



MÁSTER UNIVERSITARIO EN SISTEMAS ESPACIALES
POR LA UNIVERSIDAD POLITÉCNICA DE MADRID



POLITÉCNICA

Estructura de computadores

Objetivos

- Comprender la estructura y el diseño del computador de a bordo (OBC) del UPMSat-2.
- Visión general de la estructura del computador:
 - Arquitectura Von Neumann
 - Componentes
 - Ejecución y juego de instrucciones
- Funcionamiento del sistema de entrada/salida y dispositivos periféricos de aviónica.
- Otros mecanismos habituales: memoria cache

Índice

- Objetivos
- Bibliografía recomendada
- Introducción
- Esquema básico del computador Von Neumann.
 - La Memoria Principal
 - La Unidad Central de Proceso (CPU)
 - Unidad Aritmético-Lógica (ALU)
 - Registros
 - Unidad de control
 - Fases de ejecución de una instrucción
 - Modos de direccionamiento
 - Juego de instrucciones
 - Ejercicio

Índice

- Memoria cache
 - Motivación
 - Funcionamiento y estructura
 - Análisis
 - Ejercicio
- Sistema de entrada-salida (I/O)
 - Módulos y operaciones de I/O
- Técnicas de operaciones de entrada-salida
- Dispositivos periféricos
 - Temporizadores programables
 - Entrada-salida digital: GPIO
 - Entrada-salida analógica: ADC, DAC y PWM
 - Línea serie: UART
 - Red de comunicación: Ethernet

Índice

- Diseño e implementación:
 - FPGAs y bibliotecas de IP Cores.
- Ejemplo: computador de a bordo del UPMSat-2

Bibliografía

- Stallings, W. ``*Computer Organization and Architecture*'', Prentice Hall, 2013, 9th Edition.
- Patterson, D.A. and Hennessy,J.L ``*Computer Organization and Design: The Hardware/Software Interface*'', Elsevier Inc. 2012, 4th Edition.
- Aeroflex Gaisler ``*GRLIB IP Core User's Manual*'', available at
<http://www.gaisler.com/index.php/downloads/leongrlib>

Aclaración

- Parte de estas transparencias están basadas en las desarrolladas por el grupo de arquitectura de computadores del DATSI/UPM.
- El autor forma parte de este grupo y colaboró en la elaboración de las originales.

Introducción

- **Función básica**

Ejecución de instrucciones elementales, en las que están especificados:

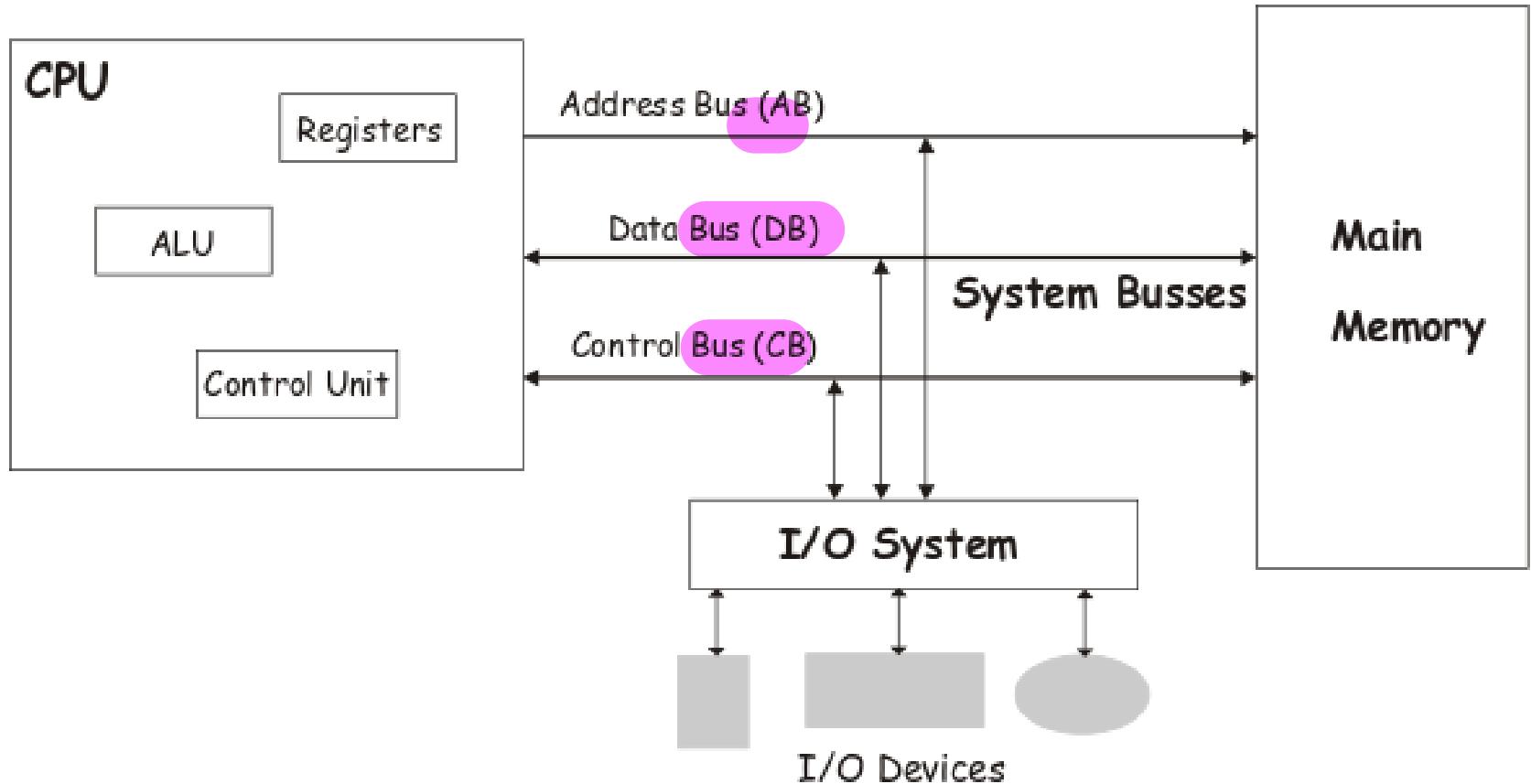
- Operación a realizar
- Datos o su localización
- Localización del resultado

Instrucciones máquina

- **Arquitectura Von Neumann**

- *Datos e instrucciones almacenados en memoria única de lectura/escritura*
- *Contenido de la memoria accesible por direcciones*
- Ejecución *implícitamente* secuencial
- Registro contador de programa (PC).

Esquema básico del computador Von Neumann.



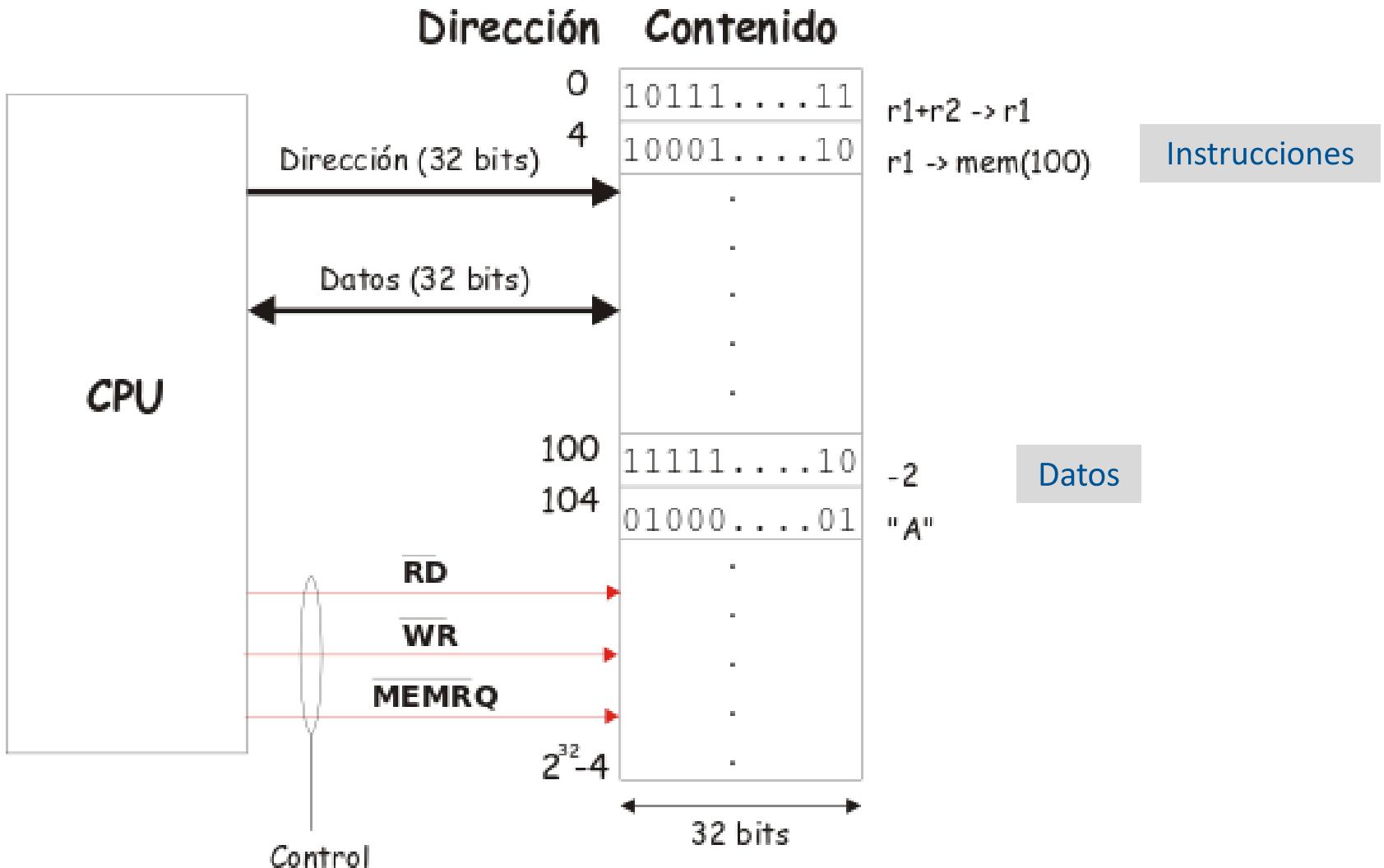
Parámetros característicos.

- Ancho de palabra o palabra: número de líneas del bus de datos y número de bits de los registros de la CPU.
 - 8, 16, 32, 64 bits
- En el tamaño de la memoria principal y de la memoria que se direcciona mediante señales binarias no se usa el SI:
 - Kilo (2^{10}), Mega (2^{20}), Giga (2^{30}), Tera (2^{40}) bytes
 - A veces Ki, Mi, Gi, ...
- Frecuencia de reloj: frecuencia a la que realiza las operaciones elementales la CPU.
 - Mega hercios (MHz), Giga hercios (GHz)
- Capacidad de cómputo :
 - Velocidad de ejecución de instrucciones: MIPS, MFLOPS
 - Velocidad de ejecución de Benchmarks: specint, specfp
- Ancho de banda o velocidad de transferencia/transmisión:
 - KB/s (KBps), MB/s (MBps), Kb/s (Kbps), Mb/s (Mbps)

Terminología.

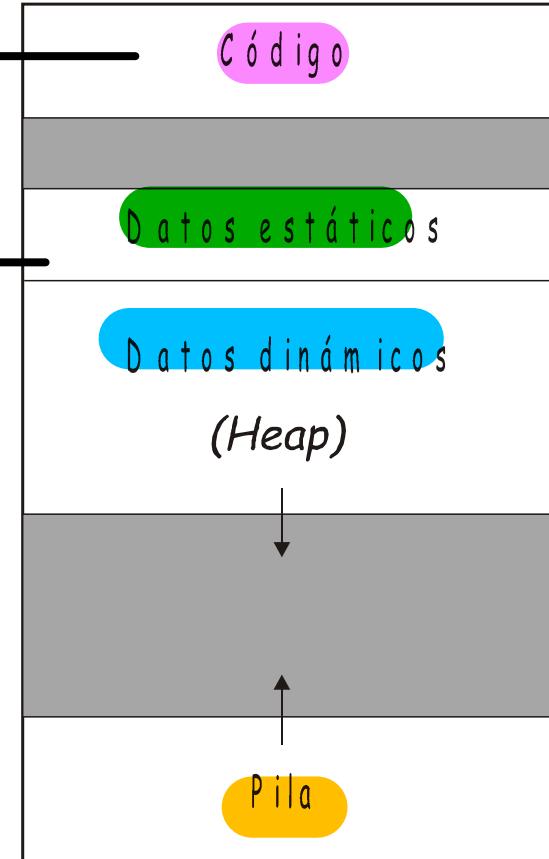
- Procesador: CPU
 - Microprocesador: procesador integrado en un único chip.
 - Intel 4004 (1971) con ancho de palabra de 4 bits y 2000-3000 transistores.
 - Multiprocesador: computador con más de un procesador que comparten la memoria principal y ejecutan instrucciones en paralelo.
 - Multicore: multiprocesador integrado en único chip.
 - Manycore: multicore con muchos procesadores (> 50 ?)
 - SoC (System on Chip)
 - Computador completo (CPU(s), sistema de entrada-salida e incluso dispositivos periféricos y memorias) en un único chip.
 - Término de uso habitual en hardware configurable que permite construir computadores a medida.
 - El computador de a bordo del UPMSat-2 está basado en un SoC.

La Memoria Principal: ejemplo con datos y direcciones de 32 bits

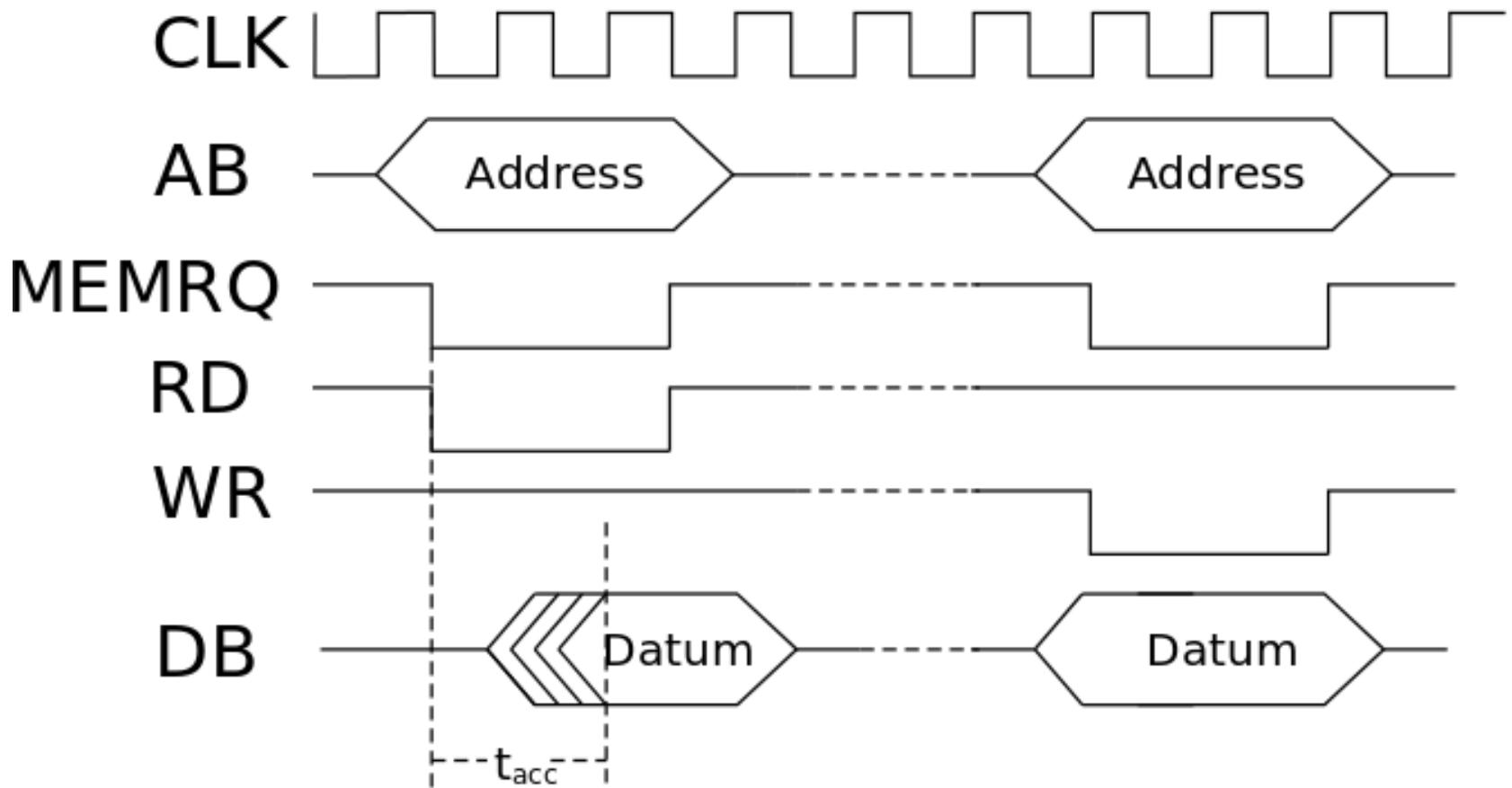


Organización de la memoria principal

- La sección de código o texto (*text*) contiene las instrucciones del programa cargado en memoria.
- La sección de datos estáticos (*data y bss*) contiene las variables globales del programa.
- La sección de pila (*stack*) contiene las variables locales de los subprogramas.
- La sección de datos dinámicos (*heap*) se usa para estructuras que se crean y destruyen explícitamente en el programa.

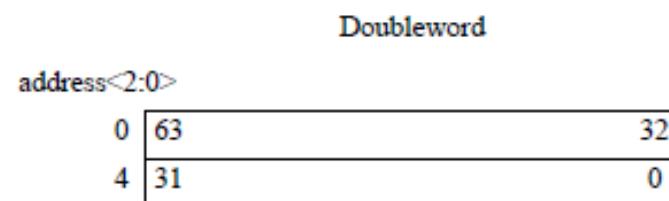
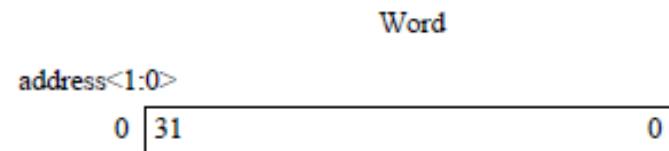
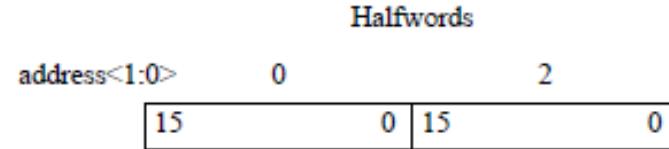
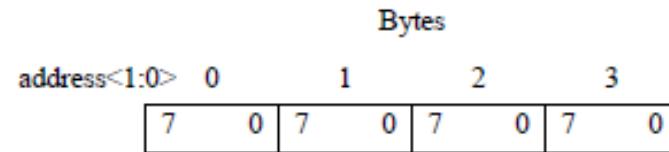


Memoria principal: ciclos de bus



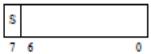
Memoria principal: tamaño de datos

- Además de transferencias de tamaño de palabra. Se pueden realizar transferencias de byte, media palabra y doble palabra.
- El tamaño de los datos se especifica en líneas auxiliares del Control Bus (CB).

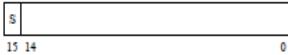


Tipos de datos

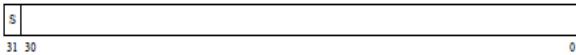
Signed Integer Byte



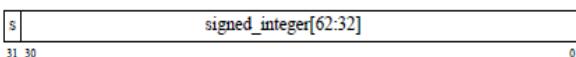
Signed Integer Halfword



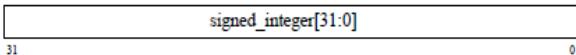
Signed Integer Word



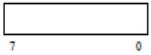
Signed Integer Double



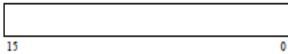
SD-1



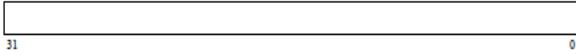
Unsigned Integer Byte



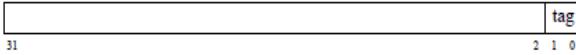
Unsigned Integer Halfword



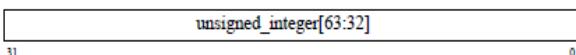
Unsigned Integer Word



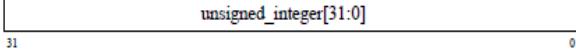
Tagged Word



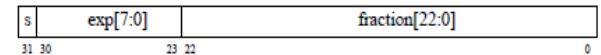
Unsigned Integer Double



UD-1

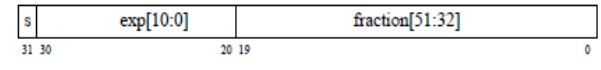


Floating-point Single

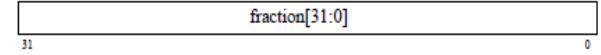


Floating-point Double

FD-0

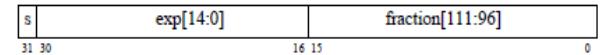


FD-1

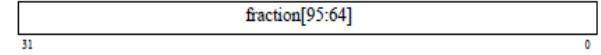


Floating-point Quad

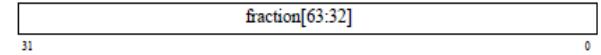
FQ-0



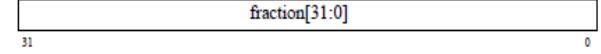
FQ-1



FQ-2



FQ-3



Tipos de memoria

Memoria volátil.

