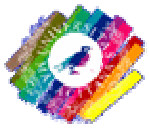
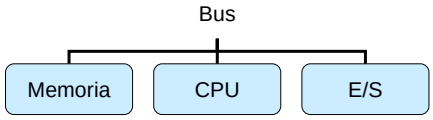
Tema 1: Introducción



- Arquitectura de Von Neumann
- Medida del rendimiento
- Jerarquía de memoria
- Técnicas para reducir el nº ciclos por instrucción
- Diseño del conjunto de instrucciones (ISA)

1.1 Arquitectura de Von Neumann

CARACTERÍSTICAS



- Memoria: almacena el programa y los datos.
- Procesador (CPU): interpreta las instrucciones
  - ✓ *Camino de datos.*
  - ✓ *Control. Las instrucciones operan sobre datos escalares.*
  - ✓ El secuenciamiento de las instrucciones es implícito (secuencial) y en algunos casos explícito (salto).
- Dispositivos de E/S

1.1.1 Evolución de los computadores

- Aparecen 3 segmentos de mercado, caracterizados por diferentes aplicaciones, requerimientos y tecnologías.
  - ✓ Computadores personales
    - Orientado a la optimización de la relación precio/prestaciones
    - Bien caracterizado en términos de aplicaciones y benchmarks.
  - ✓ Servidores
    - Orientado a la optimización del throughput.
    - Requerimientos: disponibilidad y tolerancia a fallos (p.e. Servidores Web).
    - Características: diseñados para que sean escalables.
  - ✓ Sistemas empotrados
    - Orientado a diseños de mínimo precio.
    - Requerimientos: ejecución de aplicaciones en tiempo real.
    - Características: diseñados para minimizar memoria y el consumo de potencia.

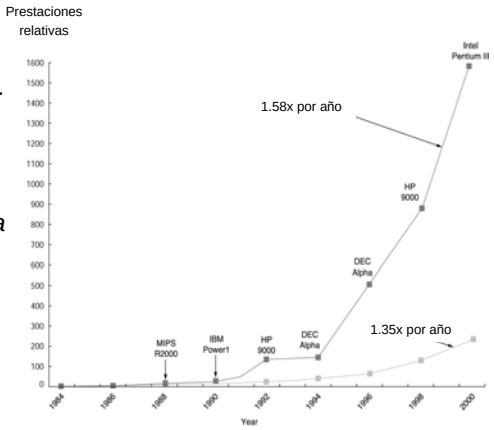
1.1.2 Tarea de los diseñadores de computadores

- Identificar los requerimientos y características de la nueva máquina y diseñarla de manera que maximice las prestaciones al coste permitido.
  - ✓ Computadores personales
    - Diseño de microprocesadores más potentes
    - Diseño de sistemas gráficos y sistemas de E/S que se integren con los microprocesadores de última generación.
  - ✓ Servidores
    - Integrar los microprocesadores de última generación en arquitecturas multiprocesador.
    - Diseño de sistemas de E/S con alta tolerancia a fallos y escalables.
  - ✓ Sistemas empotrados
    - Conseguir la máxima prestación al mínimo precio.
    - Diseño de equipos de consumo mínimo.

1.1.3 Factores que influyen en las prestaciones de un computador

1. Tecnología de diseño.
   - ✓ *35% de incremento cada año.*
2. Organización (arquitectura).
   - ✓ *A partir de los años 80*
   - ✓ *Nuevas arquitecturas (estructuras) han acelerado la mejora en el rendimiento.*

- Modificaciones estructurales son necesarias para obtener grandes mejoras de rendimt.
  - ✓ Un Intel 8086 no puede funcionar a 3GHz aunque se implemente a 0.13micras.



Fuente: Comput. Architecture. Hennessy y Patterson

1.1.4 Tendencias en la tecnología

- CPU: Incrementos en el tamaño del circuito y aumento de la densidad de integración.
  - ✓ *Crecimiento del número de transistores de aprox. 55% cada año.*
  - ✓ *No mejora la tecnología usada para el cableado -> marca un mejora más lenta de los tiempo de ciclo.*
  - ✓ *El incremento en el número de transistores y la frecuencia de conmutación, provocan un aumento en el consumo de potencia (p.e. Pentium IV a 2 GHz consume 100 Watios!).*
- Memorias DRAM.
  - ✓ *Incremento de la densidad entre 40% y 60% cada año.*
  - ✓ *Ley de Moore: el nº de TRT's por chip se dobla cada 18 meses*
  - ✓ *Tiempo de ciclo ha mejorado sólo 1/3 en diez años.*
  - ✓ *Incremento del ancho de banda.*

1.1.4 Tendencias en la tecnología (2)

- Discos magnéticos.
  - ✓ *Mejora en la densidad de 100% cada año (a partir de los 90).*
  - ✓ *Tiempo de acceso ha mejorado un tercio en diez años.*
- Tecnología de redes
  - ✓ Mejoras en latencia y ancho de banda en los últimos años.
  - ✓ El esfuerzo se ha centrado en el aumento del ancho de banda:
    - Ethernet tardó 10 años en pasar de 10 Mb a 100 Mb.
    - En 5 años se ha alcanzado 1Gb.
  - ✓ El éxito de Internet ha provocado el uso masivo de tecnología óptica y el desarrollo de conmutadores más potentes.

Ley de Moore

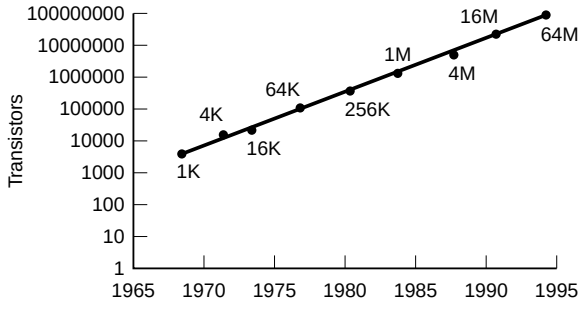


Figure 1-8. Moore's law predicts a 60 percent annual increase in the number of transistors that can be put on a chip. The data points given in this figure are memory sizes, in bits.

Fuente: Structured Comput. Organization. A. Tanenbaum

Ley de Moore

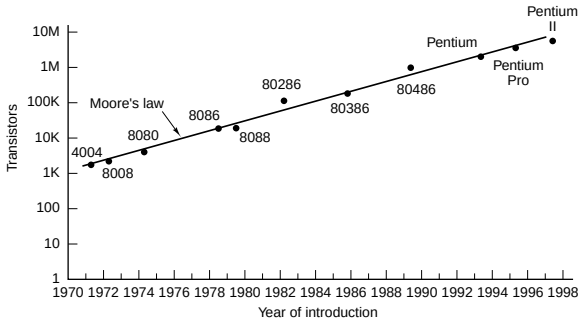


Figure 1-11. Moore's law for CPU chips.

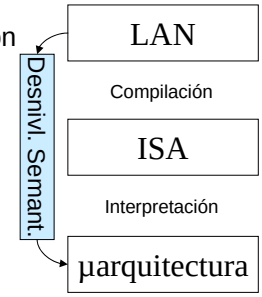
Fuente: Structured Comput. Organization. A. Tanenbaum

1.1.5 Tendencias en la organización

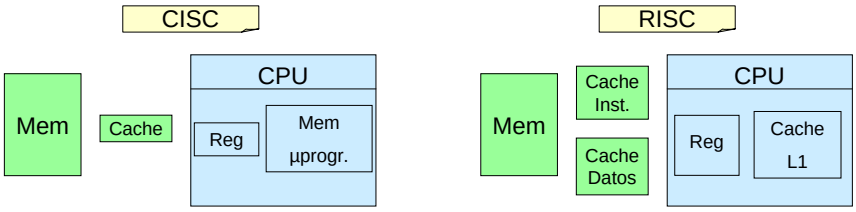
- Arquitecturas RISC.
- Jerarquía de la memoria.
- Replicación de unidades funcionales.
- Planificación dinámica de las instrucciones.
- Multiprocesadores.
- Tendencias en el software (multimedia, GUI, ...)
- Mejora en la tecnología de los compiladores.
  - ✓ *Mayor papel en el aprovechamiento de una organización de computador.*
  - ✓ *Reorganización de código, prefetching, etc.*

1.1.6. Arquitectura RISC

- El desnivel semántico entre LAN y µarquitectura se resuelve usando distintos niveles de traducción
- Dos filosofías:
  - ✓ CISC: El nivel ISA es parecido al LAN
    - Programación en ensamblador y compilador sencillos
    - Microarquitectura complicada: microprogramación
      - Unas microrutinas (SW) interpretan las instrucciones
  - ✓ RISC: El nivel ISA es parecido a la µarquitectura
    - Programación ensamblador y compilador complicados
    - Microarquitectura simple: control cableado
      - El HW directamente interpreta las instrucciones



1.1.6. Arquitectura RISC (2)



- Ocupa muchos TRTs en la memoria de microprograma
- Substituye la µmemoria por cache interna y más registros

Motivos para la migración al RISC

- ✓ Los LAN y la compilación evitan tener que programar en ASM
- ✓ Los compiladores prefieren usar inst. maquina sencillas. Las operaciones complejas se construyen con varias inst. sencillas.
- ✓ Los ejecutables ocupan más memoria, pero las memorias son mas baratas y el tiempo de acceso a memoria no es tan significativo gracias a la cache.

