TEMA 33

PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR. INSTRUCCIONES BÁSICAS. FORMATOS. DIRECCIONAMIENTOS.

ÍNDICE

1. INTRODUCCIÓN

Un ordenador digital o, mejor dicho, su parte física, sólo distingue datos de tipo binario, es decir, constituidos por dos únicos valores a los que se denomina valor 0 y valor 1 y que, físicamente, se materializan con tensiones comprendidas entre 0 y 0,4 voltios y entre 4 y 5 voltios, respectivamente. Para representar datos que contengan una información se utilizan una serie de unos y ceros y cuyo conjunto indica dicha información.

La información que hace que el hardware de la computadora realice una determinada actividad se llama *instrucción*. Por consiguiente, una instrucción es un conjunto de unos y ceros. Las instrucciones así formadas equivalen a acciones elementales de la máquina, por lo que al conjunto de dichas instrucciones que son interpretadas directamente por la máquina se denomina *lenguaje máquina*.

El lenguaje máquina fue el utilizado para programar los primeros ordenadores. Una instrucción en lenguaje máquina puede representarse de la siguiente forma:

011011001010010011110110

Esta secuencia es fácilmente ejecutada por la computadora, pero es de difícil interpretación, siendo aún más difícil la interpretación de un programa (conjunto de instrucciones) escrito de esta forma.

Con la práctica en el manejo del ordenador se cayó en la cuenta de que se podría utilizar la propia máquina para ayudar a la traducción de estos programas. Es decir, que si a una máquina elemental se le dota de un programa, también elemental, que traduzca un número determinado de caracteres alfabéticos en una secuencia de unos y ceros, se podrá escribir un programa constituido por una secuencia de grupos de caracteres alfabéticos, en el que cada uno de los grupos indicaría una acción a realizar por el ordenador y, una vez escrito el programa, sería la propia máquina la que pasaría los grupos de caracteres a bits.

De esta forma aparecieron los *lenguajes ensambladores* (assembler, en inglés). Poco a poco, con el avance de la programación (software), estas primeras y sencillas ayudas se fueron haciendo más complejas.

Se introdujo la posibilidad de indicar al ordenador la dirección de un salto en la secuencia de ejecución de un programa mediante la utilización de *etiquetas*.

A los programas que permiten pasar del programa escrito de esta manera (programa fuente, en ensamblador) al lenguaje máquina también se les llama normalmente *ensambladores*.

Aún con todas estas sofisticaciones y ayudas, el programador del lenguaje ensamblador debe conocer perfectamente el sistema físico (hardware) de la máquina con que trabaja, pues aunque emplee mnemotécnicos, etiquetas, etc., éstas sirven para indicar una posición de memoria determinada, un registro o cualquier otra parte de la máquina.

Por eso se dice que el lenguaje ensamblador es un lenguaje de bajo nivel, es decir, ligado con el hardware concreto de una determinada máquina.

Las aportaciones del lenguaje ensamblador son:

- Uso de una *notación simbólica o nemotécnica* para representar los códigos de operación. De esta forma, se evitan los códigos numéricos, tan difíciles de manejar. Normalmente, dichos códigos mnemotécnicos están constituidos por abreviaturas de las operaciones en inglés. Así, por ejemplo, la suma se representa en la mayoría de los ensambladores por ADD.

- *Direccionamiento simbólico*. En lugar de utilizar direcciones binarias absolutas los datos pueden identificarse con nombres como COSTE, TOTAL, X, Y, etc.

- Se permite el *uso de comentarios* entre las líneas de instrucciones, haciendo posible la redacción de programas más legibles.

Aparte de esto, el lenguaje ensamblador presenta la mayoría de los inconvenientes del lenguaje máquina, como su repertorio muy reducido de instrucciones, el formato rígido, la baja portabilidad y la fuerte dependencia del hardware. Por otro lado, mantiene la ventaja del uso óptimo de los recursos hardware, permitiendo la obtención de un código muy eficiente.

