Capítulo 9. Conjuntos de Instrucciones

Los elementos esenciales de una instrucción de computadora son el código de operación *(opcode),* que especifica la operación a ser realizada; las referencias a los operandos de origen y destino, que especifican las localidades de entrada y salida para la operación; y la referencia la siguiente instrucción, la cual es usualmente implícita.

Los opcodes especifican operaciones en una de las siguientes categorías: operaciones aritméticas y lógicas; movimiento de datos entre registros, registros y memoria, o dos localidades de memoria; entrada y salida; y contro.

Las referencias a operandos especifican un registro o una localidad de memoria donde hay datos para la operación. Los tipos de datos pueden ser direcciones, números, carácteres o datos lógicos.

Los procesadores direccionables por byte pueden estar categorizados como big-endian, little-endian o bi-endian. Un valor numérico multi-byte puede ser almacenado de distintas maneras: si el byte más significativo se encuentra en la dirección numérica más baja entonces es big-endian. Little-endian almacena el byte más significativo en la dirección numérica más alta. Un procesador bi-endian puede manejar ambos estilos.

Usualmente, si un programador está utilizando un lenguaje de alto nivel como C o Pascal, entonces no es visible para él la arquitectura interna de la máquina en la que está trabajando. Un lugar donde el diseñador de la computadora y el programador ven la misma máquina es en el conjunto de instrucciones. Desde el punto de vista del diseñador, el conjunto de instrucciones de la máquina le da las pautas de los requerimientos funcionales del procesador, es decir, implementar el procesador es una tarea que incluye implementar el conjunto de instrucciones.

9.1 Características de las instrucciones máquina

La operación del procesdor está determinada por las instrucciones que ejecuta, referidas como instrucciones máquina o instrucciones de computadora. A la colección de diferentes instrucciones que el procesador puede ejecutar se le llama conjunto de instrucciones.

Elementos de una instrucción

Cada instrucción debe contener la información necesaria para que el procesador pueda ejecutarla. Estos elementos son los siguientes:

- **Código de operación**: especifica la operación a ser realizada (ADD, MOVE, etc.). La operación se especifica mediante un código binario conocido como **opcode**.
- **Referencia al operando origen**: la operación puede involucrar uno o más operandos origen, es decir, operandos de entrada.
- **Referencia al operando resultado**: la operación puede producir un resultado.
- **Referencia a la siguiente instrucción**: le dice al procesador de dónde traer la siguiente instrucción una vez que la ejecución de la instrucción actual termine.

La dirección de la siguiente instrucción a ser traída puede ser una dirección real o una dirección virtual, dependiendo de la arquitectura. Generalmente, la siguiente instrucción a ser traída es la contigua a la instrucción en ejecución. Cuando es necesaria una referencia explícita, entonces se provee con una dirección de memoria principal o virtual. La manera en que dicha dirección se provee se discute en el siguiente capítulo.

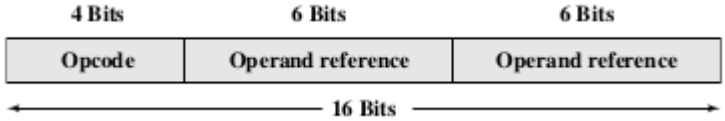
Los operandos de origen y destino pueden encontrarse en una de cuatro áreas:

- Memoria principal o memoria virtual: como con las referencias a siguiente instrucción, se debe proveer de una dirección de memoria.
- Registro del procesador: el procesador contiene uno o más registros internos que pueden ser referenciados por instrucciones. Si únicamente existe un registro, la referencia a él puede ser implícita. Si hay más de un registro, entonces a cada registro se asigna un nombre o número único y la instrucción deberá contener el número del registro deseado.
- Inmmediato: el valor del operando se encuentra en uno de los campos de la instrucción en ejecución.

- Dispositivo I/O: la instrucción deberá especificar el módulo I/O y el dispositivo para la operación.

Representación de instrucciones

Dentro de una computadora, cada instrucción es representada como una secuencia de bits. La instrucción se divide en campos, que corresponden a los elementos que la constituyen. A continuación se presenta un ejemplo simple de un formato de instrucción.



En la mayoría de conjuntos de instrucciones, se utiliza más de un formato de instrucción. Durante la ejecución de una instrucción, la instrucción se carga en el registro de instrucción del procesador (IR). El procesador debe ser capaz de extraer la información de los diferentes campos para efectual la operación requerida.

Es difícil para los programadores lidiar con las representaciones binarias de las instrucciones máquina. Es por ello que se ha convertido en una práctica común utilizar una representación simbólica de las instrucciones. Los opcodes son representados con abreviaciones que indican la operación. Algunos ejemplos comunes son: ADD (suma), SUB (resta), MUL (multiplicación), DIV (división), LOAD (cargar datos de memoria), STOR (guardar datos en memoria).

