MODULO ARQUITECTURA DE COMPUTADORES

JESÚS EMIRO VEGA
Jesus.vega@unad.edu.co

UNIVERSIDAD NACIONAL ABIERTA Y A DISTANCIA UNAD
ESCUELA DE CIENCIAS BÁSICAS TECNOLOGIA E INGENIERÍA
PROGRAMA INGENIERIA DE SISTEMAS
2008

Universidad Nacional Abierta y a Distancia
ISBN
2008

TABLA DE CONTENIDO

INTRODUCCION.
PRIMERA UNIDAD. Visión general.
INTRODUCCION.
Capitulo 1. Organización y Arquitectura de computadores
1.1. Organización y Arquitectura de computadores
1.2. Estructura y funcionamiento.
1.2.1. Funcionamiento.
1.2.2. Estructura.
Autoevaluación.
Capitulo 2. Evolución de los computadores.
2.1. Breve historia de los computadores.
2.1.1. La era mecánica.
2.1.2. Las Primeras generaciones (Electromecánicas y electrónicas de tubos de vacio).
2.1.3. La segunda generación (los transistores y los avances en programación).
2.1.4. Tercera generación (circuitos integrados y miniaturización).
2.1.5. Cuarta generación (ordenadores personales de uso domestico).
2.1.6. Historia y desarrollo en hardware y software
2.2. Arquitectura Harvard
2.3. Evolución de los microprocesadores
2.3.1. Microcomputadores.
2.3.2 Minicomputadores.
2.2.3. Computador portátil.
2.3.4 Laptops.
2.3.5. Computador de bolsillo.
2.3.6 Computadores de red (NC).
2.3.7 Otras Computadoras.
2.4 Modelos de PC.
Autoevaluación.
Capitulo 3. El papel del rendimiento.
3.1. Medidas del rendimiento.
3.2. Métricas de rendimiento.
Autoevaluación.
Capitulo 4. El computador.
4.1. Buses del sistema.
4.2. Memoria.
4.2.1 Latencia, rendimiento y ancho de banda
4.2.2 Segmentación, paralelismo y precarga.
4.2.3 Jerarquía de memoria.
4.2.4 Tecnología de memoria.
4.2.5 Organización del chip de memoria.
4.2.6 Memoria SRAM.
4.2.7 Memoria DRAM.
4.2.8 Refresco de memoria DRAM.
4.2.9 Temporización de una DRAM.
4.2.10 El modo pagina y la memorias DRAM actuales.
4.3. Entrada/Salida (E/S).
4.3.1 Funciones de la interface E/S
4.3.2 Controlador de dispositivos de E/S
4.4. El sistema Operativo.

Autoevaluación.
SEGUNDA UNIDAD 2. Unidad Central de Procesamiento.
INTRODUCCION.
Capitulo 1. Aritmética del computador y representación interna de los datos
1.1 Sistemas de numeración.
1.1.1 Representación de los números.
1.1.2 Números binarios
1.1.3 Números naturales
1.1.4 Números enteros.
1.1.5 Números racionales
1.1.6 Aritmética binaria.
1.1.6.1 Adición
1.1.6.2 Sustracción
1.1.6.3 Multiplicación.
1.1.6.4 División.
1.1.7 Desbordamiento y agotamiento.
1.1.8 Código decimal binario.
1.1.9 Código octal
1.1.10 Código hexadecimal
1.1.11 Códigos digitales.
1.1.12 Código Gray
1.1.8 Códigos alfanuméricos
1.2 ALU Unidad Aritmética Lógica
1.3 Números en coma flotante.
1.3.1 NAN Y Números desnormalizados
1.3.2 Aritmética con números en coma flotante
Autoevaluación.
Capítulo 2: Estructura y Funcionamiento de la CPU
2.1. Organización del procesador
2.1.1 Las maquinas síncronas.
2.1.2 Caminos de datos.
2.1.3 Estructura básica del procesador.
2.1.4 ALU
2.2 Organización de los registros
2.3. El ciclo de instrucción
2.4 Manejo de interrupciones
2.4.1 Interrupciones múltiples.
2.4.2. El accesos directo a memoria.
2.4.3 El puerto paralelo.
2.5 Procesador Pentium.
2.6. Procesador Power Pc.
Autoevaluación
Capitulo 3. Arquitecturas.
Autoevaluación.
TERCERA UNIDAD. Repertorio de instrucciones.
INTRODUCCION.
Capitulo 1. Lenguaje de máquina
1.1. Visión del programador
1.2 Formato de las instrucciones
1.3 Modos de direccionamiento

1.4 Instrucciones típicas.
Autoevaluación
Capitulo2. Paralelismo
2.1. Descripción
2.2. Limitaciones del paralelismo entre instrucciones
2.3. Procesadores superescalares
2.4. Ejecución fuera de orden frente a ejecución en orden
2.5 Estimación del tiempo de ejecución para procesadores con ejecución en orden
2.6 Estimación del tiempo de ejecución para procesadores con ejecución fuera orden
2.7. Procesadores VLIW
Autoevaluación.
Capitulo 3. Sistemas Multiprocesador
3.1. Características
3.2. Sistemas de paso de mensajes
3.3. Sistemas de memoria compartida
3.4. Comparación entre S de paso de mensaje y S de memoria compartida
Autoevaluación.

INTRODUCCION

En el año de 1950s, John Von Neuman propuso el concepto de una computadora que almacenara programas una arquitectura que se ha convertido en la base para la fundación de la mayoría de los procesadores comerciales usados hoy en día. En la maquina de Von Neuman, los programas y los datos ocupan la misma memoria. La máquina tiene un contador de programas (PC) qué apunta la instrucción actual en la memoria. El PC se pone al día en cada instrucción; cuando no hay ninguna rama, se captan las instrucciones del programa desde las ubicaciones de memoria secuenciales. Salvo un grupo de máquinas de investigación y una colección muy pequeña de dispositivos comerciales, todos los procesadores de hoy trabajan en este simple principio.

Teniendo en cuenta lo anterior la mayoría de los computadores se pueden dividir en tres subsistemas: el procesador, la memoria y el subsistema de entrada y salida (E/S). El procesador es el responsable de ejecutar los programas, la memoria proporciona espacio de almacenamiento para los programas y los datos a los que ellos hacen referencia y el subsistema de (E/S) permite al procesador y a la memoria controlar los dispositivos que interaccionan con el mundo exterior o almacenan datos, como el CD-ROM, el disco duro y la tarjeta de video/monitor.

Una computadora digital es una máquina que puede resolver problemas ejecutando las instrucciones que recibe de las personas; la secuencia de instrucciones que describe cómo realizar cierta tarea se llama programa. Los circuitos electrónicos de una computadora pueden reconocer y ejecutar directamente un conjunto limitado de instrucciones sencillas y todos los programas tienen que convertirse en una serie de esas instrucciones para que la computadora pueda ejecutarlos; juntas, las instrucciones primitivas de una computadora constituyen un lenguaje que permite a las personas comunicarse con la computadora; dicho lenguaje se llama lenguaje de máquina. Las personas que diseñan una computadora nueva deben decidir qué instrucciones incluirán en su lenguaje de máquina, se procura hacer las instrucciones primitivas lo más simples posible, acorde con el uso que se piensa dar a la computadora y sus requisitos de desempeño, con el fin de reducir la complejidad y el costo de los circuitos requeridos.

La arquitectura de computadores se define como la apariencia funcional que presenta a sus usuarios inmediatos, es decir los atributos y características de un sistema visible al programador; es un modelo y una descripción funcional de los requerimientos y las implementaciones de diseño para varias partes de una computadora, con especial interés en la forma en que la unidad central de proceso (CPU) trabaja internamente y accede a las direcciones de memoria. También suele definirse a la arquitectura de computadores como la forma de seleccionar e interconectar componentes de hardware para crear computadoras según los requerimientos de funcionalidad, rendimiento y costo.

Casi todos los computadores actuales son computadores de programa almacenado que representan los programas como códigos que se almacenan como datos en el mismo espacio de direcciones que estos. El concepto de programa almacenado (representando las instrucciones como códigos almacenados de memoria) fue uno de los mayores avances en los inicios de la arquitectura de computadores. Antes de dicho avance, muchos computadores eran por medio de computadores o reconexionando tarjetas de

circuito para definir el nuevo programa algo que requería una gran cantidad de tiempo y era muy propenso a errores.

Uno de los propósitos que se pretende al desarrollar este material didáctico es la de facilitar al estudiante su aprendizaje, comenzando desde una visión general de la arquitectura de computadores hasta el uso instrucciones de maquina.

Las unidades didácticas que se presentarán en este modulo son dos (3), Visión general, Unidad central de procesamiento y repertorio de instrucciones. En la primera unidad se pretende incursionar al estudiante en el aprendizaje de los conceptos básicos de la arquitectura de computadores dando una visión general de la misma, además de la aplicación práctica. En la segunda unidad se pretende que el estudiante desarrolle la habilidad en el uso de los sistemas de numeración herramientas fundamentales para el trabajo cotidiano del ingeniero de sistemas. La tercera unidad tiene como fundamento principal adentrar el estudiante en el manejo de instrucciones básicas para la programación de computadores.

PRIMERA UNIDAD
“Visión General”

Organización y Arquitectura de computadores
Evolución de los computadores.
El papel del rendimiento.
El computador.

INTRODUCCION

En estos capítulos se presenta la estructura, organización y arquitectura de los computadores, ofreciendo un análisis de los fundamentos y conceptos sobre estos componentes.

Se puede definir la arquitectura de computadores como el estudio de la estructura, funcionamiento y diseño de computadores. Esto incluye, sobre todo aspectos de hardware, pero también afecta a cuestiones de software de bajo nivel.

El Computador es un dispositivo electrónico capaz de recibir un conjunto de instrucciones y ejecutarlas, realizando cálculos sobre los datos numéricos, o bien compilando y correlacionando otros tipos de información.

Dada la naturaleza del tema de arquitectura de computadores (lógica digital y sistemas digitales) que orientan estos capítulos tanto en la realización como en la orientación se expone un resumen sobre los conceptos y principios básicos generales de la lógica digital y sistemas digitales a modo de repaso de anteriores asignaturas que contemplaron los temas aquí expuestos; por eso, se hace necesario practicar una coevaluación en la presentación del tema.

También explica como medir, informar y resumir el rendimiento, y describe los principales factores que determinan el rendimiento de un computador. Una razón importante para examinar el rendimiento es que el hardware, con frecuencia, es clave para la efectividad de un sistema completo hardware y software. Determinar el rendimiento de un sistema puede ser bastante difícil. La estructura y lo intrincados de los modernos sistemas software, junto con el amplio rango de técnicas que mejoran el rendimiento empleadas por los diseñadores hardware, han hecho mucho mas difícil la determinación del rendimiento. Sencillamente, es imposible sentarse con un manual del repertorio de instrucciones y un sistema software significativo y determinar la rapidez a la que se ejecutara el software en la maquina. En efecto, para diferentes tipos de aplicaciones pueden ser apropiadas diferentes métricas de rendimiento, y diferentes aspectos de un sistema computador pueden ser mas significativos para la determinación del rendimiento global.

Por supuesto, a la hora de realizar una selección entre diferentes computadores, el rendimiento es casi siempre un atributo importante. Medir y comparar con precisión diferentes maquinas es critico para los compradores y, por tanto, para los diseñadores.

Los vendedores de computadores también saben esto. Con frecuencia, a los vendedores les gustaría ver su maquina con las mejores prestaciones posibles, independientemente de que estas prestaciones reflejen las necesidades de la aplicación del comprador. En algunos casos se han hecho reclamaciones sobre computadores que no reúnen condiciones adecuadas para cualquier aplicación real. Por consiguiente, comprender la mejor manera de medir el rendimiento y las limitaciones de las medidas del rendimiento es importante la selección de una maquina.

Sin embargo, nuestro interés por el rendimiento va mas allá de las posibilidades de determinar el rendimiento solamente desde el exterior de una maquina. Para comprender por que una parte del software hace lo que hace, por que un repertorio de instrucciones puede implementarse para que funcione mejor que otro, o como algunas características

del hardware afectan al rendimiento, necesitamos comprender que determina el rendimiento de una maquina. Por ejemplo, para mejorar el rendimiento de un sistema software, puede ser necesario comprender que factores del hardware contribuyen al rendimiento global del sistema y la importancia relativa de estos factores. Estos factores pueden incluir lo bien que el programa utiliza las instrucciones de la maquina, lo bien que implementa las instrucciones el hardware de base y la forma en que funcionan los sistemas de memoria y de E/S. Comprender como determina el impacto de estos factores en el rendimiento es crucial para comprender las motivaciones que subyacen en el diseño de aspectos particulares de la maquina.

Dependiendo de su factor de forma, probablemente incluirá algunos componentes y características instalados previamente. Estos componentes y características, por lo general son piezas opcionales que hacen que una caja genérica se ajuste a un factor de forma en particular y a los requisitos particulares. Varios de los factores de la forma son cercanos en tamaño y ubicación de componentes, los fabricantes hacen las cajas que se puedan utilizar en un gran número de factores de acuerdo a la forma de los componentes.

Capítulo 1: Organización y Arquitectura de computadores

1.1 Organización y Arquitectura de computadores

Es importante distinguir entre arquitectura y organización del computador. La arquitectura de computadores se refiere a los atributos de un sistema que son visibles para un programador. También se refiere a los atributos que tienen un impacto directo en la ejecución lógica de un programa. La organización de computadores se refiere a las unidades funcionales y sus interconexiones, que dan lugar a especificaciones arquitectónicas.

1.2 Estructura y funcionamiento

Los computadores actuales tienen millones de componentes electrónicos lo cual los convierte en sistemas complejos. Para describirlos claramente veremos la naturaleza jerárquica de estos sistemas. Un sistema jerárquico es un conjunto de subsistemas interrelacionados, donde cada uno se organiza en una estructura jerárquica hasta alcanzar el nivel más bajo del subsistema elemental. Esta naturaleza jerárquica es importante para describir y diseñar estos sistemas. En cada nivel, el sistema consta de un conjunto de componentes y sus interrelaciones. El comportamiento en cada nivel depende sólo de una caracterización abstracta y simplificada del sistema que hay en el siguiente nivel más bajo. El diseñador se centra en la estructura y funcionamiento de cada nivel.

Estructura: Se refiere al modo en que los componentes están interrelacionados.
Funcionamiento: es la operación de cada componente individual como parte de la estructura.

Seguiremos el esquema Top down, arriba abajo por ser una metodología más clara y efectiva. El computador será descrito siguiendo este esquema

1.2.1 Funcionamiento.

La estructura y funcionamiento de un computador son, en esencia, sencillos. Existen cuatro funciones básicas en un computador:

- Procesamiento de datos: El computador debe ser capaz de procesar los datos, los cuales pueden tener una gran variedad de formas.

- Almacenamiento de datos: El computador debe permitir guardar temporalmente, los datos con los que se está trabajando en un momento dado. Se tiene pues una función de almacenamiento de datos por lo menos en el corto plazo. Igualmente se debe de tener una función de almacenamiento la posibilidad de almacenamiento de datos en el largo plazo, esto se posibilita a través del manejo de archivos de datos que se pueden recuperar y actualizar en un futuro.

- Transferencia de datos: El computador como sistema abierto debe ser capaz de transferir datos entre él mismo y el mundo exterior. Su entorno de operación se compone de dispositivos que sirven como fuente o destino de datos. Cuando se reciben o se llevan datos a un dispositivo que está directamente conectado con el computador, el proceso se conoce como entrada-salida (E/S), y este dispositivo recibe el nombre de periférico. El

proceso de transferir datos a larga distancias, desde o hacia un dispositivo remoto, recibe el nombre de comunicación de datos.

- Control: Debe haber un control de estas tres funciones que es ejercido por los entes que proporcionan instrucciones al computador. Dentro del computador una unidad de control gestiona los recursos del computador y dirige las prestaciones de sus partes funcionales en respuesta a estas instrucciones.

1.2.2 Estructura

El computador es una entidad que interactúa con su entorno externo. Todas sus conexiones con este entorno pueden clasificarse como dispositivos periféricos o líneas de comunicación.

Existen cuatro componente estructurales principales:

- CPU, Central Processing Unit, Procesador: Unidad Central de Procesamiento: controla el funcionamiento del computador y realiza funciones de procesamiento de datos.
- Memoria Principal: permite almacenar los datos.
- E/S Entrada Salida: transfiere datos entre el computador y el entorno externo.
- Sistema de interconexión: es un mecanismo que proporciona la comunicación entre la CPU, la memoria principal y la E/S.

El componente más interesante y complejo es la CPU. Sus componentes estructurales son:

- Unidad de Control: controla el funcionamiento de la CPU y del computador.
- Unidad Aritmético-Lógica: realiza las funciones de procesamiento de datos del computador
- Registros: proporcionan almacenamiento interno a la CPU
- Interconexiones CPU: son mecanismos que proporcionan comunicación entre la unidad de control, la ALU y los registros.

Autoevaluación

1. Usando un buscador en internet visita sitios relacionados con la arquitectura y organización de computadores; realizar un ensayo de alguno de estos sitios y realizar un foro en la tutoría

Capítulo 2: Evolución de los computadores

2.1. BREVE HISTORIA DE LOS COMPUTADORES

La Computación, y por tanto, las Ciencias de la Computación, tienen su origen en el cálculo, es decir, en la preocupación del ser humano por encontrar maneras de realizar operaciones matemáticas de forma cada vez más rápida y más fácilmente. Pronto se vio que con ayuda de aparatos y máquinas las operaciones podían realizarse de forma más rápida y automática.

El primer ejemplo que encontramos en la historia es el ábaco, aparecido hacia el 500 AC en Oriente Próximo, que servía para agilizar las operaciones aritméticas básicas, y que se extendió a China y Japón, siendo descubierto mucho más tarde por Europa.

También es digno de señalar el conocido Mecanismo de Antikythera, recuperado en 1900, construido alrededor del año 80 a.C., en la isla griega de Rodas, ubicada en el mar Egeo. Era un artefacto de cálculo astronómico con mecanismos de precisión. El usuario, por medio de una perilla, podía accionar un simulador en miniatura del movimiento del sol, la luna y varios planetas, teniendo a la vista la fecha en que se había dado, o se daría, tal combinación. Es tanta su sofisticación que ha sido llamado la primera computadora de Occidente.

Por otra parte, los matemáticos hindúes, árabes y europeos fueron los primeros que desarrollaron técnicas de cálculo escrito. El matemático árabe Al'Khwarizmi, alrededor del año 830 DC, escribe un libro de Aritmética, traducido al latín como *Algoritmi de numero Indorum*, donde introduce el sistema numérico indio (sólo conocido por los árabes unos 50 años antes) y los métodos para calcular con él. De esta versión latina proviene la palabra algoritmo.

2.1.1 La Era Mecánica

A finales del siglo XVI y comienzos del XVII comienza lo que denominamos Era Mecánica, en la que se intenta que aparatos mecánicos realicen operaciones matemáticas de forma prácticamente automática. En 1610, John Napier (1550-1617), inventor de los logaritmos, desarrolló las Varillas de Napier, que servían para simplificar la multiplicación. En 1641, el matemático y filósofo francés Blaise Pascal (1623-1662), con tan sólo 19 años, construyó una máquina mecánica para realizar adiciones, la Pascalina, para ayudar a su padre. Por su parte, Gottfried Wilhelm Leibniz (1646-1716) propuso el sistema binario para realizar los cálculos, construyendo una máquina que podía multiplicar, en incluso teóricamente, realizar las cuatro operaciones aritméticas. Sin embargo, la tecnología disponible le imposibilita la realización de las operaciones con exactitud. No obstante un estudiante alemán de la Universidad de Tubingen, Wilhelm Schickard (1592-1635) ya había construido una máquina de estas características entre 1623 y 1624, de la que hace unas breves descripciones en dos cartas dirigidas a Johannes Kepler. Por desgracia, al menos una de las máquinas quedó destruida en un incendio, y el propio Schickard murió poco después, víctima de la peste bubónica.

Los trabajos de Pascal y Leibniz tuvieron su continuación en 1727, cuando Jacob Leupold propuso algunas mejoras sobre el mecanismo de Leibniz. En 1777, Charles Mahon (1753-

1816), Conde de Stanhope, construyó una máquina aritmética y otra lógica, esta última llamada Demostrador de Stanhope. En 1825, el francés Charles Xavier Thomas de Colmar diseña una máquina calculadora que posteriormente consigue comercializar con éxito.

Una mención muy especial requiere el desarrollo de un telar automático por el francés Joseph Jacquard (1752-1834), en 1801. En efecto, analizando las operaciones repetitivas que requería la producción de telas, este inventor imaginó conservar la información repetitiva necesaria bajo la forma de perforaciones en tarjetas. Estas perforaciones eran detectadas mecánicamente, asegurando el desplazamiento adecuado de las guías del hilado, pudiendo una sola persona tejer complicados patrones codificados en las perforaciones de las tarjetas.

Fue Charles Babbage (1791-18171) el que diseñó una verdadera máquina procesadora de información, capaz de autocontrolar su funcionamiento. Desesperado por los errores contenidos en las tablas numéricas de la época y dándose cuenta de que la mayoría de los cálculos consistían en tediosas operaciones repetitivas, este profesor de la Universidad de Cambridge, proyecta e inicia la construcción de un nuevo tipo de calculadora. En 1821 presentó a la Royal Society una máquina capaz de resolver ecuaciones polinómicas mediante el cálculo de diferencias sucesivas entre conjuntos de números, llamada Máquina Diferencial. Obtuvo por ello la medalla de oro de la Sociedad en 1822.

Más tarde, Babbage empezó a trabajar en la Máquina Analítica, en cuya concepción colaboró directamente Ada Augusta Byron, Condesa de Lovelace, hija de Lord Byron. El objetivo perseguido era obtener una máquina calculadora de propósito general, controlada por una secuencia de instrucciones, con una unidad de proceso, una memoria central, facilidades de entrada y salida de datos, y posibilidades de control paso a paso, es decir, lo que hoy conocemos como programa. Ada Lovelace, a quien se reconoce como la primera programadora de la historia, y en honor de quien se puso el nombre de Ada al conocido lenguaje de programación, ayudó a Babbage económicamente, vendiendo todas sus joyas, y escribió artículos y programas para la referida máquina, algunos de ellos sobre juegos. Sin embargo, este proyecto tampoco pudo realizarse por razones económicas y tecnológicas.

En el 1854, George Boole publica *Las leyes del pensamiento* sobre las cuales son basadas las teorías matemáticas de Lógica y Probabilidad. Boole aproximó la lógica en una nueva dirección reduciéndola a una álgebra simple, incorporando lógica en las matemáticas. Comenzaba el álgebra de la lógica llamada Algebra Booleana. Su álgebra consiste en un método para resolver problemas de lógica que recurre solamente a los valores binarios 1 y 0 y a tres operadores: AND (y), OR (o) y NOT (no).

2.1.2 La Primera Generación (electromecánicos y electrónicos de tubos de vacío)

Para tabular el censo de 1890, el gobierno de Estados Unidos estimó que se invertirían alrededor de diez años. Un poco antes, Herman Hollerith (1860-1929), había desarrollado un sistema de tarjetas perforadas eléctrico y basado en la lógica de Boole, aplicándolo a una máquina tabuladora de su invención. La máquina de Hollerith se usó para tabular el censo de aquel año, durando el proceso total no más de dos años y medio. Así, en 1896,

Hollerith crea la *Tabulating Machine Company* con la que pretendía comercializar su máquina. La fusión de esta empresa con otras dos, dio lugar, en 1924, a la *International Business Machines Corporation* (IBM).

Sin embargo, en el censo de 1910, el sistema de Hollerith fue sustituido por uno desarrollado por James Powers. En 1911 James Powers constituyó la *Power's Tabulating Machine Company*, convirtiéndose en el principal competidor de Hollerith.

En 1900, en el Congreso Internacional de Matemáticas de París, David Hilbert (1862-1943) pronunció una conferencia de título Problemas matemáticos, en la que proponía una lista de 23 problemas que estaban sin resolver (algunos todavía lo están).

Dos de estas cuestiones fueron: ¿es la matemática completa?, es decir, ¿puede ser demostrada o refutada cualquier sentencia matemática? y ¿es la matemática consistente?, es decir, ¿es cierto que sentencias tales como 0 = 1 no pueden demostrarse por métodos válidos?. En 1931, Kurt Gödel (1906-1978) fue capaz de responder a estas dos preguntas, demostrando que cualquier sistema formal suficientemente potente es inconsistente o incompleto.

Otra de las cuestiones era: ¿son las matemáticas decidibles? es decir, ¿hay un método definido que pueda aplicarse a cualquier sentencia matemática y que nos diga si esa sentencia es cierta o no?. Esta cuestión recibió el nombre de *enstcheidungsproblem*.

En 1936, Alan Turing (1912-1954) contestó a esta cuestión en el artículo *On Computable Numbers*. Para resolver la cuestión Turing construyó un modelo formal de computador, la Máquina de Turing, y demostró que había problemas tales que una máquina no podía resolver. Al mismo tiempo en Estados Unidos contestaba a la misma cuestión Alonzo Chuch, basándose en una notación formal, que denominó *cálculo lambda*, para transformar todas las fórmulas matemáticas a una forma estándar. Basándose en estos resultados, entre 1936 y 1941, el ingeniero alemán Konrad Zuse (1910-1957), diseñó y construyó su serie de computadores electromecánicos binarios, desde el Z1 hasta el Z3. Sin embargo estos computadores no tuvieron mucha difusión, ni siquiera dentro de su país, ya que el gobierno nazi nunca confió en los trabajos de Zuse.

En 1938, Claude Shannon (1916-) demostró cómo las operaciones booleanas elementales, se podían representar mediante circuitos conmutadores eléctricos, y cómo la combinación de circuitos podía representar operaciones aritméticas y lógicas complejas. Además demostró como el álgebra de Boole se podía utilizar para simplificar circuitos conmutadores. El enlace entre lógica y electrónica estaba establecido.

Al desencadenarse la Segunda Guerra Mundial, la necesidad de realizar complicados cálculos balísticos y la exigencia de descodificar los mensajes cifrados del otro bando, impulsó el desarrollo de los computadores electrónicos de propósito general. El propio Turing fue reclutado en *Bletchley Park*, en Inglaterra, para descifrar los mensajes que encriptaba la máquina alemana Enigma, para lo que fue necesario construir la computadora Colossus.

En la Universidad de Harvard, Howard Aiken (1900-1973) en colaboración con IBM, empezó, en 1939, la construcción del computador electromecánico Mark I, en la que

trabajó como programadora Grace Murray Hopper. Pero para cuando se terminó en 1944, ya habían aparecido las primeras computadoras totalmente electrónicas, que eran mucho más rápidas.

Por otro lado, en la Universidad del Estado de Iowa, entre 1937 y 1942, John Vincent Atanasoff (1903-1995) y Clifford Berry, diseñaron y construyeron la ABC (Atanasoff-Berry Computer). Terminada en 1942, fue la primera computadora electrónica digital, aunque sin buenos resultados y nunca fue mejorada. En 1941, John W. Mauchly (1907-1980) visitó a Atanasoff y observó de cerca su impresionante maquinaria, teniendo la oportunidad de revisar su tecnología. Más tarde, Mauchly y J. Presper Eckert, Jr (1919-1995), diseñaron y construyeron, entre los años 1943 y 1946, el computador eléctrico de propósito general ENIAC. Existe una gran controversia respecto a que Mauchly copiara muchas de las ideas y conceptos del profesor Atanasoff, para construir la computadora ENIAC. En cualquier caso en las últimas fases de su diseño y construcción aparece la importante figura de John Von Neumann (1903-1957), que actúa como consultor.

Von Neumann escribió en 1946, en colaboración con Arthur W. Burks y Herman H. Goldstine, *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*, que contiene la idea de Máquina de Von Neumann, que es la descripción de la arquitectura que, desde 1946, se aplica a todos los computadores que se han construido.

Con estos fundamentos, Eckert y Mauchly construyen en la Universidad de Manchester, en Connecticut (EE.UU.), en 1949 el primer equipo con capacidad de almacenamiento de memoria, la EDVAC. Eckert y Mauchly forman una corporación para construir una máquina que se pueda comercializar, pero, debido a problemas financieros, se vieron obligados a vender su compañía a a Remington Rand Corp. Trabajando para esta compañía fue que se concluyó el proyecto Univac, en 1951.

También por esta época Maurice Wilkes construye la EDSAC en Cambridge (Inglaterra) y F.C. Williams construye en Manchester (Inglaterra), la Manchester Mark I.

Estas máquinas se programaban directamente en lenguaje máquina, pero a partir de mediados de los 50, se produjo un gran avance en la programación avanzada.

2.1.3 La Segunda Generación (los transistores y los avances en programación)

Allá por 1945 la máxima limitación de las computadoras era la lenta velocidad de procesamiento de los relés electromecánicos y la pobre disipación de calor de los amplificadores basados en tubos de vacío.

En 1947, John Bardeen, Walter Brattain y William Shockley inventan el transistor, recibiendo el Premio Nobel de Física en 1956. Un transistor contiene un material semiconductor, normalmente silicio, que puede cambiar su estado eléctrico. En su estado normal el semiconductor no es conductivo, pero cuando se le aplica un determinado voltaje se convierte en conductivo y la corriente eléctrica fluye a través de éste, funcionando como un interruptor electrónico.

Los computadores construidos con transistores eran más rápidos, más pequeños y producían menos calor, dando también oportunidad a que, más tarde, se desarrollaran los microprocesadores. Algunas de las máquinas que se construyeron en esta época fueron la TRADIC, de los Laboratorios Bell (donde se inventó el transistor), en 1954, la TX-0 del laboratorio LINCOLN del MIT y las IBM 704, 709 y 7094. También aparece en esta generación el concepto de supercomputador, específicamente diseñados para el cálculo en aplicaciones científicas y mucho más potentes que los de su misma generación, como el *Livermore Atomic Research Computer* (LARC) y la IBM 7030.

Pero esta generación se explica también por los avances teóricos que se dan.

Así, en 1950, Alan Turing publica el artículo *Computing Machinery and Intelligence* en la revista *Mind*, en el que introducía el célebre Test de Turing. Este artículo estimuló a los pensadores sobre la filosofía e investigación en el campo de la Inteligencia Artificial. Por desgracia, Turing no fue testigo del interés que desató su artículo, porque en 1952 fue detenido por su relación homosexual con Arnold Murray y fue obligado a mantener un tratamiento con estrógenos que le hizo impotente y le produjo el crecimiento de pechos. En 1957, fue encontrado muerto en su casa al lado de una manzana mordida a la que había inyectado cianuro.

En 1951, Grace Murray Hooper (1906-1992) da la primera noción de compilador y más tarde desarrolla el COBOL. Pero fue John Backus, en 1957, el que desarrolla el primer compilador para FORTRAN. En 1958, John MacCarthy propone el LISP, un lenguaje orientado a la realización de aplicaciones en el ámbito de la Inteligencia Artificial. Casi de forma paralela, Alan Perlis, John Backus y Peter Naur desarrollan el lenguaje ALGOL.

Pero el personaje más importante en el avance del campo de los algoritmos y su análisis, es Edsger Dijkstra (1930-), que en 1956, propuso su conocido algoritmo para la determinación de los caminos mínimos en un grafo, y más adelante, el algoritmo del árbol generador minimal. Más tarde, en 1961, N. Brujin introduce la notación O, que sería sistematizada y generalizada por D. Knuth. En 1957, aparece la Programación Dinámica de la mano de R. Bellman. En 1960, S. Golomb y L. Baumet presentan las Técnicas Backtracking para la exploración de grafos. Se publican en 1962 los primeros algoritmos del tipo Divide y Vencerás: el *QuickSort* de Charles Hoare y el de la multiplicación de grandes enteros de A. Karatsuba e Y. Ofman.

En 1959, Jack Kilby (1923-) presenta el primer circuito integrado, un conjunto de transistores interconectados con resistencias, en una pequeña pastilla de silicio y metal, llamada chip. Fue a partir de este hecho que las computadoras empezaron a fabricarse de menor tamaño, más veloces y a menor costo, debido a que la cantidad de transistores colocados en un solo chip fue aumentando en forma exponencial.

2.1.4 Tercera Generación (circuitos integrados y miniaturización)

A partir del circuito integrado, se producen nuevas máquinas, mucho más pequeñas y rápidas que las anteriores, así aparecen las IBM 360/91, IBM 195, SOLOMON (desarrollada por la *Westinghouse Corporation*) y la ILLIAC IV, producida por Burroughs, el Ministerio de Defensa de los EE.UU y la Universidad de Illinois.

Seymour Cray (1925-1996) revoluciona el campo de la supercomputación con sus diseños: en 1964, el CDC 6600, que era capaz de realizar un millón de operaciones en coma flotante por segundo; en 1969, el CDC 7600, el primer procesador vectorial, diez veces más rápido que su predecesor.

En cuanto a los avances teóricos, a mediados de los 60, un profesor de Ciencias de la Computación, Niklaus Wirth, desarrolla el lenguaje PASCAL, y en Berkeley, el profesor Lotfi A. Zadeh, publica su artículo *Fuzzy Sets*, que revoluciona campos como la Inteligencia Artificial, la Teoría de Control o la Arquitectura de Computadores.

En 1971, Intel introduce el primer microprocesador. El potentísimo 4004 procesaba 4 bits de datos a la vez, tenía su propia unidad lógicoaritmética, su propia unidad de control y 2 chips de memoria. Este conjunto de 2.300 transistores que ejecutaba 60.000 operaciones por segundo se puso a la venta por 200 dólares. Muy pronto Intel comercializó el 8008, capaz de procesar el doble de datos que su antecesor y que inundó los aparatos de aeropuertos, restaurantes, salones recreativos, hospitales, gasolineras...

A partir de aquí nacieron las tecnologías de integración a gran escala (LSI) y de integración a muy gran escala (VLSI), con las que procesadores muy complejos podían colocarse en un pequeño chip.

Sin embargo, hasta este momento, por motivos económicos, complejidad de uso y dificultad de mantenimiento, los computadores habían sido patrimonio de universidades, organismos militares y gubernamentales, y grandes empresas.

En 1975, Popular Electronics dedicó su portada al primer microcomputador del mundo capaz de rivalizar con los modelos comerciales, el Altair 8800.

