El procesador

Miquel Albert Orenga
Gerard Enrique Manonellas

PID_00177072

Universitat Oberta de Catalunya
www.uoc.edu

Los textos e imágenes publicados en esta obra están sujetos –excepto que se indique lo contrario– a una licencia de Reconocimiento-Compartir igual (BY-SA) v.3.0 España de Creative Commons. Se puede modificar la obra, reproducirla, distribuirla o comunicarla públicamente siempre que se cite el autor y la fuente (FUOC. Fundació per a la Universitat Oberta de Catalunya), y siempre que la obra derivada quede sujeta a la misma licencia que el material original. La licencia completa se puede consultar en: http://creativecommons.org/licenses/by-sa/3.0/es/legalcode.ca

Índice

Introducción

En este módulo estudiaremos el componente principal de un computador: el procesador, unidad central de proceso o CPU (siglas de la expresión inglesa *central processing unit*).

La función principal que tiene es procesar los datos y transferirlos a los otros elementos del computador. Estas tareas se llevan a cabo mediante la ejecución de instrucciones. Por este motivo, el objetivo principal a la hora de diseñar un procesador es conseguir que las instrucciones se ejecuten de la manera más eficiente posible.

En este módulo nos centraremos en analizar los elementos principales del procesador desde el punto de vista funcional y no entraremos a analizar los aspectos relacionados con la mejora de rendimiento.

Los elementos básicos del procesador que estudiaremos son los siguientes:

- Conjunto de registros.
- Unidad aritmética y lógica.
- Unidad de control.

Del conjunto de registros describiremos la función principal y el tipo de información que pueden almacenar.

De la unidad aritmética y lógica (ALU) veremos la función principal y el tipo de datos con los que se trabaja habitualmente. No entraremos en más detalle porque las operaciones aritméticas y lógicas ya se han estudiado ampliamente en asignaturas previas de estos estudios.

Finalmente, haremos un estudio detallado de la unidad de control en el que veremos que esta es el elemento clave para el funcionamiento correcto del procesador y la ejecución de las instrucciones.

Objetivos

Con el estudio de este módulo se pretende que el estudiante alcance los objetivos siguientes:

1. Entender la organización de un procesador en lo relativo a las unidades funcionales que lo componen: registros, ALU y unidad de control.
2. Entender cómo ejecuta una instrucción un procesador.
3. Conocer la organización del conjunto de registros del procesador.
4. Ver los conceptos básicos relacionados con la ALU.
5. Entender el funcionamiento de la unidad de control del procesador.
6. Conocer las ideas clave para diseñar una unidad de control.
7. Saber las diferencias principales entre computadores RISC y CISC.

1. Organización del procesador

La función principal de un procesador es ejecutar instrucciones y la organización que tiene viene condicionada por las tareas que debe realizar y por cómo debe hacerlo.

Los procesadores están diseñados y operan según una señal de sincronización. Esta señal, conocida como *señal de reloj*, es una señal en forma de onda cuadrada periódica con una determinada frecuencia. Todas las operaciones hechas por el procesador las gobierna esta señal de reloj: un ciclo de reloj determina la unidad básica de tiempo, es decir, la duración mínima de una operación del procesador.

Frecuencia de la señal de reloj

La frecuencia de la señal de reloj se define como el número de impulsos por unidad de tiempo, se mide en ciclos por segundo o hercios (Hz) y determina la velocidad de operación del procesador.

Para ejecutar una instrucción, son necesarios uno o más ciclos de reloj, dependiendo del tipo de instrucción y de los operandos que tenga.

Las prestaciones del procesador no las determina solo la frecuencia de reloj, sino otras características del procesador, especialmente del diseño del juego de instrucciones y la capacidad que tiene para ejecutar simultáneamente múltiples instrucciones.

Para ejecutar las instrucciones, todo procesador dispone de tres componentes principales:

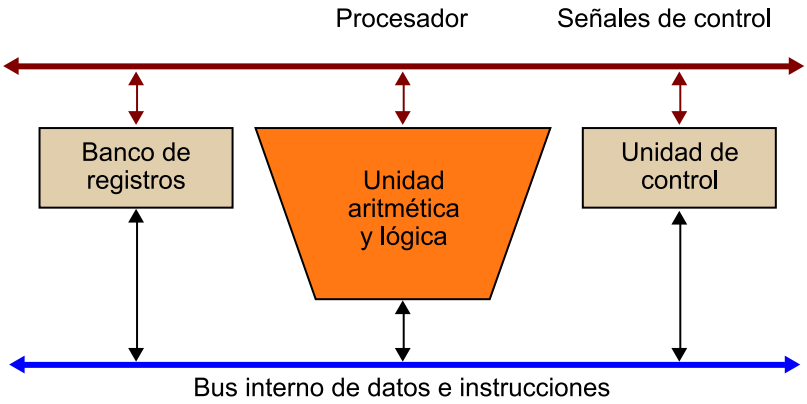
1) Un **conjunto de registros**: espacio de almacenamiento temporal de datos e instrucciones dentro del procesador.

2) Unidad aritmética y lógica o ALU[1]: circuito que hace un conjunto de operaciones aritméticas y lógicas con los datos almacenados dentro del procesador.

3) Unidad de control: circuito que controla el funcionamiento de todos los componentes del procesador. Controla el movimiento de datos e instrucciones dentro y fuera del procesador y también las operaciones de la ALU.

(1) *ALU* son las siglas de la expresión inglesa *arithmetic logic unit*.

La organización básica de los elementos que componen un procesador y el flujo de información entre los diferentes elementos se ve en el esquema siguiente:



Como se observa, aparte de los tres componentes principales, es necesario disponer de un sistema que permita interconectar estos componentes. Este sistema de interconexión es específico para cada procesador. Distinguimos dos tipos de líneas de interconexión: líneas de control, que permiten gobernar el procesador, y líneas de datos, que permiten transferir los datos y las instrucciones entre los diferentes componentes del procesador. Este sistema de interconexión tiene que disponer de una interfaz con el bus del sistema.

El término *procesador* actualmente se puede entender como microprocesador porque todas las unidades funcionales que forman el procesador se encuentran dentro de un chip, pero hay que tener presente que, por el aumento de la capacidad del nivel de integración, dentro de los microprocesadores se pueden encontrar otras unidades funcionales del computador. Por ejemplo:

- **Unidades de ejecución SIMD**: unidades especializadas en la ejecución de instrucciones SIMD (*single instruction, multiple data*), instrucciones que trabajan con estructuras de datos vectoriales, como por ejemplo instrucciones multimedia.

- **Memoria caché**: prácticamente todos los procesadores modernos incorporan dentro del propio chip del procesador algunos niveles de memoria caché.

- **Unidad de gestión de memoria** o ***memory management unit*** **(MMU)**: gestiona el espacio de direcciones virtuales, traduciendo las direcciones de memoria virtual a direcciones de memoria física en tiempo de ejecución. Esta traducción permite proteger el espacio de direcciones de un programa del espacio de direcciones de otros programas y también permite separar el espacio de memoria del sistema operativo del espacio de memoria de los programas de usuario.

- **Unidad de punto flotante** o ***floating point unit*** **(FPU)**: unidad especializada en hacer operaciones en punto flotante; puede funcionar de manera autónoma, ya que dispone de un conjunto de registros propio.

2. Ciclo de ejecución de las instrucciones

El ciclo de ejecución es la secuencia de operaciones que se hace para ejecutar cada una de las instrucciones. Lo dividimos en cuatro fases principales:

1) Lectura de la instrucción.
2) Lectura de los operandos fuente.
3) Ejecución de la instrucción y almacenamiento del operando de destino.
4) Comprobación de interrupciones.

Orden de operaciones

Las fases principales del ciclo de ejecución son comunes a la mayoría de los computadores actuales y las diferencias principales se encuentran en el orden en el que se realizan algunas de las operaciones de cada fase o en el que se realizan algunas de las operaciones en otras fases.

En cada fase se incluyen una o varias operaciones que hay que hacer. Para llevar a cabo estas operaciones, nombradas *microoperaciones*, es necesaria una compleja gestión de las señales de control y de la disponibilidad de los diferentes elementos del procesador, tarea realizada por la unidad de control.

Veamos las operaciones principales que se realizan habitualmente en cada una de las cinco fases del ciclo de ejecución de las instrucciones:

Fase 1. Lectura de la instrucción. Las operaciones realizadas en esta fase son las siguientes:

a) Leer la instrucción. Cuando leemos la instrucción, el registro contador del programa (PC) nos indica la dirección de memoria donde está la instrucción que hemos de leer. Si el tamaño de la instrucción es superior a la palabra de la memoria, hay que hacer tantos accesos a la memoria como sean necesarios para leerla completamente y cargar toda esta información en el registro de instrucción (IR).

b) Descodificar la instrucción. Se identifican las diferentes partes de la instrucción para determinar qué operaciones hay que hacer en cada fase del ciclo de ejecución. Esta tarea la realiza la unidad de control del procesador leyendo la información que hemos cargado en el registro de instrucción (IR).

c) Actualizar el contador del programa. El contador del programa se actualiza según el tamaño de la instrucción, es decir, según el número de accesos a la memoria que hemos hecho para leer la instrucción.

