**Arquitectura Von Neumann**

* La unidad central de procesamiento (CPU) está constituida por la unidad de control (UC) y la unidad aritmético-lógica (ALU).
* Datos e instrucciones deben introducirse en el sistema y los resultados se proporcionarán mediante componentes de entrada/salida (E/S).
* Se necesita almacenar temporalmente datos e instrucciones: Memoria Principal

**Repertorio de instrucciones**: Es el conjunto completo de instrucciones que se realizan en una CPU. (Código máquina, Binario) Representado simbólicamente por un conjunto de códigos de ensamblaje:

1. de operaciones:

ADD (sumar), SUB (restar), LOAD (cargar datos en un registro)

1. de operandos:

ADD BX, PEPE ; sumar contenidos de reg BX y dirección PEPE, el resultado se guarda en reg BX

**Elementos de una instrucción**

* Código de operación (“Cod Op”)
* Referencia a operandos fuentes
* Referencia al operando resultado
* Referencia a la siguiente instrucción

**¿Dónde se almacenan operandos?**

* Memoria principalo memoria virtual o en memoria cache
* Registro de la CPU
* Dispositivo de E/S

**Tipos de instrucciones**

* **Instrucciones aritmético:** Operaciones básicas: Add, Substract, Multiply y Divide. Números enteros sin/con signo. Pueden incluirse otras operaciones: Increment o Decrement (en 1 el operando). Negate: cambia el signo del operando (Ca2). Absolute: toma el valor absoluto del operando. Shift left/right: desplaza bits a izq/der un lugar
* **Instrucciones lógicas:** Operaciones que manipulan bits individualmente. Operaciones Booleanas. AND, OR, XOR, NOT. Rotate left/right: rota las posiciones de los bits a izq/der
* **Almacenamiento de datos:** instrucciones de memoria
* **Transferencia de datos:** Debe especificarse: Ubicación del operando fuente. Ubicación del operando destino. Tamaño de los datos a ser transferidos.

Modo de direccionamiento: Reg-Reg, Reg-Mem o Mem-Reg o una instrucción y diferentes direcciones: MOV destino, fuente ; copia fuente a destino

* **instrucciones de E/S:** Pocas instrucciones pero de acciones específicas IN ó OUT. Se pueden realizar utilizando instrucciones de movimiento de datos. MOVE.

Se pueden realizar a través de un controlador aparte: DMA (Direct Memory Access)

* **Control:** instrucciones de testeo y flujo del programa. Modifican el valor contenido en el registro PC.

Salto Incondicional: JMP equis ; saltar a la posicion ‘equis’

Salto Condicional: JZ equis ; saltar a la posición ‘equis’, si bandera Z=1

Salto con retorno o llamada a subrutina: CALL subrut ;saltar a la posición ‘subrut’

Para retornar al programa que llamó, se debe utilizar la instrucción RET como última instrucción del cuerpo de subrutina

**Decisiones en el diseño del conjunto de instrucciones**

* Tipos de operandos (datos)
* Repertorio de operaciones: ¿Cuántas operaciones se considerará? ¿Cuáles operaciones se realizarán? ¿Cuán compleja será cada una de ellas?
* Formatos de instrucciones: Longitud de instrucción, Número de direcciones, Tamaño de los campos.
* Registros: Número de registros de la CPU referenciables. ¿En qué registros se pueden ejecutar qué operaciones?
* Modos de direccionamiento: ¿cómo es especificada la ubicación de un operando o una instrucción?

**Modos de direccionamiento:**

* Inmediato
* Directo de memoria o Absoluto
* Directo de Registro
* Indirecto de memoria (en desuso)
* Indirecto con registro
* Indirecto con Desplazamiento
* basado, indexado o relativo al PC
* Pila (o relativo al SP)

**Pasaje de argumentos a subrutinas**

* **Vía registros:** El número de registros es la principal limitación. Es importante documentar que registros se usan. Los registros temporales se usan para intercambiar parámetros y resultados con el siguiente nivel más bajo (el procedimiento llamado por el procedimiento en curso). Los registros temporales de un nivel son físicamente los mismos que los registros de parámetros del nivel más abajo adyacente. Este solapamiento posibilita que los parámetros se pasen sin que exista una transferencia de datos real.
* **Vía memoria:** Se usa un área definida de memoria (RAM). Difícil de estandarizar.
* **Vía pila (stack):** Es el método más ampliamente usado. El verdadero “pasaje de parámetros”. Independiente de memoria y registros. Hay que comprender bien cómo funciona porque la pila (stack) es usada por el usuario y por el sistema.

En x86, SP apunta al último lugar usado

**Funcionamiento de una pila:** El operando está (de forma implícita) en la cabeza de la pila. Se requiere un registro Puntero de Pila (SP). Contiene la dirección de la cabeza de la pila. Operaciones sobre la pila: **PUSH**: operación de Apilar. **POP**: operación de desapilar. Son inversas entre sí.

El manejo de la pila para anidamiento de subrutinas es el siguiente:

Antes de llamar a la subrutina se debe apilar los parámetros a pasar y la dirección de retorno, al llamar a la subrutina de debe:

1. Salvar el estado de BP (viejo BP), es decir apilar el valor del BP
2. Cargar el valor del BP el del SP
3. Reservar espacio para datos locales, si los hay
4. Salvar valores de otros registros, si es que se van a modificar los registros
5. Acceder a parámetros, para acceder a los parámetros se le debe sumar un desplazamiento al BP para acceder a la posición de la pila en la que están los parámetros. En general el desplazamiento es: 2 (es el tamaño de BP apilado) + tamaño de dirección de retorno + total de tamaño de parámetros entre el buscado y BP
6. Escribir sentencias a ejecutar.

