ARQUITECTURA DE COMPUTADORAS

CLASE 1 - REPASO

1.1) PILA

CONCEPTOS BÁSICOS

Una pila es un conjunto ordenado de elementos, en el que sólo uno de ellos es accesible en un instante dado, usada normalmente para la gestión de llamadas y retornos a/de procedimientos. El punto de acceso se denomina cabecera de pila. La longitud de la pila es variable, y sólo se pueden añadir o eliminar elementos en la cabecera de la pila. También se denomina lista último en entrar-primero en salir (LIFO). La pila crece desde direcciones más altas hacia más bajas, siendo una estructura ordenada de datos.

Esta cuenta con dos operaciones básicas, una operación PUSH (apilar) que añade un nuevo elemento en la cabecera de la pila, y una operación POP (desapilar) que elimina el elemento de la cabecera de la misma.

IMPLEMENTACIÓN DE LA PILA

La implementación de una pila requiere que exista un cierto conjunto de posiciones, utilizadas para almacenar los elementos de la pila, dichas direcciones están memorizadas en registros de la CPU:

- Puntero de pila: contiene la dirección del tope o cabecera de la pila (stack pointer). Si se añade, o se elimina, un elemento de la pila, el puntero se incrementa, o decrementa, para que contenga la dirección de la nueva cabecera de la pila.
- Base de la pila: contiene la dirección base del bloque reservado para la pila.
- Límite de la pila: contiene la dirección del otro extremo del bloque reservado.

1.2) PASAJE DE PARÁMETROS

A nivel de arquitectura de computadoras, los métodos de paso de argumentos se simplifican en dos:

- Parámetros por valor: se pasa el valor de una variable a un procedimiento. Son considerados parámetros de entrada e independientemente del uso de este valor por parte del procedimiento, éste no puede ser modificado.
- Parámetros por referencia: se pasa la dirección de memoria del argumento, y no su valor. Esto permite que los procedimientos realicen cambios al valor original de la variable.

1.3) SUBRUTINAS

CONCEPTOS BÁSICOS

Una subrutina es una sección de código, que recibe el control en un punto de entrada, y lo devuelve en un punto de salida.

El objetivo de la subrutina es realizar una tarea definida, para lo cual se le transfiere el control (procedimiento). Una vez finalizada la tarea lo devuelve al programa que la invocó en el punto donde fue invocada.

- Brinda economía de programa: el código puede ser usado varias veces.
- Modularización: se puede subdividir el programa en unidades pequeñas, más fácilmente verificables.
- Para realizar su trabajo, se requieren pasar parámetros entre el programa que invoca y la subrutina invocada.

Funcionamiento en pasos del llamado y retorno de una subrutina:

1. El programa invoca a una subrutina con la dirección CALL. Al invocarla le transfiere el control. Es decir que el PC (Program Counter) se carga con la dirección de comienzo de la subrutina, y la CPU comienza a ejecutar las instrucciones de la subrutina.
2. Cuando la subrutina complete su tarea, la última instrucción que ejecuta es la de RET. En ese momento se carga con la dirección de la instrucción siguiente al CALL en el programa principal. Para poder recuperar dicha dirección, debimos haberla guardado previamente en la pila.

DETALLES DE USO DE SUBRUTINAS

Existen tres métodos para pasar parámetros a subrutinas:

1. Vía registros
   - Los parámetros se pasan a través de los registros de la CPU.
   - Es un método sencillo, pero limitado por el número de registros disponibles.
   - Dado que se van a modificar los contenidos de los registros, es importante documentar los registros a usar.
2. Vía memoria
   - Los parámetros se transfieren a través de un área definida de memoria (RAM).
   - Difícil de estandarizar, debido a las dificultades en asignar un área de memoria.
3. Vía pila (stack)
   - Los datos se pasan a través de la pila.
   - La principal ventaja es que es independiente de la memoria y registros.
   - Los registros no tienen que ser modificados en las subrutinas.
   - Hay que comprender bien cómo funciona porque la pila es usada por el usuario (en la invocación y en la subrutina) y por el sistema (cuando salva la dirección de retorno en el CALL, o interrupciones).

ANIDAMIENTO DE SUBRUTINAS

Una subrutina puede invocar a otra subrutina. Esto se conoce como anidamiento de subrutinas. Cada subrutina tiene asociado su propio espacio de memoria en la pila. Depende del espacio de memoria de cada una, siendo casi infinita la cantidad de anidamientos. Cuando se termina una subrutina se devuelve el control a la subrutina que la invocó.

CLASE 2 – INTERRUPCIONES

2.1) INTERRUPCIONES

CONCEPTOS BÁSICOS

Una interrupción es un mecanismo que permite alterar el proceso de “ejecución normal” de la CPU. Este mecanismo permite que la CPU suspenda la tarea que está haciendo y responda a una solicitud de atención para ejecutar otra tarea. Una vez completado el servicio de la interrupción, el procesador retoma la tarea suspendida, en el punto donde se detuvo. Por lo tanto, se requieren 3 acciones:

1. Suspender la tarea que está ejecutando el procesador.
2. Saltar a otra tarea (al gestor de interrupción), asociada a la solicitud de interrupción (conocida como servicio de interrupción).
3. Restablecer la tarea suspendida en las condiciones en las que se encontraba en el momento en el que se la detuvo.

Así el programa de usuario no tiene que incluir ningún código especial para posibilitar las interrupciones; el procesador y el sistema operativo son los responsables de detener el programa de usuario y después permitir que prosiga desde el punto en que se lo detuvo.

La forma en que operan las interrupciones se puede ver de la siguiente manera:

1. La CPU recibe, mientras está ejecutando una tarea, un pedido de interrupción.
2. La CPU salva todo o parte del estado de la CPU correspondiente a la tarea a ser suspendida. Al menos salva el Contador de programa (PC) y el registro de estado (PSW), guardándolo en la pila.
3. La CPU busca, en un área de memoria definida, la dirección de comienzo (el "vector de interrupciones") del servicio de la interrupción, y comienza a ejecutarlo.
4. Una vez finalizado el servicio de interrupción, se retorna al punto donde el programa fue interrumpido mediante una instrucción de Retorno de interrupción.
5. La ejecución de la anterior instrucción permite desapilar exactamente lo apilado cuando atendió la interrupción. De esta manera retoma la tarea suspendida en el punto en que fue interrumpida.

CARACTERÍSTICAS

Existen 2 tipos de eventos:

- Internos: ocurren debido a la ocurrencia de una situación dentro del Sistema de cómputo.
  - Programa: generadas por alguna condición que se produce como resultado de la ejecución de una instrucción, tal como "overflow", división por cero, intento de ejecutar una instrucción inexistente.
  - Temporización: generadas por un temporizador interno al procesador. Esto permite al sistema operativo realizar ciertas funciones de manera regular.
  - Fallo de hardware: generadas por un fallo tal como la falta de potencia de alimentación o un error de paridad en la memoria.
- Externos: generadas por un controlador de E/S, para indicar la finalización sin problemas de una operación o para avisar de ciertas condiciones de error.

INTERRUPCIONES MÚLTIPLES

En general, los procesadores son capaces de manejar varias interrupciones de distintas características y orígenes. Dado el origen diverso de las mismas, hay algunas que son más importantes que otras.

Las interrupciones más importantes deben tener mayor "prioridad" que las menos importantes. Cuanto mayor sea su prioridad, mayor es la urgencia para ser atendida (pudiendo interrumpir a las de menor prioridad), incluso si hay una interrupción en curso.

- Procesamiento de interrupciones de igual prioridad: las interrupciones se atienden según el orden de llegada. Cuando llega una interrupción y es atendida, se inhabilita el resto de las interrupciones de igual o menor nivel de prioridad (si llega una nueva, quedará pendiente). El procesador ejecutará el servicio de la interrupción atendida y una vez finalizado el servicio, se habilitarán nuevamente las interrupciones.
- Procesamiento de interrupciones de distinta prioridad: una interrupción de prioridad más alta puede interrumpir en cualquier momento a una que es menor prioritaria. Cuando se ha gestionado la interrupción de prioridad más alta, el procesador vuelve a las interrupciones previas. Terminadas todas las rutinas de gestión de interrupciones se retoma el programa del usuario.

TIPOS DE INTERRUPCIONES

Según la prioridad que tienen:

- No enmascarables: son interrupciones que no pueden ser ignoradas, es decir, se atienden indefectiblemente, y están asociadas a eventos críticos, peligrosos o de alta prioridad.
- Enmascarables: pueden ser, eventualmente "ignoradas". Para ello el procesador permite realizar algunas acciones que pueden inhibir la atención de la interrupción. Las interrupciones enmascarables están asociadas a operación poco críticas.

Según como se invoquen:

- Por hardware: son generadas por señales físicas asociadas a eventos externos o internos. No hay una instrucción específica en el programa que invoque a dicho tipo de interrupción.
    - Externas: son conocidas como Interrupt request. El origen de estas señales de pedido de interrupción proviene típicamente de dispositivos conectados al subsistema de E/S. Son consideradas como las verdaderas interrupciones porque son aleatorias en relación al proceso en ejecución (asíncronas).
    - Internas: son conocidas como Traps o excepciones. Son señales creadas dentro del sistema de cómputo en respuesta a situaciones propias del proceso en ejecución y no vinculadas con operaciones de E/S, por esto no son estrictamente aleatorias.
- Por software: son conocidas como Software interrupt. Son instrucciones explícitas que tienen un efecto similar a una interrupción por hardware. Como normalmente el sistema operativo administra los servicios de las interrupciones, este tipo de interrupciones permite hacer llamados a funciones de dicho sistema. Dependiendo en el equipo que trabajemos, podemos hacer los llamados o no, en caso de no poder, el proceso de administrar una tarea por interrupción sería muy complicado.

GESTIÓN DE LAS INTERRUPCIONES

1. Detectar el pedido de interrupción: el procesador examina, en cada ciclo de instrucción la presencia de interrupciones. Para poder implementar la tarea de detección de interrupciones (además de las fases de captura y ejecución), en el ciclo de instrucción se agrega la fase de gestión de interrupciones.
   La etapa de gestión de interrupciones determina la presencia o ausencia de pedido de las mismas, la presencia de un pedido de interrupción se manifiesta mediante una o varias señales discretas conocidas como banderas que la CPU examina, que se encuentran en algún registro del entorno.
2. Detener la tarea que se estaba ejecutando: en caso de no haber pedido pendiente (Flag inactivo) se inicia el ciclo de captación de la siguiente instrucción y no se interrumpe la ejecución del programa. En caso contrario, el procesador responde suspendiendo la operación del programa que estaba ejecutando.
3. Salvar el estado de la tarea que se estaba ejecutando: si hay algún pedido de interrupción pendiente, el procesador guarda en la pila del sistema, el "estado del proceso". Puede o guardar la dirección de la próxima instrucción y algún registro importante, o guardar directamente todos los registros del procesador.
   El objetivo de esta operación es el de restablecer el estado del procesador al terminar el servicio de interrupción.
4. Obtener la dirección de comienzo del servicio de la interrupción y bifurcar a dicho servicio: se obtiene la dirección donde comienza la rutina de la interrupción y carga el PC con este valor, bifurcando al servicio de la interrupción.
   Se dispone de un área de memoria reservada, conocida como área de vectores de interrupciones, donde están todas estas direcciones.
5. Ejecutar el servicio de interrupción.
6. Retornar y restaurar el estado en que estaba la tarea interrumpida.
7. Continuar con la ejecución normal de la tarea interrumpida: cuando la rutina de gestión de interrupción se completa, el procesador puede proseguir la ejecución del programa en el punto en el que se interrumpió.

2.2) PIC

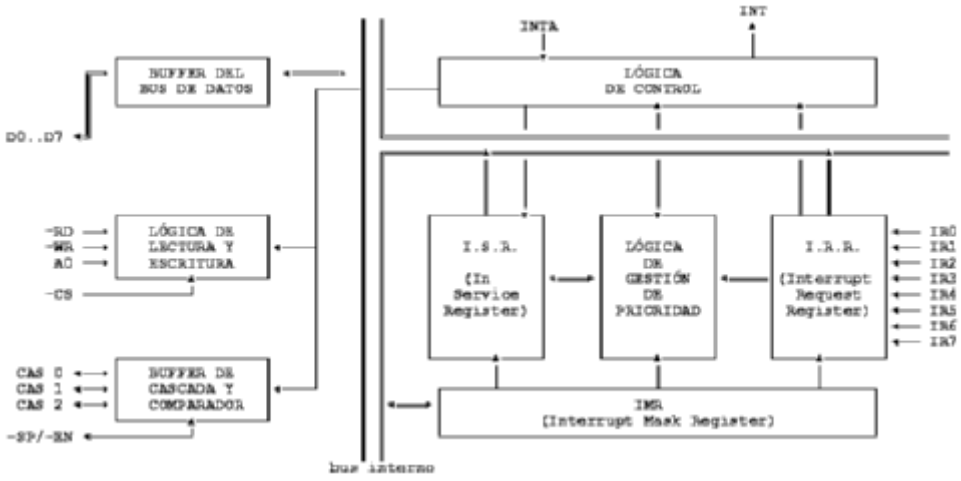
CONTROLADOR DE INTERRUPCIONES PIC

El PIC (Controlador de Interrupciones Programable) es un dispositivo interno con líneas de interrupción por hardware, donde se conectan varios dispositivos que pueden interrumpir a la CPU. De esta forma el PIC permite utilizar varios dispositivos, multiplexando los pedidos de todos ellos a la única línea de interrupción del procesador.

Cuando un dispositivo quiere interrumpir a través de la línea N, el PIC realiza las siguientes acciones:

1. Verifica que la línea esté habilitada (que no esté enmascarada).
2. Verifica que no haya otros dispositivos con mayor prioridad que también quieran interrumpir.
3. Si las dos verificaciones previas fueron exitosas, le solicita atención a la CPU con la única señal de pedido de interrupción IntR (Interrupt request) y cuando esta esté lista para atendarla, le avisa al PIC mediante la señal IntA (Interrupt acknowledge).
4. Al mismo tiempo el PIC genera en el bus de datos el número de la interrupción (vector) correspondiente a la línea N a ser atendida.
5. La CPU lee ese número y responde buscando en memoria el vector correspondiente al servicio de esa interrupción.