Actualización del PC

La actualización del contador del programa es una operación que se puede encontrar en otras fases de la ejecución de la instrucción.

Las operaciones de esta fase son comunes a los diferentes tipos de instrucciones que encontramos en el juego de instrucciones.

Fase 2. Lectura de los operandos fuente. Esta fase se debe repetir para todos los operandos fuente que tenga la instrucción.

Las operaciones que hay que realizar en esta fase dependen del modo de direccionamiento que tengan los operandos: para los más simples, como el inmediato o el directo a registro, no hay que hacer ninguna operación; para los indirectos o los relativos, hay que hacer cálculos y accesos a memoria. Si el operando fuente es implícito, vamos a buscar el dato en el lugar predeterminado por aquella instrucción.

Ved también

Los modos de direccionamiento se explican en el apartado 2 del módulo "Juego de instrucciones".

Es fácil darse cuenta de que dar mucha flexibilidad a los modos de direccionamiento que podemos utilizar en cada instrucción puede afectar mucho al tiempo de ejecución de una instrucción.

Por ejemplo, si efectuamos un ADD R3,3 no hay que hacer ningún cálculo ni ningún acceso a memoria para obtener los operandos; en cambio, si efectuamos un ADD [R1], [R2+16], hay que hacer una suma y dos accesos a memoria.

Fase 3. Ejecución de la instrucción y almacenamiento del operando de destino (resultado)

Las operaciones que hay que realizar en esta fase dependen del código de operación de la instrucción y del modo de direccionamiento que tenga el operando destino.

Ejecución de la instrucción

Las operaciones que se llevan a cabo son diferentes para cada código de operación. Durante la ejecución, además de obtener el resultado de la ejecución de la instrucción, se pueden modificar los bits de resultado de la palabra de estado del procesador.

Almacenamiento del operando de destino (resultado)

La función básica es recoger el resultado obtenido durante la ejecución y guardarlo en el lugar indicado por el operando, a menos que el operando sea implícito, en cuyo caso se guarda en el lugar predeterminado por aquella instrucción. El operando de destino puede ser uno de los operandos fuente; de esta manera, no hay que repetir los cálculos para obtener la dirección del operando.

Fase 4. Comprobación de interrupciones

Las interrupciones son el mecanismo mediante el cual un dispositivo externo al procesador puede interrumpir el programa que está ejecutando el procesador con el fin de ejecutar otro programa (una rutina de servicio a la interrupción o RSI) para dar servicio al dispositivo que ha producido la interrupción.

La petición de interrupción se efectúa activando alguna de las líneas de petición de las que dispone el procesador.

Ved también

El concepto de interrupción se explica en detalle en el apartado 3 dentro del módulo "Sistema de entrada/salida".

En esta fase se verifica si se ha activado alguna línea de petición de interrupción del procesador en el transcurso de la ejecución de la instrucción. Si no se ha activado ninguna, continuamos el proceso normalmente; es decir, se acaba la ejecución de la instrucción en curso y se empieza la ejecución de la instrucción siguiente. En caso contrario, hay que transferir el control del procesador a la rutina de servicio de interrupción que debe atender esta interrupción y hasta que no se acabe no podemos continuar la ejecución de la instrucción en curso.

Para atender la interrupción, es necesario llevar a cabo algunas operaciones (intentaremos almacenar la mínima información para que el proceso sea tan rápido como se pueda) para transferir el control del procesador a la rutina de servicio de interrupciones, de manera que, después, lo podamos recuperar con la garantía de que el procesador estará en el mismo estado en el que estaba antes de transferir el control a la rutina de servicio de interrupción:

- Almacenar el contador del programa y la palabra de estado (generalmente, se guardan en la pila).
- Almacenar la dirección donde empieza la rutina para atender la interrupción en el contador del programa.
- Ejecutar la rutina de servicio de interrupción.
- Recuperar el contador del programa y la palabra de estado.

Esta fase actualmente está presente en todos los computadores, ya que todos utilizan E/S para interrupciones y E/S para DMA.

Ved también

En el módulo didáctico "Sistema de entrada/salida" veremos que los ordenadores requieren líneas de petición de interrupción que también se pueden utilizar para otras tareas que no son específicas de E/S.

2.1. Segmentación de las instrucciones

La segmentación de las instrucciones (*pipeline*) consiste en dividir el ciclo de ejecución de las instrucciones en un conjunto de etapas. Estas etapas pueden coincidir o no con las fases del ciclo de ejecución de las instrucciones.

El objetivo de la segmentación es ejecutar simultáneamente diferentes etapas de distintas instrucciones, lo cual permite aumentar el rendimiento del procesador sin tener que hacer más rápidas todas las unidades del procesador (ALU, UC, buses, etc.) y sin tener que duplicarlas.

La división de la ejecución de una instrucción en diferentes etapas se debe realizar de tal manera que cada etapa tenga la misma duración, generalmente un ciclo de reloj. Es necesario añadir registros para almacenar los resultados intermedios entre las diferentes etapas, de modo que la información generada en una etapa esté disponible para la etapa siguiente.

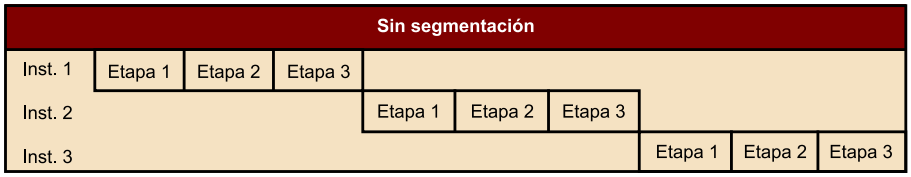
Ejemplo de segmentación de instrucciones

La segmentación es como una cadena de montaje. En cada etapa de la cadena se lleva a cabo una parte del trabajo total y cuando se acaba el trabajo de una etapa, el producto pasa a la siguiente y así sucesivamente hasta llegar al final.

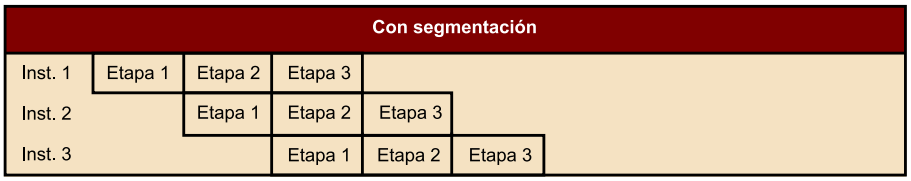
Si hay *N* etapas, se puede trabajar sobre *N* productos al mismo tiempo y, si la cadena está bien equilibrada, saldrá un producto acabado en el tiempo que se tarda en llevar a cabo una de las etapas. De esta manera, no se reduce el tiempo que se tarda en hacer un producto, sino que se reduce el tiempo total necesario para hacer una determinada cantidad de productos porque las operaciones de cada etapa se efectúan simultáneamente.

Si consideramos que el producto es una instrucción y las etapas son cada una de las fases de ejecución de la instrucción, que llamamos *etapa de segmentación*, hemos identificado los elementos y el funcionamiento de la segmentación.

Veámoslo gráficamente. Presentamos un mismo proceso con instrucciones de tres etapas, en el que cada etapa tarda el mismo tiempo en ejecutarse, resuelto sin segmentación y con segmentación:



Son necesarias nueve etapas para ejecutar las tres instrucciones.



Son necesarias cinco etapas para ejecutar las tres instrucciones.

Se ha reducido el tiempo total que se tarda en ejecutar las tres instrucciones, pero no el tiempo que se tarda en ejecutar cada una de las instrucciones.

Si se quiere implementar la segmentación para mejorar el rendimiento de un procesador, hay que tener en cuenta algunas cuestiones que implicarán cambios en el diseño del procesador. Las más importantes son las siguientes:

- ¿Cómo hay que hacerlo para que las etapas estén bien equilibradas (que todas tengan la misma duración)?

- ¿Qué recursos son necesarios en cada etapa para que todas se puedan ejecutar simultáneamente?

- ¿Cómo hay que tratar las instrucciones de salto para minimizar los efectos en el rendimiento de la segmentación?

Nota

Estas cuestiones no son nada sencillas de contestar, pero no es el objetivo de este curso entrar en el detalle de esta problemática. Solo pretendemos que entendáis el funcionamiento de la ejecución de las instrucciones utilizando segmentación.

