EC 21-22

Tema 1: Introducción

0. Vocabulario

- Arquitectura: aspectos necesarios para redactar un programa ensamblador correcto. Incluye: registros CPU, repertorio de instrucciones, modos de direccionamiento
- Organización (del computador, de CPU, de ALU): estructura + funcionamiento
  - Estructura: componentes y interconexión (foto fija)
  - Funcionamiento: dinámica de procesamiento de la información
- Computador: E/S + Memoria + CPU [ALU+UC]

1. Unidades funcionales

La arquitectura Von Neumann distingue 5 componentes:

- Entrada: codificar, digitalizar, transmitir (lectura)
  - Teclado, ratón, red, disco...
- Memoria: almacenar
  - Programas, datos de entrada, resultados de operaciones...
- CPU: unidad de procesamiento central, procesa información E/M ejecutando programas
  - ALU: unidad aritmético-lógica, operaciones
  - UC: unidad de control, circuitos
- Salida: codificar, almacenar, transmitir (escritura)
  - Pantalla, impresora, disco, red...

La memoria almacena instrucciones y datos.

Las instrucciones pueden ser...

	Ejemplos	**Unidades**
De transferencia	`mov, in, out`	M, E/S
Operaciones	`add, and`	ALU
De control	`jmp, call, ret, set`	UC

En memoria, todo son datos:

- Interpretados como programas si se leen en etapa de captación
- Compilar, desensamblar... es código usado como datos

El lenguaje máquina es el único que entiende la CPU. Las instrucciones se forman por bits agrupados en campos:

- Campo de código de operación, que indica la operación correspondiente a la instrucción
- Campos de dirección, que especifican los lugares donde se encuentran o donde ubicar los datos con los que se opera

Entrada-Salida

- **Entrada**:
  - Codificar información (operador: memoria o cpu)
    - Teclado, ratón, micrófono
  - Recuperar información previamente almacenada
    - CD/DVD, lector de tarjetas
  - Comunicar ordenadores entre sí
    - Tarjeta de red, módem
- **Salida**:
  - Codificar información resultado (operador: humano)
    - Impresora, pantalla
  - Almacenaje
    - Pen, CD/DVD
  - Comunicar con otros ordenadores
    - Red, módem
  - Muchos dispositivos son duales E/S (aceptan lectura y escritura)

Memoria

- **Almacenamiento primario** (memoria semiconductora)
  - Palabras de n bits accesibles en 1 operación básica R/W
    - Longitudes típicas: 16 a 64 bits
    - Muy frecuente memoria en Bytes para temas de alineamiento y ordenación
  - Accesible aleatoriamente (RAM) por dirección (posición)
    - Bus de direcciones, bus de datos, bus de control (R/W), tiempo de acceso
    - Tamaños de memoria típicos: de 8 a 32 GB
    - Tiempos de acceso típicos en nanosegundos (ns)
    - Jerarquía de memoria: caché > L1 > L2 > L3 > Memoria principal
      - Programa almacenado en MP
- **Almacenamiento secundario** (óptico/magnético)
  - No es memoria Von-Neumann como tal, es E/S
  - El fichero swap se considera parte de la jerarquía de memoria

CPU

Componente más rápido del ordenador

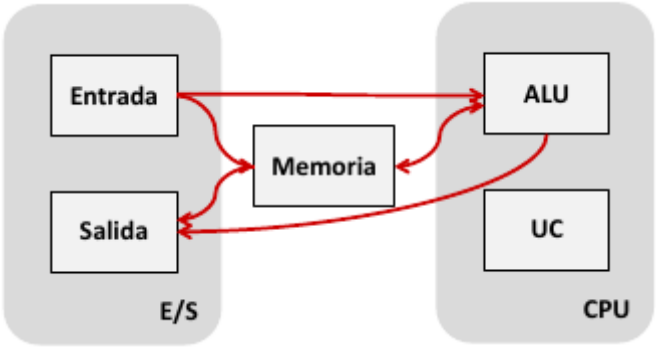
- **ALU:**
  - Registros de almacenamiento más rápidos que la L1
    - Operandos y resultados desde/hacia memoria o registros
    - Arquitecturas R/R, R/M, M/M
  - Operaciones aritméticas (`add, mul, div...`), enteras y de coma flotante
  - Operaciones lógicas (`and, or...`), bit a bit
- **UC:**
  - Controla todos los demás circuitos (ALU, M, E/S)

