**RESUMEN ARQUITECTURA DE COMPUTADORAS**

[MODOS DE DIRECCIONAMIENTO 3](#_Toc126673387)

[ENTRADA / SALIDA 3](#_Toc126673388)

[**Métodos de conexión E/S** 4](#_Toc126673389)

[**Técnicas de gestión de transferencia E/S** 4](#_Toc126673390)

[**INTERRUPCIONES** 5](#_Toc126673391)

[**Ciclo de instrucción:** 5](#_Toc126673392)

[**Prioridad de interrupciones:** 5](#_Toc126673393)

[**Reconocimiento de periféricos:** 6](#_Toc126673394)

[**DMA (DIRECTO A MEMORIA)** 6](#_Toc126673395)

[**Etapas de transferencia:** 6](#_Toc126673396)

[**Tipos de transferencia:** 7](#_Toc126673397)

[**Canales de E/S:** 7](#_Toc126673398)

[SEGMENTACION DE INSTRUCCIÓNES 8](#_Toc126673399)

[**Fases del pipelining:** 8](#_Toc126673400)

[**TECNOLOGIAS DE PROCESADORES** 8](#_Toc126673401)

[**Distintos tipos de procesadores:** 8](#_Toc126673402)

[**Métodos de emisión de instrucciones:** 9](#_Toc126673403)

[**Políticas de emisión:** 9](#_Toc126673404)

[**Estados de instrucción:** 10](#_Toc126673405)

[**Implementación Súper-Escalar:** 10](#_Toc126673406)

[**RIESGOS DEL PARALELISMO (ILP)** 10](#_Toc126673407)

[**Soluciones:** 11](#_Toc126673408)

[**TÉCNICAS DE OPTIMIZACIÓN** 13](#_Toc126673409)

[SEGMENTACION DE MEMORIA 14](#_Toc126673410)

[**Jerarquía de memoria:** 14](#_Toc126673411)

[**Clasificación de cache:** 14](#_Toc126673412)

[**Métodos de correspondencia** 15](#_Toc126673413)

[**Políticas de remplazo** 16](#_Toc126673414)

[**Política de escritura** 16](#_Toc126673415)

[BUSES DEL SISTEMA 18](#_Toc126673416)

[**Partes del bus:** 18](#_Toc126673417)

[**Jerarquía de buses:** 18](#_Toc126673418)

[**Tipos de buses:** 18](#_Toc126673419)

[**Tipos de arbitrajes:** 19](#_Toc126673420)

[**Tipos de sistemas** 19](#_Toc126673421)

[**Bus PCI:** (Peripheral Component Interconect) 20](#_Toc126673422)

[**Bus SCSI:** (Small Computer System Interface) 20](#_Toc126673423)

[PROCESAMIENTO PARALELO 21](#_Toc126673424)

[Taxonomía de Flynn 21](#_Toc126673425)

[**Sistemas de alta demanda** 21](#_Toc126673426)

[**Acceso a memoria:** 22](#_Toc126673427)

[SET DE INSTRUCCIÓNES 23](#_Toc126673428)

[**CISC (Complex Instruction Set Computer)** 23](#_Toc126673429)

[**RISC (Reduced Instruction Set Computer)** 23](#_Toc126673430)

[SISTEMA VON-NEUMANN 24](#_Toc126673431)

[**Sistema Harvard:** 24](#_Toc126673432)

# **MODOS DE DIRECCIONAMIENTO**

Existen distintos métodos de pasaje de parámetros (argumentos) a procedimientos o funciones:

-Por Registro  
 Ya que cada procedimiento usa una pequeña cantidad de registros, se particiono la memoria de registros en ventanas de registros. Cada procedimiento contiene su propia ventana y ciertos registros de esta ventana se solapan con registros de otros procedimientos como una forma de “pasarse” parámetros.   
 Cada llamada a subrutina genera una nueva ventana de registros. Cada ventana de registro posee 3 tipos de registros. Registros de Parámetros, Registros Locales y Registros Temporales.  
 Registros de Parámetros: Posee los registros temporales que recibió del procedimiento que lo llamo.

Registros Locales: Son registros locales al procedimiento / función.

Registros Temporales: Se usa para intercambiar parámetros con el procedimiento llamado. Los registros Temporales son físicamente los mismos que los Registros de Parámetro del nivel más bajo (Es decir de la rutina que llamó).

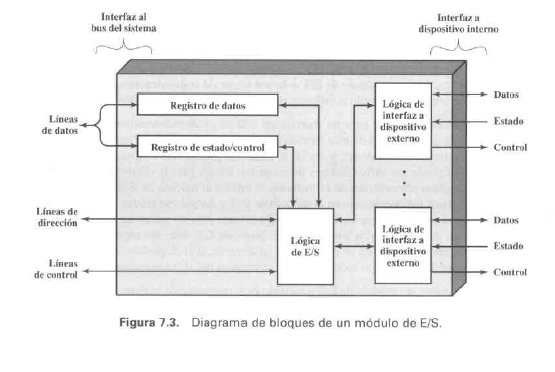
-Por Memoria  
 Se usa un área definida en la RAM, difícil de estandarizar.

-Por Pila  
 Es un conjunto ordenado de elementos, al que solo se puede acceder al puntero de la pila.   
 Hay que tener en cuenta el SP (puntero de pila), el BP (base de pila) y el límite de la pila.  
 Es el verdadero pasaje de parámetros. Se usa la pila / stack con su push y pop.  
 Es utilizado por el usuario y por el sistema.  
 Operaciones unarias: Trabaja con el puntero de la pila y guarda el resultado ahí.  
 Operaciones binarias: Trabaja con los 2 últimos, los borra y guarda el resultado   
 en la cabecera.

# **ENTRADA / SALIDA**

Los dispositivos de E/S son siempre más lentos que la CPU y la RAM, por lo que se diseñaron distintos sistemas para que esta lentitud afecte lo mínimo posible al sistema.  
  
Por esta razón se crearon los módulos de E/S, dicho modulo se comunica con la CPU y los periféricos mediante el puerto de E/S, por dicho puerto mandan la información de **control** (entrada, entrada/salida o salida), **estado** (listo / ocupado) y **datos**. Así mismo el módulo de E/S posee un **control lógico** (para el direccionamiento de datos), un **transductor** (para conversión de datos) y un **buffer** (para la retención y adaptación de 1, 8, 16, Etc. bits).

La función del módulo E/S es que el procesador vea los dispositivos pero de una forma simple.

Es decir el módulo debe guardar y ocultar los datos y detalles de los dispositivos E/S. Como por ejemplo los registros de datos con direcciones de los dispositivos, configuración de estado, etc.

Dejando al CPU únicamente términos de órdenes de lectura / escritura.

### **Métodos de conexión E/S**

**E/S Asignada en Memoria** (memory-wraped)  
 Las datos de E/S se encuentran en la memoria, es decir la entrada de datos mediante el bus de direcciones, esta mapeada directamente a la memoria. No se distingue la parte de E/S de la parte de memoria, por lo que se pueden usar todo tipo de instrucciones. La desventaja es que ocupa espacio de memoria.

