5.4 DIRECCIONAMIENTO

Un diseño cuidadoso de los códigos de operación es una parte importante de la ISA. Sin embargo, una porción considerable de los bits de un programa se gastan en especificar de dónde provienen los operandos, más que las operaciones que se efectuarán con ellos. Considere una instrucción ADD que requiere la especificación de tres operandos: dos fuentes y un destino. (Se acostumbra usar el término *operando* para referirse a los tres, pero el destino es el lugar donde se coloca el resultado.) De alguna manera, la instrucción ADD debe indicar dónde están los operandos y dónde debe ponerse el resultado. Si las direcciones de memoria tienen 32 bits, la especificación ingenua de esta instrucción requiere tres direcciones de 32 bits además del código de operación. Las direcciones ocupan muchos más bits que el código de operación.

Se usan dos métodos generales para reducir el tamaño de la especificación. Primero, si un operando va a usarse varias veces, puede colocarse en un registro. El uso de un registro para una variable tiene dos ventajas: el acceso es más rápido y se requieren menos bits para especificar el operando. Si hay 32 registros, cualquiera de ellos puede especificarse con sólo cinco bits. Si la instrucción ADD puede efectuarse usando sólo operandos en registros, bastan 15 bits para especificar los tres operandos, en lugar de los 96 que se necesitarían si todos estuvieran en la memoria.

Desde luego, el uso de registros no es gratuito. De hecho, hay ocasiones en que su uso empeora el problema. Si primero hay que cargar en un registro un operando de la memoria, se ocuparán más bits de especificación que si sólo se especificara la posición en memoria. Primero se necesita una instrucción LOAD para colocar el operando en un registro. Esto no sólo requiere un código de operación, sino también la dirección de memoria completa y la especificación del registro objetivo. Por tanto, si un operando sólo se va a usar una vez, no vale la pena colocarlo en un registro.

Por fortuna, años de mediciones muestran que los operandos se reutilizan mucho. Por consiguiente, casi todas las arquitecturas recientes cuentan con un gran número de registros y casi todos los compiladores hacen hasta lo imposible por mantener variables locales en esos registros, lo que elimina muchas referencias a la memoria. Esto reduce tanto el tamaño de los programas como el tiempo de ejecución.

Un segundo método para reducir el tamaño de la especificación consiste en especificar uno o más operandos implícitamente. Hay varias técnicas para hacerlo. Una de ellas es utilizar una sola especificación para un operando fuente y para el destino. Mientras que una instrucción ADD general de tres direcciones podría usar la forma

```
DESTINO = FUENTE1 + FUENTE2
```

una instrucción de dos registros podría estar limitada a la forma

```
REGISTRO2 = REGISTRO2 + FUENTE1
```

Esta instrucción es destructiva en el sentido de que el contenido de REGISTRO2 no se conserva. Si el valor original se va a necesitar posteriormente, primero habrá que copiarlo en otro registro. El equilibrio aquí es que las instrucciones de dos direcciones son más cortas, pero son menos generales. Diferentes diseñadores toman diferentes decisiones. El Pentium II, por ejemplo, usa instrucciones de dos direcciones en el nivel ISA, mientras que el UltraSPARC II usa instrucciones de tres direcciones.

Habiendo reducido el número de operandos de nuestra instrucción ADD de tres a dos, no nos detengamos. Las primeras computadoras tenían sólo un registro llamado **acumulador**. La instrucción ADD, por ejemplo, siempre sumaba una palabra de la memoria al acumulador, por lo que sólo era necesario especificar un operando (el de la memoria). Esta técnica funcionaba bien con cálculos sencillos, pero si se requerían varios resultados intermedios el acumulador tenía que escribirse en la memoria y más adelante tenía que recuperarse. Es por esto que la técnica se ha dejado de usar.

Ahora hemos pasado de una ADD de tres direcciones a una de dos direcciones y luego a una con una sola dirección. ¿Qué falta? ¿Cero direcciones? Sí. En el capítulo 4 vimos cómo IJVM usa una pila. La instrucciones IADD de IJVM no tiene direcciones. Tanto la fuente como el destino están implícitos. Veremos el direccionamiento de pila un poco más adelante.

