***Clase 1 repaso conceptos de Organización***

Concepto de “programa”

Se puede considerar como antecedente más cercano de los Sistemas de Cómputo a las máquinas de calcular.

Las primitivas máquinas de calcular eran de tipo mecánico, capaces de realizar una sola operación (por ejemplo suma o resta) a la vez.

Si el cálculo a resolver requería un solo tipo de operación (por ejemplo sumas o restas repetitivas) solo era necesario ingresar los datos de cada operación.

Si, por el contrario, también se cambiaba la operación, se requería además modificar la máquina.

En ambos casos, era necesaria la intervención de una persona (“operador”) para hacer esas tareas.

Operador

Datos Maquina de calcular Resultados

Este modelo en el que un “operador” tiene que actuar (modificar) sobre la máquina (es decir la “Unidad de Cálculo”) de acuerdo a la tarea a ejecutar, se puede considerar como un “modelo de Programación en hardware”, porque cuando se cambia la tarea, se requiere cambiar el medio físico que la ejecuta.

***Clase 2 Interrupciones***

En condiciones “normales”, la CPU lee y ejecuta una instrucción a continuación de la otra de manera ininterrumpida (el “bucle interno del ciclo de instrucción”).

Una interrupción es un mecanismo que permite alterar ese 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 resolver (ejecutar) otra tarea (servicio de la interrupción).

Una vez completado el servicio de la interrupción, el procesador retoma la tarea suspendida, en el punto donde se detuvo (de una manera similar al llamado a subrutina).

Concepto de interrupción:

De acuerdo al concepto de interrupción, el procesador conmuta de una tarea en ejecución a otra a ejecutarse por efecto de la presencia de un evento (la interrupción). Por lo tanto se requiere de 3 acciones:

1) Detener (suspender) la tarea que está ejecutando el procesador (suspender, no terminar ni abortar).

2) Bifurcar (saltar) a otra tarea, asociada a la solicitud de interrupción (comúnmente conocida como servicio interrupción).

3) Restablecer la tarea suspendida en las condiciones en las que se encontraba en el momento en el que se la detuvo.

Que ocurre:

1.- La CPU recibe, mientras está ejecutando una tarea (el programa en ejecución), 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), típicamente en la Pila del sistema. Esto lo hace porque lo necesita para restaurar la tarea suspendida.

3.- La CPU busca, en un área de memoria definida, la dirección de comienzo (el “vector”) del servicio de la interrupción, y comienza a ejecutar dicho servicio.

4.- Cuando la CPU termina el servicio de la interrupción, tiene que retornar al programa interrumpido. Esto se hace mediante una instrucción especial de Retorno de interrupción (RTI).

5.- La ejecución de la instrucción RTI desapila exactamente lo apilado cuando atendió la interrupción (es decir, todo o parte del estado de la CPU). Como mínimo recupera el PC y el PSW. De esta manera retoma la tarea suspendida (el programa en ejecución) en el punto en que fue interrumpida.

Características de las interrupciones:

El origen de una interrupción es la ocurrencia de un evento que requiere la intervención de la CPU.

Existen 2 tipos de eventos:

**Interno:** es decir, debido a la ocurrencia de una situación dentro del Sistema de cómputo. Algunos eventos internos que pueden producir interrupción son: un error asociado a la ejecución de una instrucción, desbordamiento aritmético (“overflow”), división por cero, temporizados propios del sistema, fallo del hardware, error de paridad en la memoria, pérdida de energía.

**Externo:** asociado a operaciones de E/S con periféricos. Algunos eventos típicos que pueden generar interrupción son: finalización de una transferencia, error en la transferencia, dispositivo indisponible, etc.

Interrupciones múltiples

La necesidad de administrar eventos de distinto origen requiere, en la mayor parte de los casos, administrar varias interrupciones.

Dado el origen diverso de las interrupciones, hay algunas que son más importantes que otras.

En general, los procesadores son capaces de manejar varias interrupciones de distinta características y orígenes.

Prioridades en interrupciones múltiples

Las interrupciones más importantes deben tener mayor “prioridad” que las menos importantes.

Cuanto mayor sea su prioridad, mayor es la urgencia para ser atendida, incluso si hay una interrupción en curso.

Si las interrupciones son de igual prioridad, se procesan en el orden en que llegan.

Si las interrupciones tienen distinta prioridad, las interrupciones de mayor prioridad pueden interrumpir a las de menor prioridad. La inversa no vale.

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 interrupción quedará pendiente.

El procesador ejecutará el servicio de la interrupción atendida.

Al finalizar el servicio se habilitarán nuevamente las interrupciones.

La interrupción pendiente será atendida.

Procesamiento de interrupciones de distinta prioridad

Una interrupción de prioridad más alta puede interrumpir en cualquier momento a una interrupción de prioridad menor.

Cuando se ha gestionado la interrupción de prioridad más alta, el procesador vuelve a las interrupciones previas (de menor prioridad).

Terminadas todas las rutinas de gestión de interrupciones se retoma el programa del usuario.

Interrupciones enmascarables y no-enmascarables

Los sistemas de cómputo pueden responder de manera diferente a los distintos tipos de interrupciones, principalmente asociado a la prioridad que tienen.

Típicamente tienen 2 categorías de interrupciones.

**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 generalmente están asociadas a operaciones menos críticas, por ejemplo de E/S.

Interrupciones por hardware y por software

Desde el punto de vista de la forma en que se invocan, las interrupciones pueden ser 2 de tipos:

**Por Hardware:** son generadas por señales físicas asociadas a eventos externos o internos. De acuerdo al origen de la señal de interrupción, pueden ser:

**Externas**

**Internas**

**Por Software:** son producto de la ejecución de instrucciones con efecto similar a una interrupción por hardware.

Interrupciones por hardware 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.

Se consideran las “verdaderas” interrupciones porque son aleatorias en relación al proceso en ejecución (es decir, pueden ocurrir en “cualquier instante de tiempo”).

El sistema de cómputo tiene que ser capaz de manejar estos eventos externos “no planeados” ó “asincrónicos”.

Pueden o no estar relacionadas con el proceso en ejecución en ese momento.

Interrupciones por hardware internas

Son conocidas como Trap 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 tal motivo no son estrictamente aleatorias.

Algunos eventos que pueden producir un Trap son:

Condiciones excepcionales: overflow en ALU de punto flotante.

Fallas de programa: tratar de ejecutar una instrucción no definida.

Fallas de hardware: error de paridad de memoria.

Accesos no alineados o a zonas de memoria protegidos

Interrupciones por software

Son conocidas como Software interrupt.

Son instrucciones explícitas que tienen un efecto similar a una interrupción por hardware.

Como normalmente el SO administra los servicios de las Interrupciones, las Interrupciones por software permiten invocar los servicios del SO asociados a las Interrupciones (en otras palabras, son “llamadas” a funciones del SO).

El SO define los lugares donde se cargan los servicios de las Interrupciones. El usuario no conoce, a priori, esos lugares, pero usa los servicios invocándolos a través de las interrupciones que maneja el SO.

Hay sistemas que no permiten hacer una llamada directa a una función del SO, por estar en una zona reservada.

Si no se pudieran usar las interrupciones por software como mecanismo de invocación de esos servicios que administra el SO, cuando se necesita administrar una tarea por interrupción, se debería:

Opción 1: escribir el servicio (bastante complicado)

Opción 2: el programa que requiere el servicio debería buscar entre todas las llamadas a funciones del BIOS y SO la que necesita, y reemplazar en el código la dirección de esa función invocada (también muy complicado).

Gestión de las interrupciones

El uso de interrupciones requiere de la gestión ordenada de las siguientes acciones básicas:

1. Detectar el pedido de interrupción

2. Detener la tarea que se estaba ejecutando

3. Salvar el estado de la tarea que se estaba ejecutando

4. Obtener la dirección de comienzo del servicio de la interrupción y bifurcar a dicho servicio

5. Ejecutar el servicio de la interrupción

6. Retornar y restaurar el estado en que estaba la tarea interrumpida.

7. Continuar con la ejecución normal de la tarea interrumpida (en el punto en el que se detuvo)

Detección del pedido de interrupción

El primer paso en la administración de las interrupciones consiste en detectar el pedido de atención.

Para ejecutar esta operación, el procesador examina, en cada ciclo de instrucción, la presencia de interrupciones.

De acuerdo a lo visto previamente, el ciclo de instrucción podía interpretarse como un bucle de ejecución interna, infinito, compuesto por 2 fases: búsqueda de la instrucción y ejecución.

Para poder implementar la tarea de detección de interrupciones, en el ciclo de instrucción anterior se agrega, además de las fases de captura y ejecución, la fase de gestión de interrupciones.

La etapa de gestión de interrupciones en el ciclo de instrucción, determina la presencia o ausencia de pedido de interrupciones.

La presencia de un pedido de interrupción se manifiesta mediante una o más señales discretas (bits) comúnmente llamadas bandera (o “Flag”) que la CPU examina.

El estado de estos flags asociados a interrupciones están en algún registro (especial) de la CPU.

Dependiendo del estado 0 o 1 del flag (es decir, del pedido de interrupción) se tienen 2 posibles caminos.

Almacenamiento del proceso a ser interrumpido

Si no hay pedido pendiente (Flag inactivo) se inicia el ciclo de captación de la siguiente instrucción (proceso “normal” de ejecución).

Si hay algún pedido de interrupción pendiente, el procesador guarda en la pila del Sistema, el “estado del proceso”.

Existen 2 estrategias de guardado del estado del proceso:

Guardar solo la próxima instrucción a ejecutar y algún registro crítico (por ejemplo el registro de estado).

Guardar todos los registros del procesador.

El objetivo de esta operación es el de restablecer el estado del procesador al terminar el servicio de la interrupción.

Bifurcación al servicio de la interrupción

Obtiene la dirección donde comienza la rutina de la interrupción y carga el PC con este valor, bifurcando de esta manera, al servicio de la interrupción.

Existen varias técnicas para obtener la dirección donde comienza el servicio de la interrupción. En general se dispone de un área de memoria reservada, donde están estas direcciones. Tener en cuenta que son varias direcciones porque el procesador es capaz de atender varias interrupciones, y habrá una dirección por interrupción.

Esta área de memoria se llama área de vectores de interrupciones.

Detección del pedido de múltiples interrupciones

Dentro del proceso de gestión de las interrupciones, lo primero que tiene que hacer la CPU es detectar el pedido de interrupción. Cuando hay múltiples fuentes de interrupción, hay varias formas para identificar el origen del pedido. Los más comunes son:

Opción 1: 1 señal física de entrada a la CPU por cada Interrupción.

Se tiene 1 señal física de entrada a la CPU por cada interrupción, es decir que hay múltiples líneas de pedido de interrupción en la CPU. Dado que disponer de líneas en la CPU para interrupción es costoso, la cantidad de señales para ser usadas en interrupciones se acota normalmente a un número reducido (por ejemplo 3 o 4).

Cada dispositivo que puede provocar interrupción tiene una entrada física de interrupción conectada directamente a la CPU.

La implementación es bastante sencilla.

Restringida por la cantidad de líneas disponibles en la CPU.

Opción 2: 1 única señal física de entrada a la CPU para todas las interrupciones e identificación por software.

Opción 1: Se tiene 1 señal física de entrada a la CPU por cada interrupción, es decir que hay múltiples líneas de pedido de interrupción en la CPU. Dado que disponer de líneas en la CPU para interrupción es costoso, la cantidad de señales para ser usadas en interrupciones se acota normalmente a un número reducido (por ejemplo 3 o 4).

Cada dispositivo que puede provocar interrupción tiene una entrada física de interrupción conectada directamente a la CPU.

La implementación es bastante sencilla.

Restringida por la cantidad de líneas disponibles en la CPU.

Opción 3: 1 única señal física de entrada a la CPU para todas las interrupciones e identificación por hardware.

Hay 1 sola entrada física de pedido de interrupción a la que están conectados todos los dispositivos.

Para poder identificar la fuente de la solicitud, la CPU recibe a continuación, típicamente a través del bus de datos, un número que identifica la fuente de la interrupción (conocido como “vector de la interrupción”).

El vector es provisto por el periférico que generó el pedido, o por algún dispositivo que se ocupe de generar el número dependiendo de la interrupción a ser atendida.

Interrupciones vectorizadas con el PIC

La opción 3 se conoce como Interrupciones vectorizadas.

El escenario que se tiene es el siguiente:

El procesador tiene una única entrada de pedido de interrupciones.