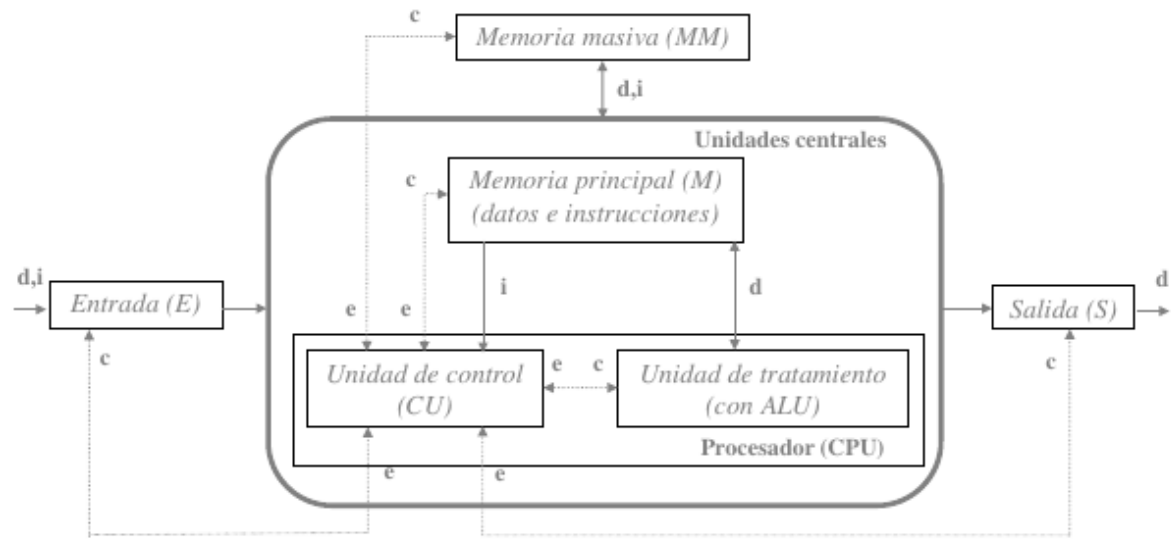
- Sigue lo que le indica el programa almacenado en MP
- Instrucciones de transferencia = señales de contol a M y E/S
- Instrucciones aritmético-lógicas = señales de control a ALU
- Temporización de señales (dirección, datos, R/W)

Posibilidades funcionamiento:

- Programa E → MP
- Datos E → MP
- Ejecución programa:
  - Datos → ALU → resultados
  - E/M → ALU → S/M
- Resultados → S

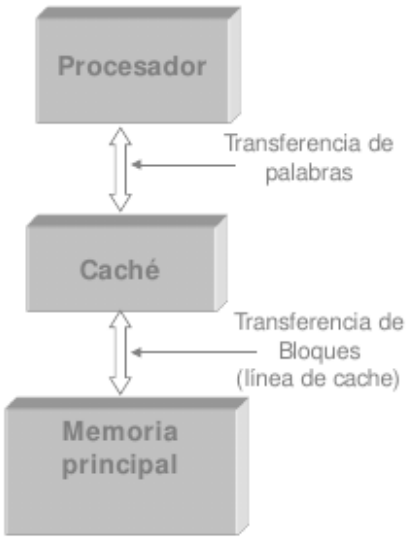
Todo controlado según indique el programa almacenado en MP, interpretado por UC

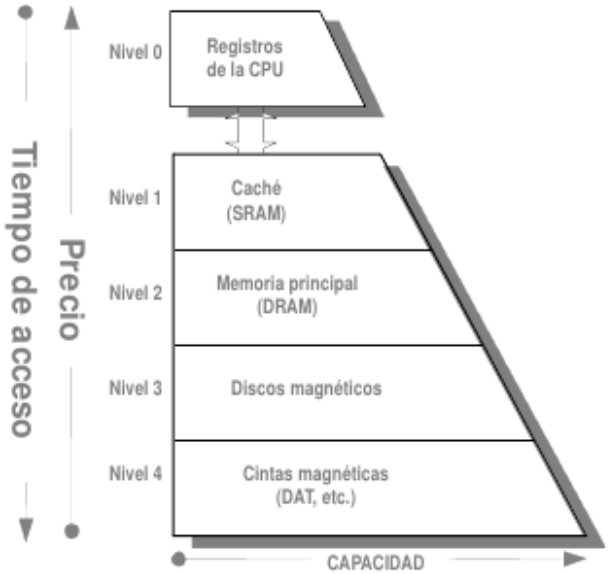




Sobre memoria...

Jerarquía de memoria





Organización en bytes

¿Tamaño posición de memoria = registro de CPU (o sea, longitud de palabra)?

Eso sería lo ideal, pero no es frecuente. Normalmente, son posiciones de 1B (direccionamiento por bytes).

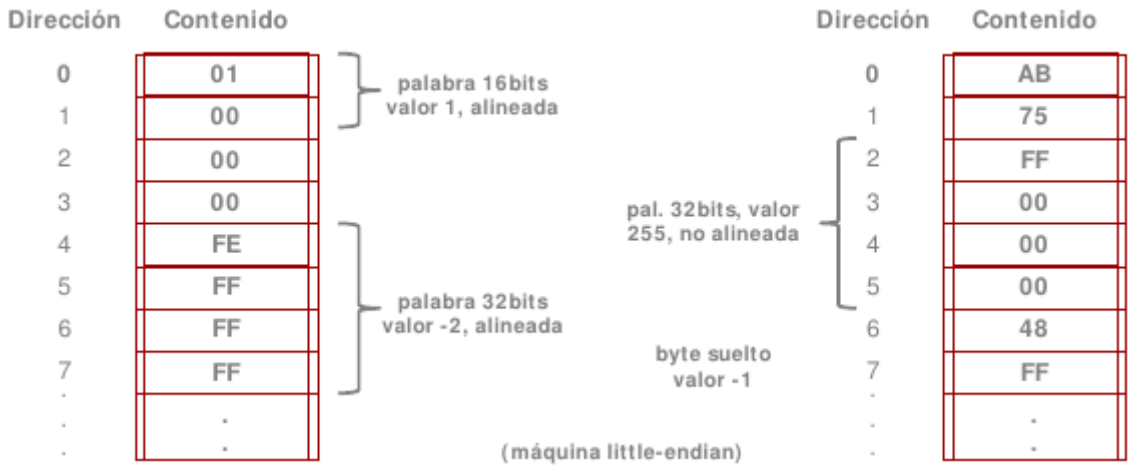
Sobre memoria de Bytes...