- RAM (Random Access Memory). Memoria que se puede leer y escribir con el mismo tiempo de acceso.
 - Se utiliza como memoria principal para contener instrucciones y datos.
 - Variantes: SRAM y DRAM.

Memoria no volátil

- ROM (Read Only Memory). Memoria que sólo se puede leer y se escribe durante su construcción.
 - Se utiliza para contener “Firmware”.
 - Variantes: PROM y EPROM.
- EEPROM (Electrically Erasable Programmable Read-Only Memory). Memoria que se puede leer y escribir pero con distintos tiempos de acceso.
 - Se utiliza como unidad de almacenamiento en computadores empotrados.
 - Variante: Flash.



Memoria del OBC del UPMSat-2

- 4 Mbytes de memoria RAM estática (SRAM) de 32 bits de palabra.
 - Tiempo de acceso de 12 ns.
 - Contiene las instrucciones del programa y sus datos.
- 2 Mbytes de memoria EEPROM de 32 bits de ancho de palabra.
 - Tiempo de acceso de 250 ns.
 - Tras una escritura hay que esperar 15 ms antes de leer o escribir en la memoria EEPROM.
 - Contiene:
 - Las instrucciones y datos inicializados del programa que se copian a RAM durante el inicio.
 - Los datos de configuración de los distintos subsistemas software: ADCS, Housekeeping, TM/TC, ...
 - La telemetría que se mandará en el próximo periodo de cobertura.
 - Los telecomandos diferidos que aún no han sido procesados.

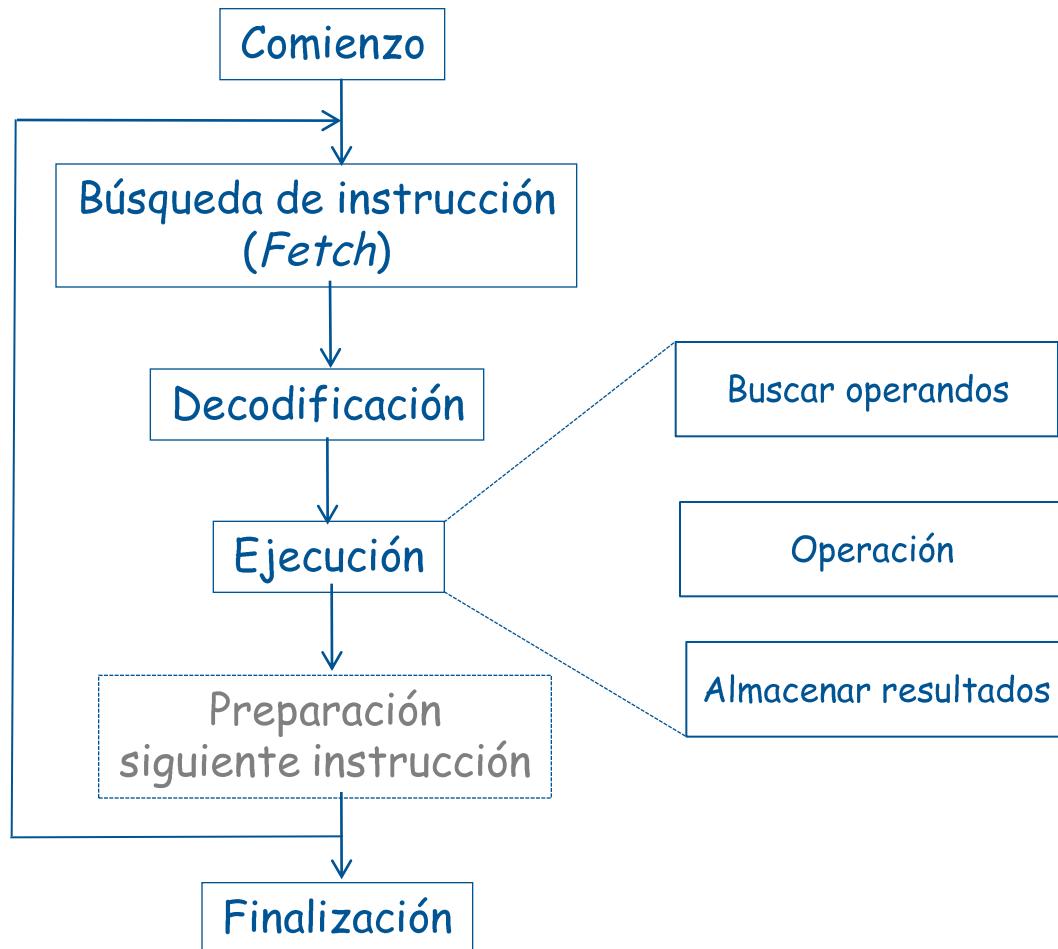
Lenguaje máquina

- Cada programa está compuesto por datos e instrucciones almacenados en memoria
- Una Instrucción máquina: Es la función básica elemental que puede ejecutar un computador.
- Son cadenas de 1 y 0 con un significado particular para cada procesador
- Propiedades:
 - Realizan una función única y sencilla.
 - Tienen un número fijo de operandos
 - Autocontenido: Contienen todo lo necesario para su ejecución (operación, operandos, dir. resultado y dir. sig instrucción)

Fases de ejecución de una instrucción



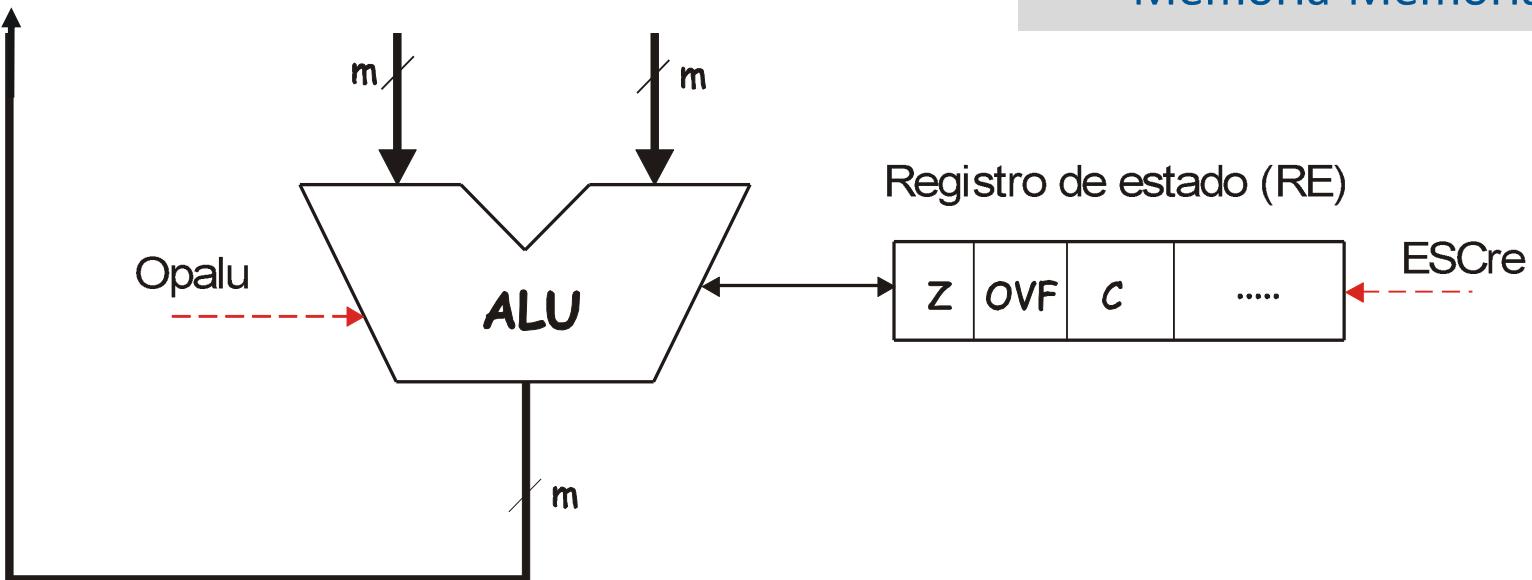
Fases de ejecución de una instrucción



Unidad Central de Proceso (CPU)

- **Unidad de Control**
 - Extrae de Mp la instrucción a ejecutar (fetch)
 - La analiza (descodifica)
 - Da las órdenes al resto de componentes
- **Unidad Aritmético-Lógica (ALU)**
 - Realiza la operación indicada por la UC sobre los datos de entrada.
 - Aritméticas: ADD, NEG, MUL, SHIFT, ...
 - Lógicas: OR, XOR, NOT, SHIFT, ...
- **Registros**
 - Memoria a corto plazo

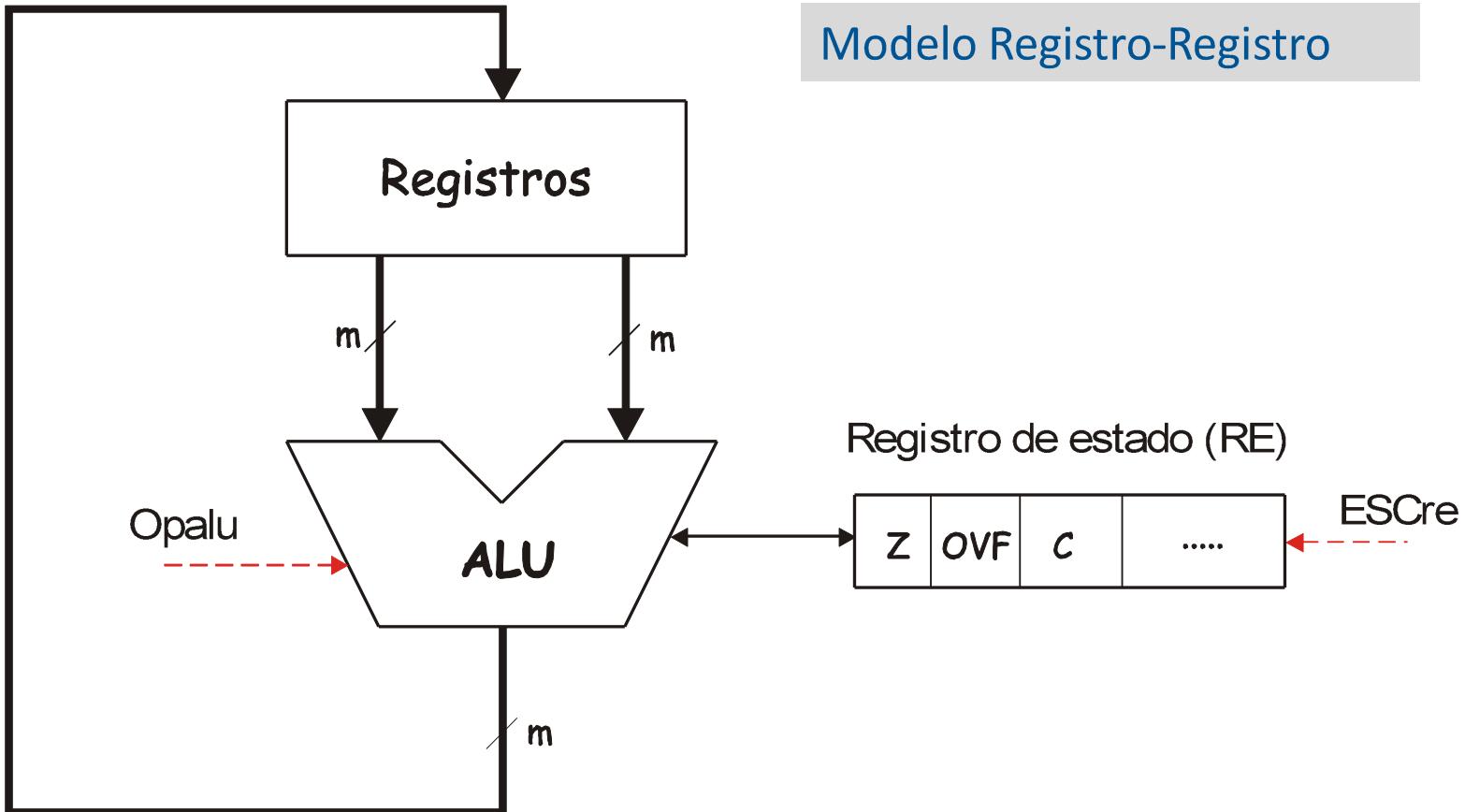
CPU: Unidad Aritmético-Lógica (ALU)



Modelos de ejecución:

- Registro-Registro
- Registro-Memoria
- Memoria-Memoria

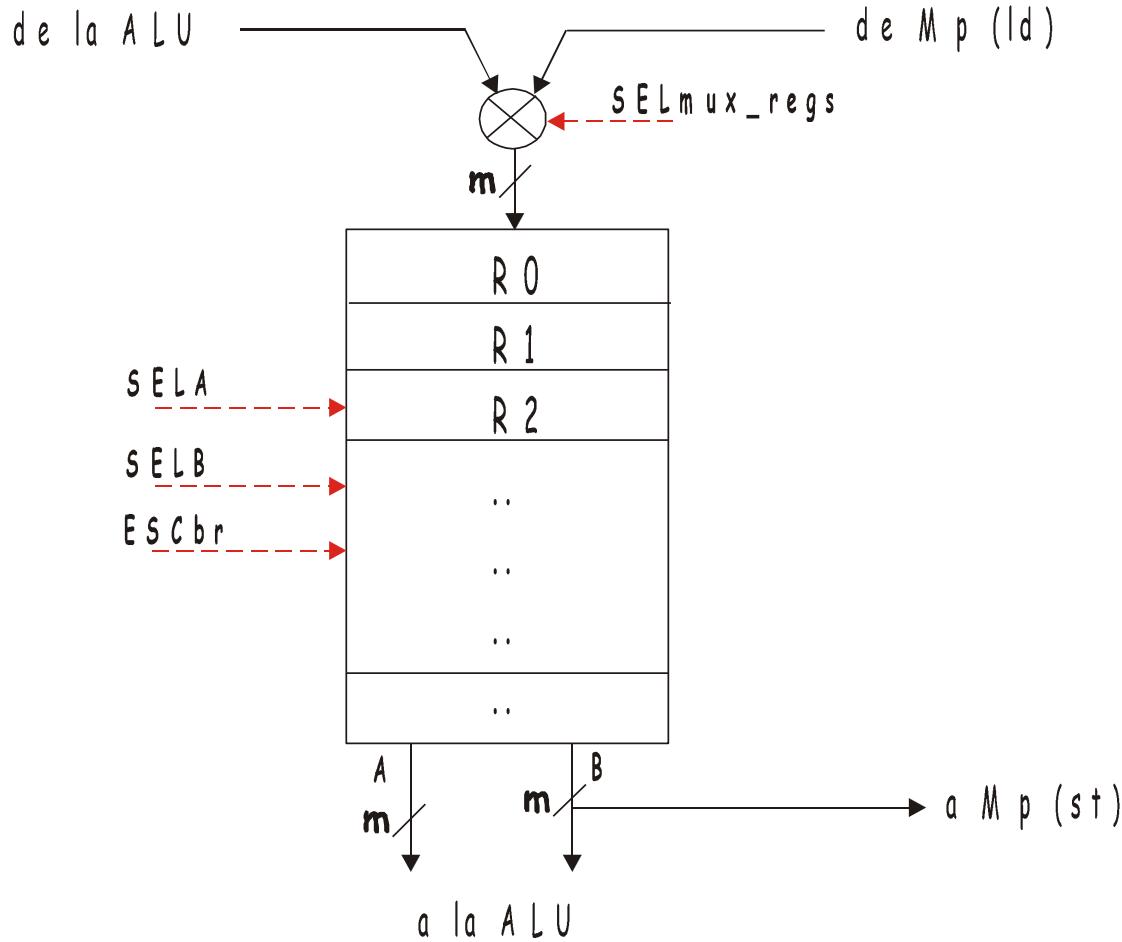
CPU: Unidad Aritmético-Lógica (ALU)



CPU: Registros

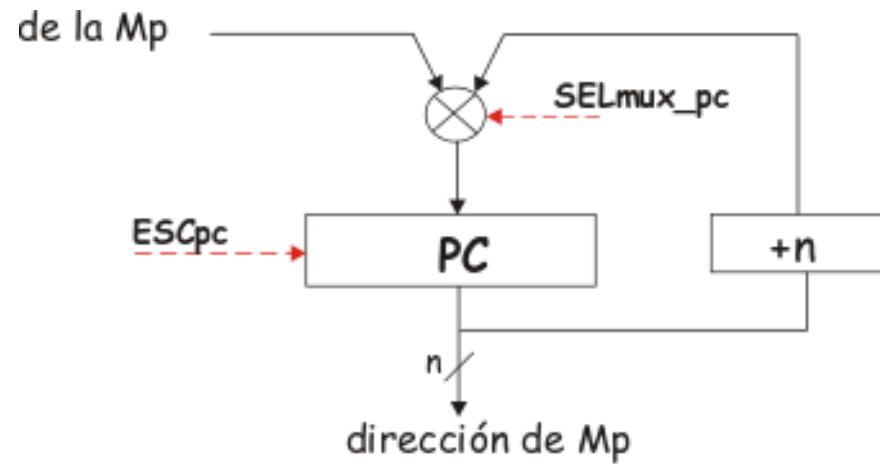
- De propósito general
 - Se usan para contener datos y direcciones de memoria
- De propósito específico
 - Se usan para contener el estado del programa
- Transparentes o no accesibles para las instrucciones
 - De uso interno para las operaciones del procesador

CPU: Registros de propósito general (BR)