Los operandos también son representados simbólicamente, por ejemplo la instrucción ADD R, Y puede significar sumar el valor contenido en la localidad Y a los contenidos del registro R. En este ejemplo Y se refiere a la dirección de una localidad de memoria y R se refiere a un registro en particular. Note que la operación se efectúa sobre los contenidos de la localidad y no su dirección. Así, es posible escribir un programa de lenguaje máquina en forma simbólica. Un programa simple puede aceptar esta entrada simbólica, convertir los opcodes y referencias a operandos a su forma binaria y construir las instrucciones máquina.

Tipos de instrucciones

Considere una instrucción de alto nivel, por ejemplo X = X + Y. La sentencia anterior instruye a la computadora a sumar el valor almacenado en Y al valor almacenado en X y poner el resultado de dicha operación en X. ¿Cómo puede ser lo anterior logrado con instrucciones máquina? Asumamos que las variables X y Y corresponden a las localidades 513 y 514. Si asumimos un set simple de instrucciones máquina, esta operación puede ser lograda con 3 instrucciones:

1. Carge un registro con los contenidos de la localidad de memoria 513.
2. Sume los contenidos de la localidad de memoria 514 al registro.
3. Guarde los contenidos del registro en la localidad de memoria 513.

Como se puede ver, una simple instrucción de alto nivel puede requerir de 3 instrucciones máquina. Ésto es típico de la relación entre los lenguajes de alto nivel y el lenguaje máquina. Un lenguaje de alto nivel expresa las operaciones en forma algebraica y concisa, usando variables. El lenguaje máquina expresa estas operaciones en forma básica involucrando el movimiendo de datos desde y hacia los registros.

El conjunto de instrucciones máquina debe ser suficiente para expresar cualquiera de las instrucciones de un lenguaje de alto nivel. Con lo anterior en mente, se pueden categorizar las instrucciones de la manera siguiente:

- **Procesamiento de datos:** instrucciones artiméticas y lógicas.
- **Almacenamiento de datos:** movimiento de datos hacia o desde un registro y/o localidades de memoria.
- **Movimiento de datos:** instrucciones I/O
- **Control:** instrucciones de prueba y de bifurcación.

Las instrucciones artiméticas proveen capacidades computacionales para procesar datos numéricos. Las instrucciones lógicas o Booleanas operan sobre los bits de una palabra en lugar de utilizar la palabra como un

número. Estos dos tipos de operaciones se realizan principalmente sobre datos que se encuentran en registros del procesador. Así, deberán existir instrucciones de memoria para mover datos entre la memoria y los registros. Las instrucciones I/O son necesarias para transferir programas y datos hacia memoria y después enviar los resultados de vuelta al usuario. Las instrucciones de prueba se utilizan para probar el valor de una palabra de datos o el estatus de una operación. Las instrucciones de bifurcación se utilizan para cambiar el flujo de ejecución de un programa a un conjunto diferente de instrucciones dependiendo de alguna decisión.

Número de Direcciones

Una manera tradicional de describir la arquitectura de un procesador es en términos de la cantidad de direcciones contenida en cada instrucción. Lo anterior se ha vuelto cada vez menos significativo con la creciente complejidad del diseño de procesadores.

¿Cuál es el número máximo de direcciones que una instrucción puede necesitar? Evidentemente, las instrucciones artiméticas y lógicas requieren de más operandos. Virtualmente todas las operaciones aritméticas y lógicas son únarias (un operando origen) o binarias (dos operandos origen). Así, se necesitaría un máximo de dos direcciones para referenciar los operandos origen. El resultado de una operación debe ser almacenado, lo cual sugiere el uso de una tercera dirección, que define al operando destino. Finalmente, después de terminar de ejecutar una instrucción, la siguiente instrucción debe ser obtenida, por lo que se requiere su dirección.

Esta manera de pensar sugiere que una instrucción puede requerir contener cuatro direcciones: dos operandos origen, un operando destino, y la dirección de la siguiente instrucción. En la mayoría de las arquitecturas, las instrucciones tienen uno, dos o hasta tres direcciones de operando, la dirección de la siguiente instrucción es implícita (se obtiene del contador de programa).

La figura siguiente compara las instrucciones típicas de uno, dos y tres direcciones que se pueden utilizar para efectuar el cálculo Y = (A – B) / [C + (D x E)]. Con tres direcciones, cada instrucción especifica dos operandos origen y un operando destino. Como elegimos no alterar el valor de ninguno de los operandos, se utiliza la localidad temporal T para almacenar los resultados intermedios. Con 3 direcciones se logra el cálculo completo en 4 instrucciones.

Instrucciones de tres direcciones son poco comunes porque requieren de formatos de instrucción grandes para poder almacenar las tres direcciones de referencia. Con instrucciones de dos direcciones, y para operaciones binarias, de una dirección, una dirección deberá cumplir la doble función de operando y resultado. Así, la instrucción SUB Y, B realiza el cálculo Y – B y almacena el resultado en Y. El formato de dos direcciones reduce los requerimientos de espacio pero introduce cierta incomodidad. Para evitar alterar el valor de un operando, se utiliza una instrucción MOVE para mover uno de los valores a una localidad temporal antes de efectuar la operación. Con 2 direcciones se logra el cálculo completo en 6 instrucciones.