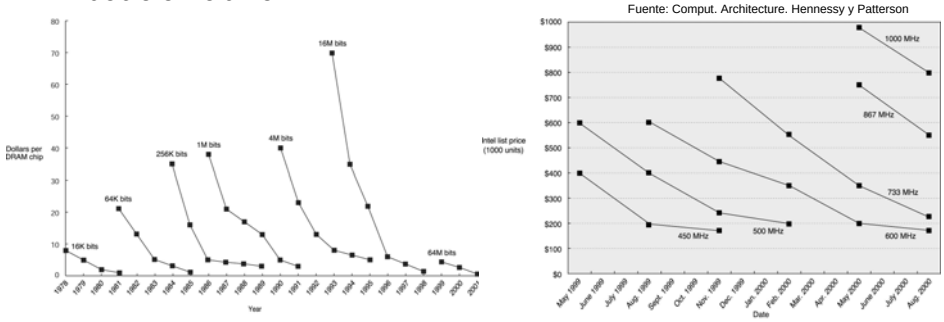
1.1.6. Arquitectura RISC (3)

- CISC
  - ✓ Conjunto de instrucciones completo o complejo
  - ✓ Control microprogramado
  - ✓ Instrucciones complejas y lentas
  - ✓ Ejecutables en memoria ocupan menos
  - ✓ La memoria de microprograma ocuapa mucho espacio y es lenta
  - ✓ Las instrucciones maquina son interpretadas por microrutinas
  - ✓ Aun usado en:
    - Controladores periféricos
    - Unidades punto flotante
- RISC
  - ✓ Conjunto de instrucciones reducido
  - ✓ Control cableado
  - ✓ Instrucciones sencillas y rápidas
  - ✓ Los ejecutables ocupan más memoria
  - ✓ Arquitectura Load/Store
  - ✓ Formato instrucciones fijo
  - ✓ Micromemoria substituida por registros y caches internas
  - ✓ La compilación para RISC es mas compleja y relevante
  - ✓ Simplifica la ejecución segmentada
  - ✓ Usado en todos los procesadores modernos

1.1.7 Factores que influyen en el coste del diseño de un computador

- *El coste de los componentes decrece con el tiempo, debido a la mejora de rendimiento en la producción.*
- *La competición entre vendedores hace que el precio de un componente se acerque a su coste real.*
- *El coste de un componente se reduce en un 10% cada vez que se doble el volumen.*

Fuente: Comput. Architecture. Hennessy y Patterson



1.2 Medida del rendimiento

- Para optimizar el rendimiento primero hay que saber medirlo
- Tiempo de latencia de una tarea ("*elapsed time*")
  - ✓ Tiempo desde que lanzas el programa hasta que acaba
  - ✓ Incluye computación, operaciones E/S, ejecución de otros proc., SO.
  - ✓ Dependiente de la carga del sistema.
- Ejemplo: ejecución en Unix del comando "time programa.exe"
  - ✓ Se ejecuta el programa.exe y "time" genera información de tiempo:
  - ✓ Por ejemplo: 90.7u 12.9s 2:39e 65%
    - Tiempo de usuario: 90.7seg. Tiempo en ejecutar inst. del programa
    - Tiempo de sistema: 12.9seg. T. en ejecutar inst de rutinas del SO llamadas desde el prog.
    - Tiempo total de CPU = 90.7+12.9 = 103.6seg
    - Elapsed time: 2:39 = 159seg.
    - Porcentaje de tiempo que el proceso (programa) ha usado la CPU = 65% = (103.6/159) x 100
    - El resto del tiempo = 159 – 103.6 = 55.4 seg, el proceso ha esperado por E/S o a otros procesos

1.2 Medida del rendimiento

- Si queremos medir el rendimiento de la CPU
  - ✓ Usar como medida el tiempo de CPU
- Tiempo de CPU (T_{CPU})
  - ✓ Incluye Tiempo CPU usuario + Tiempo CPU sistema.

$$T_{CPU} = N \times CPI \times t_{ciclo} = \frac{N \times CPI}{F}$$

- CPI: Nº de ciclos de reloj PROGRAMA / nº instrucciones ejecutadas

$$CPI = \frac{\sum_i CPI_i \times N_i}{N}$$

  - ✓ CPI: Número medio de ciclos de reloj por instrucción.
  - ✓ N_i : nº inst. tipo i.
  - ✓ CPI_i : CPI de la inst. tipo i.

1.2 Medida del rendimiento
Influencia entre parámetros

- Ejemplo: Compilamos y ejecutamos un programa en dos CPUs:
  - CPU A: Instrucciones separadas de comparación y salto: CMP AX,BX; JEQ label
    CPU B: La comparación está incluida en el salto: JEQ AX,BX,label
  - En las dos CPUs: Salto condicional: 2 ciclos. Resto instrucciones: 1 ciclo.
  - t_{cicloB} 25% más largo que t_{cicloA} (para que en 2ck se ejecute JEQ AX,BX,label)
  - El programa compilado para CPU A: 20% instrucciones salto condicional. Por tanto otro 20% son comparaciones y el resto son otras instrucciones: total N inst.
  - El programa compilado para CPU B tiene 0.8N inst. ya que la comparación y el salto se combinan en una única instrucción: de cada 100 instrucciones CPU A, 20 son de salto y 60 no salto, lo que supone un 25% de instrucciones de salto.

    $T_{CPU\ A} = N \times (0.2 \times 2 + 0.8 \times 1)\ t_{ciclo\ A} = 1.2\ N\ t_{cicloA}$

    $T_{CPU\ B} = 0.8 \times N \times (0.25 \times 2 + 0.75 \times 1) \times 1.25\ t_{ciclo\ A} = 1.25\ N\ t_{ciclo\ A}$
- Es decir, CPU A (q. ejecuta + instrucciones) es + rápida que CPU B.

1.2. Medida del rendimiento

$$T_{CPU}(\mu s) = N \times CPI \times t_{ciclo} = \frac{N \times CPI}{F(MHz)}$$

- Los factores están íntimamente relacionados:
  - *N* ➔ compilador y lenguaje máquina.
  - *CPI* ➔ lenguaje máquina y organización.
    - Se mide en la práctica por simulación y contadores HW.
  - t_{ciclo} ➔ organización y tecnología.
- El compromiso entre *N, CPI* y t_{ciclo} ha guiado el desarrollo de las nueva arquitecturas en los últimos años.

	N	*CPI*	*F(MHz)*	$T_{CPU}(\mu s)$
i386	1000	4.5	25	180
R3000	1200	1.25	25	60

1.2 Medida del rendimiento

- Otras medidas:
  - ✓ MIPS: Millones de instrucciones por segundo

$$MIPS = \frac{N}{T_{CPU}} \cdot 10^{-6} = \frac{N}{\frac{N \times CPI}{F}} \cdot 10^{-6} = \frac{F}{CPI} \cdot 10^{-6}$$

    - Inconvenientes:
      - Depende del nivel ISA: RISC tiene CPI bajo y CISC tiene CPI alto
      - Un mismo programa que tarde igual al ejecutarse en un CISC que en un RISC tiene mas MIPS en el RISC
  - ✓ MFLOPS: Millones de operaciones flotantes por segundo
    - Intenta evitar la dependencia con el nivel ISA
    - Cuenta sólo operaciones en punto flotante en lugar de instrucciones
    - Inconvenientes:
      - Solo mide las prestaciones del coprocesador matemático (FPU)
      - No distingue entre las distintas op. punto flotante: cuenta igual una + que una ÷
      - Un programa con 1000 sumas tiene mas MFLOPS que otro con 1000 divisiones

1.2.1 Comparación de rendimientos

- La mejor medida es el tiempo. Pero depende del programa.
- Uso de baterías de programas comunes para la comparación.
  - ✓ *Computadores personales*
    - *Interesa el rendimiento de la CPU y el rend. del sistema gráfico ➔ T_{CPU}.*
    - *Benchmarks para el rend. de CPU: SPEC CPU2000. Ver www.spec.org*
    - *Benchmarks para el rend. del sistema gráfico: SPECviewperf.*
  - ✓ *Servidores*
    - *Interesa el rendimiento del sistema de ficheros, de servidores Web y bases de datos ➔ número de transacciones por segundo.*
    - *Benchmarks para el rend. del sistema de ficheros: SPECSFS*
    - *Benchmarks para el rend. de servidores Web: SPECWeb, TPC-W*
    - *Benchmarks para el rend. de bases de datos: TPC. Ver www.tpc.org*
  - ✓ *Sistemas empotrados*
    - *Interesa el rendimiento de la aplicación para la que se desarrolla el sistema.*
    - *Bechmarks con kernels de aplicaciones típicas: EEMBC. Ver www.eembc.org*

1.2.1 Comparación de rendimientos (2) SPEC 2000

Año	Procesador	CINT2000	CFP2000
2000	Alpha 21264 833MHz	540	662
2000	Pentium III 1GHz	442	335
2000	Athlon 1.1GHz		331
2001	Alpha 21264 1GHz	621	756
2001	Pentium 4 2GHz	656	714
2001	Itanium 800MHz	314	645
2002	Alpha 21264C 1.25GHz	928	1364
2002	Pentium 4 2.8Ghz	984	929
2002	Itanium II 1GHz	807	1356
2002	Athlon XP 2600+ (2.13GHz)	839	710
2003	Pentium 4 3.2GHz	1249	1285
2003	Itanium II 1.5GHz	1332	2119
2003	Athlon FX-51 2.2GHz	1447	1423
2004	Athlon 64 2.4GHz	1717	1634
2004	Pentium 4 3.6GHz	1553	1514
2004	Power 5 1.9GHz (IBM Server)	1452	2702

SPEC CPU Int 2000 Listado de códigos

- 164.gzip Data compression utility
- 175.vpr FPGA circuit placement and routing
- 176.gcc C compiler
- 181.mcf Minimum cost network flow solver
- 186.crafty Chess program
- 197.parser Natural language processing
- 252.eon Ray tracing
- 253.perlbmk Perl
- 254.gap Computational group theory
- 255.vortex Object Oriented Database
- 256.bzip2 Data compression utility
- 300.twolf Place and route simulator

