Introducción

Estructura de Computadores
1ª-2ª Semana

Bibliografía:

[TOC] Temas 1-3	Apuntes Tecnología y Organización de Computadores
[HAM03] Cap.1	Organización de Computadores. Hamacher, Vranesic, Zaki. McGraw-Hill 2003
	Signatura ESIIT/C.1 HAM org
[BRY11] Cap.1	Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011
	Signatura ESIIT/C.1 BRY com
[PRI10]	Introducción a la Informática. Prieto, Lloris, Torres. McGraw-Hill Interamericana 2010
	Signatura ESIIT/A.0 PRI int

Guía de trabajo autónomo (4h/s)

- **Repaso**
  - Apuntes TOC
- **Lectura**
  - Cap.1 Hamacher
  - Cap.1 CS:APP (Bryant/O'Hallaron)
  - Guión de la Práctica 1

Bibliografía:

[TOC] Temas 1-3	Apuntes Tecnología y Organización de Computadores
[HAM03] Cap.1	Organización de Computadores. Hamacher, Vranesic, Zaki. McGraw-Hill 2003 Signatura ESIIT/C.1 HAM org
[BRY11] Cap.1	Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011 Signatura ESIIT/C.1 BRY com
[PRI10]	Introducción a la Informática. Prieto, Lloris, Torres. McGraw-Hill Interamericana 2010 Signatura ESIIT/A.0 PRI int

TOC

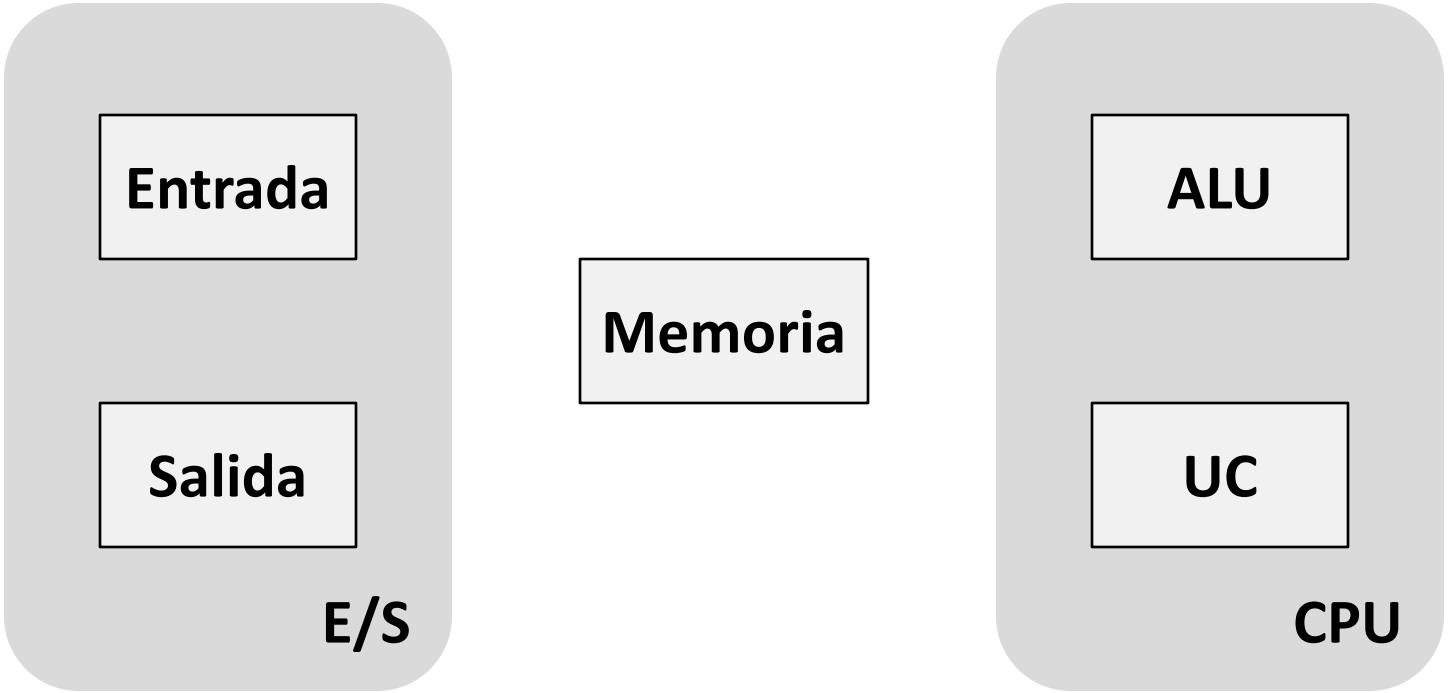
Vocabulario

- **Arquitectura**
  - Aspectos necesarios para redactar programa ensamblador correcto
  - Incluye: registros CPU, repertorio instrucciones, modos direccionamiento
- **Organización** (del computador, de la CPU, de la ALU)
  - Estructura: componentes y su interconexión (“foto fija”)
  - Funcionamiento: dinámica procesamiento información
- **Computador (digital):** E/S, M, CPU (ALU+UC)
  - Computador personal
    - Sobremesa (desktop)
    - Portátil (laptop)
  - Estación de trabajo (más prestaciones, gráficos)
  - Sistemas de empresa (más CPU y almacenamiento)
  - Servidores (bases de datos, gran volumen peticiones)
  - Supercomputadores (cálculos científicos)

Introducción

- **Unidades funcionales**
- Conceptos básicos de funcionamiento
- Estructuras de bus
- Rendimiento
- Perspectiva histórica

Unidades Funcionales



Unidades Funcionales

- **Arquitectura von Neumann**
  - Distingue 5 componentes: E/S, M, CPU (ALU+UC)

- **E: codificar / digitalizar / transmitir (lectura)**
  - teclado, ratón, red, disco, CD…
- **M: almacenar**
  - programas, datos E, resultados operaciones…
- **CPU: Unidad de procesamiento central**
  - procesa información E/M ejecutando programa
  - ALU: Unidad aritmético-lógica: operaciones
  - UC: Unidad de control: controla circuitos
- **S: codificar / almacenar / transmitir (escritura)**
  - pantalla, impresora, disco, red…

M almacena *instrucciones* y datos

- **Instrucciones máquina**
  - Transferencia (mov, in, out) M, E/S
  - Operaciones (add, and) ALU
  - Control (jmp, call, ret, set) UC
- **Concepto de "programa almacenado"**
  - Determina comportamiento máquina (salvo IRQ)
    - porque instrucciones reproducibles y flujo programa predeterminado
- **Datos**
  - En memoria, todo son datos
    - interpretado como programa (codop): si leido en etapa captación
    - Compilar, desensamblar: código usado como datos
  - Codificación:
    - Instrucciones: codops (codificación en bloque, por extension, según fabricante)
    - Enteros: binario (complemento a dos), BCD…
    - Alfabéticos: ASCII, EBCDIC…
    - Punto flotante: IEEE-754 simple/doble precisión…

TOC: 1.1 Conceptos básicos. Lenguaje máquina

- **El *lenguaje máquina* es el único que entienden los circuitos del computador (CPU). Las instrucciones se forman por bits agrupados en campos:**
  - **Campo de código de operación** indica la operación correspondiente a la instrucción.
  - **Campos de dirección** especifican los lugares (o posición) donde se encuentra o donde ubicar los datos con los que se opera.

E/S

- **Entrada:**
  - codificar información operador → M / CPU
    - teclado, ratón/palanca (junto con pantalla), micrófono
  - recuperar información previamente almacenada
    - HD, CD/DVD, lector tarjetas magnéticas…
  - comunicar ordenadores entre sí
    - tarjeta de red, módem…
- **Salida:**
  - codificar información resultado → operador humano
    - impresora, pantalla
  - almacenar para uso posterior
    - HD, CD/DVD…
  - comunicar con otros computadores
    - red, módem…
  - muchos dispositivos son duales E/S (aceptan R/W)

M

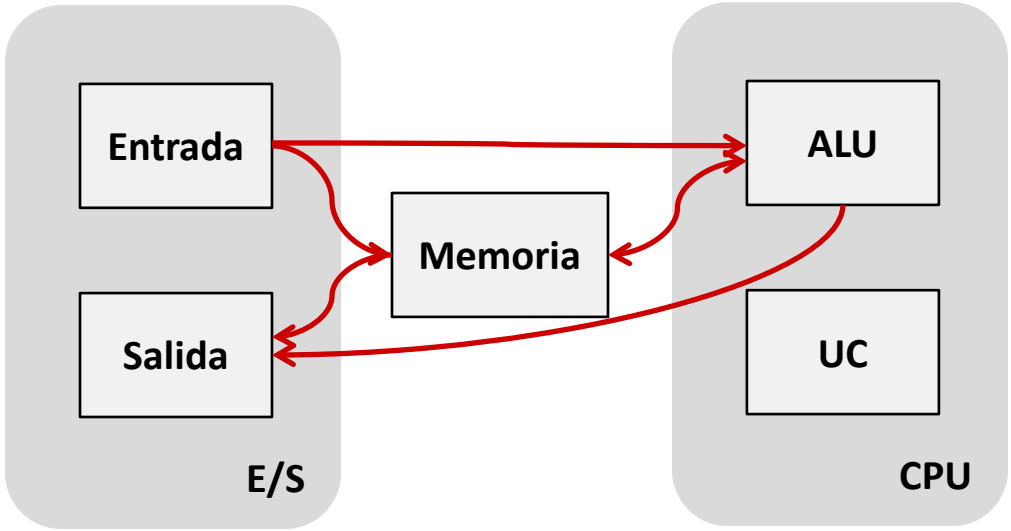
- **Memoria:**
  - Almacenamiento primario (memoria semiconductora)
    - Palabras n bits accesibles en 1 operación básica R/W
      - Longitudes palabra típicas: 16-64bits
      - Muy frecuente: memoria de bytes (asuntos alineamiento, ordenamiento)
    - Accesible aleatoriamente (RAM) por dirección (posición)
      - Bus direcciones, bus datos, bus control (R/W), T_{acceso}
      - Tamaños memoria típicos (PC): 8GB…32GB
      - Tiempos acceso típicos: ~ns (DDR4-2400 19.2GB/s CL15 Lat 12.5ns)
        - » DDR4-2400 → F_{bus}=1200MHz, T_{cyc}=0.833ns, Lat_{CAS} CL15 → 12.5ns
    - Jerarquía memoria: cache L1, L2 (on-chip), L3, MP
    - Programa almacenado en MP
  - Almacenamiento secundario (óptico/magn. E/S)
    - No es memoria von-Neumann, es E/S
    - Fichero swap se considera como parte de la jerarquía memoria

