***Arquitectura de Computadoras***

**Clase 1**

Programación en Hardware 🡪 Cuando cambiamos las tareas debemos cambiar el hardware.

Programación en Software 🡪 En cada paso se efectúa alguna operación sobre los datos.

* Para cada paso se necesita un nuevo conjunto de señales de control.
* Las instrucciones proporcionan esas señales de control.

Arquitectura de Von Neumann

* La CPU está constituida por la Unidad de Control (UC) y la Unidad Aritmético-Lógico (ALU).
* Datos e instrucciones deben introducirse en el sistema y los resultados se proporcionarán mediante componentes de E/S.
* Se necesita almacenar temporalmente datos e instrucciones en Memoria Principal.

Repertorio de Instrucciones 🡺 Conjunto completo de instrucciones que se realizan en una CPU:

* + Código máquina
  + Binario
* Representados por un conjunto de códigos de ensamblaje:
  + De operaciones:
    - ADD, SUB, LOAD
  + De operandos:
    - ADD BX, PEPE; suma contenidos de reg BX y dir PEPE, guarda el resultado en reg BX

Elementos de una Instrucción:

* Código de operación
* Referencia a operandos fuentes
* Referencia al operando resultado
* Referencia a la siguiente instrucción

Los operandos se almacenan en:

* Memoria Principal (o virtual o cache)
* Registro de la CPU
* Dispositivo de E/S
* Otras alternativas:
  + Pila
  + Acumulador
  + Memoria – Memoria
  + Registro – Registro

Tipos de Instrucciones:

* Procesamiento de Datos: Instrucciones aritmético-lógicas
* Almacenamiento de Datos: Instrucciones de memoria
* Transferencia de Datos: Instrucciones de E/S
* Control: Instrucciones de testeo y flujo del programa

Cantidad de direcciones:

* Más direcciones por instrucción:
  + Instrucciones más complejas
  + Más registros
  + Menos instrucciones por programa
* Menos direcciones por instrucción:
  + Instrucciones menos complejas
  + Más instrucciones por programa
  + La captación/ejecución de las instrucciones es más rápida

Decisiones de diseño de instrucciones:

* Tipos de operandos (datos)
* Repertorio de operaciones (cuántas, cuáles y cuán complejas)
* Formatos de instrucciones:
  + Longitud de instrucción
  + Número de direcciones
  + Tamaño de los campos
* Registros
  + Número de registros de CPU referenciables
  + En qué registros se pueden ejecutar qué operaciones
* Modos de direccionamiento: Cómo es especificada la ubicación de un operando
* RISC contrapuesto a CISC (de conjunto reducido de instrucciones a complejo)

Tipos de operandos:

* Direcciones
* Números (punto fijo o flotante)
* Caracteres
* Datos lógicos (Bits – 0 o 1)

Orden de los bytes:

* Big endian🡪El byte más significativo en la dir con valor numérico más bajo.
* Little endian🡪El byte menos significativo en la dir con valor numérico más bajo.

\*\*Problema de incompatibilidad\*\*

Accesos a la memoria:

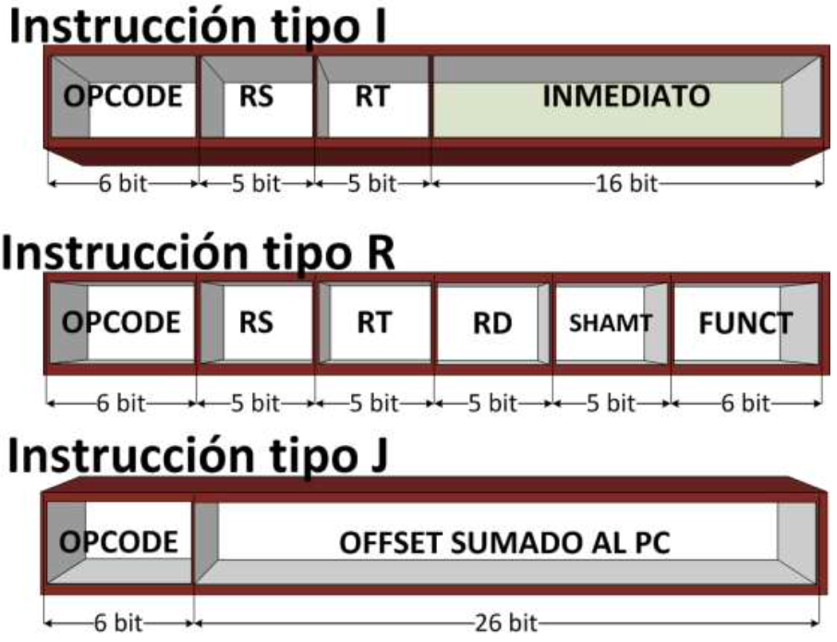
* Alineado
* No alineado 🡪 si se permiten, son más lentos.

Tipos de Operaciones:

1. Transferencia de datos
2. Aritméticas
3. Lógicas
4. Conversión
5. Entrada/Salida
6. Control del sistema
7. Control de flujo
8. Transferencia de Datos:
   1. Debe especificarse:
      1. Ubicación del operando fuente y destino
      2. Tamaño de los datos a ser transferidos
      3. Modo de direccionamiento
   2. Diferentes movimientos 🡪 Diferentes instrucciones
      1. Reg – Reg
      2. Reg – Mem
      3. Mem – Reg
   3. O, una instrucción y diferentes direcciones
      1. MOV destino, fuente; copia fuente a destino
9. Aritméticas:
   1. Operaciones Básicas:
      1. ADD, SUB, MUL, DIV
   2. Otras operaciones:
      1. INC o DEC (en 1 el operando)
      2. NEG 🡪 Cambia el signo del operando (Ca2)
      3. ABS 🡪 Toma el valor absoluto del operando
      4. Shift left/right 🡪 Desplaza bits a izq/der un lugar
10. Lógicas: (y conversión)
11. Conversión:
    1. Operaciones que manipulan bits individualmente
       1. Operaciones Booleanas: AND, OR, XOR, NOT
       2. Otras operaciones: Rotate left/right 🡪rota posiciones de los bits
    2. Operaciones para cambiar formatos de datos
       1. Conversión de binario a decimal o de EBCDIC a ASCII
12. Entrada/Salida:
    1. Pocas instrucciones pero de acciones específicas
       1. IN o OUT
    2. Se pueden realizar utilizando instrucciones de movimiento de datos
       1. MOVE
    3. Se pueden realizar a través de un controlador aparte
       1. DMA (Direct Memory Access)
13. Control de Flujo:
    1. Modifican el valor contenido en el registro PC (Contador de Programa)
       1. Salto Incondicional
          1. JMP equis; saltar a la posición equis
       2. Salto Condicional
          1. JZ equis; saltar a la posición equis, si flag Z = 1
       3. Salto con retorno o llamada a subrutina
          1. CALL subrut; saltar a la posición de subrut

Para retornar al programa se utiliza RET como última instrucción.

Formato de instrucción:



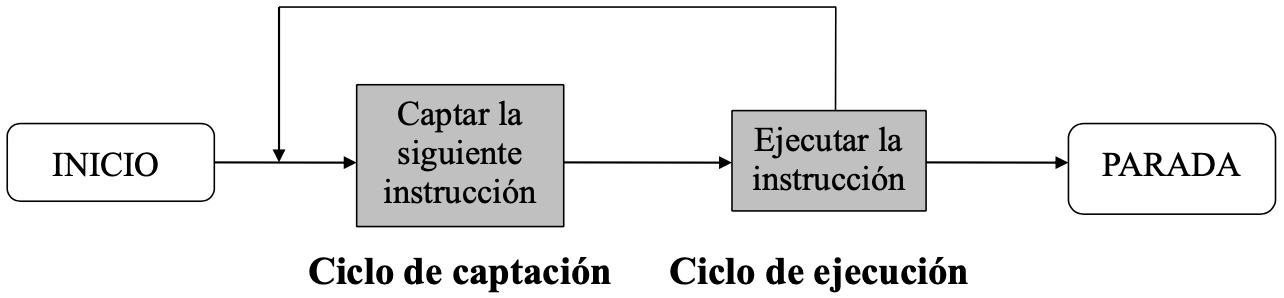
Modos de Direccionamiento:

* Inmediato 🡪 El operando está en la instrucción
* Directo de memoria o Absoluto 🡪 Dirección del operando en memoria
* Directo de Registro 🡪 ID del registro que tiene el operando
* Indirecto de memoria (en desuso)
* Indirecto con registro
* Indirecto con Desplazamiento 🡪 Desplazamiento que se suma a la dirección en ID del registro para obtener la dirección del operando en memoria
  + Basado, indexado o relativo al PC
  + Pila (o relativo al SP)

Ciclo de Instrucción Básico:

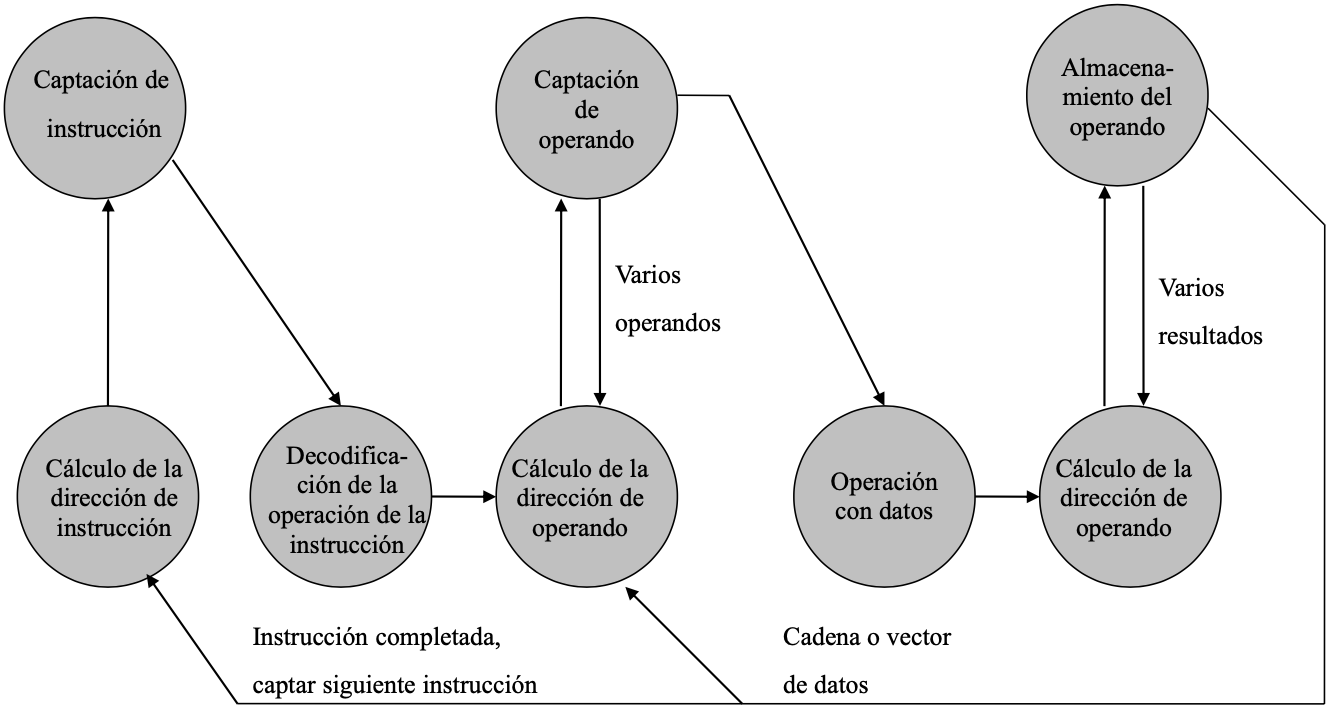
Dos pasos:

* + Captación
  + Ejecución



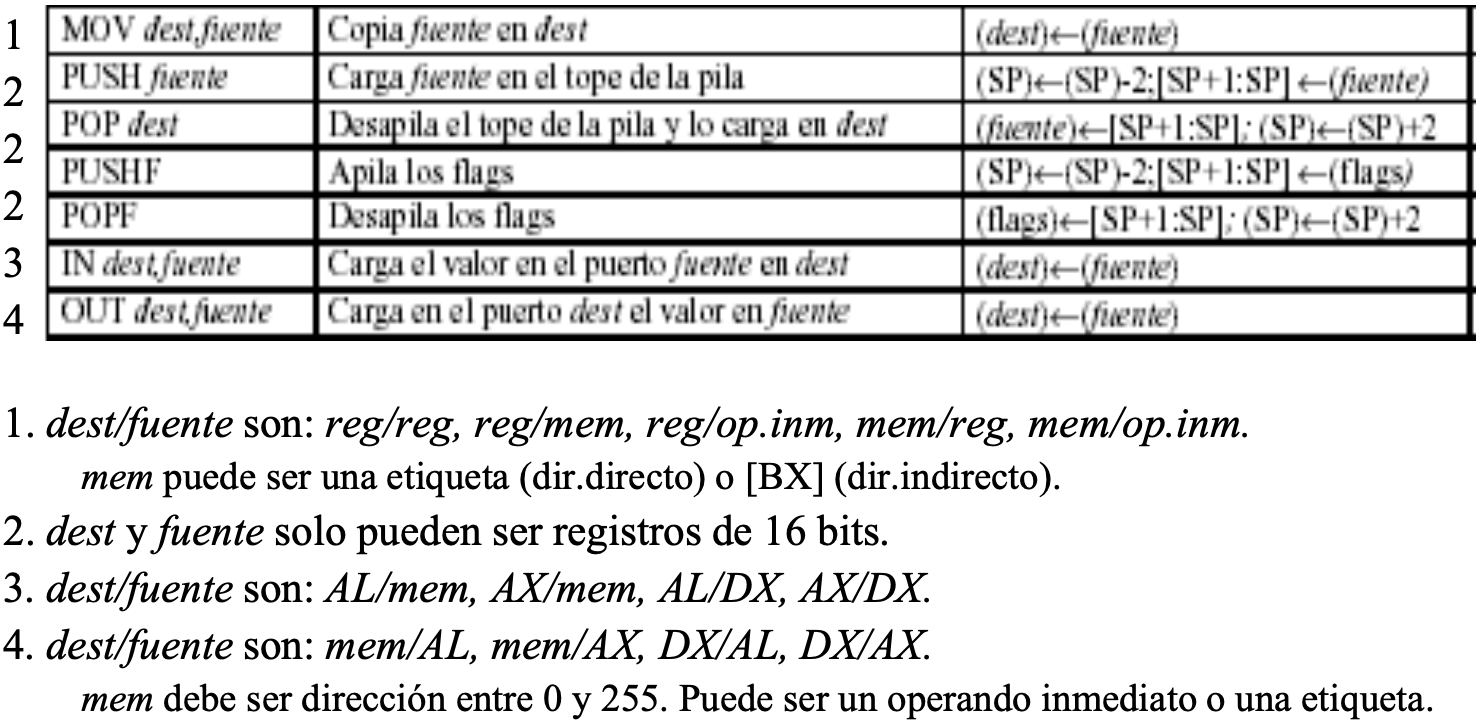
* Ciclo de Captación:
  + La dirección de la instrucción que se debe captar se encuentra en el registro PC
  + La UC capta la instrucción desde la Memoria
    - La instrucción va al registro de instrucción (IR)
  + El registro PC se incrementa
  + La UC interpreta la instrucción captada y debe llevar a cabo la acción
* Ciclo de Ejecución:
  + Acciones posibles:
    - Procesador – Memoria: Transferencia de datos CPU – Memoria
    - Procesador – E/S: Transferencia de datos CPU y módulo de E/S
    - Procesamiento de Datos: Op aritmética o lógica con los datos
    - Control: Alteración de la secuencia de ejecución (instr de salto)
  + O combinación de las acciones anteriores

Diagrama de estados del ciclo de instrucción

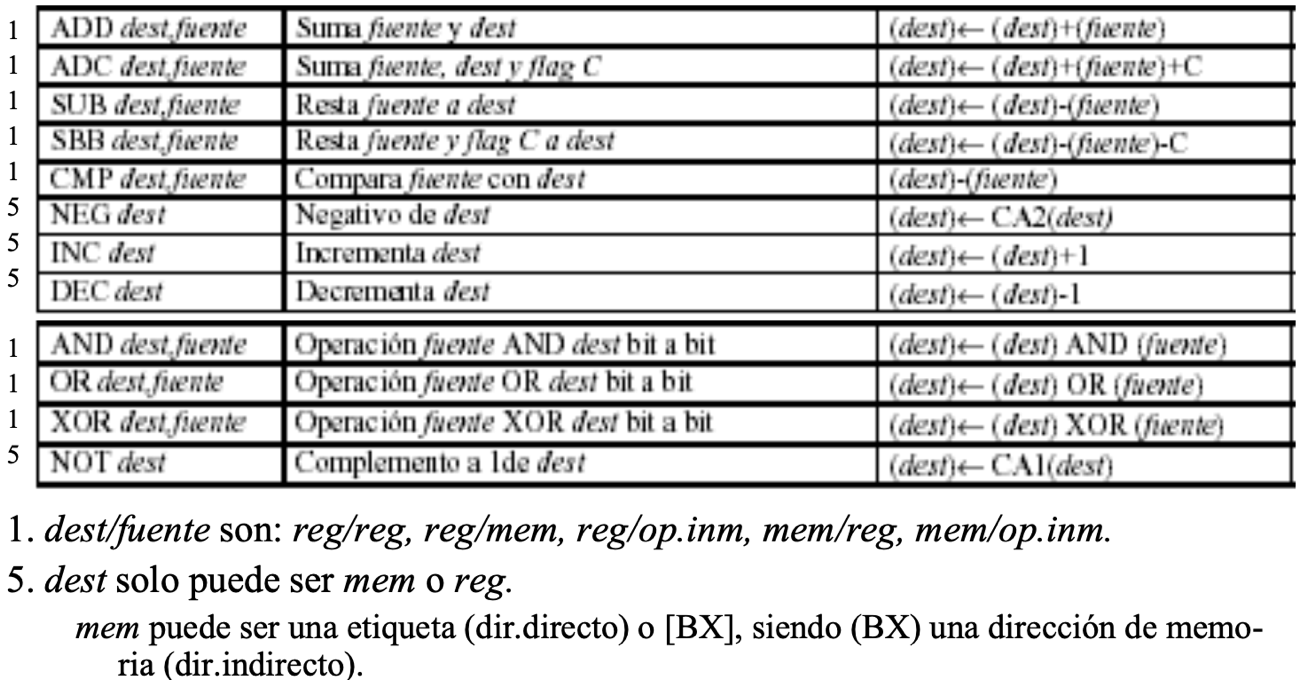


MSX88:

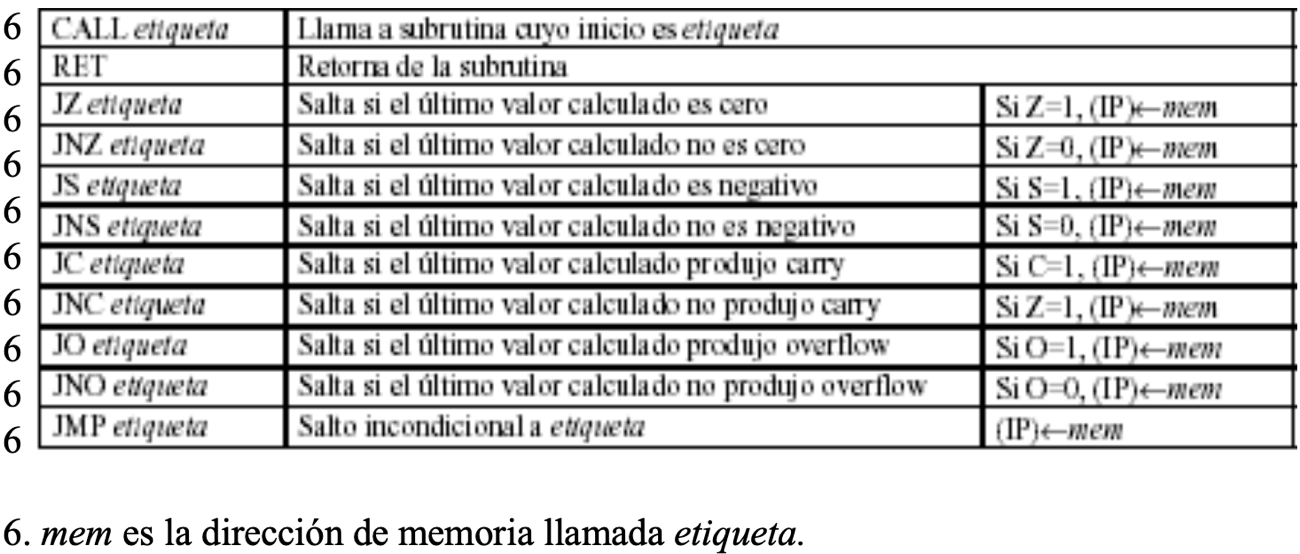
* Instrucciones de transferencia



* Instrucciones aritméticas y lógicas



* Instrucciones transferencia de control



Subrutinas:

* Características
  + Programa auto-contenido
  + Puede invocarse desde cualquier punto de un programa(mediante CALL)
  + Brinda economía (reutiliza código) y modularidad (subdivisión pequeña)
  + Requiere pasaje de argumentos (parámetros)
    - Por valor (copia de una variable)
    - Por referencia (dirección de la variables)
* Pasaje de argumentos:­
  + Vía Registros
    - El número de registros es la principal limitación
    - Es importante documentar qué registros se usan
  + Vía Memoria
    - Se usa un área definida de memoria (RAM)
    - Difícil de estandarizar
  + Vía Pila (stack)
    - Es el método más ampliamente usado
    - Independiente de memoria y registros
    - Hay que comprender bien cómo funciona porque la pila es usada por el usuario y por el sistema.

\*\*SP apunta al último lugar usado\*\*

Funcionamiento de una Pila:

* Uso del sistema: Salva la dirección de retorno cuando es llamada una subrutina o se produce una interrupción por hardware.
* Uso del usuario: Pasaje de parámetros.
* El operando está (de forma implícita) en la cabeza de la pila
* Se requiere un registro Puntero de Pila (SP) 🡪 Dirección de la cabeza de la pila
  + Simulador inicializa SP=8000H
* Operaciones sobre la pila
  + PUSH; operación de Apilar
  + POP; operación de Desapilar

\*\*Son inversas entre sí\*\*

Secuencia de dos acciones:

1. Movimiento de datos Reg-Mem o Mem-Reg
2. Modificación del puntero antes/después de la anterior
   * Tener en cuenta:
     + Dónde apunta el puntero
     + Cómo crece la pila

Definición del Procedimiento

Nombre Proc

…

…

Ret

Nombre Endp

Llamada al Procedimiento

…

Push Parametro 1

Push Parametro 2

Call Nombre

…

Posibles pasos en un procedimiento:

1. Salvar el estado de BP (viejo BP)
   1. El procedimiento comenzaría con:
      1. Push BP
      2. Mov BP, SP
   2. Esto establece a BP como puntero de referencia y es usado para acceder a los parámetros y datos locales en la pila. SP no puede ser usado para éste propósito porque no es un registro base o índice. El valor de SP puede cambiar pero BP permanece ‘quieto’.
2. Salvar estado de SP (BP=SP)
   1. Así la primera instrucción salva BP y la segunda carga el valor de SP en BP
   2. BP es el puntero al área de la pila asignada al procedimiento (frame pointer)
   3. Para acceder a los datos se deberá sumar un desplazamiento fijo a BP
3. Reservar espacio para otros registros (opcional)
   1. Reservar espacio para variables locales
      1. Se decrementa SP, reservando lugar en la pila
   2. El sistema puede utilizar al SP sin escribir sobre el área de trabajo del procedimiento
4. Salvar valores de otros registros (opcional)
   1. Se realiza un push del registro que se quiera salvar
   2. Si el procedimiento no cambia el valor de los registros, éstos no necesitan ser salvados. Normalmente son salvados después de establecer el puntero y los datos locales.
5. Acceder a parámetros
   1. En general el desplazamiento de BP para acceder a un parámetro es:
      1. 2 (tamaño de BP apilado) + tamaños de dirección de retorno + total de tamaño de parámetro entre el buscado y BP
6. Escribir sentencias a ejecutar
7. Retornar parámetros (opcional)
8. Regresar correctamente del procedimiento

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.

Simulador:

* Declaración del procedimiento:

Nombre: instrucción

.

.

* En lugar de BP se usa BX

**Clase 2**

Interrupciones 🡺 Mecanismo mediante el cual se puede interrumpir el procesamiento normal de la CPU 🡪 Pueden ser de origen interno o externo.

🡺 Cuando un dispositivo necesita la atención del procesador, envía una señal para avisarle, produce una interrupción.

Motivos de interrupción:

1. Por resultado de una ejecución de una instrucción🡪 división por cero, overflow
2. Por un temporizador interno del procesador
3. Por una operación de E/S
4. Por un fallo de hardware

¿Qué hacer?

* En casi todos los casos, implica transferir el control a otro programa (Gestor):
  + Salva el estado del procesador
  + Corrige (o responde a) la causa que ocasionó la interrupción
  + Restaura el estado original del procesador
  + Retorna a la ejecución normal del programa interrumpido

Jerarquía:

* Si hay múltiples fuentes que pueden solicitar interrupción, se establece cuáles son más importantes.
* 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.

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 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 o a zonas de memoria protegidas

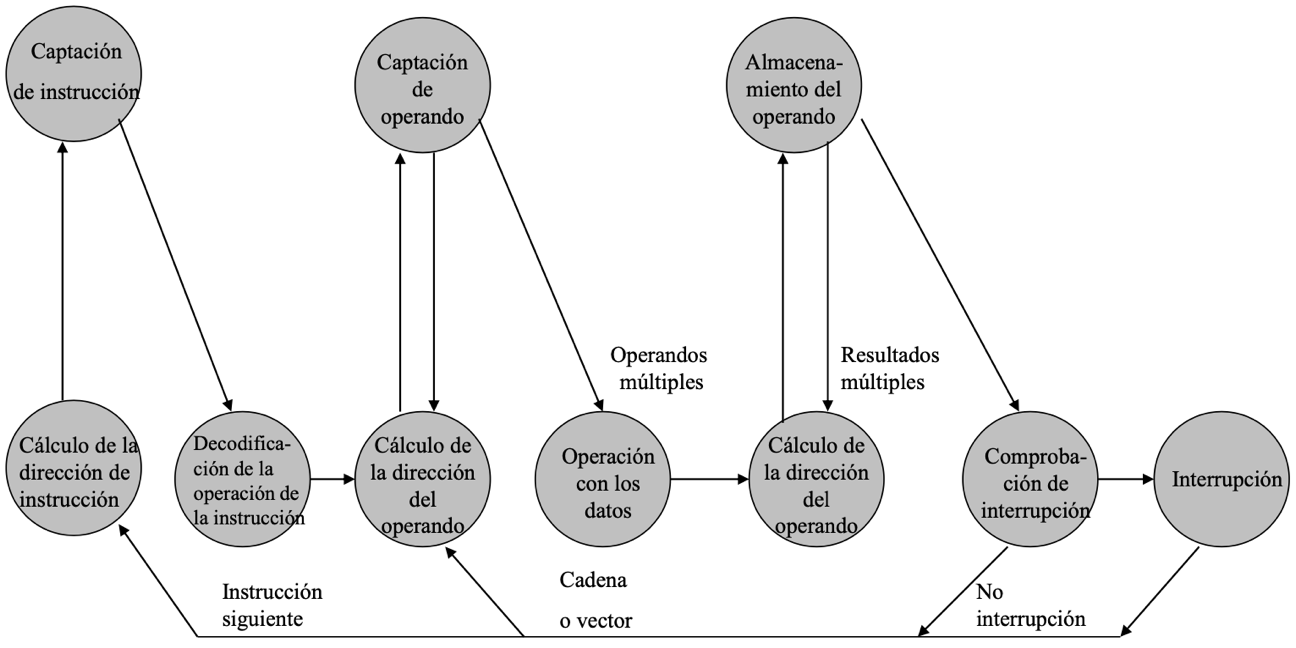
Interrupciones por Software:

* Muchos procesadores tienen instrucciones explícitas (INT N) que afectan al procesador de la misma manera que las interrupciones por hardware.
* Generalmente usadas para hacer llamadas a funciones del SO
  + Permite que las subrutinas del sistema se carguen en cualquier lugar.
* No requieren conocer la dirección de la rutina en tiempo de ejecución
* Hay sistemas que no permiten hacer una llamada directa a una dirección de la función del SO, por estar en una zona reservada.
* ¿Qué pasa si no tuviera las instrucciones por software?
  + Debería escribir todas las funciones que necesito, o
  + Al cargar un programa habría que mirar todas las llamadas a funciones del BIOS y SO y reemplazar en el código las direcciones de todas estas funciones invocadas.