SPEC CPU FP 2000
Listado de códigos

- 168.wupwise Quantum chromodynamics
- 171.swim Shallow water modeling
- 172.mgrid Multi-grid solver in 3D potential field
- 173.applu Parabolic/elliptic partial differential equations
- 177.mesa 3D Graphics library
- 178.galgel Fluid dynamics: analysis of oscillatory instability
- 179.art Neural network simulation; adaptive resonance theory
- 183.equake Finite element simulation; earthquake modeling
- 187.facerec Computer vision: recognizes faces
- 188.ammp Computational chemistry
- 189.lucas Number theory: primality testing
- 191.fma3d Finite element crash simulation
- 200.sixtrack Particle accelerator model
- 301.apsi Solves problems regarding temperature, wind, velocity and distribution of pollutants

1.2.1 Comparación de rendimientos (3)

- Problemas. Los resultados dependen de muchos factores:
  - ✓ *Versión del sistema operativo.*
  - ✓ *Carga de la máquina.*
  - ✓ *Versión del compilador.*
  - ✓ *Optimizaciones de los compiladores (algunos fabricantes hacen trucos!).*
  - ✓ *Características del hardware (memoria cache, memoria principal, etc).*

Fuente: Comput. Architecture. Hennessy y Patterson

Hardware		Software	
Model number	Powerstation 590	O/S and version	AIX version 3.2.5
CPU	66.67 MHz POWER2	Compilers and version	C SET++ for AIX C/C++ version 2.1 XL FORTRAN/6000 version 3.1
FPU	Integrated	Other software	See below
Number of CPUs	1	File system type	AIX/JFS
Primary cache	32KBI+256KBD off chip	System state	Single user
Secondary cache	None		
Other cache	None		
Memory	128 MB		
Disk subsystem	2x2.0 GB		
Other hardware	None		
SPECbase_fp92 tuning parameters/notes/summary of changes:			
FORTRAN flags: -O3 -qarch=pwrx -qhsflt -qnofold -bnso -Bl:/lib/syscalls.exp			
C flags: -O3 -qarch=pwrx -Q -qtune=pwrx -qhssngl -bnso -bl:/lib/syscalls.exp			

FIGURE 1.10 The machine, software, and baseline tuning parameters for the SPECfp92 report on an IBM RS/6000 Powerstation 590.

→ Reproducibilidad

1.2.2 Análisis cuantitativo

- Principio básico de diseño: mejorar el rendimiento del caso más frecuente.
- Speed-up: Mide la mejora del rendimiento global de un sistema cuando se mejora alguno de sus subsistemas
  - ✓ *S* = Tiempo de ejecución sin usar la mejora/Tiempo de ejecución usando una mejora
- Ley de Amdahl: La mejora en el rendimiento estará limitada por la fracción de tiempo que se puede usar dicha mejora.

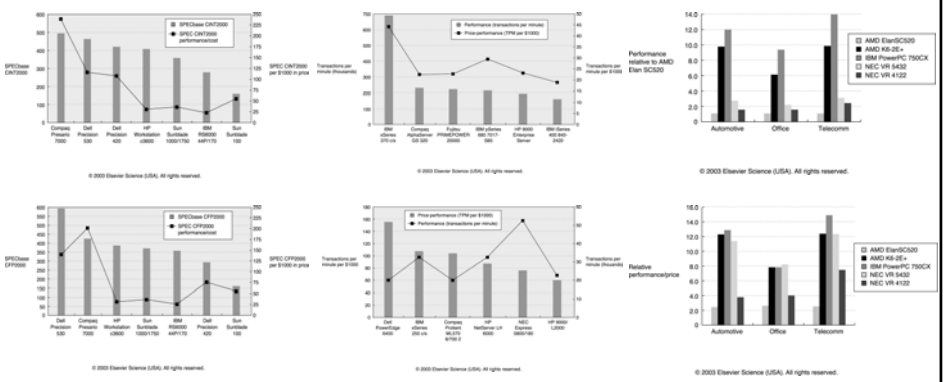
$$Speed-up = \frac{1}{(1-F_m) + F_m / S_m}$$

  - ✓ F_m = Fracción del tiempo en que se usa el subsistema mejorado.
  - ✓ S_m = Speed-up (aceleración) del subsistema mejorado.
- Limitación del Speed-up (suponiendo el Speed-up de la mejora infinito)

$$Speed-up_{maximo} = \frac{1}{(1-F_m)}$$

1.2.3 Comparación de rendimiento y precio-rendimiento

- Computadores personales
- Servidores
- Sist. empotrados



Fuente: Comput. Architecture. Hennessy y Patterson

1.3 Jerarquía de Memoria

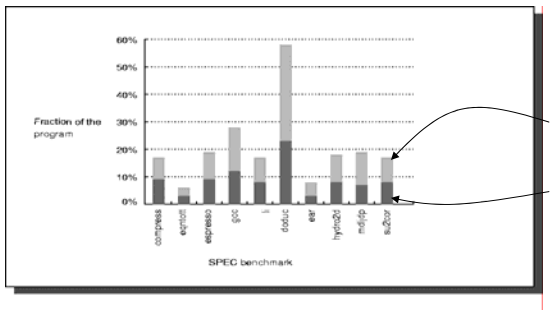
Instrucción Típica

- Lee dos operandos de memoria y escribe un resultado.
- Necesita realizar 3 lecturas y una escritura en el ciclo de instrucción
- Aspectos a tener en cuenta
  - *Ancho de banda: número de palabras accedidas por unidad de tiempo.*
  - *Latencia: tiempo transcurrido entre enviar la petición de acceso a memoria y la recepción del dato.*
- MFLOPS = nº de operaciones flotantes/(nº accesos x tacc + tcal)
  - *Supongamos tcal=0, datos de 32 bits. Una operación en punto flotante requiere 4 accesos.*
  - *10 Mflops ➔ tacc = 1/(4x10) = 25 ns ➔ 40 Macc/seg x 4 bytes/acc=160 Mbytes/seg*
  - *100 Mflops ➔ tacc= 1/(4x100)= 2.5 ns ➔ 400 Macc/seg x 4 bytes/acc=1.6 Gbytes/seg.*

El acceso a memoria limita el rendimiento

1.3.1 Concepto de localidad

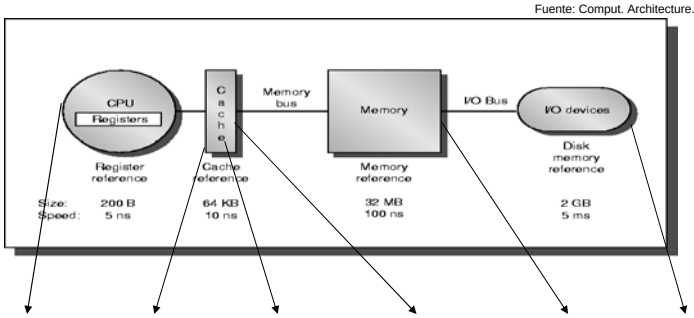
- Propiedades de los programas: alta probabilidad de referenciar
  - *Posiciones cercanas de memoria (localidad espacial).*
  - *Posiciones de memoria ya referenciadas (localidad temporal).*
- EL 90% de las referencias (dinámicas) son realizadas por el 10% del código (estático).



1.3.2 Organización de la memoria

- Aplicación: Jerarquía de la memoria
- Tener cerca del procesador lo que se espera utilizar.

Fuente: Comput. Architecture. Hennessy y Patterson

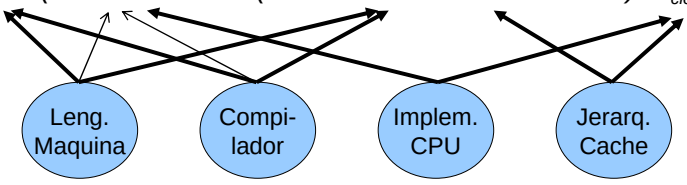


CPU	L1	L2	L3	Mem	Disco
32-512 reg	8-128K	128k-1M	512k-8M	16-512M	4-200G
1ck	1-2ck	2-3ck	5-6ck	10ck	5-12ms

1.3.3 Tiempo de procesador y jerarquía

$$T_{CPU} = N \times CPI \times t_{ciclo}$$

- *T = N x (ck de CPU/inst + ck de memoria/inst) x t_{ciclo}*
- *T= N x (ck de CPU/inst + (referencias/inst x ck/referencia) x t_{ciclo}*

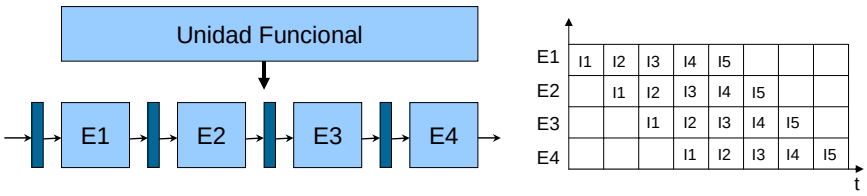


- Ciclo de cache está fuertemente relacionado con el ciclo de CPU.
- Referencia por instrucción es constante en las distintas implementaciones de una misma arquitectura.

1.4 Técnicas para reducir en nº de ciclos por instrucción

SEGMENTACIÓN

- Dividir la Unidad Funcional en etapas independientes.
- Cada etapa realiza una parte de la operación.