La estructura interna del PIC es así:



El PIC posee 3 registros principales:

- El ISR (Interrupción en servicio) que sus bits indican si se está atendiendo la interrupción de algún dispositivo.
- El IRR (Interrupciones pedidas) que sus bits indican qué dispositivos están solicitando una interrupción.
- El IMR (Máscara de interrupciones) que sus bits indican qué líneas están habilitadas. Se utiliza para habilitar o deshabilitar pedidos de interrupciones y se conoce como enmascaramiento de interrupciones.

CLASE 3 - ENTRADA-SALIDA

3.1) SUBSISTEMA E/S

INTRODUCCIÓN

Junto con el procesador y el conjunto de módulos de memoria, el tercer elemento clave de un computador es un conjunto de módulos de E/S. Debido a la gran variedad de periféricos, sus distintos funcionamientos y velocidades desiguales, el subsistema de E/S tiene que ser lo suficientemente flexible para permitir:

- La transmisión de diferentes cantidades de datos y contar con un amplio rango de velocidades de transmisión.
- Poder soportar los distintos formatos de dato y tamaños de palabras del computador a los que se conectan.

DISPOSITIVOS EXTERNOS

Los dispositivos externos conectados a un módulo de E/S son llamados periféricos y existen tres categorías:

- De interacción con humanos: permiten la comunicación con el usuario del computador. Ejemplo: impresoras.
- De interacción con máquinas: permiten la comunicación con elementos del equipo. Ejemplo: sistemas de cinta.
- De comunicación: permiten la comunicación con dispositivos remotos. Ejemplo: terminales.

3.2) PUERTO/MÓDULO DE E/S

Un módulo de E/S es un módulo administrado, por lo común, por el SO que tiene dos funciones principales:

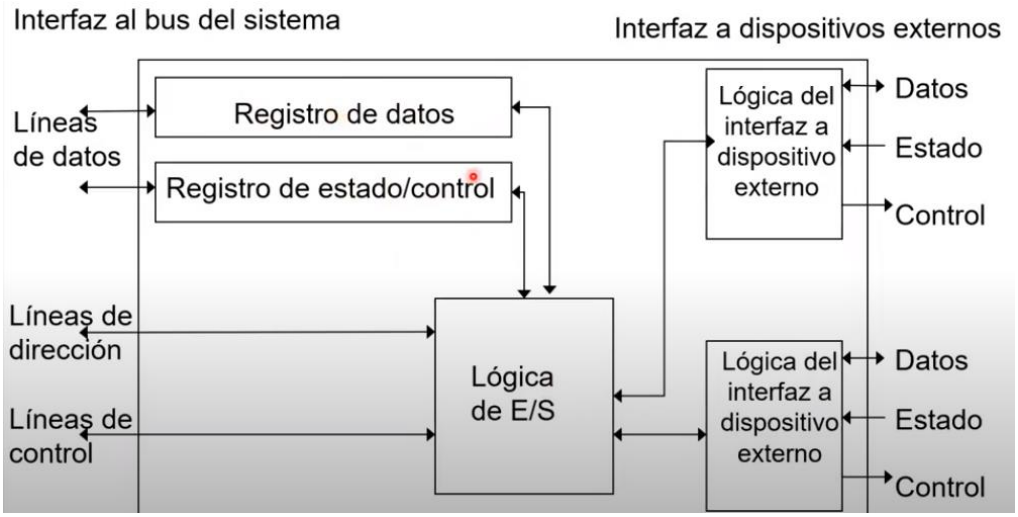
- Realizar la interfaz entre el procesador y la memoria a través del bus del sistema o un conmutador central.
- Realizar la interfaz entre uno o más dispositivos periféricos mediante enlace de datos específicos.

MÁS FUNCIONES

Las principales funciones y requisitos de un módulo de E/S se encuentran dentro de las siguientes categorías:

- Control y temporización: utilizado para coordinar el tráfico entre los recursos internos y los dispositivos externos. Por ejemplo, una transferencia de datos desde un dispositivo externo al procesador.
- Comunicación con el procesador que implica: la decodificación de órdenes (el módulo de E/S acepta órdenes del procesador y se envían generalmente utilizando líneas del bus de control), datos (el procesador y el módulo de E/S intercambian datos a través del bus de datos), información de estado (debido a la lentitud de los periféricos, es necesario conocer el estado del módulo de E/S) y reconocimiento de dirección (un módulo puede reconocer una única dirección para cada uno de los periféricos que la controla).
- Comunicación con los dispositivos: implica intercambiar órdenes, información del estado y datos.
- Almacenamiento temporal de datos: los datos provenientes de la memoria se envían al módulo de E/S en ráfagas rápidas. Los datos se almacenan temporalmente en el módulo de E/S, y después se envían al periférico a la velocidad de éste. En el sentido contrario, los datos se almacenan para no mantener a la memoria ocupada en una operación de transferencia lenta, por ello el módulo debe ser capaz de operar a distintas velocidades.
- Detección de errores: incluyen defectos mecánicos y eléctricos en el funcionamiento del dispositivo. También se mencionan cambios accidentales en los bits en la transmisión del dispositivo hacia el módulo.

ESTRUCTURA INTERNA



El módulo se conecta al resto del computador a través de un conjunto de líneas. Los datos que se transfieren a y desde el módulo se almacenan temporalmente en uno o más registros de datos. Además, puede haber uno o más registros de estado que proporcionan información del estado presente. Un registro de estado también puede funcionar como un registro de control, para recibir información de control del procesador. La lógica que hay en el módulo interactúa con el procesador a través de una serie de líneas de control. Estas son las que utiliza el procesador para proporcionar las órdenes al módulo de E/S. El módulo también debe ser capaz de reconocer y generar las direcciones asociadas a los dispositivos que controla, teniendo el módulo una dirección única o si controla a más de un dispositivo externo, un conjunto único de direcciones. Por último, el módulo de E/S posee la lógica específica para la interfaz con cada uno de los dispositivos que controla.

TIPOS DE PUERTO DE E/S

- Puerto paralelo: hay varias líneas de datos (n) que transfieren n bits simultáneamente entre el puerto de E/S y el periférico. Se requiere disponer de una conexión un cable que incluya al menos los n bits de datos.
- Puerto serie: hay una línea de dato para la transferencia entre el puerto de E/S y el periférico. Se requiere disponer de una conexión mediante un cable sencillo. Los datos deben serializarse.

TRANSFERENCIA DE INFORMACIÓN - REGISTROS DE E/S

Desde el punto de vista de la CPU, una operación de E/S requiere acceder a los registros internos del módulo de interfaz de E/S. Los registros pueden ser de lectura y/o escritura. Hay dos tipos de registros:

- De datos: interviene en la transferencia de entrada o de salida del dato a intercambiar en el sistema de cómputo y el periférico.
  - Operación de entrada: lectura de un registro de dato de entrada (registro escrito por el periférico y leído por la CPU).
  - Operación de salida: escritura de un registro de dato de salida (registro escrito por la CPU y leído por el periférico
- De control y estado: registros que controlan y registran el funcionamiento del módulo, la transferencia, y el periférico.
  - Control: adecuar la configuración del módulo para ajustar formatos, sincronizaciones, etc.
  - Estado: registrar el estado operativo del módulo y del periférico.

DIRECCIONAMIENTO - ACCESO AL SUBSISTEMA DE E/S

Existen dos técnicas de acceso a los registros anteriormente mencionados:

- Espacio de E/S compartido con memoria (memory-mapped).
  - En esta técnica los registros de los dispositivos de E/S y memoria comparten un único espacio de direcciones.
  - Los registros de E/S se comportan idéntico a una memoria de lectura/escritura.
  - No hay instrucciones específicas para E/S, se usan las mismas instrucciones de movimiento de datos a memoria. Permite una variedad de órdenes de acceso a memoria (programación eficiente).
- Espacio de E/S separada de la memoria (aislada).
  - En esta técnica los registros de los dispositivos de E/S y la memoria están en diferentes espacios de direcciones.
  - Dado que el bus de direcciones compartido por la memoria y el subsistema de E/S, se requieren señales de control adicionales para identificar a donde está accediendo la CPU.
  - Hay un conjunto limitado de instrucciones específicas de E/S, distintas de las instrucciones de acceso a la memoria. Cuando se ejecutan estas instrucciones específicas, en el bus de control se identifica el acceso al mapa de direcciones de E/S, para el resto de instrucciones se identifica el acceso a memoria.

GESTIÓN DE LA TRANSFERENCIA

Desde el punto de vista de la gestión para transferir datos entre el sistema de cómputo y el periférico, existen tres estrategias básicas de implementación:

- E/S programada y espera de respuesta.
  - La CPU interviene directamente en la transferencia de cada unidad de información con el módulo, es decir que tiene el control directo sobre la operación de E/S. Aparte:
    - Comprueba el estado del dispositivo.
    - Envía comandos de lectura/escritura.
    - Realiza la transferencia de todos los datos (de a uno).
  - El módulo de E/S realiza la acción solicitada por el procesador, y después activa los bits apropiados en el registro de estado de E/S.
  - El procesador es responsable de comprobar periódicamente el estado del módulo de E/S hasta que encuentra que la operación ha terminado (permanece ociosa).
  - La secuencia de acciones que ejecuta la CPU es:
    - 1. La CPU verifica el estado de periférico leyendo un registro del módulo de interfaz.
    - 2. Examina el estado del periférico chequeando el bit/bits que identifican dicho estado.
    - 3. Si el dispositivo no está listo, la CPU vuelve al paso 1. Este lazo significa que la CPU espera hasta que el periférico esté listo para la transferencia.
    - 4. Cuando el dispositivo está listo, la CPU transfiere un dato hacia o desde el módulo de interfaz.
    - 5. Si hay más datos que transferir vuelve al paso 1.
    - 6. Si se completo la transferencia, termina el servicio de E/S.
- E/S programada mediante interrupciones.
  - La CPU interviene directamente en la transferencia de cada unidad de información con el módulo.
  - Cada vez que el módulo esté listo (o haya completado una transferencia) avisa a la CPU con un pedido de interrupción.
  - Ahora el procesador solo inicia la transferencia al recibir el pedido de interrupción del periférico, y puede continuar realizando algún trabajo útil.
  - Procesamiento de la interrupción:
    - 1. El dispositivo de E/S envía una señal de interrupción al procesador.
    - 2. El procesador completa la ejecución de la instrucción en curso antes de responder a la interrupción.
    - 3. El procesador comprueba si hay interrupciones, determina que hay una, y envía una señal de reconocimiento al dispositivo que originó la interrupción.
    - 4. El procesador guarda en la pila la información necesaria para continuar el programa en curso en el punto en que se interrumpió (PSW y PC).
    - 5. Después, el procesador carga el nuevo PC en función de la interrupción. El control se transfiere al programa de gestión de interrupción, que realiza estas tareas:
    - 6. Guarda los contenidos de los registros del procesador, ya que podrían ser modificados.
    - 7. Procesa la interrupción.
    - 8. Cuando el procesamiento de la interrupción ha terminado, los valores de los registros almacenados se recuperan de la pila y se vuelven a almacenar en los registros.
    - 9. Se restaura el PC y el PSW, para continuar en donde se dejó el programa interrumpido.
  - Existen varias estrategias distintas para identificar la fuente de la interrupción:
    - Múltiples líneas de interrupción.
      - Se dispone de una línea de interrupción por cada dispositivo.
      - Es probable que a cada línea se le conecten varios módulos de E/S, debido a la cantidad restringida de señales de interrupción que puede manejar la CPU.

        - Consulta software (software polling).
            - Se dispone de una sola línea de interrupción para los dispositivos.
            - Cuando ocurre el pedido de interrupción, la CPU tiene que consultar a cada dispositivo para determinar quien fue el demandante (sumamente lento).
        - Conexión en cadena (Daisy chain).
            - Todos los módulos de E/S comparten una línea común para solicitar interrupciones.
            - La línea de reconocimiento de interrupción se conecta encadenando los módulos uno tras otro.
            - Una vez enviada la confirmación de parte de la CPU, el módulo que está mas adelante en la conexión encadenada responderá colocando un vector, en el bus, que lo identifica, utilizándolo como un puntero a la rutina de servicio apropiada.
        - Arbitraje de bus (vectorizada).
            - Se dispone de una sola línea de interrupción INTR para todos los dispositivos.
            - Un controlador dedicado (PIC) provee el vector que identifica la fuente de interrupción.
            - Las líneas de interrupción un orden de prioridad.
    - Múltiples interrupciones (prioridad):
        - Con varias líneas de interrupción, el procesador selecciona la línea con más prioridad.
        - Con la consulta software, el orden en el que se consultan los módulos determina su prioridad.
        - El orden en de los módulos en la conexión en cadena determina su prioridad.
        - El arbitraje de bus puede emplear un esquema de prioridad.

- Acceso directo a memoria (DMA).
    - Es una técnica de transferencia de datos entere periférico y memoria sin intervención directa del CPU.
    - Es llevada a cabo por un módulo de DMA (DMAC), encargado de llevar a cabo la transferencia.
    - El DMAC es capaz de imitar al procesador y puede recibir el control del sistema.
    - Las principales fases de transferencia son:
        - 1. Fase de inicialización: la CPU debe configurar el módulo de E/S y el DMAC con los parámetros de transferencia.
            - Inicialización interfaz de E/S:
                - Tipo de transferencia a través de líneas de control (lectura/escritura).
                - Configuración del periférico.
                - Información de control para el periférico.
            - Inicialización DMAC:

                - Número de palabras a leer o escribir, indicado a través de las líneas de datos y almacenado en el registro de cuenta de datos.
                - Tipo de transferencia (lectura/escritura).
                - Dirección de memoria inicial para la transferencia, indicada a través de las líneas de datos y almacenada en el CDMA en su registro de direcciones.
                - Información adicional para la transferencia.
        - 2. Fase de ejecución de la transferencia.
            - Cuando el periférico está listo, pide al DMAC iniciar la transferencia mediante una señal física. Cuando el DMAC recibe este pedido, pide el control del bus mediante alguna señal especial a la CPU.
            - Cuando reconoce el pedido de DMA, la CPU entrega el bus y lo deja de controlar. La CPU avisa al DMAC que liberó el bus mediante otra señal especial.
            - Una vez liberado el bus por parte del CPU, el DMAC lo toma bajo su control y ejecuta la transferencia hasta terminarla.