Para solventar en cierta medida la limitación que supone poseer un repertorio de instrucciones tan reducido, se han desarrollado unos ensambladores especiales denominados *macroensambladores*. Los lenguajes que traducen los macroensambladores disponen de macroinstrucciones cuya traducción da lugar a varias instrucciones máquina y no a una sola.

Dado que el lenguaje ensamblador está fuertemente condicionado por la arquitectura del ordenador que soporta, no se suelen escribir programas de tamaño considerable en ensamblador. Más bien se utiliza este lenguaje para afinar partes importantes de programas escritos en lenguajes de alto nivel. El lenguaje ensamblador sigue siendo importante, ya que, da al programador el control total de la máquina, y como resultado genera un código compacto, rápido y eficiente.

2. INSTRUCCIONES BÁSICAS

En Informática, una *instrucción* es un conjunto de símbolos que el ordenador es capaz de interpretar con objeto de realizar las acciones que dichos símbolos le indican.

Las instrucciones en lenguaje ensamblador son muy elementales; indican acciones muy concretas, nombrando incluso a los elementos que tienen los datos o que guardan los resultados, como son los registros, posiciones de memoria, periféricos, etc.

2.1. Tipos de instrucciones

Podemos clasificar las instrucciones, según la acción que ordenan, en los siguientes tipos:

Instrucciones de transferencia de datos: Indican a la CPU que copie el dato de un registro o posición de memoria en otro registro u otra posición de memoria.

Instrucciones de entrada y salida: Indican a la CPU que debe ordenar la transferencia de datos entre algún elemento interno del ordenador (registros, posiciones de memoria, etc.) y alguno externo (periféricos). Suele haber una instrucción para pasar datos desde un periférico al interior del ordenador (entrada) y otra para pasar el dato del interior al periférico (salida).

Instrucciones de cálculo: Indican a la CPU la operación que debe realizar. Pueden ser **aritméticas** (sumar, restar, etc.), **lógicas** (O, Y, NO, etc.), de **comparación** y de **rotación** y **desplazamiento** de los bits de un registro o posición de memoria hacia la izquierda o hacia la derecha, etc.

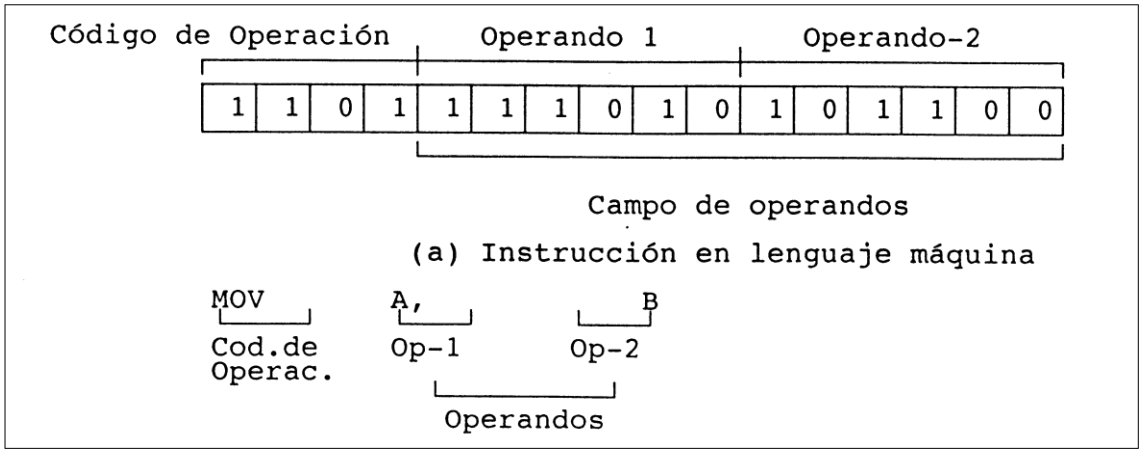
Instrucciones de ruptura de secuencia: Indican a la CPU que la siguiente instrucción que debe ejecutarse no es la que corresponde según la secuencia física del programa, sino que debe realizar un salto a otro punto del mismo programa. Estos saltos pueden ser incondicionales o condicionales, según se realicen siempre o sólo cuando de cumpla alguna condición, que, normalmente, dependerá de algún bit del registro de estado de la CPU.