Instruction	Comment
SUB Y, A, B	Y ← A − B
MPY T, D, E	T ← D × E
ADD T, T, C	T ← T + C
DIV Y, Y, T	Y ← Y ÷ T

(a) Three-address instructions

Instruction	Comment
MOVE Y, A	Y ← A
SUB Y, B	Y ← Y − B
MOVE T, D	T ← D
MPY T, E	T ← T × E
ADD T, C	T ← T + C
DIV Y, T	Y ← Y ÷ T

(b) Two-address instructions

Instruction	Comment
LOAD D	AC ← D
MPY E	AC ← AC × E
ADD C	AC ← AC + C
STOR Y	Y ← AC
LOAD A	AC ← A
SUB B	AC ← AC − B
DIV Y	AC ← AC ÷ Y
STOR Y	Y ← AC

(c) One-address instructions

Las instrucciones de una dirección son aún más simples. Para que funcionen, una segunda dirección debe ser implícita. Ésto era común en las primeras máquinas, en donde la dirección implícita era un registro del procesador llamado **acumulador (AC)**. El acumulador contiene uno de los operandos y se utiliza para almacenar el resultado. En el ejemplo, se utilizan 8 instrucciones de una dirección para completar el cálculo.

Es incluso posible que existan instrucciones sin direcciones explícitas como son las operaciones de un stack donde las instrucciones POP y PUSH tienen su única dirección de operando implícita.

La siguiente tabla muestra la interpretación de las instrucciones de 3, 2, 1 o 0 direcciones. En cada caso se asume que la dirección de la siguiente instrucción es implícita.

Number of Addresses	Symbolic Representation	Interpretation
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T − 1) OP T

AC = accumulator
T = top of stack
(T − 1) = second element of stack
A, B, C = memory or register locations

El número de direcciones por instrucción es una decisión de diseño básica. Menor cantidad de direcciones resulta en instrucciones que son más primitivas, que requieren un procesador menos complejo. También tiene como resultado instrucciones de menor longitud. Por otro lado, los programas contendrán mayor cantidad de instrucciones, lo que en general resulta en un tiempo de ejecución mayor y programas más complejos. Además, existe una consideración adicional sobre utilizar instrucciones de una o múltiples direcciones. Con instrucciones de una dirección, el programador generalmente tiene a su disposición únicamente un solo registro de propósito general, el acumulador. Con instrucciones de múltiples direcciones, es común tener varios registros de propósito general. Como las referencias a registros son más rápidas que las referencias a memoria, esto acelera la ejecución de los programas. Por cuestiones de flexibilidad y habilidad de utilizar múltiples registros, la mayoría de las máquinas contemporáneas utilizan una mezcla de instrucciones de dos y tres direcciones.

Otros factores también intervienen en la decisión de cuántas direcciones utilizar. Por ejemplo, la cuestión de si una dirección hace referencia a una localidad de memoria o un registro. Como existen menos registros, se necesitan menos bits para referenciarlos. Además, una máquina puede ofrecer una variedad de modos de direccionamiento, y la especificación del modo toma uno o dos bits. El resultado es que la mayoría de los diseños involucran una variedad de formatos de instrucción.

Diseño del conjunto de instrucciones

El diseño del conjunto de instrucciones es muy complejo porque afecta muchos aspectos de la computadora. El conjunto de instrucciones define muchas de las funciones realizadas por el procesador y, por tanto, tiene un efecto significativo en la implementación del mismo. El conjunto de instrucciones es la manera en que el programador controla al procesador, así, los requerimientos del programador también deben ser tomados en cuenta al diseñarlo.

Podría ser sorprendente que muchas cuestiones fundamentales del diseño de conjuntos de instrucciones continúan en disputa. De hecho, en años recientes, el nivel de desacuerdo sobre estas cuestiones ha crecido. Las cuestiones más importantes incluyen las siguientes:

- **Repertorio de operaciones:** cuántas y cuáles operaciones deben soportarse, y qué tan complejas son.
- **Tipos de datos:** los varios tipos de datos sobre los cuales se realizan las operaciones.
- **Formato de instrucción:** longitud de las instruccio.0nes (en bits), número de direcciones, tamaño de los diversos campos, etc.
- **Registros:** número de registros del procesador que pueden ser referenciados por las instrucciones y su uso.
- **Direccionamiento:** El modo o modos por los cuales se especifica la dirección de un operando.

Estas cuestiones están muy relacionadas entre sí y deben ser consideradas de manera conjunta al diseñar el conjunto de instrucciones.

9.2 Tipos de operandos

Las instrucciones máquina operan sobre datos. Las categorías más importantes de datos son: direcciones, números, caractéres y datos lógicos. Se discutirá posteriormente que las direcciones son, de hecho, una forma de datos. Los otros 3 tipos se examinan a continuación.

