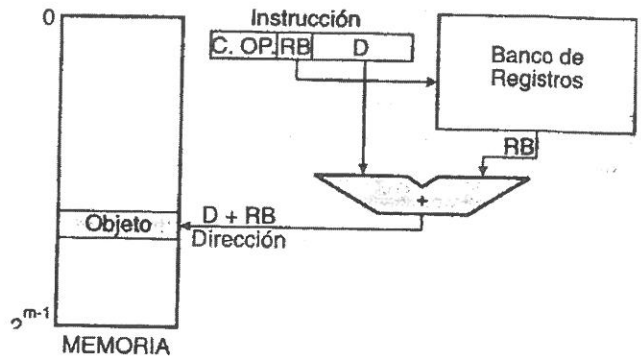
Este tipo de direccionamiento está especialmente indicado para alcanzar instrucciones próximas a la que se está ejecutando, por ejemplo, para hacer bifurcaciones que permitan construir bucles. Dado que la mayoría de los bucles son muy cortos, pues contienen unas pocas instrucciones, un desplazamiento de 1 octeto o menor es perfectamente adecuado, permitiendo un código muy compacto (esto es, unas instrucciones muy cortas). Ejemplo:

> 385B PC ← PC + 5B si C=1. Instrucción del Z80 que bifurca, si el biestable C de acarreo está a 1, a la dirección obtenida sumando 5B al contador de programa.

5.3.2 Direccionamiento directo relativo a registro base.

El direccionamiento relativo a registro base emplea como puntero un registro base RB. Generalmente, los computadores disponen de varios registros que pueden actuar como base, ya sean éstos los registros generales o bien registros específicos para ese fin.

La instrucción deberá contener la identificación del registro que se emplea como base, así como el desplazamiento. Para obtener la dirección se ha de seleccionar el registro base RB y sumarle el desplazamiento D. La siguiente figura esquematiza esta situación.



Este direccionamiento se diferencia del relativo a índice en que el registro de base no suele modificarse y el de índice sí. El direccionamiento es muy conveniente, por ejemplo, cuando se dispone de una zona de datos. Cargando en el registro base la primera posición de esta zona, se pueden alcanzar los distintos datos sin más que

conocer su posición relativa dentro de la zona de datos. Un pequeño desplazamiento será suficiente para recoger esta posición relativa. Ejemplos:

927C47B8 M(R4 + 7B8) ← 7C. Instrucción del IBM 370. El direccionamiento del destino es relativo, y viene especificado por 47B8. El primer carácter representa el registro base (esto es el registro R4) y el resto representa un desplazamiento de 12 bits de valor 7B8. Por tanto, la dirección del destino es R4 + 7B8. El origen es un inmediato de valor 7C.

FD867A A ← A + M(IX + 7A). Instrucción del Z80 que suma al registro A la posición de memoria que se obtiene de forma relativa, sumando el registro IX y el desplazamiento de 1 octeto 7A.

5.3.3 Direccionamiento directo relativo a registro índice.

El direccionamiento relativo al registro índice es una variación del anterior. En este caso, el registro puntero (que llamamos registro índice RI) es modificado para ir recorriendo los elementos de una tabla o vector. En efecto, si cada elemento de la tabla ocupa 1 palabra de la memoria y, cada vez que se emplea, el registro índice se incrementa en 1, se van obteniendo las direcciones de los elementos sucesivos de la tabla.

Es muy frecuente permitir tanto el autoincremento como el autodecremento del registro índice. En total existen cuatro alternativas diferentes:

- Preautoincremento. El registro RI es incrementado y seguidamente se obtiene la dirección como suma de RI+D.
- Preautodecremento. El registro RI es decrementado y seguidamente se obtiene la dirección como suma de RI+D.
- Postautoincremento o autoincremento. Primero se calcula la dirección RI+D y seguidamente se incrementa el registro RI.
- Postautodecremento o autodecremento. Primero se calcula la dirección RI+D y seguidamente se decrementa el registro RI.