**E/S Aislada** Las posiciones de E/S están separadas de la memoria del sistema por lo que se necesitan líneas especiales para memoria y para E/S e instrucciones para acceder a la memoria E/S (in, out, mov). Esto da la ventaja de dejar totalmente libre la memoria del sistema.

### **Técnicas de gestión de transferencia E/S**

**E/S programada con espera de respuesta** En este caso la CPU tiene el control completo de E/S, debe ejecutar un programa para la ejecución de la operación de E/S. En esta técnica la CPU se encarga de inicializar el periférico con los datos de control requeridos, posteriormente debe hacer continuamente consultas de estado para ver cuando el dispositivo está listo para enviar/recibir datos, es decir que la CPU quedaría en un estado ocioso hasta que el periférico ese disponible.

**E/S con interrupciones** El CPU envía un pedido de E/S, posteriormente cuando se haya cumplido el pedido y los datos estén listos, el modulo E/S envía un pedido de interrupción a la CPU (mientras tanto la CPU puedo seguir ejecutando otras instrucciones), en este momento la CPU pausa su cauce y se hace cargo de la trasferencia de datos a memoria o al dispositivo, posteriormente sigue con el cauce normal del programa.

**E/S con acceso directo a memoria (DMA)** El DMA logra imitar al CPU, es decir que tiene la ventaja de que pude controlar la transferencia de datos a memoria directamente sin el uso de la CPU. El DMA es capaz de solicitar el uso del bus, especificar la dirección de memoria y configurar el control (tipo de operación y señales de sincronización).

**Evolución de dispositivos E/S:**

- CPU controla periféricos.

- Se crea un dispositivo de E/S, la CPU se libera de datos específicos de dispositivos E/S.

- Se le añade las interrupciones.

- Se crea el DMA.

- Se le añade al DMA un procesador (CDMA), el cual se encarga de las instrucciones E/S.

- Se le agrega una memoria local.

Se termina transformando en una mini computadora.

## **INTERRUPCIONES**

Una interrupción es el mecanismo mediante el cual otros módulos (E/S, memoria) pueden interrumpir el procesamiento normal de la CPU.

Mejoran la eficiencia del procesador liberando a este hasta que los dispositivos de E/S estén listos para pasar / recibir datos. Cuando los dispositivos E/S están listos piden interrumpir, por lo que el programa para su cauce normal y atiende al dispositivo.

**Controlador de interrupciones / PIC:** (20h)Es el encargado de gestionar las interrupciones, con su 8 niveles de interrupción y sus direcciones de subrutina, mediante el el IMR, el IRR y el ISR. Los 8 niveles de interrupción son:  
 (Int 0) Sistema / F10, (Int 1) timer, (Int 2) handshake, (Int 3) DMA, (Int 6) read y (Int 7) write.

**Tipos de interrupciones:  
Generadas por Software** (Internas)  
Usadas para llamar a funciones del sistema operativo (OS).

**Excepciones / Traps:**  
 Son interrupciones pero de alta prioridad y generadas por el CPU, dichas interrupciones se generan cuando hay un error importante como falla de programa, de hardware o acceso a datos de memoria.

**Generadas por Hardware** (Externas a la CPU)  
 Son las verdaderas interrupciones, generadas por los dispositivos E/S

### **Ciclo de instrucción:**

**-Captación de Instrucción.**   
 Al finalizar una instrucción se comprueba los pedidos de interrupción, en caso de que haya algún pedido se guarda en memoria los datos del programa que se venía ejecutando (Dir. Prox. Instrucción y estado de CPU) y se carga la Dir. de comienzo de la interrupción. Se inhiben otras interrupciones.

**-Ejecución de Instrucción**  
 Se ejecuta la interrupción y en su finalización se reactivan las interrupciones.

**-Gestión de Interrupción:** En caso de que haya más de un pedido de interrupción, estos se van ejecutando en orden de prioridad y cuando se hayan ejecutado en su totalidad se vuelve a la ejecución del programa principal. Para esto se carga los datos previamente guardados (Dir. de la próxima instrucción y estado del CPU).

### **Prioridad de interrupciones:**

**Procesamiento de interrupciones secuenciales:** Se procesan conforme van llegando. Cuando se termina una se ejecuta la otra.

**Procesamiento de interrupciones anidadas (Priorización):** Si se recibe una interrupción de prioridad mayor, se ejecuta inmediatamente y posteriormente al finalizarla se sigue con la interrupción anterior. Método óptimo.

### **Reconocimiento de periféricos:**

**Por Interrupciones multi-nivel:**  Es muy caro pero consiste en que cada dispositivo que puede generar una interrupción tiene una entrada física al CPU, cada entrada tiene un nivel de prioridad con el cual puede o no interrumpir a otra interrupción.

**Línea de interrupción única:** (Consulta de software)  
 Se posee una única entrada común a los dispositivos, y se procede a preguntar a todos los dispositivos si solicitaron una interrupción (Polling/Encuesta).

**Interrupciones Vectorizadas:** Usan una línea de pedido común todos los dispositivos, cuando se realiza el pedido de interrupción la CPU busca y guarda un identificador (vector) del dispositivo que lo solicito, este vector suele ser la dirección del módulo o algún identificador específico. Dicha dirección / vector es colocado en el bus de datos por el dispositivo que lo solicito.

## **DMA (DIRECTO A MEMORIA)**

El DMA permite al módulo E/S el acceso directo a memoria, sin requerir al CPU más que al comienzo y al final de la transferencia.  
  
Posteriormente con el CDMA, la DMA recibe un conjunto de instrucciones las cuales son ejecutadas en su totalidad y posteriormente recién se le informa al CPU de su finalización.

### **Etapas de transferencia:**

**-Inicialización de la transferencia** – Se establecen los parámetros de transferencia

Inicialización de la interfaz – La CPU establece a interfaz  
 N° byte, tipo transferencia (R/W) y datos control (pista, sector, etc.).

Inicialización de DMA – La CPU establece a DMA  
 N° bytes, tipo transferencia (R/W), Dir. Memoria y canal (dependiendo).

-**En este momento la CPU ya se despreocupa de la transferencia.**

**-Realización de la transferencia** – Cuando el periférico le informa al DMAC que está listo.

El DMAC pide control del bus e inicia transferencia  
 Después de cada palabra se actualizan n° bytes o palabras y dirección.

-**Finalización de transferencia**

DMAC libera bus y devuelve control a CPU.

Suele informar a CPU mediante interrupción, el CPU pierde un ciclo.

### **Tipos de transferencia:**

**Por RAFAGAS** (Burst):  
 La DMA usa el bus para transferir todos los datos de una rápidamente sin tener en cuenta si la CPU necesita hacer uso del mismo para acceder a memoria. Por lo tanto se sufre un degradamiento del rendimiento del sistema, pero la transferencia se realiza más rápido.