2.1.5 Cuarta Generación (ordenadores personales de uso doméstico)

El Altair 8800, producido por una compañía llamada Micro Instrumentation and Telemetry Systems (MITS), se vendía a 397 dólares, lo que indudablemente contribuyó a su popularización. No obstante, el Altair requería elevados conocimientos de programación, tenía 256 bytes de memoria y empleaba lenguaje máquina. Dos jóvenes, William Gates y Paul Allen, ofrecieron al dueño de MITS, un software en BASIC que podía correr en el Altair. El software fue un éxito y, posteriormente Allen y Gates crearon Microsoft.

Paralelamente, Steven Wozniak y Steven Jobs, también a raíz de ver el Altair 8800 en la portada de Popular Electronics, construyen en 1976, la Apple I. Steven Jobs con una visión futurista presionó a Wozniak para tratar de vender el modelo y el 1 de Abril de 1976 nació Apple Computer. En 1977, con el lanzamiento de la Apple II, el primer computador con gráficos a color y carcasa de plástico, la compañía empezó a imponerse en el mercado.

En 1981, IBM estrena una nueva máquina, la IBM Personal Computer, protagonista absoluta de una nueva estrategia: entrar en los hogares. El corazón de esta pequeña computadora, con 16 Kb de memoria (ampliable a 256), era un procesador Intel, y su sistema operativo procedía de una empresa recién nacida llamada Microsoft.

En 1984, Apple lanza el Macintosh, que disponía de interfaz gráfico para el usuario y un ratón, que se hizo muy popular por su facilidad de uso.

2.1.6 Historia y desarrollo tanto del hardware como del software.

A continuación se ofrece una lista de algunos eventos claves que hicieron que la computadora personal llegara a ser lo que conocemos hoy. Cada uno de estos eventos fue un instrumento bien sea en el desarrollo del hardware de la PC o de su software.

Año	Evento
1961	Fairchild Semiconductor saca el primer circuito integrado disponible comercialmente.
1963	Douglas Engelbart Patenta el dispositivo apuntador del ratón.
1970	Intel Introduce el Microprocesador 4004
1971	IBM Introduce el Disco Flexible
1974	Intel Lanza el microprocesador 8080
1975	Mits Altair 8800 Se vende en forma de kit por USA $375
1976	Steven Wozniak y Steven Jobs Crean el Apple I
1977	Se conformó Microsoft Corporatión por Bill Gates y Paul Allen. Apple Computer produce el Apple II.
1978	Intel produce el Microprocesador 8086
1979	Visi Cals, la primera aplicación de alto impacto, sale a la venta. Intel produce el Microprocesador 8088.
1980	Se introduce la computadora Apple III
1981	Se introduce la PC 5150 de IBM con el Sistema Operativo PC-DOS (MS-DOS) 1.0
1982	Se introduce la computadora COMMODORE 64. Intel produce el Microprocesador 80286. Se introduce la PC-portatil de COMPAQ.
1983	Se introduce la aplicación Lotus 123. Se introduce la PC-XT de IBM. Se introduce el MS-DOS 2.0.
1984	Hewlett Packard saca la impresora Laser Jet. Phoenix saca su ROM-BIOS
1985	Intel introduce su microprocesador 80386DX. Se introduce Microsoft Windows 1.0. Las unidades CD-ROM de la PC se hacen disponibles.
1986	Se introduce la primera PC 80386.
1987	Se introduce la computadora Apple Mancistosh. IBM introduce las computadoras PS/2 con OS/2 y gráficos VGA.
1988	Intel presenta el microprocesador 80386 SX. Steven Jobs introduce la computadora NEXT.
1989	Intel anuncia el Microprocesador 486
1990	Se dio a conocer Microsoft Windows 3.0
1991	AMD da a conocer su clon del microprocesador 386.
1992	Intel da a conocer el microprocesador 486 DX2. Se lanza Windows 3.1
1993	Se anuncia el microprocesador Intel Pentium.
1994	Se anuncia el Navegador Netscape Navigator. Iomega introduce la unidad Zip.
1995	Se introduce el microprocesador Pentium Pro.
1998	Se da a conocer el microprocesador Pentium II.

Como se puede observar, muchos eventos independientes, todos ellos moderadamente relacionados, fueron instrumento para el desarrollo de la computadora personal como existe hoy. La estructura general de la PC ha cambiado muy poco desde sus comienzos a finales de la década de 1970. Sin embargo, su velocidad, sus capacidades y su potencia han aumentado casi exponencialmente.

2.2 Arquitectura Harvard.

El termino arquitectura Harvard proviene de la computadora Harvard Mark I, se encarga de almacenar instrucciones en cintas perforadas y los datos en interrupciones. Es la misma arquitectura de computadoras, posee dispositivos de almacenamiento que se encuentran separados físicamente para los datos y las instrucciones. Las partes principales de las computadoras es la memoria y la CPU, la primera guarda los datos y la CPU los procesa. A través de la memoria no solo se pueden manejar los datos sino también el lugar donde se encuentran almacenados, estos dos parámetros son de mucha importancia para la CPU.

El CPU trabaja con mucha mayor velocidad que las memorias con las que trabaja. Para que la memoria valla más rápida se aconseja suministrar una pequeña memoria llamada caché que es muy rápida. Se pueden conseguir memorias con más velocidad pero estas poseen un alto precio. Si los datos están en la caché rendirán mucho mas tiempo, pero si la caché tiene que obtener los datos a través de la memoria principal estos no perduraran mucho. La arquitectura Harvard permite que los datos y las instrucciones se almacenen en chaches separados para obtener mejor rendimiento. Se utiliza en procesadores de señal digital y en DSPs, que son utilizados en productos para procedimiento de video y audio.

2.3. EVOLUCIÓN DE LOS MICROPROCESADORES

La computadora personal de hoy, es un equipo mucho mas potente que las PC de hace tan sólo 5 años, si mencionar las diferencias existentes respecto a sus primeros días de evolución. Recuerdo que toda la historia de las computadoras se reduce aun poco más de 50 años y la computadora personal a estado presente desde hace mas o menos 20 años.

2.3.1 Macrocomputadoras (Mainframes).

La macrocomputadora o Mainframes, puede llenar literalmente una habitación. Estas computadoras grandes, se utilizan para suplir las necesidades de computación de compañías y corporaciones grandes y también en grandes centros de telecomunicaciones. Son muy potentes y de gran capacidad de almacenamiento y procesamiento. La desventaja de las macrocomputadoras para ser utilizadas como computadoras personales en su tamaño, su inmensa potencia de computación y su precio.

Cada usuario de una macrocomputadora trabajaba en una terminal, que es un dispositivo que combina un monitor con un teclado y esta unido directamente a la macrocomputadora mediante un cable destinado para ese uso. Los primeros usuarios de la macrocomputadora no tenían un ratón, y todos los datos se ingresaban como texto. Es

más probable que el usuario de una macrocomputadora de hoy se conecte a está a través a una red local y utilice una PC como dispositivo o terminal.

2.3.2 Minicomputadoras

La minicomputadora se desarrollo para suplir las necesidades de computación de compañías más pequeñas y departamentos más grandes en las corporaciones. Se conoce como una computadora, tiene la misma funcionalidad de la macrocomputadoras pero a menor escala: se desarrollo principalmente para abrir nuevos mercados para las computadoras después de que la mayoría de las empresas grandes habían comprado las macrocomputadoras. Debido de los avances de la tecnología, la minicomputadora de hoy puede suplir todas las necesidades de computación de una compañía pequeña a mediana, así como funcionar como un potente servidor de comunicaciones.

2.3.3 Computador Portátil

En muchas ocasiones es imprescindible poder trasladar el computador, y con él sus datos y sus programas, al lugar donde se va a utilizar. Los primeros modelos eran unos pesados maletines con un mini monitor del tamaño de un billete y un teclado plegable o acoplable.

Actualmente todavía existen estos portátiles, pero ahora cuentan con un monitor de cristal liquido (LCD) a color de tamaño y potencia aceptables. Su principal ventaja frente al resto de modelos portátiles, radica en que permiten instalar tarjetas de expansión totalmente normales, por ejemplo tarjetas especiales para conectar instrumentos de medida y detectores que envían los datos obtenidos directamente a un programa de computador.

2.3.4 Laptops

Cuando la tecnología cristal liquido fue capaz de crear pantalla, sé acelero el salto a otra clase de computadores portátiles. Entonces nació laptops, así la pantalla, la unidad central y el teclado forman una unidad compacta: la pantalla constituye la tapa plegable del aparato y al levantarse deja libre el teclado. Solo los aparatos más modernos poseen una acumulador que les da cierta autonomía.

El computador portátil moderno se denomina" Notebook", cabe en cualquier maletín y apenas pesa tres kilos incluido el acumulador.

2.3.5 Computador de Bolsillo

Todavía hay una serie de modelos más pequeños, los denominados computadores, organizadores y base de datos de bolsillo. Estos aparatos tienen poco que ver con un PC ya que necesitan trabajar con el mínimo consumo y la mayor garantía. En pocos casos constituye una alternativa económica a los costosos Notebook, no son compatibles con IBM pero disponen de un software que les permite intercambiar datos con el computador.

2.3.6 Computadores de Red (NC)

Los computadores de red NC prometen una reducción sustancial de los gastos que incluyen el mantenimiento de una red. Se tratan de computadores económicos sin disco duro ni unidad de disquetes que únicamente pueden acceder a través de una red. El software esta instalado en un potente servidor controlado por un administrador. Así el usuario solo lo conecta a la red, luego el computador carga automáticamente de la red el sistema operativo y el software que necesite.

En un futuro se prevé su aplicación en el ámbito domestico, convenientemente equipado podría recibir los programas y datos a través de la conexión de cable del televisor, así permitiría navegar por Internet o enviar pedidos a tiendas en línea.

2.3.7 Otras Computadoras

Una supercomputadora, es una computadora muy potente utilizada principalmente en aplicaciones espaciales, militares y gubernamentales. Pueden costar decenas de millones de dólares, contiene el equivalente de miles de computadoras personales que comparten la cargo de procesamiento para solucionar problemas muy grandes y complejos en horas o días en lugar de semanas meses o años.

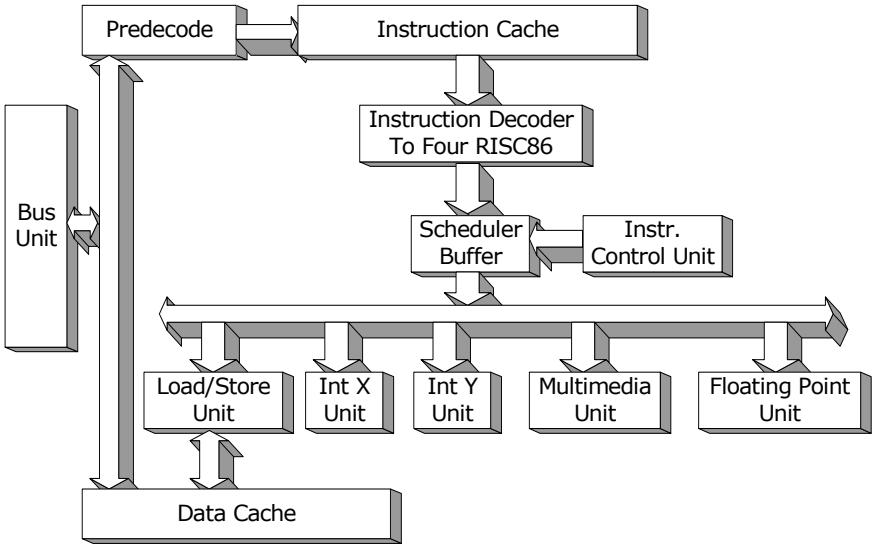
Pareciera como casi todos los dispositivos electrónicos tuvieran una computadora incorporada de alguna forma. Estos procesadores pequeños y para un solo propósito se clasifican como computadoras integradas en otros dispositivos, y su función son controlar, monitorear, o realizar alguna actividad para el dispositivo. Los controles de un horno microondas, la carburación de su auto, la función de su reloj despertador, incluso su reloj de pulso son probables que tenga por lo menos uno y probablemente más, de una computadora integrada.

2.4. MODELOS DE PC

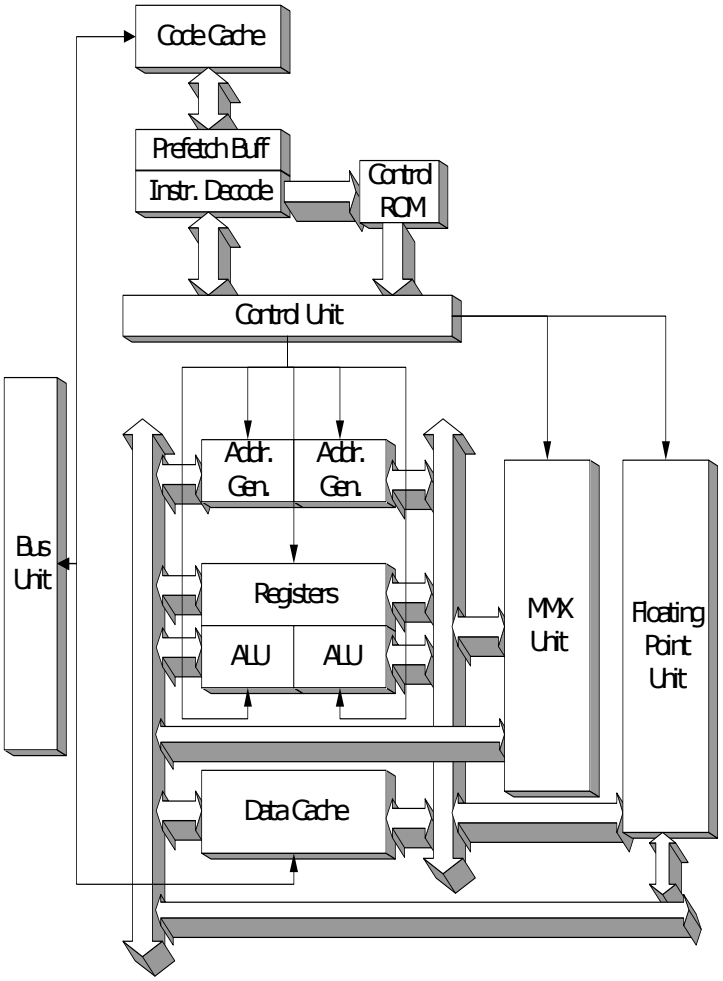
El aspecto habitual de un PC es la caja de escritorio conectada a un monitor y asistida por un teclado. La pantalla se coloca a menudo sobre la unidad central y el teclado delante. Por razones de espacio, ha originado que los PC modernos se presenten con carcasas verticales tipo torre. También existen torres de tamaño medio (Mini/ Midi – Tower). Las cajas tipo torre son mayores que las anteriores y ofrecen más espacio para unidades complementarias, por esta razón se prefieren para albergar los servidores de red.

Dependiendo de los lugares de los ambientes de trabajo se han desarrollado cajas especiales que permiten utilizar el computador en entornos más adversos, como por ejemplo naves industriales. Así en otros lugares es preciso proteger el PC y el monitor con cajas especiales para soportar golpes y movimientos bruscos.

INTEL PENTIUM MMX



AMD K6



Autoevaluación.

1. Realizar un cuadro comparativo que muestre la evolución de los procesadores. Los puntos a incluir son: año, velocidad del reloj, bus, número de transistores, memoria, dirección, memoria virtual, procesador.

Capítulo 3: El papel del rendimiento

Cuando decimos que un computador tiene mejor rendimiento que otro, ¿Qué queremos indicar? Aunque esta pregunta pueda parecer simple, una analogía con los aviones de pasajeros muestra lo sutil que puede ser la pregunta sobre rendimiento. La figura muestra algunos aviones típicos de pasajeros, junto a su velocidad de crucero, alcance y capacidad. Si quisiéramos saber cual de los aviones de esta tabla tiene el mejor rendimiento, lo primero necesitaríamos definir rendimiento. Por ejemplo, considerando las diferentes medidas de rendimiento vemos que el avión con la mayor velocidad de crucero es el Concorde, el avión con mayor alcance es el DC-8 y el avión con la mayor capacidad es el 747. Supongamos que definimos el rendimiento en función de la velocidad. Esto nos lleva a dos posibles definiciones. Se puede definir el avión mas rápido como aquel que tiene mayor velocidad de crucero, llevando a un solo pasajero de un punto a otro en el tiempo mínimo. Sin embargo, si estuviésemos interesados en transportar 450 pasajeros de un punto a otro, el 747 claramente seria el más rápido, como muestra la ultima columna de la figura. Análogamente, podemos definir el rendimiento de un computador de varias formas diferentes.

Si estuviésemos ejecutando un programa en dos estaciones de trabajo diferentes, diríamos que la más rápida es la estación de trabajo que finaliza primero el trabajo. Sin embargo, si se estuviese ejecutando en un centro de calculo que dispusiera de dos grandes computadores a tiempo compartido para ejecutar tareas suministradas por muchos usuarios, se diría que el computador mas rápido es el que completa el máximo numero de tareas durante un día. Como usuario de un computador individual, se esta interesado en reducir el *tiempo de respuesta* – el tiempo entre el comienzo y la finalización de una tarea- que también se denomina *tiempo de ejecución*. Los gestores del centro de cálculo están interesados en incrementar la *productividad* (throughput) – la cantidad total de trabajo realizado en un tiempo determinado.

Por simplicidad, normalmente utilizaremos la terminología más rápida que cuando tratamos de comparar cuantitativamente maquinas. Como rendimiento y tiempo de ejecución son recíprocos, incrementar el rendimiento requiere decrementar el tiempo de ejecución. Para evitar la confusión potencial entre los términos *incrementar y decrementar*, habitualmente diremos *mejorar el rendimiento o mejorar el tiempo de ejecución* cuando queramos indicar “incrementar el rendimiento” y “decrementar el tiempo de ejecución”.

3.1 Medidas del rendimiento

El tiempo es la medida del rendimiento del computador: el computador que realiza la misma cantidad de trabajo en el mínimo tiempo es el más rápido. El *tiempo de ejecución* de un programa se mide en segundos por programa. Pero el tiempo puede definirse de diferentes formas, dependiendo de lo que contemos. La definición mas sencilla de tiempo se denomina *tiempo de reloj, tiempo de respuesta, o tiempo transcurrido*. Este es el tiempo total para completar una tarea, incluyendo accesos al disco, accesos a memoria, actividades de entrada /salida (E/S), gastos del sistema operativo -todo-. Sin embargo, los computadores son, con frecuencia, de tiempo compartido, y un procesador puede trabajar en varios programas simultáneamente. En estos casos, el sistema puede tratar de optimizar la productividad en lugar de intentar de minimizar el tiempo transcurrido para un

programa. Por consiguiente, distinguiremos con frecuencia entre tiempo transcurrido y tiempo que el procesador esta trabajando en beneficio nuestro. El *tiempo de ejecución de la CPU* o simplemente *tiempo de CPU,* que reconoce esta distinción, es el tiempo que la CPU emplea para realizar esta tarea y no incluye el tiempo de espera de las E/S o de ejecución de otros programas. (Recordar que aunque el tiempo de respuesta experimentado por el usuario sea el tiempo transcurrido del programa, no es el tiempo de CPU). El tiempo de CPU puede descomponerse además en el tiempo empleado por la CPU en el programa, que se denomina *tiempo de usuario de CPU,* y en tiempo empleado por la CPU en realizar las tareas del sistema operativo en beneficio del programa, que se denomina *tiempo de CPU del sistema.* Diferenciar entre tiempos de CPU del sistema y de usuario es difícil de realizar con precisión, porque con frecuencia es difícil asignar responsabilidades de las actividades del sistema operativo a un programa de usuario en lugar de a otro.

La descomposición del tiempo transcurrido para una tarea se refleja en la orden de tiempo de UNIX, que en un caso devolvió 90.7u 12.9s 2:39 65%

El tiempo de CPU de usuario es 90,7 segundos, el tiempo CPU del sistema es 12,9 segundos, el tiempo transcurrido es 2 minutos y 39 segundos (159 segundos), y el porcentaje de tiempo transcurrido que es el tiempo de CPU es:

$$\frac{90{,}7 + 12{,}9}{159} = 0{,}65$$

O el 65 por 100. Más de una tercera parte del tiempo transcurrido en este ejemplo se utilizo esperando las E/S, ejecutando otros programas, o ambas cosas. A veces ignoramos el tiempo de CPU cuando examinamos el tiempo de ejecución de la CPU a causa de la imprecisión de la automedida de los sistemas operativos y de la injusticia de incluir el tiempo de CPU del sistema cuando se comparan rendimientos entre maquinas con diferentes sistemas operativos. Por otro lado, el código del sistema en algunas maquinas es el código del usuario en otras, y ningún programa corre sin que ningún sistema operativo se ejecute en el hardware, así puede darse el caso de utilizar la suma del tiempo de CPU de usuario y del tiempo de CPU del sistema como medida del tiempo de ejecución del programa.

Por consistencia, mantenemos la distinción entre el rendimiento basado en el tiempo transcurrido y el basado en el tiempo de ejecución de la CPU. Utilizaremos el término *rendimiento del sistema* para referenciar el tiempo transcurrido en un sistema *descargado,* y utilizaremos *rendimiento de CPU* para referenciar el tiempo de CPU de *usuario.* En este capitulo nos centraremos en el rendimiento de la CPU, aunque nuestras discusiones sobre como resumir el rendimiento puedan aplicarse a las medidas del tiempo transcurrido o del tiempo de CPU.

Aunque como usuarios de los computadores nos preocupamos por el tiempo, cuando examinamos los detalles de la maquina es conveniente considerar el rendimiento con otras métricas. En particular, los diseñadores de computadores pueden pensar en una maquina utilizando una medida que relacione lo rápidamente que el hardware puede realizar funciones básicas. Casi todos los computadores se construyen utilizando un reloj

que corre a una velocidad constante y determina cuando tienen lugar los eventos en el hardware. Estos intervalos discretos de tiempo se denominan *ciclos de reloj* (o pulsaciones, pulsaciones de reloj, periodos de reloj, relojes, ciclos). Los diseñadores referencian la duración de un *periodo de reloj* como el tiempo necesario para completar un *ciclo de reloj* (por ejemplo, 10 nanosegundos, o 10 ns), y la *frecuencia de reloj* (por ejemplo, 100 megahercios, o 100 MHz) como la inversa del periodo de reloj. En la sección siguiente formalizaremos la relación entre los ciclos de reloj del diseñador hardware y los segundos del usuario del computador.

3.2 Métricas de rendimiento

Los usuarios y diseñadores con frecuencia examinan el rendimiento utilizando métricas diferentes. Si pudiéramos relacionar estas métricas diferentes, podríamos determinar el efecto de un cambio de diseño en el rendimiento como lo puede ver un usuario. Como en este punto estamos confinados al rendimiento de la CPU, la medida del rendimiento en el extremo inferior es el tiempo de ejecución de la CPU. Una sencilla formula relaciona las métricas básicas (los ciclos de reloj y la duración ciclo de reloj) con el tiempo de CPU: métricas básicas (los ciclos de reloj y la duración ciclo de reloj) con el tiempo de CPU:

Tiempo de ejecución de CPU para un programa	=	Ciclos de reloj de CPU para un programa	X	Duración del ciclo de reloj

Alternativamente, como la frecuencia de reloj y la duración del ciclo de reloj son inversas:

$$\text{Tiempo de ejecución de CPU para un programa} = \frac{\text{Ciclos de reloj de CPU para un programa}}{\text{Frecuencia de reloj}}$$

Esta formula aclara que el diseñador hardware puede mejorar el rendimiento reduciendo la duración del ciclo de reloj o el numero de ciclos de reloj que se necesitan para un programa. Muchas técnicas que hacen disminuir el número de ciclos de reloj también aumentan la duración del ciclo de reloj.

EJEMPLO

Nuestro programa favorito se ejecuta en 10 segundos en el computador A, que tiene un reloj de 100 MHz. Estamos tratando de ayudar a un diseñador de computadores que construye una maquina B, que ejecutara este programa en 6 segundos. El diseñador ha determinado que es posible un incremento sustancial en la frecuencia de reloj, pero este incremento afectara al resto del diseño de la CPU, haciendo que la maquina B necesite 1,2 veces el numero de ciclos de reloj que la maquina A para este programa. ¿Qué frecuencia de reloj debería utilizar el diseñador?

RESPUESTA
Primero determinamos el número de ciclos de reloj requeridos para el programa en A:

$$\text{Tiempo de CPU A} = \frac{\text{Ciclos de reloj de CPU A}}{\text{Frecuencia de reloj A}}$$

$$10 \text{ segundos} = \frac{\text{Ciclos de reloj de CPU A}}{100 \times 10(6) \frac{\text{ciclos}}{\text{Segundo}}}$$

$$\text{Ciclos de reloj de CPU A} = 10 \text{ segundos} \times 100 \times 10(6) \frac{\text{ciclos}}{\text{Segundo}} = 1.000 \times 10(6) \text{ ciclos}$$

El tiempo de CPU para B se puede hallar mediante la siguiente ecuación:

$$\text{Tiempo de CPU B} = 1,2 \times \frac{\text{Ciclos de reloj de CPU A}}{\text{Frecuencia de reloj B}}$$

$$6 \text{ segundos} = 1,2 \times \frac{1.000 \times 10(6) \text{ ciclos}}{\text{Frecuencia de reloj B}}$$

$$\text{Frecuencia de reloj B} = \frac{1,2 \times 1.000 \times 10(6) \text{ ciclos}}{6 \text{ segundos}} = 200 \times 10(6) \text{ ciclos} = 200\text{MHz}$$

La maquina B, por tanto, debe tener dos veces la frecuencia de reloj de la maquina A para ejecutar el programa en 6 segundos.

Autoevaluación.

1. Si la versión de 1998 de un computador ejecuta un programa en 200 s y la versión del computador hecha en el 2000 ejecuta el mismo programa en 150 s, cuál es el incremento de velocidad que el fabricante ha conseguido en dos años?

2. Por qué se usan programas de prueba y conjunto de programas de prueba para medir las prestaciones de un computador?

3. Si se tienen dos implementaciones de la misma arquitectura del repertorio de instrucciones. La máquina A tiene una duración de ciclo de reloj de 10 ns y un CPI de 2.0 para un programa, y la máquina B una duración de ciclo de reloj de 20 ns y un CPI de 1.2 para el mismo programa. Qué máquina es más rápida para este programa y cuánto?

Capítulo 4: El Computador

4.1 Buses del sistema.

El bus se puede definir como un conjunto de líneas conductoras de hardware utilizadas para la transmisión de datos entre los componentes de un sistema informático. Un bus es en esencia una ruta compartida que conecta diferentes partes del sistema, como el microprocesador, la controladora de unidad de disco, la memoria y los puertos de entrada/salida (E/S), para permitir la transmisión de información.

En el bus se encuentran dos pistas separadas, el bus de datos y el bus de direcciones. La CPU escribe la dirección de la posición deseada de la memoria en el bus de direcciones accediendo a la memoria, teniendo cada una de las líneas carácter binario. Es decir solo pueden representar 0 o 1 y de esta manera forman conjuntamente el número de la posición dentro de la memoria (es decir: la dirección). Cuantas más líneas hayan disponibles, mayor es la dirección máxima y mayor es la memoria a la cual puede dirigirse de esta forma. En el bus de direcciones original habían ya 20 direcciones, ya que con 20 bits se puede dirigir a una memoria de 1 Mb y esto era exactamente lo que correspondía a la CPU.

Esto que en le teoría parece tan fácil es bastante mas complicado en la práctica, ya que aparte de los bus de datos y de direcciones existen también casi dos docenas más de líneas de señal en la comunicación entre la CPU y la memoria, a las cuales también se acude. Todas las tarjetas del bus escuchan, y se tendrá que encontrar en primer lugar una tarjeta que mediante el envío de una señal adecuada indique a la CPU que es responsable de la dirección que se ha introducido. Las demás tarjetas se despreocupan del resto de la comunicación y quedan a la espera del próximo ciclo de transporte de datos que quizás les incumba a ellas.

PROCESADOR	Bus de direcciones (bits)	Bus de datos (bits)
8086	20	16
8088	20	8
80186	20	16
80188	20	8
80286	24	16
80386 SX	32	16
80386 DX	32	32
80486 DX	32	32
80486 SX	32	32
PENTIUM	32	64
PENTIUM PRO	32	64

Este mismo concepto es también la razón por la cual al utilizar tarjetas de ampliación en un PC surgen problemas una y otra vez, si hay dos tarjetas que reclaman para ellas el mismo campo de dirección o campos de dirección que se solapan entre ellos.

Los datos en si no se mandan al bus de direcciones sino al bus de datos. El bus XT tenía solo 8 bits con lo cual sólo podía transportar 1 byte a la vez. Si la CPU quería depositar el

contenido de un registro de 16 bits o por valor de 16 bits, tenía que desdoblarlos en dos bytes y efectuar la transferencia de datos uno detrás de otro.

De todas maneras para los fabricantes de tarjetas de ampliación, cuyos productos deben atenderse a este protocolo, es de una importancia básica la regulación del tiempo de las señales del bus, para poder trabajar de forma inmejorable con el PC. Pero precisamente este protocolo no ha sido nunca publicado por lBM con lo que se obliga a los fabricantes a medir las señales con la ayuda de tarjetas ya existentes e imitarlas. Por lo tanto no es de extrañar que se pusieran en juego tolerancias que dejaron algunas tarjetas totalmente eliminadas.

4.2 Memoria

La memoria principal es la parte del ordenador donde residen los programas y los datos que estos utilizan en el momento de su ejecución. Cuando se desea ejecutar un programa, como normalmente esta almacenado en un dispositivo de almacenamiento secundario, lo primero que se hace es copiarlo en memoria. Este proceso lo realiza el sistema operativo teniendo también que gestionar las posiciones de memoria donde se va a cargar el programa una vez la copia a finalizado, el procesador inicia la ejecución de las instrucciones leyéndolas de memoria a partir de la posición donde el sistema operativo a cargado el programa

El tamaño máximo que puede tener la memoria física de un ordenador viene determinado por el numero de líneas de direcciones que posee el procesador.

Un procesador con 16 líneas de direcciones será capaz de acceder al rango de posiciones de memoria desde la 0 hasta 0ffffh.
Se define el espacio de Direccionamiento del procesador como la cantidad de posiciones de memoria a las que puede acceder.

Para implementar el sistema de memoria se utilizan memorias RAM, Caracterizadas principalmente porque su tiempo de acceso es independiente tanto de la posición a la que se quiere acceder como de la secuencia de los accesos anteriores.
Frente a estos dispositivos se encuentran las memorias de acceso secuencial. cuyo tiempo de acceso depende de la secuencia de operaciones realizadas.

4.2.1 Latencia, rendimiento y ancho de banda

Al discutir la segmentación de cauce del procesador, utilizaremos el término *latencia y rendimiento* para describir el tiempo necesario para completar una operación individual y la velocidad a la que se pueden ir completando las distintas operaciones. Estos términos también se utilizan en el análisis de los sistemas de memoria y tienen el mismo significado que el que usamos al estudiar los procesadores. Un termino adicional, se utiliza en el contexto de los sistemas de memoria, es el a*ncho de banda,* que describe la velocidad total a la que se pueden transferir datos entre el procesador y la memoria. Se puede considerar que el ancho de banda es el producto entre el rendimiento y la cantidad de datos a los que se accede en cada operación de memoria.

4.2.2 Segmentación, paralelismo y precarga.

Si todas las operaciones de memoria se ejecutaran secuencialmente, el cálculo de la latencia y el ancho de banda de un sistema de memoria serian muy sencillos. Sin embargo, muchos sistemas de memoria se diseñan de una forma tal, que la relación entre estas dos magnitudes se vuelve mas compleja. Los sistemas de memoria pueden segmentarse de la misma forma que se hace con los procesadores, permitiendo que varias operaciones se puedan realizar concurrentemente para mejorar el rendimiento. Por otro lado, muchas tecnologías de memoria necesitan una cierta cantidad de tiempo de inactividad entre accesos a memoria. Esta cantidad de tiempo se utiliza para preparar, o *precargar*, la circuitería para el siguiente acceso. La precarga de la circuitería hace parte del trabajo de acceder a la memoria antes de que llegue la dirección. Esto reduce el retardo entre el momento en el que se envía la dirección a la que se quiere acceder y el momento en el que la operación se completa. Si el sistema de memoria esta libre la mayor parte del tiempo, realizar la precarga al final de cada operación de memoria mejora las prestaciones ya que generalmente no hay ninguna otra operación esperando para usar el sistema de memoria. Si el sistema de memoria esta siendo usado la mayor parte del tiempo, la velocidad a la que se pueden ir completando las operaciones dependerá de la suma entre la latencia de memoria y el tiempo de precarga.

Otra de las formas con las que los diseñadores mejoran las prestaciones de los sistemas de memoria consiste en permitir varios accesos a memoria en paralelo. Esto se consigue normalmente incorporando varias memorias al bus de memoria del procesador, tal y como se muestra en la figura. Al utilizarse un único bus de memoria, no es posible que una referencia a memoria comience o finalice al mismo tiempo, ya que solo una petición puede usar el bus en un instante dado. Sin embargo, el procesador puede enviar peticiones de memoria a las memorias que se encuentren libres mientras espera a que se completen otras peticiones. Como las peticiones de memoria generalmente consumen varios ciclos de reloj, este mecanismo mejora la velocidad a la que se pueden ir gestionando las peticiones de memoria sin tener que incrementar el numero de pines de E/S del procesador, lo que incrementaría el coste del mismo.

Los sistemas que admiten peticiones de memoria en paralelo se pueden dividir en dos tipos.

Los sistemas de memoria *replicada* proporcionan varias copias de toda la memoria. Esto requiere decir cada copia de la memoria puede admitir cualquier petición de acceso a memoria, aunque se aumenta la cantidad de memoria necesaria en un factor igual al numero de copias.

Para mantener los mismos contenidos de cada una de las memorias, todas las operaciones de almacenamiento deben enviarse a cada copia de la memoria, haciendo que los almacenamientos sean mucho mas costosos que las cargas en cuanto a la cantidad de ancho de banda que consumen.