Hay varios “productores” de interrupciones.

Un “dispositivo especial” administra las necesidades propias de la interrupción. En la familia Intel, este dispositivo se conoce como Controlador Programable de Interrupciones (‘PIC’). El PIC se encarga, entre otras cosas, de generar el vector, administrar prioridades, habilitar interrupciones, etc.

El PIC recibe los pedidos de interrupción, típicamente de periféricos que piden atención.

El PIC solicita atención a la CPU con la única señal de pedido de interrupción IntR (Interrupt request)

Cuando la CPU está lista para atender la interrupción, le avisa al PIC mediante la señal IntA (Interrupt acknowledge).

El PIC genera en el bus de datos el número de la interrupción (vector) a ser atendida. La CPU lee ese número y busca en la memoria el vector correspondiente al servicio de esa interrupción.

Dado que se hace por hardware, es mucho más rápido.

El PIC internamente tiene 3 registros principales.

El ISR que identifica la interrupción en servicio

El IRR que contiene los pedidos de interrupción provenientes de los periféricos.

El IMR que se usa para habilitar/deshabilitar los pedidos de interrupción. Esta funcionalidad se conoce como enmascaramiento de interrupciones.

Interrupciones del i8086

A modo de ejemplo se van a describir las características más importantes de las interrupciones en el 8086. ¬ El 8086 tiene 2 interrupciones por hardware: INTR y NMI

La interrupción NMI es del tipo no-enmascarable.

La interrupción INTR es del tipo enmascarable. La INTR tiene asociado un flag IF que determina si la INTR va a ser atendida o no (de ahí el concepto de interrupción enmascarable).

El Procesador dispone de una señal de reconocimiento de interrupción INTA.

El 8086 tiene 1 instrucción de interrupción por software: INT n (con n entre 0 y 255).

La instrucción de retorno de interrupción es IRET.

Dispone de 2 banderas relacionadas con las interrupciones:

Bandera IF para habilitar/deshabilitar la INTR (si IF=0 no se atiende la INTR)

Bandera TF para habilitar/deshabilitar el modo “Trace” (single-step). El modo “trace” o “single-step” es un mecanismo implementado mediante una interrupción (la interrupción de Trace), que habilita a la CPU ejecutar de a 1 instrucción por vez.

El esquema de manejo de las interrupciones es vectorizado.

El área de memoria donde están los vectores de las interrupciones está ubicado en las posiciones más bajas de memoria (normalmente tipo RAM) es decir: 0000-03FF (= 1024 bytes)

En total hay 256 vectores, identificados como 0 a 255 (o 00- FFH), correspondiente a 256 interrupciones distintas.

Cada vector ocupa 4 bytes: 2 para el registro de segmento de código (CS) y 2 para el Contador de Programa (IP).

Como ya se mencionó anteriormente, el procesador usado en el simulador MSX88 es una versión simplificada del 8086, por lo que presenta algunas diferencias respecto de la CPU verdadera.

Interrupciones por hardware

Línea INT (interrupción enmascarable) y línea de reconocimiento de interrupción INTA

Línea NMI (interrupción no-enmascarable)

Interrupciones por software

Instrucción: INT xx

Retorno de interrupción: IRET

Los vectores de las interrupciones están en la parte más baja de la memoria.

Cada entrada (o vector) es una palabra doble (4 bytes), que contiene la dirección del procedimiento que brinda el servicio. La parte alta del vector es 0.

Ej: 0000yyyy, donde yyyy es la dirección lógica/física.

El MSX88 tiene preasignados los siguientes vectores:

Tipo 0 – finaliza ejecución de programa

Tipo 3 – punto de parada para depuración/seguimiento

Tipo 6 – lectura de entrada std. Requiere el uso de BX.

Tipo 7 – escritura de salida std. Requiere BX y AL.

Los registros internos del PIC se sitúan a partir de la dirección 20H (del espacio de direcciones de E/S).

Son accedidos con operaciones lectura y escritura en el espacio de E/S, es decir, mediante las instrucciones IN y OUT.

Las Interrupciones de hardware preasignadas en el MSX88 son:

INT0 – tecla F10 - Produce una interrupción cada vez que se presiona F10

INT1 – Timer - Conectada a la salida del Timer

INT2 – Handshake - Conectada a una salida para Handshake

INT3 – DMA - Conectada a la salida del puerto a Impresora ¬

INT4 a INT7 no usadas

***Clase 3 Subsistema E/S***

SUBSISTEMA E/S

El subsistema de E/S comprende los dispositivos que están conectados al bus del sistema y proveen los servicios de transferencia de datos con los Periféricos. Debido a la gran variedad de Periféricos con los que se requiere intercambiar información, el subsistema de E/S tiene que ser lo suficientemente flexible para permitir:

Trasmisión de diferentes cantidades de datos

Rango de velocidades de transmisión muy amplio.

Diferentes formatos de dato y tamaño de palabra. En general, todos los Periféricos son más lentos que la CPU y la Memoria. Los dispositivos que forman parte del subsistema de E/S permiten descongestionar el trabajo de la CPU.

Dispositivos periféricos

Existe una variedad muy grande de equipos periféricos. Algunos de los tipos más comunes son:

Comunicación hombre-máquina: monitor/pantalla, mouse, teclado

Almacenamiento: disco duro, CD, DVD

Impresión: impresora, escáner

Comunicación con dispositivos remotos: modem, placa de red

Multimedia: micrófono, parlantes

Automatización y control: sensores, alarmas, adquisición de datos

Módulo de interfaz de E/S

Son los dispositivos más sencillos para implementar las transferencias de E/S con periféricos.

Los puertos de E/S conectan la interfaz entre el procesador/memoria y un periférico.

Son, por lo común, administrado por el SO a través de drivers específicos.

Módulo de E/S

La conexión con el Periférico provee 2 tipos de informaciones:

1.- Datos: Información útil a transferir

2.- Control y Estado: información que permite realizar la transferencia (en lo posible libre de errores). Por ejemplo: sentido de la transferencia (de entrada, de salida), operación (de lectura, de escritura), estado del periférico (listo, no-listo, en falla).

El Módulo de E/S debe ejecutar 2 tipos de comunicaciones:

1.- Hacia el Periférico (comunicaciones externas con la periferia):

• Transferir datos con el periférico (incluye adaptación eléctrica).

• Controlar y temporizar uno o más dispositivos externos.

• Almacenar temporalmente datos (“buffer”).

• Detectar errores.

2.- Hacia el Bus (internas con CPU y Memoria):

• Interpretar las órdenes que recibe de CPU y transmitirlas al periférico.

• Transferir datos con la CPU (registros) y Memoria.

• Informar a la CPU del estado del periférico.

Un periférico tiene 2 bloques funcionales que manejan la comunicación entre el periférico y el módulo de E/S:

1.- Hacia el exterior del periférico:

• Sección de manipulación de datos (buffer/trasductor). Almacena y convierte los datos a intercambiar con el periférico

• Sección de control y estado (Lógica de control): recibe y genera las señales de control y estado del periférico

2.- Hacia el Módulo de E/S :

• Sección de manipulación de datos

• Sección de manipulación de señales de control y de estado

Esquema detallado de un Módulo de E/S

Hacia el sistema los recursos que son “visibles” a la CPU (y al programador) son básicamente registros. Existen 2 tipos de registros:

Registros de datos: que contienen la información útil recibida desde, o transmitida a, el periférico.

Registros de control y estado: que controlan las características de la transferencia, y almacenan señales de estado de la comunicación (falla, no listo, etc.).

Tipos de Puerto de E/S básicos

Existen 2 tipos básicos: serie y paralelo.

Tipos de Puerto de E/S básicos

Puerto paralelo: hay varias líneas de datos (n) que transfieren n bits simultáneamente entre el puerto de E/S y el periférico. ¬

Ejemplos: impresora paralelo, scanner

Se requiere disponer de una conexión mediante un cable que incluya al menos los n bits de datos, lo que lo hace un método bastante costoso.

Puerto serie: hay 1 línea de dato para la transferencia entre el puerto de E/S y el periférico.

Ejemplos: impresora serie, red Ethernet, mouse, teclado, etc.

Se requiere disponer de una conexión mediante un cable sencillo. El costo es mucho menor.

Los datos deben serializarse (transmitirse de 1 bit por vez, uno a continuación del otro) lo que en teoría sería mucho más lento que transmitir de a n bits simultáneamente, como lo hace el puerto paralelo.

CPU y E/S

Subsistema de E/S

Las Puertas de E/S son un tipo particular de dispositivo de E/S. Hay otros dispositivos más complejos que proveen de otras prestaciones, además de la tarea básica de implementar la transferencia de los datos, por ejemplo:

Ocultar las propiedades particulares del dispositivo periférico a la CPU: temporizados, formatos, electromecanismos, etc.

Manejar múltiples dispositivos simultáneamente.

Controlar varias funciones del dispositivo

Registros de un Puerto 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.

Dentro de Módulo de E/S hay 2 tipos de registros:

de DATOS: interviene en la transferencia de entrada o de salida del dato a intercambiar entre el Sistema de cómputo y el periférico.

de CONTROL: registros que controlan y registran el funcionamiento del módulo, la transferencia, y el periférico.

de Datos: la transferencia de un dato entre el Sistema de cómputo y el periférico consiste básicamente en:

Operación de entrada: lectura de un registro de dato de entrada (es decir un registro escrito por el periférico y leído por la CPU)

Operación de salida: escritura de un registro de dato de salida (es decir, un registro escrito por la CPU y leído por el periférico).

de Control y Estado: la supervisión y/o control de la transferencia requiere de:

Control: adecuar la configuración del módulo para ajustar formatos, sincronizaciones, etc.

Estado: registrar el estado operativo del módulo y del periférico.

Acceso al Subsistema de E/S

Desde el punto de vista de la CPU, el Subsistema de E/S está compuesto por un conjunto de registros a los que accede para una operación de entrada o de salida.

Existen 2 técnicas de acceso a estos registros:

Espacio de E/S compartido con memoria (memorymapped)

Espacio de E/S separada de la memoria

Espacio de E/S compartido con memoria (memory-mapped)

En está técnica los registros de los dispositivos de E/S y memoria comparten un único espacio de direcciones.

Los registros de la 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. Ej: Mov Reg\_dato, AL

Donde Reg\_dato es la dirección de un registro de salida del módulo de E/S (dirección idéntica a la correspondiente a una posición de memoria).

Espacio de E/S separado de memoria (modelo Intel)

En está técnica los registros de los dispositivos de E/S y la memoria están en diferentes espacios de direcciones.

Dado que el bus de direcciones es compartido por la memoria y el subsistema de E/S, se requieren señales de control adicionales para identificar a donde está accediendo la CPU: a la memoria, o a la E/S.

Hay instrucciones específicas de E/S, distintas de las instrucciones de acceso a la memoria.

El procesador dispone de instrucciones específicas para acceder a los registros que están en el subsistema de E/S.

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 las instrucciones en el bus de control se identifica el acceso a la Memoria. Ejemplos:

1) Entrada: IN dest, fuente Donde dest es AL o AX (8 o 16 bits) y fuente un número de 8 bits sin signo (0 y 255) o DX (número entre 0 y 65535).

2) Salida: OUT dest, fuente Donde fuente es AL o AX (8 o 16 bits) y dest un número de 8 bits sin signo (0 y 255) o DX (número entre 0 y 65535).

Gestión de la transferencia

Desde el punto de vista de la gestión para transferir datos entre el Sistema de cómputo y el periférico, existen 3 estrategias básicas de implementación:

E/S Programada y espera de respuesta

E/S Programada y administrada por interrupción ¬ E/S con acceso directo a memoria (DMA)

Las 2 primeras opciones requieren intervención directa de la CPU, es decir, que la CPU participa en la transferencia de todos los datos (byte o word) a transferir.

E/S Programada y espera de respuesta

La CPU interviene directamente en la transferencia de cada unidad de información (byte, word) con el módulo.

Es decir que la CPU tiene control casi directo sobre la operación de E/S. Entre otras acciones hace:

Comprueba el estado del dispositivo

Envía los comandos requeridos (por ejemplo de lectura, escritura)

Realiza la transferencia de todos los datos (de a uno)

En cada dato que es transferido, la CPU espera que el módulo E/S termine la operación, típicamente que el periférico “acepte” el dato.