5.4.1 Modos de direccionamiento

Hasta aquí no nos hemos fijado mucho en la forma en que los bits de un campo de dirección se interpretan para encontrar el operando. Una posibilidad es que contienen la dirección en memoria del operando. Además del campo grande que se necesita para especificar la dirección en memoria completa, este método tiene la restricción adicional de que la dirección se debe determinar en el momento de la compilación. Hay otras posibilidades que permiten acortar las especificaciones y también determinar las direcciones dinámicamente.

En las secciones que siguen exploraremos algunas de estas formas, llamadas **modos de direccionamiento**.

5.4.2 Direccionamiento inmediato

La forma más sencilla de especificar un operando es que la parte de dirección de la instrucción realmente contenga el operando mismo en lugar de una dirección u otra información que describa la ubicación del operando. Semejante operando se llama **operando inmediato** porque se obtiene automáticamente de la memoria al mismo tiempo que se obtiene la instrucción misma; por tanto, está disponible inmediatamente para usarse. En la figura 5-16 se muestra una posible instrucción para cargar el registro R1 con la constante 4.

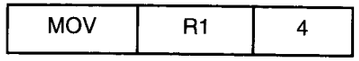

Figura 5-16. Instrucción inmediata para cargar 4 en el registro 1.

El direccionamiento inmediato tiene la virtud de no requerir una referencia extra a la memoria para obtener el operando. Su desventaja es que con esta técnica sólo se puede suministrar una constante. Además, el número de valores está limitado por el tamaño del campo. No obstante, muchas arquitecturas usan esta técnica para especificar constantes enteras pequeñas.

5.4.3 Direccionamiento directo

Un método para especificar un operando en la memoria es sencillamente dar su dirección completa. Este modo se llama **direccionamiento directo**. Al igual que el direccionamiento inmediato, el uso del direccionamiento directo tiene limitaciones: la instrucción siempre accesará a la misma localidad de memoria exactamente. Si bien el valor contenido en ella puede cambiar, la dirección no puede alterarse. Por ello, el direccionamiento directo sólo sirve para accesar variables globales cuya dirección se conoce en el momento de la compilación. Por otra parte, muchos programas tienen variables globales, por lo que este modo se usa ampliamente. Los detalles de cómo la computadora sabe cuáles direcciones son inmediatas y cuáles son directas se explicarán más adelante.

5.4.4 Direccionamiento por registro

El direccionamiento por registro es igual al directo en lo conceptual pero especifica un registro en lugar de una posición de memoria. Dada la importancia de los registros (por su acceso tan rápido y sus direcciones tan cortas) este modo de direccionamiento es el más común en la mayor parte de las computadoras. Muchos compiladores se esfuerzan por determinar a cuáles variables se accesará con mayor frecuencia (por ejemplo, el índice de un ciclo) y colocan estas variables en registros.

Este modo de direccionamiento se conoce simplemente como **modo de registro**. En las arquitecturas de carga/almacenamiento como el UltraSPARC II, casi todas las instrucciones

emplean este modo de direccionamiento exclusivamente. La única ocasión en que no se usa este modo de direccionamiento es cuando un operando se pasa de la memoria a un registro (instrucción LOAD) o de un registro a la memoria (instrucción STORE). Incluso en el caso de esas instrucciones, uno de los operandos es un registro: dónde está o dónde se colocará la palabra de memoria.

5.4.5 Direccionamiento indirecto por registro

En este modo, el operando que se especifica viene de la memoria o va a ella, pero su dirección no está fija en la instrucción, como en el direccionamiento directo. En vez de ello, la dirección está contenida en un registro. Cuando una dirección se usa de esta manera, se llama **apuntador**. Una gran ventaja del direccionamiento indirecto por registro es que puede hacer referencias a la memoria sin pagar el precio de tener una dirección de memoria completa en la instrucción. También permite usar diferentes palabras de la memoria en diferentes ejecuciones de la instrucción.