Ordenamiento en memoria en Bytes

- Little-endian (criterio del extremo menor)
  - Primero se almacena el byte menos significativo (LSB en la posición M más baja, MSB en posición más alta)
- Big-endian (criterio del extremo mayor)
  - Primero se almacena el byte más significativo (MSB en la posición M más baja)

Alineamiento en memoria de Bytes

Una palabra de n bytes está alineada si comienza en una dirección múltiplo de n. Algunas memorias exigen alineamiento para acceder a la memoria (o si no hay error en el bus), mientras que otras acceden más rápido si hay alineamiento (pero no es obligatorio). La palabra tampoco debería cruzar línea de caché o de página.



Clasificaciones M/N y pila-acumulador-RPG

Los tipos de CPU se determinan por los operandos de las instrucciones ALU, también suele afectar a los operandos de instrucciones de transferencia.

La clasificación M/N es: operaciones ALU que admiten N operandos, M de ellos de memoria. Algunas 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 memoria)
  - Repertorio: load M, store M, add M, and M
- **Máquinas RPG (Registros de propósito general) x/2 o x/3:**
  - Múltiples acumuladores
  - Repertorio: move R/M R/M, Add R/M R/M R/M

RPG: clasificación R/M

- **Arquitecturas R/R**
  - 0/2 o 0/3
  - `Add r1 r2 r3`
- **Arquitecturas R/M**
  - 1/2 o 1/3 (2/3 poco frecuente)
  - `Add r1 A`
  - Típico de CISC
- **Arquitecturas M/M**
  - 2/2 o 3/3 (poco frecuente)
  - `Add A B`
  - Permite operar directamente en memoria
  - Demasiados accesos a memoria por instrucción máquina

Sobre los repertorios...

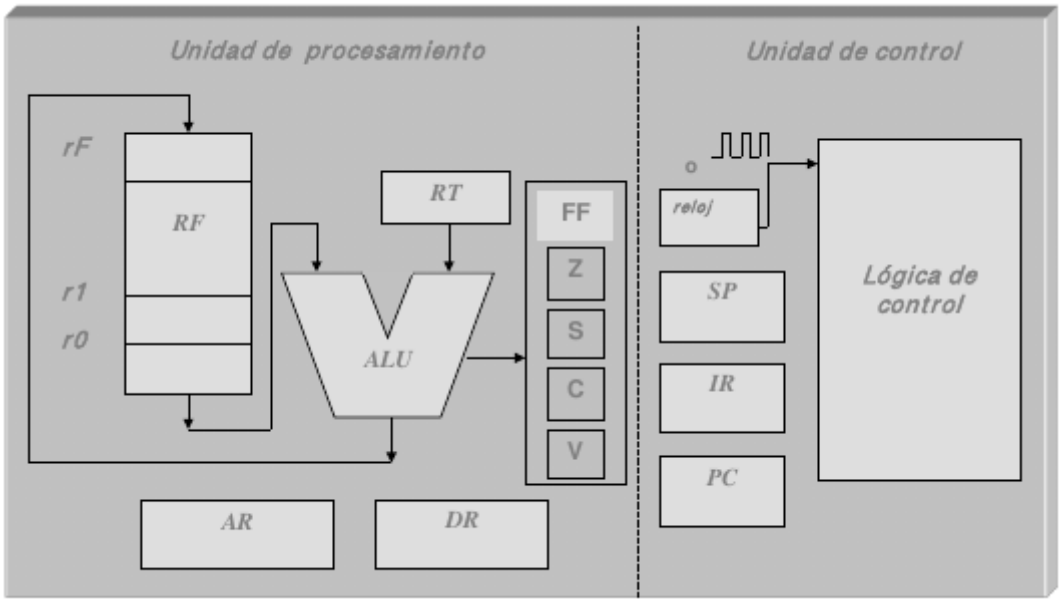
- **ISA (Instruction Set Machine)**
  - Arquitectura del repertorio: registros, instrucciones, modos de direccionamiento...
- **RISC (Reduced Instruction Set Computer)**
  - Computador de repetorio reducido
  - 0/2 o 2/3 (R/R)

    - Pocas instrucciones, pocos modos, formato de instrucción sencillo
    - UC sencilla equivale a muchos registros
- **CISC (Complex Instruction Set Computer)**
    - Computador de repertorio complejo
    - 1/2, 1/3 y el resto
    - "Más próximo a lenguajes de alto nivel"

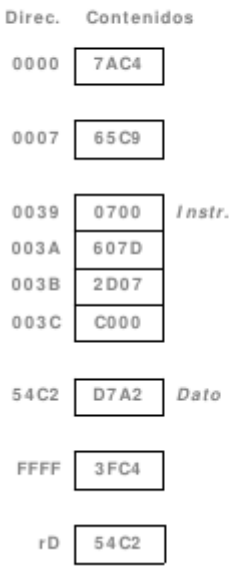
El debate RISC vs CISC ya no existe, los diseños actuales son mixtos.