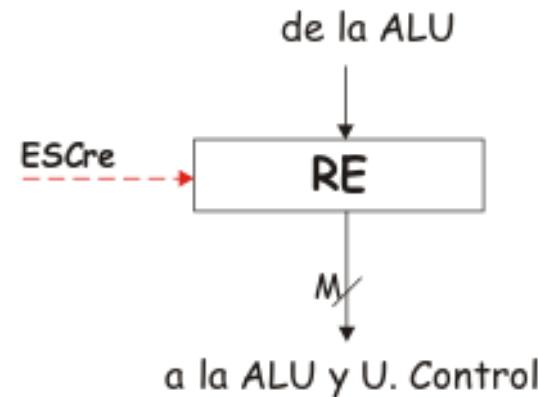


CPU: Registros de propósito específico

Contador de programa

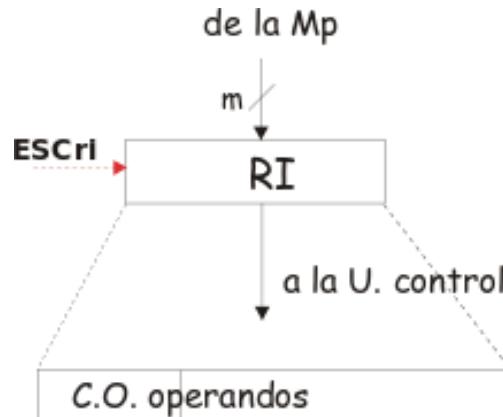


Registro de estado

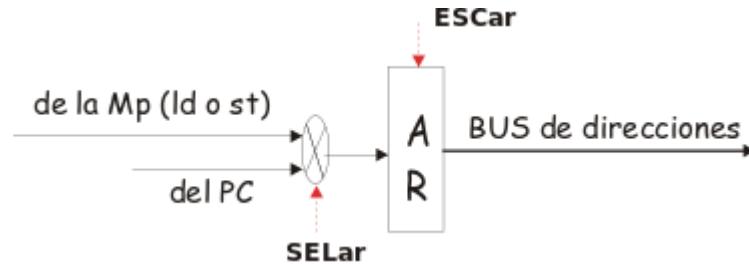


CPU: Registros transparentes

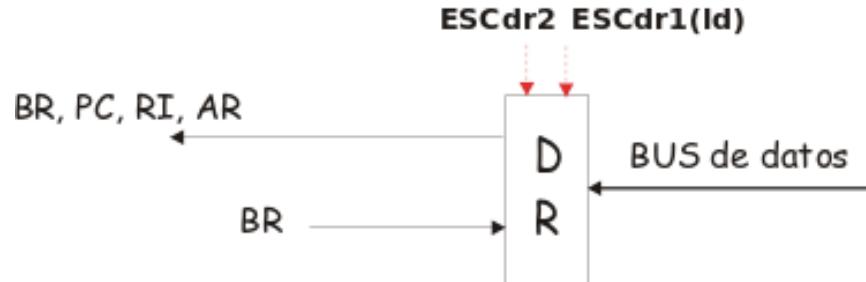
Registro de instrucción



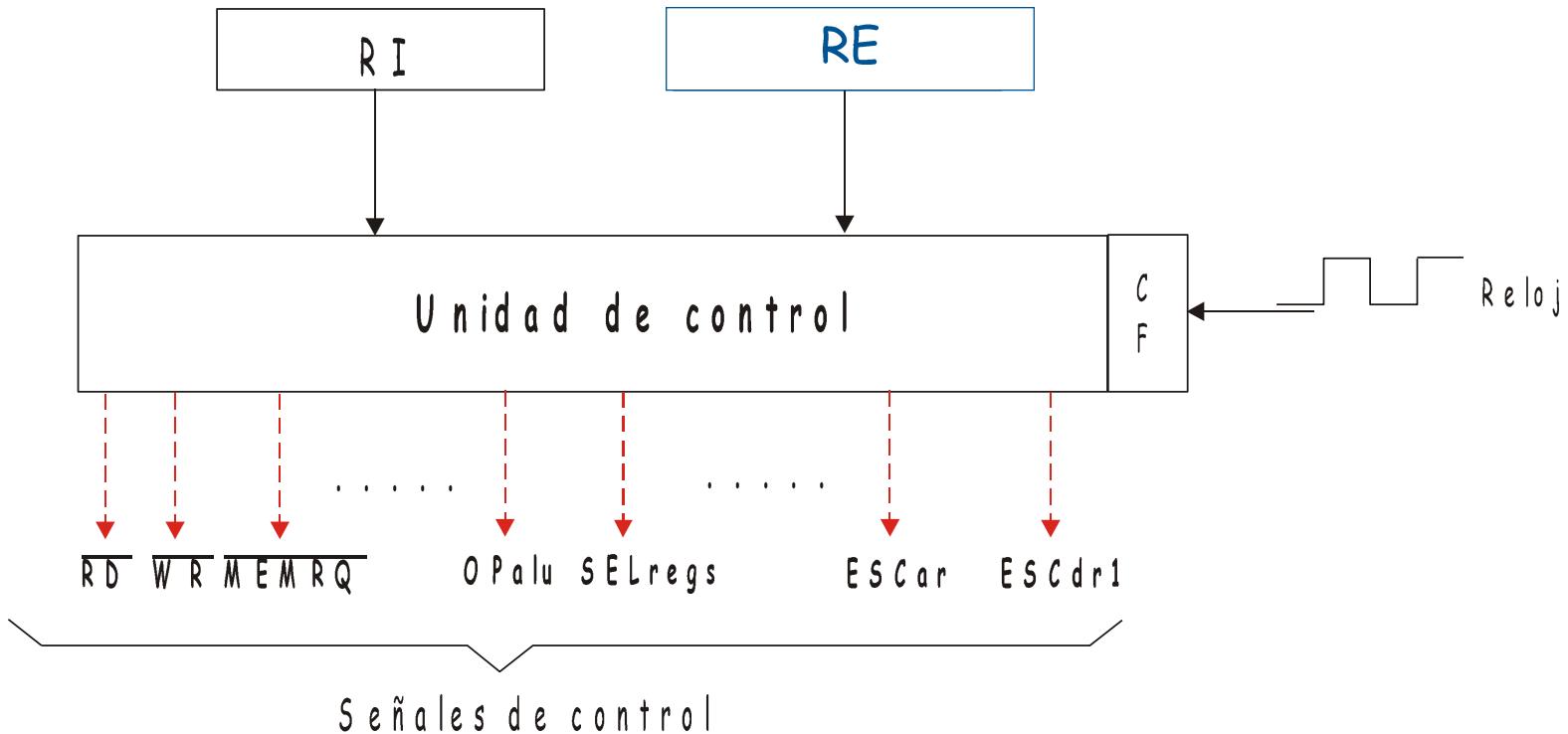
Registro de direcciones



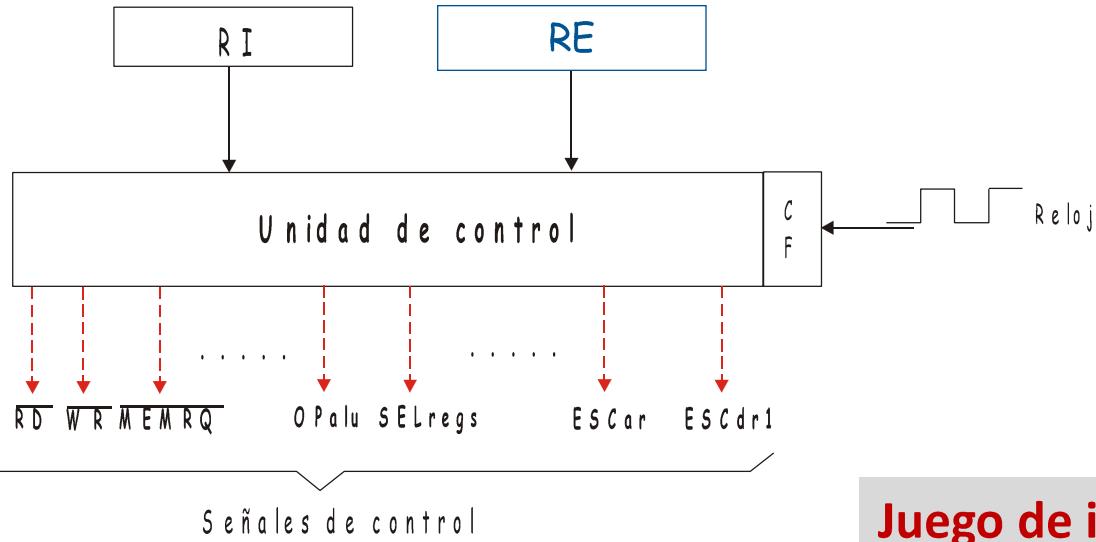
Registro de datos



CPU: Unidad de Control



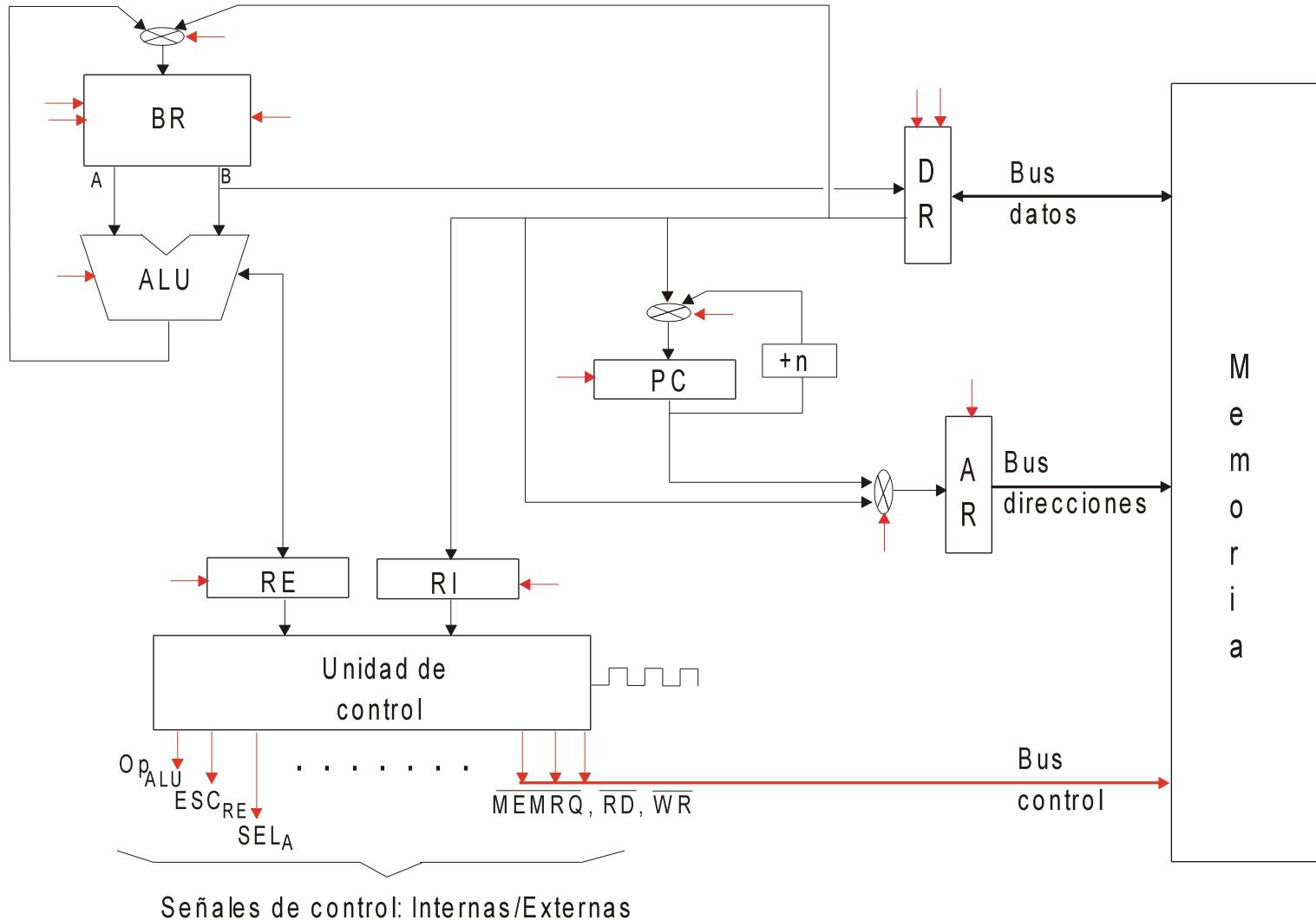
CPU: Unidad de Control



Juego de instrucciones:

- Transferencia (*ld, st, mov, in, out*)
- Procesamiento (*add, and, shift*)
- Salto (*jump, jumpz, call, ret*)

Diagrama simplificado



Coprocesador de coma flotante (FPU)

- Es una unidad especializada en operaciones aritméticas con datos en coma flotante.
 - Suma, resta, multiplicación y división
 - Funciones trigonométricas y transcendentales.
- También contiene registros de propósito general para contener datos en coma flotante.
- No realiza *fetch* de instrucciones sino que está subordinada a la CPU.
- Normalmente son conformes al IEEE Standard for Floating-Point Arithmetic (IEEE 754).
 - Realizan las operaciones en simple, doble y precisión extendida.

Ejemplo de instrucciones a ejecutar

```
program Example is
  ...
  One : constant Integer := 1;
  I : Integer := 2;
  -- The compiler stores I in
  -- memory address 1000
  -- and loads One in r2
  ...
begin
  loop
    I := I - One;
    exit when I = 0;
  end loop;
  ...
end Example;
```

Dirección	Lenguaje ensamblador
0	ld r2, #1
1	ld r1, /1000
2	1000
3	sub r1 r1 r2
4	saltar si Z=0
5	3
6	...
1000	2
	00000...010

Juego de instrucciones

- Conjunto de instrucciones que ejecuta el procesador.
- La codificación de las instrucciones deben encajar en pocos formatos.
 - La tendencia actual es que ocupen una palabra.
- Formato de instrucción: Representación de una instrucción y especificación de cada campo:
 - Código de operación
 - Operандos (direcciones)



Modos de direccionamiento

- Forma en la que se accede a una instrucción o dato.
- **OBJETO**: Instrucción o dato al que se desea acceder
- **DIRECCIÓN**: Lugar en el que reside el objeto. Puede estar almacenado en:
 - La instrucción.
 - Un registro
 - La memoria

Direccionamiento **inmediato**

- El **objeto** está contenido en la propia instrucción.

Instrucción



• **ADD .R1,#4**

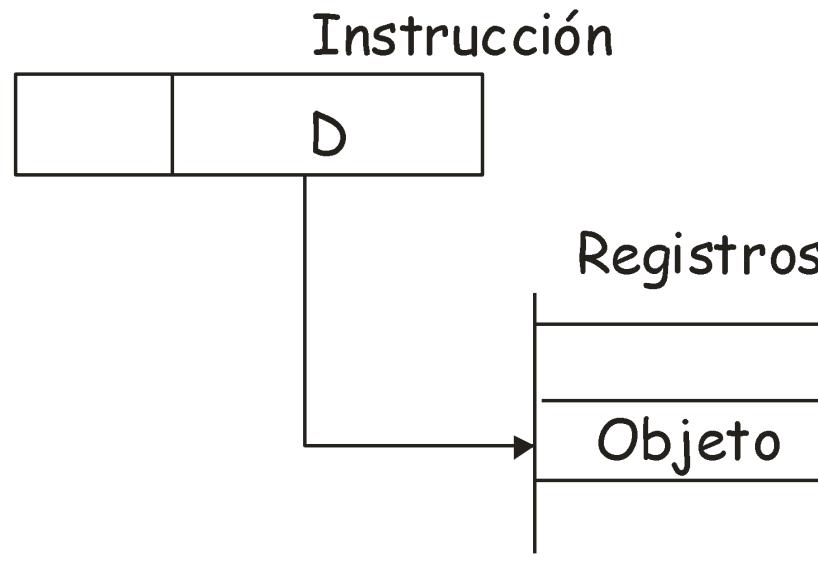
R1 <= R1 + 4

Direccionamiento directo

- El objeto no está contenido en la propia instrucción. La instrucción contiene el lugar (dirección) donde está almacenado el objeto.
- **ABSOLUTO:** Si la instrucción contiene la dirección completa del objeto
- **RELATIVO:** Si la instrucción contiene la dirección del objeto de forma parcial. Todos los direccionamientos relativos lo son a memoria.

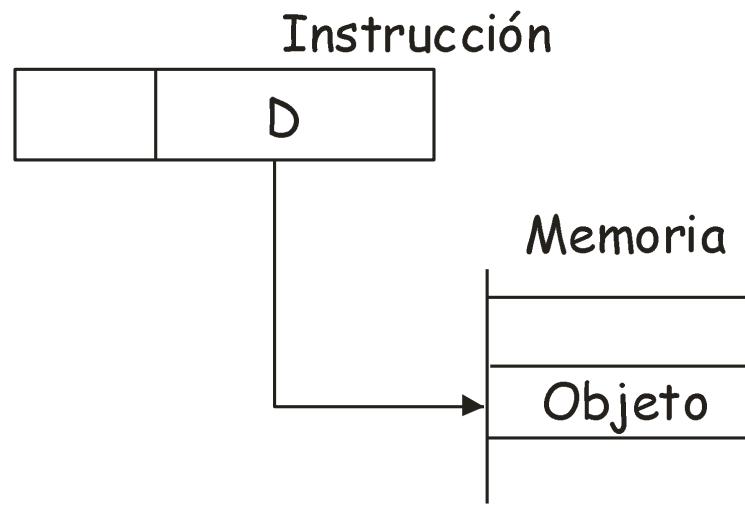
Direccionamiento directo absoluto a registro

- El objeto del direccionamiento está contenido en un registro. La instrucción contiene el registro que contiene el objeto del direccionamiento.
- **ADD .R4,.R5** $R4 \leq R4+R5$



Direccionamiento directo absoluto a memoria

- El objeto del direccionamiento está contenido en una dirección de memoria. La instrucción contiene la dirección completa de memoria que contiene el objeto del direccionamiento.
- **LD .R4,/1000** $R4 \leq M(1000)$



Direccionamientos relativos

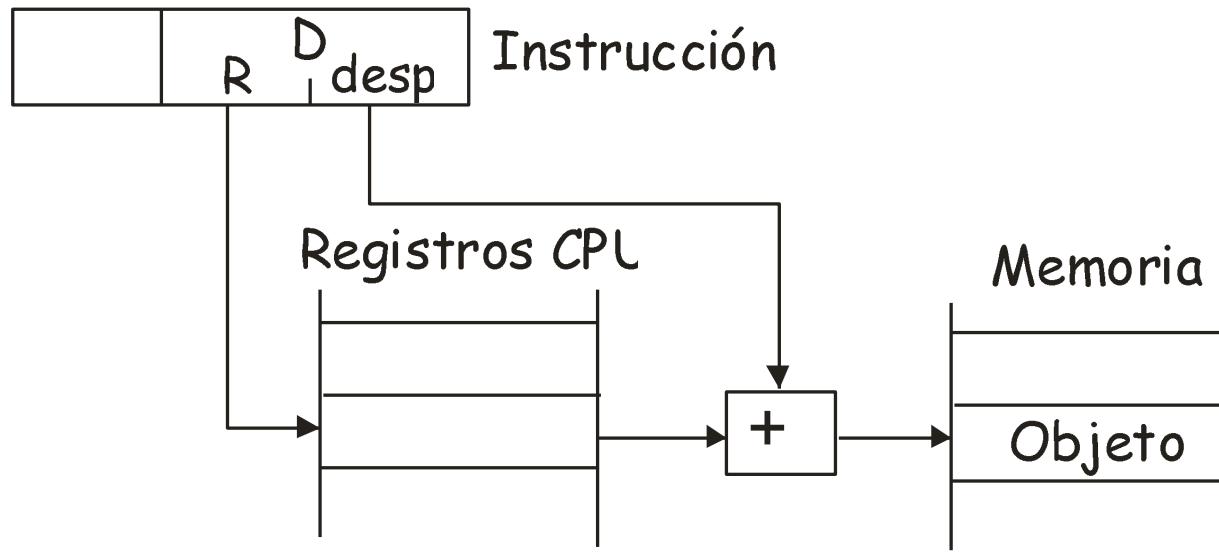
- El objeto del direccionamiento está contenido en una dirección de memoria. La instrucción contiene la dirección especificada en “partes”.
- Dependiendo de cómo se especifique la dirección:
 - Relativo a **registro base**
 - Relativo a **PC**
 - Relativo a **registro índice**

Direccionamiento relativo a registro base

- La dirección de memoria viene especificada en dos partes:
 - **Registro Base:** Registro de propósito específico o general que contiene una dirección a memoria.
 - **Desplazamiento:** Valor entero con signo.
- La dirección efectiva se calcula:
$$\text{Dir_Efectiva} = \text{Registro_Base} + \text{Desplazamiento}$$

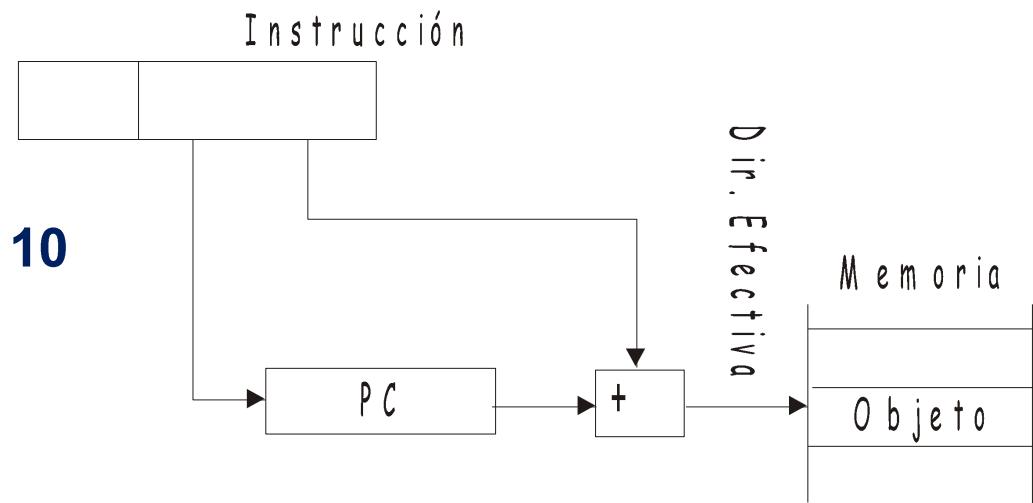
Direccionamiento relativo a registro base

- **LD .R1,#4[R7]** $R1 \leqslant \text{MEM}(R7+4)$
- El registro base se carga una dirección de memoria que contiene un conjunto de datos a los que se accede conociendo su posición relativa frente al comienzo de dicha zona.
 - Se usa para acceder a componentes de estructuras de datos.