3. Registros

Los registros son, básicamente, elementos de memoria de acceso rápido que se encuentran dentro del procesador. Constituyen un espacio de trabajo para el procesador y se utilizan como un espacio de almacenamiento temporal. Se implementan utilizando elementos de memoria RAM estática (*static RAM*). Son imprescindibles para ejecutar las instrucciones, entre otros motivos, porque la ALU solo trabaja con los registros internos del procesador.

El conjunto de registros y la organización que tienen cambia de un procesador a otro; los procesadores difieren en el número de registros, en el tipo de registros y en el tamaño de cada registro.

Organización de registros

Para ver cómo se organizan los registros y cómo se utilizan en una arquitectura concreta, podéis consultar los módulos "Programación en ensamblador (x86-64)" y "Arquitectura CISCA".

Una parte de los registros pueden ser visibles para el programador de aplicaciones, otra parte solo para instrucciones privilegiadas y otra solo se utiliza en el funcionamiento interno del procesador.

Una posible clasificación de los registros del procesador es la siguiente:

- Registros de propósito general.
- Registros de instrucción.
- Registros de acceso a memoria.
- Registros de estado y de control.

3.1. Registros de propósito general

Los registros de propósito general son los registros que suelen utilizarse como operandos en las instrucciones del ensamblador. Estos registros se pueden asignar a funciones concretas: datos o direccionamiento. En algunos procesadores todos los registros se pueden utilizar para todas las funciones.

Los registros de datos se pueden diferenciar por el formato y el tamaño de los datos que almacenan; por ejemplo, puede haber registros para números enteros y para números en punto flotante.

Los registros de direccionamiento se utilizan para acceder a memoria y pueden almacenar direcciones o índices. Algunos de estos registros se utilizan de manera implícita para diferentes funciones, como por ejemplo acceder a la pila, dirigir segmentos de memoria o hacer de soporte en la memoria virtual.

3.2. Registros de instrucción

Los dos registros principales relacionados con el acceso a las instrucciones son:

- ***Program counter*** **(PC)**: registro contador del programa, contiene la dirección de la instrucción siguiente que hay que leer de la memoria.
- ***Instruction register*** **(IR)**: registro de instrucción, contiene la instrucción que hay que ejecutar.

3.3. Registros de acceso a memoria

Hay dos registros necesarios para cualquier operación de lectura o escritura de memoria:

- ***Memory address register*** **(MAR)**: registro de direcciones de memoria, donde ponemos la dirección de memoria a la que queremos acceder.
- ***Memory buffer register*** **(MBR)**: registro de datos de memoria; registro donde la memoria deposita el dato leído o el dato que queremos escribir.

La manera de acceder a memoria utilizando estos registros es la siguiente:

1) En una **operación de lectura**, se realiza la secuencia de operaciones siguiente:

a) El procesador carga en el registro MAR la dirección de la posición de memoria que se quiere leer.
b) El procesador coloca en las líneas de direcciones del bus el contenido del MAR y activa la señal de lectura de la memoria.
c) El MBR se carga con el dato obtenido de la memoria.

2) En una **operación de escritura**, se realiza la secuencia de operaciones siguiente:

a) El procesador carga en el registro MBR la palabra que quiere escribir en la memoria.
b) El procesador carga en el registro MAR la dirección de la posición de memoria donde se quiere escribir el dato.
c) El procesador coloca en las líneas de direcciones del bus el contenido del MAR y en las líneas de datos del bus, el contenido del MBR, y activa la señal de escritura de la memoria.

3.4. Registros de estado y de control

La información sobre el estado del procesador puede estar almacenada en un registro o en más de uno, aunque habitualmente suele ser un único registro denominado *registro de estado*.

Los bits del registro de estado son modificados por el procesador como resultado de la ejecución de algunos tipos de instrucciones, por ejemplo instrucciones aritméticas o lógicas, o como consecuencia de algún acontecimiento, como las peticiones de interrupción. Estos bits son parcialmente visibles para el programador, en algunos casos mediante la ejecución de instrucciones específicas.

Cada bit o conjunto de bits del registro de estado indica una información concreta. Los más habituales son:

- **Bit de cero**: se activa si el resultado obtenido es 0.
- **Bit de transporte**: se activa si en el último bit que operamos en una operación aritmética se produce transporte; también puede deberse a una operación de desplazamiento.
- **Bit de desbordamiento**: se activa si la última operación ha producido un resultado que no se puede representar en el formato que estamos utilizando.
- **Bit de signo**: se activa si el resultado obtenido es negativo.
- **Bit de interrupción**: indica si las interrupciones están habilitadas o inhibidas.
- **Bit de modo de operación**: indica si la instrucción se ejecuta en modo supervisor o en modo usuario. Hay instrucciones que solo se ejecutan en modo supervisor.
- **Nivel de ejecución**: indica el nivel de privilegio de un programa en ejecución. Un programa puede desalojar el programa que se ejecuta actualmente si su nivel de privilegio es superior.

Los **registros de control** son los que dependen más de la organización del procesador. En estos registros se almacena la información generada por la unidad de control y también información específica para el sistema operativo. La información almacenada en estos registros no es nunca visible para el programador de aplicaciones.

4. Unidad aritmética y lógica

La unidad aritmética y lógica o ALU[2]es un circuito combinacional capaz de realizar operaciones aritméticas y lógicas con números enteros y también con números reales. Las operaciones que puede efectuar vienen definidas por el conjunto de instrucciones aritméticas y lógicas de las que dispone el juego de instrucciones del computador.

(2)*ALU* es la abreviatura de *arithmetic logic unit*.

Veamos primero cómo se representan los valores de los números enteros y reales con los que puede trabajar la ALU y, a continuación, cuáles son las operaciones que puede hacer.

1) Números enteros. Los números enteros se pueden representar utilizando diferentes notaciones, entre las cuales hay signo magnitud, complemento a 1 y complemento a 2. La notación más habitual de los computadores actuales es el complemento a 2 (Ca2).

Representación de la información

En este módulo no analizaremos en detalle la representación de números ni la manera de operar con ellos. Este tema lo hemos tratado con detenimiento en el módulo "Representación de la información" de la asignatura *Fundamentos de computadores*.

Todas las notaciones representan los números enteros en binario. Según la capacidad de representación de cada computador, se utilizan más o menos bits. El número de bits más habitual en los computadores actuales es de 32 y 64.

2) Números reales. Los números reales se pueden representar básicamente de dos maneras diferentes: en punto fijo y en punto flotante. El computador es capaz de trabajar con números reales representados en cualquiera de las dos maneras.

En la **notación en punto fijo** la posición de la coma binaria es fija y se utiliza un número concreto de bits tanto para la parte entera como para la parte decimal.

En la **notación en punto flotante** se representan utilizando tres campos, esto es: signo, mantisa y exponente, donde el valor del número es $\pm \text{mantisa} \cdot 2^{\text{exponente}}$.

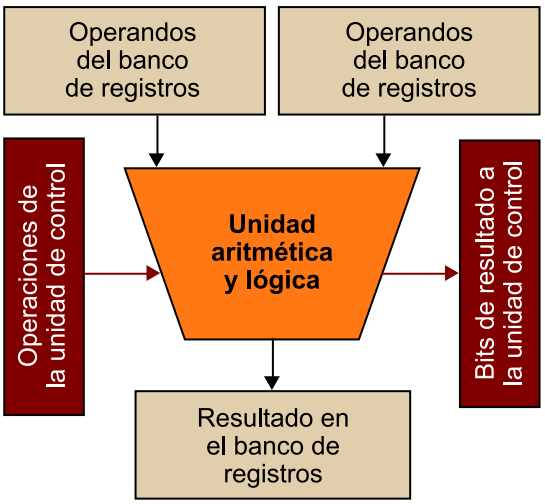
El IEEE ha definido una norma para representar números reales en punto flotante: IEEE-754. La norma define diferentes formatos de representación de números binarios en punto flotante. Los más habituales son los siguientes:

- **Precisión simple**: números binarios en punto flotante de 32 bits, utilizan un bit de signo, 8 bits para el exponente y 23 para la mantisa.
- **Doble precisión**: números binarios en punto flotante de 64 bits, utilizan un bit de signo, 11 bits para el exponente y 52 para la mantisa.
- **Cuádruple precisión**: números binarios en punto flotante de 128 bits, utilizan un bit de signo, 15 bits para el exponente y 112 para la mantisa.

La norma define también la representación del cero y de valores especiales, como infinito y NaN (*not a number*).

Las operaciones aritméticas habituales que puede hacer una ALU incluyen suma, resta, multiplicación y división. Además, se pueden incluir operaciones específicas de incremento positivo (+1) o negativo (–1).

Dentro de las operaciones lógicas se incluyen operaciones AND, OR, NOT, XOR, operaciones de desplazamiento de bits a la izquierda y a la derecha y operaciones de rotación de bits.



En los primeros computadores se implementaba la ALU como una única unidad funcional capaz de hacer las operaciones descritas anteriormente sobre números enteros. Esta unidad tenía acceso a los registros donde se almacenaban los operandos y los resultados de cada operación.