Aquí se puede llamar a otra subrutina o regresar a la anterior, si se vuelve a hacer una llamada se debe apilar los parámetros a pasar a la subrutina y la dirección de retorno. La subrutina llamada debe repetir los pasos anteriores más los que siguen. Sino se hace otra llamada a subrutina también se deberá seguir los siguientes pasos:

7.Retornar parámetro, si se tiene que retornar datos

8.Regresar correctamente del procedimiento (desapilar todo lo que apilo, volver a cargar a cargar el valor del BP que tenía antes de entrar a la subrutina

**Salida del procedimiento**

* Los registros salvados en la pila deben ser descargados en orden inverso.
* Si se reservó espacio para variables locales, se debe reponer SP con el valor de BP que no cambió durante el procedimiento.
* Reponer BP.
* Volver al programa que llamó al procedimiento con RET.

**Ciclo de instrucción**

* **Captación:** La dirección de la instrucción que se debe captar se encuentra en el registro Contador de Programa (PC). La UC capta la instrucción desde la Memoria. La instrucción va al registro de instrucción (IR). El registro PC se incrementa (a no ser que se indique lo contrario). La UC interpreta la instrucción captada y debe lleva a cabo la acción requerida
* **Ejecución:** Acciones posibles: Procesador – memoria. Transferencia de datos CPU - Memoria. Procesador - E/S. Transferencias de datos CPU y módulo de E/S. Procesamiento de datos: Alguna operación aritmética o lógica con los datos. Control: Alteración de la secuencia de ejecución. (Instrucción de salto o combinación de las acciones anteriores)
* **Ciclo de interrupción:** Se comprueba si se ha solicitado alguna interrupción (indicada por una señal (flag) de pedido de interrupción). Si no hay señal se capta la siguiente instrucción. Si hay algún pedido de interrupción pendiente: Se suspende la ejecución del programa en curso. Guarda su contexto (próxima instrucción a ejecutar y el estado del procesador). Carga el PC con la dirección de comienzo de una rutina de gestión de interrupción. Se inhiben otras interrupciones. Finalizada la rutina de gestión, el procesador retoma la ejecución del programa del usuario en el punto de interrupción.

**Interrupciones:** Mecanismo mediante el cual se puede interrumpirel procesamiento normal de la CPU (ejecución secuencial de instrucciones de un programa). Pueden ser de origen interno o externo a la CPU.

*¿Porqué Interrumpir?*

Por resultado de una ejecución de una instrucción (Ej: desbordamiento aritmético (“overflow”), división por cero). Por un temporizador interno del procesador (permite al S.O. realizar ciertas funciones de manera regular). Por una operación de E/S (Ej: para indicar la finalización normal de una operación). Por un fallo de hardware (Ej: error de paridad en la memoria, pérdida de energía).

*¿Qué hacer si interrumpen?*

En casi todos los casos, implica transferir el control a otro programa (el GESTOR) que:

* salve el estado del procesador corrija (o responda a) la causa que ocasionó la interrupción
* restaure el estado original del procesador
* retorne a la ejecución normal del programa interrumpido

**Jerarquía de interrupciones**

Si hay múltiples fuentes que pueden solicitar interrupción se establece cuáles son más importantes. Ser consideran:

* No Enmascarables: las que NO pueden ignorarse. Indican eventos peligrosos o de alta prioridad.
* Enmascarables: pueden ser ignoradas. Con instrucciones podemos inhibir la posible solicitud.

Tipos de Interrupción / Fuentes de interrupción:

**Interrupciones por hardware:** Son las generadas por dispositivos de E/S. Son las “verdaderas” interrupciones.

* El sistema de cómputo tiene que manejar estos eventos externos “no planeados” ó “asincrónicos”.
* No están relacionadas con el proceso en ejecución en ese momento.
* Son conocidas como interrupt request.

**Traps/excepciones:** Interrupciones por hardware creadas por el procesador en respuesta a ciertos eventos como:

* Condiciones excepcionales: overflow en ALU de punto flotante.
* Falla de programa: tratar de ejecutar una instrucción no definida.
* Fallas de hardware: error de paridad de memoria.
* Accesos no alineados ó a zonas de memoria protegidos

**Interrupciones por software:** Muchos procesadores tienen instrucciones explícitas que afectan al procesador de la misma manera que las interrupciones por hardware.

* Generalmente usadas para hacer llamadas a funciones del SO.
* Esta característica permite que las subrutinas del sistema se carguen en cualquier lugar.
* No requieren conocer la dirección de la rutina en tiempo de ejecución.

**Interrupciones múltiples**

* *Interrupciones inhabilitadas:* • El procesador puede y debe ignorar la señal de petición de interrupción si se produce una interrupción en ese momento. • Si se hubiera generado una interrupción se mantiene pendiente y se examinará luego una vez que se hayan habilitado nuevamente. • Ocurre una interrupción, se inhabilitan, se gestiona la misma y luego se habilitan otra vez. • Por lo tanto las interrupciones se manejan en un orden secuencial estricto.
* *Definir prioridades:* • Una interrupción de prioridad más alta puede interrumpir a un gestor de 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.

**Reconocimiento de interrupciones**

* *Interrupciones multinivel:* Cada dispositivo que puede provocar interrupción tiene una entrada física de interrupción conectada a la CPU. Es muy sencillo, pero muy caro.
* *Línea de interrupción única:* Una sola entrada física de pedido de interrupción a la que están conectados todos los dispositivos. Se debe “preguntar” a cada dispositivo si ha producido el pedido de interrupción (técnica Polling/encuesta).
* *Interrupciones vectorizadas:* El dispositivo que quiere interrumpir además de la señal de pedido de interrupción, debe colocar en el bus de datos un identificador (vector). Lo coloca el periférico directamente o Controlador de Interrupciones (que se ocupa de todo).