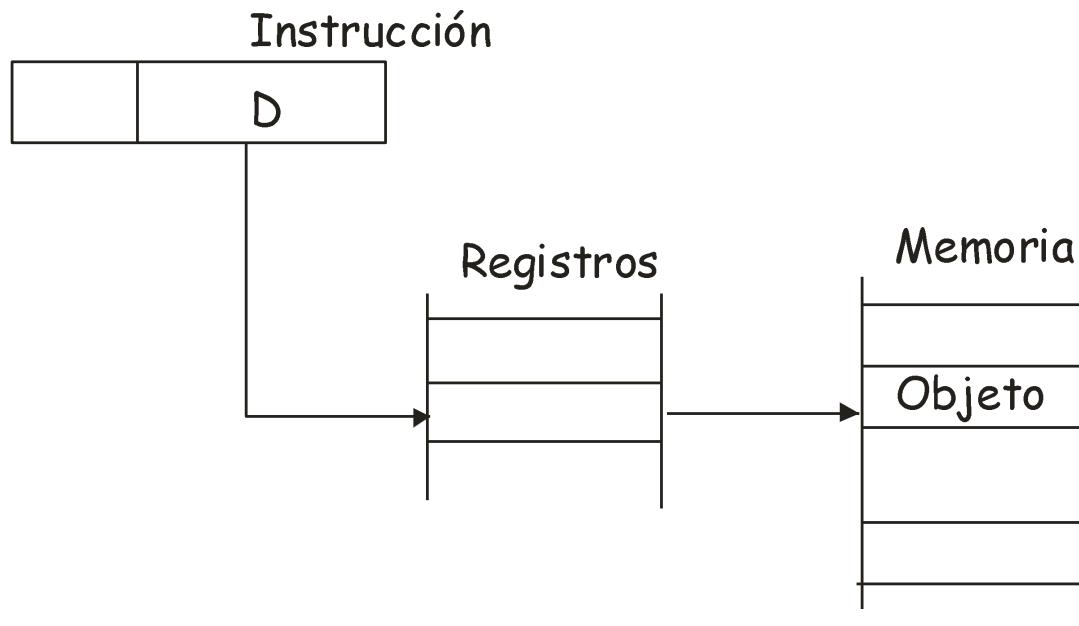
Direccionamiento relativo a PC

- Es un direccionamiento relativo a registro base en el que el registro base es el PC.
- El objeto de este direccionamiento suelen ser instrucciones.
- Permite alcanzar instrucciones “cercanas” a la que se está ejecutando.
- Ejecución de saltos “cortos”
- **BR \$10 $PC \leq PC + 10$**



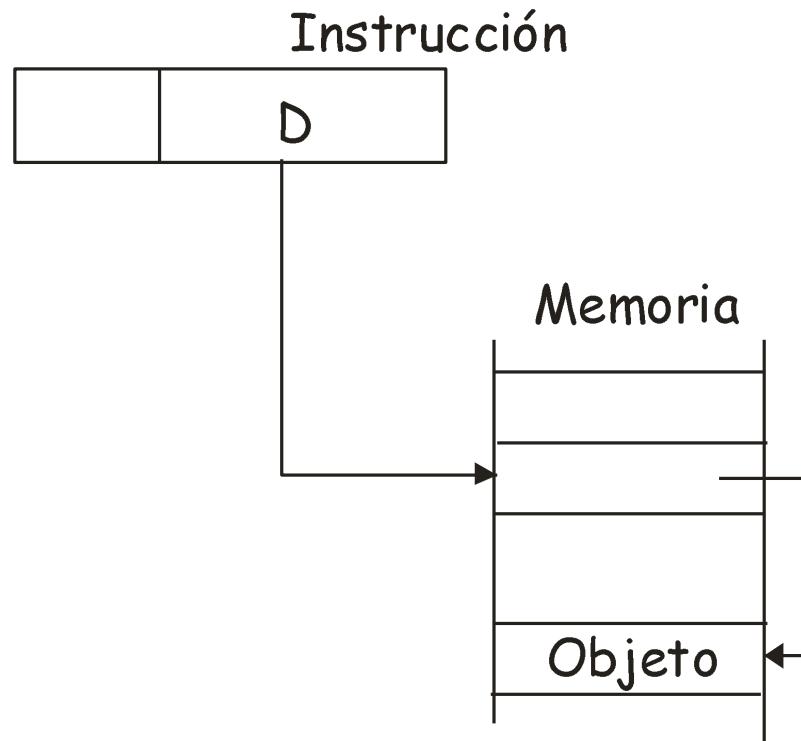
Direccionamiento indirecto a registro

- La instrucción contiene la especificación del registro que contiene la dirección de memoria donde está almacenado el objeto
- **LD .R1,[.R4]** $R1 \leqslant \text{MEM}(R4)$



Direccionamiento indirecto a memoria

- La instrucción contiene una dirección de memoria donde está contenida la dirección donde se almacena el objeto.
- **LD .R1,[/1000] R1 <= MEM(MEM(1000))**



Juego de instrucciones

- Conjunto de instrucciones del computador.
- “Herramientas básicas” con las que construir programas complejos
- Debe ser completo y eficaz.
- Tipos:
 - Transferencia
 - Aritméticas
 - Lógicas
 - Bifurcaciones
 - Desplazamiento y rotación
 - De bit

Transferencia de datos

- Mueven datos entre registros, registros y posiciones de memoria y entre posiciones de memoria.
- No modifican los biestables de estado.
- LD, ST y MOVE
 - LD .R2, #4[.R4] $R2 \leqslant \text{MEM}(R4+4)$
 - ST .R2, #4[.R4] $\text{MEM}(R4+4) \leqslant R2$
 - MOVE .R2,.R4 $R4 = R2$
 - MOVE [.R2],[.R4] $\text{MEM}(R4) \leqslant \text{MEM}(R2)$
- PUSH y POP
 - PUSH .R1 $SP \leqslant SP-4; \text{MEM}(SP) \leqslant R1$
 - POP .R1 $R1 \leqslant \text{MEM}(SP); SP \leqslant SP + 4$

Bifurcaciones incondicionales

- Modifican la secuencia del programa. Lo habitual es que la siguiente instrucción que se ejecuta sea la siguiente en secuencia. En este caso no es así.
- No modifican los biestables de estado.
- BR. Le sigue un direccionamiento a memoria cuyo objeto es la siguiente instrucción a ejecutar
- **BR /1000 PC <= 1000**
- **BR #4[R4] PC <= R4+4**
- **BR \$10 PC <= PC + 10**

Bifurcaciones condicionales

- Bcc dir. Dir es un **direcciónamiento a memoria**. Se utilizan para **realizar iteraciones** hasta que se cumple una condición determinada.

Si $cc = 1$ entonces se ejecuta la bifurcación
Si no se continua en secuencia

- Cc: Z, NZ, C, NC, V, NV, P, N o alguna combinación de estos biestables.
- **BZ /1000 Si $Z = 1$ $PC \leq 1000$**
- **BNC #4[R4] Si $C = 0$ $PC \leq R4+4$**
- **BP \$10 Si $S = 0$ $PC \leq PC + 10$**

Bifurcaciones con retornos

- Se utilizan para implementar saltos a subrutinas. Una subrutina es un fragmento de código que se utiliza en varias partes de un programa.
- Una vez realizado el subprograma (subrutina) se retorna a la instrucción siguiente desde la que se bifurcó. Por lo tanto es necesario almacenar la dirección de retorno.
- CALL dir y RET
- La dirección de retorno habitualmente se almacena en la pila lo que permite llamadas anidadas y recursión:
 - **CALL /1000** $SP \leq SP - 4; MEM(SP) \leq PC ;$
 $PC \leq 1000$
 - **RET** $PC \leq MEM(SP) ; SP \leq SP + 4$

Aritméticas y comparación

- **ADD .R1, .R2** $R1 \leq R1 + R2; \text{ mod. flags}$
- **SUB .R1, .R2** $R1 \leq R1 - R2; \text{ mod. flags}$
- **MUL .R1, .R2** $R1 \leq R1 \cdot R2; \text{ mod. flags}$
- **DIV .R1, .R2** $R1 \leq R1 / R2; \text{ mod. flags}$
- **ADDC .R1, .R2** $R1 \leq R1 + R2 + c; \text{ mod. flags}$
- **SUBC .R1, .R2** $R1 \leq R1 - R2 - c; \text{ mod. flags}$
- **CMP .R1, .R2** $R1 - R2 ; \text{ mod. flags}$

Lógicas

- Realizan la operación lógica indicada por el mnemónico bit a bit.
- Establecen el modelo de ejecución del computador
- **AND .R1,.R2 R1 <= R1 AND R2; mod. Flags**
- **XOR .R1,#4 R1 <= R1 XOR 4; mod. Flags**
- **OR .R1,.R2 R1 <= R1 OR R2; mod. Flags**
- **NOT .R1 R1 <= NOT R1; mod. Flags**
- Se utilizan para trabajar con máscaras
- **AND .R1, #1 ; Si Z = 1 el número es par**

Desplazamiento

- Realizan desplazamientos de bits a izquierda/derecha.

- **Lógicos.**

- **SHL .R1**



- **SHR .R1**



- **Aritméticos.** Realizan una multiplicación por 2 (ASL) o división por 2 (ASR).

- **ASL .R1**

- ASR .R1**

Rotación

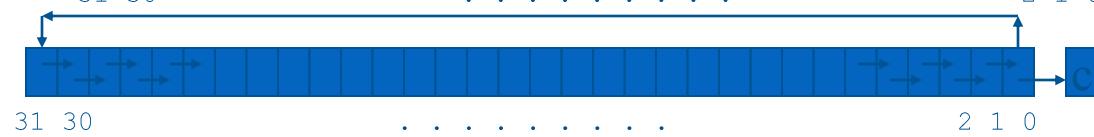
- Realizan rotaciones de bits a izquierda/derecha.

- **Rotación.**

- **ROL .R1**



- **ROR .R1**

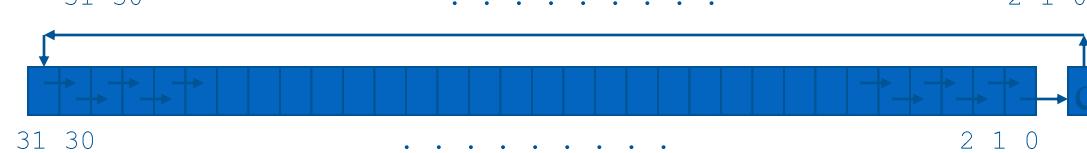


- **Rotación con acarreo.**

- **ROLC .R1**



- **RORC .R1**



De bit

- Ejecutan operaciones con un bit: pone a 1 o a 0 un bit.
- **CLR.I #3, .R1 ; Pone a 0 el bit 3 de R1**
- **SET.I #3, .R1 ; Pone a 1 el bit 3 de R1**
- **TEST.I #3, .R1 ; Z <= NOT(R1(3))**

Ejercicio

- Objetivo: identificar instrucciones y modos de direccionamiento y para qué se usan.
- Se utilizan listados de código ensamblador junto con el fuente correspondiente.
- El fichero con el fuente y los listados generados se encuentran en el moodle de la asignatura.
 - Todos los ficheros se pueden ver con un editor de texto.
- Para generar los listados se utilizaron tres compiladores distintos:
 - Compilador para i386/Linux
 - Compilador para SPARCv8 (OBC del UPMSat-2)
 - Compilador para ARMv7E-M (Cortex-M4)

Ejercicio

- Fichero fuente: product.adb
 - Multiplica dos matrices y muestra el resultado.
- 6 listados: product-*-* .asm
 - Generados para i386, LEON3 y ARM con dos niveles distintos de optimización:
 - O0 : sin optimizar el código máquina
 - O2 : con nivel de optimización 2 (alto)
- Ejemplo: product-i386-O0.asm es el listado correspondiente a la arquitectura intel i386 sin optimización.

Ejercicio

- Ejemplo de línea de listado (i386):

6: 81 ec ac 00 00 00 sub \$0xac,%esp

- El primer campo es la dirección de memoria principal donde se ubicaría.
- El segundo el código máquina de la operación.
- Por último la instrucción ensamblador correspondiente a dicho código máquina.
- En este caso se trata de una instrucción de resta que resta del registro esp el valor inmediato 172 (AC en hexadecimal) y el resultado se almacena en el mismo registro.

Memorias cache

Motivación

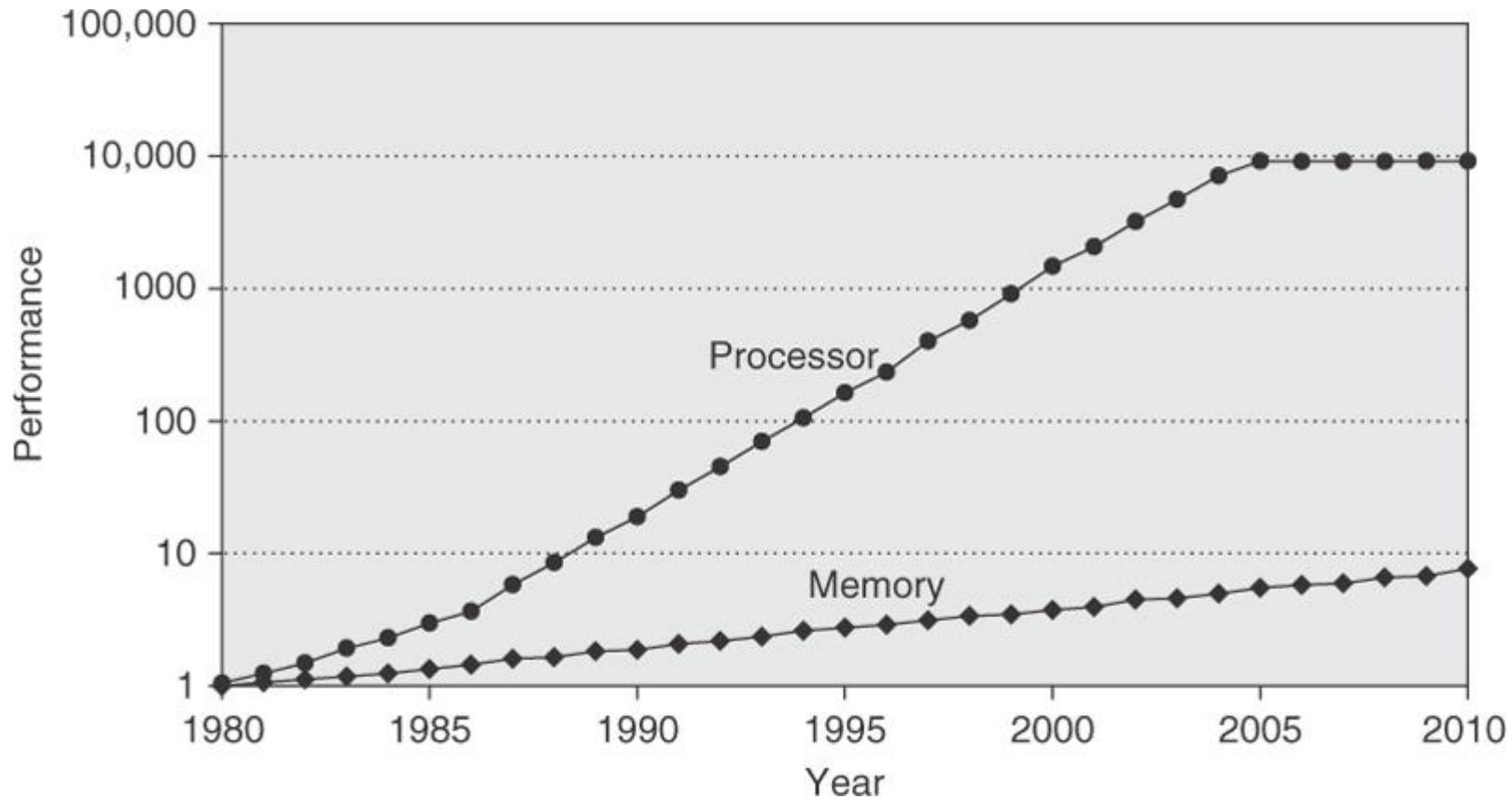


Figure 2.2 Starting with 1980 performance as a baseline, the gap in performance, measured as the difference in the time between processor memory requests (for a single processor or core) and the latency of a DRAM access, is plotted over time. ([Hennessy & Patterson 5th ed](#))

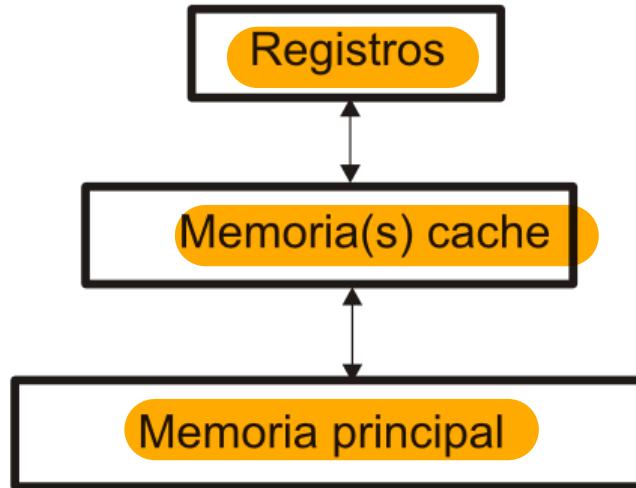
Jerarquía de memorias

Capacidad y
Tiempo de acceso

0,5 ns
 $< 1KB$

0,5 - 20 ns
 KB o MB

50 - 70 ns
 GB



Gestión

Compilador

Hardware

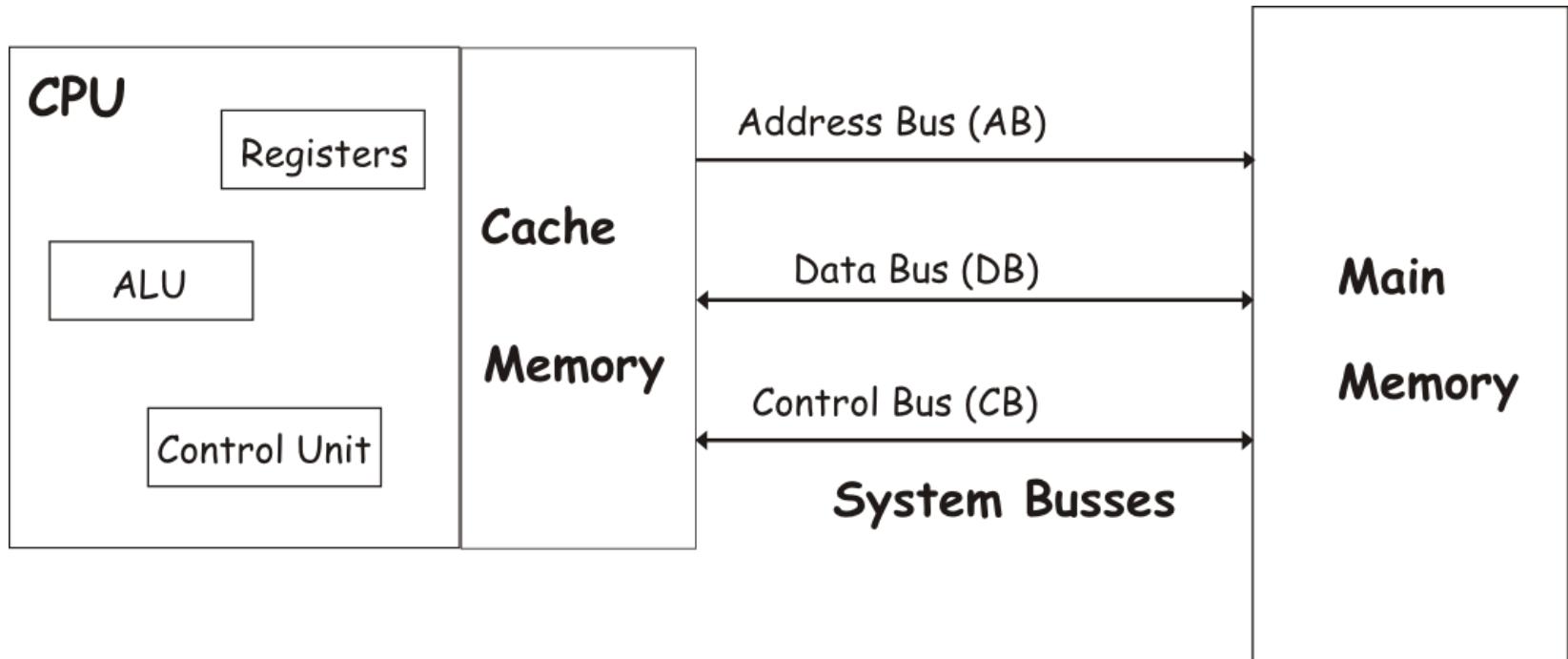
S.O.