**Por ROBO DE CICLO** (Cycle-stealing):  
 La DMA hace uso del bus en los periodos de instrucciones en las que el CPU no va a necesitar entrar a memoria, ya sea porque posee los datos en cache o por que dicha instrucción no requiere de la misma. La DMA transfiere una palabra y vuelve a preguntar si puede seguir hasta terminar con todos los datos. Este método no degrada la velocidad el sistema, pero la velocidad de transferencia de datos E/S es más lenta. Por lo tanto este método es más eficiente para transferencias de múltiple palabras.

### **Canales de E/S:**

Son una extensión del DMA, con la completa capacidad y autonomía de ejecutar instrucciones E/S.

Proceso:  
 - CPU ordena inicio de ejecución de programa en memoria.

- Programa indica dispositivos, áreas de memoria, prioridades y acciones.

- El Selector o Multiplexor se encargar de todo lo relacionado E/S

**Tipo de Canales:**

**Selector:** Controla varios dispositivos uno por vez, dicho canal va a transferir la totalidad de un dispositivo el cual va a ser manejado en su totalidad por el controlador o módulo de E/S.

**Multiplexor:** Puede manejar E/S con varios dispositivos simultáneamente. Posee 2 formas, multiplexor de bytes (acepta y transmites caracteres) y multiplexor de bloques (intercala bloques de datos de distintos dispositivos).

# **SEGMENTACION DE INSTRUCCIÓNES**

El pipelining es una técnica de hardware invisible al nivel programador la cual logra aumentar la productividad (through-put) más no la velocidad de ejecución de una instrucción, consiste en la separación de las instrucciones en 5 fases para la posible ejecución en simultaneo de múltiples instrucciones. Para lograr esto se hace que cada instrucción realice una fase distinta al mismo tiempo, por lo que se logra estar ejecutando hasta 5 instrucciones simultáneamente.

### **Fases del pipelining:**

**BUSQUEDA (F, fetch, MI)**

Se accede a memoria.

Se incremente PC.

**DECODIFICACION (D, decode, BR)**

Se decodifica la instrucción.

Se accede a los registros por el/los operandos (si es necesario).

Se calcula el valor del operando inmediato (si hace falta).

**EJECUCIÓN (E, execute, ALU)** Se ejecuta la operación en ALU

**ACCESO A MEMORIA (M, memory Access, MD)** Se accede a memoria (si es necesario).

**ALMACENAMIENTO (W, writeback, BR)** Se guarda el resultado en un registro (si es requerido).

## **TECNOLOGIAS DE PROCESADORES**

### **Distintos tipos de procesadores:**

**-Procesador Escalar:** Ejecución secuencial.

-**Procesador Escalar Segmentado:** Con segmentación del cauce (pipeline).

**-Procesador Súper-Escalares:** Se le denomina procesadores súper-escalares a los procesadores que tienen duplicadas algunas o todas las unidades funcionales de la CPU/ALU, de esta forma se logra llevar a cabo más de un instrucción simultáneamente.

**-Procesador Súper-Escalares Segmentados:** (se usan en la actualidad)Son Súper-Escalares pero con segmentación del cauce. Es decir vamos a tener una pipeline con hasta 2 procesos de búsqueda simultáneos, 2 decodificación, 2 Ex, 2 M y 2 WB. Tiene un gran grado de paralelismo.

**-Procesador Súper-Segmentado:** (se usan en la actualidad)Son procesadores en los cuales muchas etapas van a tardar menos de un ciclo en la ejecución, por lo que en un ciclo se realizan por Ej. 2 decodificaciones. Es decir permite aumentar la frecuencia del ciclo del reloj. No tienen duplicadas las unidades funcionales. Tiene un gran grado de paralelismo.

**-Multi-Hebra (Multi-Threading)** [[1]](#footnote-1)**:** (se usa en la actualidad)Se caracteriza en que separa los procesos en threads (hebras) los cuales se van ejecutando secuencialmente, pero pueden ser interrumpidas y remplazadas por otros threads de otro proceso. Cada thread incluye el contador de programa, puntero de pila y su propia área para datos en la pila. Así mismo también existen Súper-Escalar Multi-Hebra Entrelazado, VLIM Multi-Hebra con bloqueo, etc. Son lo más rápido.   
 Existe multi-hebra explicito y implícito.

Estos nuevos procesadores (Súper-Escalares, Súper-Segmentados y Multi-Hebra) requieren una nueva forma de ordenar las instrucciones, requieren de un Paquete de Emisión del cual sus instrucciones deben poder ser ejecutadas simultáneamente.

Esto lograra una mayor velocidad pero generaran más riesgos estructurales, para esto se crearon métodos de Emisión de Instrucciones.

### **Métodos de emisión de instrucciones:**

**Estáticas**:   
 Este grupo de instrucciones esta generado por el compilador.   
 Este genera el paquete de emisión evitando riesgos de datos y control.

**Dinámicas:** En este caso el paquete es creado por el CPU.   
 Este modelo es usado por las Súper-Escalares.

### **Políticas de emisión:**

Puede ser el caso que muchas instrucciones cumplan lo requerido para ser emitidas, pero que no todas puedan ser emitidas en simultáneo. Por otro lado el paralelismo requiere un orden de instrucciones más específico para una velocidad óptima. Por lo que se suele reposicionar las instrucciones, con la única restricción de que el resultado sea el correcto.  
Para esto se tiene en cuenta 3 cosas; El orden en que se captan las instrucciones, el orden en el que se ejecutan y el orden en que actualizan los registros y las posiciones de memoria.

**Emisión ordenada y finalización ordenada:** Las instrucciones se ejecutan y escriben ESTRICTAMENTE en el orden que están, si una se para, las siguientes también.   
 Formato común, ejecución secuencial.

**Emisión ordenada y finalización desordenada:** La finalización desordenada mejora la velocidad de las instrucciones que necesitan muchos ciclos ya que no se requiere esperar a la instrucción anterior para terminar. Permite la máxima cantidad de paralelismo de la máquina.

La emisión se para cuándo hay alguna dependencia. Aparecen las dependencias WAW.  
 Requiere una lógica más compleja. Complica la gestión de las interrupciones.

**Emisión desordenada y finalización desordenada:** Se agrega la emisión desordenada, para esto se agrega un buffer llamado ventana de instrucciones la cual se ubica entre la etapa de decodificación y de ejecución. Esta se encarga de la ordenación y emisión de instrucciones independientes. Las instrucciones se ejecutan cuando cumplan las condiciones requeridas (UF y operandos requeridos). Aparecen atascos WAR.

Es decir permite anticipar las instrucciones, teniendo en cuenta los posibles atascos, logrando un paralelismo máximo.

En los procesadores segmentados la etapa ID (Instrucción Decodification) se particiona en 3 partes Pre-Decodificación, Decodificación (Dispatcher / ventana de instrucciones) y Emisión (Issue). Una vez que la instrucción llega a la ventana de instrucciones, esta se encarga de esperar a la disponibilidad de los operandos y unidades funcionales requeridas, si ambas se cumple la ventana realiza la emisión.