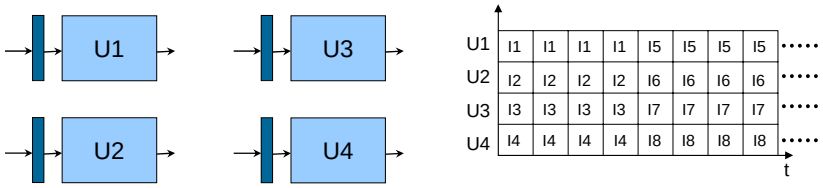


- Rendimiento
  - ✓ F_m = 1 (no real).
  - ✓ S_m = número de etapas.
  - ✓ S = número de etapas.

1.4 Técnicas para reducir en nº de ciclos por instrucción (2)

PARALELISMO

- Replicar unidades funcionales.
- Cada réplica realiza completamente la operación.



- Rendimiento
  - ✓ F_m = 1 (no real).
  - ✓ S_m = número de réplicas.
  - ✓ S = número de réplicas.

1.4.1 Relación con la tecnología

- La segmentación y el paralelismo son técnicas útiles cuando el número de puertas disponibles aumenta más rápidamente que la velocidad de una puerta.
  - ✓ Eso es cierto ya que
    - El número de puertas crece cuadráticamente
      - La tecnología de fabricación mejora linealmente (de 10µ en 1971 a 0.13µ en el 2003)
      - Pero la oblea de Si tiene 2 dim. → crecimiento cuadrático del nº TRTs
    - La velocidad de las puertas crece linealmente
    - Otros problemas de integración a gran escala
      - El retardo de transmisión escala muy pobremente
      - El consumo de potencia será el principal limitador en el futuro.
- Mejoras en la velocidad de puerta se incorporan ortogonalmente modificando el tiempo de ciclo.

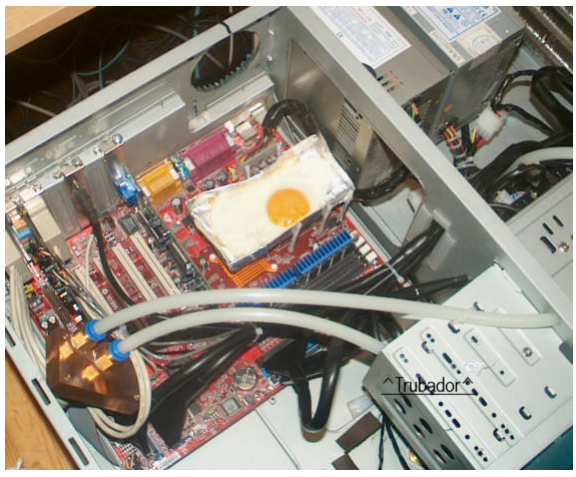
Elevado consumo de potencia

- http://www.phys.ncku.edu.tw/~htsu/humor/fry_egg.html

Componentes del PC:

Athlon XP1500+ CPU
MSI K7T266 Pro-2 Raid Motherboard
720gig Hard Drives
Juno P6 Case
24x Liteon CDRW

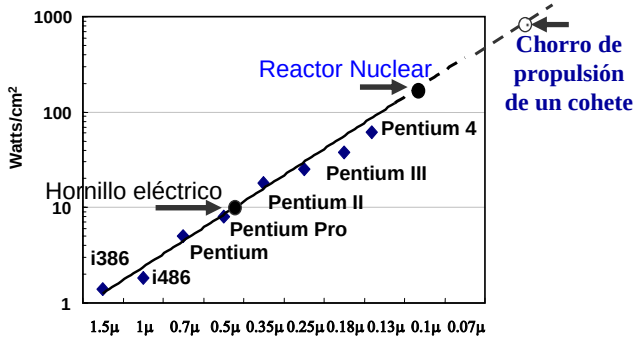
Tiempo: 11 minutos



¿Hasta donde podemos llegar?

- Densidad de potencia en Vatios/cm^2

Superficie del sol



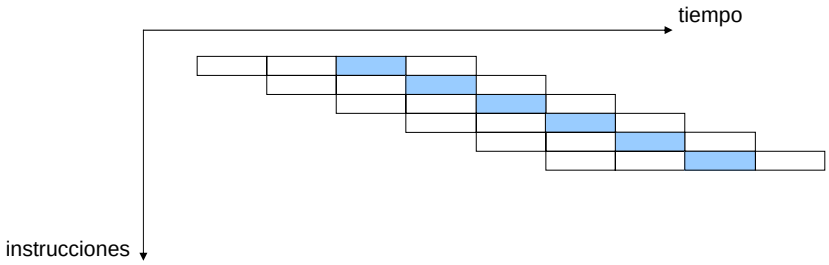
Extraído de un curso de Arq. de Computadores del Dr. Avi Mendelson

1.4.2 Reducción del CPI

- Objetivo: reducir el CPI.
- Explotar el paralelismo entre instrucciones de forma transparente al lenguaje de alto nivel o ensamblador.
  - ✓ *Compagina las ventajas de la programación secuencial con la disminución del tiempo de cálculo en una ejecución concurrente.*
  - ✓ *La concurrencia no es visible o poco visible al programador de lenguaje máquina.*
- Hay que garantizar que el resultado que se obtiene es el mismo
  - ✓ *Las operaciones de lectura y escritura de las variables deben de efectuarse en el orden que especifica el programador.*

1.4.3 Concurrencia en la ejecución de instrucciones. Evolución (1)

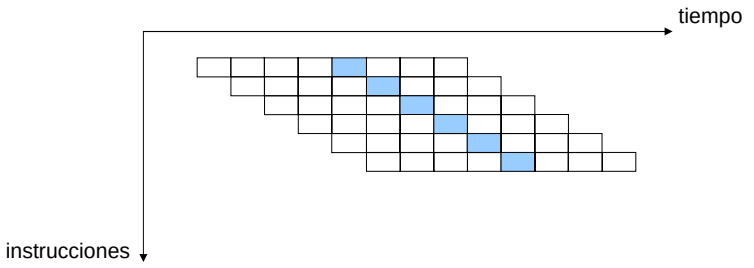
PROCESADOR SEGMENTADO



- 1 instrucción por ciclo
- MIPS (R2000, R3000), SPARC, H.P. (Precision Architecture). Intel 8086 hasta el i486.

1.4.3 Concurrencia en la ejecución de instrucciones. Evolución (2)

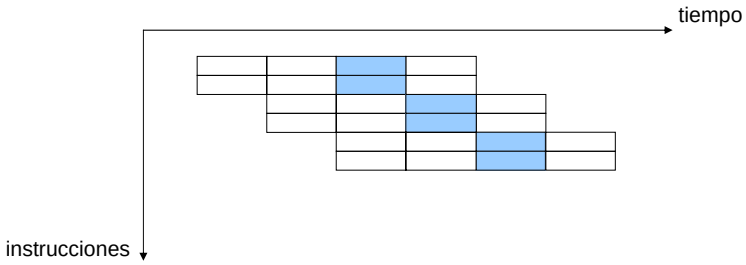
PROCESADOR SUPERSEGMENTADO



- Subdivisión de cada etapa en subetapas. Reducción del tiempo de ciclo.
- MIPS R4000

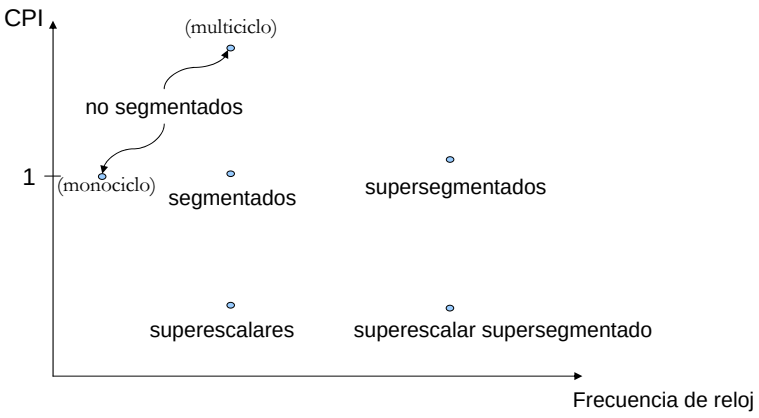
1.4.3 Concurrencia en la ejecución de instrucciones. Evolución (3)

PROCESADOR SUPERESCALAR



- Varias instrucciones por ciclo.
- Intel (i860, Pentium, PII, PIII y P4), IBM (RS-6000, Power PC), DIGITAL (Alpha), SuperSPARC, H.P. (PA-RISC).
  En general, todos los procesadores modernos.

1.4.4 Relación CPI-Frecuencia de reloj



1.5 Diseño del conjunto de instrucciones

- Nivel ISA: Instruction Set Architecture
  - ✓ Parte de la máquina visible...
    - Al programador en lenguaje ensamblador
    - Al compilador de lenguaje de alto nivel
  - ✓ No confundir con el nivel ASM
    - Completa el nivel ISA con directivas, macros, etc, que simplifican la tarea al programador

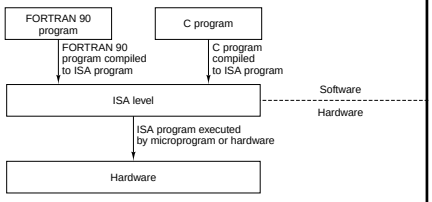


Figure 5-1. The ISA level is the interface between the compilers and the hardware.