“Controlador” de
caché

Propiedad de Inclusión

Nota: los tiempos de acceso y capacidades mostrados son similares a los de los computadores convencionales con un *gap* en el rendimiento similar al de la gráfica anterior. En los computadores espaciales los tiempos de acceso a los registros y cache son mayores y las memoria principal y cache tienen menor capacidad.

Esquema básico con memoria cache



Funcionamiento

- La CPU accede a una posición de memoria (fetch o ld/st).
- Se comprueba si la información está en memoria cache
- Si está
 - Se sirve el acceso desde memoria cache (el más rápido)
- Si no está
 - Se accede a memoria principal
 - Se transfiere la información desde memoria principal a memoria cache
 - La memoria cache transfiere la información a la CPU
- **Nota:** se copia siempre un conjunto de direcciones consecutivas: bloques de caché (unas decenas de bytes)

Proximidad de referencias

- Los programas tienden a hacer referencia a datos e instrucciones “próximos” a los recientemente utilizados.
 - **Temporal**: mismas direcciones a las que se ha accedido en un pasado reciente.
 - **Espacial**: direcciones próximas a las que se ha accedido recientemente.
 - **Secuencial** : direcciones consecutivas.

Ejemplo:

```
sum = 0;  
for (i=0; i<1000; i++)  
    sum = sum + v[i];
```

Ejemplo de traza

- **Traza** de un programa: secuencia de direcciones a las que accede durante su ejecución.
- **Ejemplo** anterior con direccionamiento a byte, 4 bytes/instr., 4 bytes/v[i], v desde dir. 10000 y código a partir de dir. 0

0	and r3, r0, 0
4	or r1, r0, 0
8	or r2, r0, low(v)
12	or.u r2, r2, high(v)
16	buc: ld r4, r2, 0
20	add r3, r3, r4
24	add r2, r2, 4
28	add r1, r1, 1
32	cmp r15, r1, 1000
36	bb1 lt, r15, \$buc

Traza de referencias

0	4	8	12	16	10000	20	24	28	32	36
				16	10004	20	24	28	32	36
				16	10008	20	24	28	32	36
				...						
				16	13992	20	24	28	32	36
				16	13996	20	24	28	32	36

1000
iteraciones

Proximidad temporal: acceso a instrucciones (en vertical)

Proximidad espacial: acceso a datos $v[i]$ (10000 .. 13996)

Proximidad secuencial: acceso a instrucciones (en horizontal)

Bloque o línea de cache

- **La proximidad temporal aconseja:**
 - Enviar el contenido de la dirección referenciada al nivel más próximo a la CPU
- **La proximidad secuencial (espacial):**
 - Enviar también el contenido de las direcciones cercanas.

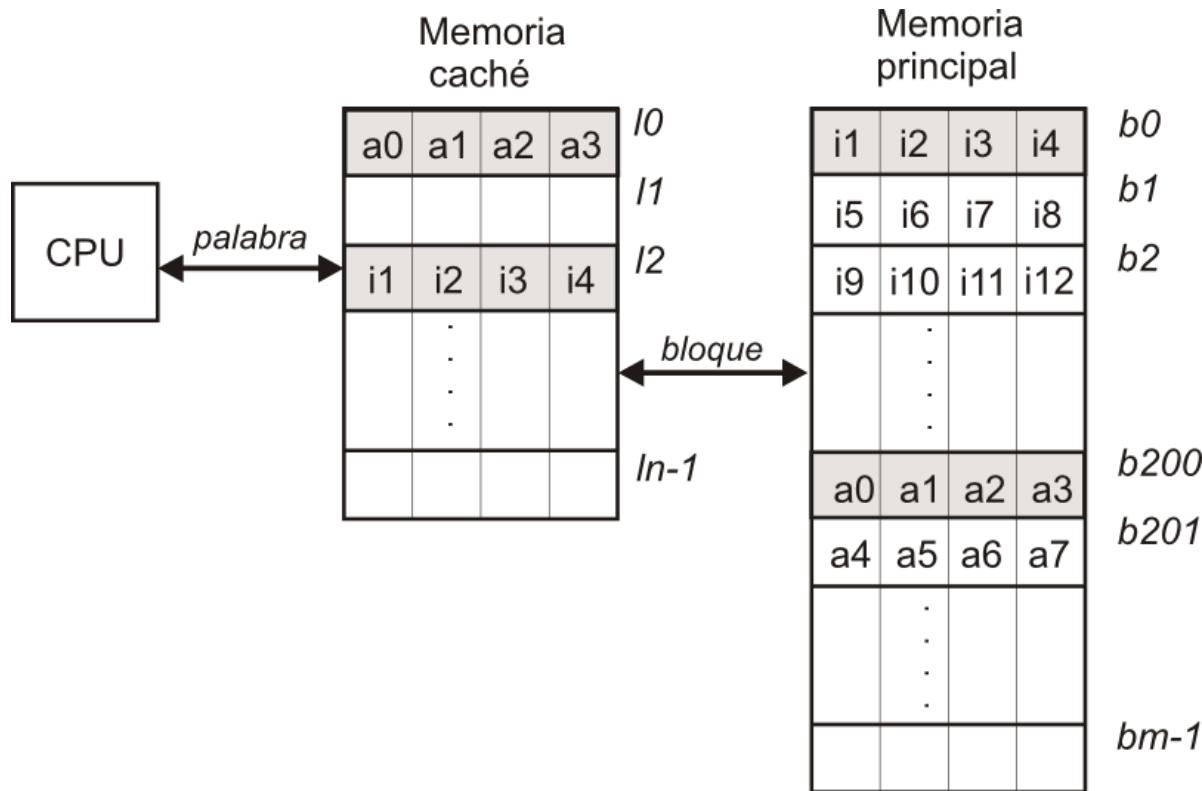


Bloque:

- Agrupación de palabras de memoria consecutivas.
- Unidad de información que puede, o no, estar presente en un nivel de la jerarquía (indivisible).
- Unidad de información que se transfiere entre dos niveles consecutivos de la jerarquía.

Memoria cache

- Memoria rápida de pequeña capacidad, situada entre la CPU y la Memoria principal.
- Contiene parte de la información residente en la Memoria principal



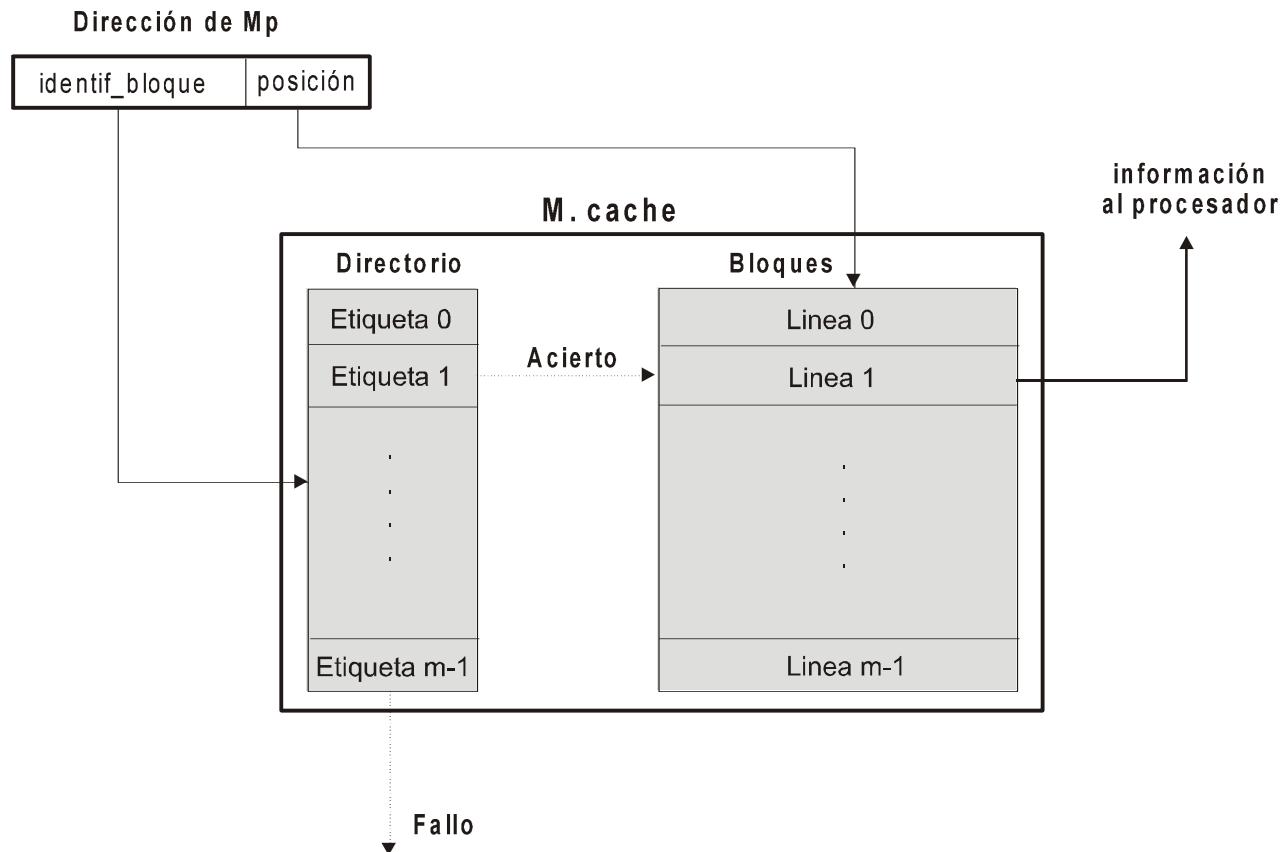
Componentes

1. **almacenamiento ('líneas')**: donde se albergan físicamente los bloques* que se suben del nivel inferior
2. **directorio**: el contenido depende de la política de ubicación y permite conocer qué bloque de Mp está almacenado en cada línea
3. **controlador**: implementa todas las políticas y realiza el servicio a los fallos

(*) El almacenamiento se estructura en **líneas**, y cada una contiene un conjunto de dirs. consecutivas de Mp: **bloque de caché**: entre 4 y 64 bytes. El espacio de direcciones de Mp se considera dividido en bloques de igual tamaño

Fucionamiento interno

- Se accede mediante una dirección de Mp
- ¿Cómo saber si la información buscada está en la Mca?



Ubicación de bloques

Ejemplo UPMSat-2:

Mca:

Capacidad total : 16 KB [2^{14} bytes]

Tamaño de los bloques: 32 bytes [2^5 bytes]

————→ Nº de líneas = 512 líneas [2^9 líneas]

Mp:

Capacidad: 4 MB [2^{22} bytes]

————→ Nº de bloques = 2^{17} bloques de Mp

Política de ubicación:

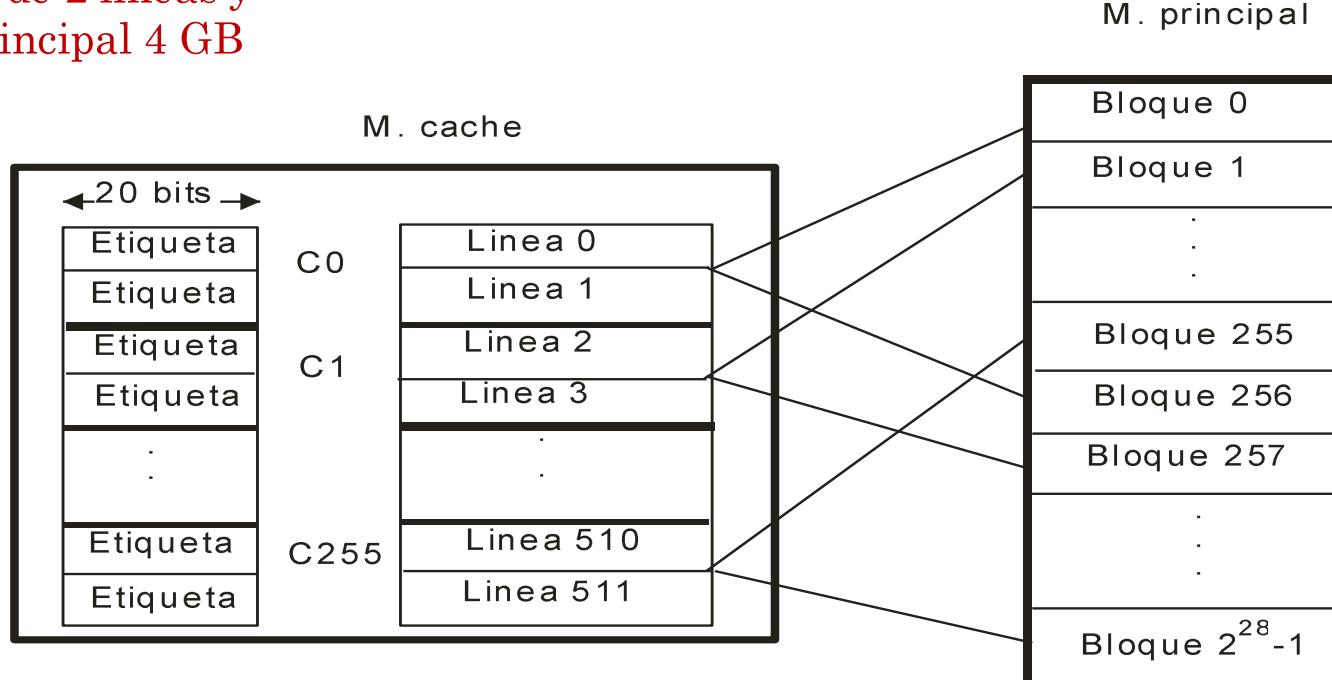
Bloque_i de Mp —————→ Línea_j de Mca

Ubicación asociativa por conjuntos

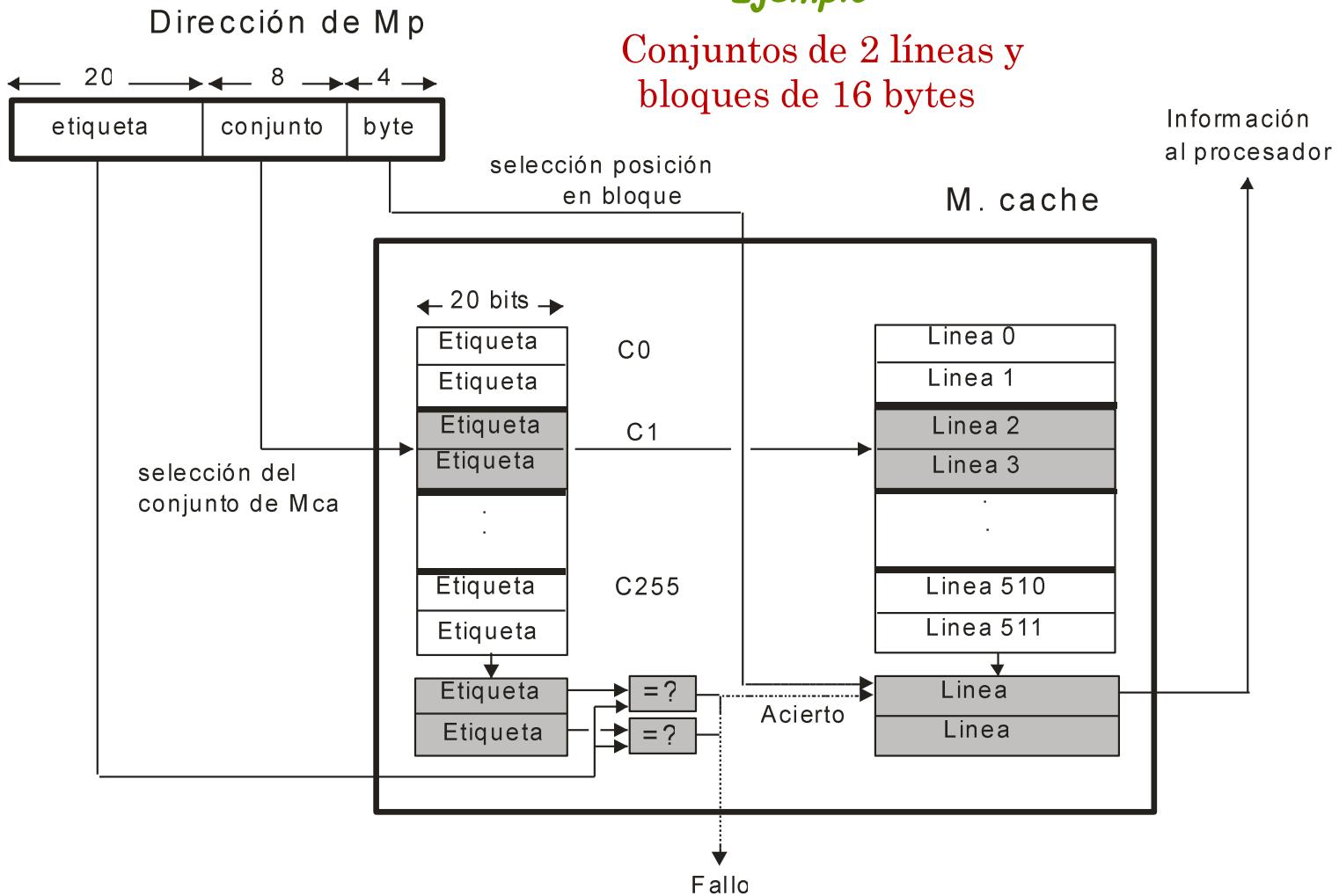
- Cada bloque de Mp puede situarse en cualquier línea de un conjunto predefinido:
bloque i => conjunto i módulo (nºconjuntos de Mca)

Ejemplo:

Conjuntos de 2 líneas y
memoria principal 4 GB



Ubicación asociativa por conjuntos



Análisis

- Sencillez, reducido coste de implementación y consumo energético si el número de bloques por conjunto es reducido. Podría ser un sólo bloque por conjunto.
 - Mayor complejidad, coste y consumo energético cuanto mayor sea el número de bloques por conjunto.
- Puede producir fallos adicionales si la traza del programa usa simultáneamente más bloques que se ubican en el mismo conjunto de los que caben.
 - Estos conflictos originan multitud de fallos.
 - Menor probabilidad de conflicto cuanto mayor sea el número de bloques por conjunto.

Ejemplo: intel Core i7-6700 (Skylake)

- Memorias caché:

Bloques de 64B

Conjuntos de 8 bloques

L1i: 32KB (4 ciclos)

L1d: 32KB (4 ciclos)

L2: 256KB (12 ciclos)

L3: 8MB (38-42 ciclos)

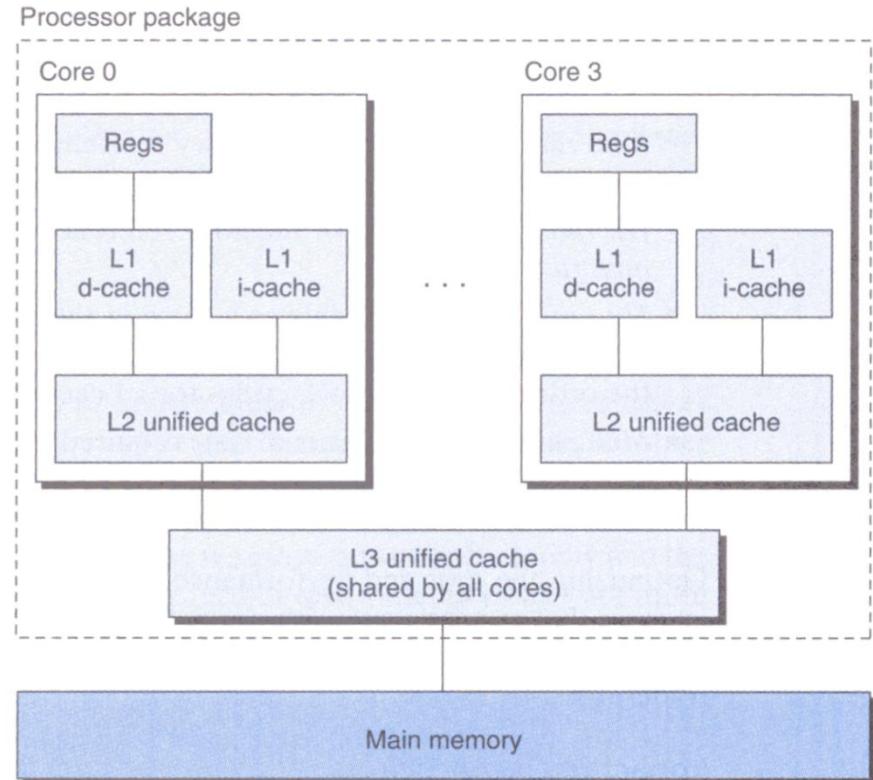
- Memoria RAM:

Hasta 64 GB (42 ciclos + 51ns)

Hasta 34.1 GB/s

- Frecuencia máxima de reloj 4 GHz

(Computer Systems: A Programmer's Perspective, Second Edition, Randal E. Bryant and David R. O'Hallaron)



Ejercicio con el OBC del UPMSat-2

- 2 memorias caches separadas para instrucciones y datos.
- Tamaño de las caches 32 KB cada una.
- Bloques de 32 bytes.
- 4 bloques por conjunto.
- Por lo tanto están organizadas en 128 conjuntos de 4 líneas cada uno.
- Se medirá la diferencia de tiempo de ejecución de 2 *benchmarks* con las caches habilitadas e inhibidas.
 - De este modo se comprobará el aumento de la velocidad de ejecución (*speedup*) provocado por la memoria cache.