**Controlador de interrupciones PIC:**

Un PIC es un dispositivo controlador programable de instrucciones. Es un dispositivo usado para combinar varias fuentes de interrupciones sobre una o más líneas del CPU, mientras que permite que los niveles de prioridad sean asignados a sus salidas de interrupción. Cuando el dispositivo tiene múltiples salidas de interrupción a imponer, las impondrá en orden de su prioridad relativa. Los modos comunes de un PIC incluyen prioridades duras, prioridades rotativas, y prioridades en cascada. Los PICs a menudo permiten la conexión en cascada de sus salidas a las entradas entre uno y otro.

Los PICs típicamente tienen un conjunto común de registros: Interrupt Request Register (IRR), In-Service Register (ISR), Interrupt Mask Register (IMR). El IRR especifica qué interrupciones están pendientes de reconocimiento, y es típicamente un registro interno que no puede ser accedido directamente. El registro ISR especifica qué interrupciones han sido reconocidas, pero todavía están esperando por un final de interrupción (EOI). El IMR especifica qué interrupciones deben ser ignoradas y no ser reconocidas. Un esquema simple de registros como este, permite hasta dos distintas peticiones de interrupción estén pendientes a un tiempo, una esperando por reconocimiento, y una esperando por EOI.

**Registros internos PIC**

• EOI: para comandos • Para fin de int escribir 20H • IMR: máscara de int

• enmascara con ‘1’ • IRR: petición de int • Indica con bit en 1 • ISR: int en servicio

• Indica con bit en 1 • INT0...INT7 • c/u con su vector

**Los registros internos del PIC se sitúan a partir de la dirección 20H.**

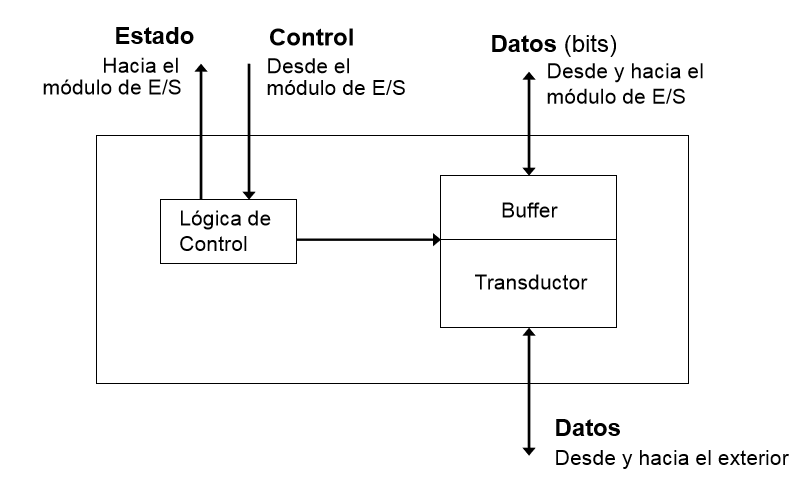
• Son accedidos con operaciones lectura y escritura en el espacio de E/S (IN y OUT).

• Interrupciones hardware asignadas • INT0 – tecla F10 • INT1 – Timer • INT2 – Handshake

• INT3 – DMA • INT4 a INT7 no usadas

**Módulo de E/S:** Realiza la interfaz entre el procesador y la memoria (bus) y los periféricos. Pueden manejar uno o más periféricos.

**Dispositivo externo tipo**



Un módulo de entrada/salida permite que el procesador vea a una amplia gama de dispositivos externos de manera simplificada. El módulo debe ocultar los detalles de temporización, electromecánica y formatos de los dispositivos, para que el procesador pueda funcionar en términos de órdenes de lectura y escritura.

El módulo se conecta con el resto de la máquina a través de líneas, como pueden ser las líneas del bus del sistema. Los datos que se transfieren a y desde el módulo, se almacenan temporalmente en registros. Además, puede haber registros de estado que proporcionan información del estado actual. Un registro de estado también puede funcionar como información de control, para recibir información de control del procesador. La lógica que hay en el módulo interactúa con el procesador a través de una serie de líneas de control, usadas por el procesador para proporcionar las órdenes al módulo de E/S. El módulo también debe ser capaz que reconocer y generar las direcciones asociadas a los dispositivos que controla. Cada módulo E/S tiene una dirección única, y si controla más de un dispositivo, tiene un conjunto de direcciones únicas. Cada módulo posee la lógica específica para cada una de las interfaces de los dispositivos que controla.

**Capacidades de un módulo de E/S**

• Ocultar las propiedades del dispositivo a la CPU. • Ocuparse de uno o varios dispositivos.

• Controlar o no las funciones del dispositivo. • Canales de E/S o procesador de E/S (manejo de parte importante de la carga del procesamiento). Presentes en Mainframes. • Controlador de E/S o controlador de dispositivo (manejo primitivo). Presentes en microcomputadoras

**Operación de Entrada o Salida -** Requiere:

* Direccionamiento:
* E/S asignada en memoria (memory-mapped): Dispositivos de E/S y memoria comparten un único espacio de direcciones. E/S se parece a la memoria de lectura/escritura. No hay órdenes específicas para E/S. Variedad de órdenes de acceso a memoria (programación eficiente).
* E/S aislada: Espacios de direcciones separados. Necesidad de líneas especiales de E/S y de memoria. Órdenes específicas para E/S. Conjunto limitado de instrucciones.
* Transferencia de información: Lectura o escritura
* Gestión de la transferencia: Mecanismos de sincronización y control de la transferencia de datos

**Técnicas de gestión de E/S**

* *E/S Programada con espera de respuesta*

Intercambio de datos entre la CPU y el módulo. La CPU tiene control directo sobre la operación de E/S. Comprobación del estado del dispositivo. Envío de comandos de lectura/escritura. Transferencia de datos. La CPU espera que el módulo E/S termine la operación. Por lo tanto la CPU permanece ociosa durante un período de tiempo (no deseable).