Números

Todos los lenguajes máquina incluyen tipos de datos numéricos. Incluso en el procesamiento de datos no numéricos, existe la necesidad de números que actúen como contadores, tamaño de campos, etc. Tres tipos de datos numéricos son comunes en las computadoras:

- Enteros binarios o de punto fijo
- Números de punto flotante
- Decimales

Los primeros dos fueron examinados en el capítulo anterior. En cuanto a los números decimales, si bien es cierto que todas las operaciones internas de la computadora son binarias por naturaleza, los usuarios humanos del sistema lidian con números decimales. Así, es necesario efectuar una conversión de decimal a binario al introducir los datos y de binario a decimal al presentar los resultados. Para aplicaciones en las cuales se trate con una gran cantidad de I/O y cálculos computacionales, comparativamente, simples y pequeños, es preferible almacenar y operar con los datos en forma decimal. La representación más común para este propósito es el **decimal empaquetado**.

Con el decimal empaquetado, cada dígito decimal se representa con un código de 4 bits. Así 0 = 0000 y 9 = 1001. Note que lo anterior es bastante ineficiente puesto que sólo 10 de las 16 posibles combinaciones son utilizadas. Para formar los números, se concatenan códigos de 4 bits, usualmente en múltiplos de 8 bits (por ejemplo, 246 = 0000 0010 0100 0110). Este código es mucho menos compacto que la representación binaria pero evita la necesidad de conversión.

Caracteres

Si bien, los datos tipo texto son muy convenientes para los humanos, no pueden ser almacenados y trasmitidos en su forma de caracteres. Así, un número de codigos se han diseñado para representar caracteres como cadenas de bits. Probablemente el más utilizado en la actualidad es el Código Estándar Americano para el Intercambio de Información o ASCII. Cada caracter se representa como un patrón único de 7-bits, por tanto, se pueden representar 128 caracteres diferentes. Algunos de estos patrones se utilizan como caracteres de **control**: utilizados para controlar la impresión de los otros caracteres en la página (ej. Salto de línea) o para ayudar en los procedimientos de comunicación (ej. Inicio/fin de transmisión). Los caracteres ASCII casi siempre son almacenados y transmitidos en 8 bits. El octavo bit se utiliza como bit de paridad.

Datos lógicos

Normalmente, cada palabra o cualquier otra unidad direccionable (byte por ejemplo) es tratada como un sola unidad. En algunas ocasiones es útil considerar una unidad de n bits como n elementos de 1 bit. Cuando los datos se consideran de esta manera, se les llama datos lógicos. Existen dos ventajas del enfoque lógico. Primero, en ocasiones se desea almacenar un arreglo booleano en el que cada elemento pueda tomar sólo dos valores 1 (verdadero) y 0 (falso). Con datos lógicos, la memoria puede ser utilizada más eficientemente para este tipo de arreglos. Segundo, hay ocaciones en las que deseamos manipular los bits individuales de un elemento de datos. Por ejemplo, si las operaciones de punto flotante son implementadas en software, necesitamos poder desplazar los bits significativos en algunas operaciones.

Note que, en el ejemplo anterior, los mismos datos son utilizados algunas veces como lógicos y otras como numéricos. El “tipo” de datos está determinado por la operación que se realiza sobre ellos.

9.3 Tipos de operaciones

El número de opcodes diferentes varía de máquina a máquina. Sin embargo, los mismos tipos generales de operaciones se encuentran en todas las máquinas. Una categorización útil es la siguiente:

- Transferencia de datos
- Aritméticas
- Lógicas
- Conversión
- I/O
- Control de sistema
- Transferencia de control

Transferencia de datos

Type	Operation Name	Description
Data Transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination

El tipo de instrucción más fundamental es la transferencia de datos. Estas instrucciones deben especificar varias cosas. Primero, la ubicación de los operandos de origen y destino. Cada ubicación puede ser memoria, un registro o el tope de un stack. Segundo, la longitud de los datos a ser transferidos. Tercero, como con todas las instrucciones con operandos, el modo de direccionamiento para cada operando.

La elección de instrucciones de transferencias de datos a incluír en el conjunto de instrucciones ejemplifica los tipos de compromisos que el diseñador debe hacer. Por ejemplo, la ubicación general de un operando (en memoria o en registro) puede especificarse ya sea en el opcode o en el mismo operando. La tabla anterior muestra ejemplos de las instrucciones de transferencias de datos de la IBM EAS/390 más comunes. Note que existe variantes para indicar la cantidad de datos a trasnferir (8, 16, 32 o 64 bits). También hay diferentes instrucciones para transferencias registro a registro, registro a memoria, memoria a registro, y memoria a memoria. En contraste, la VAX tiene una instrucción MOV que especifica si un operando está en registro o en memoria como parte del mismo operando.