### **Estados de instrucción:**

**Distribuida:** Instrucción enviada a estación de reserva (ventana de instrucciones).

**Emitida:** Sale de estación de reserva hacia unidad funcional.

**Finalizada:** Sale de unidad funcional y pasa a buffer de reordenamiento con el resto de registros, donde todavía no es visible por el programador.

**Terminada:** Cuando ya ha sido escrito en un registro con los renombramientos necesarios, ya es visible al programador.

**Retirada:** Cuando, en caso de ser necesario, el dato ya es escrito en memoria.

### **Implementación Súper-Escalar:**

-Estrategia de captación simultanea de múltiples instrucciones.  
 -Lógica para determinar dependencia verdadera entre registros.  
 -Mecanismos para iniciar múltiples instrucciones en paralelo.  
 -Recursos para la ejecución de múltiples instrucciones en paralelo.  
 -Mecanismos para entregar el estado del procesador en orden correcto.

En caso de excepciones imprecisas (errores), se terminan de ejecutar las instrucciones en proceso y se corta la fila de instrucciones. Una vez terminada las instrucciones en ejecución, se ejecuta la instrucción que dio error y posteriormente se reanuda el cauce.

## **RIESGOS DEL PARALELISMO (ILP)[[2]](#footnote-2)**

Dado que no todas las instrucciones necesitan todas las etapas, las etapas F y M requieren acceso a memoria, existen los saltos de control (los cuales cambian la siguiente instrucción), entre otras cosas esto puede generar errores en el programa.

**Atasco del cauce (stall)**:

**Estructural:** (Dependencia del sistema)Provocado por conflicto de recursos, cuando 2 instrucciones requieren del mismo recurso Ej.: Memoria, ALU, etc.

**Por dependencia de control:** (Dependencia relativa al procedimiento)  
 Cuando la ejecución de una instrucción depende de cómo se ejecute otra (Cuando se realiza un salto, con 2 posibles caminos).

**Incondicionales:** La dirección destino debe ser calculada pero es fija, un solo resultado es posible.

**Condicionales:** La dirección destino va a variar según el resultado de la instrucción previa. Por lo que le programa puede proseguir en 2 caminos.

**Por dependencia de datos:** Cuando 2 instrucciones utilizan el mismo dato. Hay 3 tipos:

**RAW (lectura después de escritura):** (Dependencia Verdadera) Cuando una instrucción intenta leer un dato que la instrucción anterior tiene que modificar. Ej.: Cuando 2°i [[3]](#footnote-3) requiere un dato que modifica 1°i.

**WAW (escritura después de escritura):** (Dependencia de Salida)Una instrucción escribe un dato después que una posterior (Ej. P.F.).  
 Ej.: Cuando una 2°i va a escribir en R3 antes que 1°i escriba en R3, queda valor incorrecto.

**WAR (escritura después de lectura):** (Anti-dependencia)Una instrucción modifica un valor antes que una posterior lo lea.  
 Ej.: Cuando una 2°i cambia el valor de un registro que debe usar una 1°i.

### **Soluciones:**

**Riesgos estructurales:** (simple) **-**Duplicación de recursos de Hardware (Se agrega un Sumador y restador a la ALU).  
 -Separación de memoria de instrucciones y datos.  
 -Turnar acceso a Registros (Escritura en primera mitad de ciclo, lectura 2nda).

**Riesgos de dependencia de datos:**

**Riesgos RAW:** Mediante una unidad de riesgos o un compilador se detecta si:  
 HARDWARE:  
 -Mediante fowarding.[[4]](#footnote-4)

SOFTWARE:  
 -Instrucciones NOP.  
 -Reordenación del código.

**Riesgos WAW y WAR:**  Renombramiento de registros.

**Riesgos de control:  
 HARDWARE:**

**Detención del cauce:** Se basa en que después de un salto siempre se retrasa un ciclo el inicio de la próxima instrucción.

**Mejora de ruta de datos: (fowarding)** Se adelanta la llegada del resultado, sacado el resultado directamente de la salida de la ALU, de esta manera se consigue un atraso de 1 ciclo.

**Métodos de predicción de salto:**

**Técnica estática:  
 Predicción de que se / no se salta:** Consiste en la “predicción” de que nunca va a saltar, o siempre va a saltar.   
 Es decir que siempre va empezar a hacer la siguiente operación del caso que sea.

**Predicción según código de operación:** Hay instrucciones que son más probables a saltar que otras, este sistema llega a un 75% de acierto.

**Técnica dinámica:**

**Conmutador:** Guarda el resultado de la última vez que se ejecutó el salto, bueno para bucles.

**Tabla de historia de saltos** (BTB, branch-target buffer):  
 Posee una pequeña memoria muy rápida donde se guardan direcciones de la instrucción de bifurcación tomada, instrucción o dirección destino e historia de uso.

**Flujos múltiples:** Se pre-capta todos los cauces y sus respectivos saltos, posteriormente se elige el correcto. Posee la desventaja que provoca retardos en el acceso al bus y registros.

**Pre-captar el destino de salto:** Se pre-capta la instrucción destino al salto así como también las siguientes instrucciones a la bifurcación, hasta que se ejecute la instrucción bifurcación.

**Buffer de bucles:** Posee un buffer muy rápido que trabaja en la etapa de captación de instrucciones. Consiste en que comprueba el buffer, donde guarda las instrucciones previas, antes de hacer la captación de memoria y de esta forma analiza si saltara o no.  
 Muy eficaz para pequeños bucles y saltos.

**SOFTWARE:**

**Salto retardado:** (delay slot)Se basa en adelantar una instrucción (que no repercuta al salto) para ir realizando alguna operación en el tiempo que se espera el resultado para el salto, en caso de que no se pueda adelantar una instrucción se coloca un NOP.

**Reordenamiento del Codigo:** Se debe reordenar el código manualmente para evitar dependencia, o en caso de no ser posible insertar una instrucción NOP.

## **TÉCNICAS DE OPTIMIZACIÓN**

Hay 3 técnicas fundamentales de optimización: Duplicación de recursos, Emisión desordenada y renombramiento de registros.

**DUPLICACIÓN DE RECURSOS:**  
 Es la forma en la que funciona los procesadores Súper-Escalares, el cual funciona con la duplicación de recursos y el dispatcher como cerebro.

**EMISION Y FINALIZACIÓN DESORDENADA:**

**RENOMBRAMIENTO DE REGISTROS**:  
 El Renombre de registros consiste en que cuando una instrucción modifica un registro, se guarda el valor en un registro nuevo y se actualiza la dirección de referencia hacia el nuevo registro, es decir se van re asignando dinámicamente los registros. Quedando por Ej. R1a y R1b, y las instrucciones posteriores que necesitan acceder a R1 van recibiendo la dirección del nuevo registro R1 (R1b).