- El DMAC avisa al periférico que puede iniciar la transferencia y este empieza a transferir los datos, a través del bus, con la memoria, de a uno por vez.
- Por cada palabra transferida, los registros del DMAC se van actualizando.

▪ 3. Finalización de la transferencia.

- Cuando la transferencia se ha terminado, el módulo de DMA libera el bus y envía una señal de interrupción al procesador para avisar que finalizó el proceso.
- Es así como la CPU retoma el control del bus, y gracias a la interrupción verifica el resultado de la transferencia vía los registros internos del DMAC.

○ Técnicas de transferencia por DMA:

▪ Por ráfaga: el DMAC solicita el control del bus a la CPU. Cuando la CPU concede el bus, el DMAC no lo libera hasta haber finalizado la transferencia de todo el bloque de datos completo.

▪ Por robo de ciclo: el CDMA toma el control del bus, transfiere el dato y luego devuelve el control del bus. Luego la CPU ejecuta una instrucción y si el CDMA necesita hacer otra transferencia vuelve a pedir el uso del bus y transfiere el dato, repitiendo esto hasta que finalice la transferencia del bloque completo. No se debe guardar el contexto.

○ La principal ventaja es la eficiencia, dado que la CPU se libera de tener que controlar la transferencia de los datos. Solo prepara la transmisión, y verifica el resultado de la misma.

○ La principal desventaja se origina en el uso del bus. Como las transferencias por DMA pueden tener mayor prioridad que la CPU, se puede degradar el rendimiento de la CPU si el DMAC hace uso intensivo del bus.

3.3) CANALES DE E/S

CARACTERÍSTICAS

Los canales de E/S representan una extensión al concepto de DMA y tienen la habilidad de ejecutar programas de servicios de E/S, lo que les permite tener un completo control de la transferencia de datos. La CPU no ejecuta las instrucciones de E/S, si no que las realiza el procesador incluido en el canal. El programa que ejecuta el procesador interno del canal está almacenado en la memoria principal. La CPU solo indica al canal de E/S que debe ejecutar un programa de la memoria, y el programa especifica el dispositivo o dispositivos, área o áreas de memoria para almacenamiento, entre otras cosas, y el canal de E/S sigue estas instrucciones y controla la transferencia de datos.

TIPOS DE CANALES

Hay dos tipos básicos de canales de E/S.

- Canal selector.
  - Controla varios dispositivos de velocidad elevada, y en un instante dado, se dedica a transferir datos de a uno de esos dispositivos.
  - Es decir, que el canal de E/S selecciona un dispositivo y efectúa la transferencia de datos.
  - Cada dispositivo es manejado por un controlador, o módulo de E/S.
  - Por lo tanto, el canal de E/S ocupa el lugar de la CPU en el control del módulo de E/S.
  - Solo puede atender un dispositivo a la vez.
- Canal multiplexor.
  - Puede manejar las E/S de varios dispositivos al mismo tiempo.
  - Para dispositivos de velocidad reducida, un multiplexor de byte acepta o transmite caracteres tan rápido como es posible a varios dispositivos.
  - Para dispositivos de velocidad alta, un multiplexor de bloque entrelaza bloques de datos de los distintos dispositivos.

3.4) INTERCONEXIÓN CON BUSES

¿QUÉ ES UN BUS?

Un bus es un camino de comunicación entre dos o más dispositivos. Se trata de un medio de transmisión compartido. Al bus se conectan varios dispositivos, y cualquier señal transmitida por uno de esos dispositivos está disponible para que los otros dispositivos conectados al bus puedan acceder a ella.

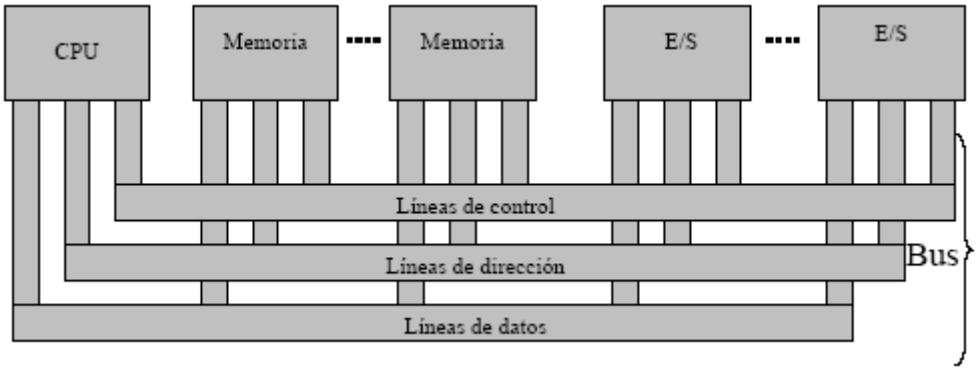
Un bus está constituido por varios caminos de comunicación, o líneas. Cada línea es capaz de transmitir señales binarias representadas por 1 y por 0. En un intervalo de tiempo, se puede transmitir una secuencia de dígitos binarios a través de una única línea.

Los computadores poseen distintos tipos de buses que proporcionan comunicación entre sus componentes a distintos niveles dentro de la jerarquía del sistema. El bus que conecta los componentes principales del computador se denominan bus del sistema.

ESTRUCTURA DEL BUS

El bus del sistema está constituido usualmente por una amplia cantidad de líneas. A cada línea se le asigna un significado o una función particular. Las líneas pueden ser de datos, de direcciones y de control.

- Las líneas de datos proporcionan un camino para transmitir datos entre los módulos del sistema. El conjunto constituido por estas líneas se denomina bus de datos. El bus de datos tiene un ancho y el número de líneas determina cuantos bits se pueden transferir al mismo tiempo en dicho bus.
- Las líneas de dirección se utilizan para designar la fuente o el destino del dato situado en el bus de datos. La anchura del bus de direcciones determina la máxima capacidad de memoria posible en el sistema. Además, las líneas de direcciones son usadas para direccionar los puertos de E/S.
- Las líneas de control se utilizan para controlar el acceso y el uso de las líneas de datos y de direcciones. Las señales de control transmiten tanto órdenes (operación a realizar) como información de temporización (indican la validez de los datos y las direcciones) entre los módulos del sistema.



ELEMENTOS DE DISEÑO DE UN BUS

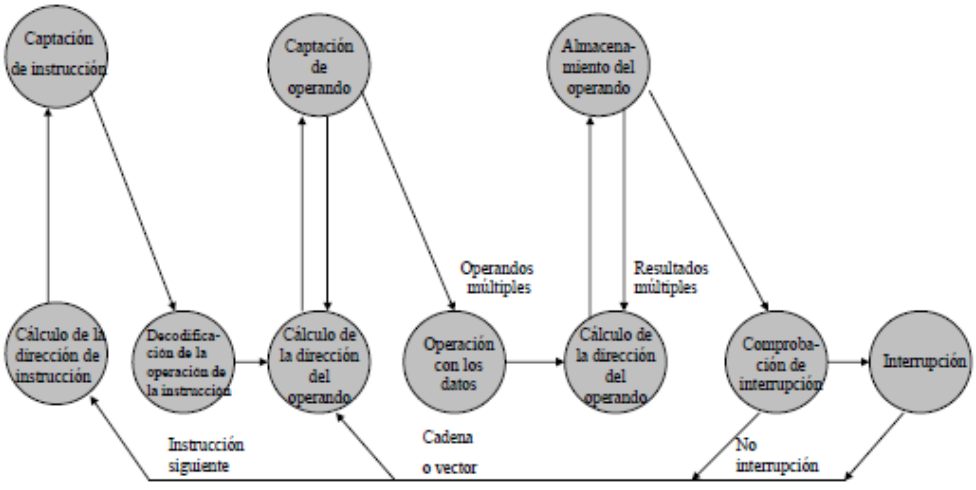
Hay unos pocos parámetros o elementos de diseño que sirven para distinguir y caracterizar a los buses:

- Tipo: las líneas de bus se pueden dividir en dos tipos genéricos: dedicadas y multiplexadas.
  - Dedicado: se refiere a un bus que está reservado exclusivamente para la comunicación entre un componente específico del sistema y otros dispositivos.
  - Multiplexado: indica si el bus multiplexa (combina) múltiples señales o canales de datos en un solo medio de transmisión.

- Método de arbitraje: se refiere al proceso utilizado para gestionar y controlar el acceso de múltiples dispositivos al bus.
  - Centralizado: se refiere al enfoque de arbitraje en el que un único controlador central administra el acceso al bus.
  - Distribuido: indica un método de arbitraje en el que múltiples dispositivos compiten entre sí por el acceso al bus.
- Temporización: hace referencia a la coordinación de las operaciones de transferencia de datos en el bus en relación con el tiempo.
  - Síncrono: las operaciones de transferencia de datos están sincronizadas con una señal de reloj común.
  - Asíncrono: las operaciones de transferencia de datos no están sincronizadas con una señal de reloj común. Las operaciones pueden ocurrir en cualquier momento.
- Anchura del bus: número de líneas o cables en el bus que se utilizan para transmitir datos simultáneamente.
  - Dirección: líneas de un bus que se utilizan para transmitir direcciones de memoria o de registros a los dispositivos conectados.
  - Datos: líneas de un bus que se utilizan para transmitir los propios datos entre los dispositivos.
- Tipo de transferencia de datos: se refiere al movimiento de información desde una ubicación de almacenamiento hacia otro componente del sistema.
  - Lectura: proceso de obtener datos desde una fuente hacia un dispositivo que los solicita.
  - Escritura: proceso de enviar datos desde un dispositivo que los genera hacia un destino, como la memoria o un dispositivo periférico.
  - Bloque: transferencia de una cantidad específica de datos agrupados en una unidad llamada bloque.

CLASE 4 - SEGMENTACIÓN DE CAUCE

DIAGRAMA DE ESTADOS DEL CICLO DE INSTRUCCIÓN



4.1) SEGMENTACIÓN DE INSTRUCCIONES

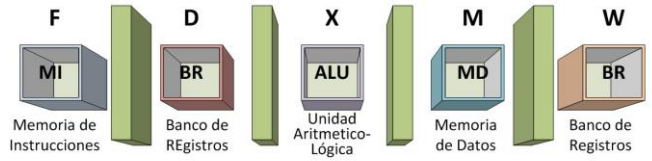
SEGMENTACIÓN Y SOLAPAMIENTO DE CAUCE (PIPELINING)

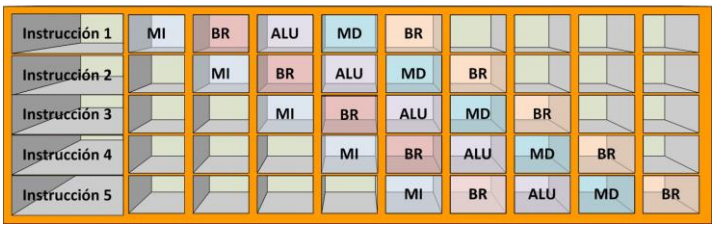
La segmentación de cauce es una forma efectiva de organizar el hardware de la CPU para realizar más de una operación al mismo tiempo. Consiste en descomponer el proceso de ejecución de las instrucciones en fases o etapas que permitan una ejecución simultánea (explotando el paralelismo entre las instrucciones de un flujo secuencial). Las instrucciones ejecutadas no necesitan esperar la terminación de la previa para comenzar a resolverse.

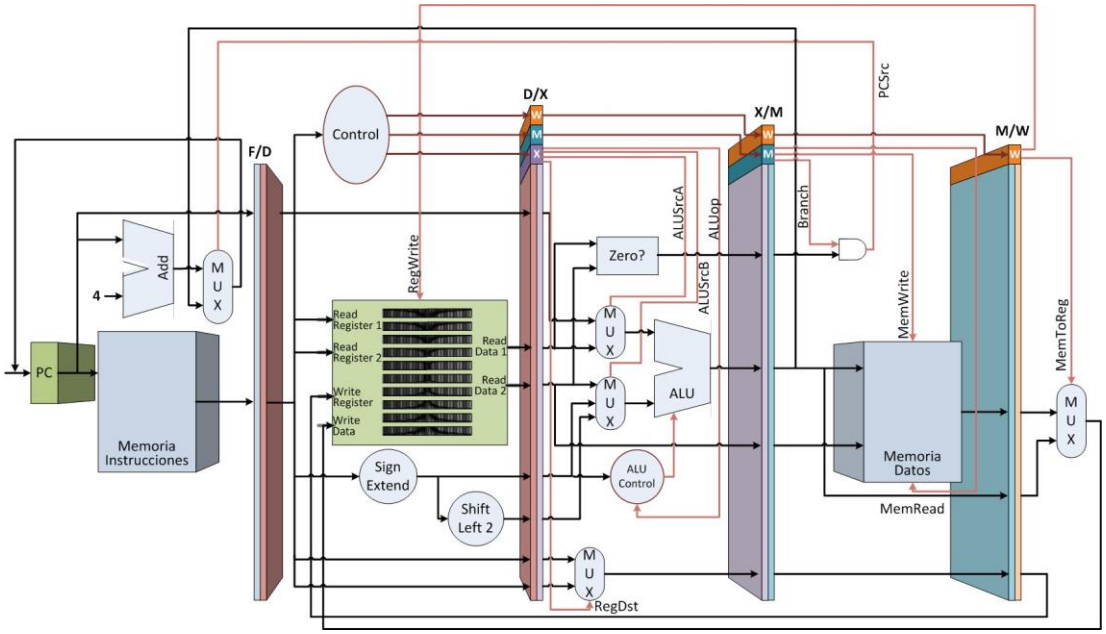
CONCLUSIONES

- Cada instrucción pasa, durante su ejecución, por varias etapas, en cada una se realiza solo una parte del todo.
- La entrada de una nueva instrucción puede hacerse antes de que se terminen las anteriores.
- Por ende, varias instrucciones están siendo manipuladas simultáneamente, cada una en un estado de ejecución distinto.
- Dado que el procesador tarda menos tiempo en resolver un conjunto de instrucciones, la segmentación mejora las prestaciones.
- Como el programador no interviene en la segmentación y solapamiento de las instrucciones, la segmentación es “invisible” al programador.