Operation Mnemonic	Name	Number of Bits Transferred	Description
L	Load	32	Transfer from memory to register
LH	Load Halfword	16	Transfer from memory to register
LR	Load	32	Transfer from register to register
LER	Load (Short)	32	Transfer from floating-point register to floating-point register
LE	Load (Short)	32	Transfer from memory to floating-point register
LDR	Load (Long)	64	Transfer from floating-point register to floating-point register
LD	Load (Long)	64	Transfer from memory to floating-point register
ST	Store	32	Transfer from register to memory
STH	Store Halfword	16	Transfer from register to memory
STC	Store Character	8	Transfer from register to memory
STE	Store (Short)	32	Transfer from floating-point register to memory
STD	Store (Long)	64	Transfer from floating-point register to memory

En términos de acción para el procesador, las operaciones de transferencia de datos son probablemente las más simples. Si el origen y el destino son registros, el procesador simplemente mueve los datos de un registro a otro; dicha operación es intera para el procesador y no involucra a los buses, caché o memoria principal. Si uno o ambos operandos están en memoria, entonces el procesador deberá realizar algunas o todas las siguientes acciones:

1. Calcular la dirección de memoria, basándose en el modo de direccionamiento.
2. Si la dirección se refiere a memoria virtual, traducir de memoria virtual a memoria real.
3. Determinar si se encuentra en caché.
4. Si no está en caché, dar un comando al módulo de memoria.

Aritméticas

Arithmetic	Add	Compute sum of two operands
	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand

La mayoría de las máquinas proveen las operaciones artiméticas básicas de suma, resta, multiplicación y división. Éstas son invariablemente implementadas para enteros con signo, aunque seguido también se implementan para punto flotante y números decimales empaquetados.

Otras posibles operaciones incluyen una variedad de instrucciones de un operando, por ejemplo: valor absoluto, negación, incremento, decremento. La ejecución de instrucciones aritméticas puede involucrar operaciones de transferencias de datos para llevar los operandos a la ALU y posteriormente entregar los resultados.

Lógicas

Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT (complement)	Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end

Las máquinas proveen una variedad de operaciones para manipular bits individuales de una palabra u otras unidades direccionables que se basan en las operaciones Booleanas: AND, OR, XOR, etc. Adicionalmente, las operaciones por bit, se proveen funciones de desplazamiento y rotación. En un desplazamiento **lógico**, los bits de una palabra se mueven a la izquierda o a la derecha. En un extremo, un bit se pierde; en el otro extremo se agrega un 0. El desplazamiento **aritmético**, trata los datos como un entero con signo y no desplaza el bit de signo. En un desplazamiento aritmético hacia la derecha, el bit de signo se replica en la posición a su derecha.

En un desplazamiento aritmético hacia la izquierda, se aplica un desplazamiento lógico hacia la izquierda con todos los bits, pero el bit de signo se retiene. La rotación, o desplazamiento cíclico, preserva todos los bits sobre los que opera.

Input	Operation	Result
10100110	Logical right shift (3 bits)	00010100
10100110	Logical left shift (3 bits)	00110000
10100110	Arithmetic right shift (3 bits)	11110100
10100110	Arithmetic left shift (3 bits)	10110000
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101

Conversión

Conversion	Translate	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

Las instrucciones de conversión son aquellas que cambian el formato u operan sobre el formato de los datos. Un ejemplo es la conversión de decimal a binario.

Entrada/Salida

Input/Output	Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	Output (write)	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination

Para las instrucciones de entrada y salida existen una variedad de enfoques: I/O aislada, I/O mapeada a memoria, DMA, el uso de un procesador I/O independiente. Muchas implementaciones proveen sólo unas pocas instrucciones I/O, donde las acciones específicas son determinadas por parámetros, códigos o palabras de comando.

Control de sistema

Son aquellas que sólo pueden ser ejecutadas mientras el procesador se encuentra en un estado privilegiado o está ejecutando un programa en una área especial privilegiada de la memoria. Típicamente, estas instrucciones se reservan para el uso del sistema operativo. Por ejemplo, una instrucción que lea o modifique la llave de protección de almacenamiento como la que se utiliza en el sistema de memoria del EAS/390. Aotro ejemplo es el acceso a bloques de control de proceso en un sistema de multiprogramación.

Transferencia de control

Transfer of Control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued

Para todos los tipos de operación discutidos hasta ahora, la siguiente instrucción a ser realizada es la que sigue de forma inmediata, en memoria, a la instrucción actual. Sin embargo, una parte significativa de las instrucciones en cualquier programa tienen como función principal cambiar la secuencia de ejecución de las instrucciones. Para estas instrucciones, la operación realizada por el procesador es actualizar el contador de programa con la dirección de alguna instrucción en memoria.