Ciclo de Interrupción (Añadido al ciclo de instrucción):

* 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.

Diagrama de estados de un ciclo de instrucción con interrupciones



Interrupciones múltiples:

* Inhabilitadas:
  + El procesador puede 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, debe colocar en el bus de datos un identificador (vectir)
    - Lo coloca el periférico directamente, o
    - Controlador de Interrupciones (se ocupa de todo)

PIC 🡺 Dispositivo Controlador Programable de Interrupciones.

* Tiene 16 entradas externas 🡪 IR 0 .. IR 15 – Interrup Request
* Una salida 🡪 IntR – Le avisa al Procesador que tiene una interrupción
* Y una entrada del procesador🡪 IntA – El procesador le avisa si puede atenderla
* Pasos:

1. El PIC recibe un pedido de interrupción de un dispositivo externo y prioriza éste con otros pedidos que puedan llegar o estar pendientes.
2. Un pedido de interrupción es enviado al procesador, por la línea INTR.
3. El procesador contesta por la línea INTA y solicita un puntero de 8 bits para la tabla.
4. El procesador recibe un puntero y lo usa para acceder a la tabla de vectores (direcciones) donde se encuentra la dirección del servicio.
5. Accede a la tabla y obtiene la dirección buscada.
6. Salva en la pila la dirección actual, los flags y salta a la dirección del servicio.

Tabla de Vectores de Interrupción:

* Es el nexo entre tipo de interrupción (0..255) y el procesamiento designado para atenderla.
* Cada entrada es una doble palabra (4 bytes).
* Vectores preasignados:
  + Tipo 0 (INT 0) – Finaliza la ejecución del programa
  + Tipo 3 (INT 3) – Punto de parada para depuración/seguimiento
  + Tipo 6 (INT 6) – Lectura de entrada std. Requiere el uso de BX
  + Tipo 7 (INT 7) – Escritura de salida std. Requiere BX y AL

Controlador de Interrupciones

Registros internos PIC (Todos son de 8 bits)

* + EOI: Para comandos
    - Para fin de interrupción escribir 20H
  + IMR: Máscara de interrupción
    - Enmascara con ‘1’
  + IRR: Petición de interrupción
    - Indica con bit en 1
  + ISR: interrupción en servicio
    - Indica con bit en 1
  + INT0..INT7
    - c/u con su vector

Conexionado y direccionamiento:

* 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
* Interrupciones hardware asignadas:
  + INT0 – tecla F10
  + INT1 – Timer
  + INT2 – Handshake
  + INT3 – DMA
  + INT4 a INT7 – no usadas

Periféricos internos:

* PIO: Puertos paralelos de E/S
* HAND-SHAKE
* PIC: Controlador de interrupciones
* TIMER: Contador de eventos
* CDMA: Controlador de acceso directo a memoria

Periféricos externos:

* Barra de led’s
* Barra de interruptores (microswitchs)
* Impresora

CLI 🡪 Instrucción que deshabilita las interrupciones (no se atienden).

STI 🡪 Instrucción para habilitar las interrupciones.

OUT 🡪 Para enviar datos a un dispositivo externo.

EQU 🡪 Define una constante.

**Clase 3**

Problemas de Entrada/Salida

* Gran variedad de periféricos con varios métodos de operación
  + Trasmisión de diferentes cantidades de datos
  + A diferentes velocidades
  + Usan diferentes formatos de dato y tamaño de palabra
* Todos más lentos que la CPU y la RAM
* Necesidad de módulo de E/S (con alguna “inteligencia”)

Módulo de E/S

* Realiza la interfaz entre el procesador y la memoria (bus) y los periféricos.
* Pueden manejar uno o más periféricos.

Dispositivos Externos (TODO lo que no sea memoria o CPU):

1. E/S básicos 🡪 monitor/pantalla, mouse, teclado
2. Almacenamiento 🡪 disco duro, CD, DVD
3. Impresión 🡪 impresora, escáner
4. Comunicación con dispositivos remotos 🡪 modem, acceso/interfaz de red
5. Multimedia 🡪 micrófono, parlantes
6. Automatización y control 🡪 sensores, alarmas, adquisición de datos

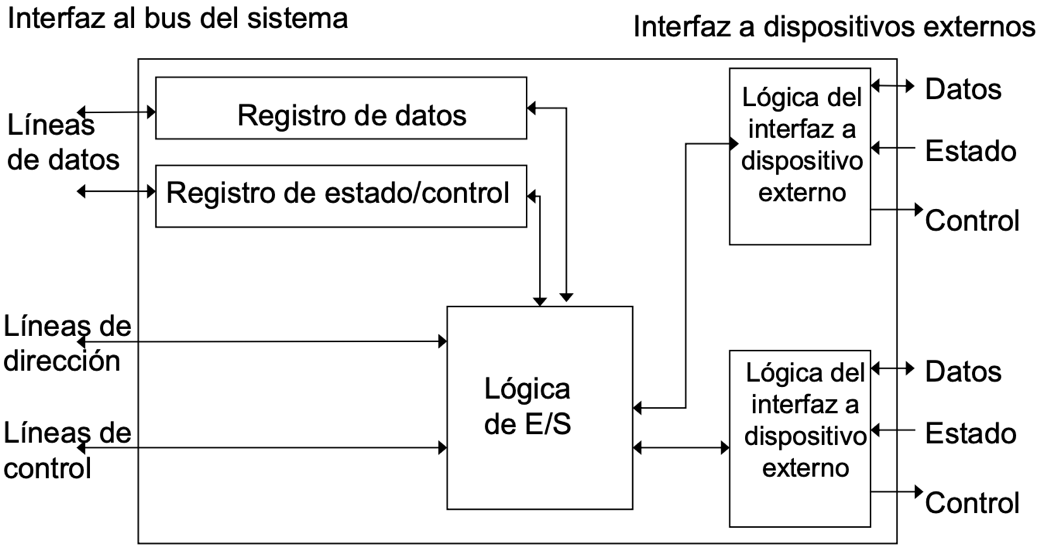
Características de un puerto:

1. Interfase entre el periférico y el módulo de E/S
2. Señales de Control, Estado y Datos
   1. Señal de Control: función a realizar (INPUT/OUTPUT)
   2. Señal de Estado: READY/NOT READY
   3. Control lógico: manejo de direccionamiento
   4. Transductor: conversión de datos
   5. Buffer: adaptación (almacena 32 y da de a 1, 8 o 16 bits)

Funciones de un módulo de E/S:

* Control y temporización de uno o más dispositivos externos.
* Interpretar las órdenes que recibe de CPU y transmitirlas al periférico.
* Comunicación con la CPU (registros) y Memoria.
* Controlas las transferencias de datos entre CPU y el periférico (convertir formatos, adaptar velocidades).
* Comunicación con los dispositivos (periféricos).
* Informar a la CPU del estado del periférico.
* Almacenamiento temporal (buffering) de datos.
* Detección de errores.

Diagrama en bloques de un módulo de E/S



Capacidades de un módulo de E/S

* Ocultas propiedades del dispositivo a la CPU
* Ocuparse de uno o varios dispositivos
* Controlar o no las funciones del dispositivo
  + Canales de E/S o procesadores de E/S (manejo de parte importante de la carga del procesamiento). Mainframes.
  + Controlador de E/S o controlador de dispositivo (manejo primitivo). Microcomputadoras.

Operación de Entrada o Salida – Requiere:

* Direccionamiento (métodos para hacer la interface del espacio de E/S):
  + E/S mapeada en memoria:
    - Dispositivos de E/S y memoria comparten un único espacio de direcciones.
    - Ventajas: puedo usar todo el conjunto de instrucciones. No hay instrucciones como IN y OUT.
    - Desventajas: ocupa espacio de memoria.
  + E/S aislada:
    - Espacios de direcciones separados.
    - Necesidad de líneas especiales.
    - Órdenes específicas para E/S (IN/OUT en lugar de MOV).

Técnicas de gestión de E/S 🡪 Cómo se comunican los dispositivos de E/S con la CPU.

1. Programada con espera de respuesta
2. Con interrupciones:
3. Con acceso directo a memoria (DMA)
4. Programada con espera de respuesta:
   1. Intercambio de datos entre la CPU y el módulo
   2. La CPU tiene control directo sobre la operación de E/S
   3. La CPU espera que el módulo termine la operación
   4. La CPU permanece ociosa durante un período de tiempo
5. Con interrupciones:
   1. La CPU NO tiene que esperar la finalización de la tarea de E/S.
   2. No se repite la comprobación de los estados de los módulos.
   3. El módulo envía un pedido de interrupción a la CPU cuando está listo nuevamente.