CPU

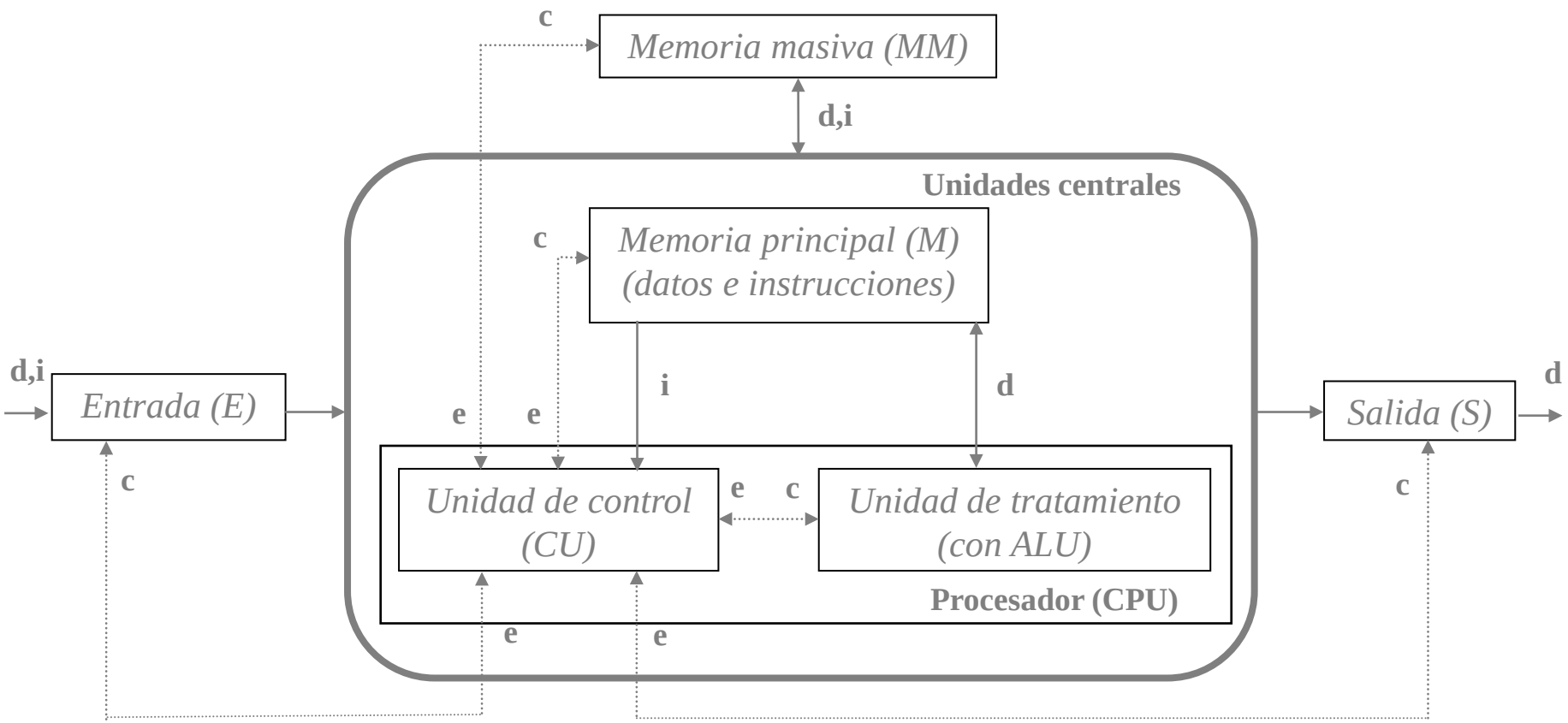
- **ALU:**
  - Componente más rápido del computador (junto con UC)
  - Registros: almacenamiento más rápido (más que L1)
    - Operandos/Resultado de/a memoria/registros
    - Arquitecturas R/R, R/M, M/M
  - Operaciones aritméticas (add, mul, div…)
    - Enteras y punto flotante
  - Operaciones lógicas (and, rol…)
    - Bit a bit
- **UC:**
  - Componente más rápido del computador (con ALU)
  - Controla todos los demás circuitos (ALU, M, E/S)
    - Según lo indicado por el programa almacenado en MP
    - Instrucciones transferencia → señales control M y E/S
    - Instrucciones aritm/lógicas → señales control ALU
    - Temporización señales (dirección, datos, R/W)

Posibilidades funcionamiento

- Programa E → MP
- Datos E → MP
- Ejecución programa: Datos → ALU → resultados
  - E / M → ALU → S / M
- Resultados → S
- Todo controlado según indique programa MP
  - Interpretado por la UC

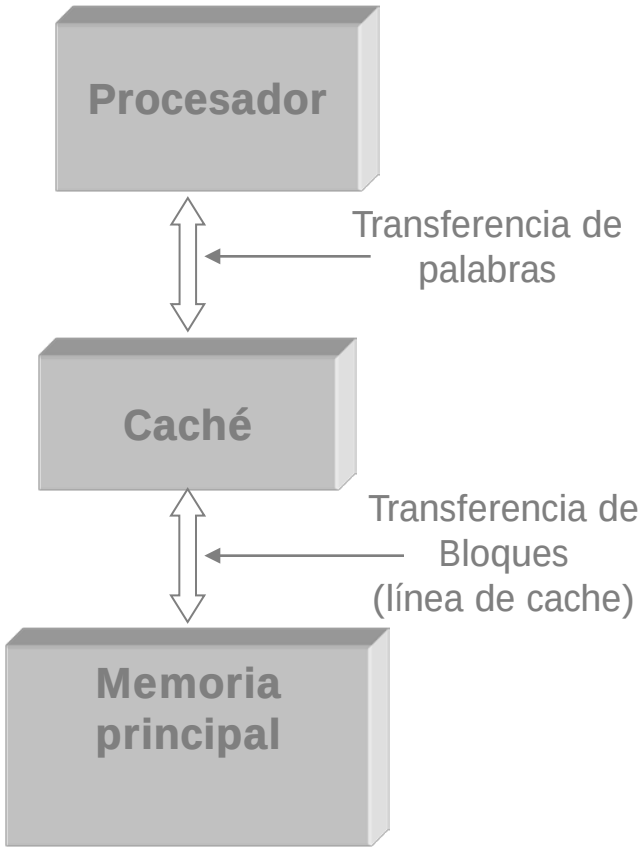


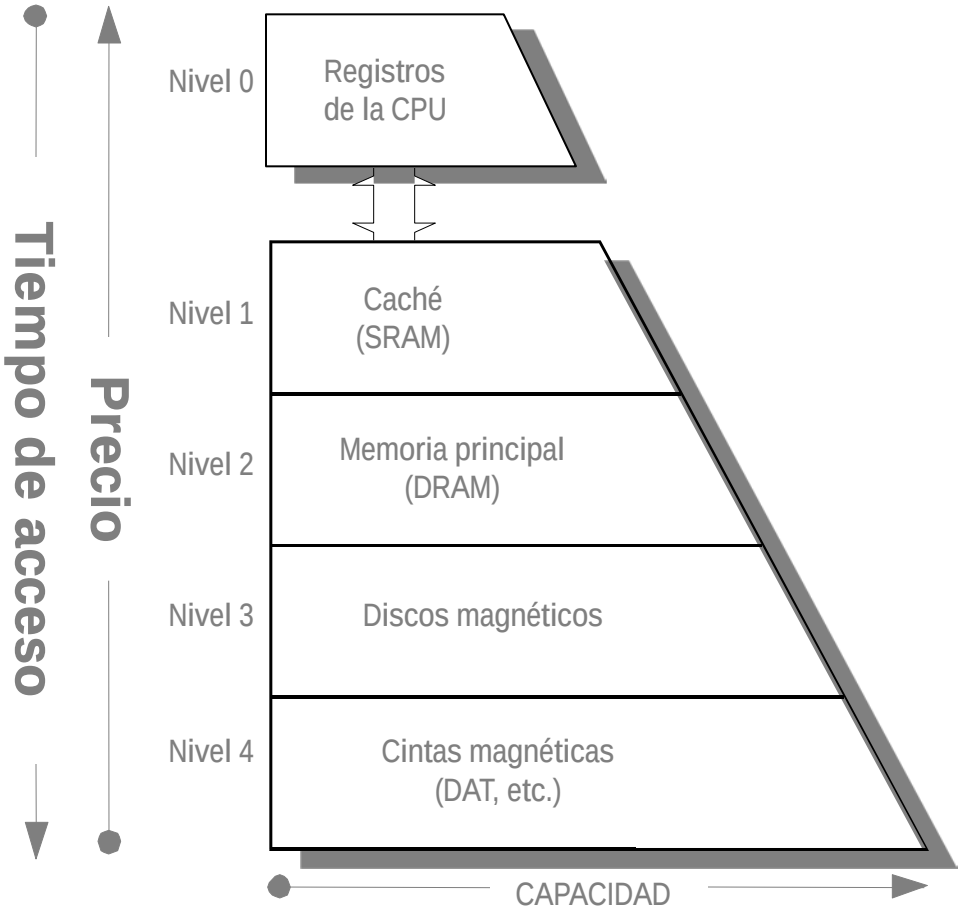
TOC: 2. Unidades funcionales de un computador



d: datos ; i: instrucciones
e: señales de estado c: señales de control

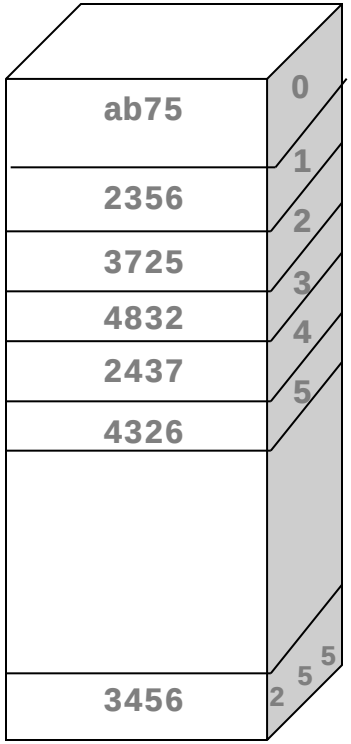
TOC: 2.2 Jerarquía de memoria





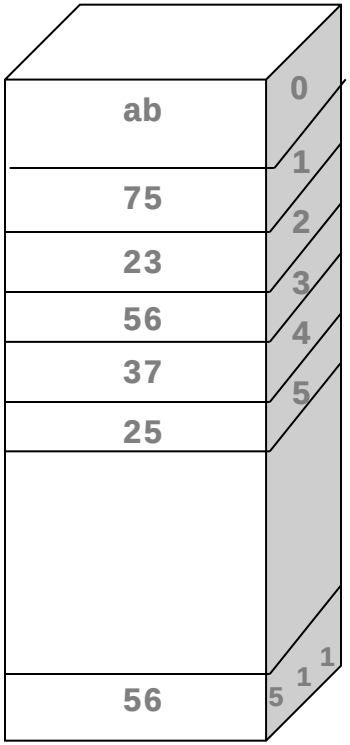
sobre Memoria

- **Organización en bytes**
  - ¿tamaño posición M = registro CPU (longitud palabra)?
    - ideal, pero no frecuente
    - típicamente, posiciones 1B (direccionamiento por bytes)
      - no necesidad empaquetamiento cadenas (strings)
      - Problemas: alineamiento, ordenamiento



TOC: hipotética memoria
256 palabras de 16bits
(apuntes TOC §1.2)

misma cantidad de memoria
organizada como 512 bytes
words alineadas, big-endian

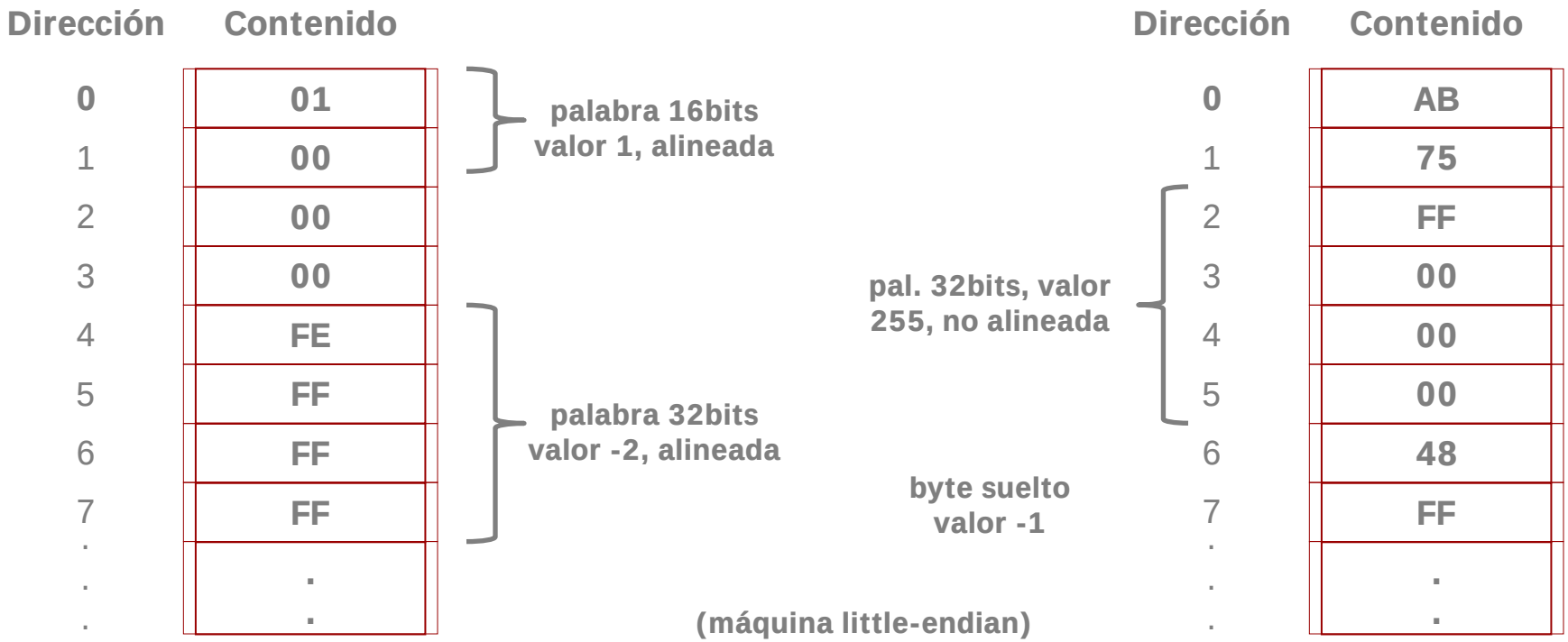


sobre Memoria de bytes

- **Ordenamiento en memoria de bytes**
  - Criterio del extremo menor (little-endian)
    - Primero se almacena el byte menos significativo (LSB)
      - LSB en posición M más baja, MSB en posición más alta
  - Criterio del extremo mayor (big-endian)
    - Primero el MSB (en posición M más baja)

sobre Memoria de bytes

- **Alineamiento en memoria de bytes**
  - Palabra de n bytes alineada ⇔ comienza en dirección múltiplo de n
    - Algunas CPUs requieren alineamiento accesos M (si no, bus error)
    - Otras acceden más rápido si acceso alineado
  - palabra no cruza línea de cache, página, etc