Para hacer operaciones en punto flotante, se utilizaba una unidad específica denominada *unidad de punto flotante*[3] o *coprocesador matemático*, que disponía de sus propios registros y estaba separada del procesador.

[3] En inglés, *floating point unit* (FPU).

La evolución de los procesadores que pueden ejecutar las instrucciones encabalgadamente ha llevado a que el diseño de la ALU sea más complejo, de manera que ha hecho necesario replicar las unidades de trabajo con enteros para permitir ejecutar varias operaciones aritméticas en paralelo y, por otra parte, las unidades de punto flotante continúan implementando en unidades separadas pero ahora dentro del procesador.

5. Unidad de control

La unidad de control se puede considerar el cerebro del computador. Como el cerebro, está conectada al resto de los componentes del computador mediante las señales de control (el sistema nervioso del computador). Con este símil no se pretende humanizar los computadores, sino ilustrar que la unidad de control es imprescindible para coordinar los diferentes elementos que tiene el computador y hacer un buen uso de ellos.

Es muy importante que un computador tenga unidades funcionales muy eficientes y rápidas, pero si no se coordinan y no se controlan correctamente, es imposible aprovechar todas las potencialidades que se habían previsto en el diseño.

Consiguientemente, muchas veces, al implementar una unidad de control, se hacen evidentes las relaciones que hay entre las diferentes unidades del computador y nos damos cuenta de que hay que rediseñarlas, no para mejorar el funcionamiento concreto de cada unidad, sino para mejorar el funcionamiento global del computador.

La función básica de la unidad de control es la ejecución de las instrucciones, pero su complejidad del diseño no se debe a la complejidad de estas tareas (que en general son muy sencillas), sino a la sincronización que se debe hacer de ellas.

Aparte de ver las maneras más habituales de implementar una unidad de control, analizaremos el comportamiento dinámico, que es clave en la eficiencia y la rapidez de un computador.

5.1. Microoperaciones

Como ya sabemos, ejecutar un programa consiste en ejecutar una secuencia de instrucciones, y cada instrucción se lleva a cabo mediante un ciclo de ejecución que consta de las fases principales siguientes:

1) Lectura de la instrucción.
2) Lectura de los operandos fuente.
3) Ejecución de la instrucción y almacenamiento del operando de destino.
4) Comprobación de interrupciones.

Cada una de las operaciones que hacemos durante la ejecución de una instrucción la denominamos *microoperación*, y estas microoperaciones son la base para diseñar la unidad de control.

5.1.1. Tipos de microoperaciones

La función básica de las microoperaciones es la transferencia de información de un lugar del computador a otro, generalmente de un registro a otro, tanto si son internos al procesador como externos. Este proceso de transferencia puede implicar solo mover la información pero también transformarla. Identificamos tres tipos básicos de microoperaciones:

1) Transferencia interna: operaciones de transferencia entre registros internos del procesador.

2) Transferencia interna con transformación: operaciones aritméticas o lógicas utilizando registros internos del procesador.

3) Transferencia externa: operaciones de transferencia entre registros internos del procesador y registros externos al procesador o módulos externos al procesador (como el bus del sistema o la memoria principal).

Ejemplos de transferencia

Una transferencia interna puede consistir en cargar el contenido del registro PC en el registro MAR para obtener la siguiente instrucción que hemos de ejecutar; una transferencia interna con transformación de información puede consistir en incrementar un registro, llevando el contenido del registro a la ALU y recoger el resultado para guardarlo en otro registro, y una transferencia externa puede consistir en llevar el contenido de un registro de estado de un dispositivo de E/S a un registro del procesador.

La nomenclatura que utilizaremos para denotar las microoperaciones es la siguiente:

Registro de destino ← Registro de origen
Registro de destino ← Registro de origen <operación> Registro de origen / Valor

Nota

Los registros, tanto de origen como de destino, pueden ser también de módulos externos al procesador.

5.1.2. Ciclo de ejecución

Las microoperaciones sirven de guía para diseñar la unidad de control, pero antes de entrar en el detalle de la implementación, analizaremos la secuencia de microoperaciones que habitualmente se producen en cada fase del ciclo de ejecución de las instrucciones.

Esta secuencia puede variar de una arquitectura a otra e, incluso, puede haber microoperaciones que estén en fases diferentes. Eso depende en buena parte de las características de la arquitectura: el número de buses, a qué buses tienen acceso los diferentes registros, si hay unidades funcionales específicas como registros que se puedan autoincrementar sin hacer uso de la ALU, la manera de acceder a los elementos externos al procesador, etc.

A continuación veremos las microoperaciones que se llevan a cabo en cada una de las fases del ciclo de ejecución para una arquitectura genérica desde el punto de vista funcional: cuáles se deben realizar y en qué orden. En el próximo apartado analizaremos con más detalle la dependencia temporal entre las microoperaciones en razón de los recursos que ha utilizado cada una.

Lectura de la instrucción

La fase de lectura de la instrucción consta básicamente de cuatro pasos:

1) MAR ← PC: se pone el contenido del registro PC en el registro MAR.

2) MBR ← Memoria: se lee la instrucción.

3) PC ← PC + Δ : se incrementa el PC tantas posiciones de memoria como se han leído (Δ posiciones).

4) IR ← MBR: se carga la instrucción en el registro IR.

Hay que tener presente que si la instrucción tiene un tamaño superior a una palabra de memoria, este proceso se debe repetir tantas veces como sea necesario.

Las diferencias principales que encontramos entre diferentes máquinas en esta fase son cómo y cuándo se incrementa el PC, ya que en algunas máquinas se utiliza la ALU y en otras se puede utilizar un circuito incrementador específico para el PC.

La información almacenada en el registro IR se descodifica para identificar las diferentes partes de la instrucción y determinar las operaciones necesarias que hay que efectuar en las fases siguientes.

Lectura de los operandos fuente

El número de pasos que hay que hacer en esta fase depende del número de operandos fuente y de los modos de direccionamiento utilizados en cada operando. Si hay más de un operando, hay que repetir el proceso para cada uno de los operandos.

El modo de direccionamiento indica el lugar en el que está el dato:

- Si el dato está en la instrucción misma, no hay que hacer nada porque ya lo tenemos en la misma instrucción.
- Si el dato está en un registro, no hay que hacer nada porque ya lo tenemos disponible en un registro dentro del procesador.
- Si el dato está en la memoria, hay que llevarlo al registro MBR.

Ejemplo

Veamos ahora algún ejemplo de ello:

- Inmediato: el dato está en la misma instrucción y, por lo tanto, no hay que hacer nada: IR (operando).
- Directo a registro: el dato está en un registro y, por lo tanto, no hay que hacer nada.
- Relativo a registro índice:
  - **MAR ← IR(Dirección operando) + Contenido IR(registro índice)**
  - **MBR ← Memoria**: leemos el dato.

IR(campo)

Campo, en IR(campo), es uno de los campos de la instrucción que acabamos de leer y que tenemos guardado en el registro IR.

La mayor parte de los juegos de instrucciones no permiten especificar dos operandos fuente con acceso a la memoria, ya que el dato obtenido se deja en el MBR y, por lo tanto, si se especificaran dos operandos de memoria, se debería guardar el primero temporalmente en otro registro, lo que causaría un retraso considerable en la ejecución de la instrucción.

En arquitecturas con un único bus interno (o de más de un bus, pero con determinadas configuraciones de acceso a los buses) también hay que añadir microoperaciones para guardar temporalmente información en registros cuando se trabaja con más de un dato al mismo tiempo, como por ejemplo cuando se hace una suma.

Ejecución de la instrucción y almacenamiento del operando de destino

El número de pasos que hay que realizar en esta fase depende del código de operación de la instrucción y del modo de direccionamiento utilizado para especificar el operando de destino. Se necesita, por lo tanto, una descodificación para obtener esta información.

Para ejecutar algunas instrucciones es necesaria la ALU. Para operar con esta, hay que tener disponibles al mismo tiempo todos los operandos que utiliza, pero la ALU no dispone de elementos para almacenarlos; por lo tanto, se deben almacenar en registros del procesador. Si no hay un bus diferente desde el que se pueda captar cada uno de los operandos fuente y donde se pueda dejar el operando de destino, se necesitan registros temporales (transparentes al programador) conectados directamente a la ALU (entrada y salida de la ALU) y disponibles al mismo tiempo, lo que implica el uso de microoperaciones adicionales para llevar los operandos a estos registros temporales.

Si los operandos fuente se encuentran en registros disponibles para la ALU, la microoperación para hacer la fase de ejecución es de la manera siguiente:

Registro de destino ← Registro de origen <operación> Registro de origen o Valor

Si el operando de destino no es un registro, hay que resolver antes el modo de direccionamiento para almacenar el dato, de manera muy parecida a la lectura del operando fuente.

Veamos cómo se resolverían algunos de estos modos de direccionamiento.