# **SEGMENTACION DE MEMORIA**

Siempre se busca la máxima capacidad de memoria más rápida posible.

Es decir que sería ideal si se pudiese tener una memoria infinita con la velocidad de los registros, pero si tenemos muchos registros deberíamos utilizar más bytes para las direcciones de los registros, lo cual ralentizaría la velocidad de lectura hacia los mismos.

Por lo tanto lo que se hizo fue crear la jerarquía de memoria con diferentes tipos de memoria que se van balanceando entre capacidad y velocidad. En estas se van copiando a sus hijos pequeñas partes de memoria según se vaya necesitando, siempre manteniendo la inclusión[[5]](#footnote-5) y la coherencia[[6]](#footnote-6).

### **Jerarquía de memoria:**

Registros: Se encuentran en el CPU con un tamaño de byte / KB, velocidad 0,25 ns.

Cache: Se encuentran en la CPU o chip pero con capacidad KB/MB y velocidad 1 ns.

Memoria: Conocido como RAM con una capacidad (MB/GB), velocidad 100ns.

Memoria Virtual: La virtualización del HDD/SDD (GB/TB) pero con una velocidad 5 ms.

### **Clasificación de cache:**

Existen varios niveles de cache, en un principio estaba L1, posteriormente se añadió L2 off-chip que después paso a in-chip y se añadió una L3 off-chip que actualmente también es in-chip.

**Cache multinivel:** A medida que fue avanzando disponibilidad de espacio dentro de los microprocesadores se fue haciendo posible la inserción de nuevo niveles de cache on-chip, lo cual permite acceder a mayor cantidad de datos más rápido lo cual beneficia a la velocidad de sistema. No obstante el añadir más niveles de cache aumenta la complejidad por el tamaño del mismo, el algoritmo de sustitución y las políticas de escritura.

**Caches unificada / separadas:** Tener una cache unificada significa que en la misma se guardan los datos y las instrucciones, esto tiene la ventaja que se va auto nivelando la cantidad de información cada tipo, consiguiendo una mayor efectividad (velocidad).  
 Pero por otro lado la cache separada se divide en datos o instrucciones, esto posee ventajas para sistemas con alto paralelismo, ya que estos requieren muchos accesos a instrucciones generando un uso intensivo del bus.

Para saber que copiar a las memorias más rápidas se utilizan los **principios de localidad de referencia** que cumplen todos los **programas**.  
 **Principio de localidad temporal:** Dice que los elementos (datos o instrucciones) a los que se accedió recientemente son muy probables de volver a ser accedidos en un futuro próximo.

**Principio de localidad Espacial:** Dice que los elementos recientemente ejecutados o próximos a un elemento ejecutado son muy probables de ser ejecutados en la brevedad.

Con estos principios se copia, con un 90% de efectividad, la próxima instrucción a los sub-hijos (Memoria y cache); Y el 10% restante de las veces se tiene que acceder a la memoria padre (más lenta) en busca de la instrucción requerida.

El CPU y Cache se manejan mediante el bus de datos usualmente 64 bits (Windows 64 o 32 etc.), decimos que se pasan **palabras**. Mientras que entre la memoria y la cache se pasan **bloques**, ósea más cantidad de datos dependiendo los principios de localidad. Cuanto más chica y más cerca del procesador se encuentre la unidad de almacenamiento, datos más chicos se les pasa.   
Siempre se les pasa a sus hijos el dato y la dirección de memoria (etiqueta) donde está almacenado el dato. Así mismo para saber de qué forma copiar los datos de memoria a cache se utilizan ciertos métodos de correspondencia.

### **Métodos de correspondencia**

**Correspondencia totalmente asociativa:** El controlador de cache le asignara el lugar que encuentre disponible, es decir que no hay una regla que indique donde se va a guardar. No posee índice.  
 El problema es que para encontrar un bloque se deben recorrer todo la cache.

**Correspondencia Directa:** Cada bloque posee un único lugar en cache, siempre el mismo. Dicha ubicación está determinada por N° Dir. Bloque mod N° líneas cache = Dir. de cache para ese bloque.   
 Si coincide con una dirección previamente escrita, se tapa al anterior.  
 La desventaja es que para programas de repetición se estaría constantemente sobrescribiendo los bloques, resultando ineficiente.

**Correspondencia asociativa por conjunto:** (Asociativa 2/4 vías)Es un punto medio entre las 2, esto se hace teniendo **conjuntos** (que son un grupo de líneas de cache, por lo general cada conjunto son entre 2 y 4 bloques) y para saber a qué conjunto pertenece el bloque se hace N° Dir. Bloque mod N° conjuntos en cache = N° conjunto de dicho bloque.  
 De esta forma solo se deberá buscar entre los 2/4 conjuntos que poseen el n° de bloque buscado y al poseer varias líneas no se estaría sobre escribiendo tanto.

**Tipo Correspondencia:**

El tipo de correspondencia es como están guardados las direcciones de memoria de los datos.  
Con la **ETIQUETA** más el **INDICE** se consigue el bloque.   
El **BO** es la dirección precisa de cada elemento del bloque.  
La suma de todos nos da la dirección en memoria del dato.

Para este Ej. Usamos bloques de 4 datos y 4 líneas, Por lo tanto una capacidad máxima de 16 datos.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Totalmente asociativa** | | | | |  |
| **Etiqueta** | **00** | **01** | **10** | **11** | Se usan **6 bits** para la dirección. |
| **101011** | 1A | FF | 32 | 4C | Dir. **10101100** = 1A |
|  |  |  |  |  | Dir. **10101101** = FF |
|  |  |  |  |  | Dir. **10101110** = 32 |
| **011000** | D5 | 00 | A3 | 02 | Dir. **10101111** = 11 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Directa** | | | | |  |
|  | **Etiqueta** | **00** | **01** | **10** | **11** | Se usan **4 bits** para la dirección. |
| **00** | **0010** | 1A | FF | 32 | 4C | Dir. **00100000** = 1A |
| **01** |  |  |  |  |  | Dir. **00100001** = 1A |
| **10** |  |  |  |  |  | Dir. **00100010** = 1A |
| **11** | **1100** | D5 | 00 | A3 | 02 | Dir. **00100011** = 1A |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Asociativa por conjunto** | | | | |  |
|  | **Etiqueta** | **00** | **01** | **10** | **11** | Se usan **5 bits** para la dirección y tiene **2 conjuntos** por índice. |
| **0** | **00101** | 1A | FF | 32 | 4C | Dir. **00101000** = 1A Dir. **11000000** = D5 |
| **1** |  |  |  |  |  | Dir. **00101001** = FF Dir. **11000001** = 00 |
|  |  |  |  |  |  | Dir. **00101010** = 32 Dir. **11000010** = A3 |
| **0** | **11000** | D5 | 00 | A3 | 02 | Dir. **00101011** = 4C Dir. **11000011** = 02 |
| **1** |  |  |  |  |  |  |

### **Políticas de remplazo**

Para remplazar los bloques y decidir por cual, se crearon los algoritmos de sustitución.  
Dichos algoritmos están ejecutados por el **controlador de memoria cache**

**Correspondencia directa:** A cada bloque le corresponde un único lugar por lo que simplemente lo remplaza.

**Correspondencia asociativa:**

**LRU:** (Last Recently Used)  
 Remplaza al bloque que lleva más tiempo sin ser referenciado**.**

**FIFO:** (First In First Out)  
 El primero que entro es el primero que se va.

**LFU:** (Least Frecuently Used)  
 El que menos se haya usado es el remplazado.

**Aleatorio:** Se sustituye una línea al azar.

### **Política de escritura**

Determina cómo va actuar la CPU cuando tenga que actualizar los valores de un dato.