Fuente: Structured Comput. Organization. A. Tanenbaum

- Si eliminamos el nivel ISA...
  - ✓ Pierdes las ventajas de la compilación sobre la interpretación
  - ✓ El HW tendría que interpretar directamente el LAN
    - Es más lento, difícil soportar más de un LAN, no puedes aplicar optimizaciones en tiempo de compilación ya que no se compila el LAN.
- El nivel ISA debe diseñarse para ofrecer
  - ✓ Hacia arriba (LAN) facilidades al compilador de LAN
  - ✓ Hacia abajo (HW) instrucciones que se ejecuten eficientemente

1.5.1 Clasificación de arquitecturas ISA

- Cuatro tipos de arquitecturas básicas
  - ✓ Ejemplo: secuencia compilada para la instrucción C=A+B
    - A, B y C son variables almacenadas en memoria. RPG = registros propósito general

Pila	Acumulador	Registro-Memoria (RPG)	Load-Store (RPG)
`Push A`	`Load A`	`Load R1,A`	`Load R1,A`
`Push B`	`Add B`	`Add R1,B`	`Load R2,B`
`Add`	`Store C`	`Store C,R1`	`Add R3,R2,R1`
`Pop C`			`Store C,R3`

- Arquitecturas antiguas basadas en pila o en acumulador
  - ✓ Más antiguo aun cuando se permitían las arquitecturas memoria-memoria
- Actualmente: Registro-Memoria (CISC) o Load-Store (RISC)
  - ✓ Con RPG no se impone un orden al calcular (con pila sí)
  - ✓ RPG almacenan variables de uso frecuente y reducen tráfico con memoria

1.5.1 Clasificación de arquitecturas ISA

- Es recomendable tener un gran número de RPGs
  - ✓ Para evaluar expresiones, pasar parámetros y el resto para variables
- Clasificación de arquitecturas en función de
  - ✓ nº de operandos que pueden residir en memoria (de 0 a 3)
  - ✓ nº max. de operandos explícitos en la instrucción (2 o 3)

nº de dir. de memoria	Max. nº de operandos	Ejemplos
0	3	SPARC, MIPS, HP-PA, PowerPC, Alpha
1	2	Intel 80x86, Motorola 68000
2	2	VAX
3	3	VAX

Tipo	Ventajas	Desventajas
Reg-Reg (0,3)	Simple, formato inst. fijo	Programas ejecutables con más instruc.
Reg-Mem (1,2)	Acceso a datos sin carga previa	Uno de los operandos se machaca con el resultado.
Mem-Mem (3,3)	Programas compactos que consume menos memoria	Formato inst. variable. Mucho tráfico de datos con mem.

Conclusión: usar RPGs en modo (0,3): Load-Store

1.5.1 Clasificación de arquitecturas ISA. Ejemplos: Pentium II y UltraSPARC II

Fuente: Structured Comput. Organization. A. Tanenbaum

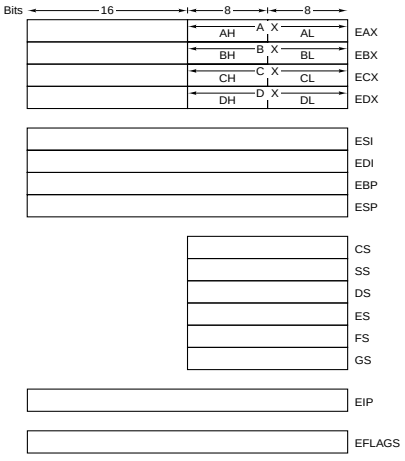


Figure 5-3. The Pentium II's primary registers.

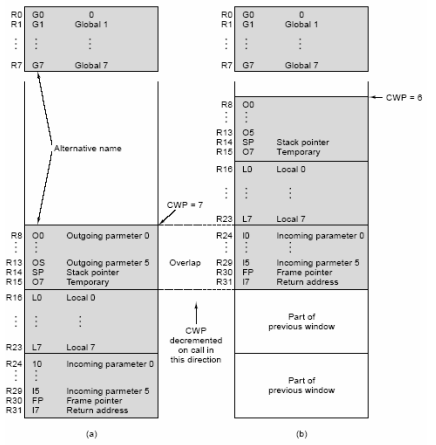


Figure 5-5. Operation of the UltraSPARC II register windows.

1.5.2 Modos de direccionamiento

- ¿Cómo se interpreta una dirección de memoria?
  - ✓ Little Endian: El byte con dirección “xx..xx00” es el LSB (intel)
  - ✓ Big Endian: El byte con dirección “xx..xx00” es el MSB (motorola, Sparc, DLX)
  - ✓ Alineamiento en memoria: dirección múltiplo del tamaño del dato
    - Ejemplo: Una palabra de cuatro bytes debe estar en direcciones múltiplo de 4 (“xxx...xxx00”)
    - Ejemplo: Palabra de 8 bytes alineada en máquina littel endian:

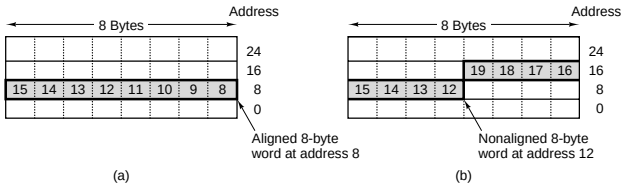


Figure 5-2. An 8-byte word in a little-endian memory. (a) Aligned. (b) Not aligned. Some machines require that words in memory be aligned.

Fuente: Structured Comput. Organization. A. Tanenbaum

1.5.2 Modos de direccionamiento

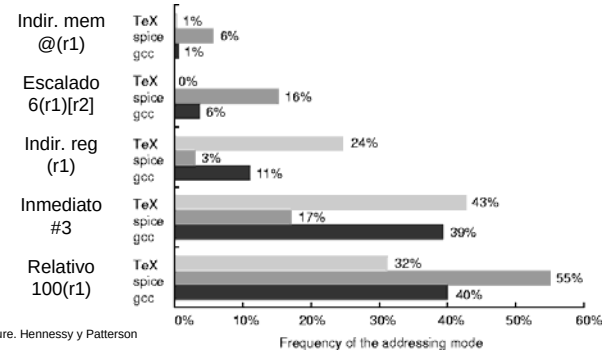
- ¿Cómo especificar la dirección efectiva del operando?

Tipo	Ejemplo	Significado	Cuando usarlo
Registro	`add r4,r3`	`r4←r4+r3`	Valores en registro
Inmediato	`add r4,#3`	`r4←r4+3`	Constantes
Directo o absoluto	`add r4,(100)`	`r4←r4+mem(100)`	Acceso a datos estáticos
Indirecto (deferred)	`add r4,(r1)`	`r4←r4+mem(r1)`	Acceso mediante punteros
Relativo (desplazamiento)	`add r4,100(r1)`	`r4←r4+mem(r1+100)`	Acceso a variables locales
Indexado	`add r4,(r1+r2)`	`r4←r4+mem(r1+r2)`	Acceso a arrays
Indirecto a memoria	`add r4,@(r1)`	`r4←r4+mem(mem(r1))`	r1 es un puntero a un puntero
Escalado	`add r4,6(r1)[r2]`	`r4←r4+mem(6+r1+r2×d)`	Recorrido de arrays

- En el direccionamiento con escalado “d” es el tamaño en bytes de los datos contenidos en el array

1.5.2 Modos de direccionamiento

- El dir. directo a registro se usa en el 50% de las instrucciones
- Porcentaje de uso de los restantes modos de direccionamiento:



Fuente: Comput. Architecture. Hennessy y Patterson

Los modos inmediato y relativo dominan el porcentaje de uso

1.5.3 Operaciones en el conjunto de instrucciones (IS)

- Tipos de instrucciones:
  - ✓ Movimiento de datos, Aritméticas, Lógicas, Control
- En general las instrucciones más simples son las más usadas:

Posición	Instrucción 80x86	% de uso
1	`Load`	22%
2	`Salto condic.`	20%
3	`Comparación`	16%
4	`Store`	12%
5	`add`	8%
6	`and`	6%
7	`sub`	5%
8	`mov reg, reg`	4%
9	`call`	1%
10	`return`	1%
	Total:	96%

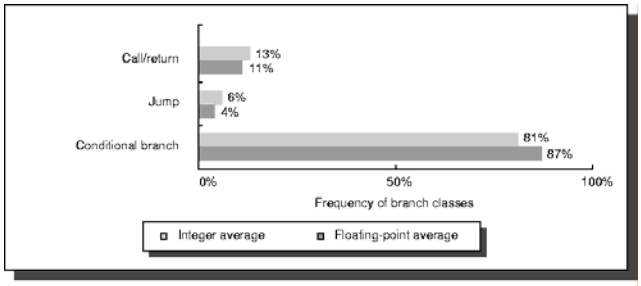
- Las 10 instrucciones más usadas
  - ✓ 5 programas del SPECint92
  - ✓ Compilados para i80x86
- Conclusiones:

Proporcionar primitivas, NO soluciones

Es poco útil proporcionar instrucciones específicas y adaptadas a un LAN particular.