Clasificaciones m/n y pila-acumulador-RPG

- **Tipos de CPU según operandos de las instrucciones ALU**
  - también suele afectar a operandos instrucciones transferencia
- **Clasificación m/n**
  - Operaciones ALU admiten n operandos, m de ellos de memoria

- **Combinaciones típicas**
  - Máquinas pila: 0/0
    - Repertorio: Push M, Pop M, Add, And…
  - Máquinas de acumulador: 1/1
    - Operando implícito: registro acumulador A (más rápido que M)
    - Repertorio: Load M, Store M, Add M, And M…
  - Máquinas de RPG (Registros de Propósito General): (x/2, x/3)
    - Múltiples “acumuladores”
    - Repertorio: Move R/M R/M, Add R/M R/M R/M

RPG: Clasificación R/M

- **Para máquinas RPG**
  - Arquitecturas R/R (registro-registro)
    - 0/2, 0/3
      - Add R1, R2, R3
    - típico de RISC
  - Arquitecturas R/M (registro-memoria)
    - 1/2, 1/3 (2/3 poco frecuente)
      - Add R1, A
    - típico de CISC
  - Arquitecturas M/M (memoria-memoria)
    - 2/2, 3/3 (poco frecuente)
      - Add A, B
    - permite operar directamente en memoria
    - demasiados accesos memoria por instrucción máquina

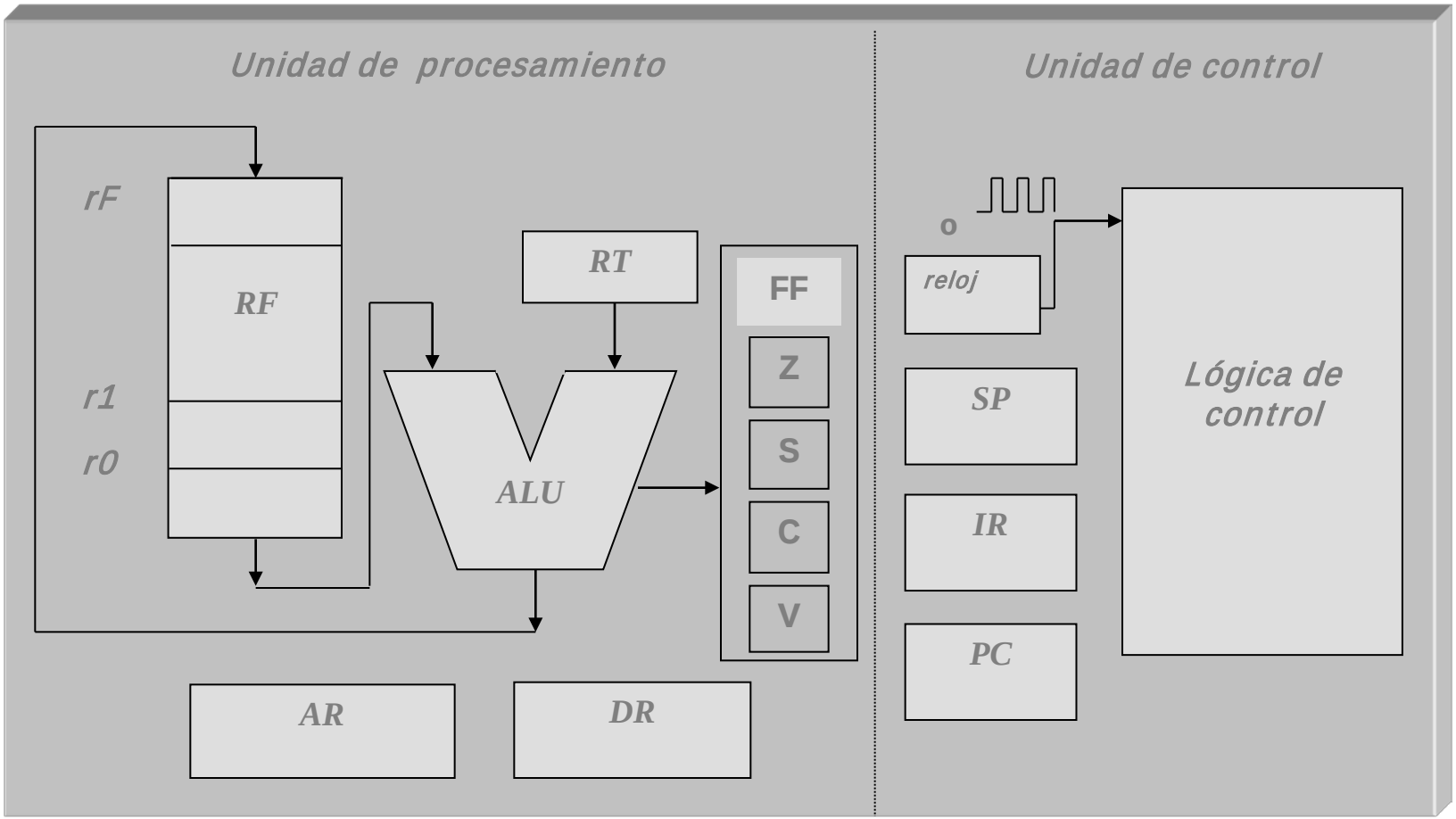
sobre Repertorios

- **ISA**
  - Arquitectura del Repertorio (Instruction Set Architecture)
  - Registros, Instrucciones, Modos de direccionamiento…
- **RISC**
  - Comput. repertorio reducido (Reduced Instruction Set Computer)
  - 0/2, 0/3
  - Pocas instrucciones, pocos modos, formato instrucción sencillo
  - UC sencilla → muchos registros
- **CISC**
  - Comput. repertorio complejo (Complex Instruction Set Computer)
  - 1/2, 1/3 (y resto)
  - “más próximos a lenguajes alto nivel”
  - Debate RISC/CISC agotado, diseños actuales mixtos

Introducción

- Unidades funcionales
- **Conceptos básicos de funcionamiento**
- Estructuras de bus
- Rendimiento
- Perspectiva histórica

TOC: 2.1 Elementos internos de un procesador

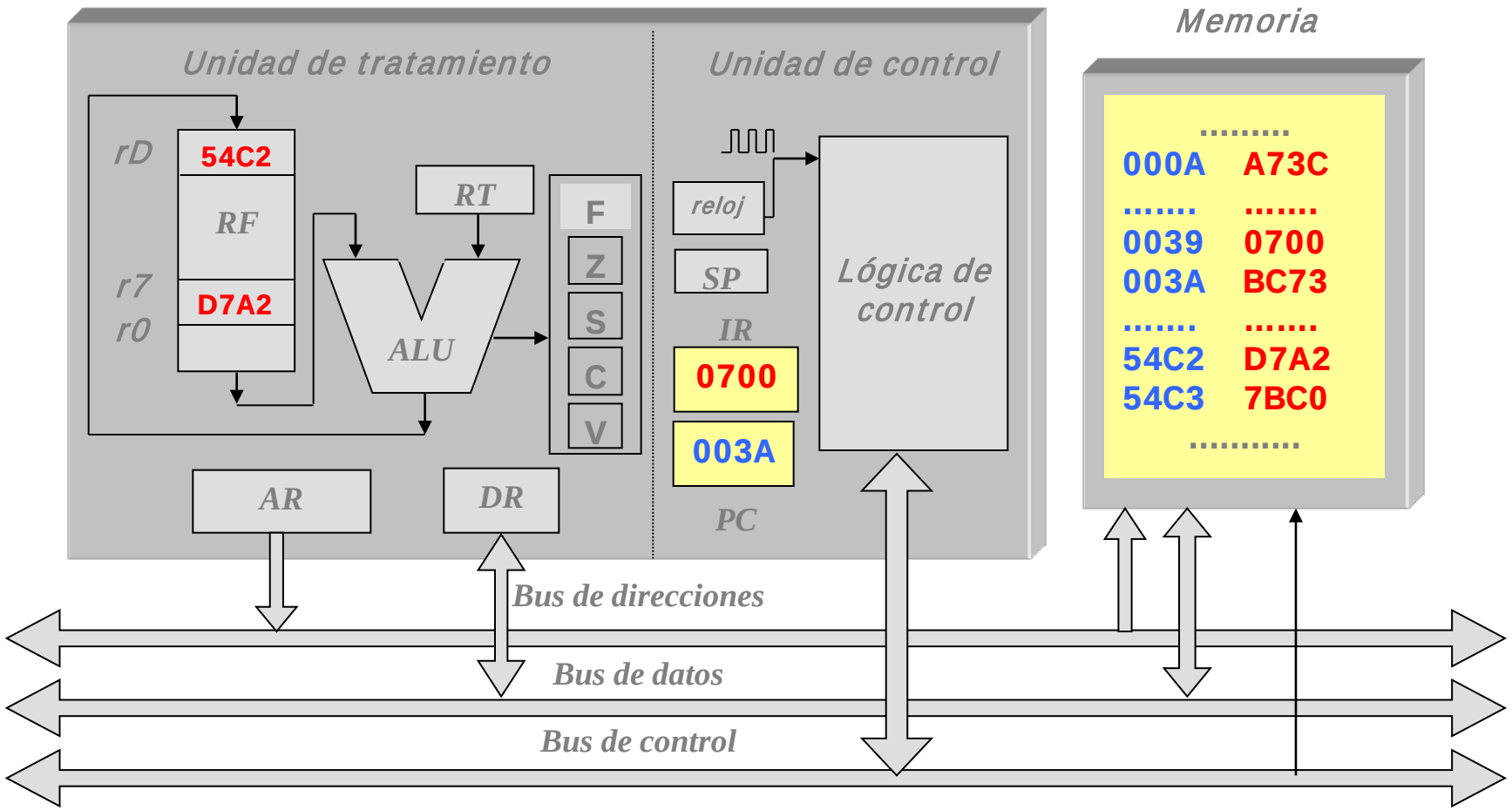


TOC: 2.1 Ejecución de Mov (rD), r7

Direc.	Contenidos	
0000	7AC4	
0007	65C9	
0039	0700	*Instr.*
003A	607D	
003B	2D07	
003C	C000	
54C2	D7A2	*Dato*
FFFF	3FC4	
rD	54C2	

Fase	Microoperación	Contenidos de los registros				
		PC	IR	AR	DR	r7
Valores iniciales						
Captación de instrucción	AR ← PC	0039		0039		
	DR ← M(AR)	0039		0039	0700	
	IR ← DR	0039	0700	0039	0700	
	PC ← PC+1	003A	0700	0039	0700	
Ejecución de instrucción	AR ← rD	003A	0700	54C2	0700	
	DR ← M(AR)	003A	0700	54C2	D7A2	
	r7 ← DR	003A	0700	54C2	D7A2	D7A2