2. Conceptos básicos de funcionamiento

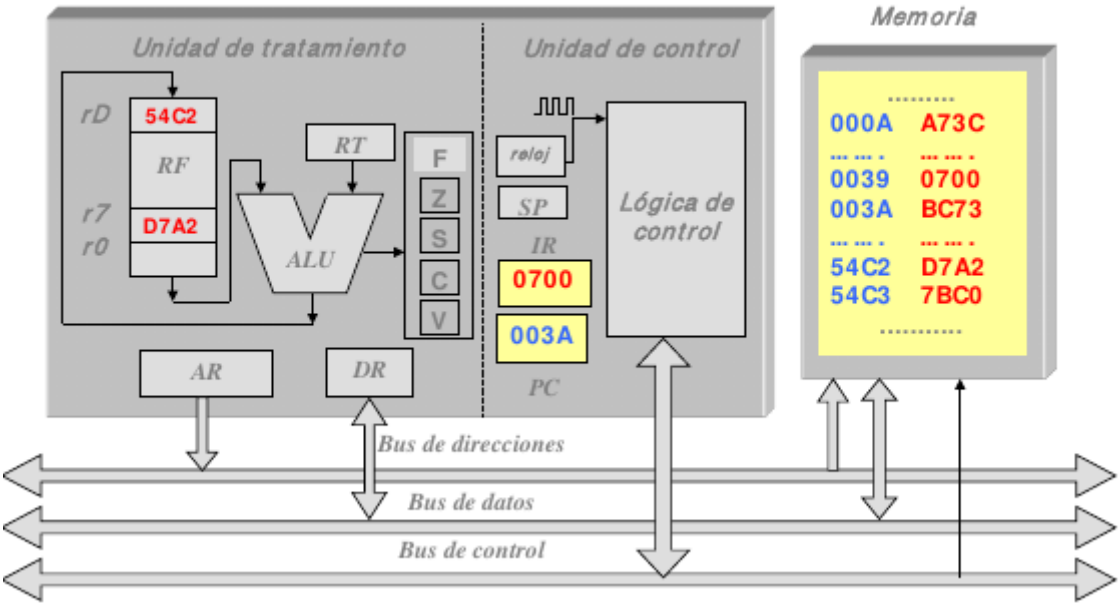
Elementos internos de un procesador



Ejecución de `mov rD, r7`



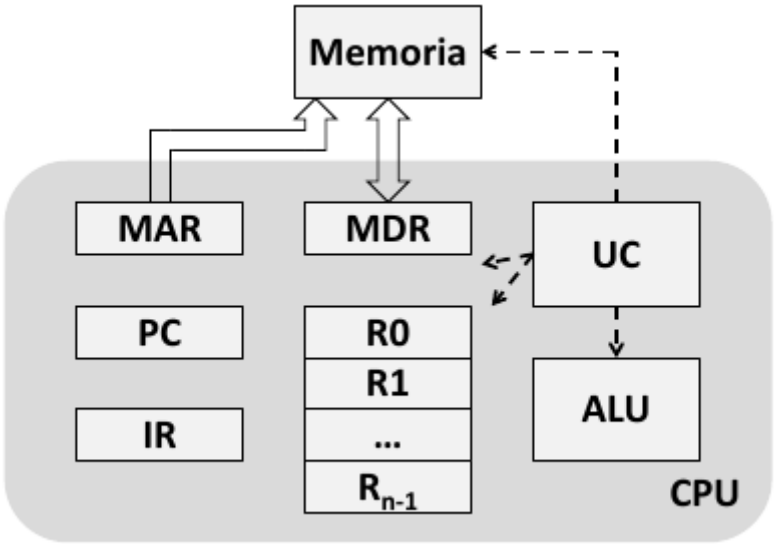
Fase	Microoperación	Contenidos de los registros				
		PC	IR	AR	DR	r7
Valores iniciales						
Captación de instrucción	AR ← PC	0039		0039		
	DR ← M(AR)	0039		0039	0700	
	IR ← DR	0039	0700	0039	0700	
	PC ← PC+1	003A	0700	0039	0700	
Ejecución de instrucción	AR ← rD	003A	0700	54C2	0700	
	DR ← M(AR)	003A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



Ciclo de ejecución de instrucciones: fases

Tenemos un programa en MP. La CPU (particularmente la UC) tiene un PC (contador del programa):

- Posición en MP de la siguiente instrucción
- **Captación (*Fetch*):** leer dicha posición. IR ← M[PC]
  - Usando MAR/MDR (memory [address/data] register), Instruction Register
  - Incrementar PC
- **Decodificación (*Decode*):** desglosar codop/operandos (registros)
  - Posible etapa **operando M (*Operand*)**: captar dato, incrementar PC
- **Ejecución (*Execute*):** llevar datos a ALU y operar
- **Almacenamiento (*Write/Store*):** salvar resultado a registros/MP



Add A, r0

- M[posición de A] + r0 y almacenar → en r0
  - Ensamblador traduce la posición de A como 100
  - El valor anterior de r0 se pierde, el de posA se conserva
  - Arquitectura R/M

Pasos básicos de la UC:

- PC apunta a la posición donde se almacena la instrucción
- **Captación**
  - MAR ← PC
  - Read
  - Aumentar PC (PC++)
  - Tacc
  - MDR ← bus
  - IR ←MDR
- **Decodificación**: se separan campos de instrucción:
  - Codop - ADD
    - mem+reg → reg
  - Dato1 - 100 (posición de A)
    - Direccionamiento directo, leer M[100]
  - Dato2 - 0
    - Direccionamiento registro, llevar R0 a ALU
- **Operando**:
  - MAR ← 100
  - Read
  - Tacc
  - MDR ← bus
  - ALU in1 ← MDR
- **Ejecución**:
  - ALU in2 ← R0
  - Add
  - Talu
- **Almacenamiento**:
  - R0 ← ALU out

Otras consideraciones

- **Arquitectura M/M:** varias captaciones operando
  - PC++ so las direcciones ocupan más posiciones de M
- **Arquitectura R/M:** cuando resultado en M (`Add R0,A`)
  - Acceso a 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 se pierde
  - Ventaja: CPU más simple, veloz, pequeña (longitud/formato de instrucción)

- **Ciclo interrumpido por IRQ → ISR**
  - Mecanismo subrutina/salvar contexto (PC y estado)
  - Salvo eso, comportamiento totalmente predeterminado por el programa
- **CPU completa** (+L1+L2...+L3) en un chip VLSI
  - La CPU nunca lee de memoria un dato aislado
    - Lee de caché
    - Si hay fallo, se trae un bloque entero

Sobre los formatos de instrucción...

- **RISC**

  Formato instrucción sencillo, tal vez sólo 2-3: transferencia, ALU, ctrl
  - Pocas instrucciones y modos
  - Muchos registros
  - 0/2 o 0/3
- **CISC**

  Varios formatos de instrucción, distintas longitudes, codops de longitud variable también
  - Muchas instrucciones y modos
  - Menos registros
  - 1/2, 1/3 (y el resto)

Modos de direccionamiento

Un número acompañando a un codop puede significar muchas cosas según el formato de instrucción, la instrucción concreta... Cada operando de la instrucción tiene su propio modo de direccionamiento:

- **Inmediato:** el número es el valor del operando

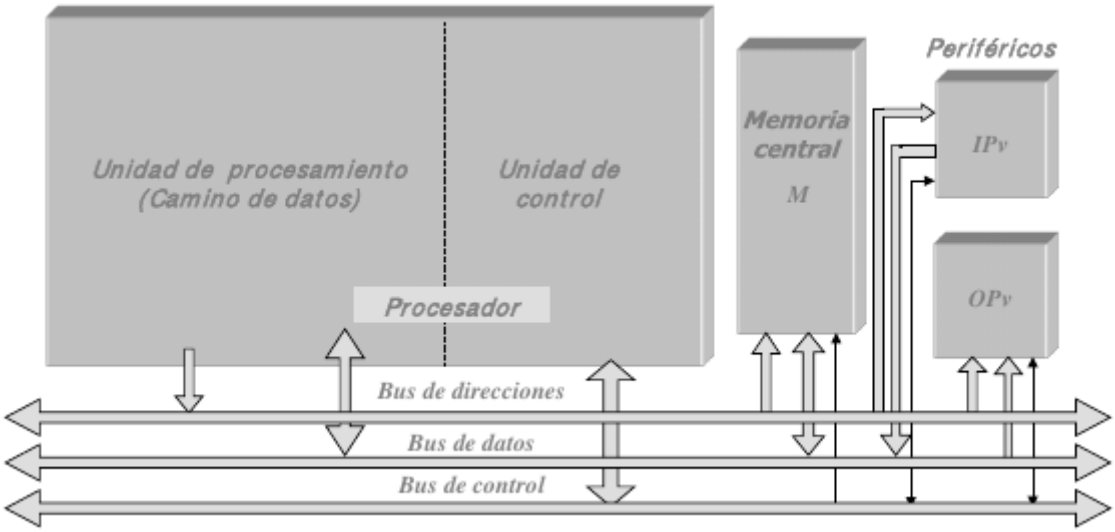
```
$0, $variable...
```

- **Registro:** el número es un índice de registro (ese registro es el operando)

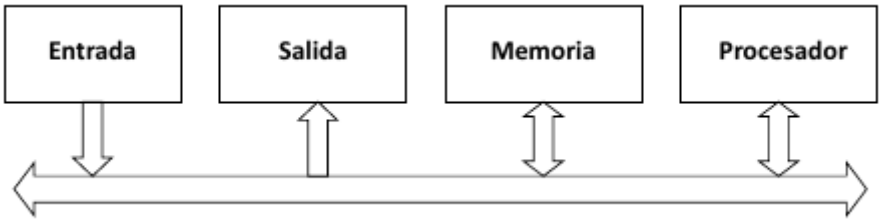
```
%eax, %ebx...
```

- **Memoria:** la instrucción lleva índices de registros y/o desplazamiento (dirección de memoria). La dirección efectiva (EA) es la suma de todos ellos. El operando es M[EA]
  - **Directo**: solo dirección (despl) -- `op=M[despl]`
  - **Indirecto**: solo registro (reg) -- `op=M[reg]`
  - **Relativo a base**: registro y desplazamiento -- `op=M[reg+despl]`
  - **Indexado**: índice (*escala) y dirección -- `op=M[despl+index*scale]`
  - **Combinado**: todo -- `op=M[despl+base+index*scale]`