ETAPAS DEL CAUCE







- Etapas:
  - IF: Búsqueda de instrucción:
    - Se accede a memoria (MI) por la instrucción.
    - Incrementa PC.
  - ID: Decodificación de instrucción:
    - Se determina el código de operación.
    - Se accede al banco de registros (BR) por el/los operando/s.
    - Se calcula el valor del operando inmediato con extensión de signo (si hace falta).

    - EX: Ejecución de instrucción:
        - Se ejecuta la operación en la ALU.
    - MEM: Escribir o leer de memoria de datos (MD).
    - WB: Se almacena el resultado en el banco de registros (BR).
- Recursos de la máquina:
    - MI: memoria de instrucciones.
    - BR: banco de registros.
    - ALU: unidad aritmético-lógica.
    - MD: memoria de datos.
    - Se deben segmentar las señales que controlan las unidades funcionales porque cada unidad está operando con diferentes instrucciones.
- Se considera que cada instrucción requiere cinco ciclos de reloj, uno por cada segmento.
- En cada ciclo se incorpora una nueva instrucción, sin haber completado las anteriores.
- Para implementar la segmentación del cauce se requiere intercalar registros entre cada etapa.
- Los registros separan las unidades funcionales, para que puedan operar con distintas instrucciones.

PRESTACIONES DE UN CAUCE SEGMENTADO

El máximo rendimiento teórico se obtiene cuando se completa una instrucción en cada ciclo de reloj. En esas condiciones las unidades funcionales están trabajando simultáneamente con distintas instrucciones. Cuanto mayor es el número de tapas del cauce, mayor es su potencial para conseguir aceleración. No se mejora la velocidad de la instrucción, sino que la principal ventaja es el incremento de la productividad (throughput), es decir la cantidad de instrucciones resueltas en un período de tiempo determinado.

Sin embargo, los beneficios potenciales de etapas adicionales en el cauce se contrarrestan por los incrementos en coste, los retardos entre etapas, y el hecho de que se encontrarán saltos que requieran vaciar el cauce.

RIESGOS EN EL CAUCE

Los conflictos mencionados anteriormente son denominados riesgos, y existen tres tipos:

- Estructurales: son conflictos provocados por el uso de los recursos del sistema. Los recursos típicamente son: memoria, ALU, registros.
    - En el caso de que la memoria fuera una sola habría un conflicto cuando dos etapas quieren acceder a memoria. La solución es dividir la memoria para reducir los conflictos por accesos a memoria.
    - Otra opción para evitar el conflicto por el acceso a un recurso es retardar la ejecución de la tarea los ciclos de reloj necesarios hasta que desaparece el conflicto. Pero al perder ciclos de reloj se pierde tiempo y baja el rendimiento del procesador.
- Por dependencia de datos: son conflictos originados entre dos o más instrucciones que comparten un mismo dato dentro del cauce.
    - Una etapa podría depender del contenido de un registro que podría verse alterado por una instrucción previa que aún esté en el cauce.
    - Los operandos fuente o destino de una instrucción no están disponibles en el momento en que se necesitan en una etapa determinada del cauce.
    - Los tres tipos de dependencias de datos son:
        - Lectura después de Escritura (RAW, dependencia verdadera): una instrucción escribe un dato que otra lee posteriormente.
        - Escritura después de Escritura (WAW, dependencia en salida): una instrucción escribe un dato que otra escribe posteriormente, sólo ocurre si se permite que las instrucciones se adelanten unas a otras.

        - Escritura después de Lectura (WAR, anti dependencia): una instrucción lee un dato que otra escribe posteriormente.
- Por dependencia de control: son conflictos que ocurren cuando la ejecución de una instrucción depende de cómo se ejecute otra.
    - Son riesgos que pueden ocurrir cuando se va a ejecutar una instrucción de salto condicional.
    - Una instrucción tiene que calcular el nuevo valor que modifica el valor del PC. La próxima instrucción no puede comenzar hasta que no se resuelva el dato.
    - Una forma de resolver el conflicto es retrasar varios ciclos la próxima instrucción hasta que resuelva el cálculo de la instrucción de salto.

CLASE 5 - POSIBLES SOLUCIONES A ATASCOS

5.1) SOLUCIONES

INTRODUCCIÓN - SOLUCIÓN ELEMENTAL

En la clase anterior se identificaron los tipos de conflictos que aparecen en un procesador segmentado real: atascos estructurales, por dependencia de datos y por dependencia de control.

La solución más elemental para superar estos atascos, es agregar puntos de parada en el cauce hasta que el conflicto desaparezca. El gran problema que tiene este tipo de solución es que el rendimiento cae fuertemente por la cantidad de tiempos muertos que se insertan. Hay varias técnicas que permiten reducir y, y en algunos casos, eliminar los efectos de los riegos.

SOLUCIÓN A LOS CONFLICTOS ESTRUCTURALES

La solución a estos conflictos consiste en “agregar más hardware”. Algunas estrategias usadas son:

- Duplicación de recursos de hardware. Ejemplo: agregar sumadores o restadores además de la ALU.
- Separación del recurso en conflicto. Ejemplo: separar memorias de instrucciones y datos (Harvard).
- Subdividir el acceso al recurso. Ejemplo: escribir en el primer semiciclo de reloj y leer en el segundo.

SOLUCIÓN A LOS CONFLICTOS POR DEPENDENCIA DE DATOS

Las soluciones se pueden implementar de dos maneras:

1. Por hardware:
    - Retardos en las etapas del cauce: consiste en agregar retardos o ciclos de parada en la ruta de datos.
        - El gran problema de esta situación es que los ciclos de parada agregados reducen fuertemente la performance del procesador segmentado.
    - Forwarding (adelantamiento): este método consiste en pasar directamente desde una unidad funcional a otra, el resultado obtenido en una instrucción a las instrucciones siguientes que lo necesitan como operando.
        - Esta solución puede ser sencilla de implementar si se identifican todos los adelantamientos y se pueden adelantar los datos a las unidades que lo necesitan.
        - Se debe implementar un nuevo hardware (CPU más compleja: más cara o más lenta).
        - Se deben habilitar caminos de retroalimentación de la salida de la ALU a la entrada, de MEM a la ALU.
        - No elimina al 100% los atascos RAW si no que los minimiza.
2. Por software:
    - Introducción de instrucciones NOP: se insertan entre las instrucciones que tienen dependencia de datos, pero genera retardos que reducen la performance del procesador.

- Reordenamiento de instrucciones: en lugar de NOP, se separan las instrucciones con dependencia de datos, pero hay que tener cuidado con no modificar el comportamiento del programa.

SOLUCIÓN A LOS CONFLICTOS POR DEPENDENCIA DE CONTROL

Los saltos pueden ser incondicionales (donde la dirección de destino se debe determinar lo más pronto posible, dentro del cauce, para reducir la penalización) o condicionales (introducen el riesgo adicional por la dependencia entre la condición de salto y el resultado dependiente de una instrucción previa).

Igual que antes, la forma más elemental de resolver el conflicto es agregando tiempos muertos hasta que desaparece el programa. Se puede mejorar la situación anterior si se modifica la ruta de datos para reducir la cantidad de ciclos muertos, adelantando la resolución de los saltos. El objetivo es evaluar tempranamente la condición de salto.

La solución propuesta no es tan sencilla de implementar. Generalmente, las técnicas para mitigar el impacto en la performance de procesadores segmentados, de las instrucciones de salto pueden ser dos tipos:

- Técnicas por hardware: que consiste en resolver los conflictos a nivel de hardware mediante predicción de saltos.
  - Técnicas estáticas: no tienen en cuenta información previa de la ejecución del programa.
    - Se presume una condición (salta o no salta) y se ejecuta en base a esa predicción.
    - Predecir que nunca se salta: asume que el salto no se producirá, y por lo tanto siempre capta la siguiente instrucción.
    - Predecir que siempre se salta: asume que el salto se producirá, y por lo tanto siempre capta la instrucción destino del salto.
  - Técnicas dinámicas: tiene en cuenta la historia previa del programa en ejecución.
    - Conmutador Saltar/no saltar: estando en uno de las dos condiciones, se requieren dos predicciones fallidas consecutivas para conmutar al otro estado.
      - Esta predicción tiene un buen comportamiento en lazos donde el resultado de una consulta se puede repetir muchas veces antes de que cambie el resultado de esta.
    - Tabla de historia de saltos (BHT): se implementa usando una pequeña cache asociada a la etapa de búsqueda. La cache tiene tres campos: una dirección de una instrucción del tipo bifurcación, una dirección del destino/instrucción destino y n bits de estado (historia de uso).
      - Cada vez que se precapta una instrucción, se busca en la cache BHT si la instrucción es de ramificación.
      - Se toma una decisión predictiva en función de los bits de la historia de uso.
      - Si la predicción es saltar, se capta la dirección de salto. Si esta falla, se actualiza la tabla.
      - Si la instrucción de ramificación no estaba en la BHT, se la carga como nueva entrada.
    - Predicción según el código de operación: se basa en la suposición de que algunas instrucciones de salto tienen mayor probabilidad de saltar o de no saltar.
      - Si la instrucción de salto involucra un lazo que se repite muchas veces, la condición que se usa para repetir este lazo ocurrirá mas veces que la se usa para salir de este.
    - Varios cauces: en esta solución se usan cauces distintos para ejecutar simultáneamente los cauces, el que corresponde a la opción de no saltar y el que corresponde a la instrucción de salto.
      - Cuando se determina el resultado de la ramificación, se utiliza el cauce correcto y se descarta el que no lo es.
      - Se aumentan las búsquedas, provocando retardos en el acceso al bus y registros.
      - Si hay nuevos saltos y los cauces están ocupados, se necesitan nuevos cauces.

- Precaptación del destino del salto (prefetch branch target): se precapta la instrucción destino del salto, además de las instrucciones siguientes a la bifurcación.
  - La instrucción se guarda hasta que se ejecute la instrucción de bifurcación.
- Buffer de bucles: se usa una memoria muy rápida, gestionada por la etapa de captación de instrucciones, que contiene las últimas instrucciones recientemente buscadas.
  - Si hay una instrucción de salto, el hardware comprueba si está en el buffer de bucles. Si es así, la próxima instrucción se busca desde el buffer.
  - Muy eficaz para pequeños bucles y saltos, si el buffer es capaz de contener todo el bucle. Es parecido a una cache, solo que contiene instrucciones consecutivas.

- Técnicas por software: que intentan resolver los conflictos por software a nivel del compilador.
  - Las técnicas de resolución por software de conflictos se basan en tratar de realizar trabajo útil mientras el salto se resuelve.
  - Cuando hay una instrucción de salto, se necesita uno o mas ciclos de tiempo para determinar si se va a ejecutar el salto o no. Ese tiempo se llama hueco o ranura de retardo de salto (Branch delay slot).
  - Hay máquinas que captan y ejecutan siempre la instrucción siguiente a una instrucción de ramificación.
  - El compilador puede tratar de insertar, en dichos huecos, instrucciones útiles que en lo posible no dependan del salto. De esta manera se elimina el conflicto y se hace trabajo útil.
  - Una técnica en caso de no ser posible insertar instrucciones, es agregar instrucciones del tipo NOP para evitar el conflicto.
  - La otra forma de llenar los huecos que quedan luego de las instrucciones de salto es mediante el reordenamiento de las instrucciones.

CLASE 6 - PROCESADORES RISC Y CISC

6.1) CISC

¿POR QUÉ CISC?

Existió una tendencia hacia repertorios de instrucciones más ricos, que incluyan un mayor número de instrucciones e instrucciones más complejas. Esta tendencia ha sido motivada por dos razones principales: el deseo de simplificar los compiladores, y el deseo de mejorar las prestaciones.

La primera de las razones, la simplificación de los procesadores, consistía en generar una secuencia de instrucciones máquina para cada sentencia de HLL (lenguajes de alto nivel), pero esto era mucho más difícil de lo que parecía, ya que se debía optimizar el código generado para minimizar su tamaño, reducir el número de instrucciones ejecutadas y mejorar la segmentación.

La otra razón importante es la esperanza de que un CISC produzca programas más pequeños y rápidos. Como el programa ocupa menos memoria, hay un ahorro de este recurso. El problema de este razonamiento es que está lejos de ser cierto que un programa para RISC sea más pequeño que para un RISC. En muchos casos, el programa el CISC, expresado en lenguaje máquina simbólico, puede ser más corto pero el número de bits de memoria que ocupa no tiene por qué ser más pequeño.

CARACTERÍSTICAS DE LOS PROCESADORES CISC

Los conceptos anteriores sentaron las bases de diseño de los procesadores CISC (Complex Instruction set computer - procesador con repertorio de instrucciones complejo):

- Repertorios de instrucciones grandes.

- Debido a que hay más instrucciones en un CISC, son precisos códigos de operación más largos y de longitud variable, produciendo instrucciones más largas.
- Muchos modos de direccionamiento.
- Capacidad de implementar sentencias de HLL con muy pocas instrucciones de máquina. Esta idea consiste en resolver una sentencia de HLL "por hardware".
- El repertorio de instrucciones complejo exige una Unidad de control más compleja y lenta (penalización).

ESTUDIOS

Se realizaron distintos estudios sobre el comportamiento de los procesadores tipo CISC. Los estudios se realizaron de dos maneras distintas: estáticos (computaban cantidades basados en el código objeto ejecutable, no tiene en cuenta el flujo del programa ejecutado) y dinámicos (computaban cantidades en base a la ejecución de diferentes programas).

Los estudios se orientaron a:

1. Uso del repertorio de instrucciones: estaban orientados a determinar el uso de las instrucciones, y, por lo tanto, su interacción con la memoria.
    - Los accesos a memoria tienen mucho impacto en el tiempo de ejecución del programa.
    - Los llamados a procedimiento consumen mucho tiempo y tiene que hacerse lo más eficiente posible para optimizar los accesos a memoria. Sobre los procedimientos:
        - La mayoría de los programas tienen una secuencia corta de llamadas seguida por la secuencia de retornos.
        - La mayoría de las variables son locales y las referencia a operandos están muy localizadas.
2. Uso de los operandos: intentaban determinar cuáles eran los tipos de datos usados y su frecuencia de uso.
    - Uso de variables locales (80%) del procedimiento.
    - Uso de punteros del tipo variable local y escalar para manejo de estructuras de datos.
    - Uso intensivo de registros.
3. Secuencia de ejecución de instrucciones: tenían como objetivo determinar el impacto de disponer de un repertorio de instrucciones largo y complejo, en la Unidad de Control, y en el cauce de datos que tiene relación con la eficiencia en la ejecución de una secuencia de instrucciones.