*Detalle:* La CPU solicita la operación de E/S al módulo. El módulo E/S realiza la operación. El módulo E/S activa los bits de estado del dispositivo direccionado y espera. La CPU comprueba periódicamente el estado de esos bits, hasta que detecta que la operación fue completada. En caso contrario la CPU espera y vuelve a comprobarlo más tarde.

*La CPU da una orden:* • Control: indica al módulo qué hacer • Test: comprueba el estado del módulo y sus periféricos • Lectura/Escritura: Transfiere datos desde o hacia el dispositivo por el bus de datos.

* *E/S con interrupciones*

La CPU no tiene que esperar la finalización de la tarea de E/S, puede seguir procesando. No se repite la comprobación de los estados de los módulos. El módulo envía un pedido de interrupción a la CPU cuando está listo nuevamente.

*E/S mediante interrupciones ¿qué hace la CPU?*

La CPU envía una orden de lectura (READ). El módulo E/S obtiene los datos del periférico mientras que la CPU realiza otro trabajo. La CPU chequea si hay pedidos de interrupciones pendientes al final de cada ciclo de instrucción. El módulo E/S emite un pedido de interrupción a la CPU. La CPU detecta el pedido, guarda el contexto, interrumpe el proceso y realiza la gestión de la interrupción. La CPU solicita los datos. El módulo E/S transfiere los datos.

*Interrupciones múltiples*

Todas 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.

* *E/S con acceso directo a memoria (DMA)*

El controlador de DMA es un dispositivo capaz de controlar una transferencia de datos entre un periférico y memoria sin intervención de la CPU.

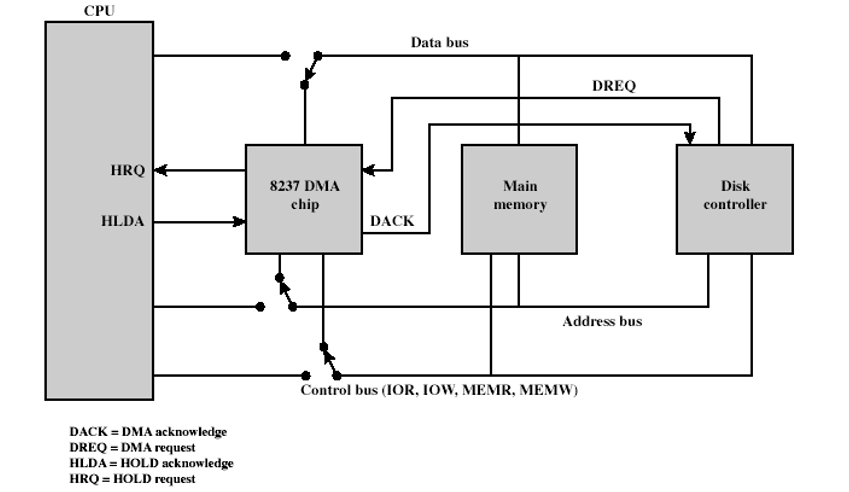
DMA (Direct Memory Access) es un módulo adicional en el bus de sistema, que tiene la capacidad de imitar al procesador y tomar el control del sistema, con la función de realizar transferencias de E/S de datos, de manera directa entre los periféricos y la memoria, a través del bus del sistema. Para hacerlo, el módulo de DMA debe utilizarlo solo cuando el procesador no lo necesita, o debe forzar al procesador a que suspenda temporalmente su funcionamiento. Esta última técnica es la más común, y se denomina **robo de ciclo**, puesto que, en efecto, el módulo de DMA roba un ciclo de bus.

Cuando el procesador desea leer o escribir un bloque de datos, envía una orden al módulo de DMA, incluyendo la siguiente información:

* Si se solicita una lectura o escritura, usando la línea de control de R/W entre el procesador y el módulo DMA
* La dirección del dispositivo de E/S en cuestión, indicada a través de las líneas de datos
* La posición inicial de memoria a partir de donde se lee o se escribe, indicada a través de las líneas de datos y almacenada por el módulo de DMA en su registro de direcciones
* El número de palabras a leer o escribir también indicado a través de las líneas de datos y almacenada en

el registro de cuenta de datos.

Después el procesador continúa con otro trabajo. Ha delegado la operación de E/S al módulo de DMA que se encargara de ella. EL módulo de DMA transfiere el bloque completo de datos, palabra a palabra directamente desde o hacia la memoria, sin que tenga que pasar a través del procesador. Cuando la transferencia ha terminado, el módulo de DMA envía una señal de interrupción al procesador.

**Bus del sistema y DMA**

**Problema que puede haber**

Se puede degradar el rendimiento de la CPU si el DMAC hace uso intensivo del bus: Si el bus está ocupado en una transferencia DMA, la CPU no puede acceder a memoria para leer instrucc./datos

*El problema se reduce con el uso de memoria cache*

* La mayor parte del tiempo, la CPU lee instruc. de la cache, por lo que no necesita usar el bus d 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 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 para realizar sus transferencias.