1.5.3 Operaciones en IS: inst. control

- Distinguimos 4 tipos de instrucciones de control
  - ✓ Salto condicional (branch), Salto incondicional (jump)
  - ✓ Procedimientos: llamada (call) y retorno (return)
- Frecuencia relativa de cada uno de los tipos



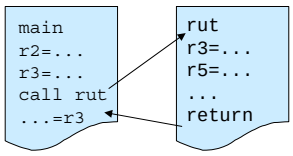
Fuente: Comput. Architecture. Hennessy y Patterson

1.5.3 Operaciones en IS: inst. control

- La dirección destino del salto se puede especificar como:
  - ✓ Un desplazamiento a sumar al PC (ocupa pocos bits y permite relocalizar el código)
  - ✓ El contenido de un reg. (útil para implementar "case" o funciones virtuales en C++)
- La condición de salto se puede especificar como:
  - ✓ Código de condición (CC) en un reg. de estado (SR)
    - Usado en arq. CISC. El uso del SR limita la reordenación de instrucciones
    - Ejemplo: CMP AX, BX; JEQ label;
  - ✓ Registro de condición.
    - Se usa un registro para contener la evaluación de la condición. Consume un registro.
    - Ejemplo: SEQ R1,R2,R3; JNEZ R1, label (set r1 a 1 si r2=r3 y luego salta si r1 es 1)
  - ✓ Comparación y salto unificados
    - Reduce el número de instrucciones del programa, pero es mucho trabajo para 1 inst.
    - Ejemplo: JEQ R2,R3, label (si r2=r3 salta a la instrucción etiquetada con label)
- La comparación más frecuente es igual/no-igual (sobre todo con 0)

1.5.3 Operaciones en IS: inst. control

- Llamadas a procedimientos: salvar dir. retorno y reg. que se modif.
- Dos alternativas:
  - ✓ Salva el que llama: antes de llamar se salvan los reg. necesarios
  - ✓ Salva el llamado: dentro de la rutina se salvan los reg. que se usen



• **Salva el que llama**: salva r3 antes de llamar

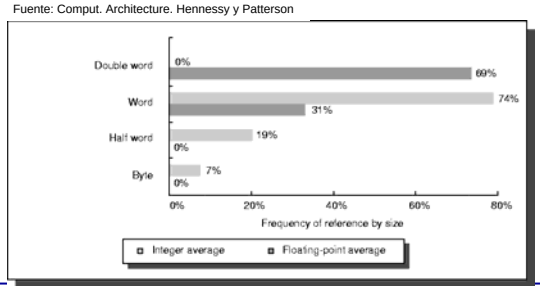
• **Salva el llamado**: al comenzar la rutina se salvan r3 y r5

- La dirección de retorno se guarda:
  - ✓ Arquitecturas antiguas: en la pila, es decir en la memoria principal
  - ✓ Arquitecturas modernas:
    - En un registro. En ese caso, el retorno es realmente un salto al contenido del registro
    - En una pila interna al procesador. Implementado en el Itanium con el nombre RSB

1.5.4 Tipo y tamaño de los operandos

- El tipo de los datos se incluye en el código de operación (CO)
- Tipos comunes
  - ✓ Enteros: Carácter (un byte), half word (2 bytes), word (4 bytes)
  - ✓ Punto flotante (FP): simple precisión (1 word), doble prec. (2 word)
  - ✓ Arquitecturas para aplicaciones no numéricas incluyen BCD
- Frecuencia de uso de los distintos tipos de datos para el SPEC92

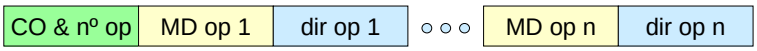
Fuente: Comput. Architecture. Hennessy y Patterson



Byte y half-word son poco usados
+
haz el caso común rápido
=
no implementar inst. para estos tipos

1.5.5 Codificación del conjunto de inst.

- El formato de instrucción contiene información del CO y operandos
- En arquitecturas Load-Store el modo de dir. se codifica en el CO
- Compromisos entre:
  - ✓ Tener el mayor nº posible de operandos y modos de direccionamiento (MD)
  - ✓ El impacto del tamaño de los campos para reg. y modos de dirección.
    - Si el tamaño es grande, las instrucciones son grandes y el programa ejecutable es grande
  - ✓ Tener un formato de inst. de tamaño fijo y múltiplo de byte
    - Facilita la implementación HW y permite segmentar el procesador
    - Si el formato de inst. es variable, hasta que no has decodif. una inst. no puedes buscar la sig.
- Formato de codificación variable (ej. VAX):



- Formato de codificación fija (ej. DLX, MIPS, PowerPC, HP-PA, SPARC)

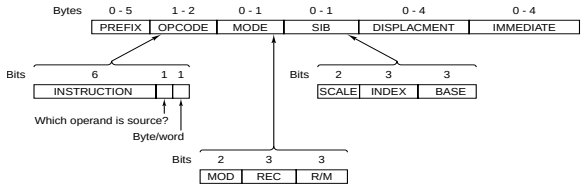
1.5.5 Codificación del conjunto de inst.

- Ventajas de codificación con formato fijo
  - ✓ Tamaño único para todas las instrucciones
  - ✓ Etapa de decodificación sencilla con tiempo de decodificación constante.
- Inconvenientes de codificación con formato fijo
  - ✓ Aplicable cuando hay pocos tipos de MD y pocas direcciones a mem.
  - ✓ No puedes ajustar el tamaño de la inst. al número de bits que necesita
- Ejemplo de codificación variable en el VAX:

```
addl3 r1, 737(r2), (r3)
```

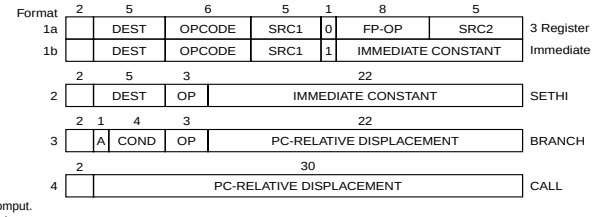
  - ✓ addl3 significa: suma de 32 bits con tres operandos. El CO ocupa 1 byte
  - ✓ r1 es el reg. destino: ocupa 1 byte (4 bits con el MD y 4 bits con el nº reg)
  - ✓ 737(r2) ocupa 3 bytes: MD (4bits), nº reg (4bits), desplaz. 737 (2bytes)
  - ✓ (r3) ocupa 1 byte: MD (4bits), nº reg (4bits).
  - ✓ Total: 6 bytes y 3 accesos a memoria (1 para inst y 2 operandos fuente)
- En el VAX: instrucciones con tamaño entre 1 y 53 bytes!!

1.5.5 Codificación del conjunto de inst. Ejemplos: Pentium II y SPARC



= 17 bytes maximo!

Figure 5-13. The Pentium II instruction formats.



Fuente: Structured Comput. Organization. A. Tanenbaum

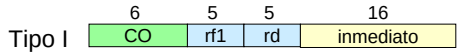
Figure 5-14. The original SPARC instruction formats.

1.5.6 Nivel ISA del DLX

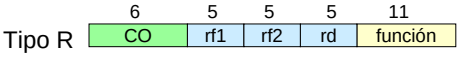
- 32 registros de propósito general (RPG)
  - ✓ r0, r1, r2, ..., r31. Todos de 32 bits
  - ✓ r0 siempre contiene el valor cero.
- 32 registros punto flotante (FPR)
  - ✓ f0, f1, f2, ..., f31. De simple precisión (32 bits)
  - ✓ f0, f2, ..., f30. De doble precisión (64 bits, concatena dos reg de 32 bits)
- Tipos de datos: word (32 bits), half-word (16 bits) y byte. FP simple y doble precisión.
- Byte y half-word se almacenan en RPG rellenando con 0's o ext. de signo.
- Direcciones de 32 bits. Modos de direccionamiento:
  - ✓ inmediato (addi r1,r2,#3) y relativo (lw r3,5(r1))
  - ✓ indirecto a registro poniendo desp=0 (0(r1)) y absoluto poniendo 5(r0)
- Big Endian. Todos los accesos deben estar alineados.

1.5.6 Nivel ISA del DLX
Codificación de las operaciones

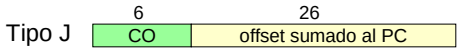
- Instrucciones de cálculo del tipo registro-registro
  - ✓ **OP** rd, rf1, rf2 → rd←rf1 **OP** rf2
- Instrucciones de acceso a memoria (word, half-word y byte)
  - ✓ lw rd, #X(rf1) → rd←Mem[X+rf1]
  - ✓ sw #X(rf1), rd → Mem[X+rf1]←rd
- Formato de las instrucciones



- lw y sw con desp: lw rd, inm(rf1) o sw inm(rf1),rd
- Todos los dir. inmediatos: rd←rf1 op inm
- Salto condic. BEQZ rf1, inm
- Salto por reg. rd=0, rf1=destino, inm=0



- Operaciones reg-reg: rd←rf1 func rf2
- El campo func. codifica la op.: add, sub, ...
- Movimiento entre reg. y lec/esc de reg. especial.



- Salto y salto con enlace (llamada a subrutina)
- Trap y retorno de excepción

1.5.6 Nivel ISA del DLX
Instrucciones de Load/Store

Example instruction	Instruction name	Meaning
LW R1,30(R2)	Load word	Regs[R1]←$_{32}$ Mem[30+Regs[R2]]
LW R1,1000(R0)	Load word	Regs[R1]←$_{32}$ Mem[1000+0]
LB R1,40(R3)	Load byte	Regs[R1]←$_{32}$ (Mem[40+Regs[R3]]$_0$)^{24} ## Mem[40+Regs[R3]]
LBU R1,40(R3)	Load byte unsigned	Regs[R1]←$_{32}$ 0^{24} ## Mem[40+Regs[R3]]
LH R1,40(R3)	Load half word	Regs[R1]←$_{32}$ (Mem[40+Regs[R3]]$_0$)^{16} ## Mem[40+Regs[R3]]##Mem[41+Regs[R3]]
LF F0,50(R3)	Load float	Regs[F0]←$_{32}$ Mem[50+Regs[R3]]
LD F0,50(R2)	Load double	Regs[F0]##Regs[F1]←$_{64}$ Mem[50+Regs[R2]]
SW 500(R4),R3	Store word	Mem[500+Regs[R4]]←$_{32}$ Regs[R3]
SF 40(R3),F0	Store float	Mem[40+Regs[R3]]←$_{32}$ Regs[F0]
SD 40(R3),F0	Store double	Mem[40+Regs[R3]]←$_{32}$ Regs[F0]; Mem[44+Regs[R3]]←$_{32}$ Regs[F1]
SH 502(R2),R3	Store half	Mem[502+Regs[R2]]←$_{16}$ Regs[R31]$_{16..31}$
SB 41(R3),R2	Store byte	Mem[41+Regs[R3]]←$_{8}$ Regs[R2]$_{24..31}$

FIGURE 2.22 The load and store instructions in DLX. Fuente: Comput. Architecture. Hennessy y Patterson

1.5.6 Nivel ISA del DLX
Instrucciones aritmético/lógicas

Fuente: Comput. Architecture. Hennessy y Patterson

Example instruction	Instruction name	Meaning
`ADD R1,R2,R3`	Add	`Regs[R1]←Regs[R2]+Regs[R3]`
`ADDI R1,R2,#3`	Add immediate	`Regs[R1]←Regs[R2]+3`
`LHI R1,#42`	Load high immediate	`Regs[R1]←42##0`^{16}
`SLLI R1,R2,#5`	Shift left logical immediate	`Regs[R1]←Regs[R2]<<5`
`SLT R1,R2,R3`	Set less than	`if (Regs[R2]<Regs[R3]) Regs[R1]←1 else Regs[R1]←0`

FIGURE 2.23 Examples of arithmetic/logical instructions on DLX, both with and without immediates.