Para ver qué utilidad podría tener usar una palabra diferente en cada ejecución, imagine un ciclo que procesa los elementos de un arreglo unidimensional de enteros con 1024 elementos y calcula la suma de los elementos en el registro R1. Fuera del ciclo, puede hacerse que algún otro registro, digamos R2, apunte al primer elemento del arreglo, y que otro registro, digamos R3, apunte a la primera dirección más allá del arreglo. Con 1024 enteros de 4 bytes cada uno, si el arreglo comienza en *A*, la primera dirección más allá del arreglo será *A* + 4096. En la figura 5-17 se muestra un código de ensamblador típico para realizar este cálculo en una máquina de dos direcciones.

```
        MOV R1,#0          ; acumular la suma en R1, inicialmente 0
        MOV R2,#A          ; R2 = dirección del arreglo A
        MOV R3,#A+1024     ; R3 = dirección de la primera palabra después de A
CICLO:  ADD R1,(R2)        ; indirecto por registro R2 para obtener operando
        ADD R2,#4          ; incrementar R2 en una palabra (4bytes)
        CMP R2,R3          ; ¿ya terminamos?
        BLT LOOP           ; si R2 < R3, no hemos acabado; continuar
```

Figura 5-17. Programa genérico en ensamblador para calcular la suma de los elementos de un arreglo.

En este pequeño programa usamos varios modos de direccionamiento. Las primeras tres instrucciones usan el modo de registro para el primer operando (el destino) y el modo inmediato para el segundo operando (una constante indicada por el signo #). La segunda instrucción coloca la *dirección* de *A* en R2, no su contenido. Eso es lo que el signo # le dice al ensamblador. Asimismo, la tercera instrucción coloca en R3 la dirección de la primera palabra después del arreglo.

Lo interesante es que el cuerpo del ciclo mismo no contiene ninguna dirección de memoria. En la cuarta instrucción se usa el modo de registro y el indirecto por registro; en la quinta se usa el modo de registro y el inmediato, y en la sexta se usa dos veces el modo de registro.

La instrucción de ramificación BLT podría usar una dirección de memoria, pero lo más probable es que especifique la dirección a la que lleva la rama con una distancia de 8 bits relativa a la instrucción BLT misma. Al evitar totalmente las direcciones de memoria hemos producido un ciclo corto y rápido. Por cierto, este programa es realmente para el Pentium II, sólo que cambiamos los nombres de las instrucciones y los registros, así como la notación, para hacerlo más comprensible, ya que la sintaxis del lenguaje ensamblador estándar del Pentium II (MASM) es rarísima, una reliquia de la anterior vida de la máquina como un 8088.

Vale la pena señalar que, en teoría, hay otra forma de realizar este cálculo sin usar direccionamiento indirecto por registro. El ciclo podría haber contenido una instrucción que sumara *A* y R1, como

```
ADD R1,A
```

Luego, en cada iteración del ciclo la instrucción misma podría incrementarse en 4, de modo que después de una iteración diga

```
ADD R1,A+4
```

y así hasta terminar.

Un programa que se modifica a sí mismo de esta manera es un programa **automodificable**. La idea se le ocurrió al mismísimo John von Neumann y era razonable en las primeras computadoras, que no tenían direccionamiento indirecto por registro. Hoy día los programas automodificables se consideran de muy mal gusto y difíciles de entender; además, no pueden compartirse entre varios procesos al mismo tiempo. Otra desventaja es que ni siquiera funcionan correctamente en máquinas con caché de nivel 1 dividida, si la caché I no cuenta con circuitos para efectuar escrituras de vuelta (porque los diseñadores supusieron que los programas no se modifican ellos mismos).

5.4.6 Direccionamiento indexado

En muchos casos es útil poder hacer referencia a palabras de memoria que están a una distancia conocida de un registro. Ya vimos algunos ejemplos con IJVM en los que se hace referencia a variables locales indicando su distancia respecto a LV. Direccionar la memoria dando un registro (explícito o implícito) más una distancia constante se denomina **direccionamiento indexado**.

El acceso a variables locales en IJVM utiliza un apuntador a la memoria (LV) en un registro más una distancia pequeña en la instrucción misma, como se muestra en la figura 4-19(a). Sin embargo, también es posible hacerlo del otro modo: con el apuntador a la memoria en la instrucción y la distancia pequeña en el registro. Para ver cómo funciona esto, considere el cálculo siguiente. Tenemos dos arreglos unidimensionales *A* y *B*, de 1024 palabras cada uno, y queremos calcular A_i AND B_i para todos los pares y luego obtener el OR de estos 1024 productos booleanos para ver si hay al menos un par distinto de cero en el conjunto. Una estrategia sería colocar la dirección de *A* en un registro, la dirección de *B* en un segundo registro, y luego incrementarlas juntas, de forma análoga a como hicimos en la figura 5-17.