Instrucciones de control de la CPU: Modifican el contenido del registro de estado o pueden hacer que la CPU se detenga en la interpretación de instrucciones hasta nueva orden, que se dará mediante algún dispositivo externo a ella.

2.2. Partes constituyentes de una instrucción

En toda instrucción podemos diferenciar dos partes o **campos**. Una de estas partes indica a la CPU la operación que debe ejecutar, y se llama *campo de código de operación*; la otra le indica de dónde debe tomar los datos y dónde poner los resultados, y se denomina *campo de operandos*.

En la Figura se muestra una instrucción indicando el campo de código de operación y los campos de operandos. Como se puede apreciar, la parte (a) de la figura representa una instrucción en el lenguaje máquina, es decir, un conjunto de unos y ceros que la máquina es capaz de interpretar de forma sencilla, pero que al programador le resulta más complicado hacerlo. La parte (b) de la figura muestra cómo sería la misma instrucción, pero ahora en lenguaje ensamblador. En esta forma de representación el programador la entiende fácilmente, pero para que la máquina sea capaz de ejecutarla deberá traducirla primero a lenguaje máquina.



Campo código de operación: Este campo es obligatorio en todas las instrucciones, ya que, indica a la CPU precisamente la operación que debe realizar. Cada instrucción u operación estará codificada, en lenguaje máquina, en una serie de ceros y unos de forma que no haya dos instrucciones diferentes que tengan el mismo código.

El número de bits que se emplean en el código puede ser idéntico en todas las instrucciones de una determinada máquina, en cuyo caso se dice que esa máquina tiene un código de operación fijo; o puede variar, en cuyo caso se dice que la máquina tiene un código de operación variable o expandido.

El código de operación suele estar ocupando los bits más significativos de la palabra, los situados más a la izquierda, como se ve en la Figura (a). Si una instrucción ocupa más de una palabra, el código de operación estará en la primera palabra que lea la CPU, para que ésta pueda ir interpretando la instrucción y saber si después hay más palabras pertenecientes a la misma instrucción o no.

También en lenguaje ensamblador se indica el campo de código de operación en primer término.

A los caracteres que forman el código de operación en lenguaje ensamblador se les denomina *mnemotécnicos*.

Campo de operandos: Además del código de operación, en una instrucción podemos tener otro campo, como ya se ha indicado anteriormente, llamado campo de operandos. Este campo no está presente en todas las instrucciones, ya que hay algunas que no emplean datos y otras en la que la localización de los mismos está implícita en el propio código de operación. La longitud de este campo es normalmente variable, dependiendo del número de operandos que utilice la instrucción y de la forma que se indique a la CPU el acceso a los mismos. De existir campos de operandos, se codifica la dirección de los mismos a continuación del campo de código de operación, como se puede ver en la Figura.

En lenguaje ensamblador se emplean símbolos alfanuméricos para representar los operandos. Pueden ser nombres de variables que representen posiciones de memoria, registros, etc., utilizando una representación decimal, octal, hexadecimal, etc.

2.3. Formatos

Como ya se ha visto en los apartados anteriores, una instrucción puede tener, además del código de operación, un número variable de operandos; pueden ser varios, sólo uno o ninguno. Esto hace que una misma máquina pueda tener instrucciones de diferentes formas y longitudes.

Las variantes posibles en cuanto a las longitudes de código de operación, campos de operando e instrucción son:

(a) Longitud de instrucción fija, código de operación extendido.

(b) Longitud de instrucción variable, código de operación fijo.