Directo a memoria:

- MBR ← <Resultado ejecución>
- MAR ← IR(Dirección operando)
- Memoria ← MBR

Relativo a registro base:

- MBR ← <Resultado ejecución>
- MAR ← Contenido IR(RB) + IR(Desplazamiento operando)
- Memoria ← MBR

Hay que tener presente que en muchas arquitecturas el operando de destino es el mismo que uno de los operandos fuente y, por lo tanto, los cálculos consecuencia del modo de direccionamiento ya están resueltos, es decir, ya se sabe dónde se guarda el dato y no hay que repetirlo.

Comprobación de interrupciones

En esta fase, si no se ha producido ninguna petición de interrupción, no hay que ejecutar ninguna microoperación y se continúa con la ejecución de la instrucción siguiente; en el caso contrario, hay que hacer un cambio de contexto. Para hacer un cambio de contexto hay que guardar el estado del procesador (generalmente en la pila del sistema) y poner en el PC la dirección de la rutina que da servicio a esta interrupción. Este proceso puede variar mucho de una máquina a otra. Aquí solo presentamos la secuencia de microoperaciones para actualizar el PC.

- **MBR** ← **PC**: se pone el contenido del PC en el registro MBR.
- **MAR** ← **Dirección de salvaguarda**: se indica dónde se guarda el PC.
- **Memoria** ← **MBR**: se guarda el PC en la memoria.

- **PC ← Dirección de la rutina**: se posiciona el PC al inicio de la rutina de servicio de la interrupción.

Ved también

Este punto lo veremos con más detalle en el módulo "Sistema de entrada/salida" de esta misma asignatura.

5.2. Señales de control y temporización

Hemos visto que cada microoperación hace una tarea determinada dentro del ciclo de ejecución de una instrucción. A pesar de la simplicidad de estas microoperaciones, llevarlas a cabo implica la activación de un conjunto de señales de control.

De manera general, entendemos una **señal de control** como una línea física que sale de la unidad de control y va hacia uno o más dispositivos del computador por los que circula una señal eléctrica que representa un valor lógico 0 o 1 y según cuáles sean los dispositivos a los que está conectado, es activo por flanco o por nivel.

Ved también

Los conceptos relacionados con las señales eléctricas los hemos trabajado en el módulo de circuitos lógicos de la asignatura *Fundamentos de computadores* y no los analizaremos en más detalle.

La mayor parte de los computadores tienen un funcionamiento síncrono, es decir, la secuencia de operaciones es gobernada por una señal de reloj. El período de esta señal de reloj (el tiempo que tarda en hacer una oscilación entera), llamado también *ciclo de reloj*, determina el tiempo mínimo necesario para hacer una operación elemental en el computador. Consideraremos que esta operación elemental es una microoperación.

Operación elemental y microoperación

La identificación "operación elemental – microoperación", en los computadores actuales, no siempre es tan fácilmente generalizable, en gran medida debido a los conceptos más avanzados de arquitectura del computador que no hemos tratado y que, por lo tanto, no tendremos en cuenta.

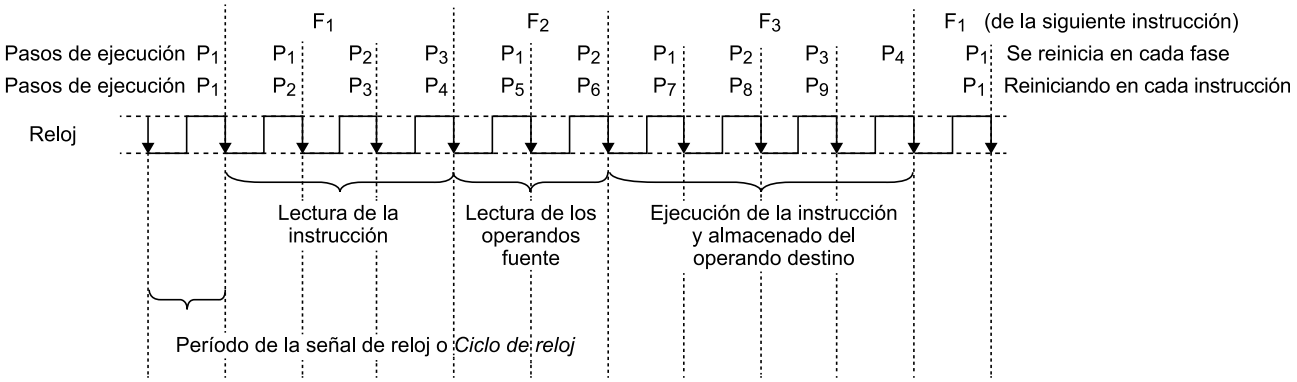
Para llevar a cabo el control de la ejecución de una instrucción, hay que realizar las cuatro fases siguientes:

- F_1: lectura de la instrucción.
- F_2: lectura de los operandos fuente.
- F_3: ejecución de la instrucción y almacenamiento del operando de destino.
- F_4: comprobación de interrupciones.

En cada una de estas fases, se efectúa un conjunto de microoperaciones y, por lo tanto, se requiere un ciclo o más de un ciclo de reloj para ejecutarlas (o ninguno si no se hace ninguna microoperación en aquella fase). Para controlar la ejecución de las microoperaciones en cada fase, se divide cada fase en una secuencia de pasos de ejecución.

Un paso de ejecución (P_i) es el conjunto de microoperaciones que se pueden ejecutar simultáneamente en un ciclo de reloj. El número de pasos de ejecución que se realizan en cada fase puede ser diferente.

El hecho de dividir el ciclo de ejecución en fases y pasos permite sistematizar el funcionamiento de la unidad de control y simplificar su diseño. Al hacerlo de esta manera, para controlar las fases y los pasos de ejecución, solo es necesario tener un contador para las fases que se reinicia al empezar cada instrucción y un contador de pasos que, en algunas máquinas, se reinicia al empezar cada instrucción y en otras se reinicia al comenzar cada fase.



Para optimizar el tiempo de ejecución de cada fase y, por lo tanto, de la instrucción, hay que intentar ejecutar simultáneamente dos o más microoperaciones en un mismo paso de ejecución. Hasta ahora, hemos visto, por una parte, que la ejecución de las microoperaciones implica la utilización de determinados recursos del computador y, por otra parte, que estas microoperaciones se ejecutan durante cierto tiempo, generalmente un ciclo de reloj. Para asegurarnos de que dos o más microoperaciones se pueden ejecutar al mismo tiempo, debemos tener en cuenta qué recursos se utilizan y durante cuántos ciclos de reloj.

CISCA

En el módulo "Arquitectura CISCA" explicamos con detalle la secuencia de microoperaciones de la ejecución de una instrucción concreta.

La agrupación de microoperaciones debe seguir básicamente dos reglas:

1) Debe seguir la secuencia correcta de acontecimientos. No se puede ejecutar una microoperación que genera u obtiene un dato al mismo tiempo que otra microoperación que lo ha de utilizar.

Por ejemplo, no se puede hacer una suma al mismo tiempo que se lee uno de los operandos de memoria, ya que no se puede asegurar que este operando esté disponible hasta el final del ciclo y, por lo tanto, no se puede garantizar que la suma se haga con el dato esperado.

2) Hay que evitar los conflictos. No se puede utilizar el mismo recurso para dos microoperaciones diferentes.

Por ejemplo, no se puede utilizar el mismo bus para transferir dos datos o direcciones diferentes al mismo tiempo.

5.3. Unidad de control cableada

Hasta ahora, hemos visto la unidad de control desde un punto de vista funcional: entradas, salidas, tareas que hace y cómo interacciona con el resto de los elementos del computador. A continuación nos centraremos en las técnicas de diseño e implementación de la unidad de control.

Estas técnicas se pueden clasificar en dos categorías:

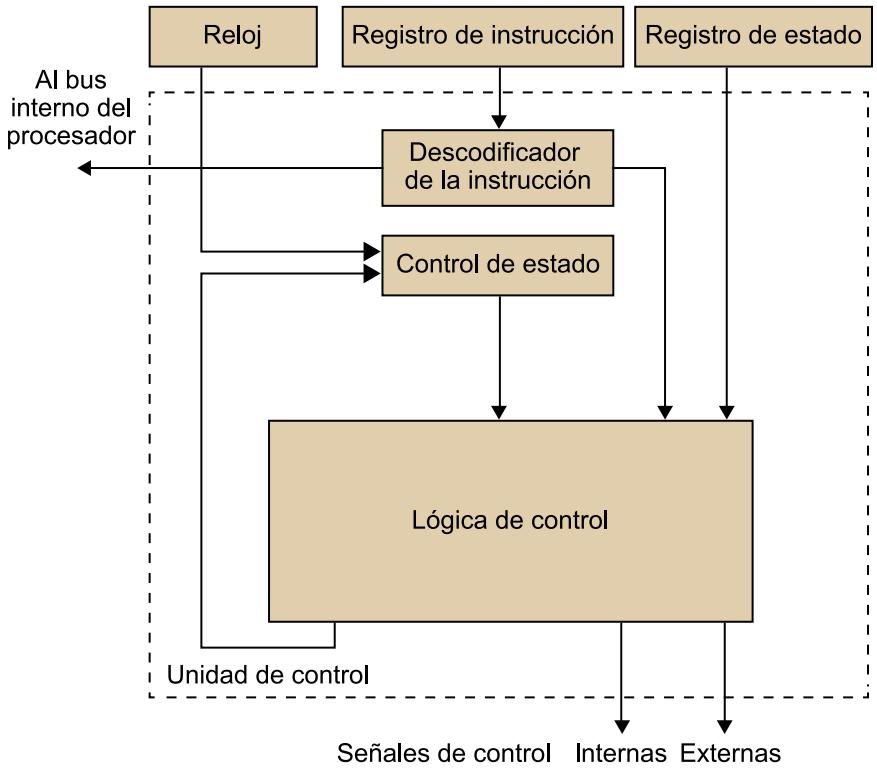
- Control cableado.
- Control microprogramado.