No cabe duda de que esto funcionaría, pero hay una forma mejor de hacerlo, como se muestra en la figura 5-18.

```
        MOV R1,#0       ; acumular el OR en R1, inicialmente 0
        MOV R2,#0       ; R2 = índice, i, del producto actual: A[i] AND B[i]
        MOV R3,#4096    ; R3 = primer valor del índice que no se usará
CICLO   MOV R4,A(R2)    ; R4 = A[i]
        AND R4,B(R2)    ; R4 = A[i] AND B[i]
        OR R1,R4        ; hacer OR de todos los productos booleanos y poner en R1
        ADD R2,#4       ; i = 1 + 4 (incremento en unidades de 1 palabra = 4 bytes)
        CMP R2,R3       ; ¿ya terminamos?
        BLT LOOP        ; si R2 < R3, no hemos terminado; continuar.
```

Figura 5-18. Programa genérico en ensamblador para calcular el OR de A_i y B_i con dos arreglos de 1024 elementos.

El funcionamiento del programa es sencillo. Necesitamos cuatro registros:

1. R1 – Contiene el OR acumulado de los productos booleanos
2. R2 – El índice *i* que se usa para procesar los arreglos
3. R3 – La constante 4096, que es el valor más bajo de *i* que no debemos usar
4. R4 – Un registro de borrador para contener cada producto a medida que se forma.

Después de inicializar los registros ingresamos en el ciclo de seis instrucciones. La instrucción que está en *CICLO* trae A_i y lo coloca en R4. Aquí, el cálculo de la fuente usa el modo indizado. Un registro, R2, y una constante, la dirección de *A*, se suman y la suma se usa para hacer referencia a la memoria. La suma de estas dos cantidades se envía a la memoria, pero no se guarda en ningún registro visible para el usuario. La notación

```
MOV R4,A(R2)
```

indica que el destino usa modo de registro con R4 como el registro y la fuente usa modo indexado, con *A* como la distancia y R2 como el registro. Si *A* tiene el valor de 124300, por ejemplo, la instrucción de máquina real para esto es probable que se parezca a la de la figura 5-19.

MOV	R4	R2	124300

Figura 5-19. Una posible representación de MOV R4,A(R2).

En la primera iteración del ciclo, R2 es 0 (porque así se inicializó), de modo que la palabra de memoria direccionada es A_0, en la dirección 124300. Esta palabra se carga en R4. En la siguiente iteración del ciclo R2 es 4, así que la palabra de memoria a la que se accede es A_1, en 124304, etcétera.

Como prometimos antes, aquí la distancia en la instrucción misma es el apuntador a la memoria, y el valor que está en el registro es un entero pequeño que se incrementa durante el cálculo. Desde luego, esta forma requiere un campo de distancia en la instrucción lo bastante grande como para contener una dirección, así que es menos eficiente que la forma anterior; no obstante, en muchos casos es la mejor manera de hacerlo.

5.4.7 Direccionamiento basado indexado

Algunas máquinas tienen un modo de direccionamiento en el que la dirección en memoria se calcula sumando dos registros más una distancia (opcional). Este modo se conoce como **direccionamiento basado indexado**. Uno de los registros es la base y el otro es el índice. Un modo así habría sido útil aquí. Afuera del ciclo podríamos haber colocado la dirección de *A* en R5 y la dirección de *B* en R6. Luego podríamos haber sustituido la instrucción que está en *CICLO* y su sucesora por

```
CICLO:  MOV R4,(R2+R5)
        AND R4,(R2+R6)
```

Si hubiera un modo de direccionamiento para acceder a la memoria de forma indirecta usando la suma de dos registros sin una distancia, habría sido ideal. Como alternativa, incluso una instrucción con una distancia de 8 bits habría sido mejor que el código original, porque podríamos asignar 0 a ambas distancias. En cambio, si las distancias siempre son de 32 bits no habremos ganado nada al usar este modo. En la práctica, las máquinas que manejan este modo suelen tener una forma con una distancia de 8 bits o de 16 bits.