Resultados

- OBC del UPMSat-2:

Caches disabled

Microseconds per KiloWhetstone := 12982.450750000

Microseconds per KiloDrystone := 413.641250000

Caches enabled

Microseconds per KiloWhetstone := 3420.320500000

Microseconds per KiloDrystone := 103.209250000

- Computador similar (Tamaño 4KB, bloques de 16 bytes y 1 bloque por conjunto):

Caches disabled

Microseconds per KiloWhetstone := 19363.969500000

Microseconds per KiloDrystone := 632.503000000

Caches enabled

Microseconds per KiloWhetstone := 4016.783250000

Microseconds per KiloDrystone := 109.937250000

Análisis de los resultados

- El *speedup* varía entre 4 y 6 según computador y *benchmark*.
- Hay que tener en cuenta que la velocidad del procesador es de sólo 20 MHz.
 - Es de esperar que el *speedup* sea mucho mayor en computadores con *velocidades de reloj* mucho **mayores**.
 - Este es el caso no sólo de computadores de ofimática sino también de computadores empotrados como los que se usan en aviónica y otras áreas.

Ejercicio de conflictos de cache

- Utilizaremos estructuras de datos que haremos coincidir en los mismos conjuntos de cache.
 - Por ejemplo: matrices.
- Si accedemos consecutivamente a direcciones de memoria que se ubican en el mismo conjunto de cache, podemos tener conflictos si el número de bloques accedidos simultáneamente es mayor que el número de bloques por conjunto.
 - Por ejemplo: en la multiplicación de matrices.
- Estos conflictos no se darán en el OBC porque tiene 4 bloques en cada conjunto pero sí en el otro computador ya que sólo hay un bloque por conjunto.
 - Para ocasionarlos en el OBC habría que hacer coincidir al menos 5 bloques. Podría darse el caso en operaciones más complejas.
- Mediremos el tiempo de cómputo de una iteración con y sin conflicto.

Resultados

- Computador similar
 - Programa sin conflictos:
Microseconds per Iteration := 0.390275130
 - Programa con conflictos:
Microseconds per Iteration := 0.448386750
- OBC
 - Programa sin conflictos:
Microseconds per Iteration := 0.355148700
 - Programa con conflictos:
Microseconds per Iteration := 0.355148700

Análisis de resultados

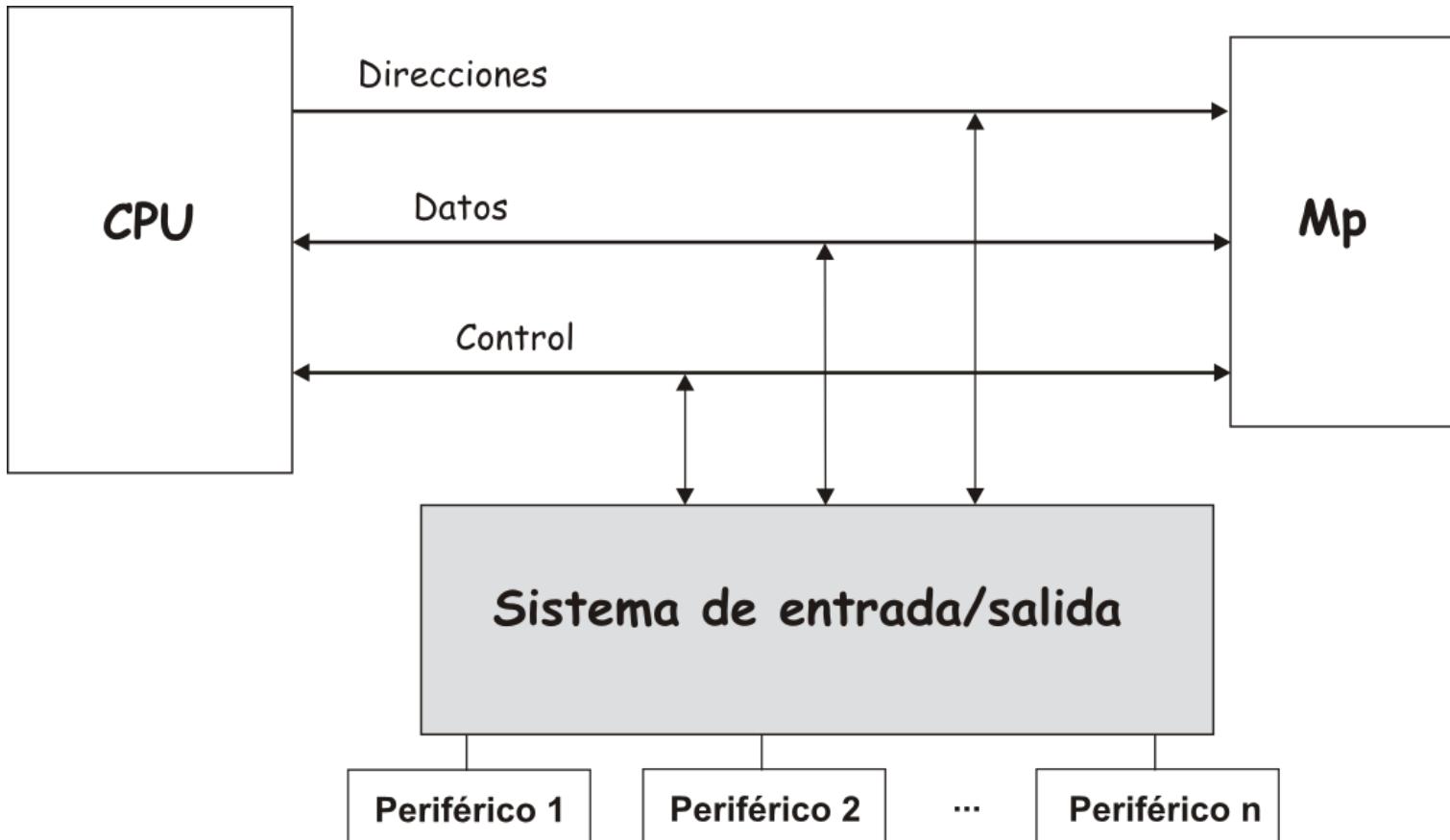
- En el OBC no se producen conflictos ya que tiene 4 bloques por conjunto. Por lo tanto no se producen fallos adicionales y el tiempo de cálculo no varía.
- En el otro computador se producen conflictos en los accesos a las matrices (instrucciones Id y st) ya que sólo hay un bloque por conjunto.
 - Se producen fallos adicionales resultando en un incremento del tiempo de cálculo del 14%
 - Hay que tener en cuenta que el impacto se reduce al acceso a datos y no a las instrucciones máquina (ciclos de fetch).
 - Si se compila sin optimización el impacto es menor porque los bucles tienen más instrucciones pero los mismos accesos a datos.
 - La diferencia de tiempo de acceso entre la memoria cache y la memoria principal no es muy grande y en un computador “terrestre” el incremento sería mayor.

Sistema de entrada/salida

Objetivos

- Conocer el **funcionamiento** de los dispositivos periféricos más importantes o habituales de los computadores.
- Comprender los problemas que surgen al intercambiar información con el mundo exterior a través de los dispositivos periféricos.
- Conocer los mecanismos que proporcionan los computadores para realizar dicho intercambio y sincronizarse con los periféricos.

Sistema de entrada/salida



Dispositivos periféricos

■ Ejemplos:

- “Ofimática”:
 - Ratón
 - Teclado
 - Impresora
 - Disco duro
 - LANCE (Local Area Network Controller for Ethernet)
 - ...
- “Industriales”
 - Sensor de temperatura, presión, campo magnético, ...
 - Motores eléctricos
 - Magnetopares
 - ...

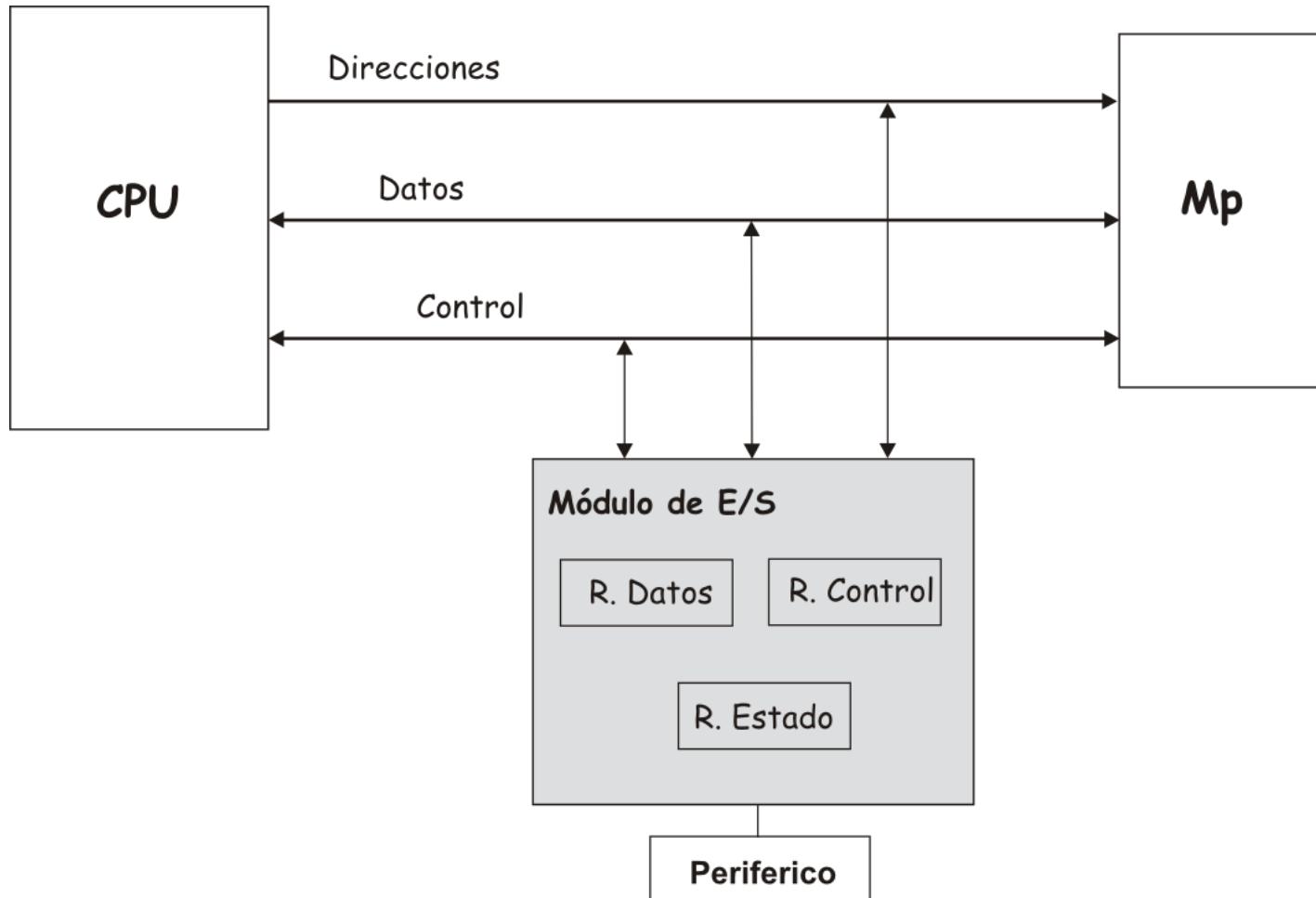
Dispositivos periféricos

- **Gran diversidad de periféricos con características muy diferentes :**
 - Modo de funcionamiento
 - Formato y tamaño de los datos
 - Velocidad de transferencia
 - Tiempo de acceso
- Por lo tanto es necesario **“unificar” la visión hardware de los periféricos**
- **Módulos de E/S:** ocultan las particularidades de cada periférico. La **CPU dialoga con todos estos módulos con los mismos mecanismos.**

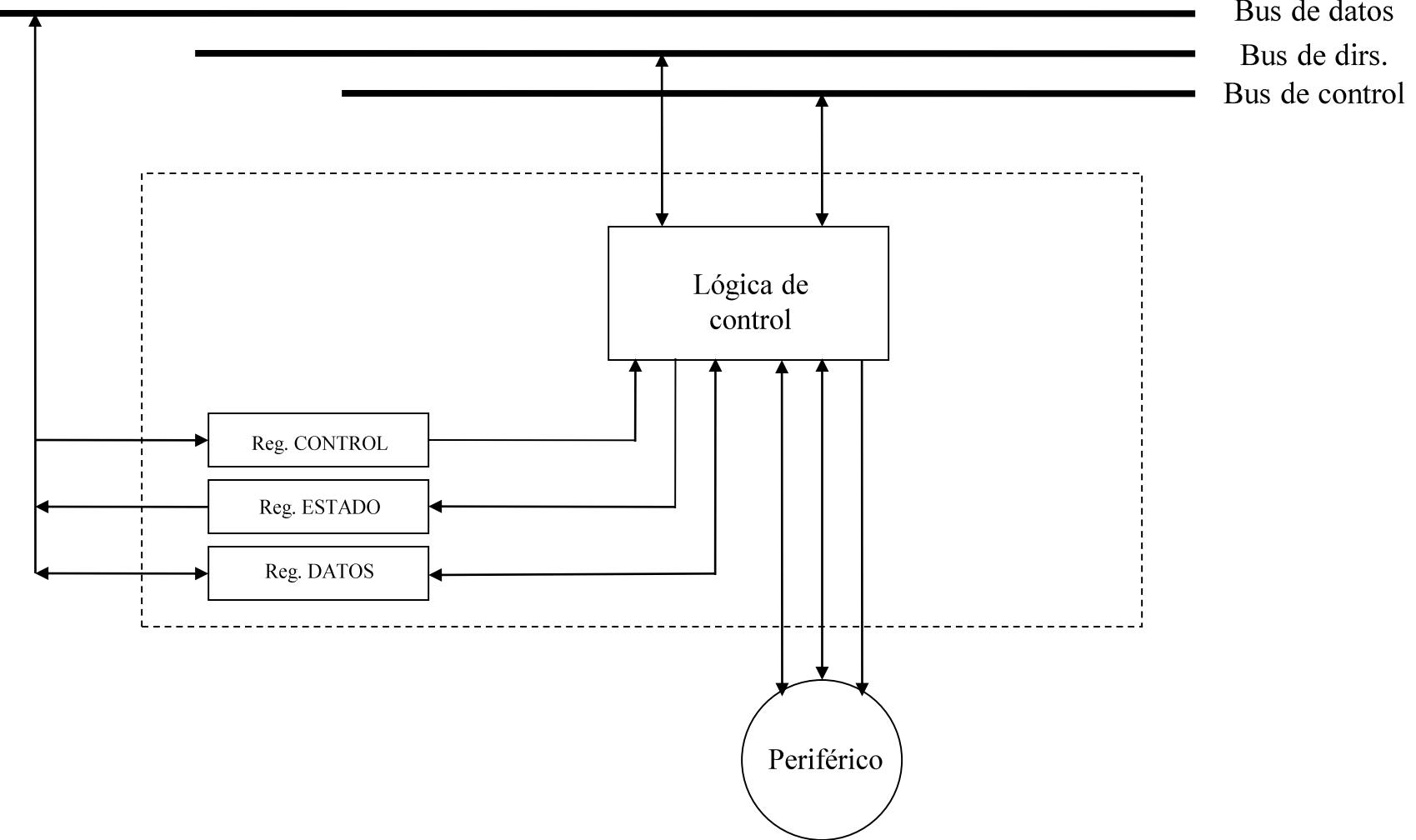
Módulos de entrada/salida

- Tienen dos interfaces: uno para dialogar con la CPU (estándar) y otro con el periférico (específico de cada periférico).
- Sus funciones son:
 - Sincronización: control y temporización de la comunicación
 - Comunicación con la CPU.
 - Comunicación con el periférico.
 - *Buffering*, por la diferencia de velocidades y tipos de datos.
 - Control del periférico.

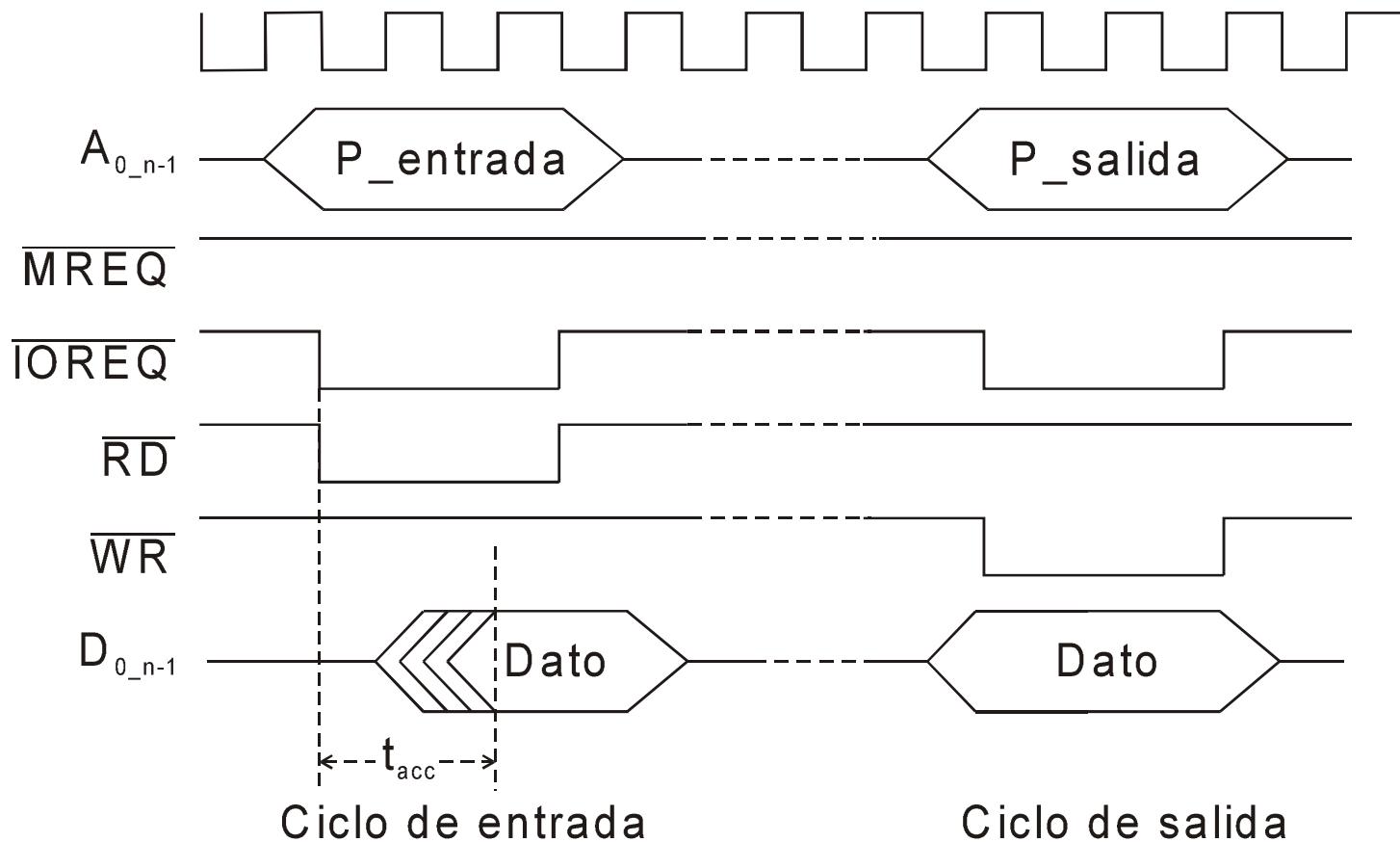
Sistema de entrada/salida



Estructura



Ciclos de bus de entrada/salida



Instrucciones de entrada/salida

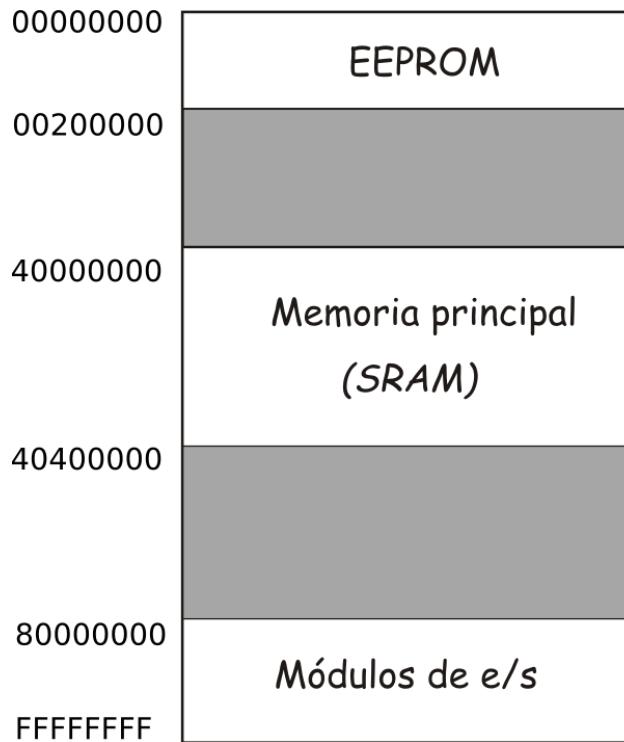
- Instrucciones de la arquitectura del computador para la transferencia entre los registros de la CPU y los registros de los módulos de E/S:

OUT .R1 , /Dir_Reg_Cont_Px; (orig → dest)

IN .R1 , /Dir_Reg_Dat_Px; (dest ← orig)

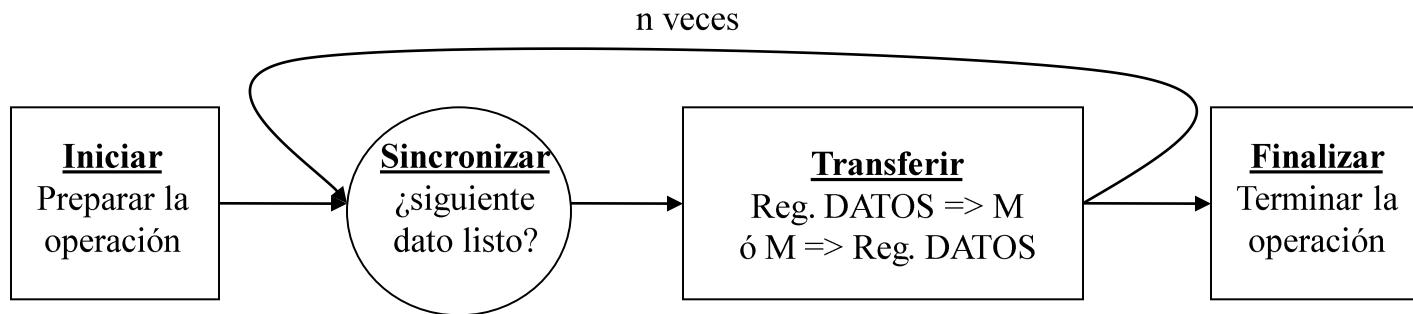
- Algunas arquitecturas no usan instrucciones específicas, usan LD y ST.
 - En este caso se usa parte del único espacio de direcciones físicas para los registros de entrada/salida (habitualmente la parte alta).

Ejemplo: OBC del UPMSat-2



Operación de entrada/salida

- Transferencia de un bloque de n palabras entre el periférico y la memoria principal.
- El tamaño lo determina el periférico: sector de un disco, bloque de Ethernet, ...
 - Puede ser igual a 1 byte en muchos dispositivos: teclado, ratón, línea serie, ...



Técnicas de entrada/salida

- Los periféricos son más lentos que la CPU y memoria.
- E/S y periférico son independientes y trabajan en paralelo.
- Técnicas de E/S:
 - E/S programada 
 - E/S por Interrupciones 
 - E/S por DMA 
- El grado de participación de la CPU varía.

E/S programada o directa

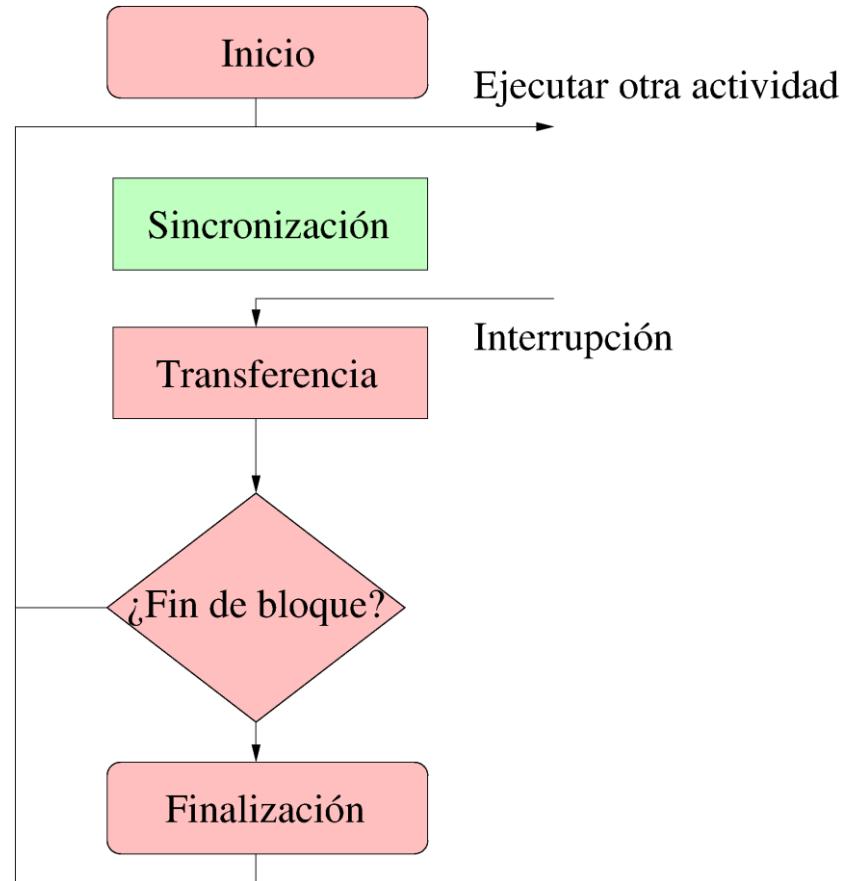
- Todas las fases las realiza la CPU mediante la ejecución de un programa.
- La sincronización se realiza ejecutando instrucciones en un bucle de espera.
 - Iterativamente se lee el registro de estado del módulo E/S y se comprueba la condición de nuevo dato (*polling*).
- La transferencia a/desde memoria se realiza ejecutando instrucciones in/out sobre el registro de datos e instrucciones ld/st sobre la dirección correspondiente de memoria.

Análisis

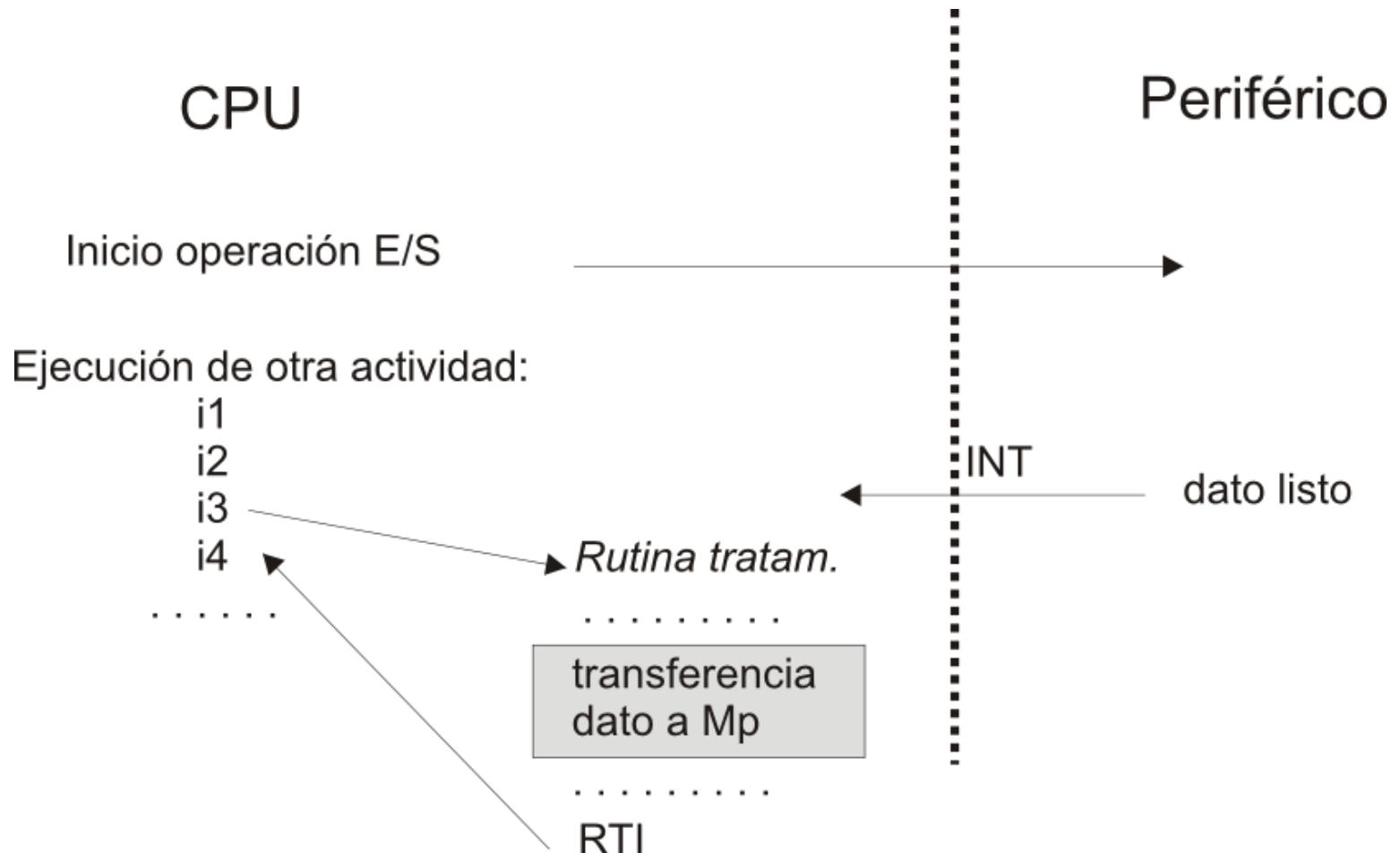
- Las operaciones se realizan por un programa que nunca abandona su flujo de ejecución.
- Los periféricos son “lentos” y se utiliza mucho tiempo de CPU en la sincronización.
 - Ejemplo: el tiempo medio de acceso de un disco duro es del orden de ms.
 - Una CPU con un reloj de GHz podría ejecutar millones de instrucciones durante ese tiempo de acceso.
- Se podrían usar esas instrucciones para realizar otras actividades en un sistema concurrente.
- Esta técnica es válida para pequeños sistemas dedicados.
 - Ejemplo: control remoto del proyector (ETSIInf.)

Entrada/Salida mediante interrupciones

- La CPU no se encarga de la sincronización.
- El módulo avisa a la CPU cuando está listo para una nueva transferencia.
- Se ahorra mucho tiempo de CPU que se usa para ejecutar otras actividades.



Entrada/Salida mediante interrupciones



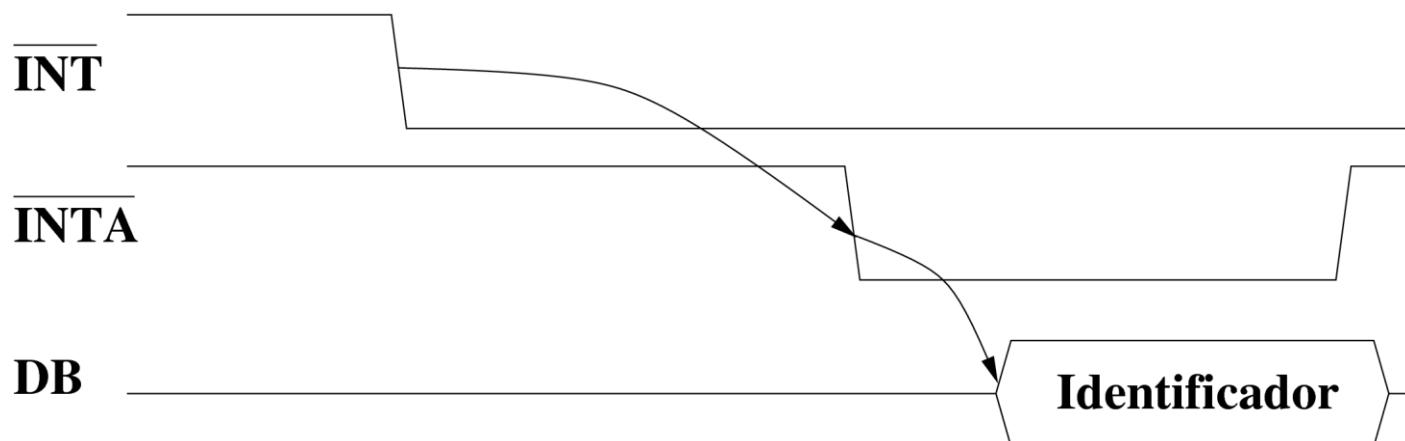
Solicitud de interrupciones



- Mediante una nueva señal a la unidad de control.
- El módulo la mantiene activa hasta que la interrupción es atendida.
- No se puede impedir que el módulo pida interrupciones, pero existen instrucciones (privilegiadas) para inhibir **DI (Disable Interrupts)** y habilitar **EI (Enable Interrupts)** la atención a las interrupciones.

Reconocimiento de interrupciones

- Mediante el **ciclo de bus de reconocimiento de interrupciones** la CPU pide, activando una nueva señal INTA (INTerrupt Acknowledge), que el peticionario de la interrupción (módulo E/S) se identifique.
- El identificador se carga durante la inicialización del computador en el registro del vector de interrupción del módulo de entra/salida.



Módulo de entrada/salida

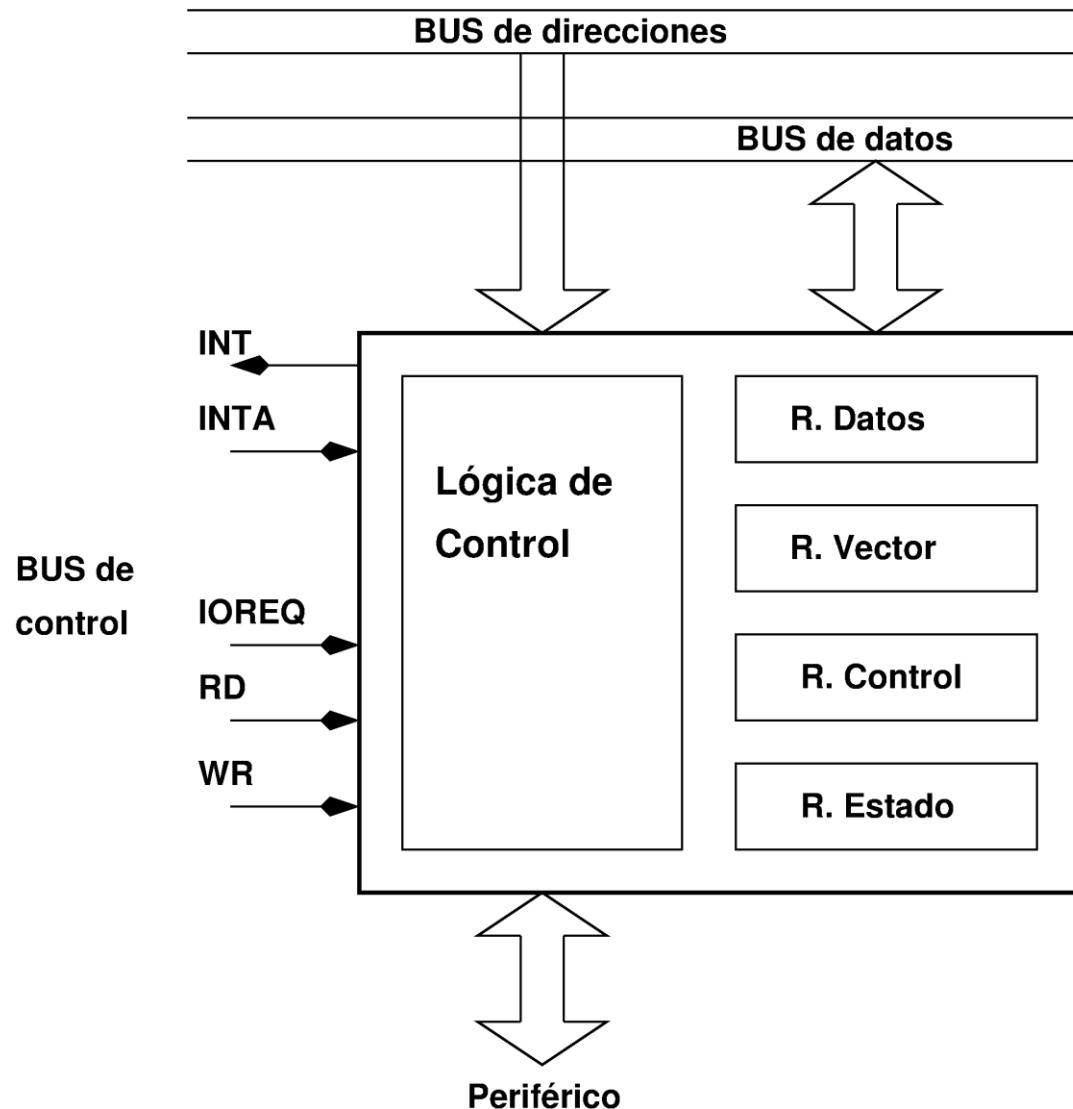
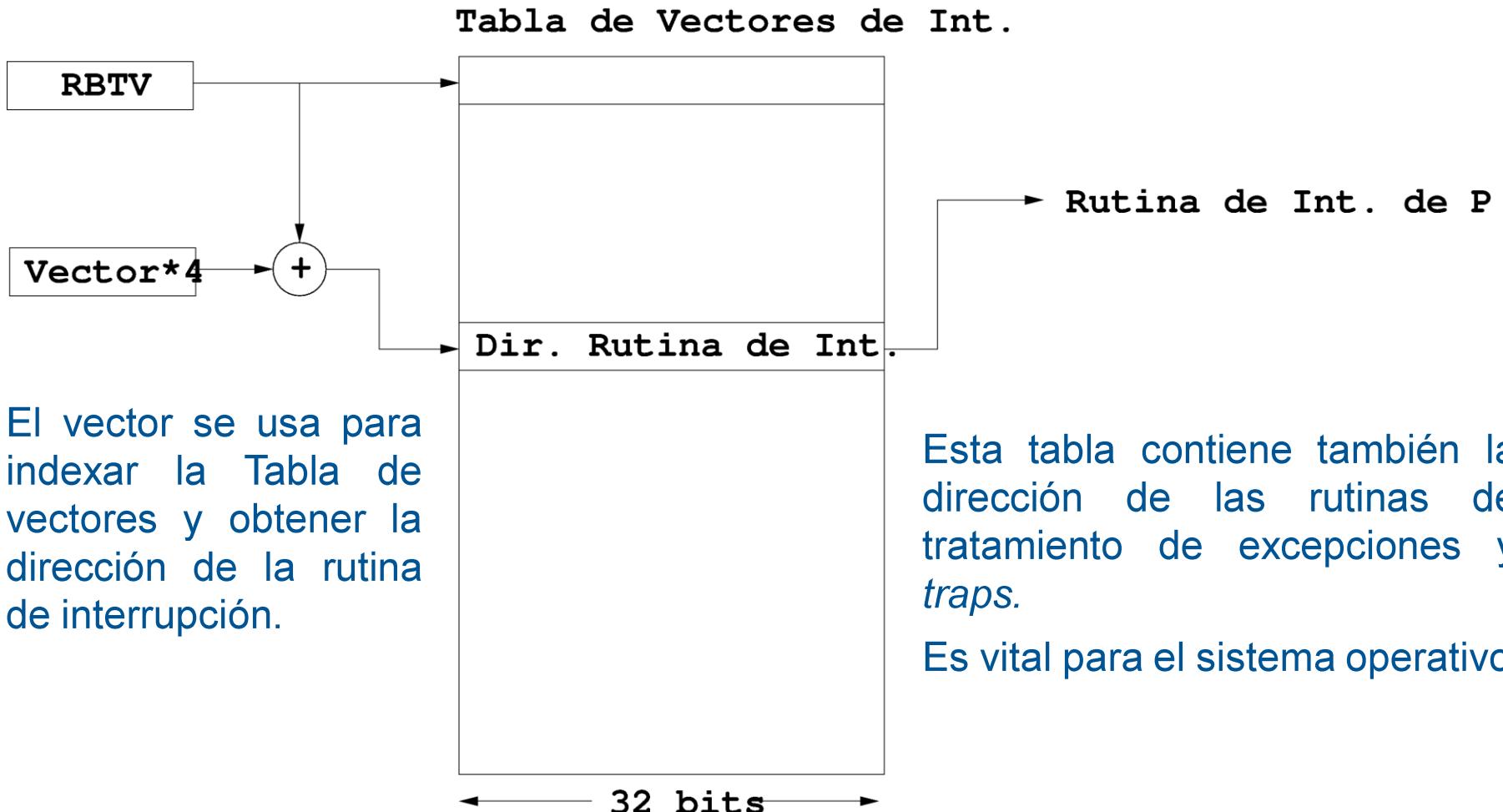