La unidad de control cableada es, básicamente, un circuito combinacional que recibe un conjunto de señales de entrada y lo transforma en un conjunto de señales de salida que son las señales de control. Esta técnica es la que se utiliza típicamente en máquinas RISC.

Para analizar con más detalle la unidad de control cableada, estudiaremos los elementos básicos que forman parte de ella.

5.3.1. Organización de la unidad de control cableada

En el diagrama siguiente se muestran los módulos que forman la unidad de control y la interconexión entre unos y otros.



A continuación, se describen detalladamente cada uno de los módulos que componen esta unidad de control:

1) Descodificador de la instrucción. La función básica de este módulo es descodificar la instrucción. Obtiene la información del registro IR y proporciona una salida independiente para cada instrucción de máquina, y también se encarga de suministrar los valores inmediatos (especificados en la instrucción misma) al bus interno del procesador para trabajar con estos valores.

2) Control de estado. La función de este módulo es contar los pasos dentro de cada fase del ciclo de ejecución de la instrucción. Recibe la señal de reloj (secuencia repetitiva de impulsos que se utiliza para delimitar la duración de las microoperaciones) y las señales de la lógica de control necesarias para controlar las fases del ciclo de ejecución, y genera una señal diferente para cada paso dentro de cada fase del ciclo de ejecución y las señales para identificar la fase dentro del ciclo de ejecución.

3) Lógica de control. Este módulo es un circuito combinacional que efectúa las operaciones lógicas necesarias para obtener las señales de control correspondientes a una microoperación. Recibe las señales de los otros dos módulos de la unidad de control cableada y del registro de estado y suministra información del resultado de esta microoperación al módulo que efectúa el control de estado.

Este circuito se puede implementar mediante dos niveles de puertas lógicas, pero a causa de diferentes limitaciones de los circuitos lógicos se hace con más niveles. Sin embargo, continúa siendo muy rápido. Hay que tener presente que las técnicas modernas de diseño de circuitos VLSI facilitan mucho el diseño de estos circuitos.

Ejemplo

Utilizando de referencia las señales de control de la arquitectura CISCA, y la secuencia de microoperaciones que tienen para ejecutar las instrucciones:

$$\text{MBRoutA} = F_1 \cdot P_3 + I_k \cdot F_3 \cdot P_1 + ...$$

Donde I_k son las instrucciones de transferencia del juego de instrucciones donde se transfiera un dato que se ha leído de memoria a un registro.

De esta expresión interpretamos que la señal de control MBRoutA es activa cuando estamos en el paso 3 de la fase 1 de cualquier instrucción (no especifica ninguna, ya que la fase de lectura de la instrucción es común a todas las instrucciones), o en el paso 1 de la fase 3 de la instrucción I_k, o en el paso..., y se continúa de la misma manera para el resto de los casos en los que es necesario activar la señal.

Como se puede ver, es un trabajo muy complejo y laborioso y el hecho de rediseñar el computador puede implicar cambiar muchas de estas expresiones.

Nota

No analizaremos con más detalle esta implementación cableada, ya que tendríamos que analizar más exhaustivamente la problemática del diseño de circuitos lógicos, lo que no es el objetivo de estos materiales. Encontraréis información acerca de la técnica de diseño en bibliografía específica sobre este tema.

5.4. Unidad de control microprogramada

La microprogramación es una metodología para diseñar la unidad de control propuesta por M. V. Wilkes en 1951, que pretende ser un método de diseño organizado y sistemático que evite las complejidades de las implementaciones cableadas.

Esta metodología se basa en la utilización de una memoria de control que almacena las microoperaciones necesarias para ejecutar cada una de las instrucciones y en el concepto de **microinstrucción** que veremos más adelante.

En los años cincuenta la propuesta pareció poco viable, ya que hacía necesario disponer de una memoria de control que fuera, al mismo tiempo, rápida y económica. Con la mejora de velocidad de los circuitos de memoria y su abaratamiento, devino una metodología muy utilizada, especialmente en el diseño de unidades de control complejas, como es el caso de las arquitecturas CISC.

Como ya sabemos, la ejecución de una instrucción implica realizar una serie de microoperaciones, y cada microoperación implica la activación de un conjunto de señales de control, de las cuales cada una puede ser representada por un bit.

La representación del conjunto de todas las señales de control (combinación de ceros y unos) da lugar a lo que denominamos **palabra de control**.

Palabra de control

Hay que tener presente que una palabra de control determina todas las señales de control que genera la unidad de control, ya que se utilizan tanto para indicar qué recursos se utilizan como para asegurar que el resto de los recursos no interfieren en la ejecución de las microoperaciones en curso.

Si dos o más microoperaciones se pueden ejecutar en un mismo período, se pueden activar las señales de control necesarias en una misma palabra de control.

5.4.1. Microinstrucciones

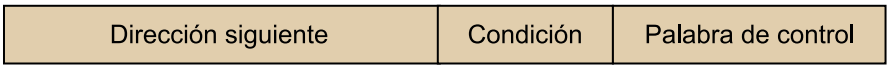
Llamamos **microinstrucción** a la notación utilizada para describir el conjunto de microoperaciones que se realizan simultáneamente en un mismo período y se representan con una palabra de control.

La ejecución de una microinstrucción implica activar las señales de control correspondientes a las microoperaciones que efectúa y determinar la dirección de la microinstrucción siguiente.

Cada microinstrucción se almacena en una memoria de control y, por lo tanto, cada una tiene asignada una dirección de memoria diferente.

Para especificar la secuencia de ejecución de las microinstrucciones, se necesita más información que la misma palabra de control. Esta información adicional da lugar a dos tipos básicos de microinstrucciones:

1) Microinstrucción horizontal. Se añade a la palabra de control un campo para expresar una condición y un campo para especificar la dirección de la microinstrucción siguiente que se tiene que ejecutar cuando se cumpla la condición.



Los bits de la palabra de control representan directamente las señales de control.

2) Microinstrucciones verticales. Para reducir el tamaño de las microinstrucciones horizontales, se codifican los bits de la palabra de control, de manera que se reduce el número de bits necesarios.

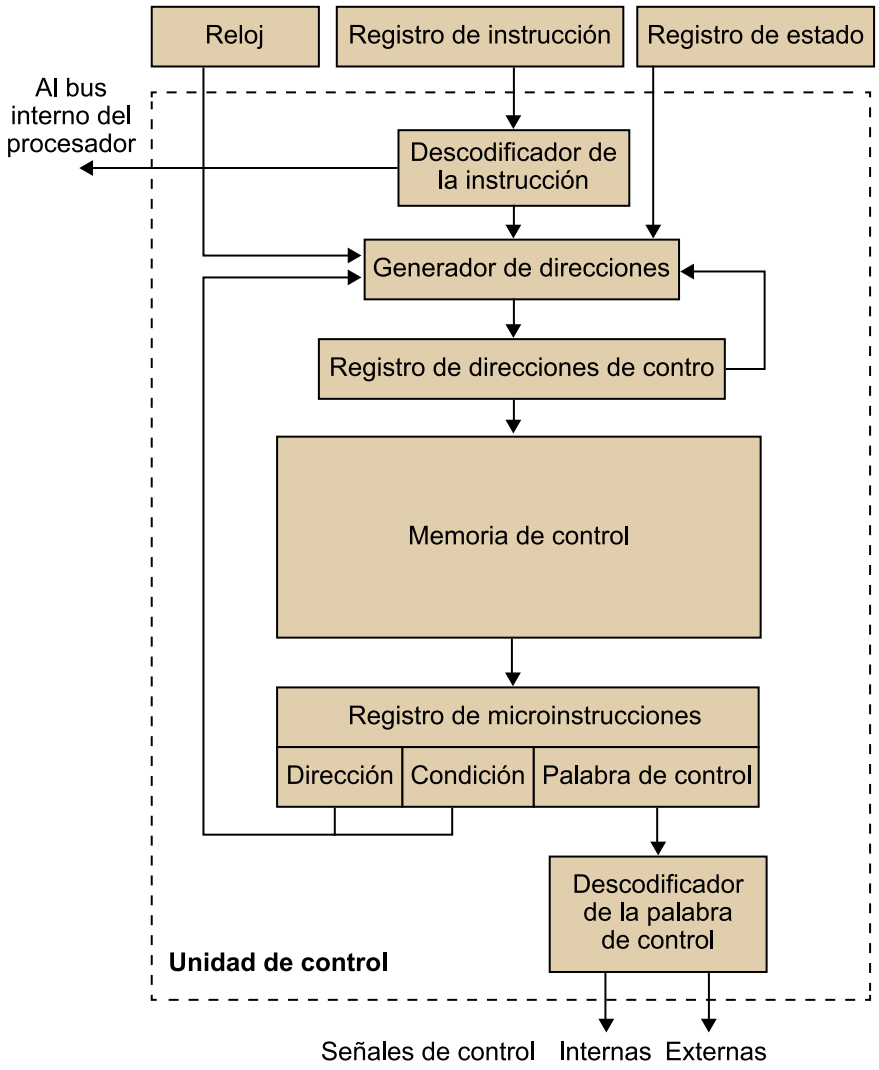
Si tenemos una palabra de control de 2^N bits se puede codificar utilizando N bits. Por ejemplo, una palabra de control de $2^4 = 16$ bits se puede codificar utilizando solo 4 bits.