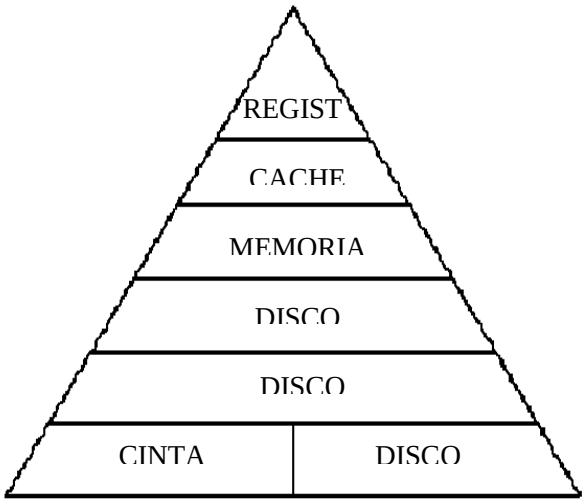
La forma mas común de sistema de memoria paralelo es un sistema de memoria por bancos. En este tipo de sistemas, los datos son divididos o *entrelazados*, entre las memorias, de tal forma que cada memoria contiene solo una fracción de los datos. Generalmente, se utilizan algunos de los bits de la dirección para seleccionar el banco de memoria en el que un determinado dato se encuentra. Por ejemplo, en el sistema

ilustrado en la figura, las posiciones de memoria cuya dirección acabe en 0b00 se pueden colocar en el banco de mas a la izquierda, los que acaben en 0b01 en el siguiente banco y así sucesivamente. Igualmente, podríamos seleccionar cualesquiera otros 2 bits de la dirección para seleccionar el banco de memoria. Normalmente, se utilizan los bits de dirección mas bajos para seleccionar el banco ya que así, los accesos a posiciones de memoria consecutivos irán a parar a bancos diferentes.

Los sistemas de memoria organizados por bancos presentan la ventaja de que no necesitan mayor cantidad de memoria que el sistema de memoria equivalente con una sola memoria y que solo hay que enviar las operaciones de almacenamiento a aquel banco que contenga la dirección en la que se escribe. Sin embargo, presentan el problema de que es posible que haya dos accesos a posiciones que se encuentren en el mismo banco, por lo que una de las operaciones tendrá que esperar hasta que la otra se complete. En la mayoría de los casos, la memoria adicional necesaria para implementar un sistema de memoria replicada no compensa la perdida en ancho de banda debida a los conflictos por el mismo banco de memoria, por lo que los sistemas de memoria por bancos son actualmente mucho mas utilizados que los de memoria replicada.

4.2.3 Jerarquía de Memoria

La solución tradicional para almacenar una gran cantidad de datos es una jerarquía de memoria, como se ilustra en la gráfica. En la cúspide están los registros de la CPU,a los que puede tenerse acceso a la velocidad máxima de la CPU. Luego viene la memoria caché, que actualmente es del orden de 32 Kb a unos cuantos megabytes. Sigue la memoria principal, con tamaños que actualmente van de 16 Mb para los sistemas más económicos hasta decenas de gigabytes en el extremo superior. Después vienen los discos magnéticos. Por último viene la cinta magnética y los discos ópticos para el almacenamiento de archivos.



Al bajar por la jerarquía, tres parámetros clave crecen. Primero, el tiempo de acceso se alarga. Los registros de la CPU pueden accederse en unos cuantos nanosegundos. Las memorias caché requieren un múltiplo pequeño del tiempo de acceso de los registros. Los accesos a la memoria principal suelen ser de unas cuantas decenas de nanosegundos. Luego viene una brecha grande, pues los tiempos de acceso a disco son de por lo menos 10 ms, y el acceso a cinta o disco óptico puede medirse en segundos si es preciso traer los medios e insertarlos en la unidad.

Segundo la capacidad de almacenamiento aumenta al bajar por la jerarquía. Los registros de la CPU pueden contener tal vez 128 bytes; los cachés unos cuantos megabytes.; las memorias principales decenas o miles de megabytes; los discos magnéticos de unos cuantos gigabytes a decenas de gigabytes. Las cintas y los discos ópticos generalmente

se guardan fuera de línea, así que su capacidad está limitada sólo por el presupuesto del propietario.

Tercero, el número de bits que se obtiene por dólar invertido aumenta al bajar por la jerarquía. Aunque los precios actuales cambian rápidamente, la memoria principal se mide en dólares/megabyte; el almacenamiento en discos magnéticos en centavos de dólar/megabyte, y la cinta magnética en dólares/gigabyte o menos.

En realidad, los sistemas de memoria de los computadores modernos están formados por jerarquías de memoria multinivel, tal y como se muestra en la parte derecha de esa misma figura. En ella se muestra una jerarquía de memoria de 3 niveles que consiste en una cache, una memoria principal y una memoria virtual.

La razón principal de que los sistemas de memoria se construyan de forma jerárquica es que el coste por bit de una tecnología de memoria es generalmente proporcional a la velocidad de dicha tecnología. Las memorias rápidas, como las RAM estáticas (SRAM), tienden a tener un alto coste por bit (tanto en lo que se refieren al precio como al área del chip), haciendo que sea prohibitivo construir la memoria de un computador usando exclusivamente estos dispositivos. Otras tecnologías más lentas, como las memorias RAM dinámicas (DRAM), son mas baratas, haciendo que sea mas factible la construcción de grandes memorias usando estas tecnologías.

En una jerarquía de memoria, los niveles mas cercanos al procesador, como es el caso de la cache de la figura, contienen una cantidad relativamente pequeña de memoria que, sin embargo, esta implementada usando una tecnología de memoria rápida que proporciona un tiempo de acceso pequeño. Conforme descendemos en la jerarquía, cada nivel contiene mas capacidad de almacenamiento pero necesitan mayor tiempo de acceso que el nivel inmediatamente superior.

El objetivo de la jerarquía de memoria es mantener en los niveles mas altos de la jerarquía los datos a los que mas referencia se hace por un programa, de tal forma que la mayoría de los accesos a memoria puedan ser gestionados por los niveles superiores. Esto hace que el sistema de memoria tenga un tiempo medio de acceso similar al tiempo de acceso del nivel mas rápido, pero con un coste medio por bit similar al del nivel mas bajo.

En general, no es posible predecir a que posiciones de memoria se va a acceder con mayor frecuencia, por lo que los computadores usan un método basado en la demanda para determinar que datos deben mantenerse en los niveles más altos de la jerarquía.

Cuando se envía una petición de memoria a la jerarquía, se comprueba el nivel más alto para ver si contiene la dirección pedida. Si es así, la petición se lleva a cabo. Si no, se comprueba el siguiente nivel mas bajo de la jerarquía repitiéndose este proceso hasta que se encuentre el dato o se llegue al nivel inferior, en el que tenemos la total certeza de que estará el dato.

Si una petición de memoria no se puede gestionar por el nivel superior de la jerarquía, se copia desde el nivel que contenga el dato un *bloque* de posiciones de memoria consecutivas, entre ellas la dirección perdida, en todos los niveles por encima en la jerarquía. Esto se hace por dos razones fundamentales. La primera es que muchas

tecnologías de almacenamiento, como las DRAM en modo pagina(que discutiremos mas tarde) y los discos duros, permiten que se lean o escriban direcciones aleatoriamente.

Esto hace que sea más rápido copiar un bloque completo de datos en los niveles superiores de la jerarquía que trae cada byte del bloque desde los niveles más bajos de la jerarquía de forma individual. La segunda es que la mayoría de los programas cumplen con el *principio de localidad*: las referencias a memoria que ocurren cercanas en el tiempo tienden a acceder a direcciones también cercanas, haciendo que sea mas probable que se vaya a acceder a otras direcciones dentro de un mismo bloque una vez que se haya accedido por primera vez a una dirección de dicho bloque.

Siempre y cuando la probabilidad de que se haga referencia a una dirección dentro de un bloque sea suficientemente alta, el uso de bloques de datos reducirán el tiempo de medio de acceso, ya que traerse el bloque completo consume menos tiempo que captar cada palabra del bloque de forma separada. Cada nivel de la jerarquía de memoria generalmente tiene su propio tamaño de bloque, dependiendo de las características de los niveles inferiores a él en la jerarquía. Por ejemplo, las caches tienden a tener tamaños de bloque de aproximadamente 64 bytes, mientras que las memorias principales generalmente tienen bloques de unos 4 KB debido a que la cantidad de tiempo necesaria para copiar un bloque grande de datos de la memoria virtual es solo un poco mayor que la necesaria para copiar 1 byte, mientras que el tiempo necesario para copiar un bloque de datos en la cache desde la memoria principal es mas parecido al resultante de tener que copiar cada byte individualmente.

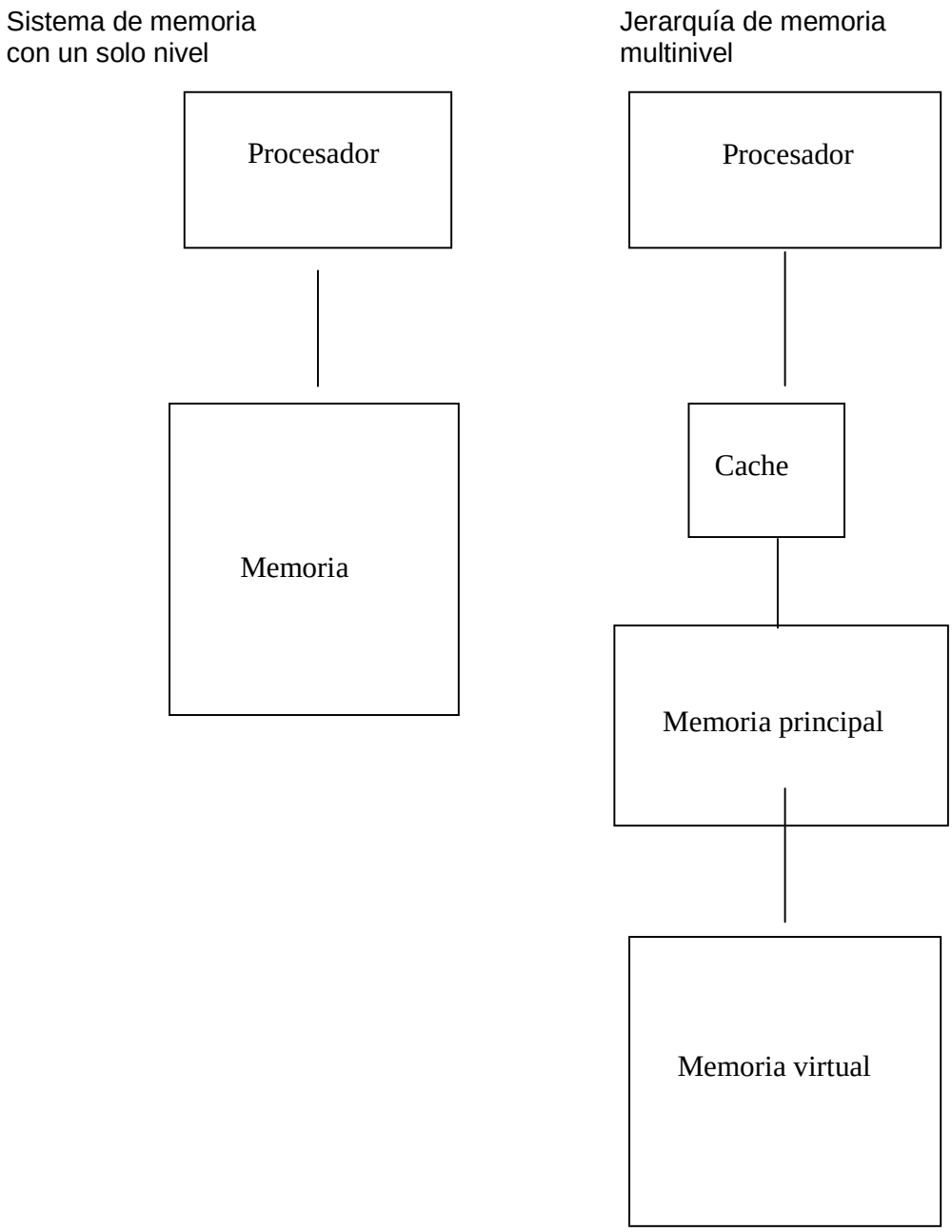
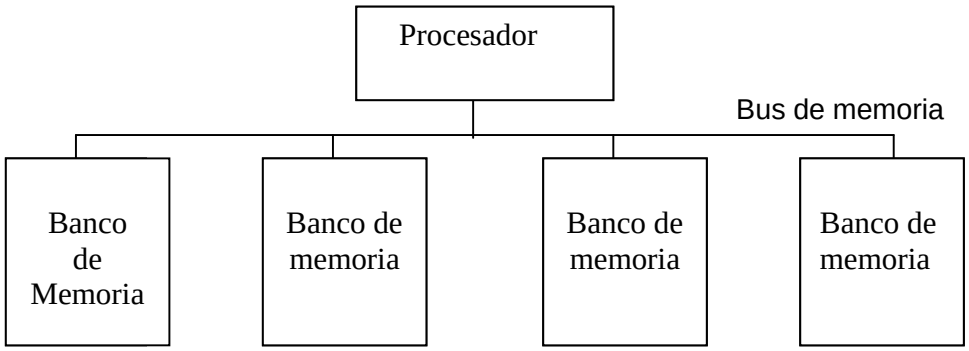


Figura: Jerarquías de memoria.

Figura: Sistema de memoria por bancos/ replicada



En la jerarquía de memoria mostrada en la figura, los diferentes niveles de la jerarquía tienen sus propios nombres específicos. Esto proviene del hecho de que cada nivel se suele implementar de forma muy diferente, así que los arquitectos de computadores utilizan diferentes nombres para describirlos. El nivel superior de la jerarquía se suele denominar la *cache*. Las cache se suelen implementar usando SRAM y la mayoría de los computadores modernos tienen al menos dos niveles de memoria cache en su jerarquía de memoria. Las caches están provistas del hardware necesario para llevar la cuenta de las direcciones que se almacenan en ellas, suelen ser relativamente pequeñas y tienen tamaños de bloque pequeños, normalmente entre 32 y 128 bytes.

La memoria principalmente de un computador, que se implementa generalmente con DRAM, deja que sea el software el que lleva la cuenta de las direcciones contenidas en ella y tiene un tamaño de bloque grande, con frecuencia de varios kilobytes. Finalmente, la memoria virtual se implementa normalmente usando los discos duros y contienen todos los datos del sistema de memoria.

Existe un conjunto de términos especiales para escribir las jerarquías de memoria. Cuando la dirección a la que una operación hace referencia se encuentra en un determinado nivel de la jerarquía de memoria, se dice que ha habido un *acierto* en ese nivel. De no ser así, se producirá un *fallo.* De forma similar, la *tasa de aciertos* de un nivel es el porcentaje de referencias que han llegado a ese nivel y han concluido en acierto, y la *tasa de fallos* es el porcentaje de referencias que ha llegado a ese nivel y han concluido en fallo. La tasa de aciertos y de fallos de un determinado nivel de jerarquía siempre suma el cien por cien. Es importante observar que ni la tasa de aciertos ni la de fallos tienen en cuenta referencias que han sido gestionadas por niveles superiores a la jerarquía. Por ejemplo, una petición que es gestionada por la cache en la jerarquía de memoria de nuestro ejemplo por encontrarse en ella el dato, no contabilizara ni en la tasa de aciertos ni en la de fallos de la memoria principal.

Tal y como se ha descrito anteriormente, cuando ocurre un fallo en un nivel de la jerarquía, se copia un bloque de datos entre los que se encuentra la dirección que ha provocado el fallo en dicho nivel. Conforme se va ejecutando un programa, el nivel se ira llenando de datos y se quedara sin espacio libre donde añadir nuevos bloques. Cuando esto sucede, debemos quitar un bloque de ese nivel de forma que se pueda disponer de espacio suficiente para el nuevo bloque.

Este proceso se denomina *reemplazo* y el método con el que el sistema selecciona el bloque a reemplazar se denomina *política de reemplazo*.

Para simplificar el proceso de reemplazo de bloques de datos de un nivel, muchos sistemas de memoria cumplen con una propiedad llamada *inclusión,* según la cual la presencia de una dirección de un determinado nivel del sistema de memoria garantiza que esa dirección este presente en todos los niveles mas bajos de la jerarquía.

Existen otros términos que describen la manera en la que la jerarquía de memoria gestiona las escrituras en memoria (almacenamientos). En los sistemas con *postescritura*, el dato que se escribe se coloca solamente en el nivel más alto de la jerarquía. Cuando el bloque que contiene el dato es eliminado de ese nivel, el dato escrito se copia en el nivel inmediatamente inferior de la jerarquía y así sucesivamente. A los bloques que contienen datos que han sido escritos se les denomina bloques *modificados,* para distinguirlos de los bloques *inalterados,* que no han sido modificados.

La implementación de los sistemas con postescritura es mucho mas sencilla si se cumple con la propiedad de inclusión, ya que nunca será necesario eliminar un bloque de un nivel para dejar espacio para los datos procedentes del nivel superior.

Por su parte, los sistemas de memoria con *escritura inmediata* copian los datos a escribir en todos los niveles de la jerarquía cuando se produce dicha escritura. Existen muchos sistemas con diferentes estrategias de escritura según el nivel de la jerarquía de que se trate.

Por ejemplo, no es inusual encontrarse computadores con caches con escritura inmediata y memoria principal con postescritura.

La decisión de usar escritura inmediata o postescritura en un determinado nivel de la jerarquía depende del compromiso entre ancho de banda y complejidad. Los sistemas con postescritura pueden tener un ancho de banda mayor ya que no necesitan que se acceda a cada nivel de la jerarquía en cada escritura, pero son, sin embargo, mas complejos que los sistemas de escritura inmediata, ya que es necesario llevar la cuenta de que bloques han sido escritos en cada nivel desde la primera vez que se copiaron en el mismo.

4.2.4 Tecnologías de memoria

En los computadores modernos se utilizan normalmente tres tecnologías diferentes para implementar el sistema de memoria: memoria RAM estática (SRAM), memoria RAM dinámica (DRAM) y discos duros. Los discos duros representan, con diferencia, la tecnología más lenta de todas ellas y se reservan para el nivel inferior de la jerarquía de memoria, la memoria virtual. Las memorias SRAM y DRAM son del orden de un millón de veces más rápidas que los discos duros, por lo que son las tecnologías escogidas para implementar la cache y la memoria principal de prácticamente todos los computadores.

4.2.5 Organización del chip de memoria

Los chips de memoria SRAM y DRAM tienen la misma estructura básica, que es la mostrada en la figura. Los datos se almacenan en una matriz rectangular de *celdas de memoria,* cada una de las cuales guarda un bit de datos. Para leer un dato de dicha

matriz de celdas, la mitad de la dirección de lectura (generalmente los bits mas significativos) se hace pasar por un decodificador. El decodificador activa la *fila* que corresponda al valor de sus bits de entrada, lo que provoca que todas las celdas de esa fila hagan llegar su dato a la *línea de bit* a la que estén conectados. La otra mitad de la dirección de memoria se usa ahora como entrada de control de un multiplexor que selecciona la línea de bit apropiada y dirige su salida a los pines de salida del chip. Para almacenar un dato en el chip, se utiliza el mismo proceso salvo que ahora el dato a escribir se conduce hasta la línea de bit apropiada y se escribe en la celda de memoria seleccionada.

La mayoría de los chips de memoria producen mas de 1 bit en la salida. Esto se consigue implementando varias matrices de celdas de memoria, cada una de las cuales va proporcionando un bit de salida, o bien construyendo un multiplexor que selecciones las salidas de varias columnas (líneas de bit) a la vez y las lleve a los pines de salida del chip.

La velocidad de un chip de memoria viene dada por varios factores entre los que se encuentra la longitud de las filas y de las columnas de la matriz, y como se han construido las celdas de memoria. A mayor número de filas y de columnas mayor será la capacidad y la resistividad de las líneas y mayor será, por tanto, el tiempo necesario para propagar una señal a lo largo de estas líneas ya que sus longitudes aumentan. Debido a esto, muchos de los chips de memoria actuales se diseñan conteniendo pequeñas matrices de celdas de memoria que hagan que la longitud de las filas y de las columnas no sea excesivamente grande.

La técnica que se utilice para construir las celdas de memoria afecta notablemente a la velocidad del chip de memoria, ya que influye en la cantidad de corriente de que se puede disponer para hacer llegar la salida de la celda de memoria a las líneas de bit, lo cual determina el tiempo necesario para propagar la salida de la celda de memoria hasta el multiplexor. Como veremos en las siguientes dos secciones, las celdas de memoria SRAM pueden proporcionar mucha mas corriente que las celdas de DRAM, lo cual es una de las principales razones por las que las SRAM son mucho mas rápidas que las DRAM.

4.2.6 Memoria SRAM

La diferencia principal entre SRAM y memorias DRAM escriba en la forma en la que se construye sus celdas de memoria. El núcleo de una celda de memoria SRAM consiste en dos inversos conectados el uno a la salida del otro. Una vez que se haya colocado un valor en la celda de memoria, la estructura en anillo de los dos inversores mantendrá el valor de forma indefinida, ya que las entradas de cada inversor siempre serán opuestas. Este es el motivo por el que las memorias SRAM se denominan memorias RAM *estáticas;* los valores almacenados en la RAM permanecerán en ella siempre que el dispositivo este encendido. Las memorias DRAM, por su parte, van perdiendo sus valores almacenados con el paso del tiempo y ese es el motivo por el que se denominan memorias RAM *dinámicas.*

Para leer un dato de la celda de memoria, la línea de la fila se pone a uno, lo que hace que los dos transistores conecten la salida de los inversores a la línea de bit y a la línea de bit complementada. Estas señales pueden ser leídas por el multiplexor y enviadas a la salida del chip. Para escribir en una celada de SRAM debemos activar la línea de la fila y proporcionar los valores apropiados en la línea de bit y su complemento. Mientras que el

dispositivo que proporcione dichos valores tenga mayor fuerza que el inversor, el valor de la línea de bit prevalecerá sobre el valor originalmente en la celda de memoria y finalmente quedara almacenado en dicha celda una vez que se seleccione la fila. Para leer el dispositivo, la dirección a la que queremos acceder debe colocarse en los pines de dirección del chip y activarse la señal de habilitación de chip del dispositivo 1. Tras un cierto retardo, el chip de memoria coloca el contenido de la dirección pedida en su salida de datos. Las operaciones de escritura sin similares a las de lectura, salvo que ahora el procesador coloca también el dato a escribir en los pines de datos del dispositivo al mismo tiempo que la dirección de acceso, y que la señal de control de escritura se activa para indicar que se va a realizar una escritura.

4.2.7 Memoria DRAM

En lugar de usar un par de inversores, se utiliza un condensador para almacenar datos. Cuando se selecciona la fila, el condensador se conecta a la línea de bit, permitiendo que el valor almacenado en la celda se pueda leer midiendo el voltaje en la línea, o se pueda escribir colocando un nuevo valor de tensión en el condensador. Las SRAM requieren de muchos más dispositivos para implementar una celda de memoria. Cada inversor se implementa generalmente con dos transistores, dando un total de seis transistores por celda de memoria (aunque algunas implementaciones pueden usar más o menos transistores). Por su parte, la celda de memoria DRAM solo necesita un transistor y un condensador, lo que ocupa mucho menos espacio en el chip.

Las memorias SRAM son generalmente mucho más rápidas que las DRAM. En las SRAM, un dispositivo activo (el inversor) proporciona el valor almacenado en la celda de memoria a la línea de bit y su complemento. En las DRAM, el condensador se conecta a la línea de bit cuando se selecciona su fila, proporcionando una señal mucho mas débil que la suministrada por los inversores un una celda de memoria SRAM. De esta forma, una celda DRAM tarda más en transferir su salida a la línea de bit que una celda SRAM en las mismas condiciones.

4.2.8 Refresco de memoria DRAM

Las memorias DRAM se denominan memorias RAM dinámicas porque los valores almacenados en cada celda de memoria no son estables. Con el paso del tiempo, las corrientes de fuga provocan que la carga almacenada en el condensador se pierda. Para evitar esta perdida del dato almacenado en la celda DRAM, esta debe refrescarse. Esencialmente, una operación de refresco lee el contenido de cada celda de memoria en una fila completa de la matriz de celdas para posteriormente volver a escribir ese mismo valor en la celda, recuperándose así el valor original. Siempre que refresquemos cada fila de DRAM con la suficiente frecuencia como para que ningún condensador haya tenido tiempo de descargarse excesivamente y el hardware pueda recuperar su valor, la memoria DRAM podrá mantener su contenido indefinidamente. Uno de los parámetros que el fabricante de un chip de DRAM debe especificar es su *tiempo de refresco,* que indica la cantidad de tiempo que una fila puede mantener su valor almacenado sin necesidad de refresco.

4.2.9 Temporización de una DRAM

A diferencia de las SRAM, la entrada de dirección de una memoria RAM dinámica se divide en dos partes, la dirección de la fila y la de la columna, las cuales se envían a la DRAM por medio de dos operaciones distintas. Normalmente, los bits de mayor orden de una dirección de memoria se utilizan para codificar la dirección de la fila y los de menor orden de una dirección de memoria se utilizan para codificar la dirección de la fila y los de menor orden para especificar la columna. Como es de esperar, la dirección de la fila selecciona la fila de la matriz de celdas DRAM en la que se encuentra el dato que se va a leer, mientras que la dirección de la columna selecciona el bit o conjunto de bits dentro de esa fila.

La señal RAS (Row Address Strobe, “selección de la dirección de fila”) indica que se esta enviando la dirección de la fila y la señal CAS (Column Adress Strobe, “selección de la dirección de columna”) indica que se esta enviando la dirección de la columna. La cantidad de tiempo total para leer de la memoria DRAM es la suma entre el retardo entre RAS y CAS (retardo RAS-CAS) y el retardo entre CAS y la obtención del dato (retardo CAS-Dato).

La operación de escritura tiene una temporización similar, salvo que, en este caso, el dato a escribir generalmente se hace llevar a los pines de datos al mismo tiempo que la dirección de columna.

Al enviar la dirección a la DRAM en dos veces, reduciremos el numero de pines necesarios en el chip de DRAM para especificar la dirección, ya que podemos usar los mismos pines para las direcciones de fila y de columna. Dividir la dirección en dos partes no incrementa substancialmente el tiempo de acceso a la memoria DRAM ya que la dirección de fila selecciona la fila de la matriz de celdas cuyo contenido va a parar a las líneas de bit, mientras que la dirección de columna selecciona la línea de bit que finalmente se hará llegar a la salida. Por tanto, la dirección de columna no se necesita realmente en la DRAM hasta que las celdas no hayan transferido sus salidas a las líneas de bit, por lo que este retraso en proporcionar la dirección de columna tras dar el de la fila no incrementa el tiempo de acceso.

4.2.10 El modo pagina y las memorias DRAM actuales

Uno de los puntos débiles del diseño del chip de memoria, es que, en cada operación, se hacen llegar al multiplexor los contenidos de una fila completa de celdas de memoria, aunque solo el de una de ellas realmente llegara a la salida. Si se pudieran guardar los contenidos de la fila en las inmediaciones del multiplexor, seria posible leer otros bits de esa misma fila simplemente seleccionando una columna diferente de la DRAM, en lugar de tener que iniciar un nuevo ciclo completo del tipo RAS-CAS. A las memorias DRAM que permiten esto se les denomina memorias DRAM con modo página.

Las DRAM con modo página añaden un *match* entre la salida de las celdas de memoria y el multiplexor. Cuando se solicita una determinada fila de la memoria, todos los contenidos de dicha fila se guardan en el *match.* Esto permite que los siguientes accesos a distintas columnas dentro de la misma fila solo tengan que especificar la dirección de la nueva columna, reduciéndose de forma considerable la cantidad de tiempo necesario para leer un bloque de datos contiguos de la memoria DRAM.

En los últimos años, se han introducido en el mercado las memorias DRAM síncronas (SDRAM). Estos dispositivos son similares a las DRAM con modo pagina salvo que requieren una entrada de reloj (las otras DRAM son dispositivos asíncronos). La mayoría de las SDRAM están segmentadas, y muchas ofrecen modos de acceso que permiten que se pueda leer o escribir varias palabras consecutivas con un solo ciclo RAS-CAS, aumentándose aun más el ancho de banda.

4.3. ENTRADA/SALIDA (E/S)

Permiten conectar la computadora con el mundo exterior.
- Un módulo de E/S es un sistema que permite conectar un dispositivo (lo que realmente interacciona con el mundo exterior) y el computador.
- Sincroniza las diferentes velocidades de transmisión de datos entre la CPU (Rápida) y los dispositivos de E/S (lentos).
- Convierte los diferentes formatos utilizados por los dispositivos y la CPU y gestiona errores de transmisión

Los dispositivos de E/S son:

De Entrada: teclado, ratón, escáner, etc.
De salida: tarjeta de video, impresora, scaner, etc.
De Entrad/Salida: modem, unidades de disco, etc.
BUS: Conjunto de hilos al que acceden varios dispositivos para poder comunicarse entre ellos. Permite la comunicación entre la CPU, la memoria y el sistema de E/S.
Tiene diferentes tipos de hilos:
Dirección: Son los que transportan las direcciones.
Datos: Son los que transportan los datos.
Control: Líneas para gestionar el tráfico de información dentro del bus.

Un computador tiene una gran cantidad de dispositivos de entrada y salida. Desde un teclado hasta un monitor de vídeo, pasando por un disco duro, todos son dispositivos que la **CPU** necesita para obtener datos o para almacenar o mostrar resultados

El principal problema de estos dispositivos periféricos es una gran variedad. Muchos de ellos comunican a un operador humano con el computador y otros sirven de conexiones entre otros computadores.

De manera adicional cada dispositivo tiene una tecnología diferente, unas características de funcionamiento distintas, y unas necesidades de atención por la **CPU** distintas. Por ello cada dispositivo no se conecta directamente con la **CPU** mediante un Bus, sino que existe una interfaz que define como se van a entender el procesador y el controlador del periférico asociado para intercambiar datos.

Los módulos E/S son entidades que contienen uno o varios controles de periféricos

Buses: Físicamente, la mayor parte de los computadores personales y estaciones de trabajo tiene una estructura similar. La disposición usual es una caja metálica con una tarjeta grande de circuitos impresos en su base, llamada tarjeta madre o motherboard. La tarjeta madre contiene el chip de CPU, algunas ranuras en las que pueden insertarse módulos DIMM, y diversos chips de apoyo. Además, contiene un bus grabado a todo su

largo, y zócalos en los que pueden insertarse los conectores de arista de tarjetas de E/S. A veces hay dos buses, uno de alta velocidad (para las tarjetas de E/S más viejas). Casi todos los sistemas tienen dos o más buses. Cada dispositivo de E/S consta de dos partes: una que contiene casi todos los circuitos electrónicos, llamada controlador, y una que contiene el dispositivo de E/S propiamente dicho, como una unidad de disco El controlador suele estar contenido en una tarjeta que se inserta en una ranura desocupada, con excepción de los controladores que no son opcionales (como el del teclado), los cuales a veces se encuentran en la tarjeta madre. Aunque la pantalla (monitor) no es opcional, el controlador de video a veces se encuentra en una tarjeta insertable para que el usuario pueda escoger entre tarjetas con o sin aceleradores de gráficos, memoria adicional, etc. El controlador se conecta a su dispositivo con un cable unido a un conector en la parte trasera del gabinete. La tarea de un controlador es dominar su dispositivo de E/S y manejar su acceso al bus. Por ejemplo, cuando un programa quiere datos del disco, envía un comando al controlador del disco, que a su vez emite comandos de búsqueda y de otro tipo a la unidad de disco Una vez que se localizan la pista y el sector apropiados, la unidad de disco comienza a enviar los datos como un flujo de bits en serie al controlador. Corresponde a éste dividir el flujo de bits tan normalmente de una o más palabras. Un controlador que lleva datos de la memoria o los escribe en ella sin intervención de a CPU está efectuando acceso directo a la memoria, mejor conocido por sus iníciales en inglés, DMA Direct Memory Access. Una vez completada la transferencia, el controlador normalmente genera una interrupción que obliga a la CPU a dejar de ejecutar su programa actual y comenzar a ejecutar un procedimiento especial, llamado manejador de interrupciones, para verificar la presencia de errores, efectuar cualquier acción especial que se requiera, e informar al sistema operativo sobre la finalización de la E/S. Cuando el manejador de interrupciones concluye su intervención, la CPU continúa con el programa que se suspendió cuando ocurrió la interrupción.

El bus no sólo es utilizado por los controladores de E/S, sino también por la CPU para obtener instrucciones y datos. Qué sucede si la CPU y un controlador de E/S quieren usar el bus al mismo tiempo? La respuesta es que un chip llamado árbitro de bus decide quién tendrá el acceso. En general, los dispositivos de E/S tienen preferencia sobre la CPU, porque los discos y otros dispositivos móviles no pueden detenerse, y obligarlo a esperar podría causar la pérdida de datos. Si no se está efectuando E/S, la CPU puede aprovechar todos los ciclos de bus para hacer referencia a la memoria. Pero si también está funcionando un dispositivo de E/S, éste solicitará el bus cuando lo necesite, y se le otorgará. Este proceso se llama robo de ciclos y hace más lenta ala computadora.

Este diseño funcionaba bien en las primeras computadoras personales, porque todos los componentes estaban más o menos balanceados. Pero a medida que aumentó la rapidez de las CPU, memorias y dispositivos de E/S, surgió un problema: el bus ya no podía manejar la carga que se le presentaba. En un sistema cerrado, como una estación de trabajo de ingeniería, la solución era diseñar un bus nuevo más rápido para el siguiente modelo. Puesto que a nadie se le ocurría pasar dispositivos de E/S de un modelo viejo a uno nuevo, la estrategia funcionaba bien.

En cambio, en el mundo de las PC era común que la gente modernizara su CPU pero quisiera seguir usando su impresora, su escáner y su módem anteriores con el nuevo sistema. Surgieron tecnologías como la serie PS/2, luego vino la ISA Industry Standard Architecture, Arquitectura Estándar de la Industria. También, la mayoría de los fabricantes

de discos y dispositivos de E/S siguieron produciendo controladores para ese bus, e IBM se encontró en la peculiar situación de ser el único fabricante de PC que no era compatible con IBM. IBM se vio obligada a apoyar el bus ISA. ISA significa arquitectura del conjunto de instrucciones (Instruction Set Architecture. Luego surgió el bus EISA ISA Extendido. También surgió el bus PCI Interconexión de componentes periféricos. Intel diseñó este bus pero decidió colocar todas las patentes en el dominio público para animar a toda la industria a que lo adoptaran. Este bus se puede usar en muchas configuraciones.