**Escritura en acierto:**   
Se denomina así cuando el dato a cambiar posee una copia en la cache.

**WRITE-BACK:** Solo se actualiza la cache, pero deja marcado que se realizó un cambio. Cuando se va a sobrescribir ahí se procede a copiar el contenido a la Mem. Principal.  
 Otros procesadores pueden necesitar el dato por lo que deberán obtenerlo de la cache. Requiriendo mas circuitería.

**WRITE-THROUGH:** Se actualiza simultáneamente la cache y la Mem. Principal.  
 Puede generar cuello de botella en caso de tener muchos CPUs.

**Escritura en fallo:**Cuando el dato a cambiar no se encuentra en la cache.

**WRITE ALLOCATE:** EL dato se lleva de la principal a la cache y se sobre escribe la cache.  
 Comúnmente se usa con WRITE-BACK.

**NO-WRITE ALLOCATE:** El bloque no se escribe en la cache, solo se escribe en la Mem. Principal.   
 Se suele usar con WRITE-THROUGH.

# **BUSES DEL SISTEMA**

Los buses son la forma de comunicación que se tiene entre la CPU, la memoria, los dispositivos de E/S y el resto de dispositivos. La señal emitida por uno de estos dispositivos está disponible para que los otros dispositivos conectados al bus puedan acceder a ella, solo un dispositivo puede transferir por cada momento dado.

Los buses están compuestos por caminos / líneas, cada línea transmite binario. Se puede usar múltiples líneas paralelas para transferir datos simultáneamente.

Para lograr el pasaje de información los buses se dividen en 3 partes:

### **Partes del bus:**

Bus de dirección:   
 Suele poseer entre 16, 20, 24 o más líneas. Es el encargado de transferir la dirección de lo que se quiere leer o escribir. Es unidireccional, del CPU a memoria o E/S.  
 Su ancho determina la capacidad máxima de memoria posible del sistema.

Bus de dato:   
 Posee 8, 16, 32 o 64. Es el encargado de transferir la información entre todos los dispositivos. Es bidireccional. Lo puede operar únicamente un dispositivo por vez.

Bus de control:  
 Posee entre 4 a 10 líneas. Es el que indica lo que se desea hacer en la dirección del bus de direcciones. Indica si se desea leer o escribir, en memoria o en E/S, las señales del clock y los pedidos de interrupción.

Dado que hay dispositivos que trabajan más lentos que otros, el tener un solo bus no es una opción viable, esto generaría retardos de propagación. Para esto se generaron distintos buses.

### **Jerarquía de buses:**

**Bus Local:** Es el que conecta la CPU con la cache y los controladores locales de E/S.

**Bus de Sistema:** Conecta la cache con la memoria y la interfaz del bus de expansión.

**Bus de Expansión:** Conecta la interfaz del bus de expansión con las entradas más lentas como la de red, la del modem, teclado, mouse, etc.

### **Tipos de buses:**

**Dedicados:** Usan distintos líneas para movimiento de datos y de direcciones.  
 Ej. 16 de datos, 16 de direcciones y 1 línea de control R/W)

**Multiplexados:** Usan las mismas líneas alternando su función, pero se necesitan más circuitos.  
 Ej. 16 de direcciones y datos, 1 de línea de control R/W y 1 de A/D (Address / Data).

### **Tipos de arbitrajes:**

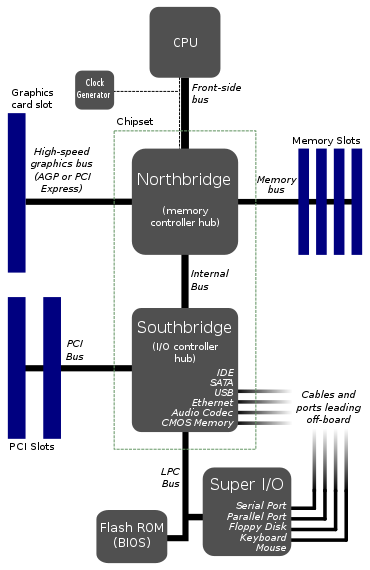
Es la forma en la que se organizan los dispositivos para usar el bus sin pisarse. **Arbitraje centralizado:** Un único dispositivo (Llamado árbitro o controlador de bus) se encarga de sincronizar todo el resto de dispositivos. Este puede ser un módulo aparte o parte del CPU.

**Arbitraje distribuido:** Cada módulo debe tener una lógica para controlar el acceso al bus, por lo que los módulos deben trabajar en conjunto para controlar el bus.

### **Tipos de sistemas**

**Sistemas Sincrónicos:** Todos los dispositivos trabajan al compás del reloj / clock, por lo general en su flanco ascendente. Se requiere una línea extra en el bus, transmitiendo el reloj.

**Sistemas Asincrónicos:** El sistema va trabajando a una velocidad variable ya que se va esperando a la respuesta de cada dispositivo. Es decir que un evento en el bus es consecuencia y depende de que se produzca un evento previo.



**Bus del sistema**

### **Bus PCI:** (Peripheral Component Interconect)

Es un bus sincrónico y con arbitraje centralizado. Se puede utilizar como bus de periférico o como arquitectura de entreplanta. Tiene buenas prestaciones para dispositivos de E/S de alta velocidad (Placa de red, Tarjeta gráfica, controladores de disco, etc.).   
Funciona con muy pocos circuitos y permite que otros buses se conecten a el.   
Existe de 32 y 64 bits.

Introduce Plug and play. Los dispositivos conectados al bus son para el procesador como cualquier otro dispositivo

Para 32 bits utiliza 49 líneas:  
 Terminal de sistema, de control y datos, de control de interfaz, de arbitraje y de error.

Para 64 bits usa las 49 más 51 líneas más.

### **Bus SCSI:** (Small Computer System Interface)

Es un sistema paralelo sincrónico, pero con la capacidad de también ser asincrónico. Posteriormente las versiones más nuevas pasaron a ser en un sistema serie usando el método de conexión point to point.   
Permite conectar múltiples dispositivos mediante su cable, trabajando en formato de Daisy-chain mediante hand-shake. Hay que tener en cuenta que hay inicio y al final de la cadena se debe colocar el controlador de interfaz (este ocupa un lugar de dispositivo).