Durante la espera, la CPU permanece ociosa (no deseable)

La secuencia de acciones que ejecuta la CPU son:

1. La CPU verifica el estado de periférico (preparado/no-preparado) leyendo un registro del módulo de interfaz.

2. Examina el estado del periférico chequeando el bit (o bits) que identifican dicho estado.

3. Si el dispositivo no está listo (por ejemplo bit=0), la CPU vuelve al paso 1. Este lazo significa que la CPU “espera” hasta que el periférico se pone en “preparado”, es decir, listo para la transferencia.

4. Cuando el dispositivo está listo, la CPU transfiere 1 dato hacia o desde el módulo de interfaz.

5. Si hay más datos que transferir vuelve al paso 1.

6.- Si se completó la transferencia, termina el servicio de E/S.

E/S Programada y administrada por interrupción

La CPU interviene directamente en la transferencia de cada unidad de información (byte, word) con el módulo.

Cada vez que el módulo está listo (o completó una transferencia) avisa a la CPU con un pedido de interrupción.

Ahora la CPU no tiene que ejecutar el lazo de comprobación del estado del módulo (pasos 1, 2 y 3 de la figura anterior). Solo inicia la transferencia al recibir el pedido de interrupción del periférico.

Durante el tiempo que el periférico no está listo, la CPU no tiene que esperar, puede seguir ejecutando otra tarea.

A continuación se describen la secuencia de acciones para una transferencia de entrada de 1 dato (por ejemplo, 1 caracter) administrado por interrupción.

La CPU inicia la operación de lectura (entrada) enviando una orden de lectura (READ) al módulo de E/S.

El módulo de E/S solicita el dato al periférico.

El periférico busca el dato, mientras la CPU continúa con sus tareas.

Cuando el módulo E/S tiene el dato enviado por el periférico emite un pedido de interrupción a la CPU.

La CPU detecta el pedido, interrumpe el proceso, y bifurca al servicio de la interrupción.

Durante la interrupción, la CPU lee el dato desde el módulo de E/S.

La CPU retorna a la tarea interrumpida.

Interrupciones en operaciones de E/S

Cuando hay varios dispositivos periféricos, con esta forma de administración de las transferencias de E/S se requiere poder identificar la fuente de interrupción. Como se vió durante el análisis de Interrupciones, existen varias estrategias distintas para identificar la fuente de la interrupción:

1) Diferentes líneas de interrupción

2) Una sola línea de interrupción y encuesta por software

3) Una sola línea de interrupción con conexión en cadena (daisy chain) tipo “hard poll” (encuesta por hardware)

4) Una sola línea de interrupción y vectorizado

1) Diferentes líneas de interrupción

Se dispone de una línea de interrupción por cada dispositivo.

Es sencilla de implementar.

Hay una limitación en la cantidad de dispositivos a conectar debido a la cantidad restringida de señales de interrupción que puede manejar la CPU.

2) Una sola línea de interrupción y encuesta por software

Se dispone de 1 sola línea de interrupción para todos los dispositivos.

Cuando ocurre el pedido de interrupción, la CPU tiene que consultar a cada dispositivo (es decir, a cada módulo de E/S) para determinar quien fue el demandante.

Este esquema de encuesta por software (también conocido como “polling”) puede resultar sumamente lento.

3) Una sola línea de interrupción con conexión en cadena

Se dispone de 1 sola línea de interrupción INTR para todos los dispositivos.

La línea de respuesta de la CPU INTA (reconocimiento de interrupción) se conecta encadenadamente a todos los módulos (conexión tipo “margarita” o “Daisy chain”).

Una vez enviada la confirmación de parte de la CPU, el módulo que está más adelante (más próximo a la CPU) en la conexión Daisy chain responderá colocando un vector (palabra), en el bus, que lo identifica.

Si hay otros pedidos de interrupción más abajo del que respondió, deberán esperar la terminación del servicio del que respondió.

4) Una sola línea de int. y vectorizado

Se dispone de 1 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 tienen un orden de prioridad, las líneas con más prioridad pueden interrumpir a las líneas con menor prioridad.

Si existe un maestro del bus, solo él puede interrumpir.

En general:

Las operaciones de E/S administradas por interrupción son más eficientes que las programadas con espera.

Ambas técnicas requieren la intervención directa de la CPU.

Al tener que intervenir la CPU en las transferencias de los datos se presentan 2 problemas:

La velocidad de transferencia depende de la capacidad de la CPU de atender estas tareas. Aunque la CPU es muy rápida, puede ser que en determinadas circunstancias no sea capaz de administrar varias transferencias simultáneamente.

La CPU puede permanecer ocupada mucho tiempo durante la operación, sin poder hacer otras tareas.

Además, si el volumen de datos a transferir es grande, el tiempo de ocupación de la CPU crece también.

Para analizar el comportamiento de la CPU en transferencias de E/S, vamos a analizar la respuesta de la CPU en 2 casos:

1) Transferencia hacia un periférico lento (impresora de 20 páginas/minuto)

2) Transferencia hacia un periférico rápido (disco de 10Mb/s)

Para los 2 casos, se analizará la respuesta del sistema para transferencias de E/S manejadas por la CPU con espera de respuesta y administrada por interrupción.

Sobre la CPU

Se tiene un procesador con un reloj de 200 MHz (período del reloj = 5 ns).

El procesador, en promedio, necesita 2 ciclos de reloj por instrucción. Este parámetro se conoce como CPI. En este caso CPI=2

Es decir, que una instrucción tarda, en promedio: Ciclo de instrucción = 2 x 5 ns = 10 ns = 10-8

El procesador tiene, por lo tanto, una capacidad de procesamiento de: N° de instrucciones por segundo = 1 / tiempo de ciclo de instrucción N° instrucciones por segundo = 1/10-8 N° Instrucciones por segundo = 100 millones de instrucciones por segundo (100 Mips)

Caso 1: Transferencia a una Impresora

Se tiene que imprimir (operación de salida) un archivo de 10 Kbytes en una impresora láser de 20 páginas por minuto.

Considerando que: 1 página tiene (estimado) 3.000 letras (es decir, 3000 caracteres) y: 1 carácter = 1 byte

La impresora imprime: 20 ppm = 20 pag/min x 3000 car/pag= 60.000 caracteres por minuto Es decir, que la velocidad de transferencia de la impresora es de : Vt = 60000 car./min = 60000 car./60 seg. = 1000 car./s = 1 Kbyte/s

Caso 1: Transferencia a una impresora a) E/S con espera de respuesta

La CPU entra en un bucle y envía un nuevo byte cada vez que la impresora está preparada para recibirlo

Si la impresora tarda 1 seg en imprimir 1 Kbyte (ya que la tasa de transferencia es de 1 kbyte/seg), necesitará: Tiempo total de transferencia = 10 seg para 10 Kbytes.

Conclusión: La CPU está ocupada con la operación de E/S durante 10 seg.

Observación: Ya que la velocidad de la CPU es de 100MIPs, en ese tiempo la CPU podría haber ejecutado: N°instrucciones en 10 seg. = 1000 millones de instrucciones

Caso 1: Transferencia a una impresora b) E/S administrada con interrupciones

La impresora genera una interrupción cada vez que está preparada para recibir un nuevo byte.

Si la gestión de la interrupción (que se llama ATI) requiere 10 instrucciones (entre las que se incluyen salvar contexto, comprobar estado, transferir byte, restaurar contexto, y retornar), entonces: Para transferir 10 Kbytes se requiere ejecutar 10.000 veces la ATI.

El tiempo que dura el servicio de la transferencia es, aproximadamente, el requerido por las 10.000 ATI. Recordando que 1 instrucción tiene un tiempo de ejecución de 10-8 seg Tiempo total de transferencia = 10.000 ATI = 10.000 x 10 instr = =10.000 x 10 x 10-8 seg = 100.000 x 10-8 seg = 10-3 seg = 0,001seg

Conclusión: La CPU está ocupada con la operación de E/S durante 0,001 seg.

Conclusiones Caso 1 - Transferencia a una impresora:

De los resultados anteriores en el uso de la CPU para el caso de una transferencia con un dispositivo relativamente lento, usando las 2 técnicas de E/S, con espera y administrada por interrupción, se pueden sacar las siguientes conclusiones.

La transferencia dura 10 segundos, porque es el tiempo que tarda la impresora en imprimir los 10.000 caracteres.

El tiempo de uso de la CPU en ambos casos es:

Caso a) con espera: 10 seg. La CPU está los 10 segundos ocupada administrando la transferencia.

Caso b) administrada por interrupción: 0,001 seg De los 10 segundos que dura la transferencia, la CPU le ocupa solo 0,001 segundo en administrarla.

Es decir, que la E/S por interrupciones reduce en 10.000 veces el tiempo que la CPU está ocupada gestionando la impresora, y por lo tanto, es mucho más eficiente.

Caso 2: Transferencia a un disco

Se tiene que transferir un archivo de memoria a disco de 10 Mbytes

Considerando que: El disco posee una velocidad de transferencia de 10 MB/s (1 byte cada 10-7 seg ó 100 nanoseg) significa que: Tiempo total de transferencia= 1 segundo.

a) E/S con espera de respuesta

La CPU entra en un bucle y envía un nuevo byte cada vez que el disco está preparado para recibirlo

Si el disco transfiere 10 Mbytes/seg, necesita 1 seg para recibir un archivo de 10 Mbytes. Tiempo total de transferencia = 1 seg para 10 Mbytes.

Conclusión: La CPU está ocupada con la operación de E/S durante 1 seg.

Observación: Ya que la velocidad de la CPU es de 100MIPs, en ese tiempo la CPU podría haber ejecutado: N°instrucciones en 1 seg. = 100 millones de instrucciones

b) E/S administrada con interrupciones

El disco genera una interrupción cada vez que está preparado para recibir un nuevo byte

Si la gestión de la interrupción (que se llama ATI) requiere 10 instrucciones (entre las que se incluyen salvar contexto, comprobar estado, transferir byte, restaurar contexto, y retornar), entonces: Para transferir 10 Mbytes tenemos que ejecutar 107 veces la ATI

El tiempo que dura el servicio de la transferencia es, aproximadamente, el requerido por las 10.000.000 ATI. Recordando que 1 instrucción tiene un tiempo de ejecución de 10-8 seg Tiempo total de transferencia = 10.000.000 ATI = 107 x 10 instr = = 107 x 10 x 10-8 seg = 1seg

Conclusión: La CPU está ocupada con la operación de E/S durante 1 seg.

Conclusiones Caso 2 – Transferencia a un disco:

De los resultados anteriores en el uso de la CPU para el caso de una transferencia con un dispositivo relativamente rápido, usando las 2 técnicas de E/S, con espera y administrada por interrupción, se pueden sacar las siguientes conclusiones.

La transferencia dura 1 segundo, porque es el tiempo que tarda el disco en transferir los 10Mbytes.

El tiempo de uso de la CPU en ambos casos es:

Caso a) con espera: 1 seg. La CPU está 1 segundo ocupada administrando la transferencia.

Caso b) administrada por interrupción: 1 seg La CPU está 1 segundo ocupada administrando la transferencia.

Es decir, que no hay diferencia entre las 2 técnicas. En ambas la CPU está ocupada 1 segundo el 100% del tiempo.

Si la velocidad del dispositivo fuera mayor, la CPU no podría hacerla.

Acceso directo a memoria (DMA)

El Acceso Directo a Memoria (DMA) es una técnica de transferencia de datos entre periférico y Memoria sin intervención directa de la CPU.

Comúnmente llevada a cabo por un “Controlador de DMA” (o DMAC) específico encargado de llevar a cabo la transferencia.

Físicamente está ubicado como se muestra a continuación.

Dado que la transferencia por DMA requiere el uso del Bus, tanto el DMAC como la CPU pueden tomarlo. El DMAC y la CPU “compiten” por el uso del Bus.

Cuando el DMAC toma el Bus, actúa como “master” durante la transferencia por DMA, y debe ser capaz de:

Solicitar el uso del bus mediante las señales y la lógica de arbitraje necesarias

Especificar la dirección de memoria sobre la que se realiza la transferencia

Generar las señales de control del bus

Especificar el tipo de operación (lectura/escritura)

Generar las señales de sincronización de la transferencia

Cuando la CPU entrega el bus al DMAC (8237 DMA chip en la figura), se desconecta lógicamente del mismo, y es el DMAC el que toma el control del bus.

El proceso de transferencia requiere realizar una serie de acciones o fases relativamente complejas. Las principales son:

1.- Fase de Inicialización

2.- Fase de ejecución de la transferencia

3.- Fase de finalización y análisis de la transferencia

1.- Fase de inicialización

En la fase de inicialización la CPU debe configurar el módulo de E/S y el DMAC con los parámetros de la transferencia.