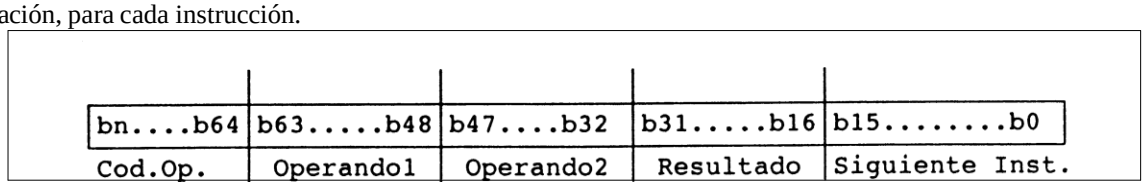
(c) Longitud de instrucción variable, código de operación extendido.

A continuación se estudiarán los diferentes formatos que podemos encontrar en diferentes máquinas, dependiendo del número de operandos o direcciones que utilicen en la instrucción.

Formato de cuatro direcciones. En la actualidad no es frecuente encontrar máquinas con este formato de instrucción. Dos direcciones indican los operandos con los que operará la CPU, otra dirección le indicará dónde debe guardar el resultado, y la otra le dirá dónde se encuentra la siguiente instrucción a ejecutar.

Estas máquinas no utilizan registro contador de programa, ni son necesarias las instrucciones de salto específicas. El problema de este formato es la gran cantidad de bits que necesitan las instrucciones para poder codificar tantas direcciones.

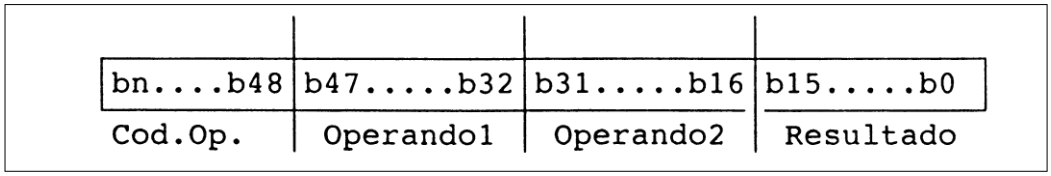
En la figura se muestra una instrucción de este formato, suponiendo que la máquina puede realizar accesos a cualquier posición de una memoria de 64K palabras de capacidad. Los bits necesarios para cada dirección son 16, por lo que se necesitarán 16 * 4 = 64 bits, más los correspondientes al código de operación, para cada instrucción.



Formato de tres direcciones. Ante la necesidad de simplificar las máquinas para abaratar costes y aumentar el rendimiento, se fue reduciendo la longitud de las instrucciones. Una manera de conseguirlo es eliminando información en las mismas.

Un avance muy importante en este sentido es la aparición del contador de programa, registro que guarda la dirección de la siguiente instrucción a ejecutar, por lo que ya no es necesario incluirla en la instrucción. De esta manera se pierde algo de flexibilidad en el secuenciamiento de las instrucciones, ya que la dirección se va calculando a medida que se ejecuta el programa de forma independiente a las instrucciones. Se hace necesario el uso de instrucciones de ruptura de secuencia, pero se ahorran bits en cada instrucción.

La figura muestra este formato suponiendo la misma posibilidad de direccionamiento que en el apartado anterior.



Formato de dos direcciones. El formato anterior aún ocupa muchas posiciones. En el ejemplo de la Figura anterior sólo los operandos necesitan 48 bits. En consecuencia, se optó por eliminar la dirección