4.3.1 Funciones de la interface de E/S

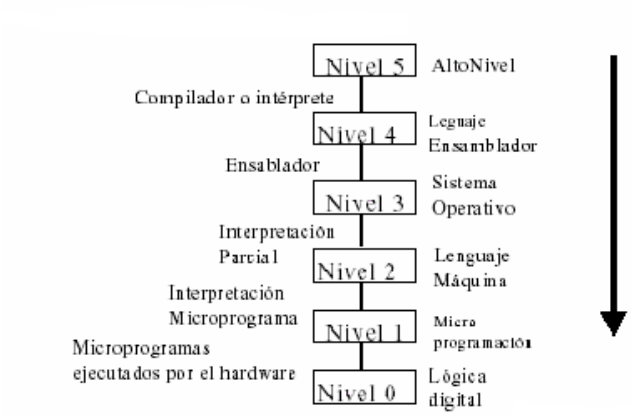
- Control y Cronometrado
- Comunicación de CPU
» Reconocimiento de dirección
» Decodificación de Comandos
» Transferencia de datos
» Informe de Estado

- Dispositivo de comunicación
» Comando
» Datos
» Información de estado

4.3.2 Controlador de dispositivos de E/S

- Los dispositivos de E/S deben operar a velocidades "mundo real"
- Más lento que el CPU
- Necesitan sincronizarse con el CPU

- ESTRUCTURA DE NIVELES DE UNA COMPUTADORA



4.4 El software Operativo: Es el conjunto de programas que controla el funcionamiento de los programas que se ejecutan y administra los recursos hardware, facilitando el uso del computador de la forma mas eficiente posible, dentro de este se incluye el sistema operativo.

El Sistema operativos: Es un programa o conjunto de programas de control que tiene por objeto facilitar el uso del computador y conseguir que este se utilice eficientemente, es un programa de control ya que se encarga de gestionar y asignar los recursos hardware a los usuarios, controla los programas de los usuarios y los dispositivos de entrada y salida.

Ejemplos:
DOS, UNIX, LINUX, OS 2, WINDOWS 95, WINDOWS 98, WINDOWS 2000, WINDOWS NT

El Software Aplicativo: Incluye programas relacionados con aplicaciones específicas como pueden ser: procesadores de textos, bibliotecas de programas para problemas estadísticos o de cálculo numérico, sistemas de administración de archivos, sistemas de administración de bases de datos, etc., también se incluyen los propios programas realizados por los usuarios.

Autoevaluación.

1. Obtener información acerca de las especificaciones del bus PCI y productos basados en el mismo en PCI Special Interest Group.

2. Dibuje y explique un diagrama de tiempos para una operación de escritura en un bus PCI.

3. Investigue cómo está organizada la memoria caché en el PENTIUM II y el PowerPC

4. Por qué las RAM han sido tradicionalmente organizadas en sólo un bit por chip mientras que las ROM están normalmente organizadas en múltiples bits por chip?

5. En casi todos los sistemas que tienen módulos de DMA, el acceso del módulo de DMA a memoria principal tiene más prioridad que el acceso de la CPU a memoria principal. Por qué?

6. Indique las razones por las que el tamaño de página en un sistema de memoria virtual no debe ser ni muy grande ni muy pequeño.

SEGUNDA UNIDAD
“Unidad Central de Procesamiento”

Aritmética del computador y representación interna de los datos
Unidad central de procesamiento.
Arquitecturas.

INTRODUCCIÓN

Los sistemas digitales actúan bajo el control de variables discretas, entendiendo por estas, las variables que pueden tomar un número finito de valores. Por ser de fácil realización los componentes físicos con dos estados diferenciados, es este el número de valores utilizado usualmente para dichas variables que, por lo tanto, son binarias.

Tanto si se utilizan en procesos de datos como en control industrial, los sistemas digitales han de realizar operaciones con números discretos. Los números pueden representarse en diversos sistemas de numeración, que se diferencian por su base. La base de un sistema de numeración es el número de símbolos distintos utilizados para la representación de las cantidades en el mismo. El sistema de numeración utilizado en la vida cotidiana es el de base diez, en el cual existen diez símbolos distintos del 0 al 9.

Por la razón expuesta el sistema de numeración mas utilizado en la realización de los sistemas digitales es el de base dos. o binario, en el cual existen solamente dos símbolos que son el 0 y el 1

Una computadora digital consiste en un sistema de procesadores interconectados, memorias y dispositivos de entrada/salida. A continuación se da una introducción hacia estos componentes y su interconexión.

Capítulo 1: Aritmética del Computador y representación interna de los datos

1.1. Sistemas de numeración y aritmética

1.1.1 REPRESENTACIÓN DE LOS NÚMEROS

N --> El conjunto de los números naturales
Z --> El conjunto de los números Enteros
Q --> El conjunto de los números Racionales
R --> El conjunto de los números Reales
C --> El conjunto de los números Complejos

El sistema de numeración es el decimal, sistema que utiliza los símbolos 0, 1, 2, 3, 4, 5, 6, 7,8 y 9

Un número entero en la numeración decimal puede constar de tantos símbolos como se desee.

Cada símbolo tiene un peso, es decir un valor por la posición que ocupa.

En el número natural 238, el 8 tiene el peso de las unidades, el 3 el de las decenas, es decir 3 grupos de 10 elementos, y el 2 tiene peso de centenas, es decir diez grupos de diez elementos cada uno. Cualquier número entero se puede poner como:

$238 = 2 \cdot 10^2 + 3 \cdot 10^1 + 8 \cdot 10^0$

A la numeración decimal se le denomina "*sistema base 10*".

Cada símbolo representa a cada dedo de las dos manos, que el hombre emplearía desde el principio de los tiempos para contar objetos.

1.1.2 Números binarios

En un ordenador, los elementos que almacenan información, son los que se indican en la figura nº2.

Sólo se pueden contar dos eventos: Hay energía almacenada, no hay energía almacenada. El sistema de numeración que representa este hay/no hay, todo/nada se denomina binario.

Al igual que en el sistema decimal un número binario está compuesto por combinaciones de estos dos símbolos, y cada símbolo tiene un peso, es decir un valor por la posición que ocupa.

En el número binario 01010011 se puede representar como

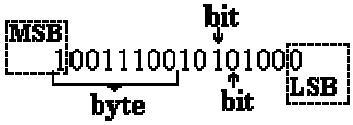
$01010011 = 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

A la numeración Binaria se la denomina "*sistema base 2*".

A los símbolos "**0**" y "**1**", se les llaman "**Bits**"

A una combinación de ocho símbolos binarios se le denomina "**Byte**".

En una combinación de bits, al que ocupa la posición más de a la izquierda se le llama "*bit más significativo*" o que tiene el mayor peso. Se le conoce por las siglas "**MSB**". Al bit que está mas a la derecha se le denomina "*bit menos significativo*", que es el que menos peso tiene. Por siglas se le denomina "**LSB**".



¿Cómo se guardan los números en los ordenadores?

1.1.3 Números naturales

Como los números pueden ser tan grandes como se quiera, tenemos que poner un límite al tamaño máximo del número que podemos guardar.

En la tabla adjunta se da el número natural máximo N_{max} que podemos guardar en función de los bytes (8bits) empleados

n=nº bytes,	**N_{max}=28n**
1	256
2	65.536
3	16.777.216
4	4.294.967.296
5	1.099.511.627.776
6	281.474.976.710.656
7	72.057.594.037.927.936
8	18.446.744.073.709.551.616

¿Como se convierten números naturales (código decimal) a números en código binario?

Se divide el número reiteradamente por dos hasta que el número que quede sea menor que dos. Durante el proceso de división si el dividendo es impar, el resto me quedará uno y el cociente será el número a dividir por dos otra vez.

Pongamos un ejemplo

nº Natural: 1074

cociente		**resto**
1074	= 537·2	**0**
537	= 268·2	**1**
268	= 134·2	**0**
134	= 67·2	**0**
67	= 33·2	**1**
33	= 16·2	**1**
16	= 8·2	**0**
8	= 4·2	**0**
4	= 2·2	**0**
2	= **1**·2	**0**

Por lo tanto $1074_{10} = 10000110010_2$

1.1.4 Números enteros

El número entero ha de tener un bit de signo, ya que puede ser negativo o positivo.

En la tabla adjunta se da el número entero máximo Z_{max} que podemos guardar en función de los bytes (8bits) empleados

n=nº bytes,	**Z_{max}= ± 28n -1**
1	± 128
2	± 32.768
3	± 8.388.608
4	± 2.147.483.648
5	± 549.755.813.888
6	± 140.737.488.355.328
7	± 36.028.797.018.963.968
8	± 9.223.372.036.854.775.808

¿Cómo pasar un número entero a binario?

Hay dos formas. La primera es pasar a binario el módulo del número entero (siempre es un número natural) y poner un bit con valor "0" a la izquierda del bit de mayor peso (MSB)si es positivo o un "1" si es negativo.

La segunda es mucho mejor porque nos facilita muchísimo las operaciones de sumar. Los pasos son los siguientes.

1. Se calcula el valor binario del módulo del entero
2. Si el entero es positivo, hacer lo especificado con los números naturales
3. Si es negativo

1. Se invierten los valores bit a bit (pasar de "0" a "1" , y de "1" a "0"). Esta operación se denomina calcular el "***complemento a uno***" de un número binario.
2. Se suma uno a resultado obtenido en el paso anterior. Se denomina calcular el "***complemento a dos***" del un número binario al proceso de calcular el *complemento a uno* y sumarle un "1"
3. Se pone un bit "1" a la izquierda del número binario obtenido

Pongamos un ejemplo

nº Entero: -384

cociente		**resto**
384	= 192·2	**0**
192	= 96·2	**0**
96	= 48·2	**0**
48	= 24·2	**0**
24	= 12·2	**0**
12	= 6·2	**0**
6	= 3·2	**0**
3	= **1**·2	**1**

Por lo tanto $384_{10} = 110000000_2$

Ejemplo

128	64	32	16	8	4	2	1	
27	26	25	24	23	22	21	20	
							1	= 1 x 20 = 1
						0		= 0 x 21 = 0
					1			= 1 x 22 = 4
				1				= 1 x 23 = 8
			0					= 0 x 24 = 0
		1						= 1 x 25 = 32
	0							= 0 x 26 = 0
1								= 1 x 27 =128
La conversión sería								=$173_{(10)}$ valor decimal

La solución es sumar los valores posiciónales positivos. La conversión decimal a binaria sería dividir el número por dos y el resto es el valor binario.

Para pasar de un número decimal a uno binario se debe dividir sucesivamente entre dos. El resultado se obtiene por el cociente final y los restos que van quedando en las sucesivas divisiones de derecha a izquierda:

173		2											
13		86		2									
1		06		43		2							
		0		03		21		2					
				1		01		10		2			
						1		**0**		5		2	
										1		2		2
												0		**1**

$173_{(10)} = 10101101_{(2)}$

Otra forma de hacerlo:

$173_{(10)}$

173/2 = 86 R=**1**

86/2 = 43 R=**0**

43/2 = 21 R=**1**

21/2 = 10 R=**1**

10/2 = 5 R=**0**

5/2 = 2 R=**1**

2/2 = **1** R=**0** **$10101101_{(2)} = 173_{(10)}$**

Dos números binarios se pueden sumar siguiendo este esquema:

0+0=0	0+1=1	1+1=10
	10110	
	+ 01101	
	100011	

Los caracteres se representan en código decimal, hexadecimal y binario. Cada carácter tiene una cadena binaria asignada y su correspondiente número decimal. Existen distintos códigos para representar cada carácter con un combinación de bits. Uno de estos códigos es el ASCII.

Muestra de algunos caracteres codificados, extraídos de una tabla de código ASCII:

Carácter	**Equivalente Binario**	**Equivalente Decimal**	**Carácter**	**Equivalente Binario**	**Equivalente Decimal**
espacio	0100000	32	a	1100001	97
. (punto)	0101110	46	b	1100010	98
0	0110000	48	c	1100011	99
1	0110001	49	d	1100100	100
2	0110010	50	e	1100101	101
3	0110011	51	f	1100110	102
4	0110100	52			
5	0110101	53			
6	0110110	54			
7	0110111	55			
8	0111000	56			
9	0111001	57			

Cálculo del ***complemento a uno***: 001111111

Cálculo del ***complemento a dos***: 1001111111 +1 =1010000000

El "1" más a la izquierda es el signo. Luego $-384_{10} = 1010000000_2$

Así mismo, para pasar de un número binario que representa a un entero a un número decimal se puede proceder simplemente como

$1010000000_2 = -512 + 0\cdot256 +1\cdot128 +0\cdot64 +0\cdot32 +0\cdot16 + 0\cdot8 +0\cdot4+0\cdot2 +0\cdot1 = -512 +128 = -384_{10}$

donde hemos dado al signo el valor negativo de su peso. Si hubiésemos representamos los números por dos bytes, por este procedimiento tendremos que 1074_{10} -384_{10} = 0000010000110010_2 + 1111111010000000_2

000000010000110010
111111111010000000

000000001010110010

Siendo $000000001010110010_2 = 2 +16+32+128+512= 690_{10}$

1.1.5 Números racionales

Un número racional es el cociente de dos números enteros. Cualquier número fraccionario tiene una parte decimal periódica.

Debemos emplear un conjunto de "n" bytes para guardar la parte entera, y otra parte de "m" bytes para guardar la parte decimal periódica. Como normalmente esto no puede ser posible, se fija el error máximo admisible al guardar el número decimal (*precisión*).

Hay que hacer constar que el signo se expresa en la parte entera, por lo que el número máximo será $Int(Q_{max})= \pm 28n -1$ y la máxima parte decimal será $Dec(Q_{max})= 28m$

Ya hemos visto como se pasa la parte entera (número entero) a su representación binaria. Vamos a ver como se pasa la parte decimal.

Si para la parte entera dividíamos por 2 el número entero, para la parte decimal multiplicaremos por dos. Quitaremos la parte entera y seguiremos multiplicando por dos hasta que la parte entera salga cero. Veamos un ejemplo.

Queremos pasar a binario el número 2,826 utilizando dos bytes para la parte entera y dos bytes para la decimal.

La parte entera es 2, que en binario es 0000000000000010 (con dos bytes)

La parte decimal es 0.826. Pasemos a binario

producto		**parte entera**
0.826·2	= 1,652	**1**
0,652·2	= 1,304	**1**
0,304·2	= 0,608	**0**
0,608·2	= 1,216	**1**
0,216·2	= 0,432	**0**
0,432·2	= 0,864	**0**
0,864·2	= 1,728	**1**
0,728·2	= 1,456	**1**
0,456·2	= 0,912	**0**
0,912·2	= 1,824	**1**
0,824·2	= 1,648	**1**
0,648·2	= 1,396	**1**
0,396·2	= 0,792	**0**
0.792·2	= 1,584	**1**
0,584·2	= 1,168	**1**
0,168·2	= 0,336	**0**

0.826_{10} =1101001101110110_2

El número se guardará como 0000000000000010_2 (parte entera) 1101001101110110_2 (parte decimal)

Si el número es negativo (por ejemplo el -2.826_{10}) haremos el complemento a uno, con el signo, de la parte entera y el complemento a dos sin el signo de la parte decimal.

El número se guardará como 1111111111111101_2 (parte entera) y 0010110010001010_2(parte decimal)

1.1.6 Aritmética binaria

En este apartado vamos a analizar las operaciones aritméticas binarias más comunes, es decir: suma, resta, multiplicación y división.

1.1.6.1 Adición Binaria

La tabla de adición binaria que se muestra más abajo es muy sencilla. En esta tabla los dos dígitos involucrados se denotan mediante X e Y. Ci es el acarreo de entrada de la adición precedente de menor orden.

Este es el clásico 1 + 1 = 2

Acarreo de entrada, Ci	1	1	1	0	1	1	1	
Dígito X	0	0	1	1	0	0	1	1
Dígito Y	0	1	0	1	0	1	0	1
Suma	1	10	10	10	1	10	10	10

Obsérvese la presencia del acarreo de salida (dígito en color rojo), el cual se genera en todas las adiciones de un bit cuando el resultado excede 1.El ejercicio anterior lo podemos explicar en notación decimal como 51+ 85 = 136.

Ejemplo:

Acarreo de entrada	0	1	1	1	1	0	0	0	1	
Dígito X	1	0	1	0	1	1	1	0	0	1
Dígito Y	0	0	1	1	0	1	0	1	0	1
Suma	1	1	1	0	0	0	1	1	1	0
Acarreo de salida	0	0	1	1	1	1	0	0	0	1

Cuando se llevan a cabo sumas, cada par de dígitos produce un resultado y un acarreo de salida si la suma excede a 1. Este acarreo se convierte en el acarreo de entrada para el siguiente dígito (de orden superior), como se muestra. por ejemplo, cuando x=1, y=1 y el acarreo de entrada es también 1, la suma es 3 (11 en binario).

El hardware que utilizan los computadores para implementar la suma consiste de unos módulos, llamados sumadores completos, que obtienen cada bit de salida

1 Acarreo ⟶ del bit menos

Ob 1 O O 1

+ Ob O 1 O 1

Ob 1 1 1 O

Según los bits de entrada correspondientes y el acarreo generado por el siguiente bit menos significativo anterior de la suma. La Figura muestra un circuito sumador de 8 bits.

1.1.6.2 Sustracción Binaria

La sustracción se puede discutir de una manera muy similar, haciendo uso de un préstamo y produciendo una diferencia. Sin embargo, en la práctica la sustracción se lleva a cabo mediante el mismo hardware que se utiliza para la adición, a través de la aritmética complementaria. En el caso binario, los números negativos se representan como el complemento a 2 del número binario positivo correspondiente. La sustracción de un número binario dado X de otro Y se lleva a cabo obteniendo el complemento a 2 de X para convertirlo en -X y sumarlo a Y. En este método, el dígito a la extrema izquierda es interpretado como el bit de signo (0 para el positivo, 1 para el negativo), el cual es tratado como cualquier otro bit, excepto que el acarreo de salida de la suma de dos bits de signo no se toma en cuenta.

El complemento a dos de un número binario se obtiene intercambiando los 1s y 0s del número original y sumar 1 al resultado.

Ejemplo:

Réstese 185_{10} a 230_{10} convirtiendo a binario y usando la aritmética de complemento a 2.

El número de dígitos binarios que se requieren para realizar el cálculo está determinado por el número mayor (incluyendo el resultado). En este caso, el número mayor es 230 y requiere 8 bits y uno adicional para el signo. De este modo,, el equivalente binario de 185 es 010111001. Obsérvese que los ceros de la izquierda no tienen ningún efecto sobre el valor numérico.

Se convierte este número al negativo correspondiente haciendo su complemento a 2:

Paso 1: Invertir los 1s y 0s.

010111001 - 101000110

Paso 2: Sumar 1.

```
  1 0 1 0 0 0 1 1 0
+               1
-----------------
  1 0 1 0 0 0 1 1 1
```

Después, empleando nueve dígitos, se convierte 230 a binario y se suma al resultado del paso 2.

```
+230 =   0 1 1 1 0 0 1 1 0
-185 =   1 0 1 0 0 0 1 1 1
         -----------------
         1 0 0 0 1 0 1 1 0 1
```

Ignorando el acarreo en el bit de signo (bit adicional a la izquierda) se obtiene 000101101, en el que se observa que el bit más a la izquierda es 0, lo que indica que el resultado es positivo. Verificación: 000101101, convertido a decimal, es +45.

Ejemplo 2 :

Réstese 230_{10} a 185_{10}, convirtiendo a binario y usando la aritmética de complemento a 2. El binario equivalente de 230 es 011100110, y su complemento a 2 se obtiene al invertir los 1s y 0s y sumando 1:

-230 = 100011010

Ahora se suma al número binario equivalente de 185:

```
-230 =   1 0 0 0 1 1 0 1 0
+185 =   0 1 0 1 1 1 0 0 1
         -----------------
         1 1 1 0 1 0 0 1 1
```

El bit más a la izquierda es un 1, indicando que el resultado es negativo. para obtener la magnitud del número, se hace el complemento a 2 del resultado, puesto que -(-X) es igual a X.

El equivalente en decimal es 45, el cual ya se determinó que es negativo.

1.1.6.3 Multiplicación: Se multiplican cada bit de uno de los factores, por todos los bit del otro factor según la tabla dada a continuación

bit multiplicando	bit del multiplicador	producto
0	0	0
0	1	0
1	0	0
1	1	1

Se multiplica de derecha a izquierda y el número resultante de la multiplicación de cada bit se suma al del anterior desplazando el peso de los bit uno a la izquierda. Pongamos un ejemplo:

$101_{10} \cdot 27_{10}$ $1100101_2 \cdot 0011011_2 =$ = 1100101 0011011 1100101 1100101 0000000 1100101 1100101 0000000 0000000 0101010100111 = 0101010100111_2 = 2727_{10}

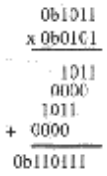
La multiplicación de enteros sin signo se hace de forma similar a la forma en que los humanos multiplicamos números decimales de varios dígitos. El multiplicando de la multiplicación se multiplica, por separado, por cada bit del multiplicador y los resultados se suman. En la multiplicación binaria, esto se ve simplificado por el hecho de que el resultado de multiplicar un número por un bit es o el mismo número o O,haciendo que el hardware sea menos complejo.

Para multiplica 11 (Ob1011) por 5 (ObO1O1). Primero, se multiplica Ob1O11 por cada bit de Ob1O11 para obtener los productos parciales mostrados en la figura. Luego, los productos parciales se suman para obtener el resultado final. Obsérvese que cada producto parcial sucesivo está desplazado una posición a la izquierda para tener en cuenta las distintas posiciones de los bits del multiplicador.

Un problema con la multiplicación de enteros es que para representar el producto de dos números de *n* bits se necesitan *2n* bits. Por ejemplo, para representar el producto de los dos números de 4 bits de la Figura se necesitan 6 bits. Muchas operaciones aritméticas pueden generar resultados que no se pueden representar con el mismo número de bits que sus entradas.

Esto se conoce como *desbordamiento o agotamiento.* En el caso de la multiplicación, el número de bits para que se produzca un desbordamiento es tan grande que los diseñadores de hardware toman medidas especiales para solucionar posibles problemas.

En algunos casos, los diseñadores ofrecen operaciones a parte para obtener los n bits más y menos significativos del resultado de una multiplicación de dos números de *n* bits. En otros, el sistema descarta los *n* bits más significativos, o los coloca en un registro de salida especial al que puede acceder el programador si fuera necesario.



1.1.6.4 División: Para dividir dos números naturales binarios se resta al dividendo el divisor, hasta que el resultado de la resta sea menor que el dividendo. El resultado es el resto y el número de veces que hemos efectuado la resta es el cociente. Pongamos un ejemplo. Queremos dividir 48 entre 13.

$48_{10} = 00110000_{2\ y}\ 13_{10} = 00001101_2$

El complemento a dos de 13_{10} es = 11110011_2

La operación está indicada en la tabla adjunta.

Dividendo	Divisor	Resto	Cociente
00110000_2	+ 11110011_2	= 00100011_2	00000001_2
00100011_2	+ 11110011_2	= 00010110_2	00000010_2
00010110_2	+ 11110011_2	= 00001001_2	00000011_2
00001001_2	+ 11110011_2	= $\mathbf{1}1111100_2$	

El último resto es negativo por lo que el cociente es tres (00000011_2) y el resto es 9 (00001001_2)

Si quisiéramos efectuar la operación 690_{10} -2.826_{10}= (parte entera){0000001010110010_2 + 1111111111111101_2 } (parte decimal) {0000000000000000_2 +0010110010001010_2}= 0,173980712890625

Parte entera	**Parte decimal**
0000001010110010 .	0000000000000000
1111111111111101 .	0010110010001010
---------- .	----------
0000001010101111 .	0010110010001010
687	**0,173980712890625**

El error cometido se debe a emplear dos bytes para guardar la información.

La división se puede implementar en un computador como restas sucesivas del dividendo con el divisor, y siendo el cociente el número de veces que el divisor se puede restar del dividendo antes de que el dividendo llegue a ser menor que el divisor. Por ejemplo, se puede dividir 15 entre 5 restando sucesivamente 5 a 15, obteniéndose 10,5, y como resultados intermedios. El cociente, 3, es el número de restas que hay que realizar antes de que el resultado intermedio sea menor que el dividendo.
Aunque se pudiera construir un hardware que implementase la división como restas sucesivas, sería poco práctico debido al número de restas necesarias. Por ejemplo, 231(uno de los números más grandes que se pueden representar con enteros sin signo de 32 bits) dividido entre 2 es 23°, 10 que supone que habría que realizar 23ºrestas para hacer esta división con restas sucesivas. Un sistema a 1 GHz, tardaría aproximadamente 1 s, mucho más que cualquier otra operación aritmética.

En lugar de esto, los diseñadores utilizan tablas de consulta para implementar la división.

Usando tablas pregeneradas, se obtienen de 2 a 4 bits del cociente en cada ciclo. Esto permite que se hagan divisiones de enteros de 32 o 64 bits en un número razonable de ciclos, aunque la división es usualmente la operación matemática básica más lenta de un computador.

1.1.7 Desbordamiento y agotamiento

La longitud de bits de un computador limita los números enteros máximo y mínimo que se pueden representar. Para enteros sin signo, con una longitud de n bits se pueden representar valores entre O y 2n- 1. Sin embargo, las operaciones aritméticas con números que se pueden representar con un número dado de bits pueden generar resultados que no se pueden representar con el mismo formato. Por ejemplo, sumar dos enteros de n bits puede generar un resultado superiora 2(2n- 1), que no.se puede representar con n bits, y es posible que se genere un resultado negativo restando dos números positivos, que tampoco se puede representar como un número sin signo de n bits.

Cuando una operación genera un resultado que no se puede expresar con el formato de sus operandos de entrada, se dice que se produce un desbordamiento o agotamiento. El desbordamiento se produce cuando el resultado de una operación es demasiado grande para representarlo con el formato y el agotamiento se produce cuando el valor absoluto del número es demasiado pequeño para representarlo con dicho formato. Los distintos sistemas manejan el desbordamiento y el agotamiento de diferentes formas. En algunos, cuando esto ocurre generan una señal de error. En otros, se reemplaza el resultado por el

valor más próximo que se pueda representar con el formato. Para números en coma flotante, el estándar IEEE especifica un conjunto de representaciones especiales que indican si se ha producido un desbordamiento o agotamiento.

1.1.8 Código Decimal Binario (BCD)

Si no consideramos tan importante el número de bits a emplear para guardar un dato numérico, podemos emplear el Código Decimal Binario, o ***BCD***. En este código cada símbolo de un número decimal (0-9)se sustituye por su correspondiente equivalencia en binario con cuatro bits. La equivalencia se da en la tabla

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Así el número 35 pasa a ser 00110101_{BCD} (3-5).

Es un código que permite una conversión muy fácil de decimal a binario y viceversa, pero que no permite fácilmente operaciones aritméticas, de tal forma que hay que trasladarlos a su equivalente binario para operar.

1.1.9 Código Octal

Es el sistema BCD pero quedándose en el 7 (*código de base 8*). Hay que pasar primero de decimal a octal, y se sustituye cada símbolo octal por su equivalente trío de bits para su paso a binario. Es un código de eficiencia máxima en bits, pero de eficiencia relativa en bytes, ya que en un bytes ocupan seis bits.

Es un código que permite aplicar con facilidad las operaciones aritméticas hasta ahora explicada.

1.1.10 Código Hexadecimal

Un código que si que permite realizar operaciones aritméticas con ellos es el hexadecimal. Su tabla es la indicada

Decimal	Hexadecimal
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Esta forma tiene la ventaja de que es mucho más sencillo para operaciones aritméticas. Un procedimiento gráfico para ver que es lo que hace este procedimiento está mostrado en la figura

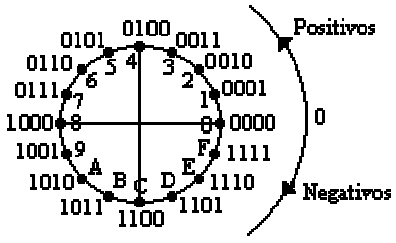


figura nº 3

Representa un ejemplo con números de cuatro bits. Los positivos giran en sentido antihorario, mientras que los negativos (complemento a dos de los positivos que se encuentran en ese lugar) giran en sentido horario.

Conversión decimal/hexadecimal.

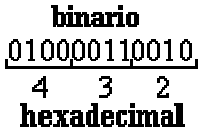
Veamos un ejemplo

cociente		**resto**
1074	= 67·16	**2**
67	= **4**·16	**3**

Se divide la parte entera del número por 16 tantas veces como el resto sea mayor que 16.

El número es $432_{16}=1074_{10}=010000110010_2$

Como se puede apreciar la conversión de hexadecimal a binario es inmediata. Vasta con representar cada dígito hexadecimal en binario como se indica en la figura



Un número negativo pasaría a ser el complemento a uno de cada uno de los dígitos más uno. Tomando dos bytes para la representación, tendremos

$-1074_{10}= 1111101111001110_2 = f\ b\ c\ e_{16}$

La operación 50_{10} -35_{10} sería

Por un lado $50_{10}= 32_{16} =00110010_2$

y por otro $35_{10} = 23_{16} = 00100011_2$

Siendo su complemento a dos de $00100011_2 =11011101_2 =DD_{16}$

donde el bit MSB representa el bit de signo.

$32_{16}- DD_{16} = 00110010_2 + 11011101_2 =00001111_2 = 0F_{16} = 15_{10}$

De la misma forma se operaría con números racionales.

1.1.11 Códigos digitales

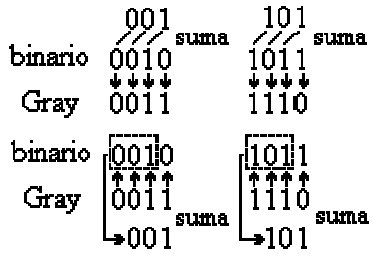
Aparte de los códigos anteriormente expuestos existen otros códigos de interés.

1.1.12 Código Gray:

Es un código binario sin pesos y no aritmético. La relación entre los símbolos decimales y las combinaciones de "1" y "0" está dada en la siguiente tabla

Decimal	Hexadecimal
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000

Es un código en el que el número de bits que cambian al aumentar o disminuir una unidad el número decimal es mínimo, es decir sólo uno. La conversión de grey a binario y de binario a grey es muy sencilla como se explica en la figura



1.1.8 Códigos alfanuméricos:

Son códigos binarios, sin peso que pueden representar letras y números

ASCII (American Standard Code for Informatión Interchange):

Cada símbolo es representado por un máximo de 7 bits. El número máximo de símbolos posibles es 27 = 128 caracteres.

EBCDIC (Extended Binary-Coded Decimal Interchange Code):

Cada símbolo es representado por un máximo de 8 bits. El número máximo de caracteres que se pueden representar es 28 = 256 caracteres.

1.2. ALU Unidad Aritmético Lógica

Es la parte del computador que realiza realmente las operaciones aritméticas y lógicas con los datos. El resto de los elementos del computador (Unidad de Control, registros, memoria y E/S) están principalmente para suministrar datos a la ALU, a fin de que ésta los procese, y para recuperar los resultados. Con la ALU llegamos al estudio de lo que puede considerarse el núcleo o esencia del computador.

Una ALU y todos los componentes electrónicos del computador, se basan en el uso de dispositivos lógicos digitales sencillos que pueden almacenar dígitos binarios y realizar operaciones lógicas booleanas elementales Los datos se presentan a la ALU en registros y en registros se almacenan los resultados de las operaciones producidos por la ALU. Estos registros son posiciones de memoria temporal internas al procesador que están conectados a la ALU. La ALU puede también activar indicadores (flags) como resultado de una operación. Los valores de los indicadores se almacenan también en otro registro dentro del procesador. La unidad de control proporciona las señales que gobiernan el funcionamiento de la ALU y la transferencia de datos dentro y fuera de la ALU.

- Ejecuta operaciones aritméticas y lógicas
- El acumulador es un registro especial

Casi todos los computadores contienen un solo circuito para obtener el AND, el OR y la suma de dos palabras de máquina. Por lo regular, un circuito de este tipo para palabras de n bits se construye con n circuitos idénticos para las posiciones de bit individuales. Una ALU puede calcular cualquiera de cuatro funciones: A and B, A or B, Negación B, o A + B.

1.3. Números en coma flotante

Los números en coma flotante se usan para representar cantidades que no se pueden representar como enteros, ya sea porque contienen valores fraccionario s o porque están fuera del rango representable dentro de la longitud de bits del sistema.

Prácticamente, todos los computadores de hoy en día utilizan la representación en coma flotante especificada en el estándar IEEE 754, en la que los números se representan con una mantisa y un exponente. De forma similar a la notación científica, el valor de un número en coma flotante es mantisa x 2exponente.

Esta representación permite representar un amplio rango de valores con relativamente pocos bits, incluyendo tanto valores fraccionarios como valores cuya magnitud es demasiado grande como para representarse como entero con el mismo número de bits.

Sin embargo, esto crea el problema de que muchos valores en el rango de la representación en coma flotante no se pueden representar exactamente, como la mayoría de los números reales que no se pueden representar con un número decimal utilizando un número prefijado de dígitos significativos. Cuando un cálculo da un valor que no se puede representar exactamente en coma flotante, el hardware tiene que redondear el resultado a un valor que se pueda representar exactamente. En el estándar IEEE 754, la forma implícita de redondear (llamada modo de redondeo) es el redondeo al más cercano.

En el redondeo al más cercano, los valores se aproximan al número representable más cercano, y los resultados que caen justamente en la mitad entre dos números representables se redondean de forma que el dígito menos significativo del resultado sea par. El estándar especifica otros modelos de redondeo que pueden seleccionar los programas, incluyendo el redondeo a O ,el redondeo a +infinito, y el redondeo a -infinito.

1.3.1 NAN Y Numeros desnormalizados

El estándar de coma flotante IEEE especifica varios patrones de bits para representar valores que no se pueden representar exactamente con el formato de coma flotante: cero, números desnormalizados y NaN (patrones de bits que no representan un número). El implícito de la mantisa de los números en coma flotante permite tener un bit adicional de precisión en la representación pero impide que se pueda representar el valor O exactamente, ya que una parte fraccionaria de la mantisa igual a O representa 1,0. Como representar O exactamente es muy importante en los cálculos numéricos, el estándar IEEE especifica que, cuando el exponente de un número en coma flotante es O, se supone que el bit principal de la mantisa es O. Por tanto, un número en coma flotante con una mantisa O y un exponente O representa al O exactamente. Esta convención también permite que se puedan representar números más cercanos a O que 1,0 x 2(1-sesgo), aunque tendrán menos bits de precisión que los números que se pueden representar con el implícito de la mantisa.