Se usaba principalmente para conectar dispositivos de memoria, como en sistemas RAID, hasta que el sistema SATA le saco el puesto.

SCSI 1: trabaja a 8 bits, velocidad 5 MBps, longitud máxima del cable 6m y 8 dispositivos.

SCSI ultra 2: trabaja a 16 bits, velocidad 80 MBps, longitud máxima del cable 12m y 15 dispositivos.

Permite únicamente a ciertos dispositivos conectarse

Los dispositivos conectados al SCSI no son vistos por el procesador al igual que los que están conectados directamente al bus del procesador.

El controlador de SCSI se encarga de la trasferencia de sus dispositivos mediante DMA

Daisy chain tiene niveles de prioridad cada dispositivo

Muchas veces se conecta el bus SCSI al bus de PCI.

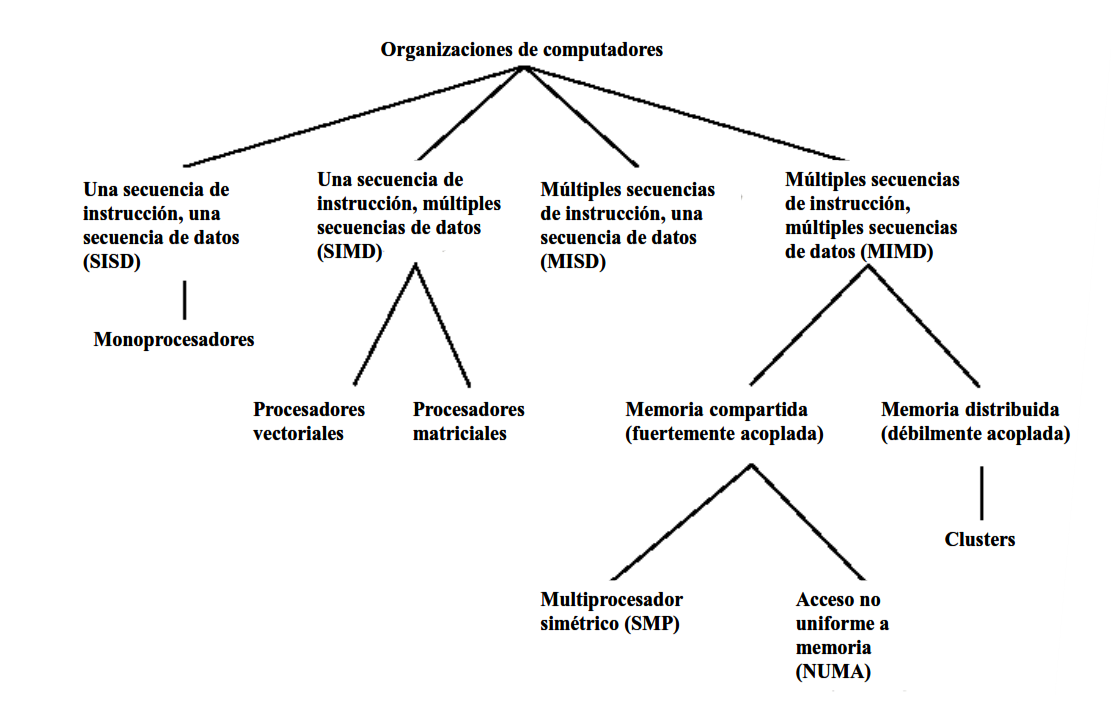
# **PROCESAMIENTO PARALELO**

Existe el **paralelismo a nivel de instrucción (ILP)** el cual mejora el rendimiento de un solo procesador, el ILP puede ejecutarse cuando las instrucciones de una secuencia son independientes y por lo tanto pueden ejecutarse en simultáneo.

El **paralelismo al nivel de la máquina**, es la capacidad del procesador de sacar partido al ILP.  
Esto depende del número de instrucciones que puedan captarse y ejecutarse al mismo tiempo (cant. De cauces paralelos) y de la capacidad para localizar instrucciones independientes.

El **paralelismo a nivel proceso** que es el paralelismo para sistemas con varios procesadores.

### Taxonomía de Flynn



**SISD:** Computadoras mono procesador, con una instrucción de una secuencia en única memoria.

**SIMD:** Cada procesador ejecuta simultáneamente la misma instrucción pero con distintos valores, cada procesador tiene su propia memoria dedicada. Se usa para vectores o matrices.

**MISD:** No se implementó nunca. Mismas instrucciones devuelven una secuencia de datos.

**MIMD:** Múltiples procesadores ejecutan una secuencia de instrucciones diferentes en simultáneo de distintos tipos de datos. Los datos se pueden guardar en memoria compartida SMP y NUMA o en memoria distribuida CLUSTERS.

### **Sistemas de alta demanda**

Son sistemas creados para trabajar con alta demanda de recursos, si bien pueden usarse como una PC normal, su uso común es para el funcionamiento de servers / networks o alto rendimiento.

**Multiprocesador simétrico (SMP):** Es una arquitectura caracterizada por el hardware y el O.S. que lo compone.Sobre hardware: Conecta 2 o más CPUs de capacidades comparables, con la constante posibilidad de ampliación (Rango normal de 16 a 64 CPUs), dichas CPUs poseen sus caches pertinentes.   
 Después todos los procesadores están conectados a un conjunto de caches L2, las memorias principales y dispositivos E/S.   
 Todos estos se pueden interconectar mediante un bus, lo cual posee la desventaja de que se satura y que tiene que trabajar todo a cierta velocidad. O Puede que cada uno tenga su propio canal hacia los mismos dispositivos.  
 Tiempo de acceso a memoria similar para todos los procesadores (UMA).  
 Fácil de administrar y configurar, menor consumo de espacio y potencia que clusters.   
 Pero con un límite de CPUs más que nada por el bus compartido.  
 El fallo de un procesador no detendrá la computadora.

**Clusters:** Múltiples computadoras completas (nodos) que trabajan conjuntamente como un único recurso de cálculo. Están conectadas mediante una LAN / WAN, con posibilidad de tener una memoria compartida (ej: una raid).  
 A cada nodo se le instala un software intermedio (middleware) el cual proporciona al sistema una imagen de sistema único además de ser el responsable de la administración (distribución de carga, respuesta a fallos, entre otros).  
 Dado que el sistema está hecho de componentes redundantes (todas computadoras iguales) su escalabilidad es absoluta e incremental y el fallo en una computadora no significa la pérdida del servicio. Mejor relación precio / prestaciones.

### **Acceso a memoria:**

**UMA:** (Uniform Memory Access) (Acceso Uniforme a Memoria)  
 El tiempo de acceso a memoria desde cualquier procesador es igual.

**NUMA:** (Non-Uniform Memory Access) (Acceso No-Uniforma a Memoria)Cada procesador tiene tiempo de acceso distinto a diferentes partes de memoria.