El inconveniente de este tipo de microinstrucciones es que los bits de la palabra de control no representan directamente las señales de control, y hace falta un descodificador para traducir los códigos que aparecen en la microinstrucción de señales de control individuales.

El conjunto de microinstrucciones necesarias para ejecutar una instrucción da lugar a un **microprograma** (microcódigo o *firmware*). Como cada instrucción realiza tareas diferentes, es necesario un microprograma diferente para cada instrucción del juego de instrucciones de la arquitectura.

5.4.2. Organización de una unidad de control microprogramada

En el diagrama siguiente se muestran los módulos que forman la unidad de control y la interconexión entre unos y otros.



A continuación, se describe con detalle cada uno de los módulos que componen esta unidad de control:

1) Descodificador de la instrucción. Este módulo identifica los diferentes campos de la instrucción y envía la información necesaria al generador de direcciones para el control de los microprogramas (generalmente, la dirección de inicio del microprograma de la instrucción en curso) y suministra los valores inmediatos (especificados en la misma instrucción) al bus interno del procesador para que pueda trabajar con ellos.

2) Generador de direcciones. Este módulo genera la dirección de la microinstrucción siguiente y la carga en el registro de direcciones de control.

3) Registro de direcciones de control. En este registro se carga la dirección de memoria de la microinstrucción siguiente que se ejecutará.

4) Memoria de control. Almacena el conjunto de microinstrucciones necesarias para ejecutar cada instrucción o microprograma.

5) Registro de microinstrucciones. Este registro almacena la microinstrucción que se acaba de leer de la memoria de control.

6) Descodificador de la palabra de control. Este módulo, cuando se utilizan microinstrucciones verticales, traduce la palabra de control incluida en la microinstrucción a las señales de control individuales que representa.

5.4.3. Funcionamiento de la unidad de control microprogramada

Las dos tareas que hace la unidad de control microprogramada son la secuenciación de las microinstrucciones y la generación de las señales de control.

Secuenciación de las microinstrucciones

Una de las tareas más complejas que hace la unidad de control microprogramada es determinar la dirección de la microinstrucción siguiente que se ha de ejecutar, tarea que lleva a cabo el generador de direcciones.

La dirección de la microinstrucción siguiente la determinan estos factores:

- **El registro de instrucción**, que solo se utiliza para determinar el microprograma propio de cada instrucción.

- **El registro de estado**, que solo se utiliza cuando hay que romper la secuencia normal de ejecución de un programa, como sucede cuando se han de evaluar las condiciones en una instrucción de salto condicional.

- **El campo de condición de la microinstrucción**. Es el factor que se utiliza más frecuentemente. Este es el factor que hay que evaluar para saber si se debe continuar en secuencia o hacer una bifurcación:
  - Si la condición indicada es falsa, la microinstrucción siguiente que se ejecutará es la que sigue en secuencia y la dirección de esta microinstrucción puede estar especificada en la misma microinstrucción o se puede obtener incrementando el valor actual del registro de direcciones de control.

  - Si la condición indicada es cierta, la microinstrucción siguiente que se ejecutará es la que indica la dirección de bifurcación y esta dirección se obtiene de la misma microinstrucción.

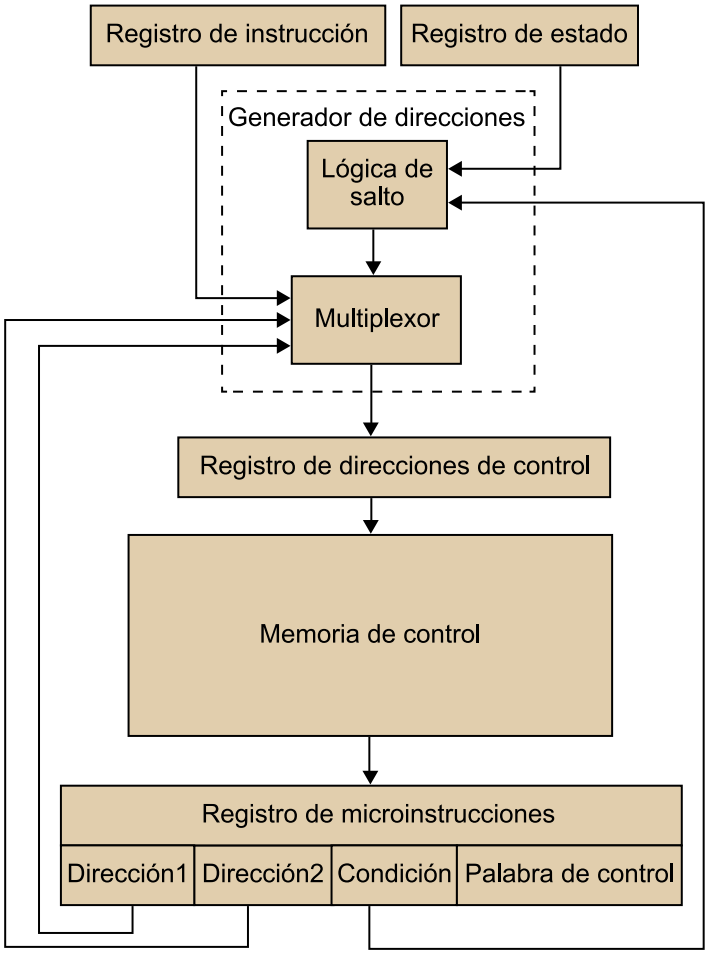
Formatos del campo de dirección y condición de la microinstrucción

El campo de dirección y condición de las microinstrucciones puede tener los formatos siguientes:

1) Dos campos explícitos. En la microinstrucción se especifican dos direcciones explícitas: la dirección de la microinstrucción que sigue en secuencia y una dirección de bifurcación. Según si se cumple o no la condición, se carga una dirección o la otra en el registro de direcciones de control.

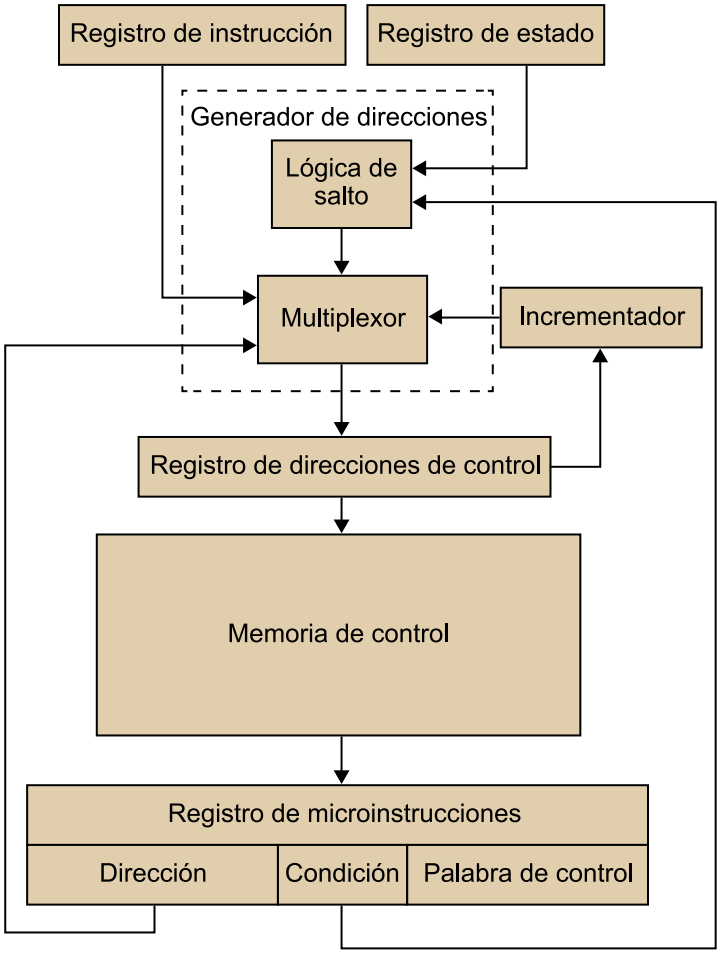
Con esta técnica no se necesita una lógica para incrementar la dirección actual y pasar a la microinstrucción siguiente que sigue en secuencia pero requiere más bits que otros formatos.