Los números en coma flotante (excepto el O) que tienen un exponente igual a O se conocen como números desnormalizados debido al O implícito de la parte entera de la mantisa. Esto contrasta con los números que tienen otros valores en el exponente, que tienen un 1 implícito en la parte entera de la mantisa y que se denominan números normalizados. Todos los números desnormalizados se supone que tienen un exponente (1 - sesgo), en lugar de (O- sesgo) que se generaría al restarle al valor del exponente el sesgo. Esto ofrece un hueco menor entre el número normalizado de menor magnitud y el número desnormalizado de mayor magnitud que se pueden representar con este formato.

El otro tipo de valor especial en el estándar de coma flotante IEEE es el NaN. NaN son las siglas de «no! a number» (patrón de bits que no representa un número), y los NaN se usan para representar condiciones de error como desbordamientos, división por O, etc. Cuando se da una de estas condiciones de error en una operación, el hardware genera un NaN corno resultado en lugar de una señal de excepción. Consecuentemente, las operaciones que tienen corno dato de entrada un NaN lo copian en la salida en lugar de realizar el cálculo habitual.

Un NaN se expresa poniendo el exponente del número en coma flotante al, menos cuando la mantisa se pone a O, en cuyo caso el número representa infinito. La existencia de NaN hace más fácil escribir programas que se puedan ejecutar en distintos

computadores, ya que los programadores pueden comprobar los resultados de cada cálculo buscando errores del programa, en lugar de fiarse de las funciones de excepción del sistema, que cambian bastante de un computador a otro. La Figura 2.8 resume la interpretación de los distintos valores del exponente y de la mantisa de un número en coma flotante.

1.3.2 Aritmética con números en coma flotante

Dadas las semejanzas entre la representación IEEE en coma flotante y la notación científica, no es sorprendente que las técnicas usadas en los computadores con aritmética en coma flotante –f sean muy similares a las técnicas usadas en la aritmética con números decimales expresados en notación científica. Un buen ejemplo de esto es la multiplicación en coma flotante.

Para multiplicar dos números en notación científica, se multiplican las mantisas de los números y se suman los exponentes. Si el resultado de multiplicar las mantisas es mayor que 10, el producto de las mantisas se desplaza de forma que haya exactamente un dígito no nulo a la izquierda de la coma decimal y se incrementa la suma de los exponentes en lo que sea necesario para mantener el valor del producto igual. Por ejemplo, para multiplicar 5 x 103por 2 x 106, se multiplican las mantisas (5 x 2 = 10) y se suman los exponentes (3 + 6 = 9) para tener un resultado inicial de 10 x 109. Como la mantisa es mayor que 10, la desplazamos a la izquierda una posición y añadimos un 1 al exponente para obtener el resultado final de 1 x 1010.

Los computadores de hoy multiplican números en coma flotante siguiendo un procedimiento similar, como se ve en la Figura 2.9. El primer paso es multiplicar las mantisas de los dos números, utilizando técnicas análogas a las usadas para multiplicar números decimales, y sumar sus exponentes. Los números en coma flotante IEEE utilizan una representación sesgada para los exponentes, de manera que sumar los exponentes de dos números en coma flotante es ligeramente más complicado que sumar dos enteros. Para obtener la suma de los exponentes, los exponentes de los dos números en coma flotante se tratan como enteros y se suman y después se resta el valor del sesgo al resultado. Esto da la representación sesgada correcta de la suma de los dos exponentes.

Una vez se hayan multiplicado las mantisas, puede ser necesario desplazar el resultado para que quede sólo 1 bit a la izquierda de la coma binaria (es decir, para que quede en la forma 1,xxxxx), e incrementar la suma de los exponentes para que el valor de la mantisa x 2exponente permanezca igual. Puede que también haya que redondear todo producto de las mantisas para que se ajuste al número de bits del campo mantisa, ya que el producto de dos mantisas de n bits requiere más de 2n bits para representarlas exactamente. Una vez se haya desplazado y redondeado la mantisa, la representación final del producto se obtiene ensamblando el producto de las mantisas y la suma de los exponentes.

Autoevaluación.

1. Convertir los siguientes números binarios en sus equivalentes decimales

a)00001110 b) 11100000 c)10000011 d)10011010

2. Convertir los siguientes números binarios en sus equivalentes hexadecimales

a)11110010 b) 11011001 c)111110.000011 d)10001.11111

3. Convertir los siguientes números decimales en sus correspondientes binarios:

a)32 c)200 c)170 d)250

4. Convertir los siguientes números decimales con signo a sus equivalentes en complemento a dos, con 8 bits

a)13 c)110 c)-25 d)-90

5. Convertir los siguientes números en complemento a 2 a sus correspondientes decimales:

a)01110000 c)00011111 c)11011001 d)11001000

6. Se van a emplear 4 bytes para guardar números Racionales en un sistema computacional. 2 Bytes se emplearán para la parte entera y 2 bytes para la parte decimal. Indicar en hexadecimal lo que guardaría si tuviéramos los números

a)40.875 c)999.125 c)5039 d)180.78515

7. Realizar las siguientes operaciones con los equivalentes números binarios de 2 bytes:

a) 1130 - 623 b)24,25 - 12,725 c)74 · 13 d)74/13

8. Convertir los siguientes números binarios puros a sus equivalentes en código Gray:

a)0110 b)10100 c)10101 d)10110

9 Convertir los siguientes números BCD a sus equivalentes decimales

a)10010000 b) 11111111 c)0111.0011 d)01100001.00000101

10. Convertir los siguientes números BCD a sus equivalentes decimales

a)10010000 b) 11111111 c)0111.0011 d)01100001.00000101

11. Realizar operaciones aritméticas como sumas y restas utilizando complemento a dos.

12. Realizar operaciones aritméticas expresando en formato de coma flotante IEEE de 32 bits

1.3. Realizar operaciones aritméticas que permitan convertir números del sistema octal a notación hexadecimal.

Capítulo 2: Estructura y Funcionamiento de la CPU

2.1. Organización del Procesador

La CPU Unidad Central de Procesamiento es el cerebro de la computadora. Su función es ejecutar programas almacenados en la memoria principal buscando sus instrucciones y examinándolas para después ejecutarlas una tras otra. Los componentes están conectados por un bus, que es una colección de alambres paralelos para transmitir direcciones, datos y señales de control. Los buses pueden ser externos a la CPU, cuando la conectan a la memoria y a los dispositivos de E/S, pero también internos. La CPU se compone de varias partes como ya hemos visto. La unidad de control se encarga de buscar instrucciones de la memoria principal y determinar su tipo. La unidad aritmética y lógica realiza operaciones como suma y AND booleano necesarias para ejecutar las instrucciones.

La CPU también contiene una memoria pequeña y de alta velocidad que sirve para almacenar resultados temporales y cierta información de control. Esta memoria se compone de varios registros, cada uno de los cuales tiene cierto tamaño de función. Por lo regular, todos los registros tienen el mismo tamaño. Cada registro puede contener un número, hasta algún máximo determinado por el tamaño del registro. Los registros pueden leerse y escribirse a alta velocidad porque están dentro de la CPU.

El registro más importante es el contador de programa PC, Program Counter, que apunta a la siguiente instrucción que debe buscarse para ejecutarse. El nombre es un tanto engañoso porque no tiene nada que ver con contar, pero es un término de uso universal. Otro registro importante es el registro de instrucciones IR, que contiene la instrucción que se está ejecutando. Casi todas las computadoras tienen varios registros más, algunos de propósito general y otros para fines específicos.

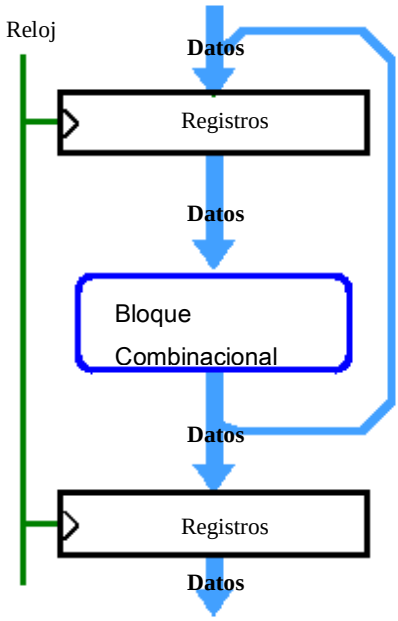
2.1.1 Las Máquinas síncronas

De nuevo, con un muy pocas excepciones - un grupo de investigación y un número pequeño de sistemas comerciales - la mayoría de las máquinas es hoy día síncronas, es decir, que se controlan por un reloj.

2.1.2 Caminos de datos

Los registros y la lógica de bloques combinacional es alternantes a lo largo de los caminos de datos, a través de la máquina. Los datos se adelantan de un registro al próximo en cada ciclo global del reloj: Cuando el borde del reloj cronometra los nuevos datos en un registro, su rendimiento actual (procesado atravesando el bloque combinacional) es puesto en el próximo registro en la tubería. Los registros son flip-flops, maestro - esclavo que permiten que la entrada sea aislada desde la salida, asegurando una transferencia "limpia" de nuevos datos en el registro. (Algunas maquinas de actuación muy alta, eg, DEC Alfa, usan seguros dinámicos aquí para reducir la propagación de retardos, cf Dobberpuhl et al.) En una máquina síncrona, el posible retraso de propagación más lento, el tpdmax, a través de cualquier bloque del combinacional, debe ser menos del tiempo del ciclo de reloj más pequeño, t_{cyc} - por otra parte ocurrirá un riesgo de la tubería y se cronometrarán datos de una fase anterior en un nuevo registro.

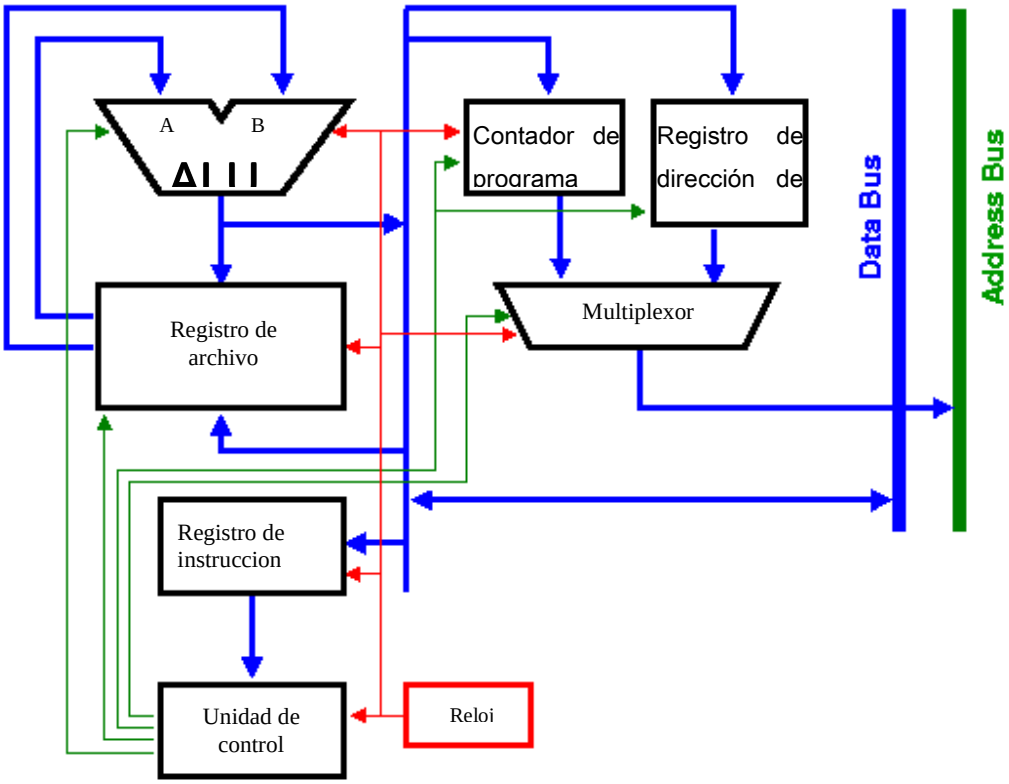
Si el t_{cyc} <el t_{pd} para cualquier operación en cualquier fase de la tubería, el borde del reloj llegará al registro antes de que los datos se hayan propagado a través del bloque combinacional.



Puede haber también claro, la regeneración loops - en que la salida de la fase actual se regenera y asegura en el mismo registro: una máquina de estado convencional. Esta clase de lógica se usa para determinar la próxima operación (ie el próximo Microcódigo, palabra o la siguiente dirección para el propósito de la sección).

2.1.3 Estructura básica del Procesador

Aquí consideraremos la estructura básica de un procesador simple. Examinaremos el flujo de datos a través de un procesador simple e identificaremos los cuellos de botella para entender lo que ha guiado el diseño de procesadores más complejos.



en un

2.1.4 ALU

La Unidad Lógica aritmética - este circuito toma dos operandos en las entradas (A y B) y produce un resultado en la salida (Y). Las operaciones normalmente incluirán, como mínimo:

- agregar, substraer
- y, o, no
- cambio derecho, cambio izquierdo

La ALU en los procesadores más complejos ejecutará muchas más instrucciones.

2.2. Organización de los registros

- El registro de Archivo

Un conjunto de ubicaciones de almacenamiento (los registros) para guardar los resultados temporales. Las máquinas tempranas tenían simplemente un registro - normalmente condicionado a un acumulador. Los procesadores de RISC modernos tendrán 32 registros por lo menos.

- Registro de instrucción

La instrucción que se esta ejecutando actualmente por el procesador se guarda aquí.

- Unidad de Control

La unidad de control decodifica las instrucciones en el registro de instrucción y los conjuntos de señales que controlan las operaciones de la mayoría de las otras unidades del procesador. Por ejemplo, el código del operación (el opcode) en la instrucción, se usa para determinar el conjunto de señales de control para el ALU, que determina las operaciones qué funcionamiento se usarán (+,-,^,v,~,shift,etc).

- El reloj

La inmensa mayoría de procesadores es síncrona, es decir, que utilizan un reloj de señal, para determinar cuándo capturar los próximos datos de palabra y realizar una operación en él. Globalmente en un procesador síncrono, un reloj común necesita ser conectado a cada unidad en el procesador.

- El contador del programa

El contador del programa retiene la dirección de memoria de la próxima instrucción a ser ejecutada. Se pone al día en todos los ciclos de instrucción para apuntar la próxima instrucción en el programa. (El control para el manejo de instrucciones de una sección – El cambio el contador del programa por un incremento simple - se ha omitido de este diagrama por claridad. Se examinarán extensivamente las ramas de instrucciones y su efecto en la ejecución del programa y eficacia después).

- El Registro de Dirección de memoria

Este registro está cargado con la dirección de los datos de la próxima palabra en ser captada de, o almacenada en la memoria principal.

- Adress Bus

Este Bus se usa para transferir las direcciones a la memoria y memoria-construccion periférica. Se maneja por el procesador que actúa como un maestro del Bus.

- Bus de datos

Este Bus lleva los datos hacia y desde el procesador, memoria y periféricos. Se manejará por la fuente de datos, memoria o dispositivos periféricos.

- Bus multiplexor

Por necesidad, los procesadores de alta actuación proporcionan direcciones separadas y Buses de datos. Para limitar dispositivos de contar pines y complejidad de Bus, algunos procesadores simples multiplican direcciones y datos hacia el mismo Bus: Naturalmente esto tiene un daño adverso en la actuación.

2.3. El ciclo de instrucción

- Ejecución de instrucciones

Examinemos los pasos en la ejecución de una simple instrucción de captación de memoria.

$101c_{16}$: lw $1,0($2)

Esta instrucción dice al procesador tomar la dirección almacenada en el registro 2, agregarle 0 y cargar la palabra encontrada a esa dirección en la memoria principal en el registro 1.

> En este, y la mayoría de los siguientes ejemplos, usaremos el conjunto de instrucciones MIPS.
> Escogimos este porque:
> - Es simple,
> - Existe en un rango extensamente disponible de máquinas producido por SGI.
> - Hay un simulador de dominio público para maquinas MIPS, que usaremos para algunos estudios de desarrollo.

Como la próxima instrucción a ser ejecutada (nuestra instrucción lw) es la dirección de memoria $101c_{16}$, el contador del programa contiene 101c.

> Por conveniencia, la mayoría de los números - sobre todo las direcciones de memoria y contenidos de instrucción - se expresará en el hexadecimal. Cuando se estén discutiendo ordenes de magnitud y actuación, se usarán los números decimales: esto generalmente es obvio en el contexto, el uso de notaciones de exponente, eg 5 x 1012.

Pasos de ejecución

1. la unidad de control pone el multiplexor para conducir el PC hacia el Bus de dirección.

2. la unidad de memoria responde poniendo $8c410000_{16}$ - el lw $1,0($2) la instrucción como un código para un procesador MIPS - en el Bus de datos de dónde es asegurado en el registro de la instrucción.

3. la unidad de control decodifica la instrucción, la reconoce como una instrucción de carga de memoria y dirige el archivo del registro para conducir los contenidos de registro

2 hacia la entrada A del ALU y el valor 0 hacia la entrada B. Al mismo tiempo, le dice al ALU que agregue sus entradas.

4. La salida del ALU es asegurada en la MAR. El controlador asegura que este valor se dirige hacia el Bus de direcciones poniendo el multiplexor.

5. cuando la memoria responde con el valor buscado, se captura en el Bus interno de datos y asegurado en el registro 1 del archivo del registro.

6. el contador del programa se pone al día para apuntar ahora la próxima instrucción y el ciclo puede empezar de nuevo.

Como otro ejemplo, se puede tomar la siguiente instrucción es una instrucción de adicion:

1020_{16}: agregar $1,$3,$4
Esta instrucción dice al procesador agregar los contenidos de registros 3 y 4 y poner el resultado en el registro 1.

1. la unidad de control pone el multiplexor para conducir el PC hacia el Bus de direcciones.

2. la unidad de memoria responde poniendo 00232020_{16} – pusimos el código de instrucción agregar $1,$3,$4 - en el Bus de datos de dónde es asegurado en el registro de instrucción.

3. la unidad de control decodifica la instrucción, la reconoce como una instrucción aritmética y dirige el archivo del registro para conducir los contenidos del registro 1 hacia la entrada A del ALU y los contenidos del registro 3 hacia la entrada B. Al mismo tiempo, le dice al ALU que agregue sus entradas.

4. La salida del ALU es asegurada en el archivo del registro en el registro de dirección
5. el contador del programa se pone al día para apuntar la próxima instrucción ahora.

- Los pipelines o Tuberías

Una sucesión de elementos de almacenados alternamente (registros o seguros) y bloques combinacionales, constituyendo un camino de datos a través de la computadora.

- Contador Del Programa

Un registro o ubicación de memoria que retiene la dirección de la próxima instrucción a ser ejecutada.

- Síncrono (sistema/maquina).

Una computadora en que las instrucciones y datos se mueven de una fase de la tubería a la próxima bajo el mando de un solo (global) reloj.

2.4 Manejo de interrupciones

Permiten:

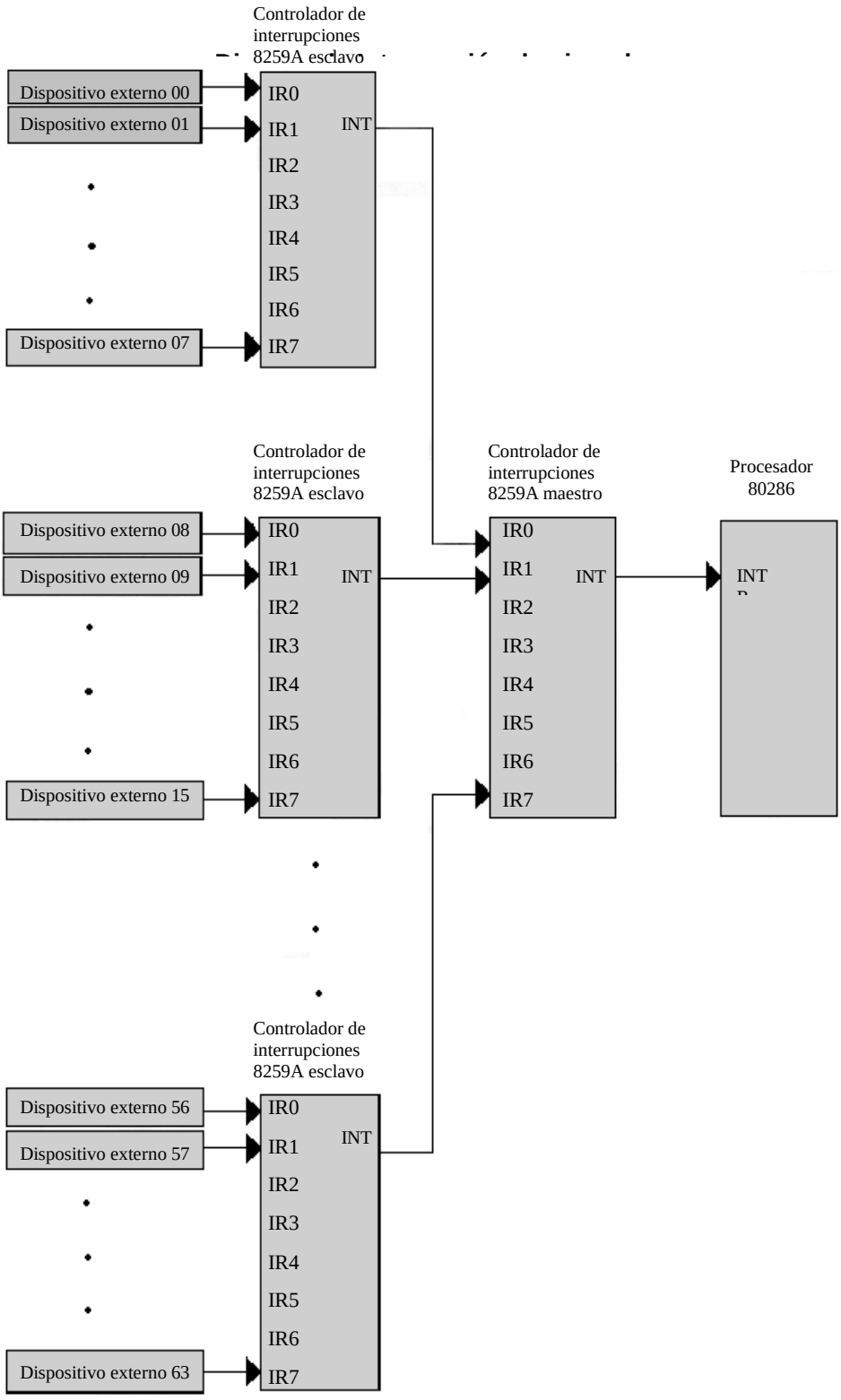
- Acceder al procesador para ejecutar los programas
- Cuando un dispositivo de E/S necesita atención, es acertada una señal de interrupción que es una entrada a CPU.
- CPU debe hacer:
» Reconocer que una interrupción ha ocurrido. Si mas de una interrupción es posible, debe determinar qué dispositivo generó la interrupción.
» Detener el programa que esta ejecutándose actualmente e iniciar el programa asociado con la interrupción. Normalmente conocido como negociante de la interrupción o rutina de servicio de interrupción.
» Cuando el negociante de la interrupción está acabado, continúa la ejecución del programa que fue interrumpido.
- Una instrucción especial (el retorno de la interrupción) se pone al final del negociante de la interrupción y causas el estado del programa interrumpido será restaurado.
- Los programas Interrumpidos tienen a menudo los datos guardados en los registros.
- negociantes Interrumpidos deben salvar y restaurar los datos del programa.
- La Mayoría de los procesadores tiene instrucciones especiales para mover los datos entre los registros y la pila.

2.4.1 Interrupciones múltiples

- Más de un dispositivo puede generar interrupciones.
- Cómo determinar el dispositivo requerido:
» Múltiples líneas de interrupción
» software Poll
- Cuando un dispositivo genera una interrupción, puede poner un Bit en su registro de estado para indicar que quiere el servicio. El CPU puede registrar los votos de los dispositivos para ver quienes quieren el servicio.
» Cadena de la margarita
- CPU envía un reconocimiento de la interrupción
- El signo se propaga a través de cada dispositivo hasta que alcance el dispositivo correcto
- El dispositivo de interrupción E/S pone la dirección de su interrupción handler hacia el bus de datos.
- La dirección es llamada a menudo vector

» Bus de arbitraje
- El dispositivo de E/S debe ganar el control del Bus
- Afirmar la línea de interrupción
- Poner el vector en el Bus de datos cuando el CPU lo reconozca

- El procesador puede desactivar las interrupciones
- Las Interrupciones pueden tener niveles de prioridad


Controlador de interrupciones 8259A esclavo
Dispositivo externo 00
Dispositivo externo 01
Dispositivo externo 07
IR0
IR1
IR2
IR3
IR4
IR5
IR6
IR7
INT
Controlador de interrupciones 8259A esclavo
Dispositivo externo 08
Dispositivo externo 09
Dispositivo externo 15
Controlador de interrupciones 8259A maestro
Procesador 80286
Controlador de interrupciones 8259A esclavo
Dispositivo externo 56
Dispositivo externo 57
Dispositivo externo 63

Casi todas las instrucciones pueden dividirse en una de dos categorías: registro-memoria o registro-registro. Las instrucciones registro memoria permiten buscar palabras de la memoria a los registros, donde pueden utilizarse como entradas de la ALU en instrucciones subsecuentes. Una palabra podría ser un entero. Otras instrucciones registro memoria permiten almacenar el contenido de un registro en la memoria.

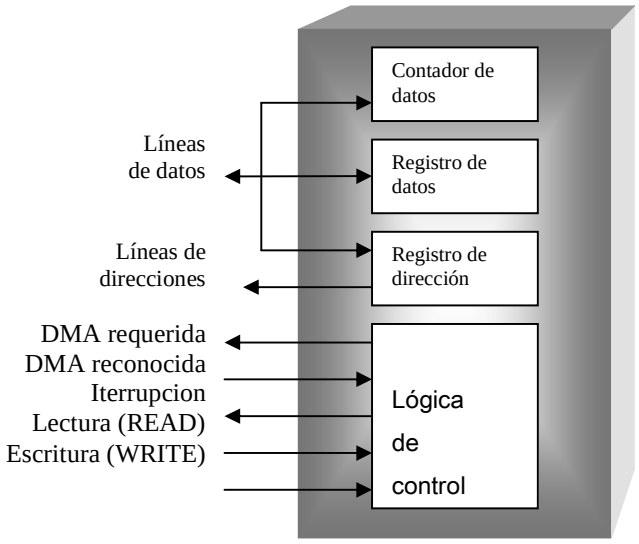
La otra clase de instrucción es la de registro registro. Una instrucción de este tipo busca dos operandos de los registros, los coloca en los registro de entrada de la ALU, realiza alguna operación con ellos y coloca el resultado en uno de los registros. El proceso de hacer pasar dos operandos por la ALU y almacenar el resultado se llama ciclo del camino de datos y es el corazón de casi todas las CPU. EN gran medida, este ciclo define lo que la máquina puede hacer. Cuanto más rápido es el ciclo del camino de datos, más rápidamente opera la máquina.

2.4.2 El Acceso directo de Memoria

- El módulo de DMA es capaz de imitar el CPU para el control del Bus
- la Interface de E/S tiene una línea de control que actúa como un maestro de Bus
- La Transferencia entre el dispositivo y la memoria sin la intervención del procesador
- CPU comienza a transferir informando

» La dirección del dispositivo de E/S involucrado
» La dirección de arranque en la memoria para leer desde o escribir hacia.
» El número de palabras para ser leídas o escritas
» Si leer o escribir se requiere

- El módulo DMA controla la transferencia incrementando la dirección de memoria y cuenta el decrecimiento
- CPU sólo está involucrada al principio y final del traslado



- En los sistemas simples, el procesador espera mientras la transferencia ocurre
- En los sistemas más complejos, el dispositivo de DMA "robo de ciclo" utiliza el Bus preferiblemente cuando el procesador esta en conflicto.
- Requiere compleja lógica de arbitraje
- El controlador de DMA puede ser compartido por varios dispositivos

2.4.3 El Puerto paralelo

- Igual que los dispositivos de buses-compatibles requieren una Interface paralela

» Proporciona buffering de datos
» Proporciona mando y sincronización

- el puerto Paralelo Típico

2.5. Procesador PENTIUM

En 1968 Robert Noyce, inventor del circuito integrado de silicio; Gordon Moore, cuya ley ya es famosa, y Arthur Rock, un inversionista de San Francisco, formaron la Intel Corporation para fabricar chips de memoria. En su primer año de operación, Intel sólo vendió 3000 dólares de chips y el negocio ha prosperado desde entonces.

Intel ha sido el número uno de los fabricantes de microprocesadores durante décadas, una posición que no parece probable que abandone. La evolución de su microprocesador más representativo es un buen indicador de la evolución de la tecnología de computadores en general. Aunque el Pentium es ahora la estrella de la línea de productos de Intel, sus procesadores P6, P7 han sido presentados. Conforme los microprocesadores se han hecho más rápidos y complejos, Intel ha mejorado el ritmo. Pero para el Pentium, el intervalo generacional se redujo a tres años. Y para sus chips P6 y P7, Intel espera mantener a sus rivales a raya.

Todos los chips Intel son compatibles con sus predecesores hasta el 8086. Un Pentium II puede ejecutar programas del 8086 sin modificación. Esta compatibilidad siempre ha sido un requisito de diseño para INTEL, a fin de que los usuarios puedan mantener su inversión en software existente. Desde luego, el Pentium II es 250 veces más complejo que el 8086, así que puede hacer una gran cantidad de cosas que el 8086 nunca pudo hacer. Estas extensiones incrementales han dado pie a una arquitectura menos elegante de lo que podría haber sido si alguien hubiera entregado a los arquitectos del Pentium II 7.5 millones de transistores e instrucciones para comenzar otra vez desde el principio.

El Pentium II es un descendiente directo de la CPU 8088 que se usó en la IBM PC original. Desde el punto de vista del software, el Pentium II es una máquina de 32 bits completa. Tiene la misma ISA en el nivel de usuario que los chips 80386, 80486, Pentium y Pentium Pro, incluidos los mismos registros, las mismas instrucciones y una implementación completa en el chip del estándar de punto flotante IEEE 754. Desde el punto de vista del hardware, el PENTIUM II es algo más porque puede direccionar 64 GB de memoria física y puede transferir datos de y a la memoria en unidades de 64 bits. Aunque el programador no puede observar estas transferencias de 64 bits, hacen que la máquina sea más rápida que una máquina de 32 bits pura.

2.6. Procesador POWER PC

En 1975, el proyecto de minicomputador 801 de IBM fue el primero en muchos de los conceptos de arquitectura usados en sistemas RISC. El 801, junto con el procesador RISC I de Berkeley, comenzó con el movimiento RISC. El 801, sin embargo, era simplemente un prototipo que intentaba demostrar los conceptos de diseño. El éxito del proyecto 801 permitió a IBM desarrollar una estación de trabajo RISC comercial, el RT PC. El RT PC, introducción tuvo éxito comercial, y tuvo muchos rivales con prestaciones comparables o mejores. En 1990, IBM produjo un tercer sistema, que incorporaba las lecciones aprendidas con el 801 y el RT PC. El IBM RISC System/6000 era una máquina superescalar RISC comercializada como una estación de trabajo de altas prestaciones; poco después de su introducción, IBM comenzó a llamarla arquitectura POWER.

Como siguiente paso IBM se alió con MOTOROLA, que había desarrollado las series 68000 de microprocesadores y APPLE, que usaba el chip de Motorola en sus computadores Macintosh. El resultado es una serie de máquinas que implementan la arquitectura PowerPC. Esta arquitectura deriva de la arquitectura Power. Se hicieron cambios para añadir características claves que no estaban y para permitir una implementación más eficiente, eliminando algunas características claves que no estaban y para permitir una implementación más eficiente, eliminando algunas instrucciones y relajando la especificación, para eliminar casos especialmente problemáticos. La arquitectura PowerPC resultante es un sistema RISC superescalar. El POWERPC se usa en millones de máquinas Apple Macintosh y en sistemas con microprocesadores embebidos. Como ejemplo de este último hecho puede citarse la familia IBM de chips de gestión de redes, que se utilizan embebidos en numerosos equipos que proporcionan la gestión del acceso a red usual a los usuarios con sistemas de distintos fabricantes.

Autoevaluación

1. Realizar ejercicios de diagramas de tiempos y diagramas de estado que muestren la estructura y funcionamiento de la CPU

2. Realizar cuadro comparativo de las diferentes tecnologías de procesadores vistas en el capítulo

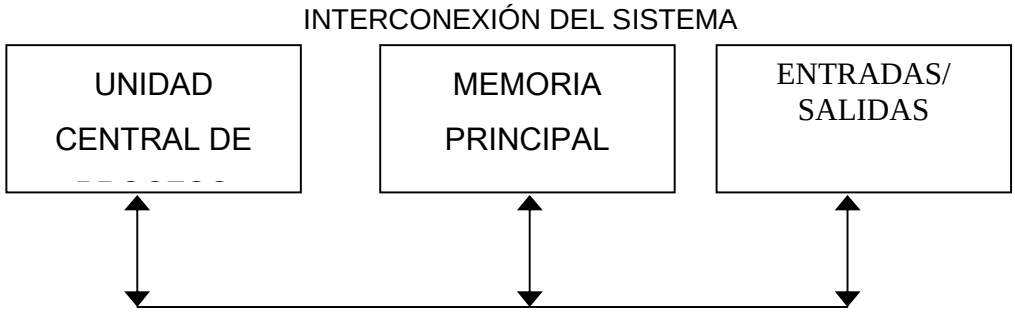
Capítulo 3: Arquitecturas

Componentes

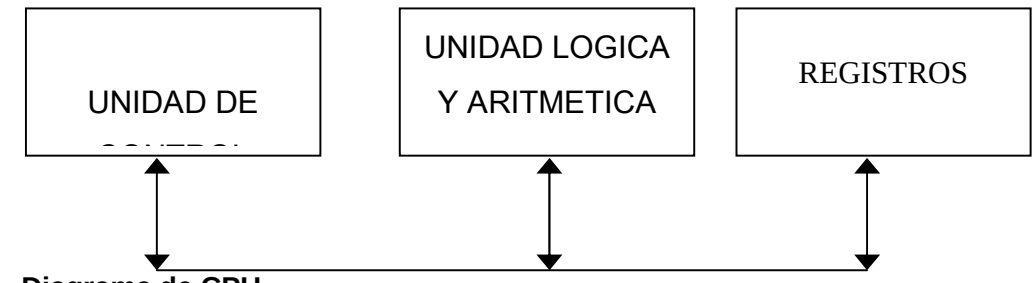
- La Maquina de Von Newman

Tres conceptos importantes:
» Los datos e instrucciones se guardan en una sola memoria leer-escribir.
» Los contenidos de la memoria son dirigidos según la situación, sin tener en cuenta el tipo de datos contenido.
» La ejecución ocurre en un modo secuencial, a menos que sea explícitamente alterado de una instrucción a la siguiente.