**Tipos de transferencias**

Si el DMAC sólo toma el control del bus durante los intervalos de tiempo en los que la CPU no hace uso del mismo el rendimiento del sistema no sufrirá degradación alguna.

**Se distinguen dos tipos de transferencias:**

*Por ráfagas (burst)*

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 rápida.
* 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.

*Por robo de ciclo (cycle-stealing)*

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.

* VENTAJAS: No se degrada el rendimiento del sistema.
* DESVENTAJAS: La transferencia tarda más tiempo en llevarse a cabo.

Para la CPU no es una interrupción: El procesador no debe guardar el contexto.

Si bien el trabajo de la CPU es lento, no será tanto como si ella realizara la transferencia. Por lo tanto, para transferencia de E/S de múltiples palabras, es la técnica más eficiente.

**Canales de E/S**

Representan una extensión al concepto de DMA. Tienen la habilidad de ejecutar instrucciones de E/S. Completo control de la transferencia de datos, por lo tanto, la CPU no ejecuta instrucciones de E/S. Programa almacenado en memoria principal. La CPU inicia la transferencia de E/S

**Tipos de canales de E/S**

*Selector*

Controla varios dispositivos de alta velocidad y uno por vez, por lo tanto, el canal se dedica para la transferencia de datos de ese dispositivo. El canal selecciona un dispositivo y efectúa la transferencia. Los dispositivos son manejados por un controlador o módulo de E/S. Por lo tanto, el canal de E/S ocupa el lugar de la CPU en el control de esos controladores.

*Multiplexor*

Puede manejar E/S con varios dispositivos a la vez. Multiplexor de bytes: Acepta y transmite caracteres. *Multiplexor de bloques*: Intercala bloques de datos desde distintos dispositivos.

**Segmentación de cauce:**

La segmentación de cauce o “pipelining” es una forma de organizar el hardware de la CPU para realizar más de una operación al mismo tiempo. Consiste en descomponer el proceso de ejecución de las instrucciones en fases o etapas que permitan una ejecución simultánea. Explota el paralelismo entre las instrucciones de un flujo secuencial. La segmentación es invisible al programador. Necesidad de uniformizar las etapas al tiempo de la más lenta. El diseño de procesadores segmentados tiene gran dependencia del repertorio de instrucciones.

**Diagrama de estados del ciclo de instrucción**

**Tareas a realizar por ciclo**

* *Búsqueda (F, Fetch ):* Se accede a memoria por la instrucción. Se incrementa el PC.
* *Decodificación (D, Decode ):* Se decodifica la instrucción, obteniendo operación a realizar en la ruta de datos. Se accede al banco de registros por el/los operando/s (si es necesario). Se calcula el valor del operando inmediato con extensión de signo (si hace falta).
* *Ejecución (X, Execute ):* Se ejecuta la operación en la ALU.
* *Acceso a memoria (M, Memory Access ):* Si se requiere un acceso a memoria, se accede.
* *Almacenamiento (W, Writeback ):* Si se requiere volcar un resultado a un registro, se accede al banco de registros.

**Prestaciones del cauce segmentado**

*Teórica*

El máximo rendimiento es completar una instrucción con cada ciclo de reloj. Si K es el número de etapas del cauce 🡪 Vel. procesador segmentado = Vel. secuencial x K

El incremento potencial de la segmentación del cauce es proporcional al número de etapas del cauce. Incrementa la productividad (*throughput*), pero no reduce el tiempo de ejecución de la instrucción.

**Atascos de un cauce (stall)**

Situaciones que impiden a la siguiente instrucción que se ejecute en el ciclo que le corresponde.

1. ***Estructurales:*** Provocados por conflictos por los recursos

*Riesgos estructurales:* Dos o más instrucciones necesitan utilizar el mismo recurso hardware en el mismo ciclo.

*Soluciones a riesgos estructurales:*

Simple: Replicar, segmentar o realizar turnos para el acceso a las unidades funcionales en conflicto. Duplicación de recursos hardware (sumadores o restadores además de la ALU). Separación en memorias de instrucciones y datos. Turnar el acceso al banco de registros: Escrituras en la 1º mitad de los ciclos de reloj, lecturas en la 2º mitad de los ciclos de reloj.

1. ***Por dependencia de datos:*** Ocurren cuando dos instrucciones se comunican por medio de un dato (ej.: una lo produce y la otra lo usa)

*Riesgos por dependencias de datos:* Condición en la que los operandos fuente o destino de una instrucción no están disponibles en el momento en que se necesitan en una etapa determinada del cauce.

*Tipos de dependencias de datos:*

* Lectura después de Escritura (RAW, dependencia verdadera): una instrucción genera un dato que lee otra posterior.
* Escritura después de Escritura (WAW, dependencia en salida): una instrucción escribe un dato después que otra posterior, sólo se da si se deja que las instrucciones se adelanten unas a otras
* Escritura después de Lectura (WAR, antidependencia): una instrucción modifica un valor antes de que otra anterior que lo tiene que leer, lo lea, no se puede dar en nuestro cauce simple.

*Soluciones a riesgos de datos*:

**Para riesgos RAW:** se debe determinar cómo y cuando aparecen esos riesgos. Será necesario: Unidad de detección de riesgos y/o compilador más complejo

Dos soluciones:

*Técnica hardware Conocida como Adelantamiento, Forwarding o Cortocircuito:* Consiste en pasar directamente el resultado obtenido con una instrucción a las instrucciones que lo necesitan como operando. Si el dato necesario está disponible a la salida de la ALU (Xi) se lleva a la entrada de la etapa correspondiente (Xi+1) sin esperar a la escritura (Mi o Wi). Fácil de implementar si se identifican todos los adelantamientos y se comunican a los registros de segmentación correspondientes.