Identificación del módulo que genera la interrupción

* Diferentes líneas para cada módulo
  + PC
  + Limita el número de dispositivos
* Consulta software (Poll o encuesta)
  + Ocurrido un pedido de interrupción la CPU consulta a cada uno para determinar quién fue el demandante
  + Resulta lento
* Conexión en cadena (Daisy chain) “hard poll”
  + La línea de reconocimiento de interrupción se conecta encadenando los módulo, la línea de pedido es compartida
  + Una vez enviada la confirmación de parte de la CPU el módulo responderá colocando un vector en el bus que lo identifica
  + La CPU emplea el vector como puntero para acceder a la rutina de servicio

Interrupciones múltiples:

1. Todas las líneas de interrupción tienen un orden de prioridad
2. Las líneas con más prioridad puede interrumpir a las líneas con menor prioridad
3. Si existe un maestro del bus, sólo él puede interrumpir

PIO 🡪 Interfase de Periféricos Programable

* 2 puertos paralelos de 8 bits: A y B
* Se puede programar c/ bit por separado como entrada o salida.
* 4 registros internos de 8 bits:
  + 2 de datos, PA(30H) y PB(31H)
  + 2 de control CA(32H) y CB(33H), para programar los bits datos
    - CA = un bit en 0 selecciona como salida a la línea de PA
    - CA = un bit en 1 selecciona como entrada a la línea PA
    - CB = controla de la misma manera a PB

1. DMA – Acceso Directo a Memoria:
   1. Es un dispositivo capaz de controlar una transferencia de datos entre un periférico y memoria sin intervención de la CPU.
   2. DMAC 🡪 Controlador de DMA: Debe actuar como maestro del bus durante la transferencia DMA y debe ser capaz de:
      1. Solicitar el uso del bus mediante las señales y la lógica de arbitraje necesarias
      2. Especificar la dirección de memoria sobre la que se realiza la transferencia
      3. Generar las señales de control del bus
         1. Tipo de operación (lectura/escritura)
         2. Señales de sincronización de la transferencia

Etapas de una transferencia DMA:

* Inicialización de la transferencia
  + La CPU debe enviar al interfaz del periférico y al DMAC los parámetros de la transferencia
* Inicialización del interfaz (Bus master: CPU-Bus slave: Interfaz)
  + Nº de bytes a transferir
  + Tipo de transferencia
  + Otra información de control
* Inicialización controlador DMA (Bus master: CPU-Bus slave: DMAC)
  + Nº de bytres o palabras a transferir
  + Tipo de transferencia
  + Dirección de memoria inicial para la transferencia
  + Nº de canal (DMAs con varios canales)

\*\*Después de la inicialización la CPU retorna a sus tareas y ya no se preocupa más de la evolución de la transferencia.\*\*

* Realización de la transferencia
  + Cuando el periférico está listo para realizar la transferencia se lo indica al DMAC
  + El DMAC pide el control del bus y se realiza la transferencia entre el periférico y la memoria
    - Bus master: DMAC + periférico – Bus slave: Memoria
    - Después se actualizan los registros del DMAC
      * Nº bytes o palabras a transferir
      * Dirección de memoria
* Finalización de la transferencia
  + El DMAC libera el bus y devuelve el control a la CPU
  + El DMAC suele activar una señal de interrupción para indicar a la CPU la finalización de la operación de E/S solicitada

Posible problema:

* 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 instrucciones/datos
* El problema se reduce con el uso de memoria cache
  + 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 para realizar transferencias
* En caso de computadoras sin cache
  + El procesador no utiliza el bus en todas las fases de la ejecución de una instrucción
  + El DMAC puede aprovechar estos momentos y realizar las transferencias

Tipos de transferencias:

1. Por ráfagas (burst):
   1. El DMAC solicita el control del bus y no lo libera hasta haber finalizado la transferencia de todo el bloque de datos completo.
      1. Ventajas: La transferencia se realiza en forma rápida
      2. Desventajas: La CPU no puede utilizar el bus con memoria, lo que puede degradar el rendimiento del sistema
2. Por robo de ciclo (cycle-stealing):
   1. El DMAC solicita el control del bus, realiza la transferencia de una única palabra y libera el bus. Solicita el control tantas veces como sea necesario hasta finalizar la transferencia del bloque completo.
      1. Ventajas: No se degrada el rendimiento del sistema
      2. Desventajas: La transferencia tarda más tiempo

Canales de E/S – Características:

* 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
  + Ordena ejecutar el programa que está en memoria
  + El programa especifica dispositivos, áreas de memoria a usar, prioridades y acciones ante errores

Tipos de canales de E/S:

1. Selector:
   1. 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.
   2. El canal selecciona un dispositivo y efectúa la transferencia.
   3. Los dispositivos son manejados por un controlador o módulo de E/S.
   4. Por lo tanto el canal de E/S ocupa el lugar de la CPU en el control de esos controladores.
2. Multiplexor:
   1. Puede manejar E/S con varios dispositivos a la vez.
   2. Multiplexor de bytes: Acepta y transmite caracteres.
   3. Multiplexor de bloques: Intercala bloques de datos desde distintos dispositivos.

**Clase 4**

Segmentación de Instrucciones o Cauce (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.

Características:

1. Le segmentación es una técnica de mejora de prestaciones a nivel de diseño hardware.

2. La segmentación es invisible al programador.

3. Necesidad de uniformizar las etapas 🡪 Al tiempo de la más lenta.

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

Tareas a realizar por ciclo:

* Búsqueda (IF, Fetch)
  + Se accede a memoria por la instrucción
  + Se incrementa el PC (en 4 bytes)
* Decodificación (ID, Decode)
  + Se decodifica la instrucción, obteniendo operación a realizar.
  + 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.
* Ejecución (EX, Execute)
  + Se ejecuta la operación en la ALU
  + Calcula la dirección de operandos de memoria
* Acceso a memoria (MEM, Memory Access)
  + Si se requiere un acceso a memoria, se accede
* Almacenamiento (WB, 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 cause =>

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, pero no reduce el tiempo de ejecución de la instrucción.

MIPS 64

– Características:

* Bus de datos de 64 bits
* Registros de 64 bits
* 32 registros de propósito general (R0..R31)
* 32 registros de coma flotante (F0..F31)
* Instrucciones de longitud fija
* Código de operación de longitud fija
* Sólo 2 instrucciones acceden a memoria (Load/Store)
* Memoria separada para instrucciones y datos
* Cauce segmentado en 5 etapas de 1 ciclo cada una

– Riesgos:

* Estructurales 🡪 Provocados por conflictos por los recursos
  1. Problema: Dos o más instrucciones intentan usar el mismo recurso de hardware a la vez (ej. Acceder a memoria)

Solución: Memoria Separada (Harvard)

* 1. Problema: Se escriben y leen los registros a la vez

Solución: Escritura en el 1º sub-ciclo y lectura en el 2º

* Dependencia de datos (detectada en la etapa ID)
  1. Dependencia verdadera (RAW): Cuando una instrucción depende de los resultados de otra instrucción, de manera que ambas no pueden ejecutarse de forma solapada.
  2. Anti-dependencia (WAR): Cuando una instrucción j escribe un registro que lee la instrucción i. Se debe mantener el orden original para asegurar que i lee el valor correcto.
  3. Dependencia en Salida (WAW): Se produce cuando la instrucción i y la j escriben en el mismo registro. Se debe mantener el orden entre las instrucciones para que el valor final escrito corresponda a la instr. j.
  + Soluciones:
    - Software
      * Inserción de instrucciones NOP
      * Reordenación de código
    - Hardware
      * Detención del cauce 🡪 Espera a que el dato esté escrito en registro para utilizarlo (RAW en etapa ID)
      * Forwarding (adelantamiento) 🡪 Utilizar el dato ni bien esté disponible (RAW en etapa EX)
* Dependencia de control (saltos) 🡪 La ejecución de una instrucción depende de cómo se ejecute otra
  + Problema: Una instrucción que modifica el valor del PC no lo ha hecho cuando se tiene que comenzar la siguiente.
  + Soluciones:
    - Software
      * Salto Retardado 🡺 Siempre se ejecuta la instrucción debajo del salto, y luego toma lugar el salto
        + Inserción de instrucciones NOP
        + Reordenación de código
    - Hardware
      * Detención del cauce 🡪 El cause se detiene hasta comprobar la condición del salto y saber dónde saltar. Penalización de una etapa.
      * Predicción del salto 🡪 Si falla, penalización de un ciclo.

**Clase 5**

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.

Tratamiento de saltos:

* Técnica Hardware: Predicción de saltos para evitar la parada
* 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.

Predicción de salto:

* 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 del 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
      * Nº bits de estado (historia de uso)

Salto retardado 🡪 Realizar trabajo útil mientras el salto se resuelve

* Hueco o ranura de salto (delay-slot) es el período de penalización o parada luego de una instrucción de salto.
* El compilador trata de situar instrucciones útiles en los huecos de retardo.
* Las instrucciones en los huecos de retardo de salto se captan siempre.
* Requiere reordenar las instrucciones.

Otras soluciones hardware:

* Predecir según el código de operación
  + Hay instrucciones con más probabilidades de saltar
* Flujos múltiples
* Precaptar el destino del salto
* Buffer de bucles

Flujos múltiples:

* Varios cauces (uno por cada opción de salto)
* Precaptan cada salto en diferentes cauces
* Se debe utilizar el cauce correcto
* Desventajas:
  + Provocar retardos en el acceso al bus y a los registros.
  + Si hay múltiples saltos, se necesita un mayor número de cauces.

Problemas:

* En la etapa ID se comprueba si el salto se lleva a cabo o no y dónde saltar.
* No se puede comprobar antes porque primero hay que decodificar la instrucción.
* Si en la etapa IF se pudiera saber si hay que saltar o no, el pipe tomaría el camino correcto.
* ¿Cómo saber si la instrucción es un salto si todavía no fue decodificada?

BTB: Branch Target Buffer

* Se “agrega” en la etapa IF una memoria “muy rápida” (BTB) donde se almacenan las direcciones (IP) de memoria donde están los saltos solamente. Si saltó o no la última vez y la dirección donde saltó.
* Cada vez que se busaca una instrucción se comprueba si esta dirección está en la tabla (BTB). Si está es porque es un salto.
* Así sabemos que es un salto, sin decodificar, y si saltó y a dónde la última vez.
* En la tabla se anotan la dirección del salto (PC), dónde saltar, y la predicción del salto.
* La tabla es analizada en la etapa IF.