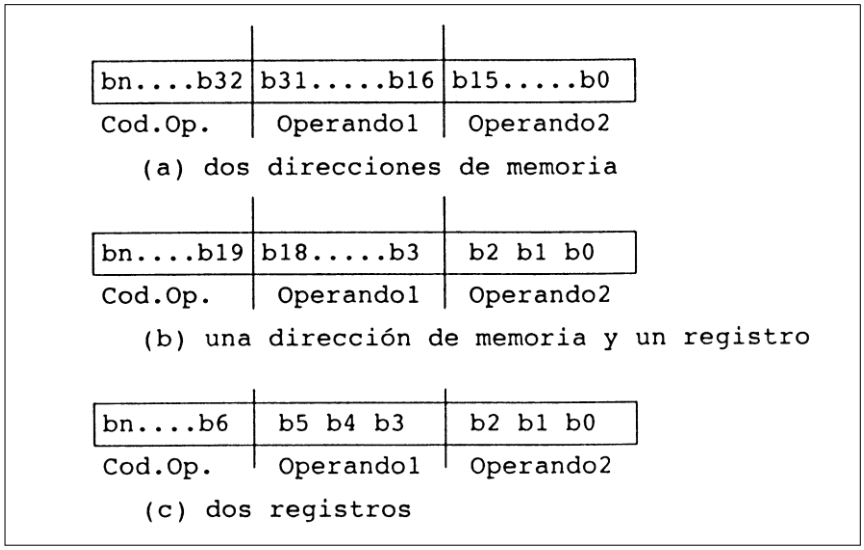
que indica dónde se guarda el resultado, empleándose par este fin uno de los dos operandos especificados en la instrucción. El operando que recoge el resultado se llama destino; el otro se llama operando fuente. El operando destino pude utilizarse también para aportar el dato que se necesita para realizar la operación indicada en el código de operación junto con el operando fuente. Indudablemente, después de ejecutada la instrucción el contenido del operando destino se habrá modificado, perdiéndose el contenido anterior.

Este formato tiene **variantes**. Una consistiría en emplear un registro para especificar uno de los operandos; de esta forma se reduce la longitud total de la instrucción, ya que para indicar un registro de la CPU son necesarios muchos menos bits que para una dirección de memoria.

Por ejemplo, tomando la misma capacidad de memoria que en los apartados anteriores, vimos que para indicar una dirección de memoria eran necesarios 16 bits. Si suponemos que la CPU posee 8 registros de trabajo, serían suficientes tres bits para hacer referencia a cada uno de ellos. Esta variante del formato de dos direcciones también se denomina formato de una dirección y media.

Otra variante consistiría en especificar como operandos dos registros, con lo que se reduciría aún más la longitud de la instrucción.

En la Figura se muestran las variantes de este formato.



Formato de una dirección. En este formato se indica el registro de forma implícita en el código de operación, por lo que en el campo de operandos queda sólo la dirección de un operando. El otro operando, que recibe el resultado, se supone que es un determinado registro de trabajo de la CPU, denominado generalmente *acumulador*.

Formato de cero direcciones. Existen máquinas que utilizan para guardar datos una estructura denominada pila. La pila no es más que un conjunto de posiciones consecutivas de memoria, y para acceder a las mismas se utiliza un registro llamado "apuntador de pila", que guarda la dirección de la cabecera de la pila, o elemento al que vamos a acceder. Cuando almacenemos un nuevo dato, este registro se incrementará y, cuando saquemos un dato, se decrementará para indicar el siguiente elemento.

Con esta estructura sólo hace falta especificar la operación, ya que la CPU supone que los operandos están guardados en la pila, donde los busca con la ayuda del apuntador de pila, y una vez realizada la operación supone que el resultado debe guardarlo en la misma pila.

3. MODOS DE DIRECCIONAMIENTO

Las diferentes formas de indicar en el campo de operandos dónde se encuentran los datos que la CPU tendrá que procesar se denominan "modos de direccionamiento".

Aunque los diseñadores de procesadores mezclan estos modos para aprovechar las ventajas de unos y otros o conseguir determinadas características en las máquinas, básicamente podemos distinguir cuatro modos de direccionamiento:

- Inmediato.
- Directo o absoluto.
- Indirecto.
- Indexado.

Modo de direccionamiento inmediato. En este método, el valor del operando es el que consta en la propia instrucción. Se emplea este modo de direccionamiento cuando se desea utilizar constantes en el programa.