Existen un número de razones por las que las operaciones de transferencia de control (no confundir con control de transferencia) son requeridas:

1. Es esencial el poder ejecutar una instrucción más de una vez y, probablemente, miles de veces. Una aplicación puede requerir miles, tal vez millones, de instrucciones para ser implementada. Esto sería impensable si cada instrucción tuviera que ser escrita de manera individual. Si una tabla o lista de elementos va a ser procesada, es necesario un ciclo: una secuencia de instrucciones que se ejecuta repetitivamente para procesar los datos.
2. Virtualmente todos los programas involucran algún tipo de toma de decisiones. Nos gustaría que la computadora haga una cosa si una condición se cumple, y haga otra cosa si otra condición se cumple. Por ejemplo para calcular la raíz cuadrada de un número, antes de realizar el cálculo se revisa el signo del número, si el número es negativo entonces no se efectúa la operación.
3. Escribir un programa de computadora grande, o incluso uno medianón, es una tarea bastante difícil. Es conveniente tener mecanismos que nos permitan dividir la tarea en pequeñas partes en las que se pueda trabajar de manera independiente una a la vez.

Instrucciones de bifurcación

Una instrucción de bifurcación, también llamadas instrucciones de salto, tiene como uno de sus operandos la dirección de la siguiente instrucción a ser ejecutada. Más comúnes son las **instrucciones de salto condicional**. En ellas, el salto se realiza sólo si cierta condición se cumple. De otra forma, la siguiente instrucción en la secuencia se ejecuta (se incrementa el contador de programa). Una instrucción de bifurcación en donde el salto siempre se efectúa se conoce como **salto incondicional**.

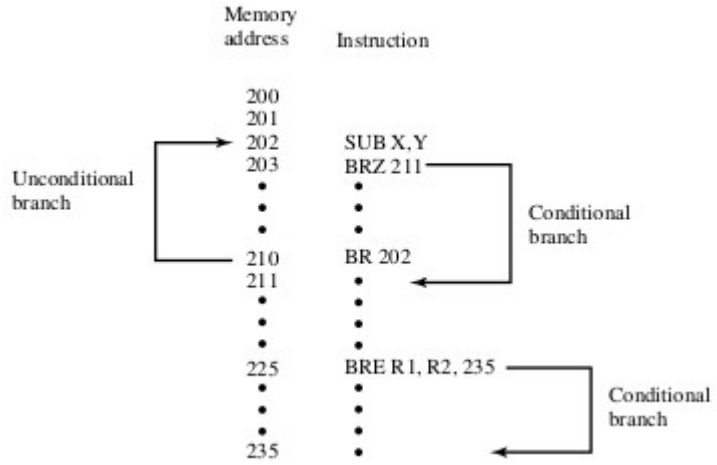
Existe dos formas comunes de generar la condición de una instrucción de salto condicional. Primero, la mayoría de las máquinas proveen un código de condición de 1 o más bits que se establece como resultado de algunas operaciones. Este código puede ser visto como un pequeño registro visible al usuario. Por ejemplo, una operación aritmética puede establecer un código de condición de 2 bits con uno de los siguientes valores: 0,

positivo, negativo o desbordamiento. En dicha máquina, podría haber cuatro instrucciones de salto condicional:

- BRP X – Salta a X si el resultado es positivo
- BRN X – Salta a X si el resultado es negativo
- BRZ X – Salta a X si el resultado es cero
- BRO X – Salta a X si ocurre desbordamiento

Otro enfoque puede ser utilizado con instrucciones de tres direcciones BRE R1, R2, X puede significar saltar a X si los contenidos de R1 y R2 son iguales.

La siguiente figura muestra ejemplos de estas operaciones. Note que un salto puede ser hacia *adelante* (una instrucción con una dirección más alta) o hacia *atrás* (dirección anterior). El ejemplo muestra cómo un salto condicional y un salto incondicional se pueden utilizar para crear un ciclo de instrucciones. Las instrucciones en las localidades 202 a 210 se ejecutarán repetidamente hasta que el resultado de X-Y sea 0.



Instrucciones de omisión

Otra forma de instrucción de transferencia de control es la instrucción de omisión (*skip instruction*). Estas instrucciones incluyen una dirección implícita. Típicamente, implican que una instrucción debe ser omitida o brincada; así la dirección implícita es igual a la dirección de la siguiente instrucción más el tamaño de una instrucción.

Las instrucciones de omisión no requieren un campo de dirección destino, por ejemplo la instrucción incrementa-y-omite-si-cero (ISZ, *increment-and-skip-if-zero*). Considere el fragmento de programa a la derecha.

```
301
 •
 •
 •
309    ISZ    R1
310    BR     301
311
```

En este framento, las dos instrucciones de transferencia de control se utilizan para implementar un ciclo iterativo. R1 se establece con el netativo del número de iteraciones a realizar. Al final del ciclo, R1 se incrementa. Si no es 0, el programa salta de regreso al inicio del ciclo. De otra manera, el salto se omite, y el programa continúa con la siguiente instrucción.