- `mov r4,r3` se implementa mediante `add r4,r3,r0`
- `mov r4,#3` se implementa como `addi r4,r0,#3`
  - ✓ También se puede escribir lo mismo mediante: `li r4,#3`

1.5.6 Nivel ISA del DLX
Instrucciones típicas de control

Example instruction	Instruction name	Meaning
`J name`	Jump	`PC←name; ((PC+4)-2`^{25}`) ≤ name < ((PC+4)+2`^{25}`)`
`JAL name`	Jump and link	`R31←PC+4; PC←name; ((PC+4)-2`^{25}`) ≤ name < ((PC+4)+2`^{25}`)`
`JALR R2`	Jump and link register	`Regs[R31]←PC+4; PC←Regs[R2]`
`JR R3`	Jump register	`PC←Regs[R3]`
`BEQZ R4,name`	Branch equal zero	`if (Regs[R4]==0) PC←name; ((PC+4)-2`^{15}`) ≤ name < ((PC+4)+2`^{15}`)`
`BNEZ R4,name`	Branch not equal zero	`if (Regs[R4]!=0) PC←name; ((PC+4)-2`^{15}`) ≤ name < ((PC+4)+2`^{15}`)`

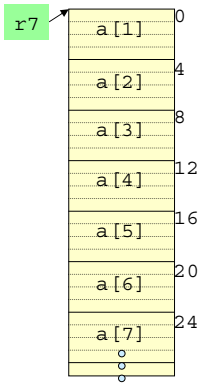
FIGURE 2.24 Typical control-flow instructions in DLX. Fuente: Comput. Architecture. Hennessy y Patterson

- `jal` se utiliza para saltar a subrutina
  - ✓ `jalr` cuando la dirección de la subrutina está en un registro
- Instrucciones en PF con sufijo F o D (distinto CO):
  - ✓ `addf, addd, subf, subd, multf, multd, divf, divd`

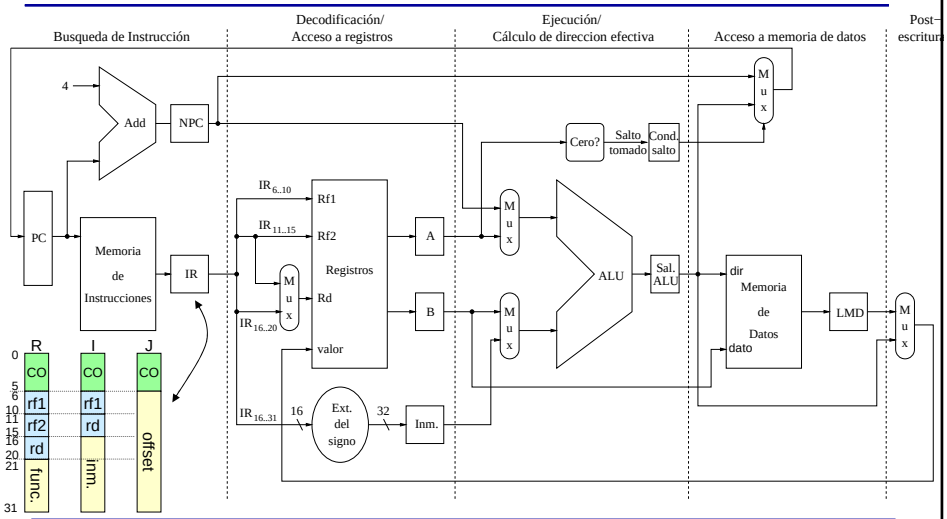
1.5.7 Nivel ISA del DLX
Programa ejemplo

- Calcular el valor máximo de un array de 10 nº apuntado por r7
  - ✓ r1 almacena el max, r2 el candidato a max, r3 será el reg. condición

```
        lw   r1,(r7)  ; 1er max
        addi r5,r0,#9 ; 9 iteraciones
lazo:   addi r7,r7,#4 ; r7++
        lw   r2,(r7)  ; candidato
        slt  r3,r1,r2 ; si r1<r2 → r3←1
        beqz r3,nomax ; si r3==0 salta
        add  r1,r0,r2 ; r1←r2
nomax:  subi r5,r5,#1 ; r5--
        bnez r5,lazo  ; otra iteración
        sw   max,r1   ; r1 a memoria...
        end
```



1.5.7 Implementación del DLX



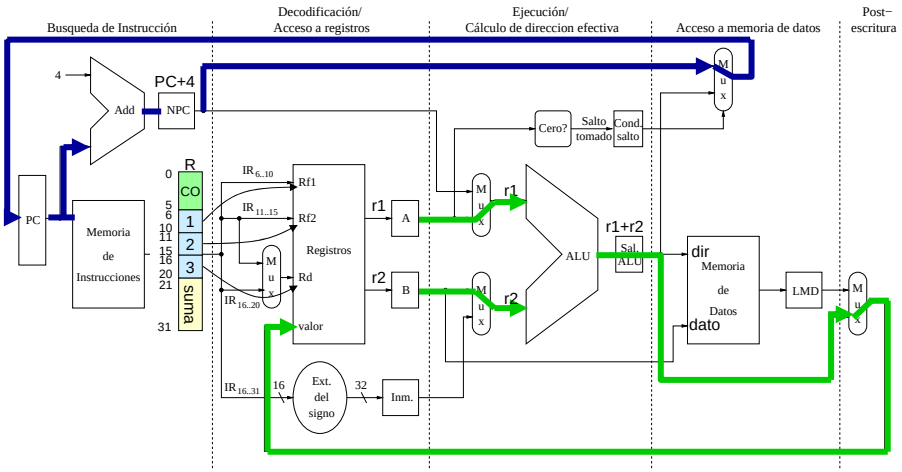
1.5.7 Implementación del DLX

- Búsqueda de Instrucción (BI)
  - ✓ IR←Mem(PC)
  - ✓ NPC←PC+4
- Decodificación / Lectura del banco de registros (DI)
  - ✓ A←Regs($IR_{6..10}$); (rf1)
  - ✓ B←Regs($IR_{11..15}$); (rd en tipo I y rf2 en tipo R)
  - ✓ Inm←((IR_{16})^{16}## $IR_{16..31}$); (campo inm. de 16bits con ext. de signo a 32b)
- Ejecución / Cálculo de dirección efectiva (EJ)
  - ✓ Load/Store: SalALU← A+Inm (Tipo I, calc. dir. efectiva)
  - ✓ ALU reg-reg: SalALU← A func B (Tipo R)
  - ✓ ALU reg-inm: SalALU← A op Inm (Tipo I)
  - ✓ Branch: SalALU← NPC + Inm y Cond←(A op 0) (Tipo I)
    - El reg. A (que contiene rf1) se compara con 0 para decidir si se salta o no.
    - El tipo de comparación está codificada en el CO. En `BEQZ` es "==" y en `BNEZ` es "!="

1.5.7 Implementación del DLX

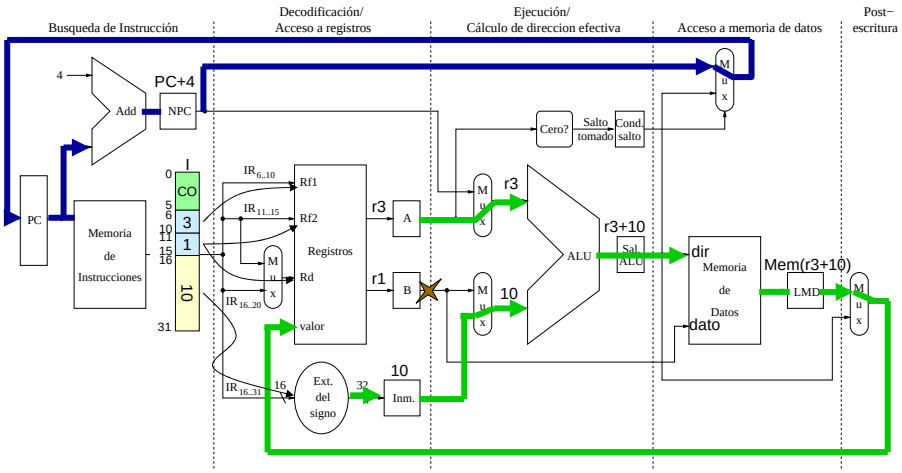
- Acceso a Memoria (M)
  - ✓ Load: LMD←Mem(SalALU)
  - ✓ Store: Mem(SalALU)←B
  - ✓ Branch: if (Cond) PC←SalALU else PC←NPC;
- Post-Escritura (Write-Back) (PE)
  - ✓ ALU reg-reg: Regs($IR_{16..20}$)←SalALU; (Tipo R, rd←SalALU)
  - ✓ ALU reg-inm: Regs($IR_{11..15}$)←SalALU; (Tipo I, rd←SalALU)
  - ✓ Load: Regs($IR_{11..15}$)←LMD; (Tipo I, rd←LMD)
- Posibles implementaciones:
  - ✓ Monociclo: Todas las instrucciones consumen 1ck de reloj
    - El periodo de ck se debe ajustar para permitir la ejecución de la instrucción más lenta
  - ✓ Multiciclo: Cada etapa (BI, DI, EJ, M, PE) consume 1ck
    - Las instrucciones pasan sólo por las etapas necesarias consumiendo más o menos ck's
    - El CPI de cada instrucción depende de cuantas etapas tenga que atravesar