**CC-NUMA:** (Cache-Coherent NUMA) (NUMA con Coherencia de Cache)  
 Una computadora NUMA donde la coherencia de cache se mantiene en todos los procesadores.

CC-NUMA puede ser utilizada en combinación con SMP, de esta forma se logra sistemas de Multiprocesador Simétrico con la memoria cache compartida y una escalabilidad mayor. Es decir se arman computadoras SMP (con menos CPU que un SMP normal) las cuales se conectan entre ellas mediante el sistema CC-NUMA.

# **SET DE INSTRUCCIÓNES**

## **CISC (Complex Instruction Set Computer)**

**MSX88 Vonsim**

Busca facilitar el trabajo al escritor de compiladores.

Tiende a buscar tener una instrucción máquina para algunas instrucciones de HLL[[7]](#footnote-7). Es decir el compilador posee instrucciones de HLL, esto hace que la compilación ocupe menos tamaño.

Requiere la implementación de micro-programación en la unidad de control, para representar las instrucciones más complejas.

Produce programas más pequeños, lo cual significa que usa menos memoria. Pero NO es más rápido que el mismo programa en RISC ya que la micro-programación es más lenta que una instrucción básica. La memoria hoy en día es barata y sobra.

Pese a tener instrucciones más complejas estas pocas veces son usadas. El compilador suele usar las más sencillas, generando códigos de operación más largos para una NO mejora de velocidad. Para que se implementen estas instrucciones se debería complejizar la Unidad de Control y/o se debería aumentar la memoria de control, esto ralentizaría las instrucciones,

El uso de instrucciones complejas requiere la restricción de interrupciones en ciertos momentos.

## **RISC (Reduced Instruction Set Computer)**

**WIN-MIPS 64**

Repertorio de instrucciones limitado y fijo.

Operaciones de registro a registro, con una gran cantidad de los mismos.

Modos de direccionamiento y formatos de instrucción sencillos.

Énfasis en la optimización del cauce.

Los formatos de instrucciones son R, I y Control / J. Poseen una longitud fija (32 bits). Esto permite tener poco o no tener micro-código en la UC logrando tener una mayor velocidad

Memoria de datos y registros separados (32 registros uso general, 16/32 de P.F.)

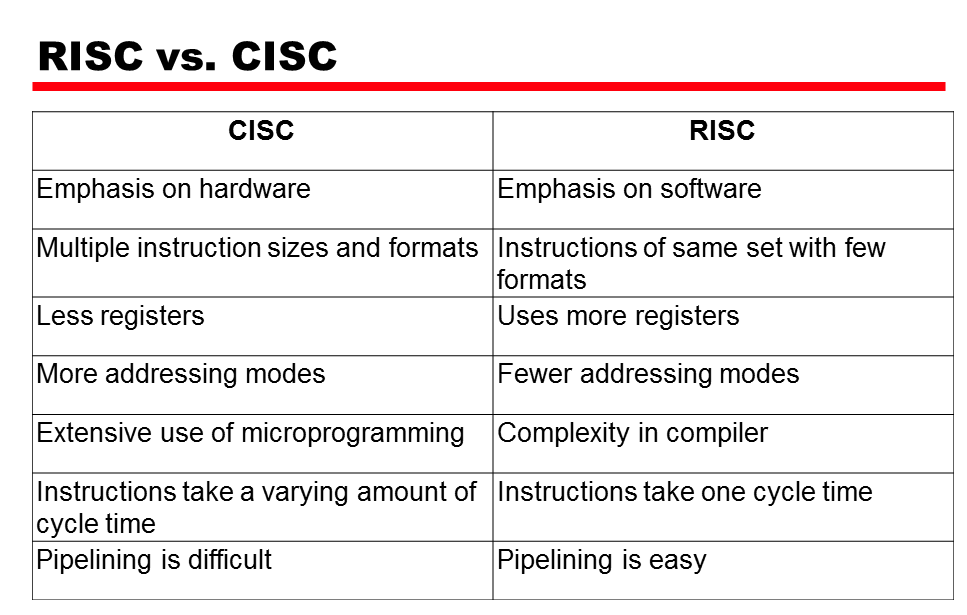
Posee menos modos de direccionamiento que el CISC. Mayoría de operaciones son registro a registro. No posee operaciones que combinen carga/almacenamiento con cálculos aritméticos.

Es decir, el tener instrucciones más simples permite una mejor reordenación del código, sacar instrucciones de bucle, etc.

Así mismo el tener operaciones simples se utilizan unidades de control cableadas, las cual son más rápidas que las micro-programadas

Tener instrucciones más sencillas permite una mayor efectividad con segmentación de cauce.

RISC trabaja mejor con tecnologías Súper-Escalares.

**COMPARACION:**

No existen maquinas RISC y CISC que sean directamente comparables.

Se llegó a la conclusión que ambos tienen partes buenas y malas. Por lo que los sistemas RISC adoptaron parte de lo CISC y viceversa.

# **SISTEMA VON-NEUMANN**

El modelo Von-Neumann de 5 sistemas fue el primero en trabajar en binario, poseía una unidad de control, una memoria compartida (de datos e instrucciones), una unidad de cálculo, unidad de entrada y unidad de salida.

Después se encontraba la de 3 sistemas con una unidad de E/S, una CPU (con la unidad de cálculo, control y registros dentro) y una memoria.

### **Sistema Harvard:**

Por el otro lado se encuentra el sistema Harvard. Si bien las diferencias entre estos dos sistemas no están del todo explicitas, lo que si diferencia a los dos sistemas es que el sistema Harvard posee la memoria de instrucciones en modo solo-lectura. En un principio poseía la memoria de instrucciones y datos separadas, pero con el avance de la tecnología dejo de ser un problema tener la memoria unificada por lo que hubo ciertos procesadores Harvard que unificaron la memoria nuevamente.

1. Más info: Organización y Arquitectura de Computadoras, William Stallings, Capítulo 18, Inciso 4, Página 686 de la 7ma edición. [↑](#footnote-ref-1)
2. ILP: Instruction Level Paralelism, Paralelismo a Nivel de Instrucciones. [↑](#footnote-ref-2)
3. 1°i y 2°i: Hace referencia a una primera instrucción (1°i) y a una segunda instrucción (2°i) refiriéndose a su orden en el cauce. [↑](#footnote-ref-3)
4. Fowarding: Es una técnica simple que consiste en el adelantamiento del dato a la próxima instrucción, consiste en pasarle el dato a la próxima instrucción apenas se obtiene el resultado de la salida de la ALU. [↑](#footnote-ref-4)
5. Inclusión: Los datos almacenados en un nivel han de estar almacenados en los niveles superiores a él. [↑](#footnote-ref-5)
6. Coherencia: Las copias de la misma información almacenada en los distintos niveles deben tener los mismos valores. [↑](#footnote-ref-6)
7. HLL (High Level Language): Lenguaje de Alto Nivel. [↑](#footnote-ref-7)