Instrucciones de llamadas a procedimientos

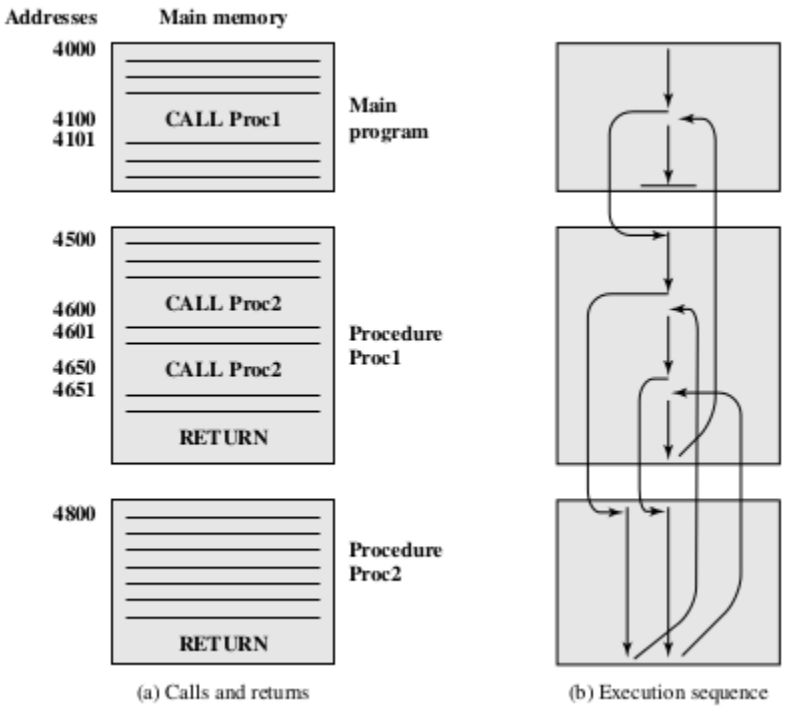
Tal vez la inovación más importante en los lenguajes de programación son los procedimientos. Un procedimiento es un programa de computadora auto-contenido que se incorpora a un programa más grande. En cualquier momento de un programa, un procedimiento puede ser invocado o llamado. El procesador está instruído para ir y ejecutar el procedimiento pro completo y, posteriormente, regresar al punto en que la llamada tuvo lugar.

Las dos razones principales para el uso de procedimeintos son economía y modularización. Un procedimiento permite que la misma pieza de código sea utilizada muchas veces. Ésto es importante para la economía en el esfuerzo de programación y para hacer uso eficiente del espacio de almacenamiento del sistema.

Los procedimientos también permiten que tareas grandes de programación puedan ser subdivididas en pequeñas unidades.

El mecanismo de un procedimiento involucra dos instrucciones básicas: una instrucción de bifurcación que salte de la localidad actual al procedimiento, y una instrucción de retorno que regrese del procedimiento al lugar desde el cuál fue llamado. Ambas son instrucciones de salto.

La figura siguiente ilustra el uso de procedimientos para construir un programa. En éste ejemplo, existe un programa principal que comienza en la localidad 4000. Este progarma incluye una llamada al procedimiento PROC1, que comienza en la localidad 4500. Cuando se encuentra la instrucción de llamada a procedimiento, el procesador suspende la ejecución del programa principal y comienza la ejecución de PROC1 trayendo la siguiente instrucción de la localidad 4500. Dentro de PROC1, existen dos llamadas a PROC2 en la localidad 4800. En cada caso, la ejecución de PROC1 se suspende y PROC2 es ejecutado. La declaración RETURN provoca que el procesador regrese al programa que realizó la llamada y continúe con su ejecución en la instrucción siguiente a la instrucción CALL.



Existen tres puntos dignos de notar:

1. Un procedimiento puede ser llamado desde más de una localidad.
2. Una llamada a procedimiento puede aparercer en otro procedimiento (anidamiento).
3. Cada llamada a procedimiento es compensada con una llamada a RETURN.

Como nos gustaría llamar procedimientos desde una variedad de lugares, el procesador deberá de alguna manera guardar la dirección a la que debe regresar para que cuando el procedimiento haya concluído se pueda seguir con la ejecución del programa principal. Existen tres lugares comounes para almacenar la dirección de regreso:

- Registro
- Inicio del procedimiento llamado
- Tope de un stack

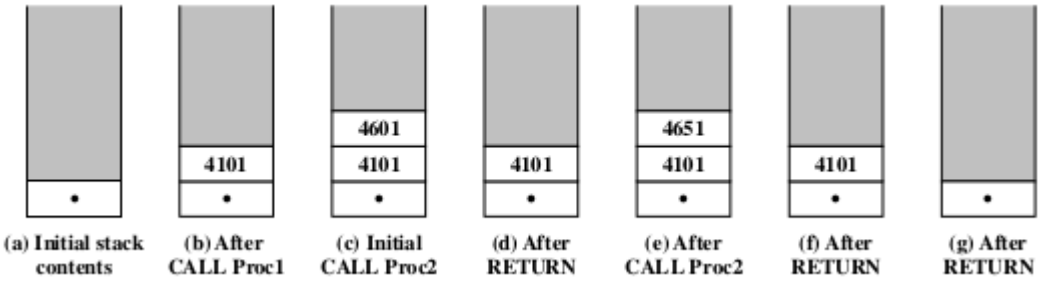
Considere una instrucción CALL X, para llamar al procedimiento en la localidad X. Si se utiliza el enfoque de registro, CALL X provocará las siguientes acciones: RN <- PC + delta, PC <- X donde RN es un registro que siempre es utilizado para éste propósito, PC es el contador de programa y delta es la longitud de la instrucción. El procedimiento llamado puede ahora almacenar el contenido de RN para ser utilizado después.