Pre-captación del destino de salto:

* Se pre-capta 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.

WAR 🡪 Write After Read

Anti-dependencia:

* Cuando una instrucción j escribe un registro que lee la instrucción i, se debe mantener el orden original para asegurar que i lee le valor correcto.

WAW 🡪 Write After Write

Dependencia de Salida:

* Cuando la instrucción i y la j escriben en el mismo registro. Se debe mantener el orden entre las instrucciones para que el valor final escrito corresponda a la instrucción j.

**Clase 6**

RISC 🡪 Computadoras de repertorio reducido de instrucciones

1. Características:
2. Gran número de registros de uso general o mejorar tecnología de compiladores para optimizar el uso de los registros.
3. Repertorio de instrucciones limitado y sencillo.
4. Énfasis en la optimización de la segmentación de instrucciones.

Finalidad del CISC:

* + - * 1. Facilitar el trabajo del escritor de compiladores.
        2. Mejorar la eficiencia de la ejecución:
  + Secuencias complejas de operaciones en microcódigo
    - * 1. Dar soporte a HLL más complejos.

Inconvenientes del CISC:

* El software resulta mucho más caro que el hardware.
* El nivel del lenguaje era cada vez más complicado.
* Salto semántico 🡪 diferencias entre operaciones HLL y de la Arquitectura
* Todo esto conduce a:
  + Repertorios de instrucciones grandes
  + Más modos de direccionamiento
  + Varias sentencias de HLL implementadas en el Hardware

Operaciones:

* Asignación: Movimiento de datos
* Estamentos condicional (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 las 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 seguidas por la correspondiente secuencia de retornos
  + La mayoría de las variables son locales
  + Las referencias a operandos están muy localizadas

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:
  + Es recomendable un repertorio con instrucciones simples (reducido).

Amplio banco de registros:

* Aproximación por Software:
  + El compilador es necesario para asignar registros
  + Asignación de registros a las variables que se usen más en un período de tiempo dado
  + Requiere el uso de sofisticados algoritmos de análisis de programas
* Aproximación por Hardware:
  + Utilización de más registros
  + De esta manera, más variables pueden mantenerse en registros durante períodos de tiempo más largos



Controversia RISC y CISC:

* Cuantitativa 🡪 Comprobación 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”.
  + La mayoría de las máquinas son una mezcla de ambas.

**Clase 7**

Jerarquía de memoria:

* Organizada en niveles que son ubicados en distintos lugares físicos.
* Fabricados con tecnologías diferentes que se gestionan de manera independiente.
* Registros (en CPU)
* Caché: Cantidad pequeña de memoria rápida.
* Memoria Principal
* Memoria Virtual
* Objetivo: La velocidad del sistema deberá ser, aproximadamente, la del nivel más rápido al costo del nivel más barato.
* A media 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.

Propiedades a cumplir:

1. Inclusión: Los datos almacenados en un nivel han de estar almacenados en los niveles inferiores a él.
2. Coherencia: Las copias de la misma información en los distintos niveles deben contener los mismos valores.

¿Por qué funciona la jerarquía?

* Principio de localidad de referencia (Los cumplen los programas):
  + Localidad Temporal 🡪 Los elementos de memoria referenciados recientemente, volverán a serlo en un futuro próximo.
  + Localidad Espacial 🡪 Los elementos de memoria cuyas direcciones están próximas a los últimos referenciados, serán referenciados.

Funcionamiento de la caché:

* La CPU solicita contenido de 1 dirección de memoria.
* La cache, ¿tiene ese dato?
  + Si es así, lo obtiene de la misma.
  + Si no, se lee el bloque que contiene esa dirección desde la memoria principal y copia en la caché.
    1. Después la caché entrega el dato requerido a la CPU
* La caché incluye etiquetas para identificar qué bloque de la memoria principal está en cada una de sus líneas.

Acierto (hit) 🡪 Se encuentra en la caché el dato solicitado.

Fallo (miss) 🡪 NO se encuentra en la caché el dato solicitado. Se gestionan mediante hardware y el procesador de detiene hasta que el dato esté disponible.

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.

Ubicación de un bloque:

* Correspondencia Directa 🡪 Un bloque sólo puede estar almacenado en un lugar de la caché.
  + Nº línea caché = Nº bloque ref. ***mod*** Nº líneas caché
  + Ventajas:
    - Simple
    - Poco costosa
    - Hay una posición concreta para cada bloque dado
  + Desventaja:
    - Si un programa accede a dos bloques que se corresponden a la misma línea de forma repetida, las pérdidas de cache serán muy grandes.
* Correspondencia Totalmente Asociativa 🡪 Un bloque puede almacenarse en cualquier lugar de la caché.
  + Ventajas:
    - Un bloque de memoria principal puede colocarse en cualquier línea de la cache
    - La etiqueta identifica unívocamente un bloque de memoria
  + Desventajas:
    - Todas las etiquetas de las líneas se examinan para buscar una coincidencia
    - Búsqueda costosa
* Correspondencia Asociativa por Conjuntos 🡪 Un bloque puede almacenarse en un conjunto restringido de lugares en la caché.
  + Un conjunto es un grupo de líneas de la caché
  + Nº conjunto = Nº bloque ref. ***mod*** Nº conjuntos caché
  + Ventajas:
    - Combina lo mejor de las otras correspondencias
    - La cache se divide en un grupo de conjuntos
    - Un bloque determinado corresponderá a cualquier línea de un conjunto determinado

Política de Reemplazo:

* Algoritmos de sustitución:
  + En correspondencia directa:
    - El que ocupa el lugar del nuevo
  + En correspondencia asociativa:
    - LRU (menos recientemente usado)
    - FIFO (más antiguo)
    - LFU (menos frecuentemente usado)
    - Aleatoria

Algoritmos de sustitución:

* Correspondencia Directa:
  + No hay elección.
  + Sólo hay una posible línea para cada bloque.
  + Se necesita una sustitución de esa línea (sí o sí).
* Correspondencia Asociativa:
  + Los algoritmos deben implementarse en hardware.
  + Menos recientemente usado (LRU)
    - Requiere controles de tiempos
    - En correspondencias asociativas por conjuntos de 2 vías, ¿cuál de las 2 líneas es la LRU?
  + Primero en entrar – primero en salir (FIFO)
    - Requiere controles de acceso
    - Se sustituye aquella línea que ha estado más tiempo en la cache
  + Menos frecuentemente usado (LFU)
    - Requiere controles de uso
    - Se sustituye aquella línea que ha experimentado menos referencias
  + Aleatoria
    - Se sustituye una línea al azar

Políticas de escritura:

* Se debe evitar inconsistencia de memorias en el caso de escrituras.
* Tener en cuenta:
  + La CPU escribe sobre una línea de caché.
    - 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.
* 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 de memoria principal para mantener actualizada cada caché 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
* En fallo:
  + Write Allocate
    - La información se lleva de la memoria principal a la caché. Se sobre-escribe 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

**Anexo Clase 7**

Estructuras de Interconexión 🡪 Todas las unidades han de estar interconectadas

* Existe una serie de sistemas de interconexión.
* Las estructuras sencillas y múltiples son las más comunes

Bus 🡪 Es un camino de comunicación entre dos o más dispositivos. Suele agruparse:

* Varios caminos de comunicación o líneas con función común

Bus de Datos 🡪 Transmite datos (datos e instrucciones)

Bus de Dirección 🡪 Indica la fuente o destino de un ‘dato’

* El ancho del bus determina la máxima capacidad de memoria posible en el sistema.

Bus de Control 🡪 Transmite información de señales de control y temporización

* Señales de escritura/lectura en memoria
* Petición de interrupción
* Señales de reloj

¿Cómo son?

* Es un conjunto de conductores eléctricos paralelos
* Líneas de metal
* Poseen conectores para colocar ‘tarjetas’

Problemas de un único bus:

1. Conectar muchos dispositivos a un bus produce Retardos de Propagación
   1. Si el control pasa de uno a otro, puede afectar las prestaciones
2. La mayoría de los sistemas utilizan varios buses para solucionar esto
   1. Jerarquía de buses

Tipos de buses:

* Dedicados 🡪 Uso de líneas separadas para direcciones y para datos
  + 16 líneas de direcciones
  + 16 líneas de datos
  + 1 línea de control de lectura o escritura
* Multiplexados 🡪 Uso de las mismas líneas
  + 16 líneas de direcciones o datos
  + 1 línea de control de lectura o escritura
  + 1 línea de control para definir direcciones o datos

Arbitraje del bus:

* El control del bus puede necesitar más de un módulo
* Sólo una unidad puede transmitir a través del bus en un instante dado
* Clasificación:
  + Centralizado:
    - Un único dispositivo hardware es responsable de asignar tiempo en el bus: Controlador del bus o Árbitro.
  + Distribuido:
    - Cada módulo puede controlar el acceso al bus.
    - Cada módulo dispone de lógica para controlar el acceso.

Temporización 🡺 Forma de coordinar los eventos en el bus.