5.4.8 Direccionamiento de pila

Ya mencionamos que es preferible hacer las instrucciones de máquina lo más pequeñas posible. El límite para reducir la longitud de las direcciones es no tener direcciones. Como vimos en el capítulo 4, son posibles instrucciones con cero direcciones, como IADD, si se usan con una pila. En esta sección examinaremos con mayor detalle el direccionamiento de pila.

5.4.12 Modos de direccionamiento del UltraSPARC II

En la ISA UltraSPARC todas las instrucciones usan direccionamiento inmediato o de registro con excepción de las que direccionan memoria. En el modo de registro, los 5 bits simplemente indican cuál registro usar. En el modo inmediato, una constante de 13 bits (con signo) proporciona los datos. No hay otros modos para las instrucciones aritméticas, lógicas y similares.

Tres clases de instrucciones direccionan memoria: las LOAD, las STORE y una instrucción de sincronización de multiprocesadores. Las instrucciones LOAD y STORE tienen dos modos para direccionar la memoria. El primer modo calcula la suma de dos registros y luego la usa para acceso indirecto. El otro es una indexación tradicional, con una distancia de 13 bits (con signo).

5.4.13 Modos de direccionamiento de la JVM

La JVM no tiene modos de direccionamiento generales en el sentido de que toda instrucción tiene unos cuantos bits que le indican cómo calcular la dirección (como en el Pentium II). En vez de ello, cada instrucción tiene asociado un modo de direccionamiento específico. Puesto que la JVM no tiene registros visibles, los modos de direccionamiento de registro e indirecto por registro ni siquiera son posibles. Unas cuantas instrucciones, como BIPUSH, usan direccionamiento inmediato. El único otro modo disponible es el modo indizado, empleado por ILOAD, ISTORE, LDC_W y varias otras instrucciones para especificar una variable

relativa a algún registro implícito, por lo regular LV o CPP. Las instrucciones de ramificación también usan el modo indexado, con PC como registro implícito.

5.4.14 Análisis de los modos de direccionamiento

Ya hemos estudiado una buena cantidad de modos de direccionamiento. Los que usan el Pentium II, el UltraSPARC II y JVM se resumen en la figura 5-28. Sin embargo, como ya explicamos, no todas las instrucciones pueden usar todos los modos.

Modo de direccionamiento	Pentium II	UltraSPARC II	JVM
Inmediato	×	×	×
Directo	×		
Registro	×	×	
Indirecto por registro	×		
Indexado	×	×	×
Basado-indexado		×	
Pila			×

Figura 5-28. Comparación de modos de direccionamiento.

En la práctica, no se necesitan muchos modos de direccionamiento para tener una ISA eficaz. Puesto que casi todo el código que se escribe en este nivel es generado por compiladores, el aspecto más importante de los modos de direccionamiento de una arquitectura es que las opciones sean pocas y claras, y que los costos (en términos de tiempo de ejecución y tamaño del código) puedan calcularse con facilidad. Lo que eso implica generalmente es que una máquina debe adoptar una postura extrema: o bien debe ofrecer todas las opciones posibles, o sólo debe ofrecer una. Cualquier término medio implica que el compilador enfrentará decisiones que tal vez no tenga los suficientes conocimientos o sofisticación para tomar.

Así, las arquitecturas más limpias por lo regular sólo tienen un número muy reducido de modos de direccionamiento, con límites estrictos para su uso. En la práctica, tener los modos inmediato, directo, de registro e indexado es suficiente para casi todas las aplicaciones. Además, todos los registros (incluidos el apuntador a variables locales, el apuntador a la pila y el contador de programa) deberán poder usarse cuando se requiera un registro. Los modos de direccionamiento más complejos tal vez reduzcan el número de instrucciones, pero a expensas de introducir secuencias de operaciones que no se pueden ejecutar fácilmente en paralelo con otras operaciones secuenciales.

Ya terminamos nuestro estudio de los diversos equilibrios que puede haber entre códigos de operación y direcciones, y las distintas formas de direccionamiento. Al estudiar una computadora nueva, es recomendable examinar las instrucciones y los modos de direccionamiento no sólo para ver cuáles están disponibles, sino también para entender por qué se tomaron esas decisiones y qué consecuencias habrían tenido otras decisiones.