- Diagrama de bloque



- Diagrama de bloque del computador



- Diagrama de CPU

- Unidad de control

- LA MEMORIA

» Guarda instrucciones y datos

- ENTRADAS/SALIDAS

» Llamadas periféricos
» Utiliza instrucciones y datos de entrada y salida

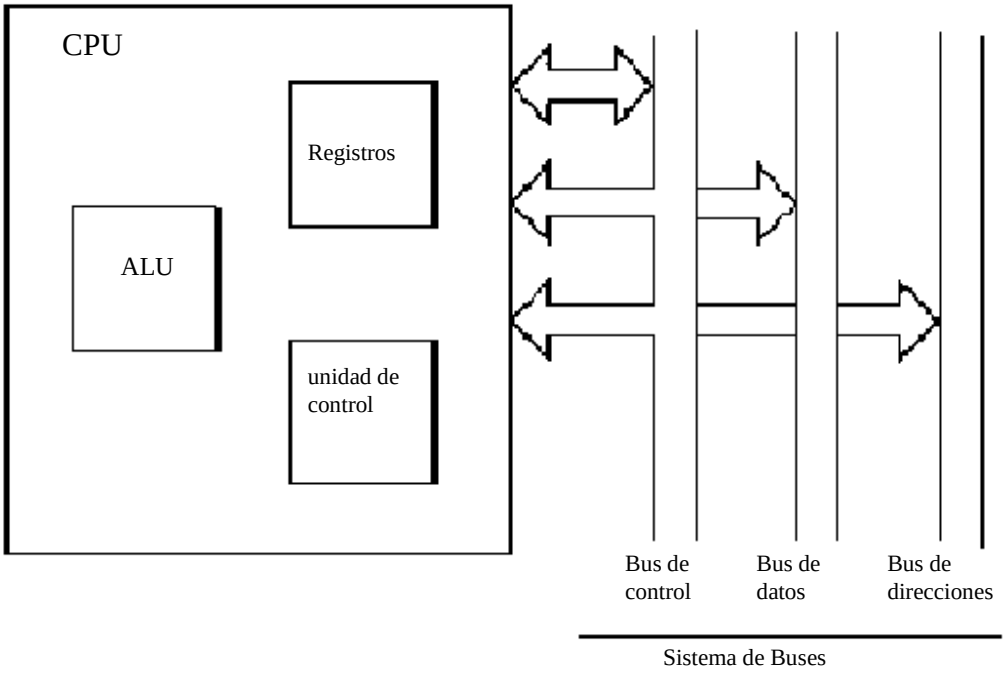
- La CPU es la combinación de:

» Los registros
» La Unidad aritmética y Lógica.
–Realiza las funciones aritméticas (suma, resta)
–Realiza las funciones lógicas (Y, O, Registro)
» La Unidad de mando o control
–Coordina las funciones del computador

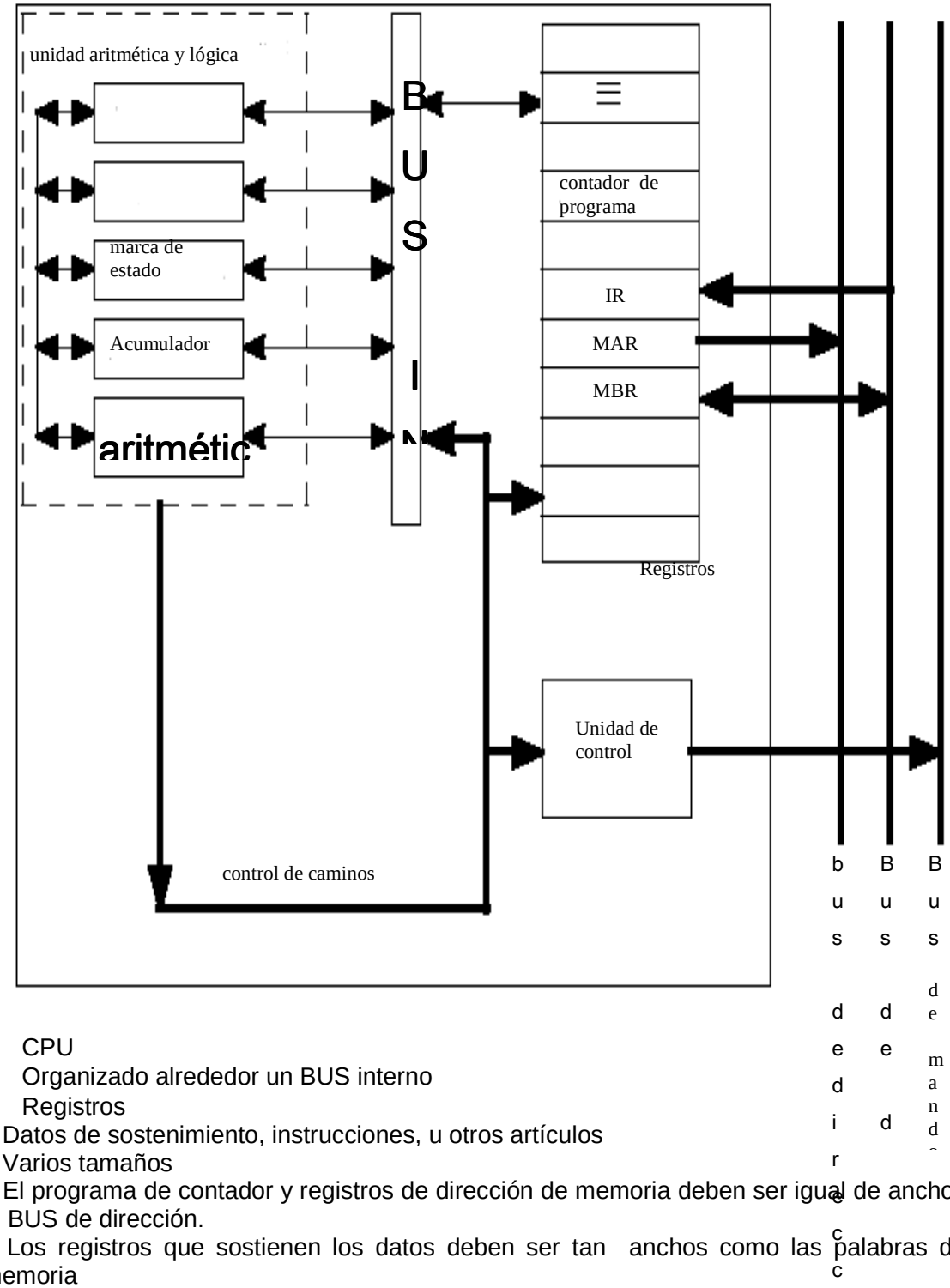
- SISTEMA DE INTERCONEXION

» BUS

- Unidad central de procesos (CPU)



- La CPU con el sistema de buses.



- CPU
- Organizado alrededor un BUS interno
- Registros

» Datos de sostenimiento, instrucciones, u otros artículos
» Varios tamaños
» El programa de contador y registros de dirección de memoria deben ser igual de anchos al BUS de dirección.
» Los registros que sostienen los datos deben ser tan anchos como las palabras de memoria

- Unidad Lógica Aritmética

- ALU

- EJECUTA OPERACIONES ARITMETICAS Y LOGICAS

- EL ACUMULADOR ES UN REGISTRO ESPECIAL

» Fuente de uno de los operandos
» El destino del resultado
» No siempre lo aplique a los procesadores más complejos

- Estado de Marca (los estados del Procesador formulan)

» Los Bits individuales para guardar la información sobre los resultados de operaciones
» Ejemplos
–El resultado es cero
–Acarreo
–La inundación
–El resultado es negativo

- Sub Unidades

» Suma
» Pruebas lógicas
» Funcionamientos lógicos
» cambiando
» comparación
» Multiplicación y división

Procesadores CISC y RISC.

- Clasificación de procesadores simples

ERA	ESTILO	HARDWARE	SOFTWARE
Antes 1970	RISC	Simple	Lenguaje ensamblador
~ 1970 - ~2000	CISC	Complejo Diseñado para simplificar tareas compiladas	Lenguajes de alto nivel FORTRAN Algol Pascal C
~ 1980 -	RISC	Simple Diseñado para trabajar tan rápido como sea posible	Lenguajes de alto nivel FORTRAN Pascal C optimizando necesidades compiladas
2000 -	RISC/ VLIW?	Simple Diseñado para trabajar tan rápido como sea posible Múltiples instrucciones en paralelo	Lenguajes de alto nivel FORTRAN Pascal C Java optimizando requerimientos compilados

- Conjunto de Instrucciones Complejas (CISC)

Como los lenguajes de alto nivel empezaron a hacerse populares, arquitectos de la computadora intentaron hacer máquinas con las capacidades de emparejar estructuras usadas por programadores.
Como un ejemplo, complejo que dirige los modos, se agregó para que una expresión como:

X = a[j++];

Pueda transformarse a una sola instrucción de la máquina.

En ese momento, la memoria estaba relativamente cara y lenta, para que compactando un programa en tan pocas instrucciones como sea posible, sea vista como una meta deseable. Esto significa que los programas más pequeños cargaron más rápidamente y que la memoria (escasa y cara) estaba más disponible para los datos.

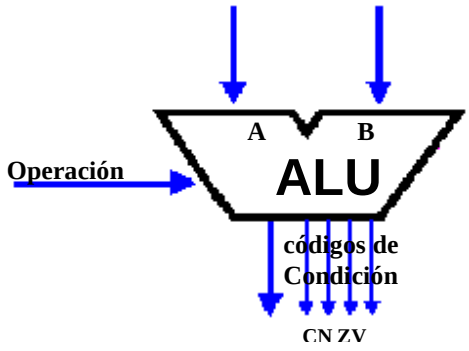
- Arquitecturas Significativas (CISC)

IBM	360, 370	Una extensa familia de procesadores mainframe fueron diseñados y fabricados en estas series.
DEC	PDP-11	En un momento la familia de minicomputadores más popular.
DEC	VAX	DEC fue el sucesor de su familia PDP-11- una de las primeras maquinas de 32 Bits.
Intel	x86	Empezando con los 8088 que impulsó los primeros PC IBM, la arquitectura del x86 se ha desarrollado con la popularidad de PCs.
Motorola	68K	Una familia de procesadores de 32-Bits uso tempranamente los Apple Macintoshes y extensivamente los incluyo en los sistemas.

- ALU

La Unidad Lógica Aritmética, es el "núcleo" de cualquier procesador: Es la unidad que realiza los cálculos. Un ALU típico tendrá dos puertos de entrada (A y B) y un puerto del salida (Y). También tendrá una entrada del control que le indica qué operación (agregue, substraiga, y, o, etc.) para realizar y adicionar salidas para los códigos de condición (llevar, inundar, negar, resultado cero).

ALU puede ser simple y realizar sólo unas operaciones: la aritmética de enteros (agregar, substraer), lógica booleana (y, o, complemento) y cambios (izquierda, corrija, ruede). Las ALU simples pueden encontrarse en procesadores pequeños de 4 y 8-Bits incluidos en los sistemas usados.



El ALU más complejo soportara un rango más ancho de operaciones de enteros (multiplicar y dividir), las operaciones del punto flotantes (agregar, substraer, multiplicar, dividir) e incluso las operaciones matemáticas (raíz cuadrada, el seno, el coseno, logaritmos, etc.). Hasta hace unos años, muchos procesadores de alta actuación no contenían soporte para multiplicar o dividir enteros y operaciones de punto flotante.

El mercado más grande para el propósito general de los procesadores programables es comercializarlo; dónde comúnmente las operaciones aritméticas son suma y resta. El multiplicar enteros y todas las otras operaciones más complejas se realizaron en el software - aunque esto toma tiempo considerable (multiplicar un entero de 32-Bits necesita 32 sumas y cambios), la frecuencia baja de estas operaciones significó que su baja velocidad disminuyó muy poco desde la actuación global de la máquina.
Así diseñadores asignarían su valiosa área de silicón para cache y otros dispositivos que tenían un impacto más directo en la actuación del procesador en el mercado designado.

Más recientemente, la geometría del transistor se ha encogido al punto dónde es posible conseguir 107 transistores en un solo chip. Así se hace factible para incluir el punto flotante de ALU en cada chip - probablemente más económico que diseñar procesadores separados sin la capacidad del punto flotante. De hecho, algunos fabricantes proporcionarán procesadores idénticos por otra parte con y sin la capacidad de punto flotante.
¡Esto puede lograrse económicamente marcando chips que tenían defectos sólo en la región de la unidad del punto flotante como procesadores " sólo - entero" y vendiéndolos a un precio más bajo para la información comercial que procesa el mercado! Esto tiene un efecto deseable de aumentar su rendimiento semiconductor significativamente - una unidad de punto flotante es bastante compleja y ocupa una área considerable de silicón - mire un micrográfico de chips típicos. Así la probabilidad de defectos en esta área es bastante alta.

En los procesadores simples, el ALU es un bloque grande de lógica combinacional con operandos A y B y los opcodes (códigos de operación) como las entradas y un resultado,(Y), más los códigos de condición como salidas. Los operandos y opcodes son aplicados en un borde del reloj y se espera que el circuito produzca un resultado antes del próximo borde del reloj. Así el retraso de la propagación a través del ALU determina un periodo mínimo del reloj y pone un límite superior a la frecuencia del reloj.

En los procesadores avanzados, el ALU es pesadamente el pipelined para extraer el throughput de la instrucción más alta. Las velocidades del reloj más rápidas son ahora posibles porque los funcionamientos complejos (eg las operaciones del punto flotantes) se hacen en fases múltiples: cada fase individual es más pequeña y más rápida.

- Unidad de control

- Genera los signos del mando

» Conecta los registros al BUS
» Controla la función del ALU
» Proporciona los signos cronometrando al sistema

- Todas las acciones de la unidad de control son asociadas con la decodificación y ejecución de instrucciones (deducir y ejecutar ciclos)

- Memoria

- Cada situación de memoria tiene una única dirección
- Para escribir, presente la dirección y los datos
- Para leer, presente la dirección y mire los datos
- Signos

» dirección
» datos
» control

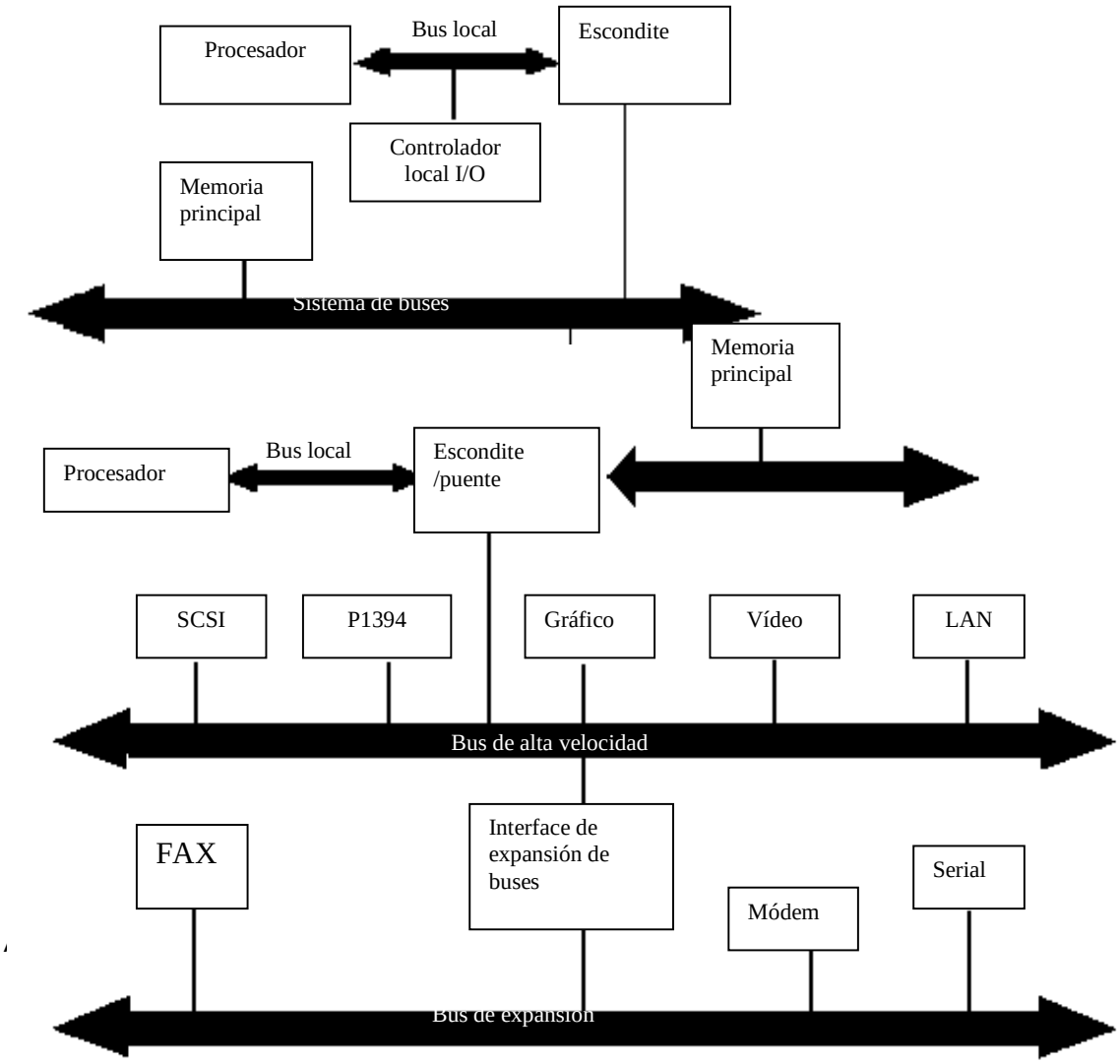
- Buses

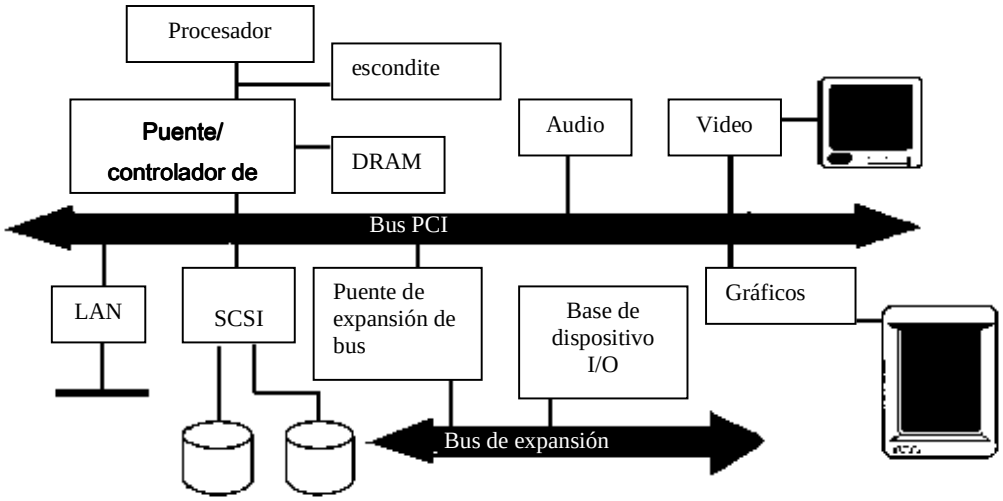
- Usados para comunicar entre las partes del computador
- Solo una emisión en un momento
- Sólo el dispositivo dirigido puede responder
- 2 niveles

» Interno
» Externo

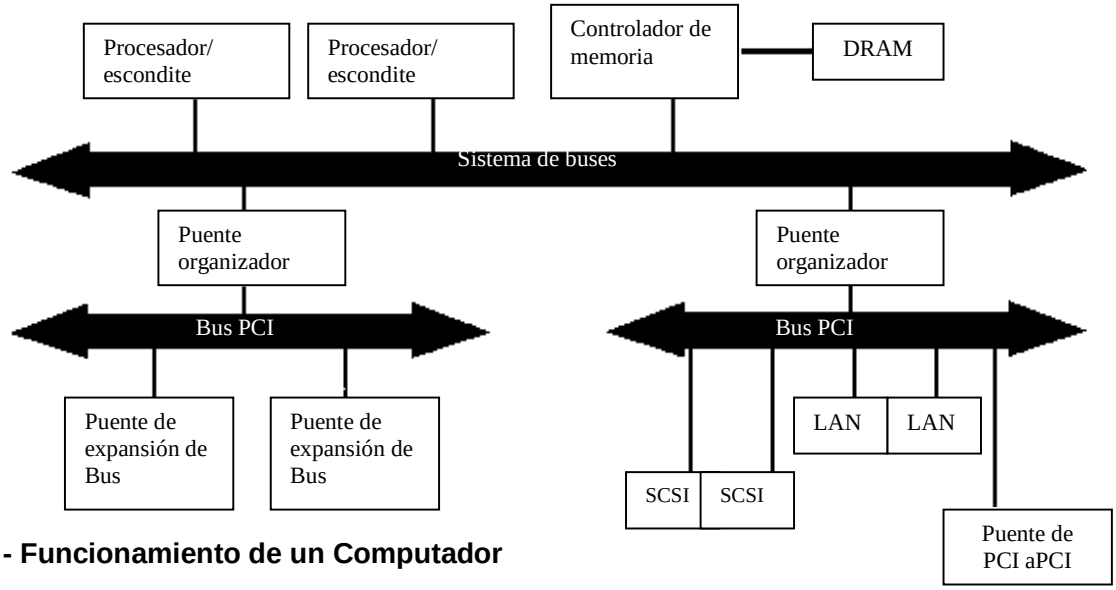
- Grupos de líneas de señales

» Dirección
» Datos
» Signos de control





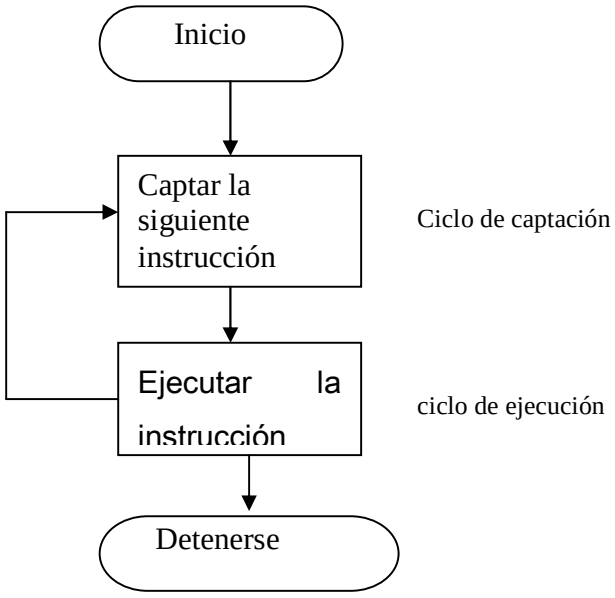
SISTEMA TÍPICO DESKTOP



- Funcionamiento de un Computador

- Pasos Básicos

» Ciclo de captación
–Captar la siguiente instrucción de la memoria a la unidad del mando
–Descifre esta instrucción
» Ciclo de ejecución
–Obedezca o ejecute la instrucción
» Empieza una nueva búsqueda de ciclo



- Instrucciones

- Las Instrucciones son modelos de Bits

» Puede rajarse en varios campos
» Un campo especifica la función a ejecutar
» Otros campos especifican la dirección en la memoria (o registros) de operando y dónde poner el resultado
» El número y tipo de campos del operando dependen de la instrucción

- Grupo de Instrucciones

- Grupo Aritmético

» Sumar
» Restar
» Multiplicar
» Dividir

- Grupo lógico

» Y
» Ó
» Registrar
» Rotar

- Mover o copiar grupo

» Transfiera los datos entre las ubicaciones de memoria
» Transfiera los datos entre los registros
» Transfiera los datos entre las ubicaciones de memoria y registros
» Transfiera los datos entre las ubicaciones de memoria y dispositivos de I/O

- Transferir el grupo de control

» salto o rama
» condiciones de transferencia
» llamadas del subprograma
» el retorno del subprograma

- Grupo Especializado

» Restablecer
» Parada
» Interrumpir Control

- Ejecución de la instrucción

- El ciclo de captación es básicamente idéntico para todas las instrucciones

- El ciclo de ejecución depende de la instrucción

» Es único para cada tipo de instrucción

- Los pasos Internos requeridos son controlados por la unidad de mando o control.

- Ciclo de captación

- Cargar los contenidos del contador del programa en el registro de dirección de memoria.
- Conectar el MAR al Bus externo
- Instruir la memoria para realizar una operación de lectura que produce datos colocados en el Bus de datos.
- Almacenar el valor del Bus de datos en el registro de memoria Buffer
- Transferir el valor de MBR dentro del registro de instrucciones
- Incrementar los contenidos del contador del programa para apuntar a la siguiente ubicación en la memoria

- Ejecutar Ciclo

- Diferir de la instrucción a la instrucción.
- Diferir de la computadora a la computadora
- Ejemplo de un ADD (ACC < -registro + ACC)

» Transfiera los contenidos del acumulador a la vía lógica Aritmética del Bus interno
» Transfiera los contenidos del registro dirigidos a la otra vía de entrada Lógica Aritmética del Bus interno.
» Signo de la Lógica Aritmética para agregar
» Guarde el resultado en el acumulador.

Autoevaluación.

1 Realice un mapa conceptual del capítulo

TERCERA UNIDAD
“Repertorio de Instrucciones”

Lenguaje de máquina
Paralelismo
Sistemas Multiprocesador

INTRODUCCIÓN

Un sistema de computador completo incluye tanto hardware como software. El hardware consta de los componentes físicos y todo el equipo asociado. El software se refiere a los programas que son escritos para el computador. Es posible familiarizarse con los diversos aspectos del software de un computador sin preocuparse por los detalles de cómo opera el hardware del mismo. Es también posible diseñar partes del hardware sin un conocimiento de las capacidades del software. Sin embargo, todos los que tienen que ver con la arquitectura del computador deben tener un conocimiento tanto del hardware como del software porque las dos ramas se influencian la una con la otra.

Si un programador está usando un lenguaje de alto nivel como C++, muy poco de la arquitectura de la máquina es visible. Un punto importante es que el diseñador del computador y el programador pueden ver la misma máquina, es el repertorio de instrucciones. Desde le punto de vista del diseñador, el conjunto de instrucciones máquina informa de las especificaciones funcionales de la CPU: implementar la CPU es una tarea que, en buena parte, implica implementar el repertorio de instrucciones máquina. Desde el punto de vista del usuario, quien elige programar en lenguaje máquina (realmente en lenguaje ensamblador) se hace consciente de la estructura de registros y de memoria, de los tipos de datos que soporta directamente la máquina y del funcionamiento de la ALU.

La segmentación de cauce, una técnica importante que aumenta las prestaciones de los computadores solapando la ejecución de varias instrucciones. Esto permite que las instrucciones puedan ser ejecutadas a una mayor velocidad de la que seria posible si cada instrucción tuviera que esperar a que la instrucción previa se completara antes de que pudiera empezar su ejecución. Exploramos técnicas para aprovechar el paralelismo entre instrucciones mediante la ejecución de varias instrucciones simultáneamente, mejorando aun mas las prestaciones. Los procesadores actuales suelen emplear segmentación de cauce además de técnicas para aprovechar el paralelismo entre instrucciones, así que supondremos que todos los procesadores ILP descritos en este capitulo están segmentados a no ser que se indique explícitamente lo contrario.

La segmentación de cauce mejora las prestaciones al incrementar la frecuencia con la que las instrucciones pueden ser ejecutadas. Sin embargo, existen limites para mejora de prestaciones que se pueden alcanzar usando segmentación de cauce.

Conforme vamos añadiendo mas y mas etapas en el cauce, el retardo de los registros de acopio requeridos entre las etapas empieza a ser una parte importante del tiempo de ciclo, reduciendo el beneficio de incrementar la profundidad del cauce. Otro aspecto importante es que al incrementar la profundidad del cauce se incrementa el retardo de los saltos y la latencia de las instrucciones, aumentando el número de ciclos en los que el cauce se detiene entre instrucciones dependientes.

Dado que las restricciones tecnológicas y la disminución de las prestaciones conforme se aumenta la profundidad del cauce limitan la frecuencia máxima del reloj de un procesador para un proceso de fabricación determinado, los diseñadores han empezado a aprovechar el paralelismo para mejorar las prestaciones mediante la realizaron de varias tareas a la vez. Los sistemas de computación paralelos suelen pertenecer a una de estas dos familias: sistemas multiprocesador y procesadores con paralelismo entre instrucciones, que se diferencian en el tamaño de las tareas que pueden ejecutar en paralelo. En los

sistemas multiprocesador, se ejecutan en paralelo tareas relativamente grandes, como procedimientos o iteraciones de un bucle. En el extremo contrario se encuentran los procesadores ILP, que ejecutan instrucciones independientes de forma simultanea.

Los procesadores que aprovechan el paralelismo entre instrucciones han tenido mucho mas éxito que los sistemas multiprocesador en el sector de mercado cubierto por los PC o las estaciones de trabajo para aplicaciones de propósito general porque pueden mejorar las prestaciones de los programas convencionales, cosa que no es posible con los sistemas multiprocesador. En concreto, los procesadores superescalares pueden obtener ganancias en el tiempo de ejecución de programas que fueron compilados para procesadores secuenciales (sin ILP) sin necesidad de recompilacion. La otra arquitectura que trataremos en este capitulo, los procesadores VLIW, requiere que los programas sean recompilados, pero obtiene muy buenas prestaciones en programas escritos en lenguajes secuenciales como C o FORTRAN cuando se recompilan para un procesador VLIW.

El procesador contiene varias unidades de ejecución de instrucciones, cada una de ellas lee sus operandos y escribe su resultado en un único banco de registros centralizado. Una vez que una operación escribe su resultado en el bando de registros, dicho resultado será accesible para el resto de unidades de ejecución en el siguiente ciclo, permitiendo que las operaciones se ejecuten en unidades de ejecución diferentes de las que generan sus entradas. Los procesadores con paralelismo entre instrucciones suelen disponer de un complejo hardware de atajo que anticipa los resultados de cada instrucción a todas las unidades de ejecución para reducir el retardo entre instrucciones dependientes.

Las instrucciones que forman un programa son gestionadas por la lógica de emisión de instrucciones, que lanza las instrucciones a las unidades de ejecución en paralelo. Esto permite que los cambios de flujo de control, como los saltos, puedan procesarse simultáneamente en todas las unidades de ejecución, facilitando la tarea de escribir y compilar programas para procesadores con paralelismo entre instrucciones.

En la figura todas las unidades de ejecución se han dibujado como módulos idénticos. Sin embargo, en la mayoría de los procesadores actuales, algunas o todas las unidades de ejecución solo pueden procesar un subconjunto de las instrucciones del procesador. La opción mas frecuente es separar las unidades de ejecución de las de coma flotante, ya que estas operaciones requieren un hardware muy diferente. Implementar estos dos tipos de hardware como unidades de ejecución separadas aumenta el número de instrucciones que pueden ser ejecutadas en paralelo sin incrementar significativamente la cantidad de hardware necesario. En otros procesadores, algunas de las unidades de ejecución de enteros se diseñan para ejecutar solo algunas instrucciones de enteros, generalmente las más frecuentes. Esto reduce el tamaño de estas unidades de ejecución, aunque implica que algunas combinaciones de instrucciones de enteros independientes no pueden ser ejecutadas en paralelo.

En esta sección examinaremos los sistemas multiprocesador. Un multiprocesador es un sistema de cómputo que tiene varias CPU y un solo espacio de direcciones visible para todas las CPU. La máquina ejecuta una copia del sistema operativo, con un conjunto de tablas que incluyen las tablas en las que se lleva la contabilidad de cuáles páginas de memoria están ocupadas y cuáles están libres. Cuando un proceso se bloquea, su CPU guarda su estado en las tablas del sistema operativo y busca en esas tablas otro proceso que pueda ejecutar. Es esta imagen de sistema único lo que distingue un multiprocesador de una multicomputadora.

Capítulo 1: Lenguaje de Máquina

1.1. Visión del programador

El funcionamiento de la CPU está determinado por las instrucciones que ejecuta. Estas instrucciones se denominan instrucciones máquina o instrucciones del computador. Al conjunto de instrucciones distintas que puede ejecutar la CPU se le denomina repertorio de instrucciones de la CPU.

Para escribir un programa de un computador se tiene que especificar, directa o indirectamente, una secuencia de instrucciones de máquina. Las instrucciones de máquina dentro del computador forman un patrón binario ue es difícil, si no imposible, para que la gente trabaje y lo entienda. Es preferible escribir programas con símbolos más familiares del conjunto de caracteres alfanuméricos. Como consecuencia, hay necesidad de traducir los programas simbólicos orientados al usuario en programas binarios que sean reconocidos por el hardware.
Un programa escrito por el usuario puede depender o no del computador físico que corre este programa. Por ejemplo, un programa escrito en Fortran estándar al código binario del comptuador disponible en la instalación particular. Pero el programa traductor es dependiente de la máquina debido a que eél debe traducir el pgroama Fortran a código binario reconocido por el hardware del computador particular utilizado.