1.5.7 Implementación del DLX
Ejemplo: `add r3,r1,r2` (tipo R)



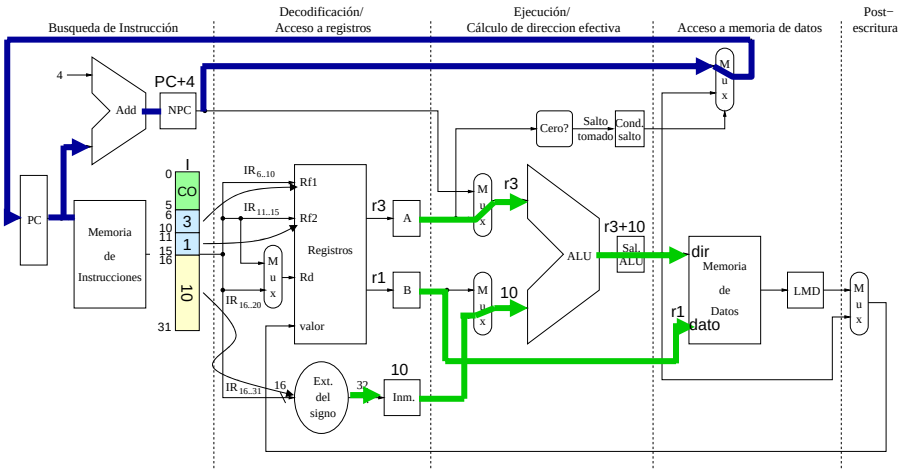
- Necesita pasar por 5 etapas (4 etapas si actualizamos PC en EJ o PE)

1.5.7 Implementación del DLX
Ejemplo: `lw r1,10(r3)` (tipo I)



- Necesita pasar por las 5 etapas

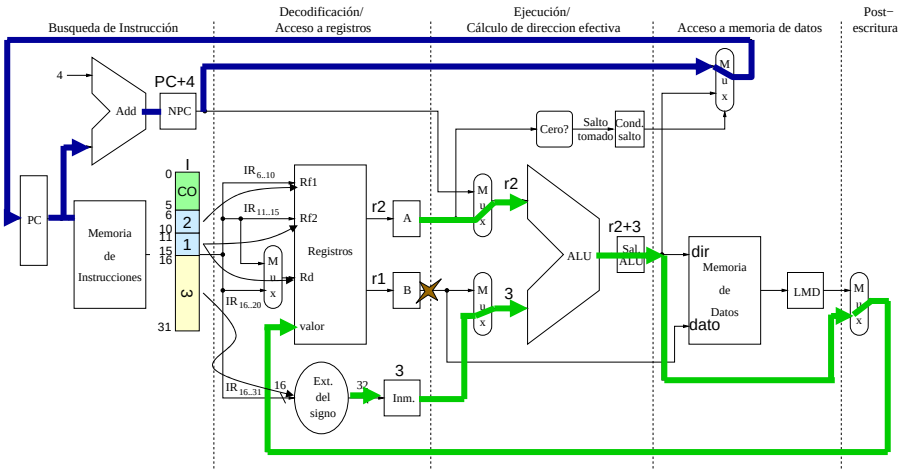
1.5.7 Implementación del DLX
Ejemplo: **`sw 10(r3),r1`** (tipo I)



- Necesita pasar por 4 etapas

Von Neumann Rendimiento Jerarquía memoria Reducción CPI ISA 67

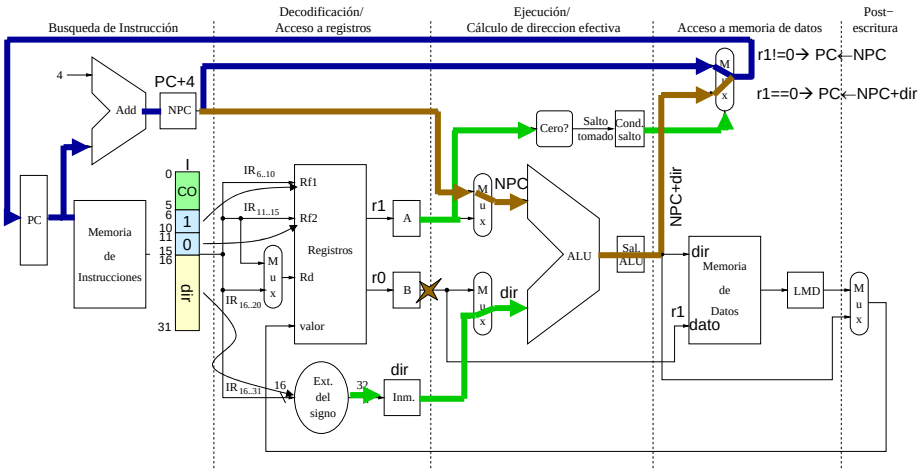
1.5.7 Implementación del DLX
Ejemplo: **`addi r1,r2,#3`** (tipo I)



- Necesita pasar por las 5 etapas (4 etapas si actualizamos PC en EJ o PE)

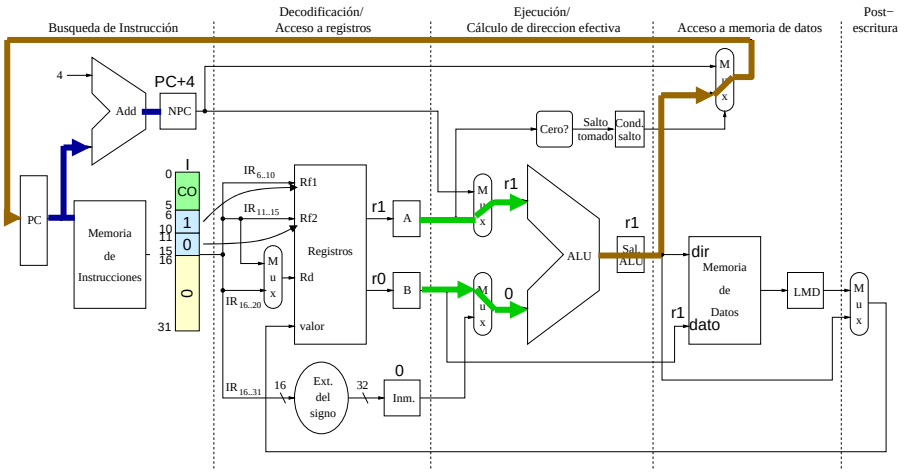
Von Neumann Rendimiento Jerarquía memoria Reducción CPI ISA 68

1.5.7 Implementación del DLX
Ejemplo: **`beqz r1,dir`** (tipo I)



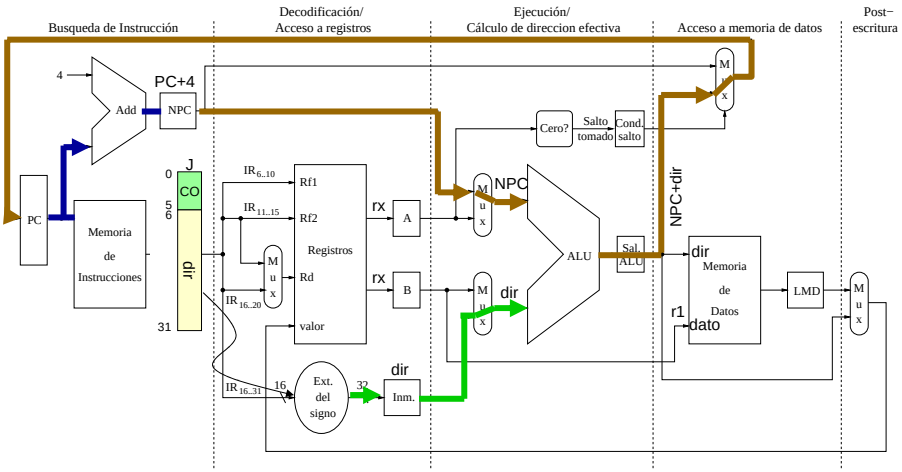
- Necesita pasar por 4 etapas

1.5.7 Implementación del DLX
Ejemplo: **`jr r1`** (tipo I)



- Necesita pasar por 4 etapas

1.5.7 Implementación del DLX
Ejemplo: `j dir` (tipo J)



- Necesita pasar por 4 etapas. Necesita ext. de signo de 26 bits.

Bibliografía

- J.L HENNESSY, D. PATTERSON. Computer Architecture: a quantitative approach. Ed. Morgan Kaufman, Segunda Edición 1996. Tercera Edición 2003
  - ✓ Apartados 1.1 a 1.4: Capítulo 1
  - ✓ Apartado 1.5: Capítulo 2
- ANDREW S. TANENBAUM. Structured Computer Organization. Cuarta Edición Ed. Prentice Hall, 1999.
  - ✓ Apartados 1.1 a 1.4: Capítulo 1
  - ✓ Apartado 1.5: Capítulo 5