En general, el incremento deberá adaptarse a la longitud de los operandos empleados. El VAX, que permite operandos de 1, 2 ó 4 octetos, tiene direccionamientos con autoincremento o autodecremento que utilizan automáticamente incrementos de 1, 2 ó 4, de acuerdo al operando al que se refiera la operación. Por su lado, el UNIVAC 1100 divide al registro índice en dos partes Xm y Xi. La parte Xm actúa como el registro. índice clásico, sumándose al desplazamiento, mientras que la parte Xi contiene un incremento de 18 bits, que puede ser positivo o negativo. Después de calcular la dirección, Xm se modifica, añadiéndole Xi. Ejemplo:

5A4537B8 R4 ← R4+M(R3+R5+7B8). Operación de suma en binario del IBM 370. El registro 5 es el registro índice y el 3 es el base. Sin embargo, el IBM 370 no tiene ni autoincremento ni autodecremento.

5.3.4 Direccionamiento a pila.

El direccionamiento a pila es un caso particular de direccionamiento relativo, muy empleado en los microprocesadores y minicomputadoras.

La máquina deberá disponer de uno o varios registros SP, que realicen la función de puntero de la pila. Cada uno de ellos contiene la dirección de la posición de memoria

principal que actúa de cabecera de pila. La pila puede crecer según direcciones de memoria crecientes o decrecientes. Para el caso de direcciones crecientes, si el puntero SP contiene la cantidad 3721 y se introduce un nuevo elemento en la pila, SP pasará, a tener la dirección 3722. Si, por el contrario, se elimina un elemento deberá pasar a 3720. Los direccionamientos requeridos son:

- Para insertar un nuevo elemento, se realiza un direccionamiento relativo al registro SP con preautoincremento.
- Para extraer un elemento, se debe hacer postautodecremento.

Si la pila crece según direcciones decrecientes, para insertar un elemento hay que hacer preautodecremento y para extraerlo postautoincremento.

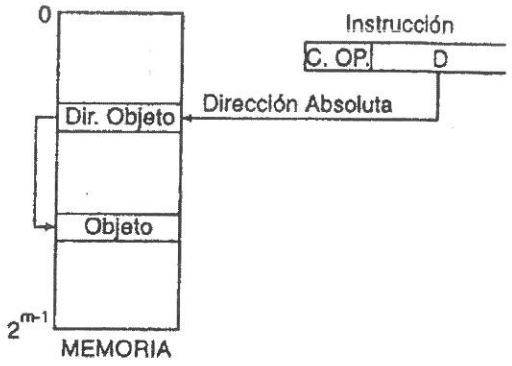
El direccionamiento a pila permite instrucciones muy compactas, puesto que, si sólo se dispone de un único registro SP, como es muy corriente, la instrucción no requiere ninguna información de dirección. Ejemplo:

F1 A ← M(SP+l); F ← M(SP); SP ← SP+2. Operación del Z80 que extrae el primer elemento de la pila y lo transfiere al registro F, extrae el segundo elemento de la pila y lo transfiere al registro A y postautoincrementa SP en 2.

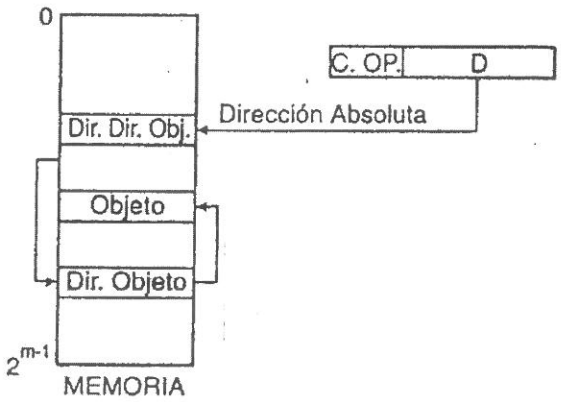
5.4 Direccionamiento indirecto.