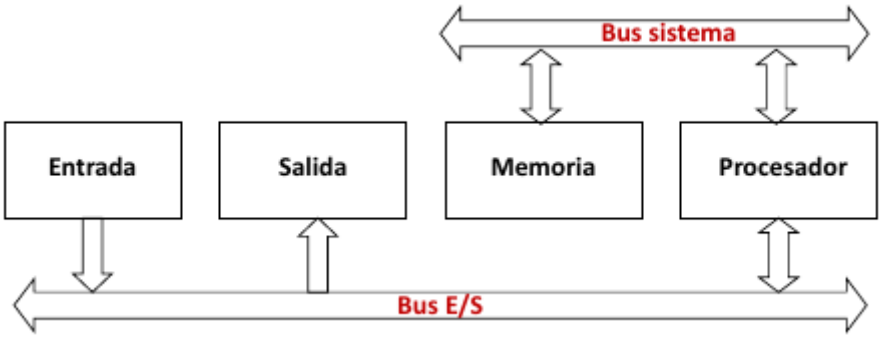
3. Estructuras de bus



- **Justificación de buses paralelos:**
  - E/S, M, CPU deben conectarse para pasar datos
  - Representación binaria/velocidad de transferencia
    - Palabras de n bits M/ALU → bus datos n bits
    - Direcciones de m bits M → bus addr m bits
    - Bus control para líneas UC (R/W, etc)
- **Bus único:**
  - CPU escribe bus dirección y control R/W
  - E/S/M comprueban si es su dirección
    - Solo en ese caso se conectan al bus de datos
    - Evita cortocircuito en bus de datos
  - Ventaja: sencillez, bajo coste, facilidad de conexión
    - Fácil añadir más dispositivos

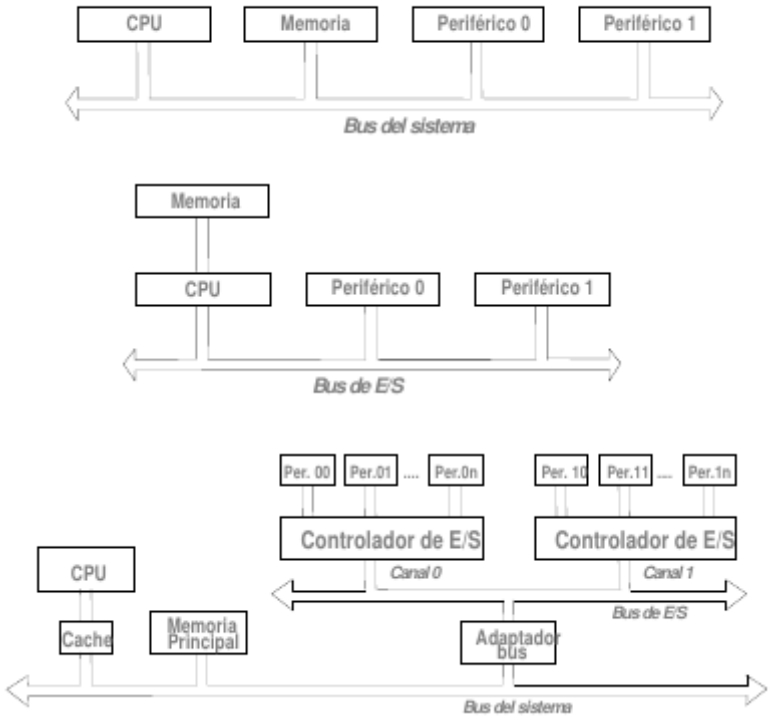


- **Buses múltiples:**
  - Típicamente bus del sistema (CPU-M) y bus E/S
    - Aunque también puede haber múltiples buses E/S (separamos dispositivos según su velocidad) o incluso doble bus del sistema (memoria datos/programa, arquitectura Harvard)
  - Ventajas: uno más rápido, ambos funcionan en paralelo
  - Inconveniente: mayor coste y complejidad



Adaptación de velocidades

- **Velocidad de componentes:** CPU > Memoria >> E/S
- **Estados de espera:** alargamos el ciclo de bus si no se activa la señal RDY (si no está listo). Esto permite conectar periféricos lentos a bus único
- **Buffers/IRQ:**
  - Dispositivo lento almacena los datos en buffer rápido
    - Evita retrasar la 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
  - Read CPU encarga lectura, dispositivo hace IRQ cuando esté listo



Decodificación

Evita el cortocircuito en el bus de datos. Supongamos, por ejemplo, que la CPU es el único dispositivo activo del bus:

- Es decir, puede escribir bus de direcciones y de control
  - Cuando lo hace, se convierte en master del bus
  - Luego veremos multiprocesadores, controladores DMA etc.
  - Varios dispositivos activos requieren arbitraje para decidir el master

- El resto de dispositivos son pasivos: sólo escuchar el bus de direcciones, no pueden escribir, solo leer
    - Cuando la CPU les habla, se convierten en seguidores
    - Es decir, se conectan al bus de datos y obedecen la orden R/W

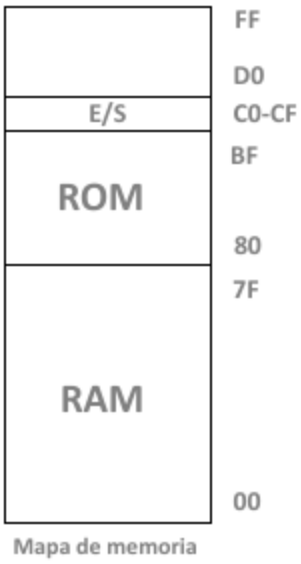
El número de bits del bus de direcciones determina el espacio de memoria.