Las conclusiones finales a las que se arribaron luego de muchos estudios fueron:

1. Optimización: se necesita optimizar el tiempo de ejecución de las instrucciones más usadas y las que consumen más tiempo.
2. Simplificación: de las instrucciones, para resolver más rápidamente.
3. Ajuste: del cauce de datos para resolver las instrucciones más usadas lo más rápido posible.
4. Registros: usar los registros en forma intensiva, para manejo de operandos. Disponer de un banco de registros significativo para reducir los accesos a memoria.

6.2) RISC

CARACTERÍSTICAS DE LOS PROCESADORES RISC

Los primeros RISC (Repetorio Reducido de Instrucciones) tenían, entre otras, las siguientes características relevantes:

- Repertorio de instrucciones reducido y básico.
- Formato de instrucción fijo y sencillo (todas las instrucciones tienen la misma cantidad de bits o bytes).
    - Solo se usa un formato o unos pocos.
    - La longitud de las instrucciones es fija, y alineada en los límites de una palabra.

    - Las posiciones de los campos, en especial la del código de operación, son fijas, haciendo que la decodificación de este código y el acceso a los operandos en registros puedan tener lugar simultáneamente.
- Unidad de control cableada y no microprogramada.
- Diseño del cauce optimizado para ejecutar una instrucción por ciclo.
    - Las instrucciones máquina de un RISC no deberían ser más complicadas que las microinstrucciones de las máquinas CISC, y deberían ejecutarse más o menos igual de rápido.
- Operaciones registro a registro.
    - Esta forma de diseño simplifica el repertorio de instrucciones y la unidad de control.
    - Otra ventaja es que tal arquitectura fomenta la optimización del uso de registros, ya que los operandos accedidos frecuentemente, permanecen en el almacenamiento de alta velocidad.
- Modos de direccionamiento sencillos.
    - Direccionamiento a registro. Se pueden incluir varios modos adicionales, como el desplazamiento y el relativo al contador de programa.
    - Otros modos más complejos se pueden sintetizar por software a partir de los simples.
    - Simplifica nuevamente el repertorio de instrucciones y la unidad de control.
- Uso intensivo de banco de registros con optimización por software.
- Máquina tipo LOAD-STORE: solo se accede a memoria con instrucciones de movimiento de datos. Instrucciones aritméticas y lógicas solo entre registros.

VENTANA DE REGISTROS

Un aspecto esencial de los procesadores RISC es disponer de un banco de registros amplio, para reducir los accesos a memoria, y simplificar las instrucciones. La optimización del banco de registros se puede hacer de dos maneras:

- Por hardware: agregando más registros.
    - Ventajas:
        - Reducción de accesos a memoria.
        - Disponibilidad para más variables escalares locales.
    - Desventajas:
        - Más bits en la instrucción para identificar el registro.
        - En llamadas y retornos de subrutina hay que considerar que se requiere tiempo para pasar parámetros, asignar espacio para las variables locales, y devolver resultados (mucho tiempo).
            - Si la subrutina usara registros para manipular los tipos de datos de entrada, propios, de salida y globales, necesitaría disponer de un banco de registros para cada tipo.
            - Si cada subrutina administra su banco de registros, excepto los globales (comunes y accesibles por todas las subrutinas), que son comunes a todos, la cantidad real de registros de cada subrutina se limita a un número no muy grande.
            - Pasar argumentos de una subrutina a otra requiere de un tiempo considerable.
            - Cuantos más registros tenga la subrutina, más tiempo se consume en pasar la información.
            - Cuantos más registros necesite localmente, más tiempo se requiere en reservarlos.
    - La solución a los problemas en el uso de procedimientos y pasaje de parámetros es la Ventana de registros y consiste en:
        - 1) Asignar a cada subrutina un banco de registros propio.
        - 2) Superponer los registros donde recibe los parámetros una subrutina con los registros donde pasa los argumentos la subrutina que la llama.
            - Cada subrutina accede a una ventana de registros, de los cuales los primeros y los últimos están solapados, y son usados para pasaje de parámetros.

- El banco se implementa como un buffer de tipo circular, con una capacidad determinada de "ventanas", basada en la "profundidad" de anidamientos admitidas por el procesador.
- La cantidad de registros accesibles por cada subrutina es la misma, pero cambian los registros físicos a los que accede.
- Para el manejo de variables globales se pueden usar dos soluciones:
    - 1. El compilador asigne posiciones de memoria a las variables: es ineficiente para variables globales a las que se accede frecuentemente porque requiere accesos a memoria que son inherentemente lentos.
    - 2. Incorporar en el procesador un conjunto de registros para variables globales accesible en todos los niveles. Este banco de registro no es mostrado en la ventana.

- Por software: optimizando la asignación de registros a las variables que se usen más en un período de tiempo.
    - Los lenguajes de HLL no tienen referencias explícitas a los registros.
    - Como la asignación de registros se hace en la compilación, el uso optimizado de registros es responsabilidad del compilador (no del programador).
    - Las estrategias de optimización tratan de asignar los pocos registros a las variables más usadas o las que más tiempo permanecerán en el registro.
    - Una estrategia consiste en suponer que se disponen de infinitos registros ("virtuales"). A cada variable del programa se le asigna un registro virtual.
        - El compilador mapea (asigna) el número ilimitado de registros simbólicos a un número de registros reales.
        - Los registros simbólicos que no se solapan pueden compartir el registro real. Si se agotan los registros reales, algunas de las variables se asignan a posiciones de memoria.
        - En la optimización se usa la técnica denominada coloreado de grafos.

6.3) COMPARACIÓN ENTRE RISC Y CISC

DIFERENCIAS

Se puede apreciar que:

- Repertorio de instrucciones: los CISC tienen muchas más instrucciones que los RISC.
- Tamaño de instrucción: en los CISC es muy variable, mientras que en los RISC es fija.
- Modos de direccionamiento: los CISC tiene una buena variedad de modos de direccionamiento, mientras que los RISC son mínimos.
- Banco de registros: en los RISC son amplios, y más limitados en los CISC.
- Memoria de control: típicamente microprogramada en los CISC, lógica "cableada" en los RISC.

¿ES UNO MEJOR QUE EL OTRO?

Se han realizado números mediciones comparativas entre procesadores RISC y CISC, pero las comparaciones tienen varios problemas: no existe un par de máquinas RISC y CISC directamente comparables, y no hay un conjunto de programas de prueba definitivo.

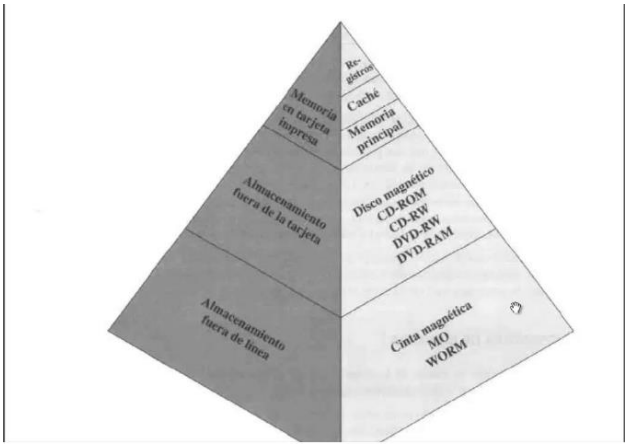
No existe una marcada diferencia de performance entre uno y otro, y no está clara la barrera que separa uno u otro estilo. Hoy en día, la mayoría de las máquinas son una mezcla de ambas, incluyendo las mejores ventajas de ambos.

CLASE 7 - MEMORIA

7.1) JERARQUÍA DE MEMORIA

CONCEPTOS BÁSICOS

La forma en que se organizan coordinadamente los distintos tipos de memoria se conoce con el nombre de Jerarquía de Memoria. La jerarquía de memoria es un método de administración del almacenamiento de información (memoria) estructurado en varios niveles ubicados físicamente en distintos lugares, con tecnologías, costos, tamaños y velocidades distintas.



- A menor tiempo de acceso, mayor costo por bit.
- A mayor capacidad, menor costo por bit.
- A mayor capacidad, mayor tiempo de acceso.

- Registros: manejados por el compilador.
- Cache: la administración se hace por el hardware.
- Memoria principal: la administración la puede hacer o el hardware, sistema operativo o programador.

Los principales objetivos de una jerarquía de memoria son:

- Maximizar tamaño: idealmente disponer de una “capacidad ilimitada”, equiparada al tamaño del nivel más grande.
- Optimizar velocidad: simular que se dispone de un banco de memoria “ultrarrápida”, próximo a la velocidad del nivel más rápido.
- Minimizar el costo total: implementar una memoria a un costo cercano al del nivel más lento.

¿POR QUÉ FUNCIONA LA JERARQUÍA DE MEMORIA?

La jerarquía de memoria funciona eficazmente debido a la explotación del principio de localidad de las referencias y a la combinación de diferentes tipos de memoria con características complementarias en términos de velocidad, capacidad y costo.

El principio de localidad de las referencias establece que el procesador en períodos de tiempos cortos, trabaja principalmente con agrupaciones (clusters) fijas de referencias a memoria. De esta manera es posible organizar los datos a través de la jerarquía, colocando los datos más frecuentemente utilizados en los niveles superior de la jerarquía, de tal manera que el porcentaje de accesos a cada nivel siguiente más bajo, sea sustancialmente menor que al nivel anterior. Este principio puede aplicarse a través de más de dos niveles de memoria, como sugiere la jerarquía.

Una vez explicado este principio, podemos dividirlo en dos subprincipios:

- Localidad temporal de las referencias: se refiere a la tendencia de un programa a acceder repetidamente a los mismos datos en un corto período de tiempo. En otras palabras, si un dato ha sido accedido recientemente, es probable que se vuelva a acceder en el futuro cercano.
- Localidad espacial de las referencias: se refiere a que es altamente probable que los próximos elementos de memoria referenciados estén en las proximidades de los últimos referenciados.

Además, para que la memoria se comporte como una jerarquía debe cumplir dos propiedades:

- Inclusión: los datos almacenados en un nivel han de estar también almacenados en los niveles inferiores a él.
- Coherencia: las copias de la misma información en los distintos niveles deben contener los mismos valores.

7.2) MEMORIA CACHÉ

PRINCIPIOS BÁSICOS

La memoria caché es una memoria pequeña y muy rápida, que se ubica entre la memoria principal y la CPU. Puede localizarse en un chip separado o dentro de la CPU, o en ambos lugares. El objetivo de la memoria caché es lograr que la velocidad de la memoria sea lo más rápida posible, consiguiendo al mismo tiempo un tamaño grande al precio de memorias semiconductoras menos costosas.

ORGANIZACIÓN

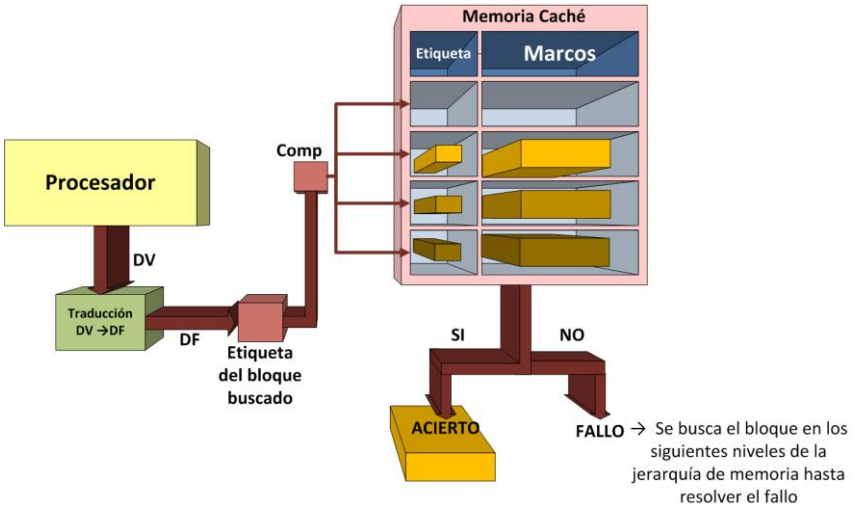
La información contenida en la caché se organiza en bloques (ranuras) de longitud fija. En dichos bloques se copian algunos bloques de la memoria principal y la cantidad de bloques copiados depende tanto del tamaño de la memoria caché como del bloque. Cada ranura tiene asociada una etiqueta (porción de la dirección de memoria principal) para identificar el bloque de la memoria que tiene copiado, y el conjunto de todas las etiquetas forman el directorio de la caché.

FUNCIONAMIENTO

Cuando el procesador necesita de un dato, se genera una dirección de memoria y se hace una comprobación para determinar si dicho dato se encuentra en la caché. Pueden ocurrir dos situaciones:

- Acierto: si tiene el dato, se lo entrega al procesador (a la velocidad de la cache).
- Fallo: si no lo tiene, un bloque de memoria principal, consistente en un cierto número de palabras, se transfiere a la cache, y, después, el dato es entregado a la CPU ya que, debido al fenómeno de localidad de las referencias, es probable que se hagan referencias futuras a otras palabras del mismo bloque solicitado.
    - Latencia: es el tiempo necesario para completar un acceso a memoria (depende de la memoria).
    - Ancho de banda: es la velocidad a la cual se puede transferir el dato.

La eficiencia del uso de la cache depende de la cantidad de veces que “acierta” la cache.



ELEMENTOS DE DISEÑO DE LA CACHÉ

Para el diseño de la caché hay que tener en cuenta varias consideraciones:

TAMAÑO DE CACHÉ

- La caché debe ser suficientemente grande para contener la mayor cantidad posible de información.
- Cuanto más grande es la caché, mayor es el número de puertas implicadas en direccionarla. El resultado es que cachés grandes tienen a ser ligeramente más lentas que las pequeñas.
- El tamaño de las cachés está también limitado por las superficies disponibles de chip y de tarjeta que contiene el procesador.