El direccionamiento indirecto comienza con un direccionamiento directo (ya sea absoluto o relativo), pero se diferencia de aquél en que la dirección obtenida no apunta al objeto deseado sino su dirección. Por tanto, para obtener el objeto deseado se requiere un acceso adicional a memoria principal.

La siguiente figura esquematiza un direccionamiento indirecto absoluto.



Por su lado, la siguiente figura plantea indirecciones de más de un nivel, aunque no parece que ello tenga una gran utilidad, puesto que exige varios accesos a memoria y ocupa varias posiciones, de la misma.



Una aplicación típica del direccionamiento indirecto consiste en el acceso a diversas informaciones mediante una tabla de punteros. En efecto, mediante un acceso indirecto, a través del elemento correspondiente de esa tabla, se puede acceder a la información deseada.

La indirección se puede añadir a todos los tipos de direccionamiento relativos vistos anteriormente. Cabe, además, plantear la indirección en diversos puntos del cálculo de la dirección. Veamos algunas alternativas, para el caso de direccionamiento indirecto relativo a índice y base.

M(M(RB+RI+D)) Se calcula la dirección primaria como RB+RI+D y seguidamente se realiza la indirección.

M(M(RB+D)+RI) Se calcula la dirección primaria como RB+D. Se accede a memoria (indirección) y al contenido de esta posición se añade el RI, obteniéndose así la dirección definitiva.

M(M(RB)+D+RI) Se accede a la posición de memoria indicada por RB (indirección) y al valor obtenido se le suma el desplazamiento D y el registro índice RI, obteniéndose así la dirección definitiva.

Se podrían complicar estos ejemplos con autoincrementos o autodecrementos.

A pesar de las múltiples alternativas que presenta el direccionamiento indirecto relativo, lo más frecuente es que primero se calcule la dirección relativa y seguidamente se aplique la indirección.

Aunque son múltiples las aplicaciones de este direccionamiento, muchas arquitecturas no lo incluyen (por ejemplo el IBM 370).

5.5 Direccionamiento implícito.

En el direccionamiento implícito, la instrucción no contiene información sobre la ubicación del objeto, porque éste está en un lugar predeterminado (registro o posición de memoria).

la ventaja del direccionamiento implícito es que no ocupa espacio en la instrucción. Por el contrario, su inconveniente es que restringe la aplicación de la mencionada operación.

Varias de las instrucciones del Z80 que se han utilizado anteriormente emplean direccionamiento implícito del registro A, esto es, del acumulador. Este registro no viene especificado en la instrucción, pero es uno de los operandos.

5.6 Direccionamientos del IEEE 694.

El estándar IEEE 694 considera los tipos de direccionamientos que se incluyen en la siguiente tabla.

Modo	Prefijo	Ejemplo
Absoluto	Prefijo /	/dir
Página base	Prefijo !	!dir
Indirecto	Corchetes	[dir]
Relativo a PC	Prefijo $	$dir
Inmediato	Prefijo #	#valor
Relativo a registro base	Corchetes	desp[.reg] o [.reg,desp]
Registro	Prefijo .	.dir
Autopreincremento	Prefijo ++	++dir
Autopostincremento	Sufijo ++	dir++
Autopredecremento	Prefijo --	--dir
Autopostdecremento	Sufijo --	dir--

El direccionamiento relativo a registro base se puede indicar con corchetes y un desplazamiento que puede preceder a dichos corchetes, o que puede incluirse dentro de ellos. Ambas notaciones se refieren al direccionamiento en el que la dirección de un dato se obtiene sumando un desplazamiento y el contenido de un registro base.

Observemos que el direccionamiento a registro con un nivel de indirección se puede contemplar como un caso particular de direccionamiento relativo a registro con desplazamiento nulo.