Un mapa de memoria es un dibujo de dónde está cada dispositivo en espacio de 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



Mapa de memoria

Decodificación completa

Usamos todos los bits de direcciones. El bit más significativo (MSB) decodifica el dispositivo/módulo (CS). El menos significativo (LSB) direcciona dentro del dispositivo (dirección)

Decodificación parcial

m bits de direcciones se dejan sin usar. El dispositivo aparece repetido 2^m veces en memoria.

Software de sistema

Cómo conseguir crear programa → MP → ejecutar

- Software del sistema implicado:
    - Shell - intérprete de comandos: recibe órdenes del user
        - EXEC: llamada para cargar y ejecutar la aplicación
    - Editor: permite crear y archivar código fuente
    - Compilador/Enlazador: código objeto/ejecutable
    - Sistema de ficheros (crear, copiar, abrir, leer)
    - Sistema E/S

Como se consigue encender → arrancar SO

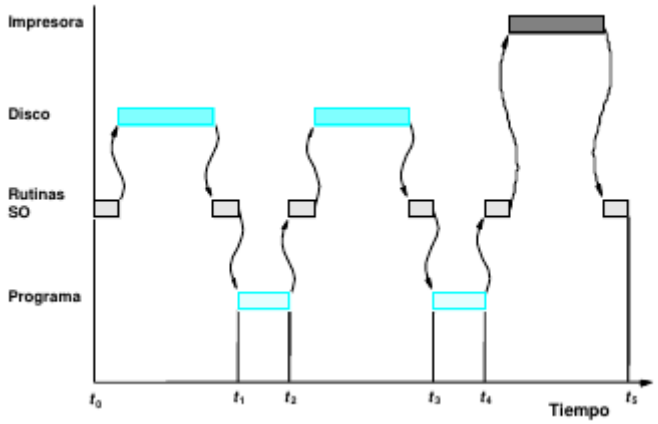
- Soporte hardware: dirección de bootstrap
- ROM en espacio memoria apuntado
- Bootloader primario, carga arranque HD/FD/CD...
- Bootloader secundario (menú escoger SO, grub...)

Llamadas al sistema

- Usuario teclea nombre aplicación → EXEC
  - El shell invoca EXEC, proporcionando nombre de fichero
- EXEC carga aplicación HD → M, pasa control
  - El propio SO proporciona zona de memoria para cargar la aplicación
  - EXEC retorna a aplicación y ella retorna a shell
- Aplicación invoca OPEN/READ/CLOSE
  - Proporciona zona de memoria donde leer contenido
- Aplicación calcula resultado, invoca PRINT/EXIT
  - Proporciona datos a imprimir o código de retorno

Gestión de recursos del SO

- ej. solapar E/S final con carga de la siguiente tarea
- ej. conmutar proceso en cuanto haga E/S para aprovechar el tiempo de espera



4. Rendimiento

Medida definitiva: tiempo de ejecución del programa. Pero cuál programa? Para eso sirven los benchmarks.

El tiempo de ejecución puede depender del diseño de CPU, del repertorio de instrucciones, del compilador (benchmarks en lenguajes de alto nivel), versión del SO, librerías...

Benchmarks CPU ejercitan sólo la CPU (tiempo de procesamiento), influido por prestaciones CPU, M, cachés, buses... La caché conserva lo accedido recientemente y accede más rápidamente: ventaja en ejecución en bucles etc.

Reloj del procesador

UC emplea varios ciclos de reloj en ejecutar una instucción. Los pasos básicos ocupan un ciclo (comutar señales de control).

```
Frecuencia de reloj = 1/P
```

Ej. *500 MHz = 1 / 2ns*

1.25GHz = 1 / 0.8ns

Ecuación básica de rendimiento

T = tiempo para ejecutar el programa benchmark

N = instrucciones (recuento dinámico de bucles/subrutinas). Es decir, N no es necesariamente igual al número de instrucciones en el programa dado.

S = ciclos/instrucciones ("pasos básicos" de media)

```
T = N*S / R

tiempo = [instrucciones * ciclos/instrucción] / frecuencia
```

Lo ideal: N y S bajos, R alta:

- N (instrucciones) dependen del compilador/repertorio de instrucciones
- S (ciclos/instr) depende de la implementación de la CPU
- R depende de la tecnología y diseño CPU

Alterar uno modifica a los otros: aumentar R puede ser a costa de aumentar S, lo importante al final es obtener el menor tiempo posible.