Ejemplo de circuito generador de direcciones con dos campos explícitos



2) Un campo explícito. En la microinstrucción se especifica explícitamente una dirección: cuando se cumple la condición indicada, se carga el registro de direcciones de control con la dirección explícita. Cuando no se cumple la condición, se carga el registro de direcciones de control con el valor incrementado del registro de direcciones de control.

Ejemplo de circuito generador de direcciones con un campo explícito



3) Microinstrucciones de formato variable. En este formato solo están el campo de condición y el campo de palabra de control. En las microinstrucciones de salto, se utiliza una parte del campo de la palabra de control como dirección de bifurcación, lo que obliga a utilizar un bit de la microinstrucción para indicar si es necesario interpretar esta parte del campo de la palabra de control como la dirección de la microinstrucción siguiente o como parte de la palabra de control.

La ventaja principal de este método es que nos permite reducir el tamaño de las microinstrucciones y no complica excesivamente la lógica de control de la unidad de control.

Generación de las señales de control

Generalmente las unidades de control microprogramadas no utilizan un formato totalmente horizontal de la palabra de control, sino un cierto nivel de codificación para ahorrar espacio en la memoria de control.

Lo más habitual es agrupar los bits de la palabra de control en campos; cada campo contiene un código que, una vez descodificado, representa un conjunto de señales de control.

Un campo con k bits puede contener 2^k códigos diferentes. Cada código simboliza un estado diferente de las señales de control que representa e indica, para cada una, la activación o la desactivación.

Consideramos que las señales de control son activas si el bit que las representa vale 1 e inactivas si este bit vale 0.

A la hora de elegir una codificación debemos tener presente cuáles son las diferentes combinaciones de señales de control que tienen que ser activas o no en un mismo instante.

La codificación puede ser *directa* (un solo nivel de codificación) o *indirecta* (dos niveles de codificación). En el segundo caso el valor descodificado que toma un campo sirve para saber qué señales de control representa un segundo campo, ya que pueden ser diferentes según el valor que tome el primer campo.

La agrupación de los bits de la palabra de control en campos puede responder a diferentes criterios; lo más habitual es agruparlos según los recursos que representan: se tratan todos los recursos de la máquina de manera independiente y se asigna un campo a cada uno: un campo para la ALU, un campo para la entrada/salida, un campo para la memoria, etc.

5.5. Comparación: unidad de control microprogramada y cableada

A continuación presentamos las ventajas e inconvenientes de la unidad de control microprogramada respecto a la unidad de control cableada.

Las ventajas son:

- Simplifica el diseño.
- Es más económica.
- Es más flexible:
  - Adaptable a la incorporación de nuevas instrucciones.
  - Adaptable a cambios de organización, tecnológicos, etc.

- Permite disponer de un juego de instrucciones con gran número de instrucciones. Solo hay que poseer una memoria de control de gran capacidad.
- Permite disponer de varios juegos de instrucciones en una misma máquina (emulación de máquinas).
- Permite ser compatible con máquinas anteriores de la misma familia.

- Permite construir máquinas con diferentes organizaciones pero con un mismo juego de instrucciones.

Los inconvenientes son:

- Es más lenta: implica acceder a una memoria e interpretar las microinstrucciones necesarias para ejecutar cada instrucción.
- Es necesario un entorno de desarrollo para los microprogramas.
- Es necesario un compilador de microprogramas.

Intel y AMD

Intel y AMD utilizan diseños diferentes para sus procesadores con organizaciones internas diferentes pero, gracias a la utilización de unidades de control microprogramadas, los dos trabajan con el mismo juego de instrucciones.

6. Computadores CISC y RISC

En el diseño del procesador hay que tener en cuenta diferentes principios y reglas, con vistas a obtener un procesador con las mejores prestaciones posibles.

Las dos alternativas principales de diseño de la arquitectura del procesador son las siguientes:

- Computadores CISC[4], cuyas características son:
  - El formato de instrucción es de longitud variable.
  - Dispone de un gran juego de instrucciones, habitualmente más de cien, para dar respuesta a la mayoría de las necesidades de los programadores.
  - Dispone de un número muy elevado de modos de direccionamiento.
  - Es una familia anterior a la de los procesadores RISC.
  - La unidad de control es microprogramada; es decir, la ejecución de instrucciones se realiza descomponiendo la instrucción en una secuencia de microinstrucciones muy simples.
  - Procesa instrucciones largas y de tamaño variable, lo que dificulta el procesamiento simultáneo de instrucciones.

(4) *CISC* es la abreviatura de *complex instruction set computer*; en español, computador con un juego de instrucciones complejo.

- Computadores RISC[5], que tienen las características siguientes:
  - El formato de instrucción es de tamaño fijo y corto, lo que permite un procesamiento más fácil y rápido.
  - El juego de instrucciones se reduce a instrucciones básicas y simples, con las que se deben implementar todas las operaciones complejas. Una instrucción de un procesador CISC se tiene que escribir como un conjunto de instrucciones RISC.
  - Dispone de un número muy reducido de modos de direccionamiento.
  - La arquitectura es de tipo *load-store* (carga y almacena) o registro-registro. Las únicas instrucciones que tienen acceso a memoria son LOAD y STORE, y el resto de las instrucciones utilizan registros como operandos.
  - Dispone de un amplio banco de registros de propósito general.
  - Casi todas las instrucciones se pueden ejecutar en pocos ciclos de reloj.
  - Este tipo de juego de instrucción facilita la segmentación del ciclo de ejecución, lo que permite la ejecución simultánea de instrucciones.
  - La unidad de control es cableada y microprogramada.

(5) *RISC* es la abreviatura de *reduced instruction set computer*; en español, computador con un juego de instrucciones reducido.

La segmentación permite que la ejecución de una instrucción empiece antes de acabar la de la anterior (se encabalgan las fases del ciclo de ejecución de varias instrucciones), gracias a que se reduce el tiempo de ejecución de las instrucciones.

Los procesadores actuales no son completamente CISC o RISC. Los nuevos diseños de una familia de procesadores con características típicamente CISC incorporan características RISC, de la misma manera que las familias con características típicamente RISC incorporan características CISC.

Ejemplo

PowerPC, que se puede considerar un procesador RISC, incorpora características CISC, tales como un juego de instrucciones muy amplio (más de doscientas instrucciones).

Los últimos procesadores de Intel (fabricante de procesadores típicamente CISC) también incorporan características RISC.

Resumen

La función principal del procesador es procesar los datos y transferirlos a los otros elementos del computador. Está formado por los elementos básicos siguientes:

- Conjunto de registros.
- Unidad aritmética y lógica.
- Unidad de control.

El ciclo de ejecución de la instrucción se divide en cuatro fases. En la tabla se pueden ver de manera esquemática las operaciones que se realizan en cada fase:

Inicio del ciclo de ejecución	
Fase 1: lectura de la instrucción	Leer la instrucción.
	Descodificar la instrucción.
	Actualizar el PC.
Fase 2: lectura de los operandos fuente	Calcular la dirección y leer el primer operando fuente.
	Calcular la dirección y leer el segundo operando fuente.
Fase 3: ejecución de la instrucción y almacenamiento del operando de destino	Ejecutar la instrucción.
Fase 4: comprobación de interrupciones	Comprobar si algún dispositivo ha solicitado una interrupción.

Los registros se han clasificado en los tipos siguientes:

- Registros de propósito general.
- Registros de instrucción.
- Registros de acceso a memoria.
- Registros de estado y de control.

La unidad aritmética y lógica hace operaciones aritméticas y lógicas con números enteros y números reales en punto fijo y en punto flotante.

Hemos estudiado con detalle la unidad de control y hemos visto los aspectos siguientes:

- Microoperaciones.
- Señales de control y temporización.

- Unidad de control cableada.
- Unidad de control microprogramada.
- Ventajas e inconvenientes de la unidad de control microprogramada con respecto a la implementación cableada.

Finalmente, hemos hablado de las características principales de los computadores CISC y RISC.