- Lenguajes de programación

Un programa es una lista de instrucciones o enunciados para dirigir el computador para que realice las tareas de procesamiento de datos requeridos. Hay varios tipos de lenguajes de programación que uno puede escribir para un computador pero el computador puede ejecutar solamente programas cuando ellos se presentan internamente en forma binaria. Los programas escritos en cualquier otro lenguaje deben ser traducidos a la respresentación binaria de instrucciones antes de que puedan ser ejecutados por el computador. Los programas escritos para un computador deben estar en una de las siguientes categorías:

1. Código binario: esta es una secuencia de instrucciones y operando en binario que enumera la representación exacta de instrucciones como aparecen en la memoria del computador.
2. Código octal o hexadecimal: esta es una traducción equivalente del código binario a representación octal o hexadecimal.
3. Código simbólico: el usuario emplea símbolos (letras, números, o caracteres especiales) para la parte de operación, la parte de dirección y otras partes del código de instrucción. Cada instrucción simbólica puede traducirse en una instrucción codificada en binario. Esta traducción es hecha por un programa especial denominado un ensamblador. Debido a que un ensamblador traduce los símbolos de este tipo de programa simbólico se conoce como un programa de lenguaje ensamblador.
4. Lenguaje de programación de alto nivel: estos son lenguajes especiales desarrollados para reflejar los procedimientos utilizados en la solución de un problema antes que preocuparse por el comportamiento del hardware del computador Un ejemplo de un lenguaje de programación de alto nivel es el C++. El emplea símbolos y formatos orientados al problema. El programa es escrito en una secuencia de enunciados en la forma que la gente prefiera pensar cuando está resolviendo el problema. Sin embargo,

cada uno de los enunciados debe traducirse en una secuencia de instrucciones binarias antes de que el programa pueda ser ejecutado en un computador. El programa que traduce un programa de lenguaje de alto nivel a binario se denomina un compilador.

Estrictamente hablando, un programa de lenguaje de máquina es un programa de categoría binaria. Debido a la equivalencia simple entre binario y el octal o hexadecimal, es costumbre referir a la catergoria como lenguaje de máquina. Debido a la relación uno a uno entre las instrucción simbólica y su equivalente binario, un lenguaje ensamblador es considerado un lenguaje de nivel de máquina.

Lenguaje de programación se refiere al estudio de la estructura d elos diversos lenguajes de programación de alto nivel. Este estudio se lleva a cabo independiente d ecualquier dispositivo de cómputo particula y su hardware. Puesto que estamos interesados en las relaciones entre software y hardware, el término utilizado aquí tiene connotaciones diferentes puesto que incluye lenguajes de máquina de alto nivel.

1.2. Formatos de las instrucciones

Cada instrucción debe contener la información que necesita la CPU para su ejecución. Los elementos constitutivos de una instrucción máquina son:

- **Código de operación:** Especifica la operación a realizar (suma, E/S, etc.), La operación se indica mediatne un código binario, denominado código de operación o, abreviadamente, codop.
- **Referencia a operandos fuente:** La operación puede implicar a uno o más operandos fuente, es decir, operandos que son entradas para la instrucción.
- **Referencia al operando resultado:** La operación puede producir un resultado.
- **Referencia a la siguiente instrucción:** Dice a la CPU de dónde captar la siguiente instrucción tras completarse la ejecución de la instrucción actual.

La siguiente instrucción a capturar está en memoria principal o, en el caso de un sistema de memoria virtual, bien en memoria principal o en memoria secundaria (disco). En la mayoría de los casos, la siguiente instrucción a captar sigue inmediatamente a la instrucción en ejecución. En tales casos no hay referencia explícita a la siguiente instrucción. Cuando sea necesaria una referencia explícita, debe suministrarse la dirección de memoria principal o de memoria virtual.

Los operandos fuente y resultado pueden estar en alguna de las siguientes áreas:

Memoria principal o virtual: como en las referencias a instrucciones siguientes, debe indicarse la dirección de memoria principal o de memoria virutal.

Registro de la CPU: salvo raras excepciones, una CPU contine uno o más registros que pueden ser referenciados por instrucciones máquina. Si sólo existe un registro, la referencia a él puede ser implícita. Si existe más de uno, cada registro tenfrá asignado un número único, y la instrucción debe contener el número del registro deseado.

Dispositivo de E/S: La instrucción debe especificar el módulo y ldispositivo de E/S para la operación. En el caso de E/S asignadas en memoria, se dará otra dirección de memoria principal o virtual.

1.3. Modos de direccionamiento

Dentro del computador, cada instrucción se representa por una secuencia de bits. La instrucción está dividida en campos, correspondientes a los elementos constitutivos dela misma. En la mayoría de los repertorios de instrucciones se emplea más de un formato. Durante su ejecución, la instrucción se escribe en un registro de insturcción IR de la CPU. La CPU debe ser capaz de extraer los datos de los distintos campos dela instrucción para realizar la operación requerida.

Es difícil, tanto para los programadores como para los lectores de un libro de texto, manejar las representaciones binarias de las instrucciones máquina. Por ellos es una práctica común utilizar representaciones simbólicas de las instrucciones máquina.

Los codops se representan mediante abreviaturas, denominadas nemotécnicos, que indican la operación en cuestión. Ejemplos usuales son:

ADD Sumar
SUB Restar
MPY Multiplicar
DIV Dividir
LOAD Cargar datos de memoria
STOR Almacenar datos en memoria (memorizar)

Los operandos tambien suelen representarse simbólicamente. Por ejemplo, la instrucción
ADD R,Y
Puede significar sumar el valor contenido en la posición de datos Y al contenido del registro R. En este ejemplo, Y hace referencia a la dirección de una posición de memoria, y R a un registro particular. Observe que la operación se realiza con el contenido de la posición, no con su dirección.

Es posible pues, escribir un programa en lenguaje máquina de forma simbólica. Cada codop simbólico tiene una representación binaria fija, y el programador especifica la posición de cada operando simbólico. Por ejemplo, el programador podría comenzar con una lista de definiciones:

X=513
Y=514

Y así sucesivamente. Un sencillo programa aceptaría como entrada esta información simbólica, convertiría los codops y referencias a operandos a forma binaria y construiría las instrucciones máquina binarias.

Es raro encontrar ya programadores en lenguaje máquina. La mayoría d elos programas actuales se escriben en un lenguaje de alto nivel o, en ausencia delm ismo, en lenguaje ensamblador, sobre el que trataremos al final de este capítulo. No obstante, el lenguaje máquina simbólico sigue siendo útil para describir las instrucciones máquina, y con ese fin lo utilizaremos.

1.4. Instrucciones Típicas

Considere una instrucción de alto nivel, tal y como se expresaría en un lenguaje como el BASIC o el FORTRAN. Por ejemplo:

X=X+Y

Esta senctencia ordena al comptuador sumar los valores almacenados en X y en Y, y poner el resultado en X. Supongamos que las variables X e Y corresponden a las pociiones 513 y 514. Considerando un repertorio simple de instrucciones máquina, la operación podría llevarse a cabo con tres instrucciones:

1. Cargar un registro con el contenido de la posición de memoria 513
2. Sumar al registro el contenido de la posición de memoria 514
3. Memorizar el contenido del registro en la posición de memoria 513

Una sola instrucción en un lenguaje de alto nivel puede requerir tres instrucciones máquina. Este es un caso típico de relación entre un lenguaje de alto nivel y un lenguaje máquina. Un lenguaje de alto nivel expresa las operaciones de forma algebraica concisa, utilizando variables.

Como hemos visto, una de las principales diferencias entre las arquitecturas RISC y las CISC es el conjunto de instrucciones que pueden acceder a la memoria. Una cuestión relacionada con ésta, que afecta tanto a las arquitecturas RISC como a las CISC, es la elección de los modos de direccionamiento que admite la arquitectura. Los modos de direcclonamiento de una arquitectura son el conjunto de sintaxis y métodos que usan las instrucciones para especificar una dirección de memoria, ya sea la dirección objeto de una referencia a memoria o la dirección de salto de una bifurcación. Dependiendo de la arquitectura, ciertos modos de direccionamiento puedenestar disponibles sólo para algunas de las instrucciones que hacen referencia a la memoria. Las arquitecturas que permiten que cualquier instrucción que hace referencia a la memoria use cualquier modo de direccionamiento se conocen como ortogonales, debido a que la elección del modo de direccionamiento es independiente de la elección de la instrucción.

Hasta ahora, hemos usado únicamente dos modos de direccionamiento: direccionamiento a través de registro en las instrucciones de carga, instrucciones de almacenamiento o instrucciones CISC que hacen referencia a memoria, y direccionamiento a una etiqueta en el caso de las instrucciones de salto. En el direccionamiento a través de registro, una instrucción lee el valor de un registro y lo usa como dirección de referencia a memoria o como destino de un salto. Utilizamos la sintaxis (rx) para indicar que se está usando el modo de direccionamiento a través de registro. Sería posible un repertorio de instrucciones que sólo empleara el modo de direccionamiento a través de registro, ya que cualquier dirección podría calcularse usando instrucciones aritméticas y almacenarse en un registro. Los procesadores incluyen otros modos de direccionamiento porque permiten reducir el número de instrucciones necesarias para calcular las direcciones, mejorando de ese modo las prestaciones. El segundo modo de direccionamiento que hemos visto hasta ahora es el direccionamiento a una etiqueta, en el cual una instrucción de bifurcación especifica su destino como una etiqueta colocada en alguna instrucción en otra parte del programa. Estas etiquetas de texto no aparecen en la versión en lenguaje máquina del programa.

De hecho, la mayor parte de las instrucciones de bifurcación no contienen en absoluto sus direcciones de destino de manera específica. En su lugar, el ensamblador enlazador traduce la etiqueta en un desplazamiento (que puede ser tanto positivo. como negativo) desde la posición de la instrucción de bifurcación a la posición de destino. En efecto, la instrucción de bifurcación le dice al procesador a cuánta distancia de ella se encuentra la instrucción destino, en lugar de dónde está situada exactamente dicha instrucción destino. El procesador suma el desplazamiento al PC de la instrucción de bifurcación para obtener la dirección destino del salto.

El hecho de usar desplazamientos en lugar de direcciones explícitas en el direccionamiento a etiquetas tiene dos ventajas. En primer lugar, reduce el número de bits necesarios para codificar la instrucción. La mayoría de los saltos tienen destinos relativamente cercanos al salto, de modo que puede usarse un número pequeño de bits para codificar el desplazamiento. Cuando un salta tiene un destino muy lejano, la dirección destino puede calcularse por medio de otras instrucciones y puede usarse un modo de direccionamiento a través de registro u otro similar. En segundo lugar, la utilización de desplazamientos en lugar de direcciones explícitas en las instrucciones de bifurcación permite al cargador colocar el programa en distintas posiciones de memoria sin necesidad de modificarlo. Si se utilizaran direcciones explícitas, la dirección destino de cada salto tendría que recalcularse cada vez que se carga el programa. Esta característica es particularmente útil en bibliotecas del sistema operativo que se enlazan dinámicamente con el programa en tiempo de ejecución, ya que tales bibliotecas no tienen la habilidad de predecir en qué dirección se cargarán.

- DISEÑO DEL REPERTORIO DE INSTRUCCIONES

Uno de los aspectos más interesantes y más analizados del diseño de un computador, es el diseño del repertorio de instrucciones del lenguaje máquina. El diseño de un repertorio de instrucciones es muy complejo, ya que afecta a muchos aspectos del computador. El repertorio de instrucciones define muchas de las funciones realizadas por la CPU y tiene, por tanto.

un efecto significativo sobre la implementación de la misma. El repertorio de instrucciones es el medio que tiene el programador para controlar la Cpu. En consecuencia, deben considerarse las necesidades del programador a la hora de diseñar el repertorio de instrucciones.

Puede sorprender saber que algunos de los aspectos más básicos relativos al diseño de repertorios O de instrucciones siguen siendo temas de controversia. Los más importantes entre dichos aspectos de diseño son:
. Repertorio de operaciones: Cuántas y qué operaciones considerar, y cuán complejas deben ser.
. Tipos de datos: Los distintos tipos de datos con los que se efectúan operaciones.
. Formatos de instrucciones: Longitud de la instrucción (en bits), número de direcciones, tamaño de los distintos campos, etc.
. Registros: Número de registros de la CPU que pueden ser referenciados por instrucciones, y su uso.
. Direccionamiento: El modo o modos de direccionamiento mediante los cuales puede especificarse la dirección de un operando.

Estos aspectos están fuertemente interrelacionados, y deben considerarse conjuntamente en el diseño de un repertorio de instrucciones. Este libro, por supuesto, debe considerados en secuencia, pero se intentará mostrar las relaciones entre ellos.
Dada la importancia del tema, un bloque amplio de la Parte In del libro se dedica al diseño del repertorio de instrucciones.

(RISC, Reduced Instruction Set Computer). La arquitectura RISC cuestiona muchas de las decisiones tomadas sobre repertorios de instrucciones de computadores comerciales contemporáneos. Un ejemplo de máquina RISC es el PowerPC.

Todos los lenguajes máquina incluyen tipos de datos numéricos. Incluso en el procesamiento de datos no numéricos se necesitan números que actúen como contadores, longitudes de campos, etc. Una distinción importante entre los números utilizados en las matemáticas ordinarias y los almacenados en un computador, es que estos últimos están limitados. Esto es cierto en dos sentidos. En primer lugar, hay un límite para la magnitud de los números repre-sentables en una máquina; en segundo lugar, en el caso de números en coma flotante, su precisión está limitada. Por tanto, el programador debe ser consciente de las consecuencias del redondeo, el desbordamiento, o el desbordamiento a cero.

En los computadores son usuales tres tipos de datos numéricos:
. Enteros o en coma fija
. En coma flotante
. En decimal
Los dos primeros se vieron con detalle en la Unidad 2. Aunque todas las operaciones internas del computador son en esencia binarias, los usuarios del sistema utilizamos números decimales. Es necesario pues, convertir de decimal a binario las entradas, y de binario a decimal las salidas. Para aplicaciones en las que hay muchas entradas/salidas frente a pocos cálculos, y cálculos comparativamente simples, a veces es preferible memorizar y operar con los números directamente en su forma decimal. La presentación más común para ello es la de decimal empaquetado.

Autoevaluacion.

1. Mostrar un programa hecho en Assembler identificando los componentes así como todo el proceso de ejecución, identificando los registros usados.

Capítulo 2: Paralelismo

2.1. Descripción : ¿Qué es el paralelismo entre instrucciones?

Los procesadores con paralelismo entre instrucciones aprovechan el hecho de que muchas de las instrucciones en un programa secuencial no dependen de las instrucciones del programa cercanas que las preceden. Por ejemplo, considere el fragmento de programa de la parte izquierda de la figura. Las instrucciones 1, 3 y 5 son dependientes entre si porque la instrucción 1 genera un valor que es usado por la instrucción 3, la cual genera un resultado que utiliza la instrucción 5. Las instrucciones 2 y 4 no usan los resultados de ninguna otra instrucción del fragmento y no generan resultados que se usen por ninguna instrucción del fragmento. Estas dependencias requieren que las instrucciones 1, 3 y 5 sean ejecutadas en orden para generar un resultado correcto, pero las instrucciones 2 y 4 pueden ejecutarse antes, después o en paralelo con cualquiera de las otras instrucciones sin cambiar los resultados del fragmento de programa.

En un procesador que ejecute las instrucciones secuencialmente, el tiempo de ejecución de este fragmento de programa seria de al menos 5 ciclos, incluso en un procesador sin segmentar con una latencia de un para cada instrucción. Por el contrario, in procesador no segmentado que sea capaz de ejecutar dos instrucciones de forma simultanea podría ejecutar este fragmento de programa en tres ciclos si cada instrucción tuviera una latencia de un ciclo, como se muestra en la parte derecha de la figura. Este tiempo de ejecución no se puede disminuir, ni incluso aumentando el numero de instrucciones que el procesador puede ejecutar en paralelo, dado que las instrucciones 1, 3 y 5 son dependiente entre si.

Este ejemplo ilustra a la vez la potencia y el punto débil del paralelismo entre instrucciones. Los procesadores ILP pueden alcanzar ganancias significativas en una amplia variedad de programas mediante la ejecución de instrucciones en paralelo, pero su mejora máxima en las prestaciones esta limitada por las dependencias entre instrucciones. En general, conforme se van añadiendo mas unidades de ejecución a un procesador, disminuye la mejora incremental en las prestaciones que resulta de la adición de cada unidad de ejecución. Sin embargo, conforme el numero de unidades de ejecución se aumenta a cuatro, ocho o mas, las unidades de ejecución adicionales quedan desocupadas la mayoría del tiempo, particularmente si el programa no ha sido compilado para sacar partido de dichas unidades.

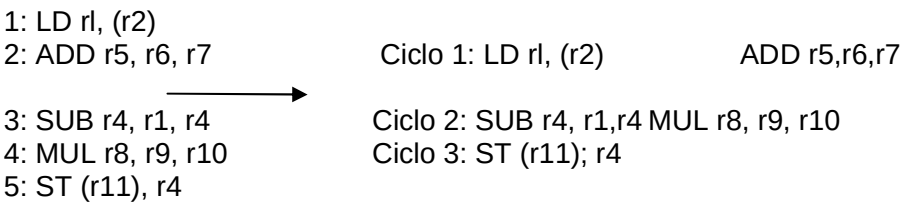


Figura. Ejemplo de paralelismo entre instrucciones.

2.2. Limitaciones del paralelismo entre instrucciones

Las prestaciones de los procesadores ILP están limitados por la cantidad de paralelismo entre instrucciones que el compilador y el hardware pueden identificar en el programa. El

paralelismo entre instrucciones esta limitado por varios factores: la dependencia de datos, las dependencias de nombre (riesgos WAR y WAW, escritura después de lectura y escritura después de escritura respectivamente) y los saltos. Adicionalmente, la capacidad de aprovechar paralelismo entre instrucciones de un procesador determinado puede estar limitada por el numero y el tipo de unidades de ejecución presentes en el procesador y por las restricciones sobre que instrucciones del programa se pueden analizar para localizar operaciones que puedan ser ejecutadas en paralelo.

Las dependencias RAW (lectura después de escritura) limitan las prestaciones al requerir que las instrucciones a las que afectan tengan que ser ejecutadas en orden para generar resultados correctos y representan una limitación fundamental en la cantidad de paralelismo entre instrucciones disponibles en los programas. Las instrucciones con dependencia WAW se deben emitir también de forma ordenada para asegurar que la ultima escritura en el registro de destino se escriba por la instrucción correcta. Las instrucciones con dependencias WAR se pueden emitir en el mismo ciclo, pero no fuera de orden, dado que las instrucciones leen sus operandos del banco de registros antes de ser emitidas. Por tanto, una instrucción que lee un registro puede ser emitida en el mismo ciclo que una instrucción que escribe en el registro y que aparece después en el programa, ya que la primera instrucción leerá sus registros de entrada antes de que la segunda instrucción genere el nuevo valor para su registro de destino. Posteriormente en este capitulo, describiremos el *renombrado de registros*, una técnica hardware que permite que las instrucciones con dependencias WAR y WAW puedan ser emitidas fuera de orden sin cambiar los resultados del programa.

Los saltos limitan el paralelismo entre instrucciones porque el procesador no puede saber que instrucciones debe ejecutar después de un salto hasta que su ejecución se haya completado. Esto requiere que el procesador tenga que esperar a que el salto se complete antes de ejecutar la siguiente instrucción. Como se mencione anteriormente, la mayoría de los procesadores incorporan hardware para predecir saltos y reducir el impacto de los mismos en el tiempo de ejecución prediciendo la dirección de destino del salto antes de que el salto se haya completado.

EJEMPLO
Considere el siguiente fragmento de programa:

```
ADD  r1, r2, r3
LD r4  (r5)
SUB r7, r1, r9
MUL r5,r4, r4
SUB r1, r12, r10
ST (r13), r14
OR  r 15, r14, r12
```

¿Cuanto tardaría en emitirse este fragmento de programa en un procesador que puede ejecutar dos instrucciones simultáneamente? ¿Y en un procesador que pueda ejecutar cuatro instrucciones simultáneamente? Suponga que el procesador puede ejecutar las instrucciones en cualquier orden siempre que no se violen las dependencias de datos, que todas las instrucciones tienen una latencia de un ciclo y que todas las unidades de ejecución del procesador pueden ejecutar cualquier tipo de instrucción.

SOLUCION

En un procesador que pueda ejecutar dos instrucciones simultáneamente, este fragmento de programa tardaría cuatro ciclos en emitirse. A continuación se muestra una posible secuencia de emisión, aunque se pueden formar mas secuencias de instrucciones que tarden lo mismo.

Ciclo 1: ADD r1,r2,r3 LD r4, (r5)
Ciclo 2: SUB r7, r1, r9 MUL r5, r4, r4
Ciclo 3 : SUB r1, r12, r10 ST (r13), r14
Ciclo 4: OR r15, r14, r12

Si el procesador pudiera ejecutar cuatro instrucciones simultáneamente, el programa se podría emitir en dos ciclos, tal y como se muestra a continuación:

Ciclo 1: ADD r1, r2, r3 LD r4, (r5) ST (r13), r14 OR r15, r14, r12
Ciclo 2: SUB r7, r1, r9 MUL r5, r4, r4 SUB r1, r12, r10

Observe que, independientemente del numero de instrucciones que el procesador pueda ejecutar simultáneamente, no es posible emitir todo el fragmento de programa en un solo ciclo, ya que existen dependencias RAW entre las instrucciones ADD r2, r3 y SUB r7, r1, r9 y entre las instrucciones LD r4, (r5) y MUL r5, r4, r4. Obsérvese también que las instrucciones SUB r7, r1, r9 y SUB r1, r12, r10, que tienen una dependencia WAR, se emiten en el mismo ciclo de reloj.

2.3. Procesadores superescalares

Los procesadores superescalares se fundamentan en que el hardware extraiga paralelismo entre instrucciones a partir de programas secuenciales. Durante cada ciclo, la lógica de emisión de instrucciones de un procesador superescalar analiza las instrucciones en el programa secuencial para determinar que instrucciones pueden emitirse en ese ciclo. Si existe en el programa suficiente paralelismo entre instrucciones, un procesador superescalar puede ejecutar una instrucción por cada unidad de ejecución y por ciclo, incluso si el programa fue originalmente compilado para ejecutarse en un procesador que podía ejecutar solo una instrucción por ciclo.

Esta capacidad es una de las grandes ventajas de los procesadores superescalares y es la razón por la cual prácticamente todas las CPU de las estaciones de trabajo y de los PC actuales son procesadores superescalares. Los procesadores superescalares pueden ejecutar programas que originalmente fueron compilados para procesadores puramente secuenciales y pueden alcanzar mejores prestaciones en estos programas que los procesadores que no son capaces de aprovechar el paralelismo entre instrucciones. Por tanto, los usuarios que compren sistemas que incorporen un procesador superescalar podrán instalar sus viejos programas en estos sistemas y obtener mejores prestaciones en ellos de las que eran posibles en sus sistemas antiguos.

La capacidad de los procesadores superescalares para aprovechar el paralelismo entre instrucciones en los programas secuenciales no significa que los compiladores sean irrelevantes en los sistemas que incorporan dichos procesadores. Es mas, el uso de un buen compilador es incluso mas critico para las prestaciones de un sistema superescalar

que para otro que dependa de un procesador secuencial. Los procesadores superescalares pueden analizar solo una pequeña *ventana* de las instrucciones del programa en cada momento para determinar que instrucciones se pueden ejecutar en paralelo. Si el compilador es capaz de planificar las instrucciones del programa de forma que en la ventana de instrucciones haya un gran numero de instrucciones independientes, el procesador, superescalar que las ejecute será capaz de alcanzar unas buenas prestaciones. Sin embargo, si la mayoría de las instrucciones de la ventana son dependientes entre si en cualquier instante de tiempo, un procesador superescalar no será capaz de ejecutar el programa mucho mas rápido de lo que lo haría un procesador secuencial.

2.4. Ejecución fuera de orden frente a ejecución en orden

Una de las decisiones mas importantes que se deben tomar a la hora de diseñar un procesador superescalar, relacionada con el compromiso entre la complejidad y las prestaciones del mismo, es si el procesador debe ejecutar las instrucciones en el orden en el que aparecen en el programa (ejecución en orden), o si el procesador puede ejecutar las instrucciones en cualquier orden siempre que no se cambie el resultado del programa (ejecución fuera de orden). La ejecución fuera de orden puede proporcionar mejores prestaciones que la ejecución en orden, pero requiere una implementación hardware mucho mas compleja.

2.5 Estimación del tiempo de ejecución para procesadores con ejecución en orden

Con la fórmula siguiente tenemos el tiempo de ejecución de los programas en un procesador segmentado como la suma del tiempo para emitir todas las instrucciones del programa y de la latencia del cauce del procesador, es decir:

Tiempo de ejecución (en ciclos) = Latencia del cauce + Tiempo de emisión - 1

Para los procesadores ILP segmentados, podemos usar la misma expresión para el tiempo de ejecución, pero el calculo del tiempo de emisión se vuelve una tarea mas compleja, ya que el procesador puede emitir mas de una instrucción por ciclo. Ya que la latencia del cauce del procesador no varia de un programa a otro, la mayoría de los ejercicios de este capitulo se centraran en la obtención del tiempo de emisión de programas en procesadores ILP.

En los procesadores superescalares con ejecución en orden, el tiempo de emisión de un programa se puede determinar analizando paso a paso y de manera secuencial el código para determinar cuando se puede emitir cada una de las instrucciones, de forma similar a la técnica usada para los procesadores segmentados que solo pueden ejecutar una instrucción cada ciclo. La diferencia clave entre un procesador superescalar con ejecución en orden y un procesador segmentado son superescalar es que el procesador superescalar puede emitir una instrucción en el mismo ciclo que la instrucción anterior del programa si no contienen ninguna dependencia de datos, siempre que el numero de instrucciones emitidas en el ciclo no supere el numero de instrucciones que el procesador puede ejecutar simultáneamente. En los procesadores en los que algunas o todas las unidades de ejecución pueden ejecutar solo algunos tipos de instrucciones, el conjunto de instrucciones emitidas debe satisfacer las restricciones de las unidades de ejecución.

EJEMPLO
¿ Cuanto tiempo tardara en ejecutarse la siguiente secuencia de instrucciones en un procesador superescalar con ejecución en orden y dos unidades de ejecución, pudiendo ejecutar cada una de ellas cualquier tipo de instrucción?
Las instrucciones de carga tienen una latencia de dos ciclos y todas las demás operaciones tienen una latencia de un ciclo. Suponga que la profundidad del cauce es de 5 etapas.

LD r1, (r2)
ADD r3, r1, r4
SUB r5, r6, r7
MUL r8, r9, r10

SOLUCION

La latencia del cauce de este procesador es de cinco ciclos. Suponiendo que la instrucción LD se emite en el ciclo *n,* la instrucción ADD no se podrá emitir hasta el ciclón *n* + 2 porque depende de la instrucción LD. La instrucción SUB es independiente de las instrucciones LD y ADD, así que podrá emitirse también en el ciclo *n* + 2 (su emisión no se puede adelantar al ciclo *n* + 1 o al ciclo *n* porque el procesador debe emitir las instrucciones en orden). La instrucción MUL es también independiente de todas las instrucciones anteriores, pero debe esperar hasta el ciclo *n* + 3 para comenzar su emisión, ya que el procesador solo puede emitir dos instrucciones por ciclo. Por lo tanto, se tardan 4 ciclos en emitir todas las instrucciones de la secuencia de código, así que podemos concluir que el tiempo total de ejecución es 5 + 4 – 1 = 8 ciclos.

2.6 Estimación del tiempo de ejecución para procesadores con ejecución fuera orden

Determinar el tiempo de emisión de una secuencia de instrucciones en un procesador superescalar con ejecución fuera de orden es bastante mas complejo que si la emisión es en orden, ya que suelen existir distintos ordenes en los que las instrucciones podrian ejecutarse. En general, el mejor método consiste en empezar por un análisis de la secuencia de instrucciones para localizar las dependencias entre instrucciones.

Una vez que se han identificado bien las dependencias, se pueden asignar a ciclos de emisión de forma que se minimice el retardo entre la ejecución de la primera y de la ultima instrucción de la secuencia.

El esfuerzo necesario para encontrar la mejor ordenación de un conjunto de instrucciones crece exponencialmente con el numero de instrucciones que forman el conjunto, ya que se deben considerar todas las posibles ordenaciones. Por tanto, supondremos que la lógica de emisión de instrucciones de un procesador superescalar impone algunas restricciones sobre el orden en el que las instrucciones se emiten, de forma que se simplifique su implementación. La hipótesis que utilizaremos es que el procesador emitirá una instrucción en el primer ciclo en el que las dependencias en el programa lo permitan 1. Si se pudieran emitir en un ciclo mas instrucciones que el numero de unidades de ejecución del procesador, la lógica de emisión utilizara una política avariciosa y emitirá aquellas instrucciones que ocurran antes en el programa, incluso si la emisión de las instrucciones en un orden diferente redujera el tiempo de ejecución de la secuencia de

instrucciones. Cuando el compilador es capaz de controlar cuando se deben emitir las instrucciones, como en el caso de los procesadores VLIW descritos en secciones anteriores, supondremos que el compilador considera todas las posibles ordenaciones de instrucciones para encontrar la que tiene un tiempo de ejecución menor, ya que el compilador puede emplear mas tiempo en la planificación de instrucciones que la lógica de emisión.

Con esta política avariciosa de emisión de instrucciones, encontrar el tiempo de ejecución de una secuencia de instrucciones en un procesador con ejecución fuera de orden se vuelve una tarea mucho más sencilla. Empezando por la primera instrucción de la secuencia, se va avanzando por las sucesivas instrucciones, asignando cada instrucción al primer ciclo en el que: todos sus operandos estén disponibles, el numero de instrucciones asignadas a dicho ciclo no supere el máximo de instrucciones que el procesador puede emitir por ciclo y el conjunto de instrucciones asignadas al ciclo no viole las limitaciones de las unidades de ejecución del procesador, incluso se emite una instrucción antes que ocurra en el programa con anterioridad. Repitiendo este proceso para todas las instrucciones de la secuencia se puede determinar cuánto tiempo tarda esta en emitirse.

EJEMPLO

¿Cuánto tiempo tardara en ejecutarse la siguiente secuencia de instrucciones en un procesador superescalar con ejecución fuera de orden y dos unidades de ejecución, pudiendo ejecutar cada una de ellas cualquier tipo de instrucción?
Las instrucciones de carga tienen una latencia de dos ciclos y todas las demás operaciones tienen una latencia de un ciclo (Esta es la misma secuencia usada en el ejemplo que ilustraba la ejecución en orden.)

```
LD r1, (r2)
ADD  r3, r1, r4
SUB r5, r6, r7
MUL  r8, r9, r10
```

SOLUCION

La única dependencia es esta secuencia esta entre la instrucción LD y la instrucción ADD (una dependencia RAW). Debido a esta dependencia, la instrucción ADD debe emitirse al menos dos ciclos después de la instrucción LD. Las instrucciones SUB y MUl podrían emitirse en el mismo ciclo que la instrucción LD, así que usando nuestra suposición avariciosa, las instrucciones LD y SUB se emitirán en el ciclo n + 2, obteniéndose un tiempo de emisión de tres ciclos para esta secuencia de instrucciones.

- CUESTIONES DE IMPLEMENTACION EN LOS PROCESADORES CON EJECUCION FUERA DE ORDEN

En los procesadores con ejecución en orden, la *ventana de instrucciones* (el numero de instrucciones que el procesador analiza para seleccionar las instrucciones que va a emitir en cada ciclo) puede ser relativamente pequeña, ya que el procesador no puede emitir una instrucción hasta que todas las instrucciones que aparecen con anterioridad en el programa hayan sido emitidas. En un procesador con n unidades de ejecución, solo

podrán emitirse, como máximo, las siguientes *n* instrucciones en el siguiente ciclo de reloj, así que generalmente es suficiente una ventana de *n* instrucciones.

Los procesadores con ejecución fuera de orden necesitan una ventana de instrucciones mucho mayor que los procesadores con ejecución en orden, para que tengan mas posibilidades de encontrar instrucciones que puedan emitirse en un determinado ciclo. Sin embargo, el tamaño de la lógica de emisión de instrucciones crece de forma cuadrática con el número de instrucciones de la ventana, ya que cada instrucción de la ventana debe ser comparada con todas las demás para determinar las dependencias entre ellas. Esto hace que las ventanas de instrucciones grandes sean muy costosas de implementar en hardware.

El procedimiento presentado antes para determinar el tiempo de ejecución de una secuencia instrucciones en un procesador con ejecución fuera de orden suponía que el procesador disponía de una ventana de instrucciones suficientemente grande como para permitir el análisis de todas las instrucciones de la secuencia simultáneamente. Si no es así, estimar el tiempo de ejecución se vuelve una tarea mucho mas difícil, ya que es necesario tener en cuenta que instrucciones se encuentran en la ventana de instrucciones de cada ciclo para seleccionar las que se van a emitir en dicho ciclo.

La gestión de interrupciones y excepciones del programa es otra cuestión difícil de implementar en los procesadores con ejecución fuera de orden. Si las instrucciones se pueden ejecutar fuera de orden, determinar exactamente que instrucciones se han ejecutado cuando ocurre una interrupción o cuando una instrucción provoca una excepción, se vuelve una tarea muy complicada.

Esto dificulta al programador a la hora de determinar la causa de una excepción y al sistema a la hora de retornar la ejecución al programa original tras haber completado la rutina de atención a una interrupción.

Para solucionar este problema, prácticamente todos los procesadores con ejecución fuera de orden usan técnica denominada *finalización en orden.* Cuando una instrucción genera un resultado, dicho resultado se escribe en el banco de registros solo si todas las instrucciones anteriores en el programa se han completado. Si esto no ocurre, el resultado se almacenara hasta que todas las instrucciones anteriores en el programa se hayan completado y será solo entonces cuando se escribirá el resultado en el banco de registros. Dado que los resultados se escriben en orden en el banco de registros, cuando ocurra una excepción el hardware puede descartar todos los resultados pendientes de ser escritos en el banco de registros. Este mecanismo les presenta a los programadores la ilusión de que las instrucciones estas siendo ejecutadas en orden, permitiéndoles depurar los errores de una forma relativamente sencilla y haciendo posible que el sistema pueda reanudar la ejecución del programa a partir de la siguiente instrucción cuando termina la rutina de atención a una interrupción. Los procesadores que utilizan esta técnica generalmente usan también una lógica de atajo u otras técnicas para anticipar el resultado de una instrucción a las instrucciones dependientes antes de que dicho resultado sea escrito en el banco de registros. Esto permite que las instrucciones dependientes se puedan emitir tan pronto como sus operandos estén disponibles, en vez de tener que esperar a que estos resultados se escriban en el banco de registros.