Finalmente hay que destacar que los incrementos y decrementos de los cuatro últimos casos de la tabla son de una unidad. En caso de desear un incremento o decremento de valor a hay que añadir ":a" tras el prefijo o sufijo correspondiente.

6 Conclusiones.

La programación en lenguaje máquina consiste en definir, en forma binaria, octal o hexadecimal, los códigos y direcciones de las instrucciones necesarias para resolver el problema propuesto. El lenguaje ensamblador surge como una herramienta para simplificar la programación en instrucciones de máquina. Esta simplificación tiene dos aspectos fundamentales: el empleo de códigos mnemónicos, y el empleo de nombres simbólicos. De esta forma, se libera al programador de tener que especificar posiciones de memoria y de recordar los códigos de cada instrucción.

La programación consiste en convertir las especificaciones o descripción del problema, redactados en lenguaje natural, en un algoritmo específico para resolver el problema. La codificación, por su parte, consiste en la conversión de estos pasos en sentencias del lenguaje empleado. El proceso de codificación en ensamblador conlleva dos partes principales: la asignación de posiciones específicas a los datos, y la traducción de los pasos del problema en instrucciones elementales del computador.

Existen dos clases de sentencias en ensamblador: las sentencias propiamente dichas, que corresponden a instrucciones máquina, y las pseudoinstrucciones, que son órdenes al programa ensamblador que se encarga de traducir el programa a código máquina.

El formato típico de ensamblador ocupa una línea por sentencia, incluyendo los campos: etiqueta, código, operandos, y comentarios. El campo etiqueta sirve para identificar la instrucción o pseudoinstrucción especificada en esa línea. El código de la instrucción contiene el código mnemónico de la instrucción que se desea. Los

operandos especifican la dirección física o el dato propiamente dicho. Los comentarios son explicativos del proceso que se realiza.

Las instrucciones del ensamblador se refieren a los códigos mnemónicos de las instrucciones de máquina que tenga un computador. En general, cada fabricante tiene sus propios mnemónicos. Sin embargo, existe una tendencia a uniformar, por lo que los estándares son cada vez más importantes. El juego de instrucciones de un computador debe cumplir dos condiciones: debe ser completo y debe ser eficaz.

En este tema hemos estudiado las instrucciones consideradas por el estándar IEEE 694, clasificadas por: instrucciones de transferencia de datos, instrucciones que modifican la secuencia del programa, instrucciones aritméticas, instrucciones de comparación, instrucciones lógicas, instrucciones de desplazamiento, instrucciones de bit, e instrucciones de entrada/salida y misceláneas.

Un modo de direccionamiento es un procedimiento que permite determinar un operando, o la ubicación de un operando o una instrucción.

El direccionamiento se llama inmediato cuando el objeto, en este caso un operando, se encuentra contenido en la propia instrucción.

Un direccionamiento se llama directo cuando expresa la dirección real del objeto. Por otro lado, el direccionamiento absoluto indica que la instrucción contiene una dirección efectiva sin compactar. Por tanto, el direccionamiento directo absoluto indica que la instrucción contiene la dirección real, sin compactar, del objeto.

En el direccionamiento directo relativo la instrucción no contiene la dirección del objeto, sino un desplazamiento sobre una dirección marcada por un puntero. En el direccionamiento relativo a contador de programa el puntero empleado es el contador de programa PC. En el direccionamiento relativo a registro base se emplea como puntero un registro base RB. En el direccionamiento relativo al registro índice el registro puntero es modificado para ir recorriendo los elementos de una tabla o vector. Por último, en el direccionamiento a pila la máquina deberá disponer de uno o varios registros SP, que realicen la función de puntero de la pila.

El direccionamiento indirecto comienza con un direccionamiento directo (ya sea absoluto o relativo), pero se diferencia de aquél en que la dirección obtenida no apunta al objeto deseado sino su dirección.

En el direccionamiento implícito, la instrucción no contiene información sobre la ubicación del objeto, porque éste está en un lugar predeterminado (registro o posición de memoria).