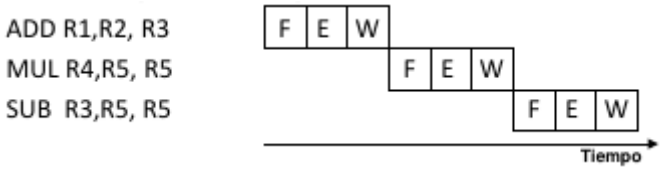
Segmentación de cauce, S ≈ 1

N*S es suponiendo una ejecución individual de las instrucciones, pero las distintas etapas hacen tareas distintas: UC puede tener circuitería separada para cada etapa:

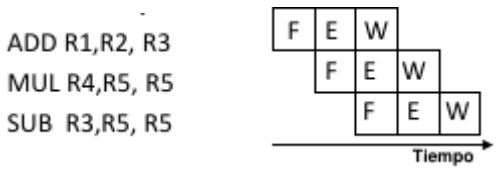
- FETCH
- EXEC
- WRITE

Una vez lleno el cauce, el valor efectivo de S=1 ciclo/instrucción. Sin embargo, se pueden dar dependencias de datos, competición por recursos, saltos... pero aún así se simplifica a S>=1, S ≈ 1

Sin segmentación



Con segmentación

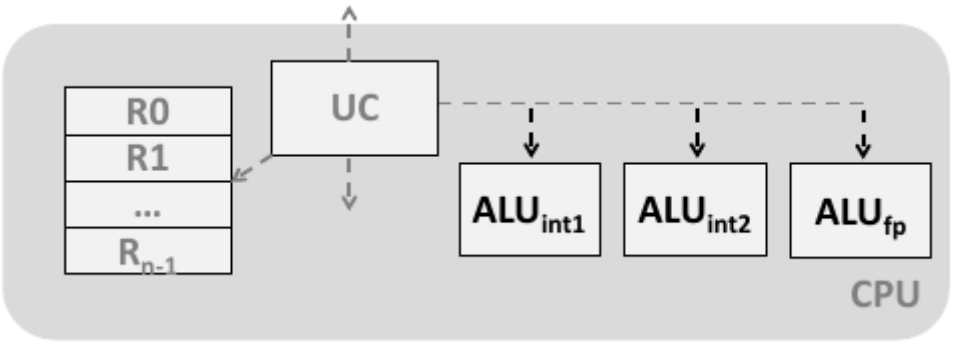


Funcionamiento superescalar, S<1

Conseguir paralelismo a base de reduplicar unidades funcionales (ej. 2 ALU para enteros, 2 para coma flotante, emiten hasta 4 operaciones simultáneas si usadas en el orden apropiado).

Combinado con segmentación de cauce podemos obtener S<1, completamos más de una instrucción por ciclo.

Es común en CPUs actuales, pero presenta dificultades a la hora de emitir en el orden correcto y, en consecuencia, obtener el resultado correcto.



Otras formas de reducir T

- **Velocidad del reloj:** mejora en la tecnología equivale a mayor frecuencia de reloj R
  - Si no cambiamos nada más, y duplicamos la frecuencia, se reducirá el tiempo a la mitad? `R*2 = T/2`? NO: la memoria también R*2, o mejorar caché L1, L2
  - Otra alternativa es aumentar S, "supersegmentación", reducir la tarea por ciclo de reloj. Es dificil predecir ganancia, puede llegar hasta a empeorar
- **Repertorio RISC/CISC:** aunque actualmente son técnicas híbridas RISC/CISC
  - RISC: instrucciones simples equivalen a mayor frecuencia R, pero menor S y mayor N
  - CISC: instrucciones complejas para menor N, pero más S
    - Si intentamos corregir la S con segmentación acabamos por competir por recursos
- **Compilador:** la optimización debe ser correcta (mismo resultado)
  - Optimizador espacial (menor N) o temporal (menor N*S). Suelen ser contrarios
    - **Espacial**: requiere conocimientos de arquitectura: repertorio, modos direccionamiento, alternativas de traducción...
    - **Temporal**: requiere conocimiento detallado de la organización: reordenación de instrucciones para ahorrar ciclos, evitar competencia por los recursos...

5. Perspectiva histórica

1ª Generación 45-55: tubos de vacío

- Von Neumann: concepto del programa almacenado
- Tubos de vacío 100-1000x, T conmut. = O (ms)
- Memoria: líneas retardo mercurio, núcleos magnéticos de ferrita
- E/S: lectura de tarjetas perforadas, cintas magnéticas
- Software: lenguaje máquina/ensamblador

2ª Generación 55-65: transistores

- Invento Bell AT&T 1947, T conmut = O (microsegundos)
- E/S: procesadores E/S (cintas) en paralelo con CPU
- Software: compilador FORTRAN

3ª Generación 65-75: circuito integrado

- Velocidad de CPU y memoria aumenta, T conmut = O (nanosegundos)
- Arquitectura: microprogramación, segmentación de cauce, memoria caché
- Software: SO multiusuario, memoria virtual

4ª Generación 75-... : VLSI

- Microprocesador: procesador completo en un 1 chip
    - MP completa en uno o pocos chips
- Arquitectura: mejoras segmentación de cauce, caché, memoria virtual
- Hardware: portátiles, PCs, WS, redes
- Mainframes siguen solo en grandes empresas
    - i8008, i8080, i8086
    - i80286, i80386, i80486

Actualidad

- Computadores sobremesa potentes y asequibles
- Internet
- Paralelismo masivo
    - Pentium PII PIII P4
    - Pentium 4F, Core 2 duo, Core i7