Inicialización interfaz de E/S: ¬ Tipo de transferencia (lectura/escritura)

Configuración del periférico

Otra información de control para el periférico (por ejemplo si es un disco se especifica el número de pista, sector, etc.)

Inicialización DMAC

Nº de bytes o palabras a transferir

Tipo de transferencia (lectura/escritura)

Dirección de memoria inicial para la transferencia

Otra información 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 el pedido del periférico, pide el control del bus mediante alguna señal especial a la CPU. La CPU típicamente dispone de algunas señales destinadas a implementar las transferencias por DMA.

Cuando reconoce el pedido de DMA, la CPU entrega (libera) el bus y se “desconecta” (ya no controla el bus) lógicamente del mismo.

La CPU avisa al DMAC que liberó el bus mediante otra señal especial.

Al liberar la CPU el bus, el DMAC toma el control del bus y ejecuta la transferencia hasta terminarla.

El DMAC avisa al periférico que puede iniciar la transferencia.

El periférico comienza a transferir los datos, a través de bus, con la memoria, de a uno por vez.

La transferencia implica que: ¬Bus master: DMAC + Periférico - Bus slave: Memoria

Después de la transferencia de cada palabra se actualizan los registros del DMAC:

Nº de bytes faltantes (o cuenta de los que se transfirieron)

Próxima dirección de memoria a donde guardar el dato (anterior o posterior de la corriente)

Cuando el número de bytes faltantes es igual a 0 significa que transfirió todos los datos y terminó la transferencia.

3.- Finalización de la transferencia

Una vez que termina la fase 2, el DMAC libera el bus y le avisa a la CPU por medio de una señal física.

La CPU retoma el control del bus.

El DMAC suele activar, además, una señal de interrupción para indicar a la CPU la finalización de la operación de E/S solicitada.

La CPU, mediante la interrupción, verifica el resultado de la transferencia vía los registros internos del DMAC. Algunos resultados a verificar son:

Transferencia OK/fallida?

Errores? Tipo de errores?

Estado periférico

Ventajas e inconvenientes del DMA

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.

Sin embargo, no necesariamente la CPU necesite todo el tiempo el bus porque:

En el caso de Computadoras con memoria caché:

La mayor parte del tiempo, la CPU lee instrucciones de la cache, por lo que no necesita usar el bus de memoria.

El DMAC puede aprovechar estos intervalos en los que la CPU está leyendo instrucciones de la cache (y por tanto no usa el bus de memoria) para realizar las transferencias.

En el caso de computadores sin cache

El procesador no utiliza el bus en todas las fases de la ejecución de una instrucción.

El DMAC puede aprovechar las fases de ejecución de una instrucción en las que la CPU no utiliza el bus.

Técnicas de transferencia por DMA

Hay varias formas distintas de implementar las transferencias por DMA. Vamos a ver 2:

Por ráfagas (burst)

Por robo de ciclo (cycle-stealing)

Método de transferencia de DMA por ráfaga

Es la vista hasta ahora. 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.

VENTAJAS: La transferencia se realiza de forma muy rápida, limitada por la velocidad del periférico.

DESVENTAJAS: Durante el tiempo que dura la transferencia la CPU no puede utilizar el bus con memoria, lo que puede degradar el rendimiento del sistema.

Método de transferencia de DMA por robo de ciclo

El DMAC solicita el control del bus a la CPU.

Cuando la CPU concede el bus al DMAC, se realiza la transferencia de una única palabra y después el DMAC libera el bus.

El DMAC solicita el control del bus tantas veces como sea necesario hasta finalizar la transferencia del bloque completo. El uso del bus se reparte entre la CPU y el DMAC.

VENTAJAS: No se degrada tanto el rendimiento del sistema y de la CPU.

DESVENTAJAS: La transferencia puede tardar un poco más de tiempo.

Notar que dado que la toma y liberación del bus por parte de la CPU no es una interrupción, es decir que el procesador no debe guardar el contexto (es decir, no está interrumpiendo su tarea).

Si bien el trabajo de la CPU es más lento (que si no estuviera presente la transferencia por DMA), no será tanto como si ella tuviera que estar desconectada del bus todo el tiempo.

En general, para transferencia de E/S de múltiples palabras, la técnica por robo de ciclo es la más eficiente, ya que permite implementar la transferencia por DMA al mismo tiempo que la CPU continúa trabajando en su tarea.

Canales de E/S

Las transferencias de E/S se pueden dividir, en función de la capacidad para interactuar con los periféricos, en varios niveles:

Nivel 1: CPU + Módulo de Interfaz de E/S o controlador: la CPU controla directamente los periféricos e interfaz, y administra la transferencia por programa (con espera).

Nivel 2: CPU + Módulo de Interfaz E/S o controlador con interrupción: la CPU controla directamente los periféricos y administra la transferencia con programa e interrupciones.

Nivel 3: DMA (DMAC + Módulo de E/S): la CPU no interviene directamente, solo prepara y supervisa la transferencia.

Nivel 4: Canal de E/S básico (Procesador básico + módulo de E/S): la CPU interviene mínimamente.

Nivel 5: Canal de E/S inteligente (Procesador inteligente + módulo de E/S): la CPU no interviene, excepto situaciones especiales.

En el nivel más alto de la escala de transferencias de E/S están los canales de E/S, que representan una extensión al concepto de DMA.

Los canales de E/S 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, 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 solamente interviene para iniciar la transferencia, y dar la orden de ejecutar el programa de E/S que está en memoria. El programa de servicio de E/S especifica dispositivos, áreas de memoria a usar, prioridades y acciones ante errores.

Hay 2 tipos básicos de canales de E/S:

Selector

Multiplexor

Canal selector de E/S

El canal controla varios dispositivos de alta velocidad, de a uno por vez.

El canal selecciona un dispositivo y efectúa la transferencia sobre el dispositivo seleccionado.

Cada dispositivo tiene asociado un controlador o módulo de E/S que lo maneja.

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 1 dispositivo a la vez.

Canal Multiplexor de E/S

El canal controla varios dispositivos de alta velocidad, incluso simultáneamente.

El canal mutiplexa la atención entre los dispositivos seleccionados. El multiplexado puede ser:

Multiplexor de bytes: acepta y transmite de a caracteres.

Multiplexor de bloques: intercala bloques de datos desde distintos dispositivos.

Puede atender varios dispositivos a la vez.

***Clase 4 Segmentación del cauce***

Introducción al nanoMIPS

➢ Para algunos de los temas a tratar a continuación, es mejor usar un modelo de procesador más sencillo del que se usó hasta ahora (8086 / MSX88).

➢ Se va a usar un modelo de procesador simplificado basado en el procesador MIPS, que es un procesador comercial tipo RISC.

El nanoMIPS tiene estas características principales:

➢ Palabra de memoria de 32 bits

➢ Espacio de direcciones de 64 bits (virtual)

➢ 32 (31 efectivos) Registros de propósito general de 64 bits (R0..R31).

➢ El registro R0 es 0 (es decir, si referenciamos al registro R0 nos devuelve 0).

➢ Como son 32 registros, se requieren 5 bits para identificarlo.

Formato de instrucción

El nanoMIPS dispone de 3 tipos de Instrucciones:

➢ Tipo I: instrucciones de acceso a memoria, únicamente las de carga y almacenamiento (“LOAD” y “STORE”).

➢ Tipo R: instrucciones aritméticas y lógicas, únicamente registro a registro (no admite operaciones con memoria).

➢ Tipo J: instrucciones de salto

➢ El acceso a la memoria se hace únicamente con las instrucciones tipo I. Por esta razón a este tipo de máquinas se las identifica como máquina tipo “LOAD/STORE”.

➢ El formato es muy regular, porque:

➢ Las instrucciones tienen todas el mismo tamaño (32 bits).

➢ Los campos con referencias son siempre los mismos.

Modos de direccionamiento

El nanoMIPS dispone formalmente de 2 modos de direccionamiento, pero si se consideran 2 casos especiales, se obtienen 2 modos más (4 en total).

➢ Inmediato: dato de 16 bits.

➢ Indirecto con desplazamiento: suma de registro más un desplazamiento (distinto de 0).

➢ Indirecto vía registro: si el desplazamiento es 0 el direccionamiento es indirecto vía registro. ➢ Directo absoluto: Si el desplazamiento es vía registro R0 (que vale 0), entonces se obtiene directo absoluto.

Repertorio de instrucciones

Para analizar el comportamiento de este procesador, se va a considerar un repertorio básico de 8 instrucciones: 2 de movimiento de datos con memoria, 5 aritméticas y logicas, y 1 de salto condicional. (mirar set de franja);

Instrucciones tipo I: Load/store

➢ RS: registro base

➢ RT: Registro de fuente o destino de la transferencia

➢ Desp: offset

➢ Instrucciones tipo R: Aritmético/lógicas

➢ RS: operando 1

➢ RT: operando 2

➢ RD: resultado

➢ Instrucciones tipo J: de salto

➢ RS: operando 1

➢ RT: operando 2

➢ Destino: es un desplazamiento respecto del PC (16 bits)

Ruta de datos e instrucciones

La ruta de los datos e instrucciones del nanoMips. Se pueden identificar:

➢ Banco de registros:

➢ Está compuesto por 32 registros (el R0=0)

➢ Se pueden leer 2 registros al mismo tiempo (doble entrada)

➢ Se puede escribir en 1 solo registro

➢ ALU:

➢ Es una unidad de cálculo típica de 2 entradas y 1 salida

➢ En la entrada se pueden seleccionar diferentes fuentes de datos

➢ Unidad de cálculo de dirección de próxima instrucción:

➢ Calcula el próximo valor con el que se carga el PC

➢ Puede ser el de la instrucción consecutiva (PC+4), o sumando un desplazamiento (instrucción de salto con desplazamiento).

También se puede apreciar que existen 2 bancos de memoria:

➢ Memoria de instrucciones:

➢ Solo contiene el programa a ejecutar

➢ Se accede únicamente para lectura, en la fase de búsqueda de la instrucción, y accedida solo a través del PC.

➢ Memoria de datos:

➢ Contiene los datos a leer o escribir

➢ Solo se accede a través de las instrucciones LOAD y STORE

Ruta de datos, instrucciones y control

La Unidad de control (CU) es una unidad relativamente sencilla que captura la instrucción y a partir de ella genera las señales de control requeridas por los demás bloques operativos. En particular, la CU necesita el campo del código de operación (OPCODE) para determinar que tareas deberá realizar para completar la instrucción corriente. El resto de los campos se usan para las identificaciones restantes. Por ejemplo:

➢ El campo RS selecciona 1 registro en el banco de registros

➢ Idem los campos RT y RD (cuando corresponda)

➢ El resto de los campos se usan para completar las órdenes a dar a las Unidades funcionales restantes (por ejemplo la ALU).

La Unidad de control (“Control”), a partir de los bits del campo OPCODE (bits 31-26) genera las señales de control para las demás unidades, por ejemplo RegWrite, ALUSrc, MemWrite, MemToReg, etc.

Diagrama de estados

Tiene una secuencia de instrucción compuesta por 5 estados.

Fase 1 - Búsqueda de instrucción (F, Fetch):

➢ ocurre en todas las instrucciones

➢ Busca y lee la instrucción en la memoria (de instrucciones)

➢ Actualizar el PC (sumar 4 al valor actual del PC): cada instrucción ocupa 4 bytes, por lo que la próxima instrucción consecutiva está en la dirección actual + 4.

Fase 2 - Decodificación (D, Decode) y acceso a registros:

➢ ocurre en todas las instrucciones

➢ Se decodifica la instrucción (del campo CODOP)

➢ Se accede al banco de registros

➢ Opcional: Extensión del signo del offset para cálculo de la dirección efectiva

Fase 3 - Ejecución (X, Execute)

➢ ocurre en todas las instrucciones

➢ Se ejecuta la operación en la ALU

Fase 4 -Acceso a memoria (M, Memory Access)

➢ ocurre solo en las instrucciones de LOAD y STORE

➢ Se accede a memoria

Fase 5 - Almacenamiento en Registro (W, Writeback)

➢ ocurre en las instrucciones que almacenan un dato en un registro

➢ También es en esta fase cuando puede calcular el desplazamiento a sumar al PC en instrucciones de salto.