Es el de más rápido acceso al dato por obtenerse en la fase de búsqueda de la instrucción, sin necesidad de volver a leer ninguna otra vez la memoria principal. Sin embargo, no permite modificar el dato sin modificar la instrucción.

Modo de direccionamiento directo o absoluto. En este modo, el campo de operandos indica la dirección en la que se encuentra el dato. Con este método debe hacerse un acceso a la memoria para buscar el valor del operando. Es, pues, un modo de direccionamiento más lento que el inmediato. Sin embargo, puede modificarse el valor del dato sin modificar para nada la instrucción; basta modificar el contenido de la posición de memoria que indica el campo de operandos.

Este método proporciona una manera sencilla y rápida de acceder a la posición de una variable; sin embargo, tiene dos problemas:

1. Que la variable siempre tiene que estar en una posición fija de memoria.
2. Que el campo de operando debe tener la longitud suficiente para poder indicar cualquier posición de memoria.

Una variante de este direccionamiento es el denominado *direccionamiento a registro*, en el cual la dirección indica que el campo operando corresponde a la de uno de los registros de la CPU, no a una posición de memoria. De esta forma se gana en rapidez, ya que el acceso a un registro es mucho más rápido que el acceso a una posición de memoria, y la longitud de la instrucción será menor.

El inconveniente de este modo de direccionamiento es que el valor a utilizar debe ponerse dentro del registro antes de usarlo y, como el número de registros es escaso, una vez utilizado debe guardarse de nuevo en la memoria.

Modo de direccionamiento indirecto. En la instrucción se indica el lugar donde se encuentra la dirección del operando. En este modo y en el siguiente se empleará el concepto de *dirección efectiva* para indicar la dirección de memoria donde se encuentra el dato.

En el modo de direccionamiento indirecto la dirección que se indica en la instrucción no es la dirección de dato, sino la dirección de la "dirección efectiva". Es decir, que para acceder al dato la CPU deberá buscar en la dirección que se indica en la instrucción y ahí hallará otra dirección, que es la que le indicará donde se encuentra el dato. Al final de la instrucción se habrán realizado dos accesos a memoria o registros, hasta llegar al dato.

¿Qué se persigue con semejante complicación de direccionamiento? Se consigue poder variar la posición del dato sin modificar la instrucción; es suficiente con variar el contenido de la dirección indicada en la instrucción, que, como hemos visto antes, puede ser un registro o una posición de memoria.

Modo de direccionamiento indexado. En este modo de direccionamiento, la dirección efectiva se calcula sumando un valor, llamado *índice*, a la dirección indicada en la instrucción. El valor índice se encuentra, generalmente, en algún registro de la CPU. En algunos ordenadores existen registros especializados en guardar estos valores. En estos casos, los registros suelen indicarse en la instrucción de forma implícita y suele haber instrucciones especiales para incrementar o decrementar estos registros denominados *registros de índice*.

La dirección efectiva se calcula sin modificar el contenido del registro de índice ni de la dirección que se indica en la instrucción. Esta última dirección también de denomina "dirección base", y el valor de índice, "desplazamiento".

El empleo de este método de direccionamiento es parecido al anterior, sin embargo, en el direccionamiento indirecto el registro que se empleaba para guardar la dirección efectiva debía tener la suficiente longitud como para indicar cualquier posición de memoria. Por ejemplo, si la memoria es de 64K palabras, el registro será como mínimo de 16 bits.

En el direccionamiento indexado el registro índice no tienen por qué ser tan grande. Por ejemplo, con un registro de 8 bits se puede acceder a 256 posiciones, desde la dirección base hasta la dirección base + 255, margen que en la mayoría de las aplicaciones es suficiente.

4. BIBLIOGRAFÍA

Alberto Prieto
Introducción a la Informática
Mc Graw-Hill, 2ª edición, 1997

Alfonso Ureña López
Fundamentos de Informática
Ra-ma, 1997