*Técnica software:* Instrucciones NOP o reordenación de código. evita los riesgos reordenando las instrucciones del código sin afectar los resultados Realizada por el compilador. Introducción de instrucciones NOP. Se genera Retardo. Reordenación de instrucciones. Máxima separación de instrucciones con dependencia RAW. Cuidado con ejecución “fuera de orden”.

1. ***Por dependencia de control:*** Ocurren cuando la ejecución de una instrucción depende de cómo se ejecute otra (ej.: un salto y los 2 posibles caminos)

*Riesgos de control (o de instrucciones):* Una instrucción que modifica el valor del PC no lo ha hecho cuando se tiene que comenzar la siguiente.

*Soluciones a riesgos de control:*

Existe una Penalización por salto. Instrucciones de salto:

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

*Condicional:* Introduce riesgo adicional por la dependencia entre la condición de salto y el resultado de una instrucción previa.

Modificación sencilla de la ruta de datos para reducir la cantidad de paradas a un solo ciclo.

Adelantar la resolución de los saltos a la etapa D: En ella se decodifica y se sabe que es un salto. Se puede evaluar la condición de salto (con restador). Se puede calcular la dirección de salto (con sumador).

**Para tratamiento de saltos hay**

***Técnica Hardware***

Predicción de saltos para evitar la parada:

*Técnicas estáticas:*

* Predecir que nunca se salta: Asume que el salto no se producirá. Siempre capta la siguiente instrucción.
* Predecir que siempre se salta: Asume que el salto se producirá. Siempre capta la instrucción destino dl salto.

*Técnicas dinámicas:*

* Conmutador saltar/no saltar: Basado en la historia de las instrucciones. Eficaz para los bucles.
* Tabla de historia de saltos (branch-target buffer). Pequeña cache asociada a la etapa de búsqueda (F)

Tres campos: Dirección de una instrucción de bifurcación. Información de la instrucción destino (dirección del destino o Instrucción destino). N bits de estado (historia de uso)

*Otras soluciones hardware:* Predecir según el código de operación (hay instrucciones con más probabilidades de saltar, la tasa de acierto puede llegar a alcanzar un 75%).

* Flujos múltiples: Varios cauces (uno por cada opción de salto). Precaptan cada salto en diferentes cauces. Se debe utilizar el cauce correcto. *Desventajas:* Provoca retardos en el acceso al bus y a los registros. Si hay múltiples saltos, se necesita un mayor número de cauces.
* Precaptar el destino del salto: Se precapta la instrucción destino del salto, además de las instrucciones siguientes a la bifurcación. La instrucción se guarda hasta que se ejecute la instrucción de bifurcación.
* Buffer de bucles: Memoria muy rápida. Gestionada por la etapa de captación de instrucción del cauce. Comprueba el buffer antes de hacer la captación de memoria. Muy eficaz para pequeños bucles y saltos.

***Técnica Software***

*Salto retardado o de relleno de ranura de retardo:* El compilador introduce instrucciones que se ejecutarán en cualquier caso después de la instrucción de salto.

*Idea:* Realizar trabajo útil mientras el salto se resuelve.

Hueco o ranura de retardo de salto (delay-slot) es el período de penalización o parada luego de una instrucción de salto. El compilador trata de situar instrucciones útiles (que no dependan del salto) en los huecos de retardo. Si no es posible, se utilizan instrucciones NOP. Las instrucciones en los huecos de retardo de salto se captan siempre. Requiere reordenar las instrucciones.

RISC: Computadoras de repertorio reducido de instrucciones (Reduced Instruction Set Computer). Tienen un

gran número de registros de uso general, el repertorio de instrucciones es limitado y sencillo (“reducido”), y se hace un gran énfasis en la optimización de la segmentación de instrucciones.

CISC: (Complex Instruction Set Computer) en cambio, tiene un repertorio de instrucciones más amplio, con la

intención de facilitar el trabajo de los escritores de compiladores. También pretenden mejorar la eficiencia de la ejecución, a través de secuencias complejas de operaciones en micro código, y dar soporte a lenguajes de alto nivel (HLL) más complejos.

El razonamiento es que, si hay instrucciones de máquina que se parezcan a las de HLL, la tarea se simplifica. El problema con esto es que las instrucciones de máquina complejas suelen ser difíciles de aprovechar, ya que el compilador debe descubrir los casos que se ajusten perfectamente.

La tarea de optimizar el código generado para minimizar su tamaño, reducir el número de instrucciones

ejecutadas y mejorar la segmentación es mucho más difícil con un repertorio complejo de instrucciones.

*Inconvenientes del CISC*

* + El software ‘es’ mucho más caro que el hardware.
  + El nivel del lenguaje era cada vez más complicado.
  + Salto semántico: Diferencias entre operaciones HLL y operaciones de la Arquitectura
  + Todo esto conduce a:
    1. Repertorios de instrucciones grandes
    2. Más modos de direccionamiento
    3. Varias sentencias de HLL implementadas en el Hardware. Por ejemplo, el CASE del VAX

*Finalidad del CISC*

* + Facilitar el trabajo del escritor de compiladores.
  + Mejorar la eficiencia de la ejecución: Secuencias complejas de operaciones en microcódigo.
  + Dar soporte a HLL más complejos.

*Características de la ejecución*

Estudios sobre programas escritos en HLL

* + Operaciones realizadas
  + Funcionamiento del procesador e interacción con memoria
  + Operandos usados
  + Tipos y frecuencia de uso
  + Organización de la memoria y modos de direccionamiento
  + Secuenciamiento de la ejecución
  + Organización del control y del cauce

Estudios dinámicos: medir durante la ejecución

*Operaciones*

Asignaciones:

Movimiento de datos.