* Sincrónica:
  + La presencia de un evento está determinada por un reloj
  + El bus incluye una línea de reloj
  + Un intervalo desde un 1 seguido de otro a 0 es un ciclo de bus
  + Todos los dispositivos del bus pueden leer la línea de reloj
  + Suele sincronizar en el flanco de subida
  + La mayoría de los eventos se prolongan durante un único ciclo de reloj
* Asincrónica

Bus PCI 🡺 Interconexión de Componente Periférico

* Líneas de señal PCI
  + 49 obligatorias
    - Líneas del sistema
      * Incluyen reloj y reset
    - Terminales de direcciones y datos
      * 32 línea multiplexeadas para direcciones y datos
      * Líneas para interpretar y validar eventos
    - Terminales de control de la interfaz
      * Temporización y Coordinación
    - Terminales de arbitraje
      * Líneas no compartidas
      * Conexión directa al árbitro del bus PCI
    - Terminales para señales de error
  + 51 opcionales
    - Extensión a 64 bits
      * 32 líneas adicionales
      * Líneas multiplexeadas
      * 2 líneas para transferir a 64 bits

**Clase 8**

Procesador Escalar 🡪 Ejecución secuencial de instrucciones.

Segmentado de Cauce 🡪 Una unidad segmentada es una secuencia de etapas con la propiedad de que nuevas operaciones pueden iniciarse mientras otras están en proceso.

Mayores prestaciones:

* Procesadores Supersegmentados:
  + Muchas operaciones no necesitan todo un ciclo de reloj 🡪 Subdividir el ciclo de reloj en sub-intervalos.
  + División de las etapas “macro” del cauce segmentado en sub-etapas más pequeñas, y se transmiten los datos a la mayor velocidad del ciclo.
  + El tiempo para las instrucciones individuales no varía
    - Aumento el grado de paralelismo
    - Incrementa la aceleración percibida
* Procesadores Superescalares:
  + Se pueden llevar a cabo más de una instrucción simultáneamente.
  + Conlleva la duplicación de algunas o todas las partes de la CPU/ALU.
  + El grado de paralelismo y, por lo tanto, la aceleración de la máquina aumentan, ya que se ejecutan más instrucciones en paralelo.

Paralelismo de instrucciones 🡺 Existe cuando instrucciones de una secuencia son independientes y pueden ejecutarse en paralelos solapándose.

* Limitaciones:
  + Dependencia de datos verdadera
  + Dependencia relativa al procedimiento
  + Conflictos en los recursos
  + Dependencia de salida
  + Anti-dependencia

Paralelismo en la máquina 🡺 Es una medida de la capacidad del procesador para sacar partido del paralelismo de instrucciones.

* Depende de:
  + Nro de instrucciones captadas por ciclo
  + Nro de unidades funcionales
  + Mecanismos para localizar instrucciones independientes
    - Identificar paralelismo y organizar F, D y E en paralelo
    - Renombre de registros (dep. de salida o anti-dependencia)
    - Ventana de instrucciones (emisión desordenada)

Sobre las instrucciones:

* En la localización de instrucciones independientes
  + Orden en que se captan
  + Orden en que se ejecutan
  + Orden en que actualizan los registros y las posiciones de memoria
* Uso óptimo del cauce: Atender dependencias
* Políticas de emisión de superescalares:
  + Emisión y finalización en orden
  + Emisión en orden y finalización desordenada
  + Emisión y finalización desordenada

Renombre de Registros:

* Las dependencias de salida y anti-dependencias surgen porque los valores de los registros no pueden reflejar la secuencia de valores dictada por el flujo del programa 🡪 Conflictos de almacenamiento.
* Esto puede detener alguna etapa del cauce.
* Cuando la ejecución de una instrucción guarda un resultado en Registro, se almacena en un registro nuevo 🡪 Renombramiento de Registros.
* Los registros se asignan dinámicamente.

Implementación superescalar:

* Estrategias de captación simultánea de múltiples instrucciones.
* Lógica para determinar dependencias verdaderas entre valores de registros y mecanismos para comunicar esos valores.
* Mecanismos para iniciar o emitir múltiples instrucciones en paralelo.
* Recursos para la ejecución en paralelo de múltiples instrucciones.
* Mecanismos para entregar el estado del procesador en un orden correcto.

Consideraciones:

* Influencia de las excepciones
  + Excepciones imprecisas 🡺 En el momento en que se produce una excepción, hay varias instrucciones en ejecución 🡪 Estado inconsistente
    - Para garantizar un estado consistente (preciso):
      * Instrucciones anteriores terminan correctamente
      * La que origina la excepción y siguientes se abortan
      * Tras la rutina de tratamiento se comienza por la que originó la excepción
  + Excepciones precisas 🡺 Con las interrupciones externas
    - La unidad de emisión deja de emitir y se cancela la cola
    - Todas las instrucciones pendientes se completan
  + Compromiso ente:
    - Ejecución desordenada 🡨🡪 Liberar las unidades de ejecución
    - Completar instrucciones en orden 🡨🡪 Excepciones precisas
    - Posible solución: Emisión desordenada y finalización ordenada

**Clase 9**

Procesamiento Paralelo:

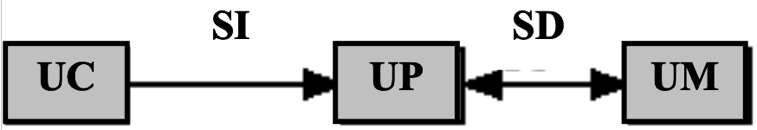
* Mejorar el rendimiento de una máquina con un solo procesador 🡪 Paralelismo a nivel de instrucción – ILP
* Arquitecturas de sistemas con varios procesadores 🡪 Paralelismo a nivel proceso

Categorías de Computadoras:

* SISD 🡪 Una secuencia de instrucciones y una secuencia de datos
* SIMD 🡪 Una secuencia de instrucciones y múltiples secuencias de datos
* MISD 🡪 Múltiples secuencias de instrucciones y una secuencia de datos
* MIMD 🡪 múltiples secuencias de instrucciones y múltiples secuencias de datos

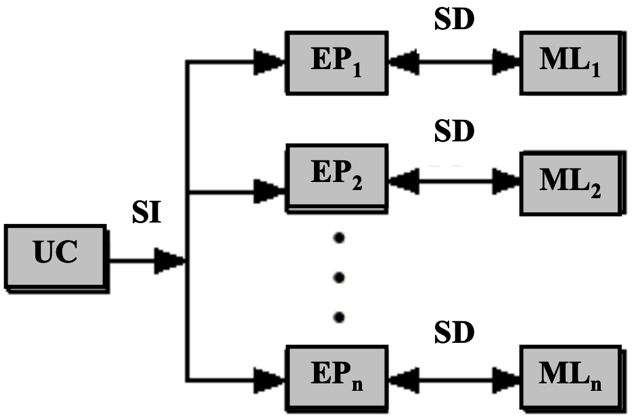
SISD:

* Un único procesador interpreta una única secuencia de instrucciones (SI)
* Datos almacenados en una única memoria (UM)
* Computadoras monoprecesador
* Organización:



SIMD:

* Una única instrucción máquina controla paso a paso la ejecución simultánea de un cierto número de elementos de proceso (EP)
* Cada elemento de proceso tiene una memoria dedicada (ML)
* Cada instrucción es ejecutada por cada procesador, con un conjunto de datos deferentes
* Procesadores vectoriales y matriciales
* Organización:

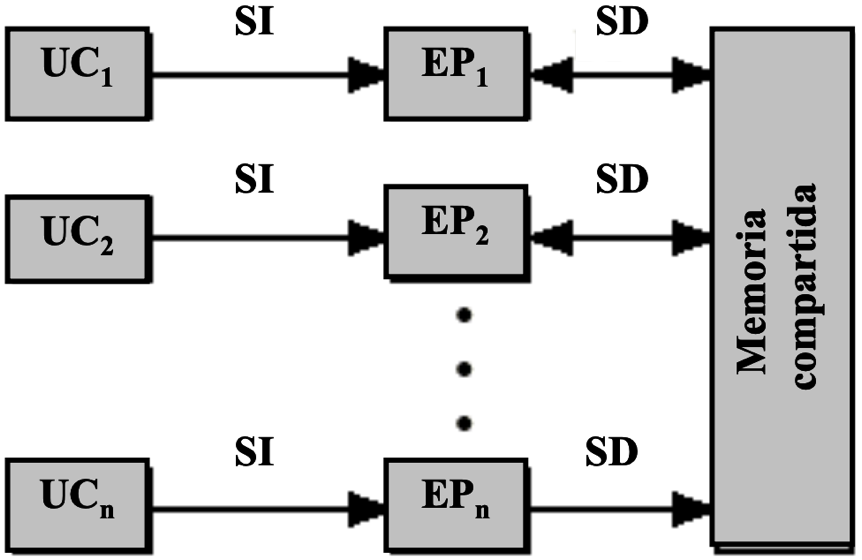


MISD:

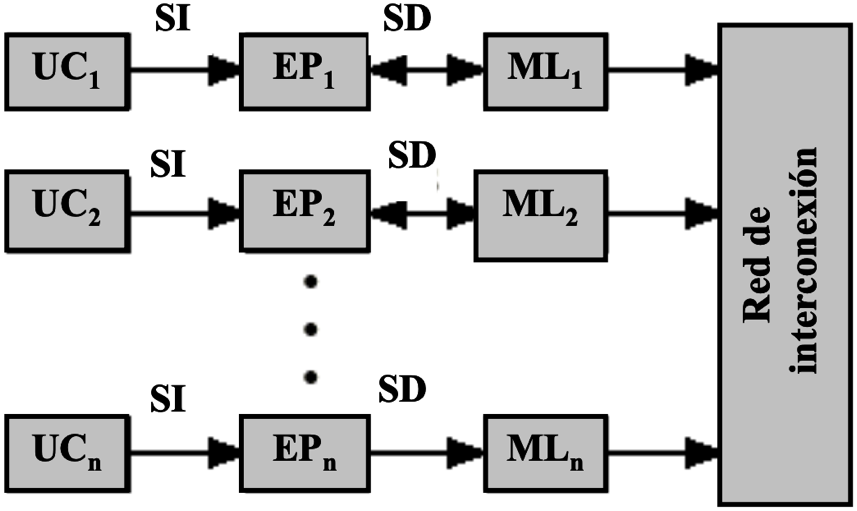
* Se transmite una secuencia de datos a un conjunto de procesadores
* Cada procesador ejecuta una secuencia de instrucciones diferente
* Esta estructura nunca ha sido implementada