TAMAÑO DE LÍNEA

- A medida que aumenta el tamaño del bloque, la tasa de aciertos primera aumenta debido al principio de localidad. Sin embargo, la tasa de aciertos comenzará a decrecer cuando el tamaño de bloque se haga aún mayor y la probabilidad de utilizar la nueva información captada se haga menor.
- Bloques más grandes reducen el número de bloques que caben en la cache.
- A medida que un bloque se hace más grande, cada palabra adicional está mas lejos de la requerida, y por tanto es más improbable que sea necesaria a corto plazo.
- Un tamaño entre 4 y 8 unidades direccionables parece estar próximo al óptimo.

NÚMERO DE CACHÉS

- Niveles de la caché: la memoria caché puede ser una sola o estar dividida en varias unidades (múltiples niveles). Los diseños más actuales incluyen tanto una cache on-chip como otra externa, resultando en una caché de dos niveles, siendo la cache interna el nivel 1 (L1) y la externa el nivel 2 (L2).
  - La implementación de caches de dos niveles permite optimizar el rendimiento del sistema al aprovechar la localidad temporal y espacial, establecer una jerarquía de memoria efectiva, reducir la latencia media y optimizar los costos del sistema.
  - La mejora potencial del uso de una cache L2 depende de las tasas de aciertos en ambas caches.
- Una cache puede estar unificada o partida. Más recientemente se ha hecho normal separar la cache en dos: una dedicada a instrucciones y otra a datos.
  - La ventaja clara de este diseño es que elimina la competición por la caché entre el procesador de instrucciones y la unidad de ejecución.
  - Además, ambas cachés pueden tener distintos tamaños y políticas de acceso y reemplazo.

FUNCIÓN DE CORRESPONDENCIA

La elección de la función de correspondencia determina cómo se organiza la cache, es decir la forma en la que se van a asignar los bloques de la memoria principal en la memoria caché.

Pueden utilizarse tres técnicas:

- Correspondencia directa: consiste en hacer corresponder cada bloque de memoria principal a sólo una línea posible de la caché.
  - Se implementa utilizando la dirección. Cada dirección de memoria principal puede verse como dividida en tres campos:
    - Tag: es el número de grupo en la memoria principal.
    - Line: es el número de bloque dentro del grupo.
    - Word: es el número de palabra dentro del bloque.
- El uso de una parte de la dirección como número de línea proporciona una correspondencia única o asignación única de cada bloque de memoria principal en la cache.

- Si la comparación da un acierto, el dato se busca en la caché. Si la comparación da una falla, el dato se trae de la memoria principal.
- Esta técnica es simple y poco costo de implementar.
- Su principal desventaja es que hay una posición concreta de caché para cada bloque dado. Si un programa accede a palabras de dos bloques diferentes en la misma línea de caché, se intercambiarán continuamente, resultando en baja tasa de aciertos y grandes pérdidas de caché.

- Correspondencia asociativa: permite que cada bloque de memoria principal puede cargarse en cualquier línea de caché.
  - La lógica de control interpreta una dirección de memoria como una etiqueta (tag) y un campo de palabra (word).
  - El campo de etiqueta identifica unívocamente un bloque de memoria principal.
  - Para determinar si un bloque está en la caché, su lógica de control (memoria del tipo asociativa) debe examinar simultáneamente todas las etiquetas de líneas para buscar una coincidencia.
  - Si la comparación da un acierto, el dato se busca en la caché. Si la comparación da una falla, el dato se trae de la memoria principal.
  - Hay flexibilidad para que cualquier bloque sea reemplazado cuando se va a escribir uno nuevo en la cache.
  - La principal desventaja de este tipo de correspondencia es la compleja circuitería necesaria para examinar en paralelo las etiquetas de todas las líneas de caché.
- Correspondencia asociativa por conjuntos: un bloque puede almacenarse en un conjunto restringido de lugares en la caché.
  - Un bloque de memoria principal puede colocarse en bloques determinados de la cache. La caché se divide en un número de conjuntos N.
  - Cada conjunto N contiene un número de líneas o ranuras.
  - Un bloque determinado corresponderá a alguna línea o ranura de un conjunto determinado.
  - La etiqueta de una dirección de memoria es mucho más corta, y se compara sólo con las etiquetas dentro de un mismo conjunto.
  - Combina lo mejor de las otras correspondencias (asociativa y directa).

ALGORITMOS DE SUSTITUCIÓN

Cuando hay un fallo de acceso a la memoria caché, se debe traer un bloque desde la memoria principal y almacenar un bloque existente en la memoria cache. El lugar a donde va a ser ubicado el bloque a traer desde la memoria requiere reemplazar un bloque existente.

- Para el caso de correspondencia directa, solo hay una posible línea para cada bloque particular, y no hay elección posible. Por lo tanto, si se requiere traer un nuevo bloque desde la memoria principal, indefectiblemente reemplazará el que está usando esa ranura actualmente.
- Para las técnicas asociativas, se requieren algoritmos de sustitución (implementados en hardware):
  - Utilizado menos recientemente (LRU): se sustituye el bloque que se ha mantenido en la caché por más tiempo sin haber sido referenciado. Requiere controles de tiempos y aprovecha la localidad temporal. Se agrega un bit de USE en cada ranura para identificar el usado recientemente.
  - Primero en entrar-primero en salir (FIFO): se sustituye aquel bloque del conjunto que ha estado más tiempo en la caché. Requiere controles de acceso para identificar el orden en que ingresaron.
  - Utilizado menos frecuentemente (LFU): se sustituye aquel bloque del conjunto que ha experimentado menos referencias. Este algoritmo podría implementarse asociando un contador a cada línea.
  - Aleatoria: consiste simplemente en reemplazar una línea al azar entre las posibles candidatas.

POLÍTICA DE ESCRITURA

Cuando la CPU guarda un resultado en la memoria, puede hacerlo si la dirección está en un bloque de la caché (acierto) o si no lo está (fallo). En cualquiera de las dos situaciones, desde el punto de vista de la coherencia de datos, se debe evitar inconsistencia de información entre las memorias principal y cache, durante los procesos de escrituras. Es decir que, aun escribiéndose el dato en la cache, el correspondiente bloque de la memoria principal debe ser actualizado en algún momento.

Además, más de un dispositivo puede tener acceso a la memoria principal (como por ejemplo un módulo de E/S). Si una palabra ha sido modificada sólo en la cache, la correspondiente palabra no es válida. Además, si el dispositivo de E/S ha alterado la memoria principal, entonces la palabra de caché tampoco es válida.

Políticas de escritura en aciertos:

- Escritura inmediata (write-through): todas las operaciones de escritura se hacen, tanto en caché como en memoria principal, asegurando que el contenido de la memoria principal siempre es válido.
    - La principal desventaja de esta técnica es que genera un tráfico sustancial que puede generar un cuello de botella.
- Post-escritura (write-back): la información sólo se actualiza en la caché y se escribe la memoria cuando se reemplaza el bloque. El bloque requiere de un bit de "sucio" para indicar cuando se lo escribió.
    - A veces porciones de memoria principal no son válidas, y los accesos por parte de los módulos de E/S solo podrían hacerse a través de la caché, complicando la circuitería y generando más problemas.

Políticas de escritura en fallos:

- Asignación sin escritura (no-write allocate): se escribe directamente en la memoria principal. La caché se usa solo en las lecturas. El bloque no se lleva a la memoria caché mientras no se lo tenga que leer.
    - Habitual con la escritura inmediata.
- Asignación con escritura (write allocate): el bloque requerido primero se copia en la caché, y luego se escribe (en la caché).
    - Habitual con post-escritura.

CLASE 8 - PROCESADORES SUPERESCALARES

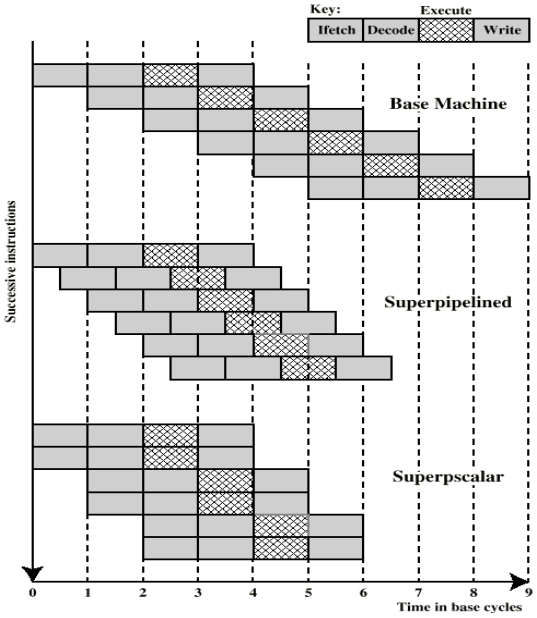
8.1) VISIÓN DE CONJUNTO

TÉCNICAS DE SEGMENTACIÓN

Existen dos técnicas para aumentar la performance del cauce usando segmentación:

- Procesadores supersegmentados: procesador en donde las funciones realizadas en cada etapa se pueden dividir en dos partes no solapadas, y que cada una se ejecute en medio ciclo de reloj.
    - Consiste en subdividir cada segmento en partes más pequeñas.
    - El tiempo de para las instrucciones individuales no varía, pero aumenta el grado del paralelismo de la máquina temporal.
    - El pipeline se hace más profundo, pero está limitado por la tecnología (frecuencia máxima).
- Procesadores superescalares: aquel donde las instrucciones comunes pueden iniciar su ejecución simultáneamente y ejecutarse de manera independiente. El término superescalar hace referencia a una máquina diseñada para mejorar la velocidad de ejecución de las instrucciones escalares.
    - Este procesador es capaz de ejecutar en paralelo dos o más instrucciones en cada etapa.
    - En cada ciclo se inician más de una instrucción, determinando el grado del procesador.
    - Para poder ejecutar dicha cantidad, se requiere la duplicación de partes de la CPU/ALU.

- El grado de paralelismo y, la aceleración de la máquina aumenta, ya que se ejecutan más instrucciones en paralelo.
- Existen limitaciones fundamentales del paralelismo: dependencia de datos verdadera, dependencia relativa al procedimiento, conflictos en los recursos, dependencia de salida y la antidependencia.



8.2) CUESTIONES RELACIONADAS CON EL DISEÑO DEL PROCESADOR SUPERESCALAR

PARALELISMO A NIVEL DE INSTUCCIONES Y PARALELISMO DE LA MÁQUINA

El paralelismo es la capacidad para ejecutar varias tareas en el mismo intervalo de tiempo. Hay dos tipos de paralelismo:

- Paralelismo a nivel de instrucciones: se refiere a las posibilidades que tiene un programa para ejecutar instrucciones en paralelo. Existe cuando las instrucciones de una secuencia son independientes, y, por tanto, pueden ejecutarse en paralelo solapándose.
  - Depende de la frecuencia de dependencias de datos verdaderas y dependencia relativas al procedimiento que haya en el código.
  - Dichos factores dependen a su vez de la arquitectura del repertorio de instrucciones y de la aplicación.
  - También depende de la espera de operación: el tiempo que transcurre hasta que el resultado de una instrucción está disponible para ser usado como operando de una instrucción posterior.
- Paralelismo de la máquina: es una medida de la capacidad del procesador para encontrar y ejecutar tareas en paralelo, sacándole partido al paralelismo a nivel de instrucciones.
  - Depende del número de instrucciones que pueden captarse y ejecutarse al mismo tiempo, y de la velocidad y sofisticación del mecanismo que usa el procesador para localizar instrucciones independientes.

POLÍTICAS DE EMISIÓN DE INSTRUCCIONES

El paralelismo de la máquina no es sencillamente una cuestión de tener múltiples réplicas de cada etapa del cauce. El procesador, además, tiene que ser capaz de identificar el paralelismo a nivel instrucciones, y organizar la capación, decodificación y ejecución de las instrucciones en paralelo. Con respecto a esto, son importantes tres ordenaciones:

- El orden en que se captan las instrucciones.

- El orden en que se ejecutan las instrucciones.
- El orden en que las instrucciones actualizan los contenidos de los registros y de las posiciones de memoria.

Cuanto más sofisticado sea el procesador, menos limitado está por la estrecha relación entre estas ordenaciones. Incluso puede alterar cualquier ordenamiento, respecto del estrictamente secuencial. La única restricción que tiene el procesador es que el resultado debe ser correcto.

Las políticas de emisión de instrucciones son los protocolos usados para el envío de las instrucciones a las unidades funcionales (emisión de las instrucciones). En función del orden en que se emiten, existen tres políticas:

1. Emisión en orden y finalización en orden: consiste en emitir instrucciones en el orden exacto en lo que lo haría una ejecución secuencial (emisión en orden) y escribir los resultados en ese mismo orden (finalización en orden).
    - Esto significa que, aunque se puedan ejecutar varias instrucciones al mismo tiempo, su resultado no se registra hasta que todas las instrucciones anteriores hayan completado su ejecución.
    - No se puede aprovechar completamente el paralelismo disponible en el hardware, lo que puede resultar en una ejecución más lenta.
2. Emisión en orden y finalización desordenada: las instrucciones se emiten exactamente como está en el programa, pero los resultados no necesariamente se completan en el mismo orden (desordenada).
    - Las instrucciones entran a las unidades de ejecución a medida que se van decodificando y hay unidades de ejecución disponibles.
    - Las emisiones de las instrucciones están limitadas por los posibles conflictos de recursos, dependencias de datos y de saltos.
    - Aparece una nueva dependencia, la dependencia de salida. Si una instrucción tarda más en completarse que otras, las instrucciones posteriores pueden completarse antes que las anteriores, usando valores incorrectos.
    - La finalización desordenada necesita una lógica de emisión de instrucciones más compleja que la finalización en orden ya debe ser capaz de detectar las nuevas dependencias y evitar que se ejecuten instrucciones en forma desordenada que puedan alterar el resultado del programa.
    - Además, es más difícil ocuparse de las interrupciones y excepciones.
    - El procesador no puede buscar más allá del punto de conflicto, es decir que no analiza si hay nuevas instrucciones que puedan emitirse sin producir conflicto.
3. Emisión desordenada y finalización desordenada: aquí, tanto la emisión como la finalización de las instrucciones pueden ocurrir en cualquier orden.
    - Para permitir esta emisión desordenada, es necesario desacoplar las etapas del cauce de decodificación y ejecución, mediante un buffer llamado ventana de instrucciones.
        - Cuando un procesador termina de decodificar una instrucción, coloca ésta en la ventana de instrucciones. Mientras no se llene, el procesador puede continuar captando y decodificando nuevas instrucciones.
        - Cuando una unidad funcional de la etapa de ejecución queda disponible, se puede emitir una instrucción desde la ventana de instrucciones a la etapa de ejecución sin que se tenga en cuenta su orden original en el programa. Las instrucciones emitidas no deben tener conflictos por recursos ni por dependencia de datos.
        - Esta etapa no es una etapa adicional del cauce, es un buffer que retiene la información necesaria para emitirla.
    - El resultado de esta organización es que el procesador tiene capacidad de anticipación, lo que le permite identificar instrucciones independientes que pueden introducirse en la etapa de ejecución.
    - Ahora hay más instrucciones dispuestas a ser emitidas, reduciendo la probabilidad de que una etapa del cauce tenga que pararse.