Preguntar de la 22 a la 27

Ciclo de instrucción – Ejecución MONOCICLO

En las descripciones anteriores, cada vez que el PC envía una dirección a la memoria de instrucciones, se accede a una nueva instrucción. Como no hay registros sincrónicos intercalados en las trayectorias de datos e instrucciones, el dato fluye por las unidades funcionales (registros, ALU, memoria, MUX, etc.) hasta que se complete la ejecución de la instrucción.

No se inicia un nuevo ciclo de instrucción hasta que el PC se carga con un nuevo valor (instrucción consecutiva o la de salto). Si el PC está sincronizado con un reloj (el de la CPU) significa que en cada ciclo de reloj se inicia una nueva instrucción. Este tipo de forma de operación se llama Ejecución MONOCICLO, y se muestra en la imagen siguiente.

nanoMIPS con ejecución Monociclo

El período del reloj se hace con el tiempo necesario para completar la instrucción más lenta.

➢ En las instrucciones más rápidas el procesador “espera” el fin del ciclo para continuar (Ej: I1, I3).

➢ La eficiencia depende del tiempo de resolución de la instrucción más lenta (I2).

Ciclo de instrucción – Ejecución MULTICICLO

En la ejecución MONOCICLO el ciclo de reloj se hace en función de la instrucción “más larga”, es decir la instrucción de LOAD.

Las instrucciones que son “más cortas” tienen que esperar un tiempo hasta que se complete el período de reloj. Si la tasa de instrucciones cortas es mucho mayor que la de las largas, se puede perder mucho tiempo inútilmente.

Una solución es fijar un ciclo de reloj más pequeño, y disponer que cada instrucción ocupe varios períodos de reloj. Este tipo de forma de operación se llama Ejecución MULTICICLO, y se muestra en la imagen siguiente.

En la ejecución MULTICICLO cada instrucción requiere varios ciclos de reloj.

➢ Las instrucciones más rápidas usan menos ciclos de reloj que las más lentas.

➢ El ciclo de instrucción es variable y se ajusta al tipo de instrucción.

Segmentación y solapamiento del cauce (Pipelining)

La segmentación del cauce (Pipelining) consiste en:

➢ Descomponer el proceso de ejecución de las instrucciones en fases o etapas.

➢ Las fases o etapas son ejecutadas por unidades separadas y capaces de operar simultáneamente.

➢ Las instrucciones se van ejecutando a medida que se liberan las unidades.

➢ Las instrucciones no necesitan esperar la terminación de la previa para comenzar a resolverse.

➢ La segmentación del cauce (pipelining) es una forma particularmente efectiva de organizar el hardware de la CPU para realizar más de una tarea al mismo tiempo. Explota el paralelismo en el flujo secuencial de instrucciones.

Para entender el concepto de segmentación de una tarea repetitiva se puede analizar un “proceso” de una lavandería que requiere 3 etapas:

➢ Lavado, que requiere 30 minutos

➢ Secado, que requiere 40 minutos

➢ Despacho, que requiere 20 minutos Las 3 etapas se ejecutan en 3 sectores (estaciones) separados. La primera forma de operar es ejecutar el proceso en forma puramente secuencial, cuando se termina con un encargue recién se empieza con el siguiente. Este modelo secuencial se muestra en la figura siguiente.

Como las unidades funcionales Lavado, Secado y Despacho son unidades distintas, pueden operar en forma simultánea.

Si se superponen (solapan) las tareas de las 3 unidades funcionales es posible obtener una mejora sustancial en el tiempo total de ejecución de los 4 encargues (A, B, C y D). Esa situación se muestra en la figura siguiente.

La mejora se obtiene debido al solapamiento de las tareas de las distintas unidades funcionales.

Se puede aplicar este concepto al proceso repetitivo de ejecutar una secuencia de instrucciones.

Supongamos que se tiene una máquina que resuelve las instrucciones en 3 fases:

➢ Búsqueda o Captación (F, Fetch)

➢ Acceso a memoria (búsqueda de la instrucción)

➢ Incremento del PC

➢ Decodificación (D, Decode)

➢ Decodificación de la instrucción

➢ Obtención de los operandos

➢ Ejecución (E, Execute)

➢ Si es procesamiento: ejecución en la ALU

➢ Si es acceso a memoria: obtención de la direc efectiva

➢ Si es salto: cálculo del destino y decisión de salto (s/n)

La secuencia de tareas se puede hacer secuencialmente o segmentadamente.

En el procesador con ejecución secuencial, se tardan 9 ciclos de reloj para completer 3 instrucciones.

➢ En el procesador segmentado de la figura anterior, de 3 etapas, se tardan 5 ciclos de reloj para completar las mismas 3 instrucciones.

➢ En ambos casos, cada instrucción require 3 ciclos de reloj.

Si la máquina tiene 4 fases para resolver una instrucción:

➢ Búsqueda o Captación (F, Fetch)

➢ Decodificación (D) ➢ Ejecución (E)

➢ Escritura o actualización (W)

Conclusiones:

➢ El comportamiento de un procesador ejecutando una secuencia de instrucciones se comporta de una forma similar a una línea de montaje en una planta de manufactura.

➢ 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 lo tanto, 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 (a nivel de diseño del hardware).

➢ Como el programador no interviene en la segmentación y solapamiento de las instrucciones, la segmentación es “invisible” (por ahora) al programador.

➢ Se puede usar tanto en procesadores RISC como CISC.

➢ El diseño de procesadores segmentados tiene gran dependencia del repertorio de instrucciones.

Segmentación y solapamiento del cauce en el nanoMIPS

Se puede aplicar el concepto de Segmentación en el nanoMIPS.

➢ De acuerdo a lo visto, el proceso de ejecución de las instrucciones en el nanoMIPS está compuesto por 5 fases o etapas.

➢ En la imagen siguiente se puede apreciar la segmentación de las fases de ejecución de una instrucción.

Los 5 segmentos del nanoMIPS son:

➢ F: búsqueda de la insttrucción en la MI

➢ D: decodificación y acceso a registros en el BR

➢ X: ejecución de la ALU

➢ M: acceso a la memoria de datos MD

➢ W: escritura del resultado en el banco de registros BR

➢ Y los recursos de la máquina son:

➢ MI: memoria de instrucciones

➢ BR: banco de registros

➢ ALU: ALU

➢ MD: memoria de datos

Para poder aplicar el concepto de segmentación en el nanoMIPS, hay que dividir (segmentar) el cauce de los datos en etapas.

➢ La forma de hacer esto es usando registros sincrónicos entre cada etapa.

➢ Los datos avanzan, de etapa a etapa, en cada ciclo de reloj, que es cuando el registro (de separación de la etapa) copia a la salida lo que tiene en su entrada.

➢ Los segmentos en que queda dividido el cauce de datos, corresponden a las 5 fases de ejecución de la instrucción: F, D, X, M y W.

➢ Este proceso se puede apreciar en la siguiente figura.

*Ejecución segmentada en 5 etapas en el nanoMIPS, de un secuencia de 5 instrucciones.*

Se considera que cada instrucción require 5 ciclos de reloj, uno por cada segmento (F-D-X-M-W).

➢ En cada ciclo se incorpora una nueva instrucción, sin haber completado las anteriores.

➢ La primera instrucción se termina en el período de reloj 5, hasta ese ciclo no se había resuelto ninguna instrucción.

➢ A partir del quinto ciclo, se termina 1 instrucción por ciclo.

➢ Notar que en el quinto ciclo el procesador está ejecutando 5 instrucciones simultáneamente, pero en distintas fases.

Para implementar la segmentación del cauce se requiere intercalar registros entre cada etapa. ➢ Los registros (sincrónicos) separan las unidades funcionales, para que puedan operar con distintas instrucciones.

➢ En cada ciclo de reloj el dato avanza de una unidad funcional a la siguiente en el cauce, hasta completar las fases requeridas por a instrucción.

➢ El cauce de datos queda segmentado de la siguiente manera.

Así como se segmenta el cauce de los datos, también se debe hacer lo mismo con las señales que controlan las unidades funcionales: MI, BR, ALU, MD, BR.

➢ Eso es así porque cada Unidad funcional está operando con diferentes instrucciones (en diferentes estados de ejecución).

➢ Por ejemplo, mientras la unidad funcional BR (Banco de registros) está buscando los operandos de la instrucción I1, la unidad funcional MI (Memoria de instrucciones) estará buscando la instrucción I2.

➢ Las señales de control se segmentan igual que el cauce de datos, usando registros sincrónicos entre cada etapa,.

Análisis de la Segmentación

El máximo rendimiento teórico se obtiene cuando se completa una instrucción en cada ciclo de reloj. En esas condiciones todas las unidades funcionales están trabajando simultáneamente con distintas instrucciones.

➢ Si K es el número de etapas del cauce, entonces: Vel. procesador segmentado = K x Vel.secuencial

El incremento potencial de la segmentación del cauce es proporcional al número de etapas del cauce.

➢ Notar que no se mejora la velocidad de ejecución de la instrucción, la segmentación incrementa la productividad (throughput), es decir la cantidad de instrucciones resueltas en un período de tiempo determinado.

Para el análisis anterior se han hecho algunas suposiciones:

➢ Todas las tareas o segmentos duran la misma cantidad de ciclos de reloj (tiempo).

➢ Todas las instrucciones siempre pasan por todas las mismas etapas.

➢ Todos las etapas pueden ser manejadas en paralelo (no hay conflictos para usarlas simultáneamente).

➢ No se consideraron instrucciones de salto. Para hacer un análisis más preciso se requiere estudiar el comportamiento de la segmentación, pero en condiciones mas realistas.

Las primeras correcciones a hacer son:

1.- No todas las instrucciones necesitan todas las etapas.

➢ Ej: en el nanoMips la instrucción SW RT, inmed(RS) no utiliza W

➢ Ej: en el MSX88: un MOV AX,mem ; no requiere X

2.- No todas las etapas pueden ser manejadas en paralelo.

➢ Ej: si la memoria no estuviera dividida y fuera una sola, los segmentos F (búsqueda de la instrucción) y M (acceso a memoria) accederían ambos a la memoria (única) de datos e instrucciones.

3.- Los programas tienen instrucciones de salto.

Los conflictos que aparecen si se tienen en cuenta las correcciones anteriores, se denominan riesgos. Existen 3 tipos de riesgos:

➢ 1.- Estructurales: son conflictos provocados por el uso de los “recursos”. Los recursos típicamente son: memoria, ALU, registros.

➢ 2.- Por dependencia de datos: son conflictos originados entre 2 o más instrucciones que comparten un mismo dato. Por ejemplo, una instrucción produce un resultado que lo necesita otra, ambas instrucciones dentro del cauce de datos.

➢ 3.- Por dependencia de control: son conflictos que ocurren cuando la ejecución de una instrucción depende de cómo se ejecute otra (ej.: un salto y los 2 posibles caminos).

1.-Riesgos Estructurales

En la imagen anterior se muestra la situación de un conflicto entre las fases F y M si la memoria fuera una sola.

➢ Por esta razón, la memoria del nanoMIPS está dividida en MI y MD, de manera de reducir los conflictos por accesos a memoria.

➢ Otra opción para evitar el conflicto por el acceso a un recurso es retardar (retrasar) 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 la performance del procesador. Y además, el retraso puede generar nuevos conflictos que también deben resolverse de alguna manera.

2.- Por dependencia de datos:

➢ Hay riesgos de conflictos por dependencia de datos, cuando un dato es usado en 2 o más segmentos del 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.

➢ Se pueden dar 3 tipos de dependencias:

Dependencia de salida WAW

Dependencia real RAW

AntiDependencia WAR

Los 3 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 (alteración en la secuencia de ejecución de instrucciones)

➢ Escritura después de Lectura (WAR, antidependencia)

➢ una instrucción lee un dato que otra escribe posteriormente

➢ no se puede dar en nuestro cauce simple

3.- Por dependencia de control:

➢ 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 salto.

➢ Una forma de resolver el conflicto es retrasar varios ciclos (3 ciclos) la próxima instrucción hasta que se resuelva el cálculo de la instrucción de salto

En la clase POWER hay ejemplos

***Clase 5 Correcciones en procesadores segmentados***

Problemas de la Segmentación

En la clase anterior se identificaron los tipos de conflictos que aparecen en un procesador segmentado real. Estos riesgos pueden producir atascos en el flujo de las instrucciones.

➢ 1.- Estructurales: provocados por conflictos en el uso de los “recursos”. Los recursos típicamente son: memoria, ALU, registros.