Estamentos condicionales (IF, LOOP):

Control secuencial.

El procedimiento llamada/retorno consume mucho tiempo.

Algunas instrucciones HLL conducen a muchas operaciones de código máquina.

*Operandos*

* + Principalmente variables escalares locales.
  + La optimización debe concentrarse en el acceso a la variables locales.

*Llamadas a procedimientos*

* + Se consume mucho tiempo.
  + Depende del número de parámetros tratados.
  + Depende del nivel de anidamiento.
  + La mayoría de los programas no tienen una larga secuencia de llamadas seguida por la correspondiente secuencia de retornos.
  + La mayoría de las variables son locales.
  + Las referencias a operandos están muy localizadas. Ventana de profundidad

*Consecuencias*

* Se puede ofrecer mejor soporte para los HLL optimizando las prestaciones de las características más usadas y que más tiempo consumen.
* Usar un gran número de registros:
* Optimizar las referencias a operandos
* Prestar cuidadosa atención al diseño de los cauces de instrucciones:
  1. Predicción de bifurcaciones, etc.
  2. Es recomendable un repertorio con instrucciones simples (reducido)

¿Por qué CISC?

* ¿Simplificación del compilador?
* Ésta primera razón parece obvia
* Instrucciones de máquina complejas son difíciles de aprovechar
* La optimización es más difícil: tamaño, velocidad.
* ¿Programas más pequeños?
* El programa ocupa menos memoria, pero la memoria hoy día es muy barata.
* El número de bits de memoria que ocupa no tiene porqué ser más pequeño al tener menos instrucciones
* Más instrucciones necesitan códigos de operación más largos.
* Las referencias a registros necesitan menos bits.
* ¿Programas más rápidos?
* Propensión a usar las instrucciones más sencillas.
* Unidad de control más compleja.
* Memoria de control del microprograma más grande.
* Aumenta el tiempo de ejecución de las instrucciones simples.
* No está nada claro que la tendencia hacia CISC fuera la apropiada.

*RISC frente a CISC*

* + No existe una clara barrera diferenciadora.
  + Muchos diseños incluyen características de ambos criterios.
  + Por ejemplo, PowerPC y Pentium II

*Características de RISC:*

Una instrucción por ciclo: Se ejecuta una instrucción máquina cada ciclo máquina. Con instrucciones sencillas y de un ciclo, hay poca o ninguna necesidad de microcódigo, las instrucciones de maquina pueden estar

cableadas, lo que significa que no hay que acceder a la memoria de control de microprograma durante la

ejecución de la instrucción.

Operaciones registro a registro: La mayoría de las operaciones deben ser de esta manera, lo que simplifica el

repertorio de instrucciones y fomenta la optimización del uso de los registros, ya que los operandos accedidos frecuentemente, permanecen en el almacenamiento de alta velocidad.

Modos de direccionamiento sencillos: se usa principalmente el sencillo direccionamiento a registro

Formatos de instrucción sencillos: Sólo se usa un formato, o unos pocos. La longitud de las instrucciones es fija, y alineada en los límites de una palabra. Las posiciones de los campos, especialmente la del código de operación, son fijas. Todo esto simplifica la unidad de control

*Controversia RISC y CISC*

* + Cuantitativa: Comparación del 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.

*Problemas de las comparaciones:*

* No existe un par de máquinas RISC y CISC directamente comparables.
* No hay un conjunto de programas de prueba definitivo.
* Dificultad para separar los efectos del hardware de los del compilador.
* Mayoría de comparaciones con máquinas de “juguete”, no con productos comerciales.
* La mayoría de las máquinas son una mezcla de ambas

*Jerarquía de memoria*

* organizada en niveles que son ubicados en distintos lugares físicos
  + fabricados con tecnologías diferentes que se gestionan de manera independiente

*Objetivo:* la velocidad del sistema deberá ser, aproximadamente, la del nivel más rápido al costo del nivel más barato.

A medida que nos alejamos de la CPU, cada nivel inferior es más grande, más lento y más barato que el nivel previo (o superior) en la jerarquía. Debe haber correspondencia de direcciones en los distintos niveles.

*Principio de localidad de referencias*

* Localidad Temporal: los elementos de memoria referenciados recientemente (datos o instrucciones), volverán a serlo en un futuro próximo 🡪 subo la palabra de nivel
* Localidad Espacial: los elementos de memoria cuyas direcciones están próximas a los últimos referenciados serán referenciados. 🡪 subo un bloque (con la palabra) de nivel

Memoria Cache

* Cantidad pequeña de memoria rápida.
* Se ubica entre la memoria principal y la CPU.
* Puede localizarse en un chip o en módulo CPU.

*Funcionamiento de la cache*

* La CPU solicita contenido de 1 dirección de memoria.
* La cache ¿tiene ese dato?
* Si es así, la obtiene de la cache (rápidamente).
* Si no está, se lee el bloque que contiene esa dirección desde la memoria principal y copia en la cache.
* Después, la cache entrega el dato requerido a la CPU.

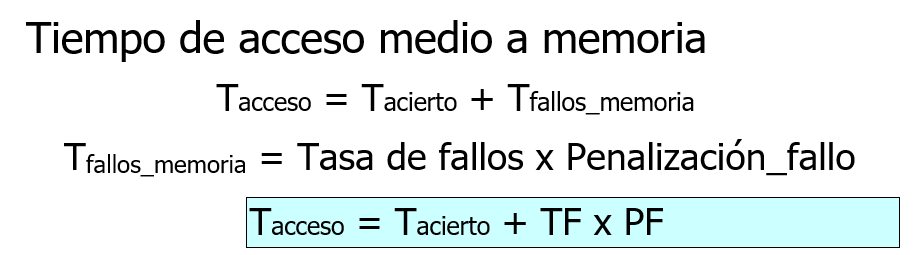
La cache incluye etiquetas para identificar qué bloque de la memoria principal está en cada una de sus líneas.