TOC: 2.1 Situación después de la ejecución



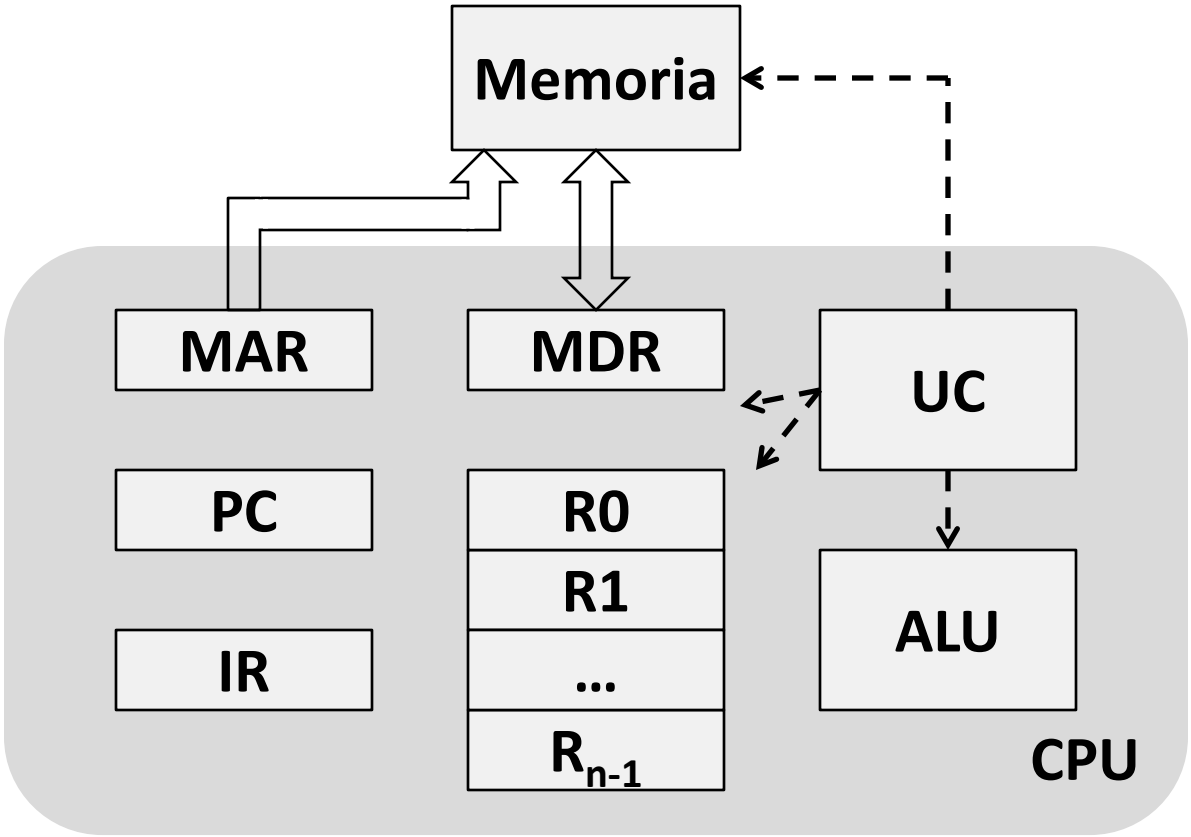
Ciclo ejecución instrucciones: fases

- **Programa en MP**
- **CPU (UC) tiene PC (*program counter*)**
  - posición MP de la siguiente instrucción
  - *Captación*: leer dicha posición IR←M[PC]
    - Usando MAR/MDR (Memory Address/Data Register), Instruction Register (IR)
    - Se interpreta como codop
    - Incrementar PC
  - *Decodificación*: desglosar codop/operandos(regs)
    - Posible etapa *Operando(M)*: captar dato/ incrementar PC
  - *Ejecución*: llevar datos ALU / operar
  - *Almacenamiento*: salvar resultado regs / MP

  - Nombres en inglés:
    - Fetch, Decode, Operand, eXecute, Write/Store

Ciclo ejecución instrucciones

- **Pensar tareas realizadas por UC para ejecutar instrucción**
  - Por ejemplo: Add A, R0
    - M[A] + R0 → R0
  - Detalles en [HAM03] Cap-1.3
  - Ejercicios similares en TOC §2.1



Add A, R0

- M[POS_A] + R0 → R0
  - Ensamblador traduce p.ej: POS_A=100
  - Valor anterior R0 perdido, el de POS_A se conserva
  - Arquitectura R/M
- **Pasos básicos de la UC**
  - PC apunta a posición donde se almacena instrucción
  - Captación: MAR←PC, Read, PC++, T_{acc}, MDR←bus, IR←MDR
  - Decodificación: se separan campos instrucción
    - Codop: ADD mem+reg → reg
    - Dato1: 100 direccionamiento directo, habrá que leer M[100]
    - Dato2: 0 direccionamiento registro, habrá que llevar R0 a ALU

    CPUs con longitud instrucción variable – dirección (100) en siguiente palabra
  - Operando: MAR←100, Read, T_{acc}, MDR←bus, ALU_{in1}←MDR
  - Ejecución: ALU_{in2}←R0, add, T_{alu}
  - Almacenamiento: R0←ALU_{out}

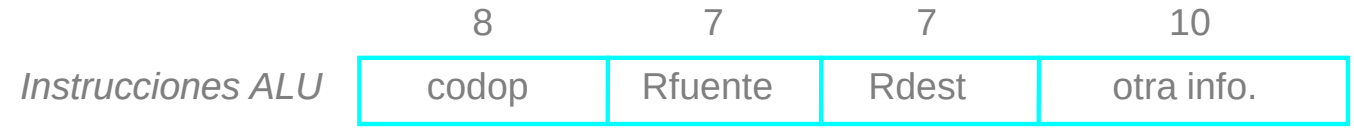
Otras consideraciones

- Arquitectura M/M: varias captaciones operando
  - PC++, si las direcciones ocupan más posiciones M
- Arquitectura R/M: cuando resultado en M (Add R0, A):
  - acceso memoria adicional (Write): M[MAR] ← MDR ← ALU_{out}
  - UC activa señal Write
- Arquitectura R/R: varias instrucciones (Load A, R1 / Add R1, R0)
  - efecto colateral: R1 perdido
  - ventaja: CPU más simple, veloz, pequeña (longitud/formato instrucción)

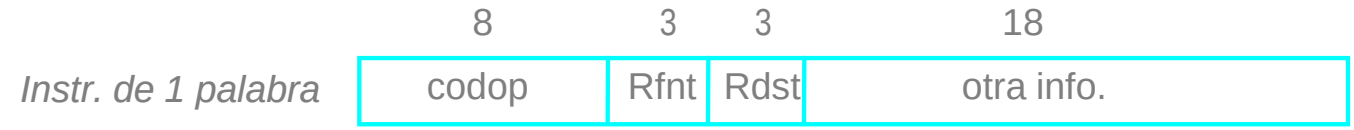
- Ciclo interrumpido por IRQ→ISR
  - mecanismo subrutina / salvar contexto (PC/estado)
  - salvo eso, comportamiento totalmente predeterminado por programa
- CPU completa (+L1+L2…+L3) en 1 chip VLSI
  - La CPU nunca lee de memoria un dato aislado
    - lee de cache
    - si hay fallo, se trae un bloque entero
    - se explica en clase así por motivos académicos

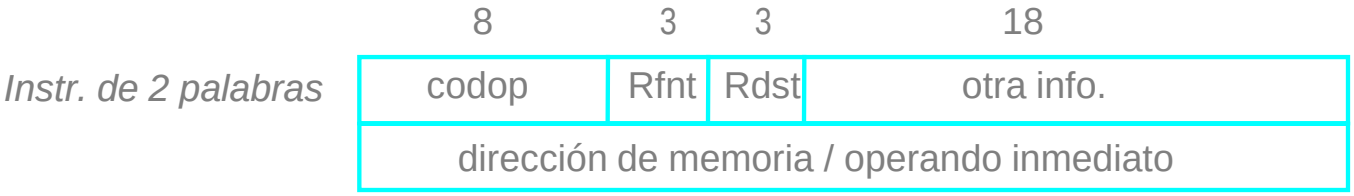
sobre Formatos de Instrucción

- **RISC**
  - Pocas instrucciones, pocos modos, muchos registros, 0/2-0/3
    - formato instrucción sencillo, tal vez sólo 2-3: transferencia, ALU, ctrl
    - ej: formato instrucciones ALU de un RISC 32bits 128regs tipo 0/2



- **CISC**
  - Muchas instrucciones y modos, menos registros, 1/2-1/3 (y resto)
    - varios formatos de instrucción, distintas longitudes, codops long. var. también





Formatos de Instrucción

- **ejemplo: IA-32 (Intel 64)**
  - Instrucciones de longitud variable, 1-15 bytes (memoria de bytes)
    - prefijos — modificar detalles de algunas instrucciones
    - codop — de 4bits a 3B + 3bits (campo reg en ModRM)
    - Mod-R/M — modo de direccionamiento (5 bits)
      - Reg — para indicar registro (hasta 8 regs)
    - SIB — para indicar 2 registros y escala índice (x1,x2,x4,x8)
    - desplazamiento — 32bits dirección memoria (u offset)
    - inmediato — 32bits valor operando

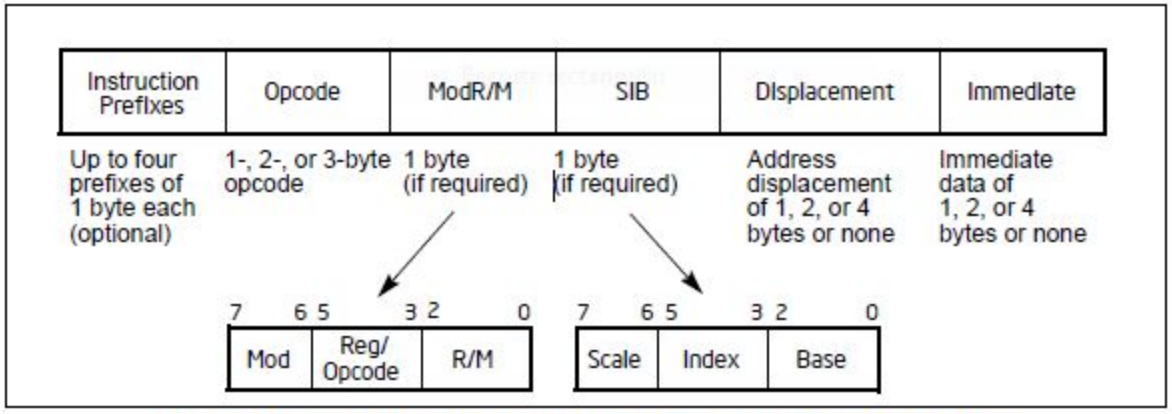


Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

Modos de Direccionamiento

- un número acompañando a un codop puede significar muchas cosas
  - según el formato de instrucción, la instrucción concreta, etc
- cada operando de la instrucción tiene su modo de direccionamiento

- **Inmediato** **(ej: $0, $variable)**
  - El número es el valor del operando
- **Registro** **(ej: %eax, %ebx…)**
  - El número es un índice de registro (ese registro es el operando)
- **Memoria** **(en general: disp(%base,%index,scale))**
  - instrucción lleva índices de registros y/o desplazamiento (dirección memoria)
  - La dirección efectiva (EA) es la suma de todos ellos. El operando es M[EA].