➢ 2.- Por dependencia de datos: son conflictos originados entre 2 o más instrucciones que comparten un mismo dato. Por ejemplo, una instrucción produce un resultado que lo necesita otra, ambas instrucciones dentro del cauce de datos.

➢ 3.- Por dependencia de control: Ejecución de instrucciones que alteran la secuencia normal de ejecución, es decir, instrucciones de salto (condicional e incondicional).

Soluciones a los problemas de la Segmentación

Solución elemental

La solución más elemental para superar los atascos, producto de los 3 tipos de riesgos que se pueden presentar en un procesador segmentado, 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, inclusive puede llegar a ser hasta inútil aplicar la segmentación.

➢ Hay varias técnicas que permiten reducir y, en algunos casos, eliminar los efectos de los riesgos.

➢ A continuación se presentan algunas de las soluciones usadas en las máquinas con segmentación del cauce.

Solución a los conflictos estructurales por uso de los recursos

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

➢ 1.- Duplicación de recursos de hardware

Ej: agregar sumadores o restadores además de la ALU.

➢ 2.- Separación del recurso en conflicto

Ej: separar memorias de instrucciones y datos.

➢ 3.- Subdividir el acceso al recurso

Ej: en el acceso a los registros (banco de registros) la escritura de un registro se puede hacer en el primer semiciclo de reloj, y la lectura en el segundo semiciclo, con lo que en un ciclo se pueden hacer las 2 operaciones, siempre y cuando la velocidad del recurso lo soporte.

Solución a los conflictos por dependencia de datos

➢ Recordemos que existen 3 tipos de dependencias de datos en un proceso de segmentación:

➢ RAW: lectura después de escritura

➢ WAR: escritura después de lectura

➢ WAW: escritura después de escritura

Las soluciones se pueden implementar de 2 maneras:

➢ 1.- por Hardware

➢ Retardos en las etapas del cauce, es la solución ya vista.

➢ Adelantamiento de operandos (forwarding) entre etapas del cauce.

➢ 2.- por Software

➢ Introducción de NOP (equivalente a insertar retardos), equivalente a introducir puntos de parada.

➢ Reordenación de código. A continuación se van a analizar las diferentes técnicas.

Solución a conflictos por dependencia de datos con ciclos de parada

La solución más sencilla para eliminar conflictos por dependencia de datos es agregando retardos o ciclos de parada en la ruta de datos.

Está claro que este tipo de solución permite resolver los conflictos por dependencia de datos.

➢ El gran problema de esta solución es que los ciclos de parada agregados reducen fuertemente la performance del procesador segmentado.

Ej en power

Solución a conflictos por dependencia de datos con forwarding o adelantamiento

Este método consiste en pasar directamente (“adelantar”), desde una unidad funcional a otra, el resultado obtenido en una instrucción a las instrucciones siguientes que lo necesitan como operando.

➢ Si el dato que necesita la instrucción i+1 ya está calculado por la instrucción i, se puede llevar (adelantar) a la entrada de la etapa de la instrucción i+1 que lo necesita, sin esperar la etapa final de escritura de la instrucción i.

➢ Esta solución puede ser relativamente sencilla de implementar si se identifican todos los adelantamientos y se pueden adelantar los datos a las unidades que lo necesitan.

En procesadores sencillos como el nanoMIPS este análisis es relativamente fácil. Si el procesador es más complejo el análisis pude resultar muy complicado.

➢ En el caso del nanoMIPS se pueden identificar 8 tipos de conflictos por dependencia de datos (del tipo RAW únicamente), y sus posibles soluciones con adelantamiento se muestran en la figura siguiente.

Ej en power

Soluciones a los problemas de la Segmentación

CASOS a considerar:

1)LW seguido de SW

2)LW seguido de Aritmética –Lógica

3)LW seguido de BEQ

4)Aritmético -Lógica seguida de SW

5)Aritmético -Lógica seguida de Aritmético -Lógica.

6)Aritmético -Lógica seguida de BEQ

7)LW seguido de LW

8)Aritmético -Lógica seguida de LW

Para cada caso se considera:

➢ A la derecha está remarcada la dependencia en la secuencia de instrucciones

➢ A la izquierda se indica donde se debe implementar el adelantamiento. Por ejemplo, en el caso 1 de LW seguido de SW, el adelantamiento debería hacerse entre la salida de la M (memoria de datos) y la propia entrada de M.

Los 8 casos encontrados, se pueden agrupar en 3 problemas de adelantamiento:

1) En los casos 1 y 4, adelantamiento desde la salida de M a la entrada de M.

2) En los casos 2, 3 y 7, adelantamiento desde la salida de M a la entrada de X.

3) En los casos 5, 6 y 8, adelantamiento desde la salida de X a la entrada de X.

Para solucionar el primer grupo (casos 1 y 4) se requiere prever un camino de datos directo desde la salida de M a la entrada de M, y para el segundo grupo (casos 2, 3 y 7) se requiere prever un camino de datos directo desde la salida de M a la entrada de X

La trayectoria de los datos desde la M hasta el BR (banco de registros) tiene una derivación que termina en multiplexores (MUX), que permiten seleccionar distintas alternativas de fuentes de datos a X (ALU) y a M (memoria de datos).

Cuando una unidad auxiliar, identificada como unidad de adelantamiento detecta en los campos de las instrucciones que se están ejecutando, las dependencias 1,2,3,4 y 7, selecciona en los MUX asociados a ella, las entradas necesarias para adelantar los datos requeridos (por X o M).

Para solucionar el tercer grupo (casos 5, 6 y 8) se requiere prever un camino de datos directo desde la salida de X a la entrada de X. Eso se puede resolver usando multiplexores (MUX) como se muestra a continuación.

La trayectoria de los datos desde la X hasta M o el BR (banco de registros) tiene una derivación que termina en los mismos multiplexores (MUX), que permiten seleccionar distintas alternativas de fuentes de datos a X (ALU) y a M (memoria de datos).

Cuando la unidad de adelantamiento detecta en los campos de las instrucciones que se están ejecutando, las dependencias 5, 6 y 8, selecciona en los MUX asociados a ella, las entradas necesarias para adelantar los datos requeridos (por X).

Conclusión:

En los análisis anteriores se puede apreciar que existen soluciones a los conflictos por dependencia de datos que permiten reducir o eliminar el impacto de estos conflictos en la performance del procesador. Los cambios introducidos no son complicados de implementar, y sus resultados justifican ampliamente su uso.

Solución a conflictos por dependencia de datos por software

Otra técnica que se usa para evitar los atascos por dependencia de datos, es a través del software.

➢ Esta solución es realizada por el compilador, por lo que, al igual que las técnicas por hardware, es “transparente” al programador. Hay 2 posibles soluciones:

➢ Introducción de instrucciones NOP (es decir, retardos) entre las instrucciones que tienen dependencia de datos, pero genera retardos que reducen la performance del procesador (equivalente al uso de puntos de parada).

➢ Reordenamiento de instrucciones, para tratar de aumentar la separación entre las instrucciones con dependencia de datos, pero hay que tener cuidado con modificar el comportamiento del programa.

Solución a conflictos por dependencia de datos mediante reordenamiento de instrucciones:

Mirar Pdf

En la imagen anterior se muestra a la izquierda el programa original (de 9 instrucciones), y a la derecha, el programa reordenado por el compilador al detectar conflictos de dependencia de datos.

Por ejemplo, hay dependencias entre las instrucciones 2 y 3 (registro R3), y entre la 3 y la 4 (registro R6). La instrucción 4 no se puede reubicar porque tiene dependencia con la 3, pero la instrucción 5 no depende de ninguna de las anteriores. Entonces, reubicando la instrucción 5 a continuación de la 2, se elimina la dependencia sin alterar la lógica del programa.

El compilador continúa con este proceso intentando reordenar las instrucciones para evitar los conflictos. En caso de no poder resolverlos, deberá insertar instrucciones de NO-OPERACIÓN (NOP) para eliminar los que persisten luego del reordenado.

Solución a los conflictos por dependencia de control

El tercer tipo de riesgo que puede haber en la segmentación es la presencia de instrucciones de salto, que pueden ser de 2 tipos:

➢ Incondicional: donde la dirección de destino se debe determinar lo más pronto posible, dentro del cauce, para reducir la penalización.

➢ Condicional: introduce el riesgo adicional por la dependencia entre la condición de salto y el resultado dependiente de una instrucción previa.

➢ Por ejemplo: en la instrucción de salto (condicional) BEQ en el nanoMIPS, recién se calcula la dirección de salto durante la fase W. Esta situación se muestra en la figura siguiente.

Igual que antes, la forma más elemental de resolver el conflicto es agregando tiempos muertos (ciclos de parada) hasta que desaparece el problema.

En el caso del nanoMIPS, se requerirían 3 ciclos de reloj para empezar la instrucción siguiente a la de salto, considerando que se puede calcular el salto en el primer semiciclo del ciclo W de la instrucción BEQ, y se puede leer el resultado en el segundo semiciclo del ciclo F de la instrucción a donde se salta.

Solución a los conflictos por dependencia de control – penalización con adelantamiento

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 a la etapa D, donde se decodifica y se accede a los registros.

➢ El objetivo del adelantamiento es evaluar tempranamente la condición de salto, usando, por ejemplo, un restador extra a la salida del banco de registros, en lugar de esperar a la etapa X asociada a la ALU, para determinar la condición de igualdad de la instrucción BEQ.

➢ Además se puede agregar un sumador adicional para calcular la dirección de salto, en lugar de esperar a la etapa X con la ALU, para adelantar la obtención del salto.

Adelantamiento en el nanoMIPS a la resolución de la instrucción BEQ y cálculo de la próxima instrucción

Ej en pdf

Con la ruta de datos modificada, la respuesta del nanoMIPS a una instrucción de salto BEQ se pierde 1 solo ciclo de reloj, en lugar de los 3 que se perdían originalmente.

Técnicas de tratamiento de saltos

En máquinas más complejas, la solución antes presentada para el nanoMIPS puede no ser tan sencilla de implementar. En general las técnicas para mitigar el impacto, en la performance de procesadores segmentados, de las instrucciones de salto pueden ser de 2 tipos:

➢ Técnicas por hardware: que consiste en resolver los conflictos a nivel de hardware mediante predicción de saltos.

➢ Técnicas por software: que intentan resolver los conflictos por software a nivel del compilador.

Técnicas de resolucion de instrucciones de salto por hardware

Estas técnicas se basan en usar estrategias para predecir el resultado de la instrucción de salto. Pueden ser de 2 tipos:

➢ Técnicas estáticas: no tienen en cuenta información previa de la ejecución del programa.

➢ Técnicas dinámicas: tienen en cuenta la historia previa del programa en ejecución.

Técnicas de predicción de saltos por hardware – estática

Básicamente 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.

En el nanoMIPS siempre capta la próxima instrucción, es decir predice que no va a saltar. Si no salta no se pierde ningún ciclo, si salta se pierde 1 ciclo.

Ventajas:

➢ Sencilla de implementar

➢ Pequeños cambios en la Unidad de control

➢ Ruta de datos sin cambios

➢ Desventajas:

➢ La instrucción a descartar puede afectar el estado del procesador (registros y/o memoria).

➢ En el nanoMIPS el ciclo que se descarta es de búsqueda de la instrucción (ciclo F) que no modifica el procesador.

Técnicas de predicción de saltos por hardware – dinámica

Está técnica se basa en el análisis de datos previos (la “historia”) sobre como se comporta el programa, para predecir la acción.

➢ Algunos ejemplos de técnicas de predicción por hardware son:

1. Conmutador Saltar/no saltar

2. Tabla de historia de saltos (branch history table)

3. Predicción según el código de operación

4. Varios cauces (uno por cada opción de salto) (Multiple stream)

5. Precaptación del destino del salto (Prefetch branch target)

6. Buffer de bucles

Conmutador Saltar/no saltar

Estando en uno de las 2 condiciones (predecir que salta o predecir que no salta), se requieren 2 predicciones fallidas consecutivas para conmutar al otro estado.

Esta predicción tiene un buen comportamiento en lazos donde el resultado de una consulta (por ejemplo un contador que tiene que llegar a 0) se puede repetir muchas veces antes de que cambie el resultado de la consulta.

Tabla de historia de saltos (BHT branch history table)

Se implementa usando una pequeña cache asociada a la etapa de búsqueda (F). La cache tiene 3 campos:

➢ Dirección de una instrucción que es del tipo bifurcación ➢ Dirección del destino ó Instrucción destino

➢ 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 la predicción falla, se actualiza la tabla

➢ Si la instrucción de ramificación no estaba en la BHT, se la carga como nueva entrada.