MIMD:

* Un conjunto de procesadores ejecuta secuencias de instrucciones diferentes en simultáneo
* Conjuntos de datos diferentes
* Se pueden dividir según la forma de comunicarse
  + Memoria compartida: SMP y sistemas NUMA

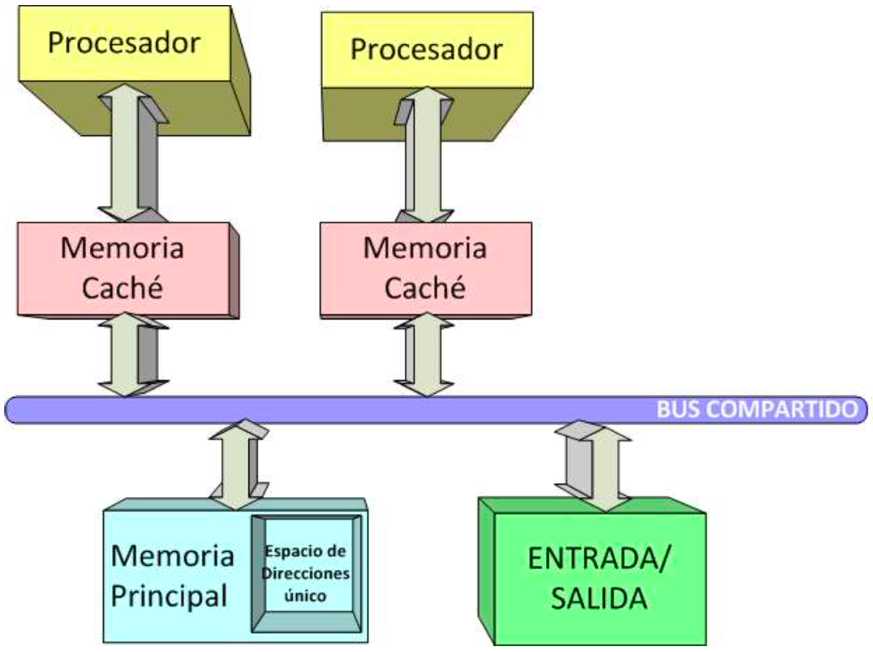


* + Memoria distribuida: Clusters



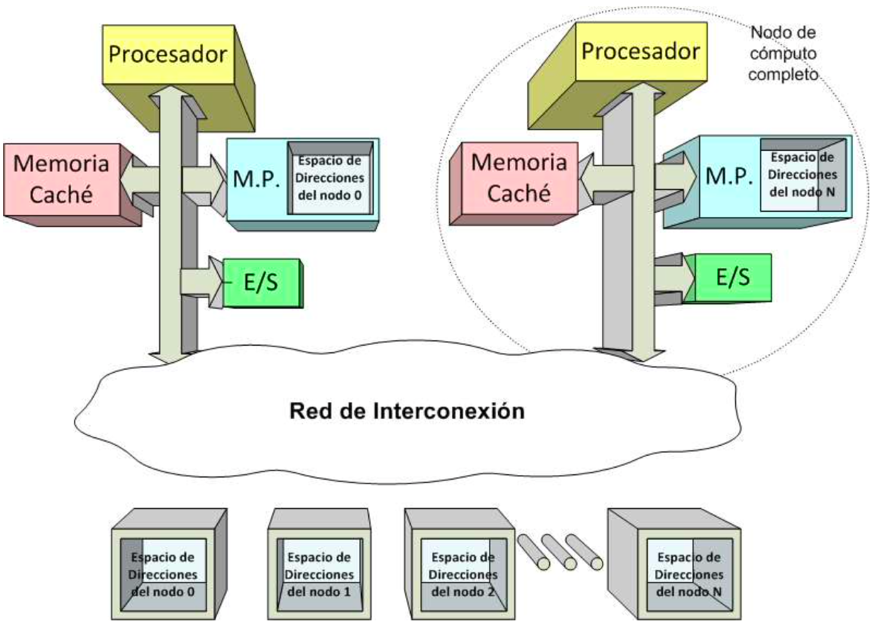
Multiprocesador Simétrico – SMP

* Computadora autónoma con las siguientes características:
  + Dos o más procesadores similares de capacidades comparables
  + Comparten la memoria principal y las E/S
  + Interconectados mediante un bus u otro tipo de sistema de interconexión
  + Tiempo de acceso a memoria similar para todos los procesadores (UMA)
  + Todos los procesadores pueden desempeñar las mismas funciones
  + Sistema operativo integrado, que proporciona la interacción entre los procesadores y sus programas
* Ventajas potenciales:
  + Mayores prestaciones: Si el trabajo a realizar puede organizarse en paralelo.
  + Buena disponibilidad: Un fallo en un procesador no detendrá la computadora.
  + Crecimiento incremental: Se pueden añadir más procesadores.
  + Escalado: En función de la cantidad de procesadores.
* Desventajas:
  + Bus de tiempo compartido:
    - La prestación está limitada por el tiempo de ciclo del bus.
    - Cada procesador debería estar equipado con una memoria cache para mejorar las prestaciones: Se reducirá el nº de accesos
    - Se pueden producir problemas de coherencia de cache:
      * Este problema debe ser resuelto por el hardware
        + Protocolos de sondeo y protocolos de directorio
* Arquitectura



Clusters 🡺 Computadoras completas interconectadas que trabajan conjuntamente como un único recurso

* Cada computadora se denomina “nodo”.
* Prestaciones y disponibilidad elevadas.
* Aplicaciones propias de un servidor.
* Son la alternativa a los SMP.
* Beneficios:
  + Escalabilidad absoluta
  + Escalabilidad incremental
  + Alta disponibilidad
  + Mejor relación precio/prestaciones
* Arquitectura



Cluster vs SMP:

* Ambos:
  + Dan soporte a aplicaciones de alta demanda de recursos
  + Disponibles comercialmente
* SMP:
  + Más fácil de administrar y configurar
  + Cercano a los sistemas de un solo procesador
    - La planificación (scheduling) es la diferencia principal
    - Menos espacio físico / Menor consumo de potencia
* Cluster:
  + Superior escalabilidad incremental y absoluta
  + Superior disponibilidad 🡪 Redundancia

UMA – Uniform Memory Access

* Igual tiempo de acceso a todas las regiones de memoria.
* Igual tiempo de acceso a memoria para los diferentes procesadores.

NUMA – Nonuniform Memory Access

* El tiempo de acceso de un procesador difiere dependiendo de la región de memoria que accede.
* Diferentes procesadores acceden a diferentes regiones de memoria a diferentes velocidades.

CC-NUMA – Cache Coherente NUMA

* Es una NUMA que mantiene coherencia de cache entre las cache de los distintos procesadores.

Motivación NUMA:

* SMP tiene límite práctico en su número de procesadores
* En clusters cada nodo tiene su propia memoria principal
* NUMA retiene las características tipo SMP y brinda multi-procesamiento a gran escala.
* Objetivo NUMA 🡪 Tener una memoria transparente del sistema y permitir nodos, cada uno con su propio bus o sistema de conexión interna.

Operación CC-NUMA:

* Cada procesador tiene cache L1 y L2
* Cada nodo tiene su propia memoria principal
* Nodos conectados por algún tipo de red
* Cada procesador ‘ve’ un único espacio de direcciones de memoria
* Orden de acceso a memoria:
  + Cache L1 (local al procesador)
  + Cache L2 (local al procesador)
  + Memoria principal (local al nodo)
  + Memoria remota (petición por red)
* Automático y transparente

Procesamiento Multihebra (Multithreading)

* Aumento de paralelismo de instrucciones.
* La secuencia de instrucciones se divide en secuencias más pequeñas llamadas hebras (threads) que pueden ejecutarse en paralelo.
* Amplia variedad de diseños multihebra.

Proceso 🡪 Un programa ‘corriendo’ en una computadora

* Propiedad de Recursos: Espacio de direcciones virtuales para almacenar la imagen de un proceso.
* Planificación/ejecución: Hay camino de ejecución (traza).
* Conmutación de Proceso (Process switch)

Hebra (Thread) 🡪 Unidad de trabajo de un proceso que puede asignarse

* Incluye un contexto de procesador (incluido PC y SP) y área de datos para su pila (stack).
* Se ejecuta secuencialmente.
* Interrumpible. El procesador cambiaría a otra hebra.
* Conmutación de hebra (Thread switch)
  + Cambio de control del procesador entre hebras de un mismo proceso

Multi-hebra

* Explícito:
  + Ejecución concurrente de instrucciones de diferentes hebras explícitas
    - Mezcla de instrucciones de diferentes hebras en cauces compartidos
    - O por ejecución paralela en cauces paralelos
  + Todos los procesadores comerciales lo usan
* Implícito:
  + Ejecución concurrente de varias hebras extraídas de un único programa secuencial
    - Definidas estáticamente por el compilador o dinámicamente por el hardware

Procesador Multi-hebra:

* PC distinto para cada hebra que pueda ejecutarse concurrentemente
  + Cantidad y tipo de HW para ejecución concurrente
* Se trata cada hebra de forma separada
  + Paralelismo entre hebras
* Aproximaciones con ejecución simultánea real
  + Multi-hebra simultánea (SMT) – Pentium 4 HT
  + Multiprocesador monochip