- Surge una nueva dependencia, la antidependencia. En lugar de que la primera instrucción produzca un valor que usa la segunda instrucción, la segunda instrucción destruye un valor que usa la primera.

RENOMBRAMIENTO DE REGISTROS

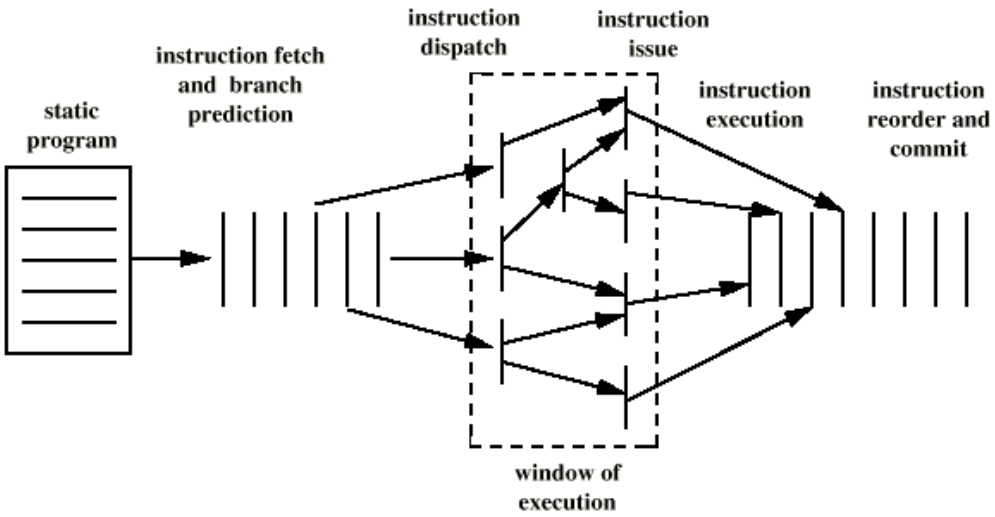
Hay un método para hacer frente al tipo de conflictos de almacenamiento (antidependencias y dependencias), que se basa en la duplicación de recursos. Esta técnica se conoce como renombramiento de registros.

Consiste en que el hardware del procesador asigna dinámicamente los registros (lógicos), que están asociados con los valores que necesitan las instrucciones en diversos instantes de tiempo. Cuando se crea un nuevo valor de registro, se asigna un nuevo registro para ese valor. Las instrucciones siguientes que referencien a ese registro, deben renombrarse para referenciar el registro físico que tiene el dato correcto. De este modo, las referencias a un mismo registro original en diferentes instrucciones, pueden referirse a distintos registros reales, suponiendo diferentes valores, y eliminando las dependencias WAW y WAR, sólo quedando las RAW.

EJECUCIÓN SUPERESCALAR

Las instrucciones son captadas secuencialmente a partir del programa estático, y decodificadas para la detección temprana de dependencias y predicción de saltos. Las instrucciones, todavía ordenadas, son enviadas a la ventana de ejecución, donde se las redistribuye en función de las dependencias verdaderas. Las instrucciones son ejecutadas en un orden basado en la disponibilidad de unidades de ejecución y en las dependencias verdaderas. Por último, las instrucciones se vuelven a poner en un orden secuencial y sus resultados se registran.

Este último paso es necesario para reordenar el almacenamiento de acuerdo al programa principal, eliminar todas las ejecuciones inservibles y que se deben descartar, y mejorar la respuesta a las interrupciones. De esta manera, las instrucciones ejecutadas producen resultados almacenados temporalmente, hasta que se graban los resultados válidos en los registros válidos.



IMPLEMENTACIÓN SUPERESCALAR

Se enumeran los siguientes elementos principales de un procesador superescalar:

- Estrategias de captación de instrucciones que capten simultáneamente múltiples instrucciones.
- Lógica para determinar dependencias verdaderas entre valores de registros, y mecanismos para comunicar esos valores a donde sean necesarios durante la ejecución.
- Mecanismos para iniciar, o emitir, múltiples instrucciones en paralelo.
- Recursos para la ejecución en paralelo de múltiples instrucciones, que incluyan múltiples unidades funcionales segmentadas y jerarquías de memoria capaces de atender múltiples referencias de memoria.

- Mecanismos para entregar el estado del procesador en un orden correcto.

INTERRUPCIONES EN PROCESADORES SUPERESCALARES

En los procesadores superescalares se trata con interrupciones internas o excepciones (rupturas por ejecución de instrucciones) y externas (rupturas por eventos externos). Este tratamiento de interrupciones es más complicado que en los procesadores convencionales ya en que el momento en que se produce una, hay varias instrucciones en ejecución y encima están desordenadas.

- Interrupciones internas o excepciones: existen dos estrategias para el tratamiento de excepciones.
  - Excepciones precisas: las instrucciones que preceden a la que produjo la excepción se completan, y las que le suceden se reinician. El comportamiento es similar al que tendría la misma computadora no segmentada.
    - Los resultados se deben almacenar en el orden en que aparecen las instrucciones. Para ello, se requiere retardar las escrituras de los resultados, para que queden en el orden en el que estaba el programa secuencial.
  - Excepciones imprecisas: no se respetan las condiciones exactas de la máquina no segmentada.
  - Para garantizar un estado consistente (preciso) del programa se requiere: que las instrucciones anteriores terminen correctamente, que la instrucción que origina la excepción y siguientes se aborten, y que tras la rutina de tratamiento se comienza por la instrucción que originó la excepción.
- Interrupciones externas: pueden tener distinto grado de precisión, dado que no necesariamente tiene que ver con las instrucciones en ejecución.
  - Interrupciones imprecisas: se completa lo que estaba en ejecución.
  - Interrupciones poco imprecisas: el software resuelve algunas inconsistencias.
  - Interrupciones precisas: garantizan que solo se completen las instrucciones previas a la interrupción. Para lograrlo, se utiliza un buffer de salida. Cuando ocurre una interrupción, se detiene la emisión de nuevas instrucciones, se cancela la cola de instrucciones pendientes y se completan todas las instrucciones en ejecución antes de procesar la interrupción.

CLASE 9 - PROCESAMIENTO PARALELO

9.1) ORGANIZACIONES CON VARIOS PROCESADORES

La taxonomía introducida primeramente por Flynn es todavía la forma más común de clasificar a los sistemas según sus capacidades de procesamiento paralelo. Flynn propuso las siguientes categorías o clases de computadores:

- Una secuencia de instrucciones y una secuencia de datos (SISD): un único procesador interpreta una única secuencia de instrucciones, para operar con los datos almacenados en una única memoria. Ejemplo: monoprocesadores.
  - Se dispone de una unidad de control (UC) que proporciona una secuencia de instrucciones (SI) a una unidad de proceso (UP). La unidad de proceso actúa sobre una única secuencia de datos (SD) captados desde la unidad de memoria (UM).
- Una secuencia de instrucciones y múltiples secuencias de datos (SIMD): una única instrucción máquina controla paso a paso, la ejecución simultánea y sincronizada de un cierto número de elementos de proceso. Cada elemento de proceso tiene una memoria asociada, de forma que cada instrucción es ejecutada por cada procesador, con un conunto de datos diferentes. Ejemplos: procesadores vectoriales y matriciales.
  - Existe una sóla unidad de control, que proporciona una única secuencia de instrucciones a cada elemento del proceso (EP). Cada elemento de proceso puede tener su propia memoria dedicada (ML), o puede haber una memoria compartida.
  - Se requiere de un conjunto ampliado de instrucciones para permitir manejas operaciones vectoriales.
  - Se deben agregar instrucciones para transferir datos entre EPs.

- Múltiples secuencias de instrucciones y una secuencia de datos (MISD): se transmite una secuencia de datos a un conjunto de procesadores, cada uno de los cuales ejecuta una secuencia de instrucciones diferente.
    - Esta estructura nunca ha sido implementada.
- Múltiples secuencias de instrucciones y múltiples secuencias de datos (MIMD): un conjunto de procesadores ejecuta simultáneamente secuencias de instrucciones diferentes con conjuntos de datos diferentes. Ejemplos: SML, clusters y sistema NUMA.
    - Hay múltiples unidades de control, y cada una proporciona una secuencia de instrucciones separada a su propio elemento de proceso.
    - El MIMD puede ser un multiprocesador de memoria compartida (SMP o sistemas NUMA), o un multicomputador de memoria distribuida (clusters).

Organizaciones de computador alternativas:

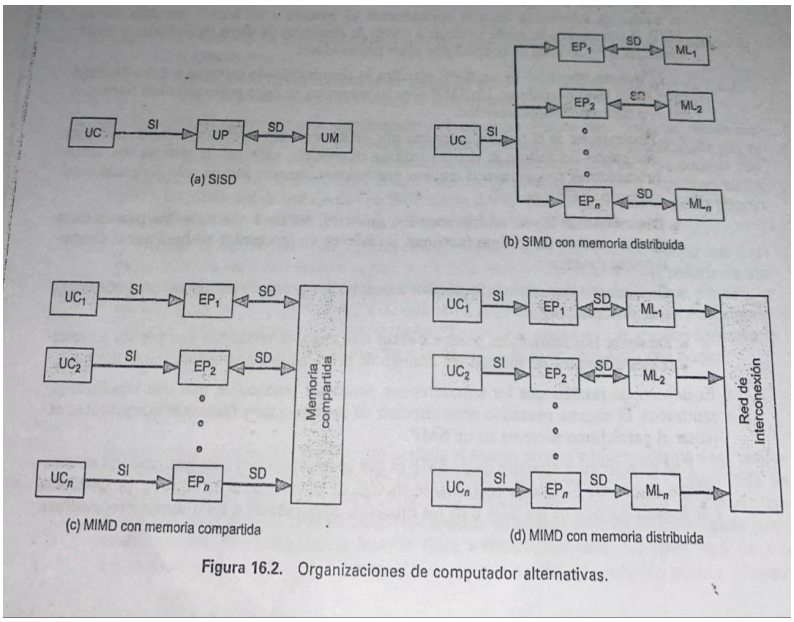


Figura 16.2. Organizaciones de computador alternativas.

9.2) MULTIPROCESADORES SIMÉTRICOS (SMP)

CARACTERÍSTICAS

El término SMP se refiere a la arquitectura hardware del computador, y también al comportamiento del sistema operativo que utiliza dicha arquitectura. Un SMP puede definirse como un computador con estas características:

1. Hay dos o más procesadores similares de capacidades comparables.
2. Estos procesadores comparten la memoria principal y las E/S, y están interconectados mediante un bus u otro tipo de sistema de interconexión, teniendo el mismo tiempo de acceso a memoria en todos los procesadores.
3. Todos los procesadores comparten los dispositivos de E/S, bien a través de los mismos canales, o bien mediante canales distintos que proporcionan caminos de acceso al mismo dispositivo.
4. Todos los procesadores pueden desempeñar las mismas funciones (simétrico).
5. El sistema está controlado por un sistema operativo integrado, que proporciona la interacción entre los procesadores y sus programas en los niveles de trabajo, tarea, fichero, y datos.
6. La interacción se puede producir a través de elementos de datos individuales, y puede existir un elevado nivel de cooperación entre procesadores.

VENTAJAS

Un SMP tiene las siguientes ventajas potenciales con respecto a una arquitectura monoprocesador:

- Prestaciones: un sistema con varios procesadores proporcionará mejores prestaciones que uno con un solo procesador del mismo tipo.
- Disponibilidad: en un multiprocesador simétrico, debido a que todos los procesadores pueden realizar las mismas funciones, un fallo en un procesador no hará que el computador se detenga.
- Crecimiento incremental: se pueden aumentar las prestaciones del sistema, añadiendo más procesadores.
- Escalado: los fabricantes pueden ofrecer una gama de productos con precios y prestaciones diferentes, en función del número de procesadores que configuran el sistema.

DESVENTAJAS

- La velocidad del sistema está limitada por el tiempo de ciclo del bus.
- Para contrarrestar esta desventaja se implementa una cache en cada procesador, pero además de ser caro, podrían producirse problemas de coherencia de cache.
- Se precisa añadir al sistema de memoria una buena cantidad de lógica, y podría representar un cuello de botella potencial para las prestaciones.

ORGANIZACIÓN

Hay dos o más procesadores idénticos (o muy similares) de capacidades comparables. Cada procesador es autónomo, incluyendo una unidad de control, una ALU, registros y, posiblemente, cache. Cada procesador tiene acceso a una memoria principal compartida y a los dispositivos de E/S, a través de un mecanismo de interconexión. Los procesadores pueden comunicarse entre sí a través de la memoria. La memoria, se organiza de forma que sean posibles los accesos simultáneos a bloques de memoria separados.

La organización de un sistema de multiprocesador puede clasificarse como:

- Bus de tiempo compartido: el bus consta de líneas de control, dirección y datos. Para facilitar las transferencias de DMA con los procesadores de E/S, se proporcionar los elementos para el direccionamiento (debe ser posible distinguir los módulos del bus), arbitraje (cualquier módulo de E/S puede funcionar temporalmente como maestro) y tiempo compartido (cuando un módulo controla el bus, este queda inhabilitado).
    - Ofrecen simplicidad para organizar el multiprocesador.
    - Generalmente es más sencillo de expandir el sistema conectando más procesadores al bus.
    - Otorgan fiabilidad, si ocurre un fallo en algún dispositivo conectado, el sistema no se vería afectado.
- Memoria multipuerto: permite el acceso directo e independiente a los módulos de memoria desde cada uno de los procesadores y los módulos de E/S.
- Unidad de control central: el controlador puede almacenar temporalmente peticiones, y realizar las funciones de arbitraje y temporización. Puede transmitir mensajes de estado y control entre los procesadores y alertar sobre cambios en las caches.