Directo	sólo dirección (disp)	op=M[disp]
Indirecto a través reg.	sólo registro (reg)	op=M[reg]
Relativo a base	registro y desplazamiento	op=M[reg+disp]
Indexado	índice (x escala) y dirección	op=M[disp + index*scale]
Combinado	todo	op=M[disp+base+idx*sc]

ej: modos IA-32

Código fuente ASM:

```
.section .text
_start: .global _start

mov     $0, %eax        # inm - registro
xor   %ebx, %ebx        # reg - registro
inc   %ebx              # reg
mov $array, %ecx        # inmediato - reg
mov  array, %edx        # directo   - reg

mov      (%ecx)       , %edx # indirecto
add      (%ecx,%ebx,4), %edx # combinado
add array(    ,%ebx,4), %edx # indexado
mov    -8(%ebp)       , %edx # rel.base
```

a veces puede ser ventajoso
+instrucciones -tamaño

inmediato ≠ directo

resto modos indirectos

Desensamblado del ejecutable:

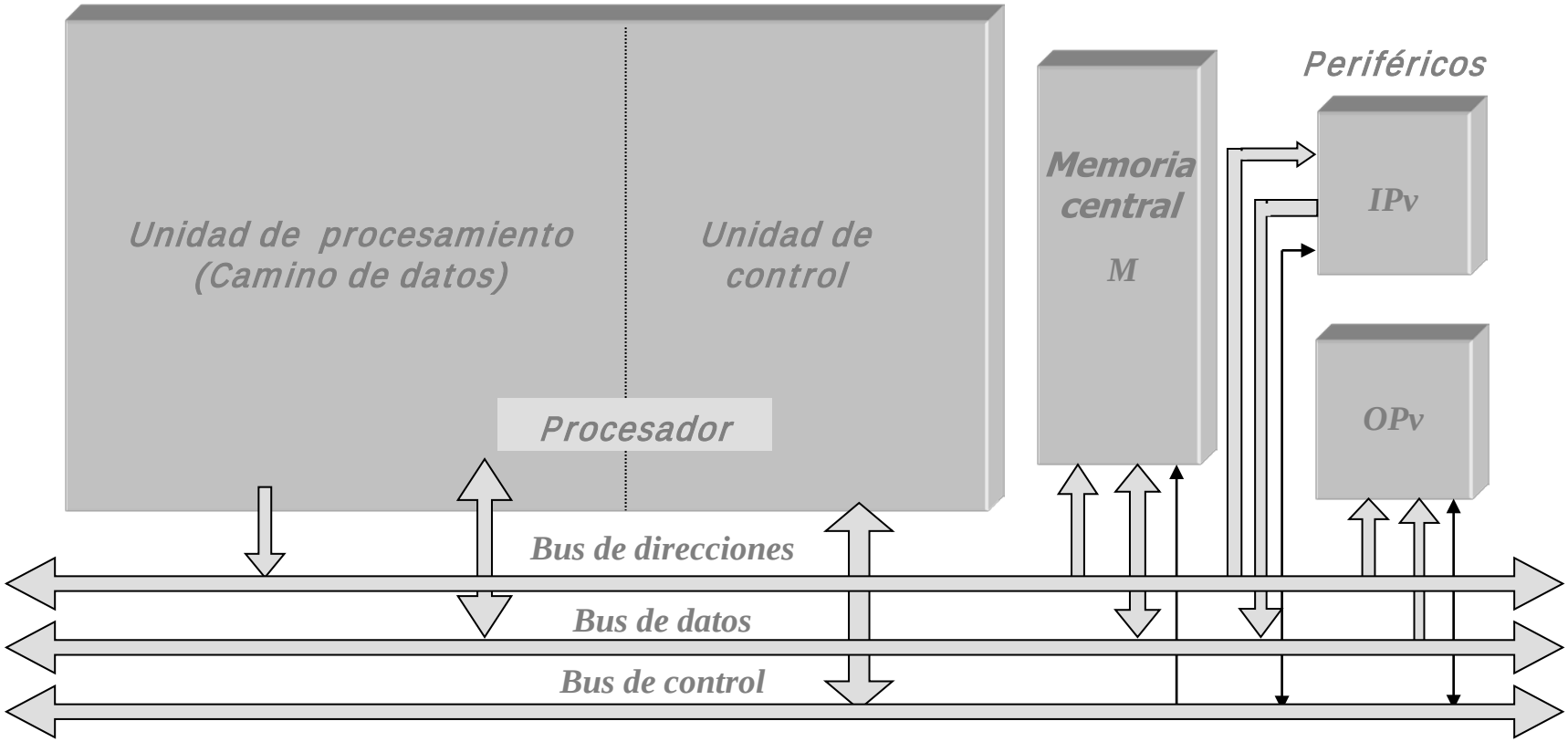
```
Disassembly of section .text:

08048074 <_start>:
 8048074: b8 00 00 00 00          mov    $0x0,%eax
 8048079: 31 db                   xor    %ebx,%ebx
 804807b: 43                      inc    %ebx
 804807c: b9 98 90 04 08          mov    $0x8049098,%ecx
 8048081: 8b 15 98 90 04 08       mov    0x8049098,%edx
 8048087: 8b 11                   mov    (%ecx),%edx
 8048089: 03 14 99                add    (%ecx,%ebx,4),%edx
 804808c: 03 14 9d 98 90 04 08    add    0x8049098(,%ebx,4),%edx
 8048093: 8b 55 f8                mov    -0x8(%ebp),%edx
```

Introducción

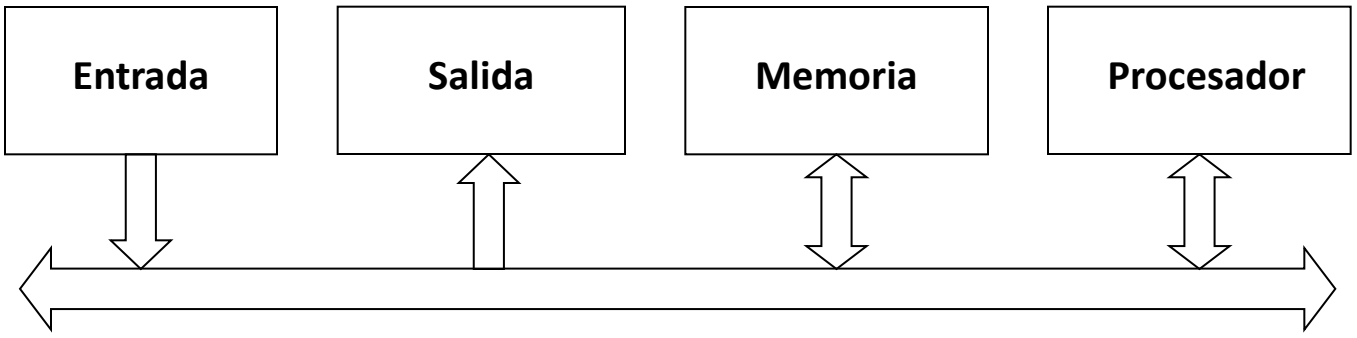
- Unidades funcionales
- Conceptos básicos de funcionamiento
- **Estructuras de bus**
- Rendimiento
- Perspectiva histórica

TOC: 2.1 Interconexión de las distintas unidades



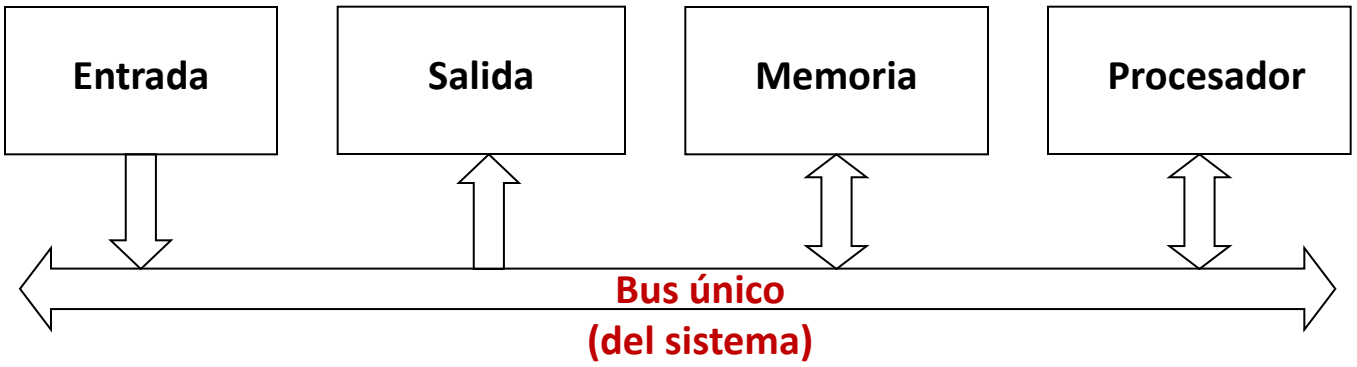
Estructuras de bus

- **Justificación buses (paralelos):**
  - E/S, M, CPU deben conectarse para pasar datos
  - Representación binaria / velocidad transferencia
    - palabras n bits M/ALU → bus datos n bits
    - direcciones m bits M → bus addr m bits
    - bus control para líneas UC (R/W, etc)



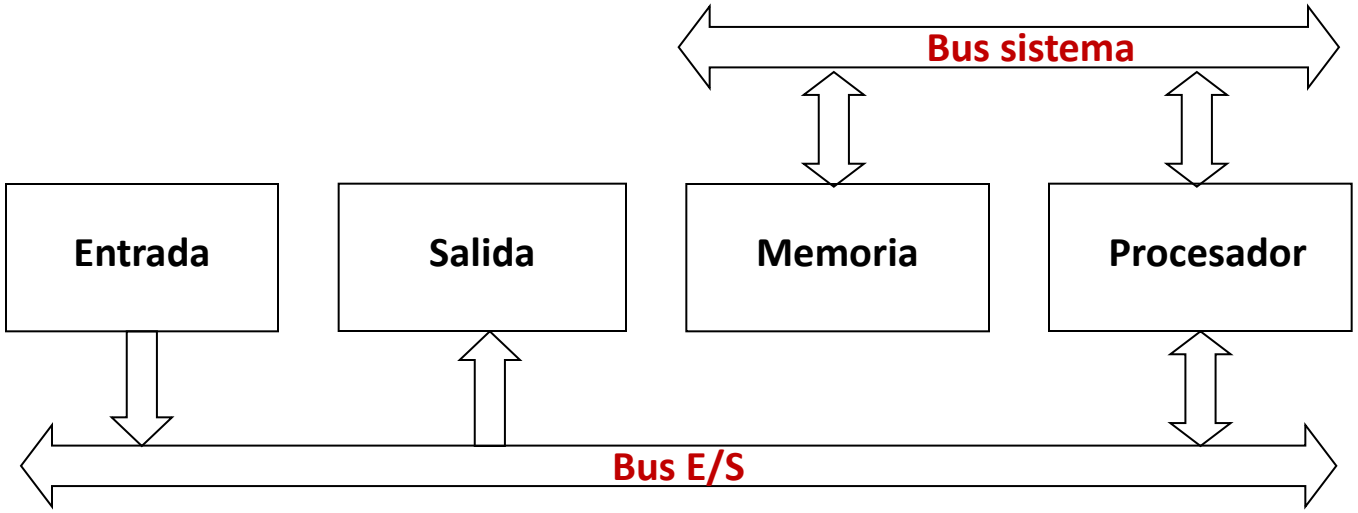
Estructuras de bus

- **Bus único**
  - CPU escribe bus dirección y control R/W
    - también escribe bus datos si Write
    - puede haber señales IOR/W separadas de MemR/W
  - E/S/M comprueban si es su dirección
    - sólo en ese caso se conectan al bus de datos
    - evitar cortocircuito bus datos
  - Ventaja: sencillez, bajo coste, flexibilidad conexión
    - fácil añadir más dispositivos
    - posibilidad líneas control arbitraje para varios master



Estructuras de bus

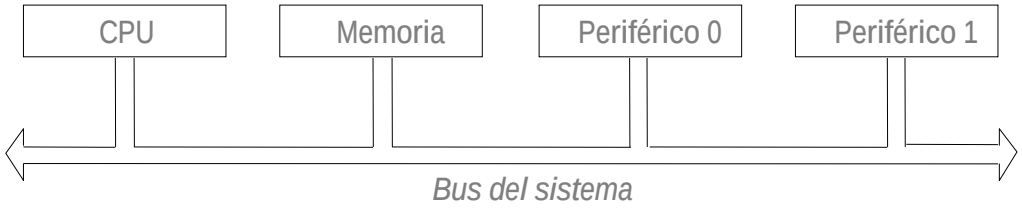
- **Buses múltiples**
  - típicamente: bus sistema (CPU-M) y bus E/S
    - también: múltiples buses E/S
      - separar dispositivos según velocidades
    - incluso: doble bus sistema
      - memoria datos/programa (arquitectura Harvard)
  - Ventajas: uno más rápido, ambos funcionan en paralelo
  - Inconveniente: coste, complejidad