- Renombrado de registros

Las dependencias WAR y WAW también se conocen como “dependencias de nombre”, porque su origen esta en el hecho de que los programas se ven forzados a reutilizar los registros, dado el tamaño limitado del banco de registros. Estas dependencias pueden limitar el paralelismo entre instrucciones en los procesadores superescalares, ya que es necesario asegurarse de que todas las instrucciones que leen de un registro terminen su etapa de lectura antes de sobrescribir dicho registro.

El renombrado de registros es una técnica que reduce el impacto de las dependencias WAR y WAW en el paralelismo mediante la asignación dinámica de cada valor producido por una instrucción a un registro nuevo, rompiendo, por tanto, las dependencias WAR y WAW. La figura, ilustra la técnica de renombrado de registros. Cada repertorio de instrucciones esta definido con un *banco de registros de la arquitectura,* que es el conjunto de registros con el que las instrucciones puede operar. Todas las instrucciones especifican sus operandos de entrada y de salida de entre los registros de este conjunto. Sin embargo, en el procesador existe un banco de registros de la arquitectura, además de una lógica de renombrado que se encarga de asignar dinámicamente los registros de la arquitectura, referidos en las instrucciones, a registros en el banco de registros hardware del procesador.

Siempre que una instrucción lee de un registro en el banco de registros de la arquitectura, el identificador de este registro se envía a través de la lógica de renombrado para determinar a que registro del banco de registros hardware se debe acceder. Cuando una instrucción escribe en un registro del banco de la arquitectura, la lógica de renombrado crea una nueva asignación del registro de la arquitectura que ha sido escrito en un registro del banco hardware que no se encuentre en uso. Por tanto, las siguientes instrucciones que lean de este registro de la arquitectura accederán al registro del banco hardware recién asignado y podrán acceder al resultado de la instrucción.

La figura ilustra como se pueden mejorar las prestaciones mediante el renombrado de registros.

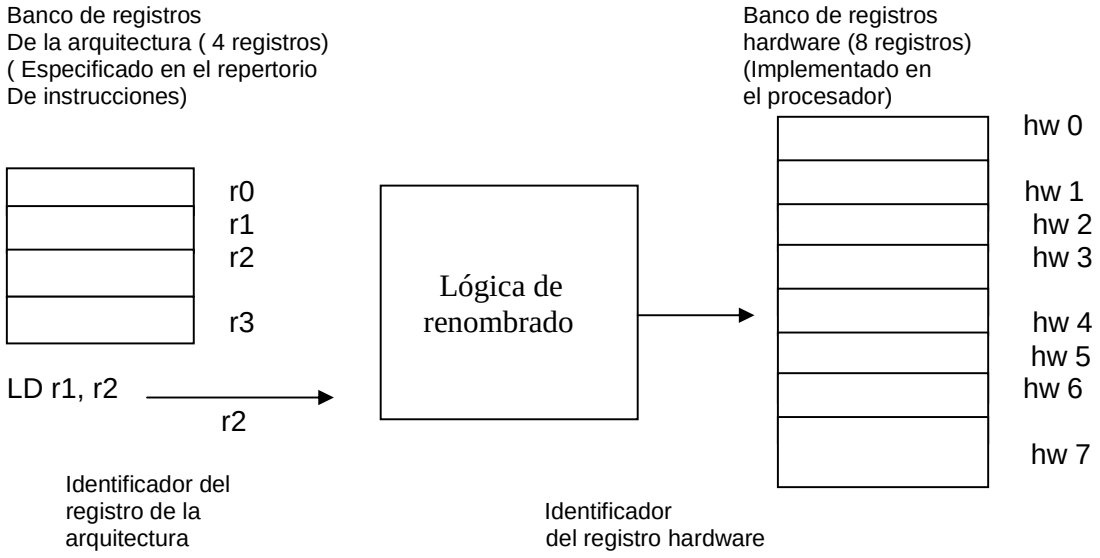


Figura: Renombrado de registros

En el programa original (antes de renombrar), existe una dependencia WAR y RAW del programa LD r7, (r3) y SUB r3, r12, r11. La combinación entre las dependencias WAR y RAW del programa hacen que su tiempo de emisión sea de al menos tres ciclos, ya que la instrucción LD debe emitirse después de la instrucción ADD, la instrucción SUB no puede emitirse antes de la instrucción LD y la instrucción ST debe esperar a la instrucción SUB.
Al renombrar los registros, la primera escritura a r3 se asigna al registro hardware hw3, mientras que la segunda escritura se asigna a hw20 (los registros del banco hardware se han escogido de forma arbitraria). Esta asignación convierte la cadena de dependencias original de cuatro instrucciones en dos cadenas de dos instrucciones, que pueden ser ejecutadas en paralelo se el procesador permite ejecución fuera de orden. En general, el renombrado de registros es de mayor utilidad en los procesadores con ejecución fuera de orden que en los procesadores con ejecución en orden, ya que los primeros pueden reordenar las instrucciones una vez el renombrado de registros ha roto las dependencias de nombre.

Antes de renombrar		Después de renombrar
ADD r3, r4, r5		ADD hw3, hw4, hw5
LD r7, (r3)		LD hw7, (hw3)
SUB r3, r12, r11	⟶	SUB hw20, hw12, hw11
ST (r15), r3		ST (hw15), hw20

EJEMPLO

Suponga un procesador superescalar con ejecución fuera de orden y 8 unidades de ejecución. ¿Cuánto tiempo tardara en ejecutarse la siguiente secuencia de instrucciones con y sin renombrado de registros si cada una de las unidades puede ejecutar cualquier instrucción y la latencia de todas las instrucciones es de un ciclo? Suponga que el banco de registros hardware es lo suficientemente grande como para alojar cada resultado de salida en un registro hardware distinto y que el cauce tiene una profundidad de 5 etapas.

LD r7, (r8)
MUL r1, r7, r2
SUB r7, r4, r5
ADD r9, r7, r8
LD r8, (r12)
DIV r10, r8, r10

SOLUCION

En este ejemplo, las dependencias WAR suponen una limitación importante en el paralelismo, forzando a que la instrucción DIV sea emitida 3 ciclos después de la primer instrucción LD, lo que hace que obtengamos un tiempo total de ejecución de 8 ciclos (las instrucciones MUL y SUB se pueden ejecutar en paralelo, al igual que las instrucciones ADD y la segunda LD). Después del renombrado de registros, el programa queda como:

LD hw7, (hw8)
MUL hw1, hw7, hw2

SUB hw17, hw4, hw5
ADD hw9, hw17, hw8
LD hw18, (hw12)
DIV hw10, hw18, hw10

(De nuevo todos los registros de renombrado se han escogido de forma arbitraria.) Al aplicar renombrado de registros, el fragmento de programa se ha dividido en tres conjuntos de dos instrucciones independientes (LD y MUL, SUB y ADD, LD y DIV). Las instrucciones SUB y segunda LD se pueden emitir ahora en el mismo ciclo que la primera LD. Las instrucciones MUL, ADD y DIV se pueden emitir todas en el siguiente ciclo, con lo que obtendremos un tiempo de ejecución de 6 ciclos.

Añadir renombrado de registros a un procesador generalmente una menor mejora que cambiar el repertorio de instrucciones del procesador para que incorpore los nuevos registros al banco de registros de la arquitectura, ya que el compilador no puede usar los nuevos registros para almacenar valores temporales. Sin embargo, el renombrado de registros permite que los nuevos procesadores mantengan la compatibilidad con programas compilados para versiones mas antiguas del procesador, ya que el repertorio de instrucciones no se ve afectado. Además, el incremento del numero de registros de la arquitectura supondría aumentar el numero de bits que ocupa una instrucción, ya que serian necesarios mas bits para codificar sus operandos y el registro de destino.

2.7. Procesadores VLIW

Los procesadores superescalares que hemos descrito en este capitulo utilizan hardware para aprovechar el paralelismo entre instrucciones, localizando las instrucciones que puede ejecutarse en paralelo dentro de un programa secuencial. Su habilidad para mejorar las prestaciones en la ejecución de programas antiguos y mantener la compatibilidad entre las generaciones de una familia de procesadores les ha hecho extremadamente populares comercialmente, pero a costa de incluir gran cantidad de hardware adicional. Los procesadores de palabra muy larga de instrucciones (VLIW, *Very Long Instruction Word)* aprovechan el paralelismo entre instrucciones de otro punto de vista, dependiendo del compilador para determinar que instrucciones se deben ejecutar en paralelo y proporcionar esta información de hardware.
En un procesador VLIW, cada instrucción especifica varias operaciones independientes que serán ejecutadas en paralelo por el hardware, tal y como se muestra en las figuras.
Cada operación en una instrucción VLIW equivale a una instrucción en un procesador superescalar o en un procesador puramente secuencial. El numero de operaciones en una instrucción VLIW es igual al numero de unidades de ejecución del procesador y cada operación especifica la instrucción que será ejecutada en su correspondiente unidad de ejecución en el ciclo en el que la instrucción se emita. Ya que el compilador es el responsable de asegurar que todas las operaciones que forman una instrucción VLIW se pueden ejecutar simultáneamente, no hay necesidad de ningún hardware que analice el flujo de instrucciones para determinar que instrucciones se pueden ejecutar en paralelo. Por tanto, la lógica de emisión de instrucciones en un procesador VLIW es mucho mas sencilla que en un procesador con el mismo numero de unidades de ejecución.
La mayoría de procesadores VLIW no disponen de marcadores en su banco de registros. En su lugar, el compilador es responsable de asegurar que una operación no se emita
Antes de que sus operandos esten disponibles. En cada ciclo, la lógica de emisión de instrucciones capta una instrucción VLIW de memoria y la emite a las unidades de

ejecución. Por tanto, el compilador puede predecir exactamente cuantos ciclos pasaran entre la ejecución de dos operaciones contando el numero de instrucciones VLIW entre ellas. Además, el compilador puede planificar instrucciones con una dependencia WAR fuera de orden siempre que la instrucción que lee el registro se emita antes de que se complete la instrucción que escribe en el registro, ya que el valor anterior del registro no será sobrescrito hasta que la instrucción que escribe en el registro se complete. Por ejemplo, en un procesador VLIW con instrucciones de carga de dos ciclos de latencia. La secuencia ADD r1, r2, r3, LD r2, (r4) se podrían planificar de forma que la operación ADD apareciera en la siguiente instrucción después de la carga, ya que la carga no sobrescribirá r2 hasta que hayan pasado dos ciclos.

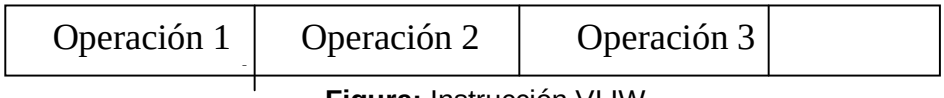


Figura: Instrucción VLIW

- PROS Y CONTRAS DE LAS ARQUITECTURAS VLIW.

Las ventajas principales de las arquitecturas VLIW son: Primera, que al ser su logica de emisión de instrucciones mas sencilla, se puede implementar con ciclos de reloj mas cortos que en los procesadores superescalares, y segunda, que el compilador controla completamente cuando se ejecutaran las instrucciones. El compilador generalmente tiene una visión mas global del programa que la lógica de emisión de instrucciones de un procesador superescalar y, por tanto, supera a la lógica de emisión a la hora de encontrar instrucciones que se puedan ejecutar en paralelo.
Por otra parte, al contar con una lógica de emisión de instrucciones mas sencilla, para un tamaño de chip determinado, es posible incorporar un numero mayor de unidades de ejecución en los procesadores VLIW que en los superescalares.
La desventaja mas importante de los procesadores VLIW es que los programas VLIW solo se ejecutaran correctamente en un procesador con el mismo numero de unidades de ejecución y las mismas latencias para las instrucciones que el procesador para el que fueron compilados, lo que hace prácticamente imposible mantener la compatibilidad entre distintas generaciones de procesadores de la misma familia. Si el numero de unidades de ejecución aumenta entre generaciones, el nuevo procesador tratara de combinar operaciones de varias instrucciones en cada ciclo, causando que operaciones dependientes se puedan ejecutar potencialmente en el mismo ciclo. Por otra parte, si las latencias de las operaciones cambian entre generaciones de una familia de procesadores, es posible que algunas operaciones se ejecuten antes de que sus operandos estén disponibles, o bien después de que sus operandos hayan sido sobrescritos por otra operación, dando lugar a un comportamiento incorrecto. Además, si el compilador no es capaz de encontrar un numero suficiente de operaciones independientes como para completar una instrucción VLIW, debe colocar explícitamente operaciones NOP (no operation) en los huecos para los que no se hayan encontrado una operación en el programa que se pueda ejecutar en ese ciclo de reloj. Esta ultima desventaja causa que un programa compilado para un procesador VLIW suela ocupar mas cantidad de memoria que si se compila para un procesador superescalar.
Debido a sus ventajas y desventajas, los procesadores VLIW se suelen emplear en aplicaciones de procesamiento digital de señales (DSP, Digital Signal Processing), en las que las altas prestaciones y el bajo coste son especificaciones criticas. Se han aplicado con menos éxito en computadores de propósito general como las estaciones de trabajo o

los PC, ya que los clientes demandan que el software sea compatible entre las diferentes generaciones de una familia de procesadores.

EJEMPLO
Muestre como planificaría un compilador la siguiente secuencia de instrucciones para un procesador VLIW con 3 unidades de ejecución. Suponga que todas las operaciones tienen una latencia de dos ciclos y que cualquier unidad de ejecución puede ejecutar cualquier tipo de instrucción.

ADD r1, r2, r3
SUB r16, r14, r7
LD r2, (r4)
LD r14, (r15)
MUL r5, r1, r9
ADD r9, r10, r11
SUB r12, r2, r14

SOLUCION
La figura muestra como se podrían planificar las operaciones. Obsérvese que la operación LD r14, (r15) se ha colocado en la instrucción anterior a la operación SUB r16, r14, r7 a pesar de que la operación SUB aparece en el fragmento de código original y de que lee el registro de destino en el que la operación LD almacenara el dato. Ya que las instrucciones VLIW no sobrescriben los registros hasta que no se completan, el valor previo de r14 permanecerá hasta dos ciclos después de que la operación LD se haya emitido, permitiendo que la operación SUB pueda leer el valor anterior de r14 y genere un resultado correcto. Al planificar las operaciones fuera de orden y de esta forma, se ha logrado que el fragmento de programa ocupe menos instrucciones VLIW. De manera similar, la operación ADD r9, r10, r11 se ha planificado antes que la operación MUL r5, r1, r9, aunque estas dos operaciones podrían haberse colocado en la misma instrucción sin incrementar el numero de instrucciones necesarias para codificar el fragmento de programa.

Instrucción 1	ADD r1, r2, r3	LD r2, (r4)	LD r14, (r15)
Instrucción 2	SUB r16, r14, r7	ADD r9, r10, r11	NOP
Instrucción 3	MUL r5, r1, r9	SUB r12, r2, r14	NOP

- Técnicas de compilación para mejorar el paralelismo entre instrucciones

Los compiladores emplean una gran variedad de técnicas para mejorar las prestaciones de los programas compilados, incluyendo la propagación de constantes, la eliminación del código muerto, y la asignación de registro. Aunque la descripción detallada de todas estas técnicas esta fuera del ámbito de este libro, esta sección cubrirá con detalle el desenrollado de bucles, una técnica que aumenta significativamente el paralelismo entre

instrucciones, y describirá brevemente la segmentación software, otra técnica de compilación empleada para mejorar las prestaciones.

- Desenrollado de bucles

Cada una de las iteraciones de un bucle suele contener poco paralelismo entre instrucciones debido a que a menudo incluye cadenas de instrucciones dependientes, y a que existe un numero limitado de instrucciones entre dos saltos. El desenrollado de bucles soluciona esta limitación transformando un bucle de *N/M* iteraciones, en el que cada una de las iteraciones del nuevo bucle se corresponde con *M* iteraciones del bucle original. Este cambio aumenta el numero de instrucciones entre dos saltos, incrementando las posibilidades del compilador o de la lógica de emisión de instrucciones de encontrar paralelismo entre instrucciones. Además, si las iteraciones del bucle original eran independientes o contenían solo unas pocas operaciones dependientes, el desenrollado de bucles puede generar muchas cadenas de instrucciones independientes donde solo existía una cadena antes de desenrollar, mejorando la capacidad del sistema de aprovechar el paralelismo entre instrucciones.
La figura muestra en ejemplo de desenrollando bucles en lenguaje C. El bucle original itera a través de los vectores de datos elemento por elemento, calculando la suma de los correspondientes elementos y almacenando el resultado en el vector de destino.

Bucle original

```
For ( i = 0; i < 100; i++) {
        a[i]=b[i]+c[i];
}
```


Bucle desenrollado

```
for  ( i = 0; i < 100; i+=2) {
        a[i]=b[i]+c[i];
a[i+1]=b[i+1]+c[i+1];
}
```

Figura. Ejemplo de desenrollado de bucles en lenguaje C.

Tras desenrollar este bucle procesamos dos elementos en cada iteración, haciendo el trabajo de dos de las iteraciones originales en una sola iteración. En este ejemplo, el bucle se ha desenrollado dos veces2. El bucle original se podría haber desenrollado cuatro si sumamos al contador del bucle *i* de 4 en 4 en cada iteración y si hacemos el trabajo de cuatro iteraciones originales en cada iteración del bucle desenrollado.
Este ejemplo también ilustra otra ventaja del desenrollado de bucles: la reducción en la sobrecarga de bucle. Al desenrollar el bucle original dos veces, se reduce el numero de iteraciones de 100 a 50. Esto reduce a la mitad el numero de instrucciones de salto condicional que se deben ejecutar al final de cada iteración, reduciendo el numero total de instrucciones que es necesario ejecutar además de mejorar el paralelismo entre instrucciones. Por tanto, el desenrollado de bucles puede ser beneficioso incluso en procesadores puramente secuenciales, aunque sus beneficios se aprecian bastante mas en los procesadores LP.
La figura 1 muestra como se podrían implementar el bucle de la figura 2 en ensamblador, y como podrían el compilador desenrollar el bucle y planificarlo para que se ejecutara eficientemente en un procesador superescalar con tipos de datos de 32 bits. Incluso en el bucle original, el compilador ha planificado el código para descubrir el mayor ILP posible al colocar las dos cargas iníciales al principio, por delante incluso de las instrucciones

ADD que implementa punteros y adelantando los incrementos de los punteros a la instrucción ADD que implementa *a[i]=b[i]+c[i].* Colocando las instrucciones independientes cerca entre instrucciones, mientras que del procesador superescalar la tarea de localizar el paralelismo entre instrucciones, mientras al colocar los incrementos de los punteros entre la carga de los datos y el calculo de *a[i]* aumenta el numero de operaciones entre las operaciones de carga y el uso de sus resultados, aumentando la probabilidad de que las instrucciones de carga se hayan completado cuando sus resultados sean necesarios.
El bucle desenrollado comienza con tres sumas para generar punteros a a[i+1], b[i+1] y c[i+1]. Al mantener estos punteros en registros diferentes de los que apuntan a a[i], b[i] y c[i] se permite que las cargas y los almacenamientos de los elementos en las posiciones *i e i* + 1 de cada vector se puedan realizar en paralelo, en vez de tener que incrementar cada puntero entre dos referencias a memoria consecutivas. Este bloque de instrucciones de inicialización para el bucle desenrollado se llama *preámbulo* del bucle. En el cuerpo del bucle, el compilador ha colocado todas las cargas en un bloque, seguido por todos los incrementos de los punteros, tras los que ha colocado el calculo de *a[i] y a[i + 1],* y finalmente los almacenamientos y el salto hacia atrás. Esta planificación maximiza tanto el paralelismo como el tiempo necesario para que las cargas de memoria se completen.

Se dice que un bucle se ha desenrollado n veces si cada litaracion del bucle desenrollado realiza el trabajo de n literaciones del bucle original.
En este ejemplo, el numero de iteraciones del bucle original era divisible entre el grado de desenrollado del bucle, facilitando el desenrollado. En muchos caso, sin embargo, el numero de iteraciones del bucle no es divisible por el grado de desenrollado, o incluso es desconocido en tiempo de compilación, dificultando el desenrollado del bucle. Por ejemplo, el compilador podría desenrollar el bucle de la figura 3 ocho veces, o bien el numero de iteraciones del bucle podría ser un parámetro de entrada del procedimiento que contiene el bucle.
En estos casos, el compilador tendrá dos bucles. El primer bucle es una versión desenrollada del bucle original que se ejecuta hasta que el numero de iteraciones que faltan sea menor que el grado de desenrollado aplicado. Entonces se ejecuta el segundo bucle y termina una a una el resto de iteraciones que faltan para completar el bucle original. En el caso de que no se conozca a priori el numero de iteraciones del bucle, como el ejemplo en el caso de un bucle que itere a través de una cadena de caracteres hasta que encuentre el carácter de fin de cadena, el desenrollado del bucle es mucho mas difícil y se necesitan técnicas mas sofisticadas.
La figura 5 muestra como se desenrorollaria el bucle de la figura anterior ocho veces. El primer bucle procesa los elementos de 8 en 8 hasta que quedan menos de 8 elementos por procesar (esta condición se detecta si *i* + 8 >=100). El segundo bucle comienza en la siguiente iteración y procesa el resto de elementos uno a uno. Puesto que *i* es una variable de tipo entero, el calculo *i* = ((100/8) * 8) no fija su valor a 100. La operación 100/8 genera solo la parte entera de la división (despreciando el resto), y multiplicando este resultado por 8 obtendremos el mayor múltiplo de 8 que esta por debajo de 100 (96), que es la posición en la que el segundo bucle debería comenzar sus iteraciones.

- SEGMENTACION DE CAUCE SOFTWARE

El desenrollado de bucles mejora las prestaciones al incrementar el numero de operaciones independientes dentro de una iteración del bucle. Por el contrario, la *segmentación de cauce software* mejora las prestaciones mediante la distribución de cada iteración del bucle original entre varias iteraciones del bucle segmentado, de forma que

cada iteración del nuevo bucle haga algo del trabajo de múltiples iteraciones del bucle original. Por ejemplo, un bucle que cargue *b[i] y c[i]* de memoria, los sume para generar *a[i],* y se escriba *a[i]* en memoria, se podría transformar de forma que en cada iteración se escribiera primero *a[i* - 1*]* en memoria, después se calculara *a[i]* basándose en los valores de *b[i + 1]* y *c[i + 1]* para preparar la siguiente iteración. Por tanto, el trabajo de calcular un elemento del vector *a* se distribuye entre tres iteraciones del nuevo bucle.
Al intercalar porciones de diferentes iteraciones del bucle original de esta forma, se aumenta el paralelismo entre instrucciones tanto como con el desenrollado de bucles. También se aumenta el numero de instrucciones entre el calculo de un valor y su uso, haciendo mas probable que el valor este preparado cuando sea necesario. Muchos compiladores combinan la segmentación software con el desenrollado de bucles mas paralelismo entre instrucciones que si se aplicara tan solo una de las dos técnicas.

Autoevaluación.

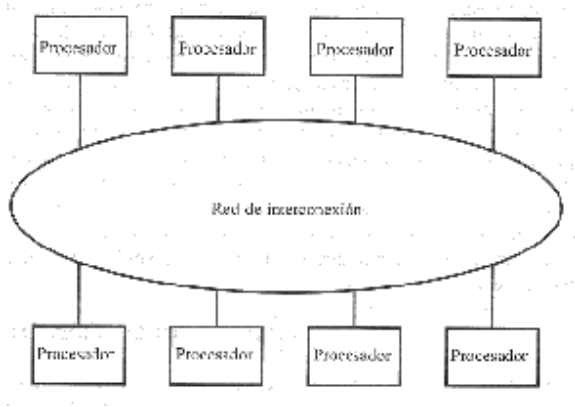
1. Qué es el paralelismo entre instrucciones? Còmo lo aprovechan los procesadores para mejorar las prestaciones?

2. Limitaciones del paralelismo

Capítulo 3: Sistemas Multiprocesador

3.1. Características

Los sistemas multiprocesador consisten en un conjunto de procesadores conectados mediante una red de comunicaciones, tal y como se muestra en la Figura. Los primeros sistemas multiprocesador solían usar procesadores específicamente diseñados para mejorar las prestaciones del sistema multiprocesador, aunque esta tendencia está cambiando recientemente, y la mayoría de los sistemas multiprocesador están basados en los mismos procesadores que se usan para construir computadores personales o estaciones de trabajo, beneficiándose del gran volumen de ventas que tienen estos últimos procesadores para bajar el precio del sistema. A medida que se incrementa el número de transistores que se pueden integrar en un chip, se están empezando a incorporar en los procesadores de propósito general algunas características para dar soporte al procesamiento multiprocesador, lo que facilita la construcción de sistemas multiprocesador basados en estos procesadores.
El diseño de las redes de interconexión de los sistemas multiprocesador es un tema que está fuera del los propósitos de este libro, así que en este capítulo trataremos la red de interconexión como una «caja negra» que permite que cualquier procesador se pueda comunicar con cualquier otro, ignorando los detalles de cómo se lleva a cabo dicha conexión.
Los sistemas multiprocesador pueden tener un sistema de memoria centralizado o distribuido.
En el caso de un sistema de memoria centralizado, solamente hay un sistema de memoria para todo el sistema multiprocesador al que se dirigen las referencias de memoria de todos los procesadores del sistema. En un sistema con memoria distribuida, cada procesador tiene su propio sistema de memoria, al que puede acceder directamente. Si un procesador desea acceder a algún dato que se encuentre en la memoria de otro procesador, el primero debe comunicarse con le segundo para pedirle el dato.



Los sistemas con memoria centralizada tienen la ventaja de que todos'los datos contenidos en la memoria son accesibles por todos los procesadores, y de que nunca habrá problemas de consistencia entre múltiples copias de un mismo dato. Sin embargo, el ancho de banda del sistema de memoria centralizado no crece conforme se aumenta el número de procesadores del sistema, además de que la latencia de la red de interconexión se suma a la latencia de cada referencia a memoria. Para aliviar estas limitaciones, la mayoría de los sistemas multiprocesador con memoria centralizada disponen de una memoria cache local para cada procesador, de forma que sólo envían a través de la red de interconexión aquellas peticiones de acceso a memoria que fallan en su cacheo Las peticiones que aciertan en la cache se atienden eficientemente y no originan tráfico por la red de interconexión, reduciendo la congestión de la misma y permitiendo que el sistema de memoria pueda soportar un número mayor de procesadores.
Los sistemas con memoria distribuida tienen la ventaja de que cada procesador tiene su propio sistema de memoria local. Esto implica que el ancho de banda total del sistema de

memoria es mayor que en un sistema con memoria centralizada y que la latencia para completar una petición de acceso a memoria es menor, ya que la memoria de cada procesador está colocada físicamente cerca de él. Sin embargo, estos sistemas tienen el inconveniente de que no todos los datos son accesibles directamente por cada procesador, ya que un procesador sólo puede leer y escribir en su sistema de memoria local. Para acceder a los datos de la memoria de otro procesador es necesario realizar una comunicación a través de la red de interconexión. Además, existe la posibilidad de que puedan existir dos o más copias de un mismo dato en distintas memorias locales, lo que puede causar que diferentes procesadores puedan tener diferentes valores para una misma variable. Este inconveniente se conoce como el problema de la *coherencia de memoria* y es una de las mayores fuentes de complejidad de los sistemas con memoria distribuida. Los sistemas de paso de mensajes, no tienen el mismo problema de coherencia que los sistemas con memoria distribuida, ya que no permiten que un procesador pueda leer o escribir en la memoria de otro procesador.
Los sistemas con memoria centralizada son a menudo la mejor opción cuando el número de procesadores es pequeño. Para este tipo de sistemas, un solo sistema de memoria puede satisfacer las demandas de ancho de banda de los procesadores, sobre todo si cada procesador dispone de una memoria cache local. Si éste es el caso, la simplificación de su diseño y la reducción en la complejidad de su programación, derivadas de no tener que gestionar múltiples memorias independientes, son un fuerte argumento a favor de este tipo de sistema de memoria. Sin embargo, conforme el número de procesadores crece, satisfacer el ancho de banda demandado por los procesadores se vuelve imposible para un sistema de memoria centralizado, con lo que se hace necesaria la utilización de un sistema de memoria distribuido. Estos sistemas también se usan cuando la latencia de la red de interconexión es tan grande que el uso de un sistema centralizado de memoria implicaría unas latencias de memoria inaceptablemente largas, incluso si el sistema de memoria centralizado pudiera alcanzar el ancho de banda demandado por los procesadores.

3.2. Sistemas de paso de mensajes

Principalmente existen dos modelos de programación para los sistemas multiprocesador: memoria compartida y paso de mensajes. En los sistemas de memoria compartida, el sistema de memoria gestiona la comunicación entre procesadores al permitir que todos los procesadores vean los datos escritos por cualquier procesador. Por el contrario, en los sistemas de paso de mensajes la comunicación se realiza mediante mensajes explícitos. Para mandar un mensaje, un procesador ejecuta explícitamente la operación ENVIAR(datos, destino), (que es generalmente una llamada a un procedimiento) para indicar al hardware que mande los datos especificados al procesador de destino. Posteriormente, el procesador de destino ejecuta la operación *RECIBIR(buffer)* para copiar los datos enviados en el *buffer* especificado, para que estén disponibles para su uso. Si el procesador que debe mandar los datos no hubiera ejecutado la operación ENVIAR antes de que el procesador de destino ejecute la operación RECIBIR, la operación RECIBIR esperará hasta que la operación ENVIAR se complete, de forma que siempre se mantenga el orden entre las operaciones ENVIAR y RECIBIR.

En los sistemas de paso de mensajes cada procesador tiene su propio espacio de direcciones Y los procesadores no pueden leer ni escribir datos en el espacio de direcciones de otro procesador. Por esta característica, muchos sistemas de paso de mensajes se implementan mediante máquinas de memoria distribuida, ya que se pueden

aprovechar de la disminución de la latencia obtenida, al asociar una memoria con cada procesador mientras evitan la complejidad de permitir que procesadores puedan acceder a otra memoria que no sea la suya.

3.3. Sistemas de memoria compartida

En los sistemas de memoria compartida la comunicación es implícita, al contrario que en los de paso de mensajes. Los sistemas de memoria compartida disponen de un único espacio de direcciones en el que todos los procesadores pueden leer y escribir. Cuando un procesador escribe en Una posición de memoria, cualquier lectura posterior de dicha posición de memoria hecha por cualquier procesador verá el resultado de la escritura, El sistema realizara las comunicaciones que sean necesarias para la operación sea visible por todos los procesadores.

3.4. Comparación entre el paso de mensajes y la memoria compartida

Es bastante probable que el paso de mensajes y la memoria compartida continúen siendo los paradigmas de comunicación dominantes en el futuro inmediato. Cada uno de ellos tiene sus ventajas y sus desventajas, de forma que no se puede decir que ninguno de ellos sea mejor que el otro. La ventaja principal de los sistemas de memoria compartida es que el computador se encarga de gestionar las comunicaciones, 10que hace posible la escritura de programas paralelos sin preocuparse realmente de cuándo se debe comunicar un dato de un procesador a otro. Sin embargo, para obtener unas buenas prestaciones, el programador debe tener en cuenta el uso que los procesadores hacen de los datos para minimizar las comunicaciones entre los procesadores (peticiones, anulaciones y actualizaciones). Este hecho es todavía más importante en los sistemas con memoria distribuida, ya que el programador también debería estudiar en la memoria de qué procesador se debería colocar la copia principal de cada dato El inconveniente de los sistemas de memoria compartida es que la capacidad del programador para controlar la comunicación entre los procesadores está limitada, ya que todas las comunicaciones son gestionadas por el sistema. Por ejemplo, en muchos sistemas la transferencia de un bloque grande de datos entre dos procesadores es más eficiente si se realiza como una sola comunicación, lo que no es posible en los sistemas de memoria centralizada porque el hardware controla la cantidad de datos que pueden transferirse en cada operación. El sistema también controla cuándo ocurren las comunicaciones, dificultando algunas optimizaciones como el envío de datos a un procesador que sabemos que los necesitará más tarde para evitarle la petición de los mismos y la espera hasta que los datos lleguen.
Los sistemas de paso de mensajes pueden obtener mejores prestaciones que los sistemas de memoria compartida al permitir que el programador controle las comunicaciones entre los procesadores, pero al precio de que el programador debe especificar explícitamente todas las comunicaciones del programa. El control de las comunicaciones mejora las prestaciones al per- mitir que un dato pueda ser enviado en el momento en el que esté disponible, en vez de cuando sea solicitado, y al ajustar la cantidad de datos que se mandan en cada mensaje a las necesidades de la aplicación, en vez de al tamaño de la línea de cache del sistema en el que se está ejecutando la aplicación. Además, en los sistemas de paso de mensajes se utilizan menos operaciones de sincronización que en los de memoria compartida, ya que las operaciones ENVIAR y RECIBIR proporcionan una gran parte de la sincronización necesaria por sí mismas. En general, los sistemas de paso de mensajes son bastante atractivos para muchas

aplicaciones científicas, debido a que la estructura regular de estos programas facilita la elección de qué datos deben ser comunicados entre los procesadores. Los sistemas de memoria compartida son más atractivos para aplicaciones irregulares, en las que es bastante difícil determinar qué datos deben ser comunicados entre los procesadores a la hora de escribir el programa. Coma cada modelo de programación es más adecuado a un conjunto de aplicaciones diferente, muchos sistemas multiprocesador están empezando a ofrecer soporte para ambos paradigmas en el mismo sistema. Esto permite a los programadores escoger el modelo de programación que mejor se ajuste a su aplicación. También permite la *paralelización incremental* de una aplicación, en la que al principio se escribe la aplicación en modo memoria compartida para reducir el tiempo de desarrollo, y posteriormente, una vez que el programa funciona correctamente, las comunicaciones críticas se rescriben usando paso de mensajes para mejorar las prestaciones, permitiéndole al programador alcanzar un compromiso entre la facilidad de implementación de la memoria compartida y las prestaciones del paso de mensajes.

Autoevaluación.

1. Realizar ejercicios de sincronización
2. Realizar ejercicios de balanceo de carga
3. Realizar ejercicios de protocolo MESI
4. Realizar ejercicios de memoria compartida frente a paso de mensaje