BHT

Tabla caché para almacenar instrucciones de ramificación con las direcciones de destino para el caso de predecir el salto.

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.

➢ Por ejemplo, si la instrucción de salto involucra un lazo que se repite muchas veces, la condición que se usa para repetir el lazo ocurrirá muchas más veces que la que se usa para salir del lazo (que ocurrirá 1 sola vez).

➢ La tasa de acierto puede llegar a alcanzar un 75%

Varios cauces (uno por cada opción de salto- Multiple stream)

En esta solución se usan cauces distintos (duplicando hardware) para ejecutar simultáneamente los cauces, el que corresponde a la opción de no saltar y el que corresponde a la instrucción de salto.

➢ Luego, cuando se determina el resultado de la ramificación, se debe utilizar el cauce correcto y descartar el incorrecto.

➢ Como se están captando simultáneamente 2 cauces distintos, aumentan las búsquedas, lo que puede provocar retardos en el acceso al bus y a los registros.

➢ Si hay nuevos saltos en los cauces en ejecución, se necesita disponer de más cauces para contemplar esta situación, lo que aumenta la complejidad del hardware.

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.

➢ El IBM 360/91 usa este método.

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. Funciona parecido a la cache, solo que contiene instrucciones consecutivas, únicamente.

Técnicas de resolución de instrucciones de salto por software

➢ Las técnicas de resolución por software de conflictos debido a las instrucciones de salto se basan en tratar de realizar trabajo útil mientras el salto se resuelve.

➢ Cuando hay una instrucción de salto, se necesita 1 ciclo (o más) 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 (en lugar de descartarla).

➢ 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.

➢ Si no es posible insertar instrucciones, se necesita agregar instrucciones del tipo NO-OPERACION (NOP) para evitar el conflicto.

La forma de llenar los huecos que quedan luego de las instrucciones de salto es mediante el reordenamiento de las instrucciones. Ese trabajo lo hace el compilador.

➢ Las instrucciones a reordenar pueden provenir de:

➢ Caso 1: Instrucciones antes del salto

➢ Caso 2: Instrucciones del destino del salto

➢ Caso 3: Instrucciones a continuación del salto

➢ La mejor solución sería reordenar instrucciones anteriores a la de salto, porque siempre deben ejecutarse y van a ser útiles.

➢ El reordenamiento con instrucciones del destino del salto o a continuación del salto deben ser tal que no modifiquen el estado del proceso aun cuando falle la predicción.

Caso 1: el hueco se rellena con una instrucción anterior al salto. Sirve siempre, es la mejor opción (no siempre es posible).

➢ Caso 2: Como la instrucción ADD R1,R2,R3 se requiere ejecutar antes de la instrucción de salto, no se puede reubicar después del salto. El compilador busca una instrucción a reubicar, en el lugar a donde puede llegar a saltar, por ejemplo SUB R4,R5,R6. La copia en el hueco, y corrige el lugar a donde salta (MULT r6,r7,r8). Si la instrucción de salto efectivamente salta, entonces la instrucción insertada en el hueco (SUB R4,R5,R6) sirve y se adelantó trabajo. Si la instrucción no salta, la instrucción insertada (SUB R4,R5,R6) se ejecuta pero no sirve y el resultado no debe alterar la lógica. En otras palabras, el valor que se cargue en R4 (en la instrucción SUB R4,R5,R6) no sirve y no debe alterar la lógica del programa.

Caso 3: Como la instrucción ADD R1,R2,R3 se requiere ejecutar antes de la instrucción de salto, no se puede reubicar después del salto. El compilador busca una instrucción a reubicar, a continuación de la de salto, por ejemplo SUB R4,R5,R6. Si la instrucción no salta, entonces la instrucción insertada en el hueco (SUB R4,R5,R6) sirve y se adelantó trabajo. Si la instrucción salta, la instrucción insertada (SUB R4,R5,R6) se ejecuta pero no sirve y el resultado no debe alterar la lógica. En otras palabras, el valor que se cargue en R4 (en la instrucción SUB R4,R5,R6) no sirve y no debe alterar la lógica del programa.

Aproximadamente el 60% de los huecos se rellenan con instrucciones diferentes a NOP (barra oscura en la imagen inferior)

➢ Del 60% de huecos rellenados con instrucciones (los 3 casos vistos antes), entre el 80 y 85% corresponde a instrucciones útiles (barras rayadas).

➢ En definitiva, el 50% de los huecos (retardos) se aprovechan con instrucciones útiles.

Segmentación en el i80486

Como ejemplo de procesador convencional (tipo CISC) se puede analizar el procesador de Intel 80486, que tiene un cauce compuesto por 5 etapas, 2 de ellas de decodificación:

➢ Fetch (F)

➢ Decode 1 (D1)

➢ Decode 2 (D2)

➢ Execute (EX)

➢ Write back (WB)

Secuencia de instrucciones sin penalización

3 instrucciones de movimiento entre memoria y registro sin retrasos.

Secuencia de instrucciones con penalización de 1 ciclo

Retardo de 1 ciclo en el cauce por carga que utiliza un puntero (con adelantamiento desde la etapa EX a la D2).

Secuencia de instrucciones con penalización de 2 ciclos

Retardo de 2 ciclos en el cauce por temporizado en una instrucción de bifurcación

Procesador segmentado nanoMIPS multifuncional

➢ En general es imprescindible que los procesadores dispongan de operaciones en punto flotante. El modelo visto del nanoMIPS no las tiene.

➢ Si se quiere agregar operaciones en punto flotante en el nanoMIPS, se requiere realizar las siguientes modificaciones:

➢ Agregar instrucciones en PF (Ej: ADD.D, LD.D)

➢ Agregar registros de PF (Ej: F1, F2, etc.) ➢ Agregar hardware para las operaciones aritméticas de: Suma, Resta, Multiplicación y División

➢ Es posible compatibilizar la segmentación con las operaciones en punto flotante.

El problema que tiene el esquema anterior es que los tiempos de ejecución de las unidades de punto flotante son muy distintos de los de la unidad de enteros.

➢ Hasta ahora habíamos considerado que todas las etapas del nanoMIPS tenían la misma duración, pero ahora las unidades funcionales de PF pueden requerir varios ciclos para completarse.

➢ Por ejemplo, la división en punto flotante lleva muchos mas ciclos que una suma de números enteros

➢ Conclusión: cuando se incorpora punto flotante, no todas las etapas del cauce van a durar lo mismo.

Si las unidades funcionales en punto flotante tardan varios ciclos, entonces una secuencia de 2 instrucciones consecutivas en punto flotante requeriría retardar la segunda instrucción hasta que la primera se complete.

➢ En el siguiente ejemplo se muestra cual es la situación: ADD.D F1, F2, F3 ADD.D F4, F5, F6

➢ No se puede empezar la segunda instrucción ADD.D F4,F5,F6 hasta que no se termine la primera.

➢ Hay una solución a este problema: segmentar las unidades funcionales de punto flotante.

Segmentando las unidades de punto flotante se pueden iniciar nuevas instrucciones en PF sin haber completado a actual

. ➢ En la figura anterior se ha segmentado:

➢ La unidad de suma en PF en 4 etapas

➢ La Unidad de multiplicación en PF en 10 etapas

➢ La unidad de división en PF en 20 etapas.

***Clase 6 Procesadores RISC y CISC***

Algunos hitos significativos en la evolución de los sistemas de cómputo

Familia de computadoras: el concepto de familia de computadoras fué introducido por IBM a partir de su modelo System/360 en 1964, basado en la disponibilidad de un conjunto de máquinas con una arquitectura común y prestaciones variables de acuerdo a las necesidades.

Arquitectura e implementación: son 2 conceptos separados que introdujo DEC en su máquina PDP-8, referidos a las características visibles al programador y las prestaciones del sistema.

Unidad de control microprogramada: idea propuesta por Wilkes en 1951 para el desarrollo sistemático de las unidades de control, e introducida por IBM en la línea S/360 en 1964.

Memoria cache: pequeña memoria de alta velocidad para balancear la capacidad de la CPU y la velocidad de transferencia de la Memoria principal. Introducida En 1968 en la IBM S/360 Modelo 85.

Memoria de estado sólido: desarrollo, en 1970, de memorias semiconductoras en reemplazo de las memorias magnéticas, lentas y complicadas de manejar, usadas hasta entonces.

Microprocesador : en 1971 aparece el primer microprocesador fabricado por Intel, el i4004.

Segmentación de cauce: Introduce el paralelismo en la naturaleza secuencial de los programas.

Procesadores múltiples: desarrollo de sistemas con múltiples procesadores para aumentar la potencia de cómputo.

Procesadores RISC: diseño conceptualmente opuesto a los procesadores CISC.

Evolución de los procesadores

El desarrollo de la tecnología de fabricación de circuitos integrados de muy alta escala de integración (VHSI) permitió incorporar cada vez mas funcionalidades dentro de un chip.

La incorporación de nuevas funcionalidades se orientó principalmente en mejorar la eficiencia en la ejecución de programas.

Un aspecto de esta mejora consistió en brindar un mayor soporte a los lenguajes de alto nivel (HLL), sobre todo los más complejos.

Dar mayor soporte a los HLL tiende a facilitar el proceso de compilación (y también el trabajo de elaboración de compiladores), sobre todo considerando que el costo del “software es típicamente mucho más caro que el hardware”.

La “distancia” entre un HLL y el lenguaje de máquina se denomina GAP semántico, y básicamente es la relación entre la cantidad de sentencias de un programa en HLL y la cantidad de instrucciones de máquina que se requieren para resolverlo.

El mayor soporte tiende a reducir el “GAP semántico”, agregando instrucciones de máquina más complejas que resuelvan las sentencias del HLL más directamente.

Procesadores CISC (Complex Instruction set computer)

Los conceptos anteriores sentaron las bases de diseño de los procesadores “convencionales”:

Repertorios de instrucciones grandes.

Formato de instrucción complejo y de longitud variable.

Muchos modos de direccionamiento.

Capacidad de implementar sentencias de HLL con muy pocas instrucciones de máquina. Esta idea consiste básicamente en resolver una sentencia de HLL “por hardware”.

De los conceptos anteriores surge el modelo de procesador conocido como CISC, es decir, procesador con repertorio de instrucciones complejo

Con el uso masivo de la tecnología VHSI y el desarrollo de procesadores tipo CISC, se empezaron a realizar diferentes estudios sobre el comportamiento de los procesadores tipo CISC.

Los estudios se orientaron a:

1. Uso del repertorio de instrucciones

2. Uso de los operandos

3. Secuencia de ejecución de instrucciones

Los estudios se realizaron de 2 maneras distintas:

Estáticos

Dinámicos

Los estudios sobre operaciones estaban orientados a determinar el uso de las instrucciones y, por lo tanto, su interacción con la memoria.

Los estudios sobre operandos básicamente intentaban determinar cuales eran los tipos de datos usados y su frecuencia de uso.

Los estudios sobre el secuenciamiento de la ejecución de instrucciones tenían como objetivo determinar el impacto de disponer de un repertorio de instrucciones largo y complejo, en la Unidad de Control (que debe resolver las instrucciones), y en el cauce de datos que tiene relación con la eficiencia en la ejecución de una secuencia de instrucciones.

Los estudios estáticos computaban cantidades basados en el código objeto ejecutable. Es una forma sencilla de obtener métricas, pero tienen el problema de que no tienen en cuenta el flujo del programa ejecutado, y pueden resultar bastante equivocados. Por ejemplo, la cantidad de instrucciones que componen un lazo no tiene nada que ver con las veces que se ejecuta.

Los estudios dinámicos computan cantidades en base a la ejecución de diferentes programas (típicamente programas de prueba o “benchmarks”). Son más realistas porque los resultados tienen en cuenta la situación real por la que pasan los procesadores.

Estudio sobre el uso de instrucciones

Este estudio permitió medir el uso de las instrucciones, y su interacción con la memoria. Se efectuó sobre VAX, PDP-11 y MC68000.

Las operaciones (instrucciones) se dividieron en:

Asignación: movimiento de datos, aritméticas y lógicas

Sentencias condicionales: IF, LOOP

Sentencias incondicionales: GOTO

Llamadas y retornos de subrutinas: CALL

Otras

Mirar PDF en compu

Estudio sobre el uso de instrucciones – Resultados

Los resultados son medidos de 3 maneras (columnas):

Aparición dinámica: cuenta la cantidad de sentencias del HLL ejecutadas.