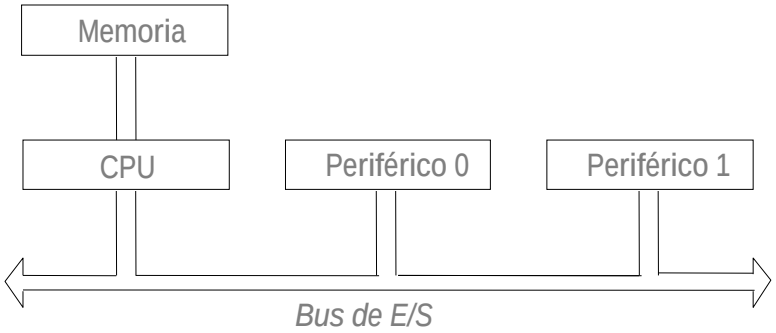


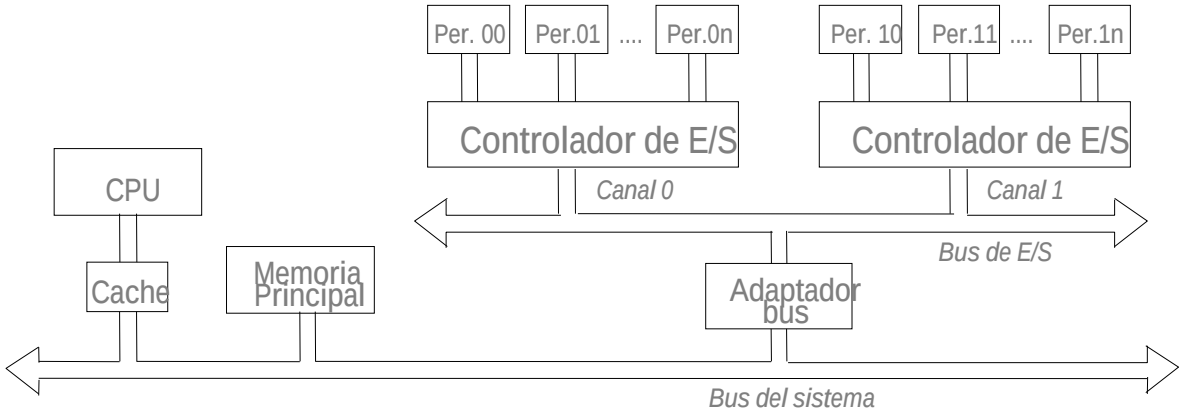
Adaptación de velocidades

- **Velocidad componentes**
  - CPU > Memoria >> E/S
- **Estados de espera:**
  - alargar ciclo bus si no se activa señal RDY (bus control)
  - permite conectar periféricos lentos a bus único
- **Buffers/IRQ:**
  - dispositivo lento almacena datos en buffer rápido
    - evita retrasar CPU con estados de espera
    - CPU se dedica a otra tarea mientras tanto
  - transferencia CPU a velocidad buffer (normal Memoria)
  - Write: CPU escribe buffer, dispositivo genera IRQ al final
    - Ej: impresora
  - Read: CPU encarga lectura, dispositivo hace IRQ cuando listo
    - Ej: escáner

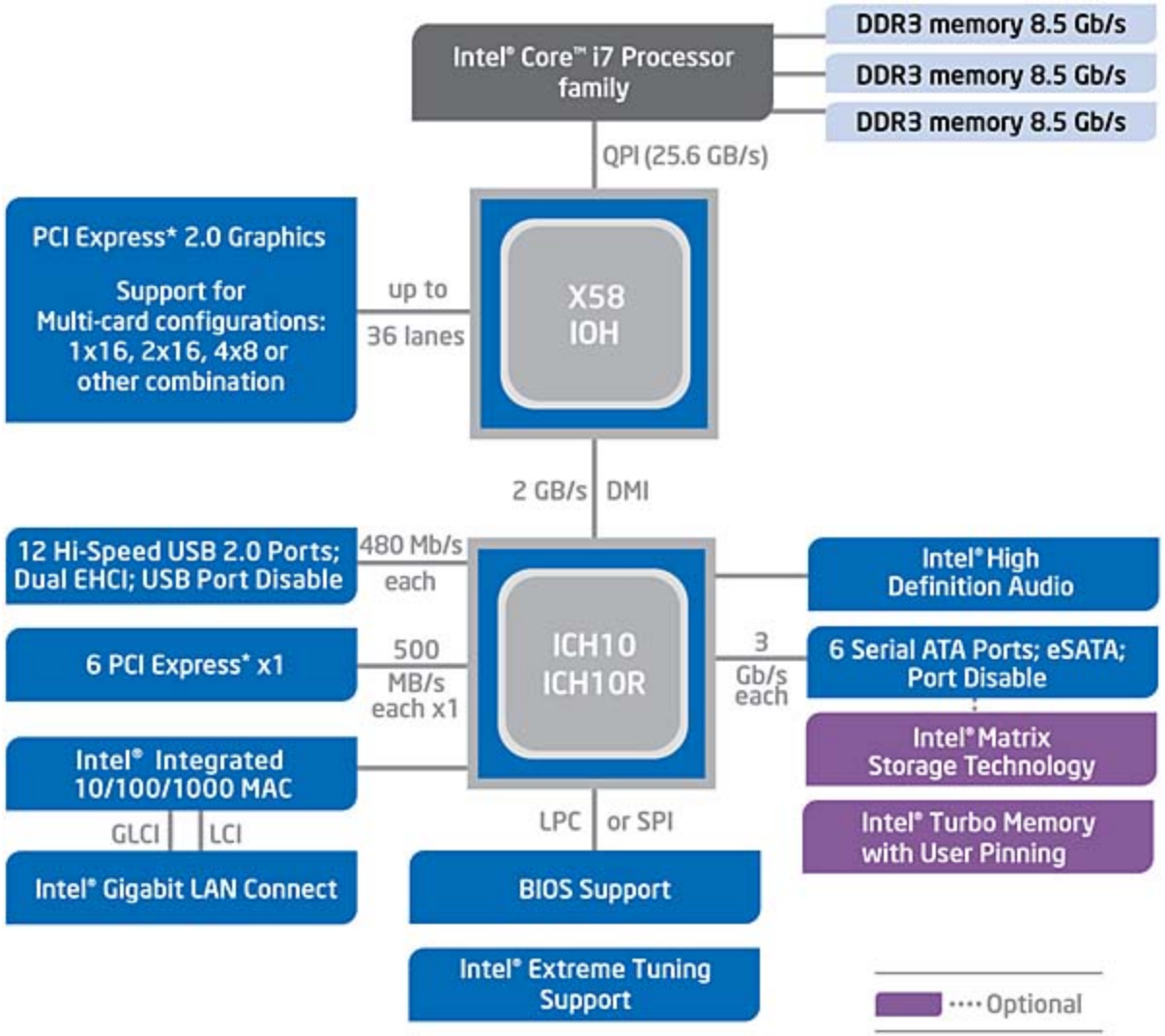
TOC: 2.4 Estructuras básicas de interconexión







TOC: 2.4 Estructuras básicas de interconexión



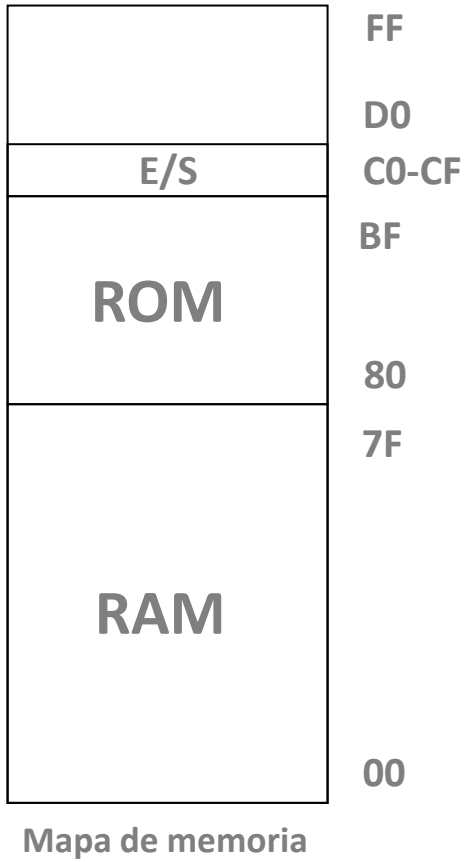
Decodificación

- **Evitar cortocircuito bus datos**
  - Suponer por ejemplo que CPU es único dispositivo activo del bus
    - es decir, que puede escribir bus Addr. y Ctrl.
      - cuando lo hace, se convierte en maestro del bus
      - Luego veremos multiprocesadores, controladores DMA, etc
      - varios activos requiere arbitraje para escoger maestro
  - demás dispositivos pasivos
    - sólo "escuchan" bus Addr, no pueden escribir, sólo leer
      - Cuando la CPU les habla, se convierten en esclavos
      - Es decir, se conectan al bus de datos y obedecen la orden R/W
  - #bits bus Addr. determina el "*espacio de memoria*"
- **Mapa de Memoria**
  - Dibujo de dónde está cada dispositivo en espacio Memoria
  - E/S puede ser "mapeada a memoria" o en espacio E/S separado

Decodificación

- **Ej: diseñar mapa memoria para**
  - 1 CPU 8bits
    - 8bits Addr. A7...A0
    - 8bits Data: D7...D0
  - 1 RAM 128 Bytes
    - decodificada en 0...127
  - 1 ROM 64 Bytes
    - a continuación
  - 1 Puerto Serie 16 Regs
    - 16 puertos de 8 bits
    - a continuación

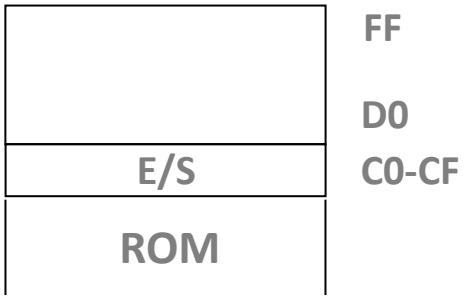
(ejemplo académico, realmente no existen tamaños tan pequeños)

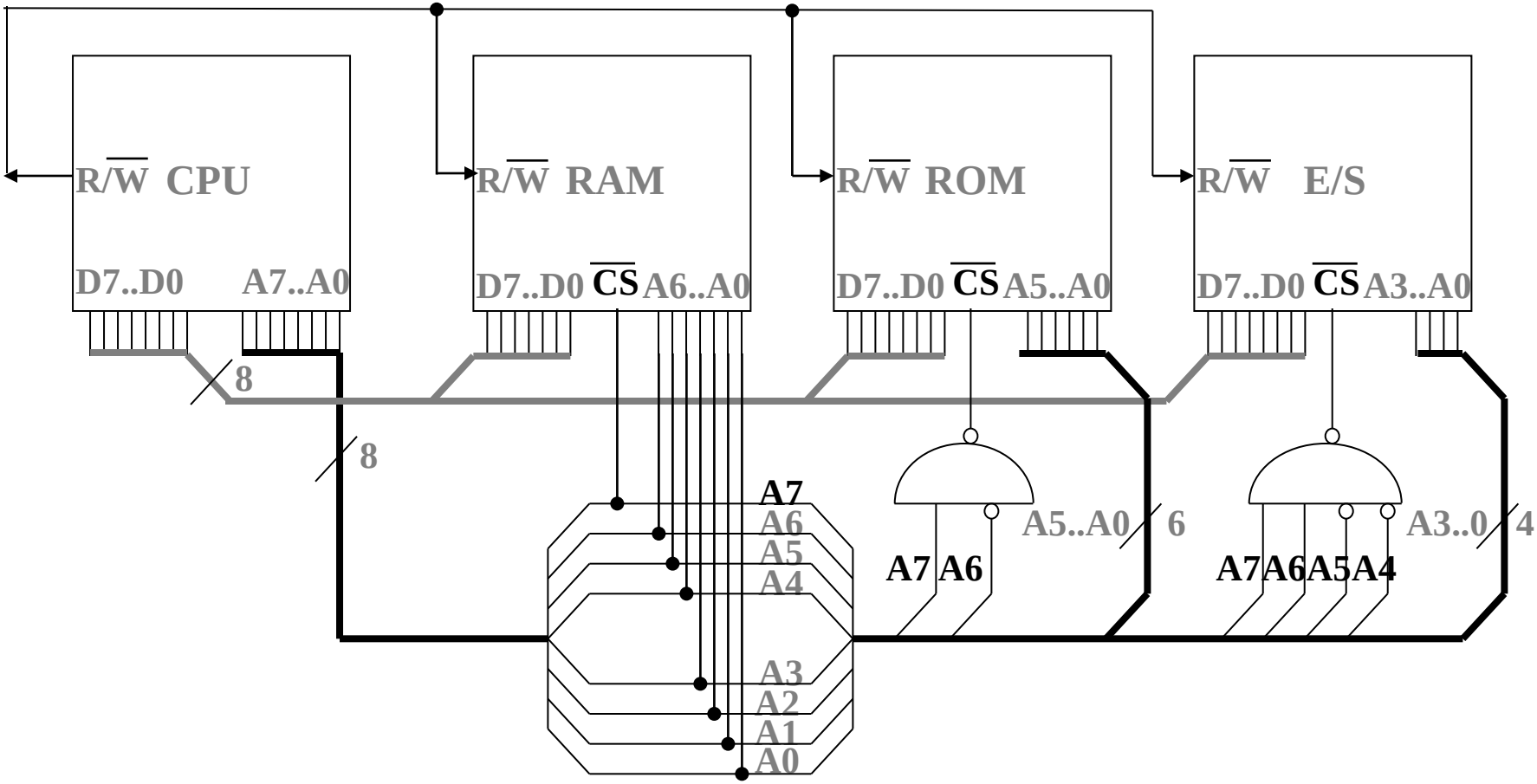


Mapa de memoria

Decodificación completa

- se usan todos los bits Addr.
  - MSB decodifican el dispositivo/módulo (CS)
  - LSB direccionan dentro del dispositivo (Addr)