*Conceptos básicos*

**Acierto (hit):** se encuentra en la caché el dato solicitado

**Fallo (miss):** no se encuentra en la caché el dato solicitado

* un bloque que contiene la palabra accedida se copia de la memoria principal a una línea de caché.
* Tiempo para servir un fallo: depende de la latencia y ancho de banda de la memoria principal.
* Latencia: tiempo necesario para completar un acceso a memoria.
* Ancho de banda: cantidad de información por unidad de tiempo que puede transferirse desde/hacia la memoria.
* Los fallos de caché se gestionan mediante hardware y causan que el procesador se detenga hasta que el dato esté disponible.



*Para mejorar las prestaciones*

* Reducir el tiempo en caso de acierto (Tacierto)
* Reducir la tasa de fallos (TF)
* Reducir la penalización por fallo (PF)

*Los principales elementos a tener en cuenta para armar una Cache son:*

* Tamaño de caché: Se necesita hacer un balance entre el tamaño (mientras más grande es más costoso) y la performance, que lo ideal es tener mucho tamaño, pero si son muy grandes tienden a ser ligeramente más lentas. También está limitado por la superficie disponible de chip y de tarjeta.
* Función de correspondencia: Son directa, asociativa y asociativa por conjunto.
* *La correspondencia Directa* es la más sencilla. Consiste en hacer corresponder cada bloque de la memoria principal a solo una línea posible de la cache.

i = j mod m; donde i es el número de línea de caché, j el número de bloque de memoria principal, m el número de líneas de la caché.

La **ventaja** es que es muy sencilla de implementar. La **desventaja** es que hay una posición concreta de chache para cada bloque dado. Por ello, si un programa referencia repetidas veces a palabras de los bloques diferentes asignados en la misma línea dichos bloquean se estarían intercambiando continuamente en la caché, y la tasa de aciertos sería baja.

* *La correspondencia asociativa* supera la desventaja de la directa, permitiendo que cada bloque de memoria principal pueda cargarse en cualquier línea de la cache. La lógica de control de la cache interpreta una dirección de memoria simplemente como una etiqueta y un campo de palabra. El campo de etiqueta identifica inequívocamente un bloque de memoria principal. Para determinar si un bloque está en la cache, su lógica de control debe examinar simultáneamente todas las etiquetas de líneas para buscar una coincidencia. La desventaja es la compleja circuitería necesaria para examinar en paralelo las etiquetas de todas las líneas de cache.

**Ventajas y desventajas**

* Un bloque de memoria principal puede colocarse en cualquier línea de la cache.
* La etiqueta identifica unívocamente un bloque de memoria.
* Todas las etiquetas de las líneas se examinan para buscar una coincidencia.
* Búsqueda costosa (en tiempo principalmente).
* *La correspondencia asociativa*: Un bloque puede almacenarse en un conjunto restringido de lugares en la caché.

**Ventajas y desventajas**

* Combina lo mejor de las otras correspondencias
* La cache se divide en un grupo de conjuntos.
* Cada conjunto contiene un número de líneas
* N vías, con N=2, 4, 8 ... etc.
* Un bloque determinado corresponderá a cualquier línea de un conjunto determinado.
* El bloque B puede asignarse en cualquiera de las líneas del conjunto i.

*Política de reemplazos -* Algoritmos de sustitución

* **En correspondencia directa:** el que ocupa el lugar del nuevo. No hay elección. Sólo hay una posible línea para cada bloque. Se necesita una sustitución de esa línea (si o sí).
* **En correspondencia asociativa:** Los algoritmos deben implementarse en hardware (para conseguir velocidad).
* LRU (menos recientemente usado): Requiere controles de tiempos. En correspondencias asociativas por conjuntos de 2 vías.
* FIFO (más antiguo): Requiere controles de acceso. Se sustituye aquella línea que ha estado más tiempo en la cache.
* LFU (menos frecuentemente usado): Requiere controles de uso. Se sustituye aquella línea que ha experimentado menos referencias.
* Aleatoria: Se sustituye una línea al azar.

***Política de escritura***

Se debe evitar inconsistencia de memorias en el caso de escrituras.

Tener en cuenta:

* + La CPU escribe sobre una línea de cache
  + El bloque de memoria principal correspondiente debe ser actualizado en algún momento.
  + Un módulo E/S puede tener acceso directo a la memoria principal.
  + En procesamiento paralelo, las múltiples CPU pueden tener caches individuales.

***Política de escritura: en acierto***

*Write-through (Escritura inmediata).*

* Se actualizan simultáneamente la posición de la caché y de la memoria principal.
* con múltiples CPU, observar el tráfico a memoria principal para mantener actualizada cada cache local.
* se genera mucho tráfico y retrasa la escritura. Write-back (Post-escritura).
* La información sólo se actualiza en la caché.
* Se marca como actualizada  bit de “sucio”.
* La memoria principal se actualiza en el reemplazo y puede contener información errónea en algún momento

***Política de escritura: en fallo***

*Write allocate*

* La información se lleva de la memoria principal a la caché. Se sobreescribe en la caché
* Habitual con write-back

*No-write allocate*

* El bloque no se lleva a la memoria caché. Se escribe directamente en la memoria principal.
* Habitual con write-through