Instrucciones de máquina: computa las instrucciones de máquina para las sentencias del HLL ponderadas al tiempo de ejecución de las mismas.

Referencias a memoria: computa la cantidad de accesos a memoria.

Se puede apreciar del estudio mostrado que:

Las sentencias de asignación son la más numerosa, seguida por las de llamadas a procedimientos y las sentencias condicionales (primera columna).

Cuando se hace la ponderación basado en la cantidad de instrucciones de máquina que resuelven las sentencias de alto nivel, las sentencias condicionales y llamadas a procedimientos son las que requieren más instrucciones de máquina (segunda columna).

Cuando se hace la ponderación basado en la cantidad de accesos a memoria, siguen predominando las sentencias condicionales y llamadas a procedimiento (tercer columna).

Los accesos a memoria tienen mucho impacto en el tiempo de ejecución del programa.

Por lo tanto, los llamados a procedimiento consumen mucho tiempo y tiene que hacerse lo más eficiente posible para optimizar los accesos a memoria

Estudios sobre llamadas a procedimientos - Resultados

En varios estudios se analizó el comportamiento en llamadas y retornos a procedimientos.

En un alto porcentaje (>98%) se pasan menos de 6 datos.

En un alto porcentaje (>92%) la cantidad de variables locales es menor a 6.

En general el nivel de anidamiento es menor a 7.

Conclusiones:

La mayoría de los programas tienen una secuencia corta de llamadas seguida por la secuencia de retornos.

La mayoría de las variables son locales.

Las referencias a operandos están muy localizadas.

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 (no reflejado en este estudio).

Conclusiones finales

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 resolverlas 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.

Procesadores RISC

Los estudios realizados sobre el comportamiento de los programas, condujeron a un nuevo enfoque sobre el diseño de procesadores, a los que se identificó genéricamente como procesadores RISC (Reduced Instruction Set Computer).

Los principales referentes de este nuevo enfoque fueron Patterson y Hennesey (“Arquitectura de computadoras – Un enfoque estructurado”).

Conceptualmente, los primeros procesadores RISC surgieron a mediados de los 80 en las Universidades de Berkeley (David Patterson, proyecto RISC 1) y Stanford (John Hennesey, proyecto MIPS-XMP).

Los primeros RISC 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). ¬ Unidad de control “cableada” (no microprogramada).

Diseño del cauce optimizado para ejecutar 1 instrucción por ciclo (uso preciso de la segmentación)

Máquina tipo LOAD-STORE: acceso a memoria solo con instrucciones de movimiento de datos. Instrucciones aritméticas y lógicas solo entre registros.

Uso intensivo de registros (de uso general) con optimización por software.

Comparación entre CISC y RISC

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.

Ventana de registros

Registros en procesadores RISC

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 2 maneras:

Por Hardware: agregando más registros

Por Software: optimizando la asignación de registros a las variables que se usen más en un período de tiempo dado.

Solución por hardware de los registros en procesadores RISC

Tener un banco de registros amplio tiene ventajas y desventajas:

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 los resultados.

Llamadas y retornos de subrutinas - Variables

Por los estudios realizados, los llamados y retornos de subrutina son los que más tiempo consumen.

Por lo tanto, es necesario hacer muy eficiente el manejo de este tipo de instrucciones.

Los estudios realizados permiten inferir:

En general, las subrutinas usan pocas variables locales.

Cada llamada recibe y pasa pocos argumentos.

Las subrutinas usan pocas variables globales.

En general, cada subrutina necesita, para operar, 4 tipos de datos:

Datos de entrada: argumentos de entrada

Datos propios: datos (variables) locales

Datos de salida: para pasar argumentos

Datos globales (vistos por todas las subrutinas)

Los estudios realizados permiten inferir:

En general, las subrutinas usan pocas variables locales.

Cada llamada recibe y pasa pocos argumentos.

Las subrutinas usan pocas variables globales.

Llamadas y retornos de subrutinas – Variables y registros

R0-R7 REGISTROS PARA DATOS GLOBALES

R8-R15 REGISTROS PARA PARAMETROS RECIBIDOS

R16-R23 REGISTROS PARA VARIABLES LOCALES

R24-R31 REGISTROS PARA ARGUMENTOS A PASAR

Cada tipo de dato requiere un banco de registros:

Datos de entrada: registros de parámetros

Datos propios: registros locales

Datos de salida: registros temporales

Datos globales: registros globales

Si cada subrutina administra “su” banco de registros (que le “pertenece”), excepto los globales que son comunes a todos, la cantidad real de registros de cada subrutina se limita a un número no excesivamente grande (24 en el ejemplo anterior), y por lo tanto, requiere pocos bits para identificarlo (5 bits).

Los registros globales son comunes a todas las subrutinas, y accesibles por todas ellas.

Por otra parte, los problemas que aparecen con el uso de subrutinas son:

Cada vez que una subrutina invoca otra subrutina (anidamiento de subrutinas) debe pasarle los argumentos, lo que produce un gasto de 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. Para solucionar los problemas en el uso de Procedimientos y pasaje de parámetros se puede usar una estrategia conocida como Ventana de registros.

La solución de usar una Ventana de registros consiste en:

1°) Asignar a cada subrutina un banco de registros propio:

Por ejemplo, los registros R8 a R31 son los registros accesibles por la subrutina. Los registros R0 a R7 son globales y accesibles por todas las subrutinas.

2°) Superponer los registros donde recibe los parámetros una subrutina con los registros donde pasa los argumentos la subrutina que la llama. Es decir:

La subrutina de nivel j+1 recibe los parámetros de la subrutina de nivel j en los registros R8 a R15. La subrutina de nivel j le pasa los parámetros a la subrutina de nivel j+1 en los registros R24 a R31,

Entonces si se solapan los registros R24 a R31 de la subrutina j con los R8 a R15 de la j+1, se pueden pasar los parámetros directamente entre la subrutina j y la j+1 a través del mismo grupo de registros (físico).

Mirar imagen pdf 34

Los registros temporarios del nivel j y los registros de parámetros del nivel j+1 son físicamente los mismos. En cambio, los registros locales son físicamente distintos para cada subrutina y solo accesibles por ella.

De esta manera, cada subrutina accede a una ventana de registros, de los cuales los primeros y los últimos están solapados (superpuestos), y son usados para pasaje de parámetros.

El banco de registros se implementa como un buffer de tipo circular, con una capacidad determinada de “ventanas”, basada en la “profundidad “de anidamientos admitidas por el procesador (basada en los estudios antes analizados, podría ser, por ejemplo, del orden de 7).

Buffer circular usado en un esquema de ventana de registros.

En el nivel 0 (w0), los registros accesibles son Ain, Aloc y Bin.

En el nivel 1 (w1), los registros accesibles son Bin, Bloc y Cin.

Los registros Bin son accesibles por la rutina de nivel 0 (por donde pasa los parámetros) y la rutina de nivel 1 (por donde recibe los parámetros).

Cada invocación de una subrutina mueve un puntero que apunta al siguiente banco de registros. El movimiento del puntero es tal que quedan superpuestos (solapados) los registros usados para pasaje de parámetros.

La cantidad de registros accesibles por cada subrutina es la misma, pero cambian los registros físicos a los que accede.

Variables globales

Para el manejo de variables globales (es decir compartidas por todas las subrutinas) se pueden usar 2 soluciones:

1. El compilador asigne posiciones de memoria a las variables: es algo 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 registros no está mostrado en la ventana de registros.

Ventana de registros (1)vs. Memoria cache(2)

(1)

Banco de registros amplio

Todos los datos son escalares y locales

Acceso individual a variables

Variables globales asignadas por el compilador

Salvaguarda/restauración basadas en la profundidad de anidamiento

Direccionamiento de registro

(2)

Cache

Datos escalares locales reciente-mente usados

Acceso a bloques de memoria

Variables locales y globales usadas recientemente

Salvaguarda/restauración basadas en el algoritmo de reemplazo

Direccionamiento de memoria

Técnicas por software

Técnicas de optimización por software

También se puede mejorar las prestaciones en los procesadores RISC, optimizando el uso de los registros.

Pero los lenguajes 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 (llamados “virtuales”). A cada variable del programa se le asigna un registro virtual (que son ilimitados).

El compilador mapea (asigna) el número ilimitado de registros simbólicos a un número fijo 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 una técnica denominada ‘coloreado de grafos’.

Características relevantes de los procesadores RISC

Formatos de instrucción sencillos (fijo).

Modos de direccionamiento sencillos.

Diseño de la Unidad de control del tipo “cableado” (sin microcódigo).

Operaciones registro a registro.

Instrucciones a memoria únicamente LOAD y STORE.

Una instrucción por ciclo (segmentación eficiente).

Típicamente máquinas tipo Harvard.

Mayor tiempo/esfuerzo de compilación

Ventajas y desventajas de los procesadores CISC.

Ventaja: El compilador es mas sencillo por disponer de un repertorio amplio de instrucciones y modos de direccionamiento

Desventaja: las instrucciones de máquina complejas son difíciles de aprovechar por el compilador, es decir las usa poco nada.

Desventaja: la optimización es más difícil de realizar.

Ventaja: los programas son más pequeños, tiene menos instrucciones, lo que probablemente implique que ocupa menos memoria. Sin embargo, la memoria hoy día es muy barata, por lo que esta ventaja es muy relativa.

El número de bits de memoria que ocupa no tiene porqué ser más pequeño al tener menos instrucciones

Desventaja: al tener repertorio de instrucciones más amplio, los campos de código de operación son más largos y aumentan el tamaño de la instrucción.

Las referencias a registros necesitan menos bits.

Procesadores RISC y CISC

Velocidad de ejecución

El objetivo principal en el desarrollo de los procesadores es mejorar la velocidad de ejecución de los programas.

Se supone que aumentar la complejidad del procesador debería mejorara su velocidad en ejecutar los programas.

Pero:

Los procesadores CISC casi no usan las instrucciones más complejas.

El repertorio de instrucciones complejo exige una Unidad de control más compleja y lenta. En particular la memoria de control en la Unidad de control (microprogramada) es muy grande y “lenta”.

Una Unidad de control más lenta aumenta el tiempo de ejecución de las instrucciones simples, es decir, penaliza las instrucciones que podrían hacerse más rápido.

En definitiva, no está comprobado que la tendencia hacia CISC fuera la apropiada.

Comparación entre RISC y CISC

Se han realizado numerosas mediciones comparativas entre procesadores RISC y CISC.

Pero las comparaciones tienen varios problemas:

No existe un par de máquinas RISC y CISC directamente comparables.

No hay un conjunto de programas de prueba definitivo.

EN los análisis, es muy difícil separar los efectos del hardware de los del compilador.

En muchos casos, las comparaciones se realizan usando prototipos o simulaciones en un “ambiente” controlado, y pocas veces con productos comerciales.

La mayoría de las máquinas son una mezcla de ambas.

Por otra parte es necesario definir qué parámetros se van a medir. En general, las evaluaciones pueden ser:

Cuantitativas: básicamente comparando los tamaño de los programas y su velocidad de ejecución

Cualitativa: revisión de soporte de lenguajes de alto nivel y uso óptimo de los recursos VLSI.

Conclusiones:

No existe una marcada diferencia de performance entre uno y otro

No está clara la barrera que separa uno u otro estilo.

Muchos diseños incluyen características de ambos criterios, por ejemplo PowerPC y Pentium II.

Procesador RISC SPARC (Sun Microsystemas)

La primer generación del SPARC se empezó a comercializar en 1987. Tenía una frecuencias de reloj de 16 a 50 MHz, y un diseño del tipo escalar (los procesadores escalares se verán más adelante).

La segunda generación, conocida como SUPER SPARC, fue liberada en 1992. Tenía una frecuencias de reloj de 33 a 50 MHz, y una arquitectura super escalar.

La tercera generación se conoció como ULTRA SPARC, y fué liberada a mediados de 1996. Disponía de una arquitectura super escalar de 4 etapas y de 64 bits, 5 unidades de coma flotante, y velocidades de reloj entre 250 y 300 MHz.

Procesador AMD 29000

El AMD 29000 fué introducido en 1988. Tenía memorias de datos e instrucciones separadas (como el nanoMIPS), y un ancho de bus de 32bits.

Otros procesadores RISC comerciales

PA-RISC: Hewelett Packard ¬ Power PC: Apple, Motoroloa e IBM (Power Macintosh) ¬ MIPS