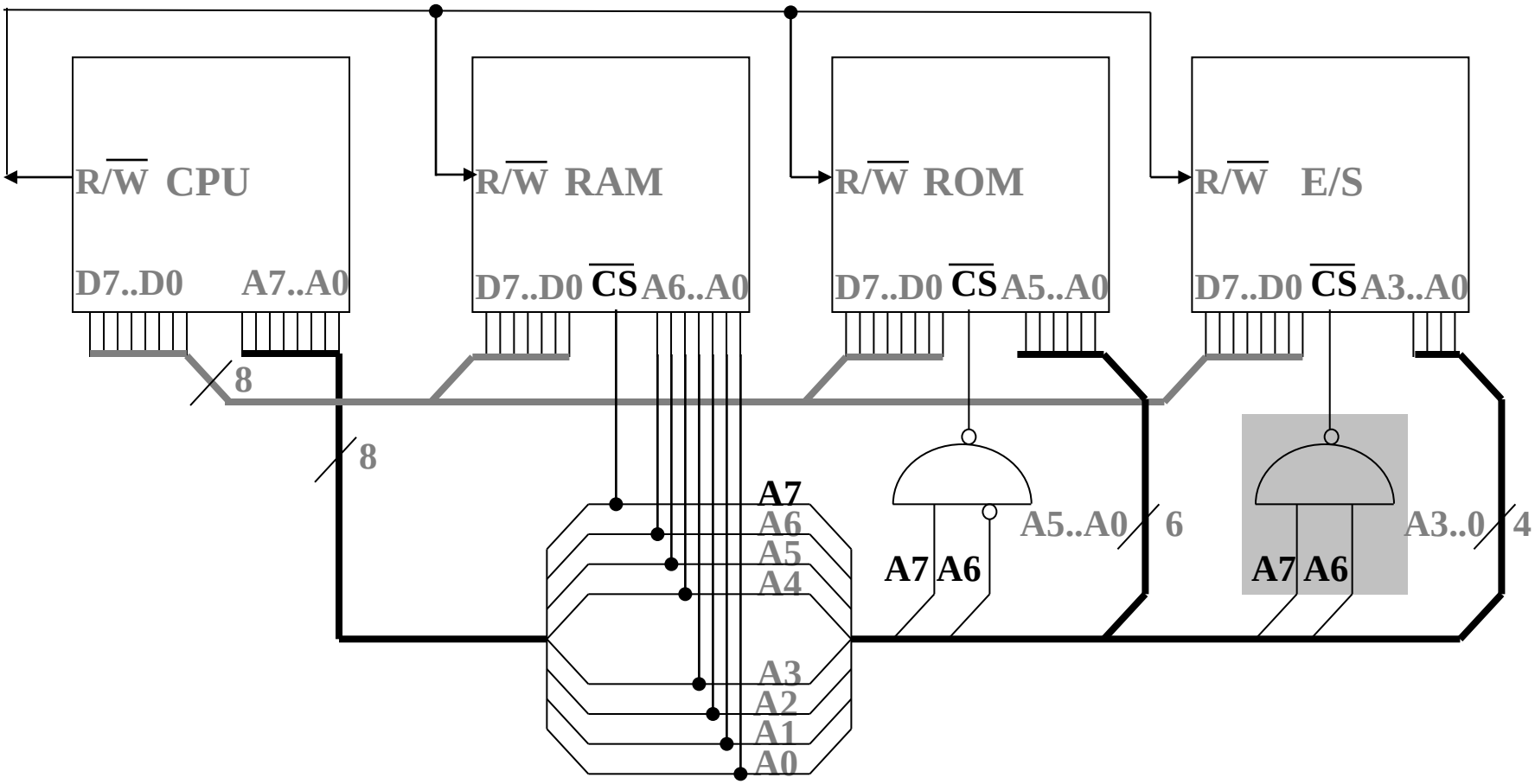




Decodificación parcial

- algunos (m) bits Addr. sin usar
  - El dispositivo aparece repetido 2^m veces en Memoria

E/S	F0-FF
E/S	E0-EF
E/S	D0-DF
E/S	C0-CF
ROM	



Software de sistema

- **Cómo conseguir crear programa → MP → ejecutar**
  - Software de sistema implicado:
    - Shell (intérprete comandos): recibe órdenes usuario
      - EXEC: llamada para cargar y ejecutar aplicación
    - Editor: permite crear código fuente (y archivar!)
    - Compilador / Enlazador: código objeto / ejecutable
    - Sistema de ficheros (crear, copiar, abrir, leer)
      - desde Shell / desde programa usuario
    - Sistema E/S
- **Cómo se consigue encender → arrancar SO**
  - soporte hardware: dirección de bootstrap
  - [Boot-P]ROM en espacio memoria apuntado
  - Bootloader primario, carga arranque HD/FD/CD…
  - Bootloader secundario (menú escoger SO, etc)…

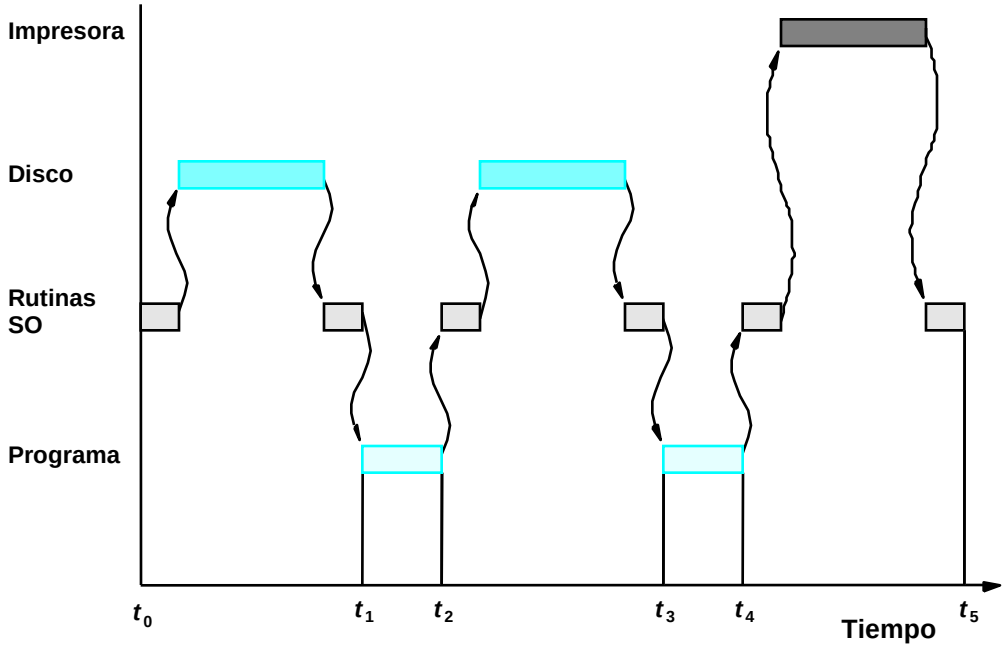
Software de sistema

- **Llamadas al sistema (ej: aplicación lee fichero/calcula/imprime)**
  - usuario teclea nombre aplicación → EXEC
    - el shell invoca EXEC, proporcionando nombre fich.
  - EXEC carga aplicación HD → M, pasa control
    - el propio SO proporciona zona M cargar aplicación
    - EXEC retorna a aplicación, y ella retornará a shell
  - aplicación invoca OPEN/READ/CLOSE
    - proporciona zona memoria donde leer contenido
  - aplicación calcula resultado, invoca PRINT/EXIT
    - proporciona datos a imprimir / código retorno

- **SO gestiona recursos (especialmente multiuser/multitask)**
  - ej: solapar E/S final con carga siguiente tarea
  - ej: conmutar proceso en cuanto haga E/S

Software de sistema

- **Pensar tareas realizadas por SO para ejecutar aplicación**
  - Por ejemplo: leer datos HD, cálculos, imprimir resultados
  - Pensar entonces cómo solapar varias de esas aplicaciones
  - Detalles en [HAM03] Cap-1.5

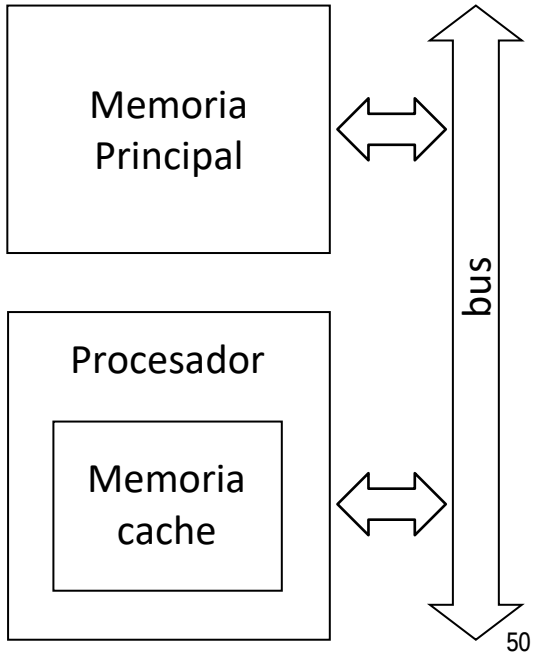


Introducción

- Unidades funcionales
- Conceptos básicos de funcionamiento
- Estructuras de bus
- **Rendimiento**
- Perspectiva histórica

Rendimiento

- **Medida definitiva: tiempo ejecución programa**
  - Problema: ¿Cuál programa? Acordar benchmarks
  - Depende de diseño CPU, repertorio instrucciones…
    - y del compilador!!! (benchmarks en lenguaje alto nivel)
    - y versión SO, librerías, etc.
- **Ejemplo anterior: t5-t0 incluye HD, LPR**
  - Tiempo transcurrido (wall-clock time)
  - Mide rendimiento sistema completo
    - Influido por prestaciones CPU, HD, LPR, etc
- **Benchmarks CPU ejercitan sólo CPU**
  - Tiempo de procesamiento (CPU time)
  - Influido por prestaciones CPU, M, caches, buses
    - cache conserva lo accedido recientemente/más rápida
    - ventaja en ejecución bucles, p.ej.