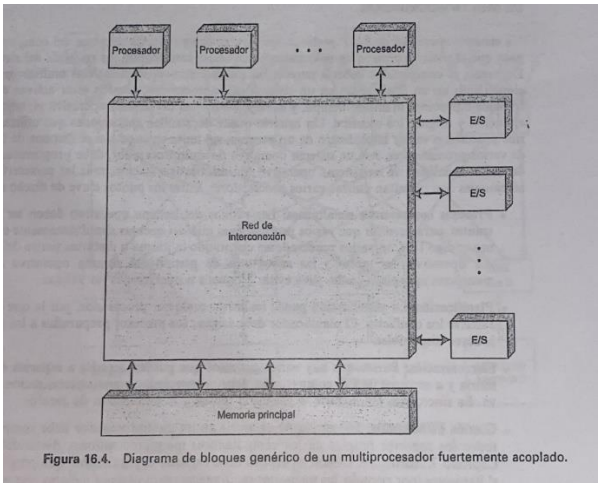


Figura 16.4. Diagrama de bloques genérico de un multiprocesador fuertemente acoplado.

9.3) CLUSTERS

DEFINICIÓN Y BENEFICIOS

Se puede definir un cluster como un grupo de computadores (nodos) completos interconectados, que trabajan conjuntamente como un único recurso de cómputo, creándose la ilusión de que se trata de una sola máquina. Como los espacios de direcciones son independientes es un sistema de memoria distribuida.

Se enumeran cuatro beneficios que pueden conseguir con un cluster:

- Escalabilidad absoluta: es posible configurar clusters grandes, que incluso superan las prestaciones de los computadores independientes más potentes.
- Escalabilidad incremental: un cluster se configura de forma que sea posible añadir nuevos sistemas al cluster en ampliaciones sucesivas.
- Alta disponibilidad: puesto que cada nodo del cluster es un computador autónomo, el fallo de uno de los nodos no significa la pérdida del servicio.
- Mejor relación precio/prestaciones: al utilizar elementos estandarizados, es posible configurar un cluster con mayor o igual potencia de cómputo que un computador independiente mayor, a mucho menos costo.

Todos los procesadores de un cluster pueden desempeñar las mismas funciones, además el sistema operativo está integrado, y proporciona la interacción entre los procesadores y sus programas. La manera en que se comunican entre procesos es en base a mensajes.

CONFIGURACIONES

Existen dos configuraciones conocidas, una sin disco compartido y otra con. En la primera, la interconexión se realiza mediante un enlace de alta velocidad, que puede utilizarse para intercambiar mensajes que coordinan la actividad del cluster. El enlace puede ser una LAN que se comparte con otros computadores no incluidos en el cluster, o puede tratarse de un medio de interconexión específico (unos nodos tendrán un enlace a una LAN o a una WAN).

La otra alternativa representada es un cluster con disco compartido. También existe un enlace entre los nodos. Además, existe un subsistema de disco (tipo RISC), que se conecta directamente a los computadores del cluster. Aparte, existen otros métodos de configuración con sus propios beneficios y limitaciones.

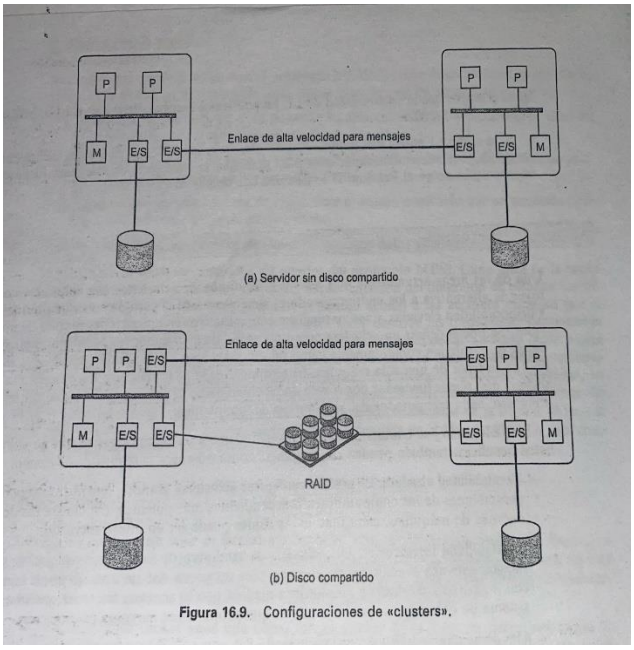


Figura 16.9. Configuraciones de «clusters».

CLUSTERS FRENTE SMP

SMP:

- Resulta más fácil de gestionar y configurar que un cluster.
- El SMP está cercano al modelo de computador de un solo procesador, para el que están disponibles casi todas las aplicaciones.
- El principal cambio que se necesita para pasar de un computador monoprocesador a un SMP se refiere al funcionamiento del planificador.
- El SMP necesita menos espacio físico y consume menos energía que un cluster comparable.
- Los SMP son plataformas estables y bien establecidas.

Clusters:

- Los clusters son superiores a los SMP en términos de escalabilidad absoluta e incremental.
- Además, también son superiores en términos de disponibilidad, puesto que todos los componentes del sistema pueden hacerse altamente redundantes.
- Cada nodo tiene su propia memoria principal privada, y las aplicaciones no ven la memoria global.

9.4) ACCESO NO UNIFORME A MEMORIA (NUMA)

TÉRMINOS RELACIONADOS A NUMA

- Acceso uniforme a memoria (UMA): todos los procesadores pueden acceder a toda la memoria principal utilizando instrucciones de carga y almacenamiento. El tiempo de acceso de un procesador a cualquier región de la memoria y el tiempo de acceso a memoria por parte de todos los procesadores es el mismo. Ejemplo: SMP.
- Acceso no uniforme a memoria (NUMA): todos los procesadores tienen acceso a todas las partes de memoria principal utilizando instrucciones de carga y almacenamiento. El tiempo de acceso a memoria de un procesador depende de la región a la que se acceda.
- NUMA con coherencia de cache (CC-NUMA): un computador NUMA en el que la coherencia de cache se mantiene en todas las caches de los tintos procesadores.

MOTIVACIÓN

Una alternativa para conseguir multiprocesamiento a gran escala mientras se mantienen las características SMP, son los computadores NUMA. El objetivo de un computador NUMA es mantener una memoria transparente desde cualquier parte del sistema, al tiempo que se permiten varios nodos de multiprocesador, cada uno con su propio bus u otro sistema de interconexión interna.

ORGANIZACIÓN CC-NUMA

En un sistema CC-NUMA, hay varios nodos independientes, cada uno de los cuales, es un SMP. Así, cada nodo contiene varios procesadores, cada uno con sus caches L1 y L2, más memoria principal. Existen un único espacio de memoria direccionable en el que, a cada posición, se asocia una única dirección válida para todo el sistema.

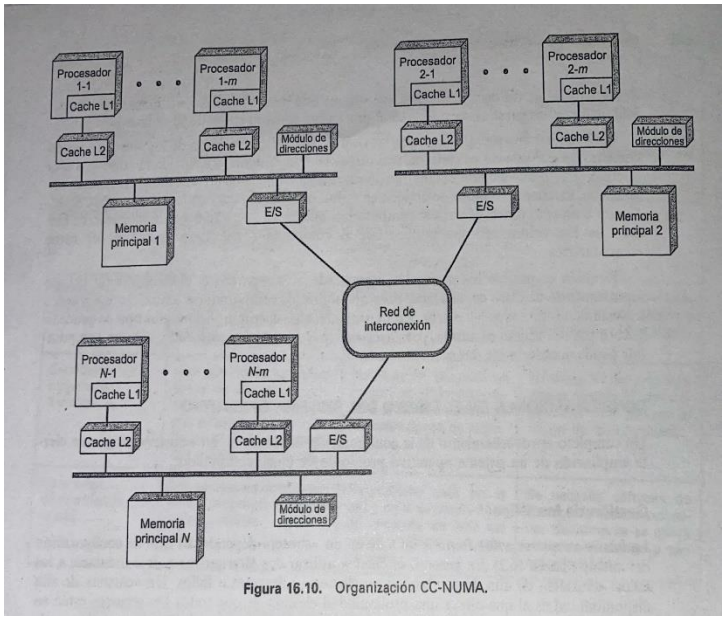


Figura 16.10. Organización CC-NUMA.

PROS Y CONTRAS DE UN COMPUTADOR NUMA

La principal ventaja de un computador CC-NUMA es que puede proporcionar un grado efectivo de prestaciones con mayores niveles de paralelismo que un SMP, sin que se precisen cambios importantes en el software. Con varios nodos NUMA, el tráfico del bus en cualquier nodo se limita a las peticiones que el bus puede manejar. Aunque, gracias a que hay dos niveles de cache, L1 y L2, los accesos a memoria se reducen, también esto se complementa gracias al principio de localidad espacial del procesador.

Esta alternativa también tiene desventajas. CC-NUMA no parece tan transparente como un SMP; se necesitan ciertos cambios en el software para adaptar el sistema operativo y las aplicaciones desde un SMP a un CC-NUMA. Un segundo aspecto a considerar es la disponibilidad.

9.5) PROCESAMIENTO MULTIHEBRA

PROCESO

Un proceso es un programa ejecutándose en un sistema que es propietario de recursos propios, y maneja un espacio de direcciones virtuales para almacenar la imagen de un proceso (code, data, stack). La comunicación entre procesos es a través de mecanismos específicos, y la conmutación entre estos requiere un cambio de los recursos asignados mediante mecanismos precisos y complejos.

HEBRA (THREAD) O HILO

Es una unidad de trabajo que:

- Tiene su propio contexto de procesador (PC y SP) y área de datos para su pila (stack).
- Comparte con otras hebras, código, variables globales, archivos abiertos por el proceso al que pertenecen, etc.
- Se ejecuta secuencialmente y es fácil de interrumpir (el procesador cambia a otra hebra si lo necesita).
- La conmutación de hebra (thread switch) es un cambio de control del procesador entre hebras de un mismo proceso.
- Un proceso está compuesto de muchas hebras o hilos.
    - Las instrucciones de diferentes hebras pueden ser paralelizadas.
    - Se pueden mejorar las prestaciones generales si se ejecutan más de una hebra en paralelo.
    - La idea es ejecutar varias hebras para aprovechar mejor los recursos libres del sistema.

Se pueden aumentar las prestaciones de un sistema en dos niveles:

- Nivel de instrucciones (IPL): aumentando la cantidad de instrucciones ejecutadas en paralelo, explotando el paralelismo de instrucciones secuenciales.
- Nivel de hebras (TPL): aumentando el nivel de hebras en paralelo, explotando el paralelismo entre instrucciones pertenecientes a distintos hilos de ejecución.

MULTITHREADING

El procesamiento de múltiples hebras (multithreading) consiste en ejecutar dos o más hebras en forma concurrente, y es necesario al menos un contador de programa para cada hebra y hardware para ejecución concurrente. Dicha ejecución puede ser de dos tipos:

- Multihebras explícitas: cuando son definidas en el programa, tanto a nivel de usuario (aplicaciones) como a nivel de sistema operativo (núcleo).
- Multihebras implícitas: cuando son definidas por el compilador (estático) o por hardware (dinámico).

El contexto de una hebra está formado por:

- El contador de programa (PC).
- Registros de datos, de direcciones (punteros), de estado, control, de segmentos.
- Datos propios en memoria (eventualmente).
- Datos en caché (eventualmente).

CAMBIO DE CONTEXTO EN MULTITHREADING

Cuando se cambia de una hebra a otra, se produce un cambio de contexto. Estos cambios pueden ser de dos tipos:

- Cambios de contexto tradicional.
  - Lo produce el sistema operativo mediante una interrupción, típicamente asociada a un timer.
  - La interrupción detiene la hebra en ejecución y el contexto de la hebra es salvado por el SO.
  - El SO recupera ese contexto y la hebra detenida anteriormente se reinicia mediante un retorno de interrupción. Esta hebra detenida no se entera que fue interrumpida.
- Cambios de contexto rápido (por hardware).
  - Se produce porque la hebra queda en espera por alguna razón (típicamente fallo de acceso a la caché).
  - Si el procesador tiene duplicados, al menos, el PC y los registros, entonces el procesador conmuta de hebra sin necesidad de salvar el contexto de la hebra detenida por el fallo.
  - Este cambio de contexto es mucho más rápido que el tradicional.

POLÍTICAS DE EJECUCIÓN MULTIHEBRA

Las hebras en ejecución usan recursos. Los recursos pueden ser asignados de dos formas:

- Estática: el particionado de recursos es fijo.
- Dinámica: el particionado de recursos cambia por algún mecanismo.

Existen cuatro políticas de ejecución multihebra:

1. CMP (Chip Multi Processor): el procesador dispone de dos o más núcleos. A cada núcleo se le asigna una hebra en forma fija.
   - La política de partición CMP es de tipo estática espacial.

2. FGMT (Fine Grained Multi Threading): el procesador conmuta de una hebra a otra en cada ciclo. La conmutación es rápida, una sola hebra por vez. El procesador dispone de dos o más contextos, para ejecutar dos o más hebras concurrentemente.
    - La política de partición es de tipo estática temporal.
    - Reduce las latencias en memoria, y evita y resuelve dependencia de datos.
    - Requiere que el compilador encuentre muchas hebras independientes.
3. CGMT (Coarse Grained Multi Threading): el procesador conmuta de una hebra a otra cuando una hebra se detiene por algún evento de gran latencia.
    - La política de partición es de tipo dinámica temporal.
    - Se deben replicar registros para conmutar rápido.
    - Se debe hacer una distribución equitativa de las hebras.
4. SMT (Simultaneous Multi Threading): hay varias hebras ejecutándose en varias unidades funcionales de un procesador superescalar (un solo núcleo).
    - La política de partición SMIT Es de tipo dinámica espacial.