La segunda posibilidad es almacenar la dirección de regreso al inicio del procedimiento. En este caso, CALL X provoca X <- PC + delta, PC <- X + 1.

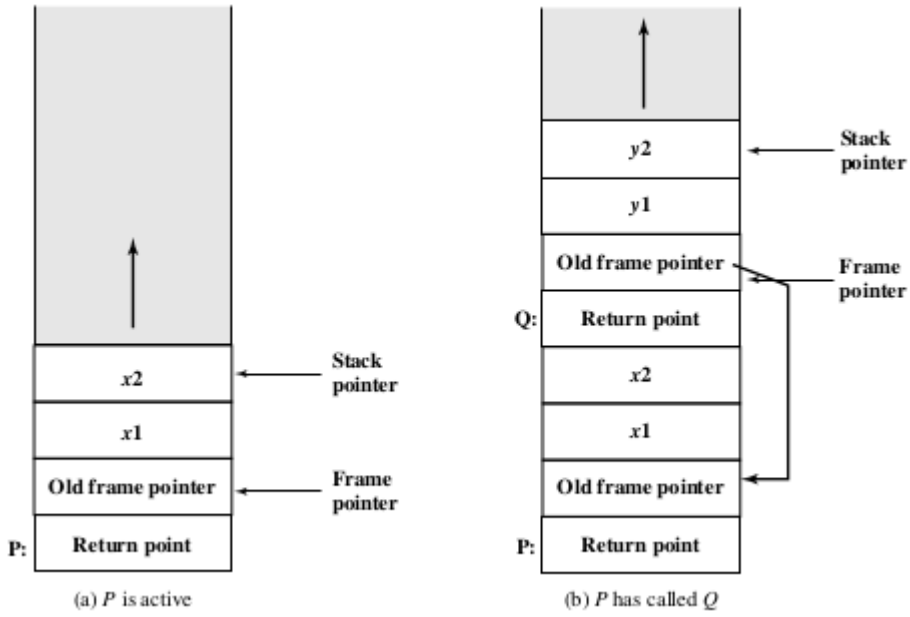
Los dos enfoques anteriores funcionan y han sido utilizados. La única limitación de estos enfoques es que complican el uso de procedimientos *reentrantes*. Un procedimiento *reentrante* es uno en el cual es posible tener varias llamadas abiertas a él al mismo tiempo. Un procedimiento recurrente (uno que se llama a sí mismo) es un ejemplo del uso de esta característica. Si se pasan parámetros a través de registros o memoria para un procedimiento reentrante, parte del código debe ser responsable de guardar dichos parámetros de manera que los registros o espacios de memoria estén disponibles para otras llamadas a procedimiento.

Un enfoque más general y poderoso es el uso de un stack. Cuando el procesador ejecuta una llamada, pone la dirección de regreso al inicio del stack. Cuando se ejecuta un RETURN, se utiliza la dirección del tope

del stack.



Además de proveer una dirección de retorno, frecuentemente es necesario pasar parámetros a las llamadas de procedimiento. Éstos pueden ser pasados en registros. Otra posibilidad es almacenar los parámetros en memoria justo después de la instrucción CALL. En este caso, el retorno deberá ser a la localidad en seguida de los parámetros. De nuevo, ambos enfoques tienen algunos problemas. Si se utilizan registros, el programa llamado y el programa que llama debe ser escrito de manera que asegure que los registros se utilicen adecuadamente. El guardar los parámetros en memoria hace difícil el intercambiar un número variable de parámetros. Ambos enfoques evitan el poder usar procedimientos reentrantes.



Un enfoque más flexible para el paso de parámetros es el stack. Cuando el procesador ejecuta una llamada, no sólo guarda en el stack la dirección de retorno, sino también los parametros a ser pasados al procedimiento. El conjunto entero de parámetros, incluyendo la dirección de retorno, que se almacena para un procedimiento se conoce como *stack frame* o marco de stack. Un ejemlo se muestra en la figura de la izquierda. El procedimiento P tiene declaradas las variables locales x1 y x2; y el procedimiento Q, que puede ser llamado desde P, tiene declaradas las variables y1 y y2. En la figura, el punto de retorno para cada procedimiento es el primer elemento almacenado en el marco de stack correspondiente. Enseguida se almacena un apuntador al inicio del marco anterior. Ésto es necesario si el número o la longitud de los parámetros a ser almacenados es variable.