Rendimiento

- **Reloj del procesador**
  - UC emplea varios ciclos de reloj en ejecutar una instrucción
  - pasos básicos 1 ciclo (conmutar señales control)
  - Frecuencia R = 1/P
    - 500MHz = 1 / 2ns
    - 1.25GHz = 1 / 0.8ns

- **Ecuación básica de rendimiento**
  - T tiempo para ejecutar programa benchmark
  - N instrucciones (recuento dinámico bucles/subrutinas)
    - N no necesariamente igual a #instr. progr. objeto.
  - S ciclos/instr. (“pasos básicos” de media)

$$T = \frac{N \times S}{R} \quad \begin{matrix} \text{ciclos} \\ \text{ciclos/s} \end{matrix}$$

Rendimiento

Ecuación básica de rendimiento

$$T = \frac{N \times S}{R} \quad \begin{matrix} \text{ciclos} \\ \text{ciclos/s} \end{matrix}$$

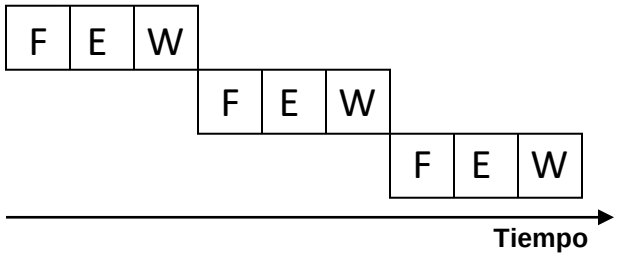
- Ideal: N y S ↓↓, R ↑↑
  - N (instrucciones) depende de compilador/repertorio
  - S (ciclos/instr) depende de implementación CPU
  - R (MHz - GHz) depende de tecnología (y diseño CPU)

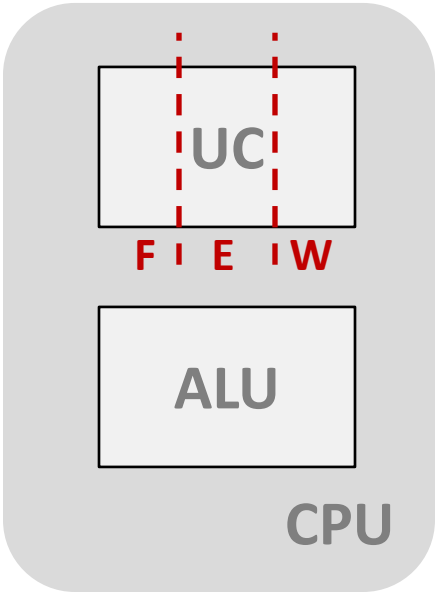
- alterar uno modifica los otros
  - aumentar R puede ser a costa de aumentar S
  - lo importante es que al final T↓

Segmentación de cauce (intenta que S≈1)

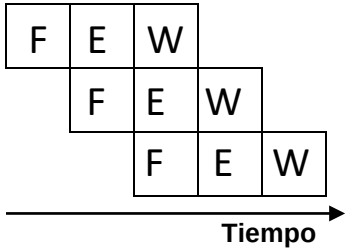
- NxS es suponiendo ejecución individual instrucciones

ADD R1,R2, R3
MUL R4,R5, R5
SUB R3,R5, R5





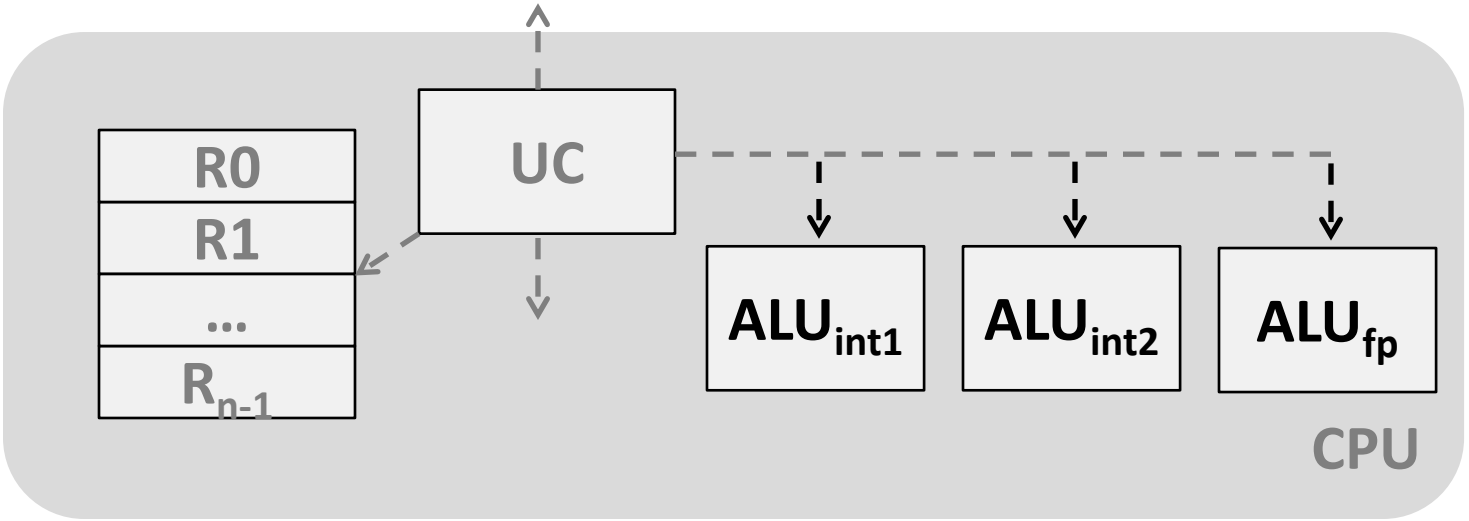
- Pero las distintas etapas hacen tareas distintas
  - UC puede tener circuitería separada para cada etapa:
    - Fetch: captación
    - Exec: ejecución
    - Write: actualización registro



- una vez lleno el cauce, valor efectivo S=1 ciclo/instr
  - dependencias datos (ej: MUL-SUB arriba, dependencia R5)
  - competición recursos (ej: almacenar resultado M/fetch instrucción+2)
  - saltos (pipeline flush)
  - S≥1, S≈1

Funcionamiento superescalar (que S<1)

- Conseguir paralelismo a base de reduplicar UFs (unidades funcionales)
  - ej: 2 ALU enteros, 2 ALU FP
    - emitir hasta 4 operaciones simultáneas (2int+2fp)
    - si orden apropiado instrucciones (en secuencia programa)
  - combinado con segmentación, puede hacer S<1
    - se completa más de 1 instrucción por ciclo
- común en CPUs actuales. Dificultades:
  - emisión desordenada
  - corrección (mismo resultado que ejecución escalar)



Otras formas de reducir T

- **Velocidad del reloj (R↑, S/R)**
  - Tecnología ↑ ⇒ R↑
    - Si no cambia nada más, Rx2 ⇒ T/2? (T = NS / R)
    - Falso: Memoria también Rx2 !!! o mejorar cache L1-L2
  - Alternativamente, S↑ ⇒ R↑
    - "supersegmentación", reducir tarea por ciclo reloj
    - difícil predecir ganancia, puede incluso empeorar

- **Repertorio RISC/CISC (N·S)**
  - RISC: instr. simples para R↑↑, pero S↓ ⇒ N↑
  - CISC: instr. complejas para N↓↓, pero S↑
    - corregir S↑ con segmentación ⇒ competición recursos
  - actualmente técnicas híbridas RISC/CISC

Otras formas de reducir T

- **Compilador (N↓)**
  - optimizador espacial (N↓) o temporal (NxS ↓)
    - usualmente contrapuestos
  - espacial: requiere conocimiento arquitectura
    - repertorio, modos direccionamiento, alternativas traducción…
  - temporal: requiere conocimiento detallado organización
    - reordenación instrucciones para ahorrar ciclos
    - evitar competición recursos
  - optimización debe ser correcta (mismo resultado)

Medida del rendimiento

- **Interesante para:**
  - diseñadores CPUs: evaluar mejoras introducidas
  - fabricantes: marketing
  - compradores: prestaciones/precio

- **Benchmark: 1 único programa acordado ?!?**
  - programas sintéticos no predicen bien T_{app}
  - programas reales muy específicos
  - colección programas considerados "frecuentes" (representativos)
  - reducir a un único número usando media geométrica
    - evitar influencia computador referencia

Medida del rendimiento

- **SPEC: System Performance Evaluation Corporation**
  - tests: CPU92, CPU95, CPU2000, CPU2006 (CPUv6)
  - CPU2006:
    - Referencia: WS UltraSPARC II 300MHz
    - CINT2006: gzip, bzip2, gcc, chess, gene seq., Perl… (12)
    - CFP2006: CFD, chromodynamics, ray-tracing, meteo… (17)

    $vel_{gzip} = Tref_{gzip}/T_{gzip}$ (vel=50 $\Rightarrow$ 50x uSPARC II)

    $vel_{SPEC} = \sqrt[n]{\prod_i vel_{prgi}}$ (n=12/17, media geom.)

- **mide efecto combinado**
  - CPU, M, SO, compilador
  - http://www.spec.org (no es gratuito)

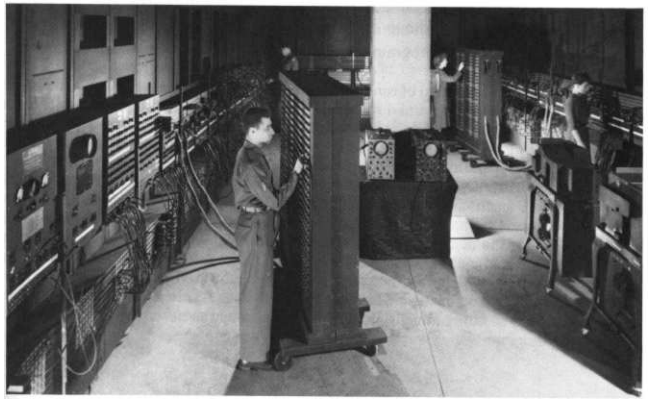
Introducción

- Unidades funcionales
- Conceptos básicos de funcionamiento
- Estructuras de bus
- Rendimiento
- **Perspectiva histórica**

Perspectiva histórica

- **2ª Guerra Mundial**
  - Tecnología relés electromagnéticos $T_{conmut.}$=O(s)
    - Previamente: engranajes, palancas, poleas
  - Tablas logaritmos, aprox. func. trigonométricas
  - Generaciones 1-2-3-4ª 1945-55-65-75-etc

- **1ª Generación (45-55): tubos de vacío**
  - von Neumann: concepto prog. almacenado
  - tubos vacío 100-1000x $T_{conmut.}$=O(ms)
  - M: líneas retardo mercurio, núcleos magn.
  - E/S: lect/perf. tarjetas, cintas magnéticas
  - software: lenguaje máquina / ensamblador
    - 1946-47 ENIAC UNIVAC
    - 1952-57 EDVAC UNIVAC II
    - 1953-55 IBM 701 702



Perspectiva histórica

- **2ª Generación (55-65): transistores**
  - invento Bell AT&T 1947 $T_{conmut.}$=O(µs)
  - E/S: procesadores E/S (cintas) en paralelo con CPU
  - software: compilador FORTRAN
    - 1955-57 IBM704 DEC PDP-1
    - 1964 IBM 7094

- **3ª Generación (1965-75): Circuito Integrado**
  - velocidad CPU/M ↑ $T_{conmut.}$=O(ns)
  - arquitectura: µProgr, segm.cauce, M cache
  - software: SO multiusuario, memoria virtual
    - 1965 IBM S/360
    - 1971-77 DEC PDP-8

Perspectiva histórica

- **4ª Generación (75-…): VLSI**
  - µProcesador: procesador completo en 1 chip
    - MP completa en uno o pocos chips
    - Intel, Motorola, AMD, TI, NS
  - arquitectura: mejoras segm. cauce, cache, M virtual
  - hardware: portátiles, PCs, WS, redes
  - mainframes siguen sólo en grandes empresas
    - 1972-74-78 i8008 i8080 i8086
    - 1982-85-89 i80286 i80386 i80486

- **Actualidad**
  - Computadores sobremesa potentes/asequibles
  - Internet
  - Paralelismo masivo (Top500, MareNostrum, Magerit)
    - 1995-97-99-01 Pentium PII PIII P4
    - 2004-06-08 Pentium 4F, Core 2 Duo, Core i7
    - 2011-15-20 Core i7 2nd-6thgen, Kaby/Coffee/Cannon/Ice Lake

Introducción

- **Unidades funcionales**
  - E/S, M, CPU (ALU+UC)
  - Memoria de bytes, alineamiento, ordenamiento
  - Clasificación arq. m/n, pila, acumulador, RPG (R/R, R/M, M/M)
  - Repertorios RISC/CISC, modos de direccionamiento
- **Conceptos básicos de funcionamiento**
  - Ciclo de ejecución
- **Estructuras de bus**
  - Bus único, buses múltiples, decodificación parcial/completa
- **Rendimiento**
  - Software de sistema, ecuación básica rendimiento, benchmarks
  - Segmentación, funcionamiento superescalar, SPEC
- **Perspectiva histórica**
  - generaciones

Guía de trabajo autónomo (4h/s)

- **Estudio**
  - Cap.1 Hamacher (incluye problemas)
- **Lectura**
  - Guión de la Práctica 2
  - Cap.3 CS:APP (Bryant/O'Hallaron)
- **Para los entusiastas: Ubuntu en el portátil** (Ubuntu LTS 18.04 en ETSIIT)
  - Posibilidades de usar Ubuntu en portátil:
    - Instalación directa (además de, o en lugar de, Windows)
    - VirtualBox + Ubuntu LTS (es +complicado, pero +ventajoso)
      - » no requiere rebotar, no toca MBR, se puede usar Windows a la vez
    - instalar paquetes **g++/make/ghex** (usar **apt** o **Synaptic**), y **default-jre** (para eclipse)
    - instalar snap **eclipse 4.8** (usar **snap** o **UbuntuSoftware**) (evitar paquete eclipse 3.8)
    - comprobar firewall con "sudo ufw [status | enable]"
  - Instalárselo e intentar Ejercicios 1-4 del guión P2