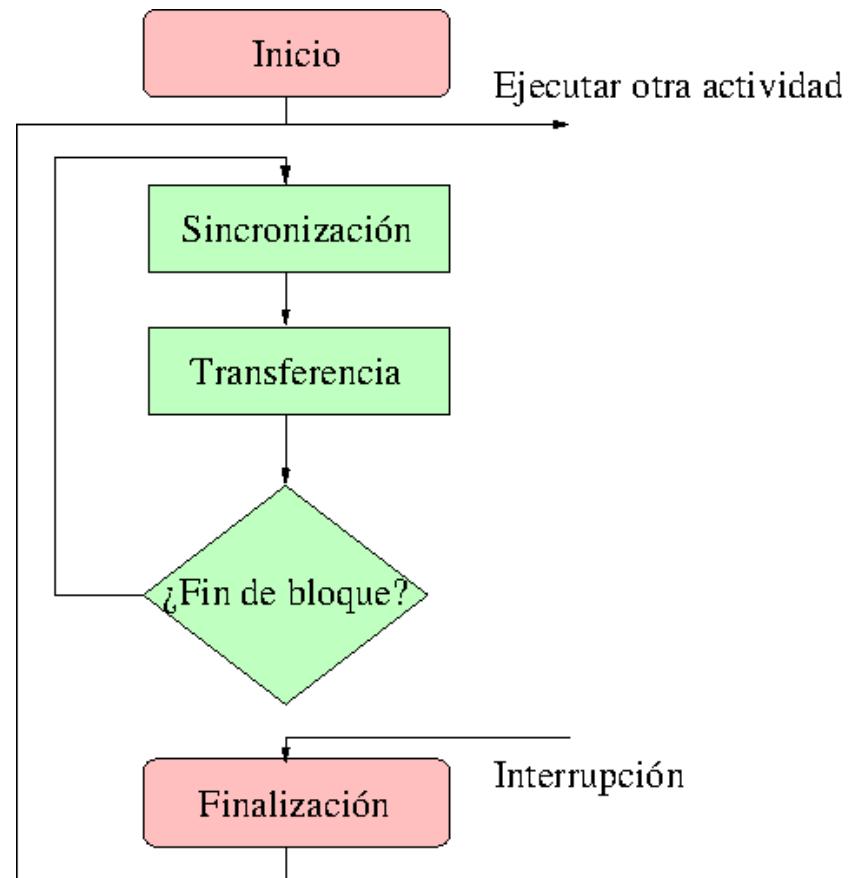
Tabla de vectores de interrupción

RBTV: Registro de propósito específico.



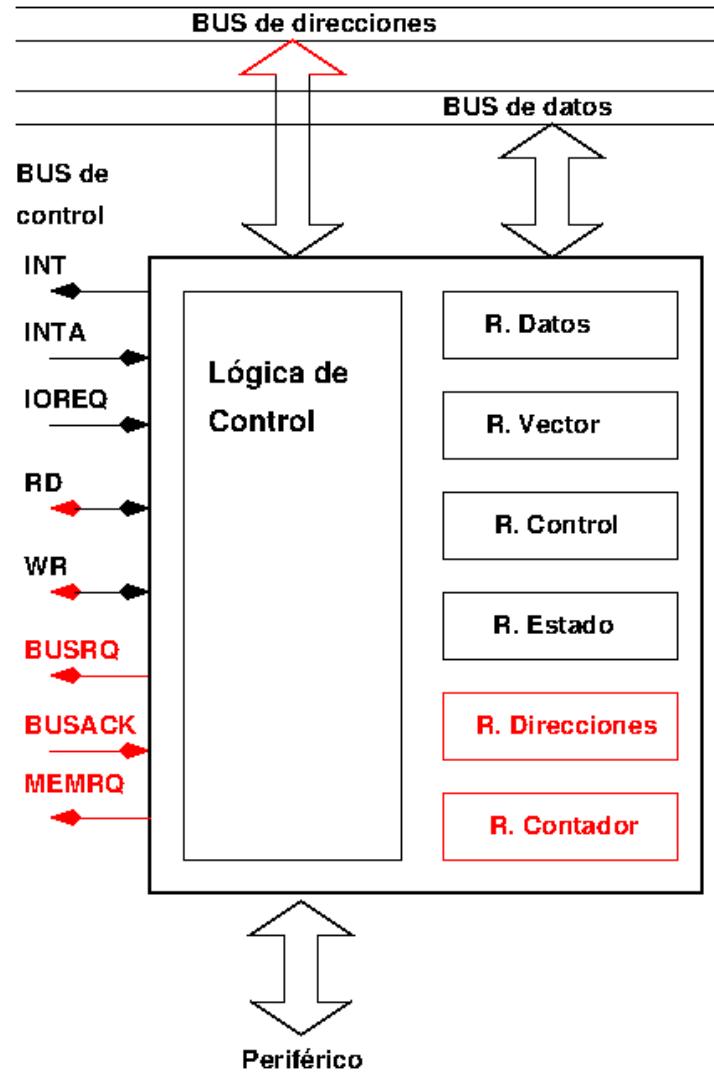
Entrada/salida mediante DMA

- La CPU se encarga de iniciar la operación.
- El módulo de entrada salida se encarga por hardware de la sincronización y transferencia y avisa cuando ha terminado mediante una interrupción.
- La CPU finaliza la operación.
- Hay una única interrupción por operación: se ahorra mucho tiempo de CPU con dispositivos de bloque.

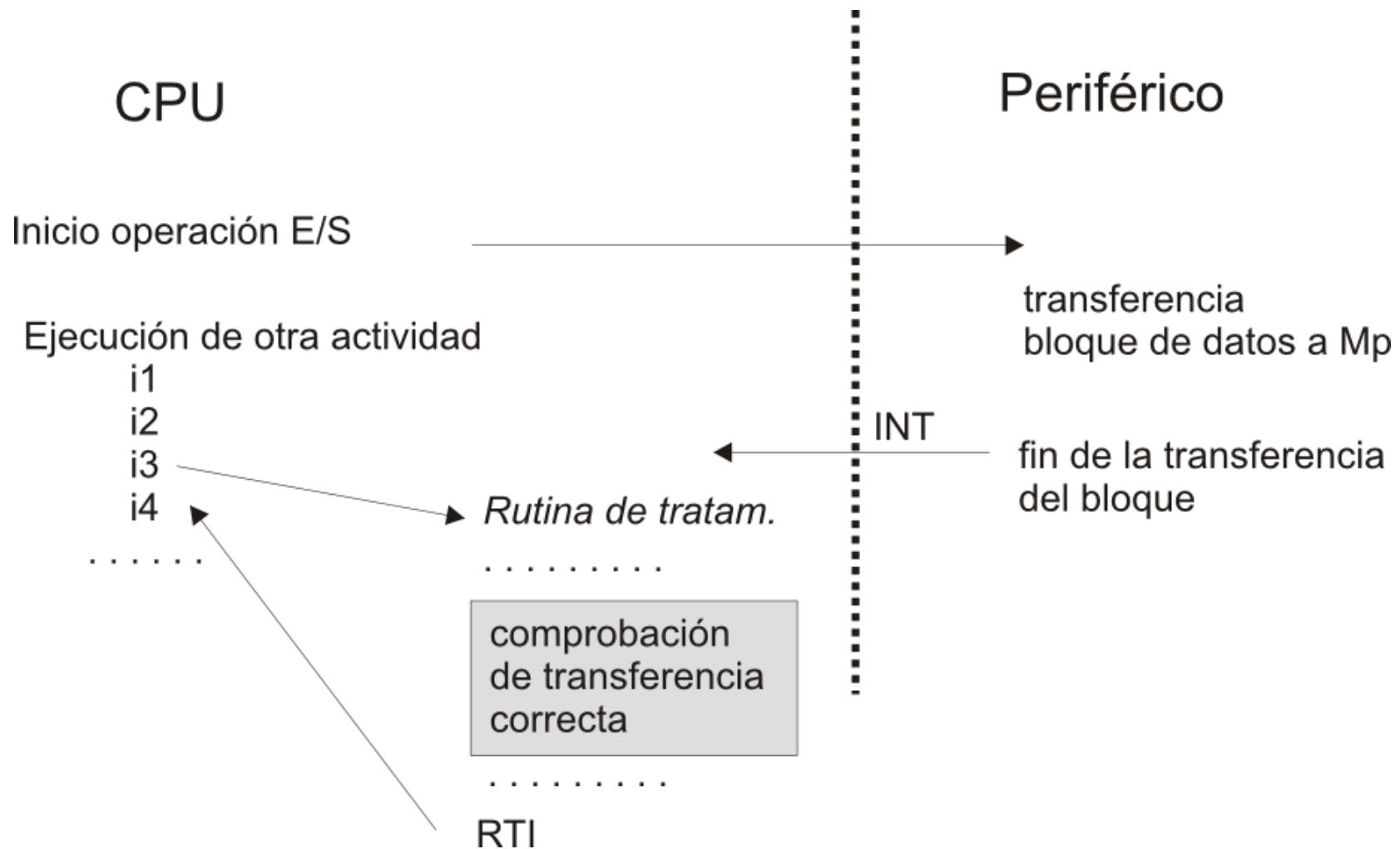


Módulo de E/S con capacidad de DMA

- El módulo, cuando hay datos listos, solicita los buses con BUSRQ.
- La CPU los cede al final del ciclo de bus en curso y lo indica con BUSACK.
- El módulo inicia el ciclo de bus para realizar la transferencia con memoria.
- Cuando acaba devuelve el bus desactivando BUSRQ.
- La CPU recupera los buses, desactiva BUSACK y continua con la ejecución de instrucciones.



Entrada/salida mediante DMA



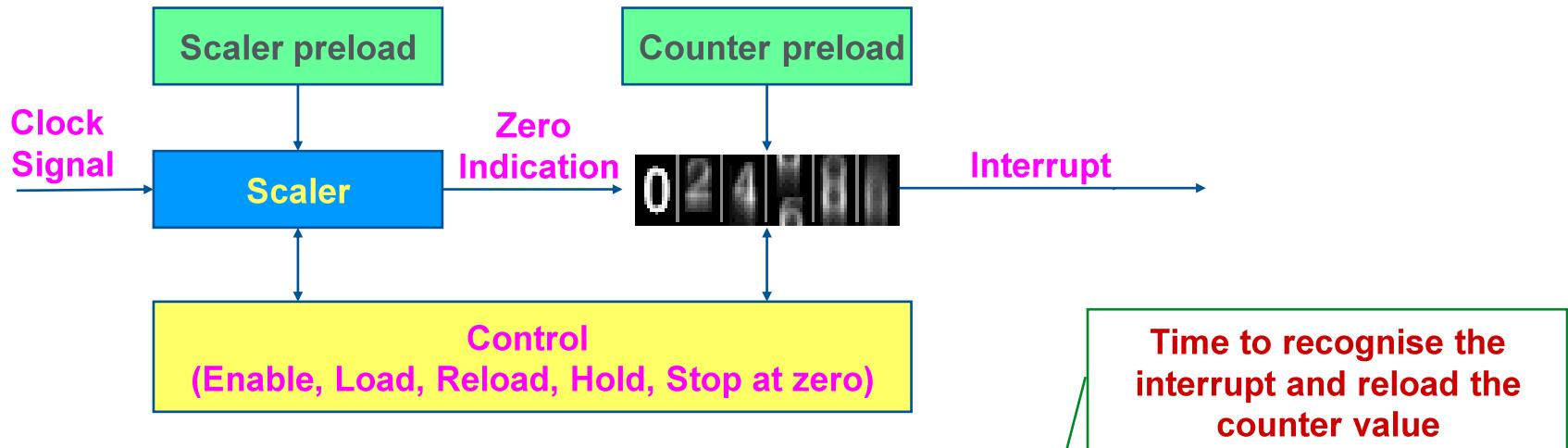
Ejercicio: caso real

Mecanismo de interrupciones de la familia de microcontroladores AVR

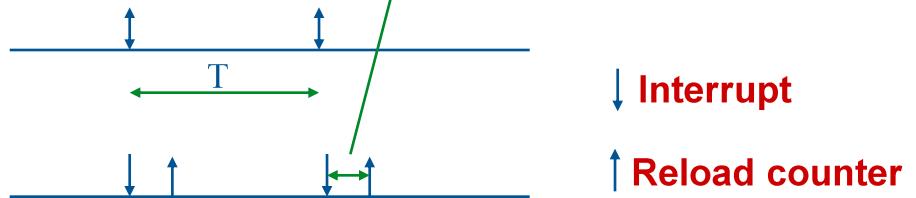
- Estudio del mecanismo de interrupciones de la familia de microcontroladores AVR de 8 bits.
- Presentes en el hardware de libre distribución Arduino.
- <http://www.avr-tutorials.com/interrupts/about-avr-8-bit-microcontrollers-interrupts>

● Dispositivos periféricos

Hardware timers



- ◆ Operational modes
 - Periodic counter
 - Interval counter



Hardware timers

- They are used by the operating systems for implementing clocks and delays.
 - UPMSat-2 OBC has two: one for real-time clock used as periodic counter and other for delays used as interval counter.
 - They are reset every time the OBC is reset.
 - There is another one in the Power Supply Unit (PSU) that holds the mission clock.
 - It starts the count just after the separation and it is connected directly to batteries.
 - It does not issue interrupt requests.

Watch dog timers

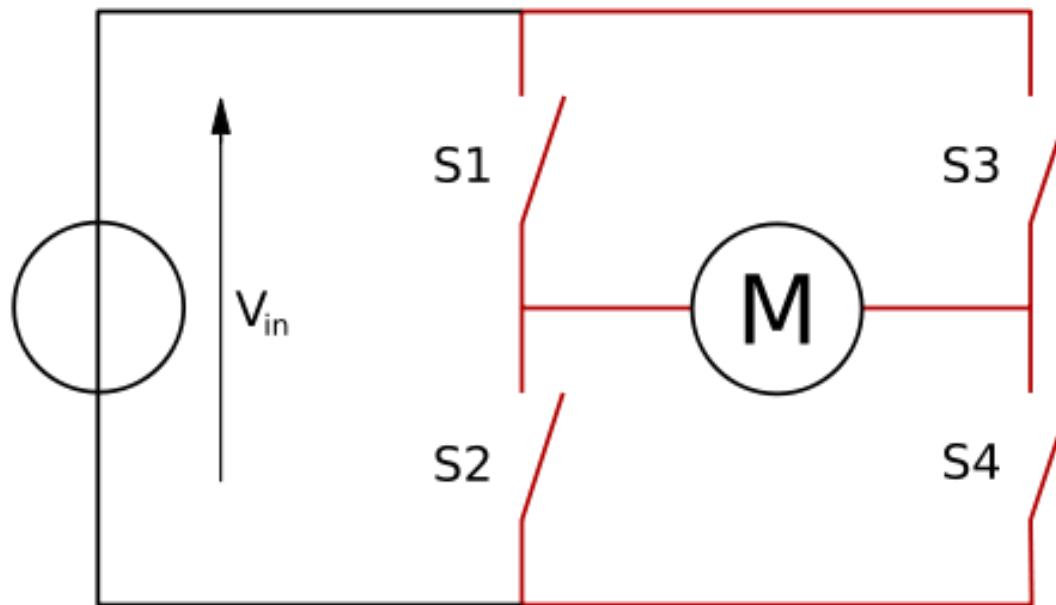
- A watch dog timer is a hardware timer connected to the processor reset line.
 - The counter has to be reloaded periodically otherwise the processor is reset.
 - They provide tolerance to software faults:
 - Infinite loop.
 - Operating system hang up.
 - ...
- UPMSat-2 has two watch dog timers:
 - One in the OBC with 10 seconds count
 - One in the PSU with 30 seconds count

Digital inputs/outputs

- It consists of registers that are allocated in the I/O address space.
 - UPMSat-2 has three 32-bits registers plus one 16-bits register. i.e. up to 112 digital inputs/outputs.
- Instructions OUT to the corresponding I/O address set the outputs with proper values.
 - For example: to activate/deactivate protections.
- Instructions IN to the corresponding I/O address read the state of inputs.
 - For example: to read battery status (high, low, critical).
- It can issue interrupt requests if the state of inputs change.

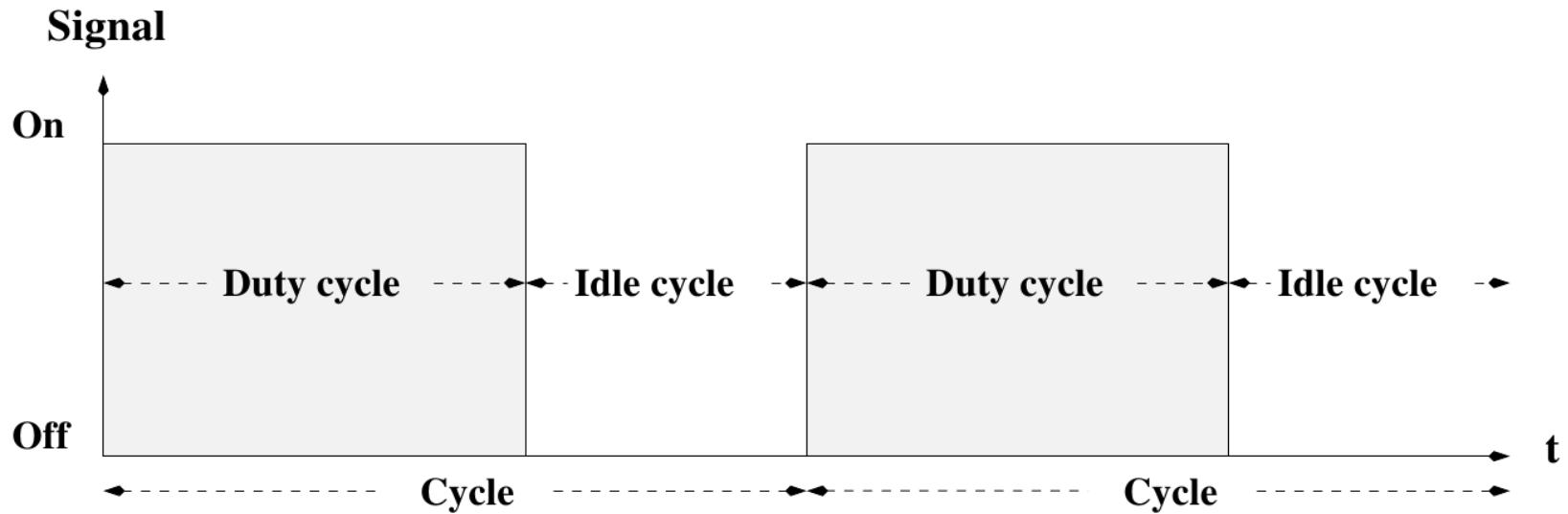
Analog output by Pulse Width Modulation (PWM)

- Some devices such as motors, lamps, magnetorques, ... can be actuated by digital outputs and H bridges.



By Cyril BUTTAY - Own work, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=854051>

PWM waveform

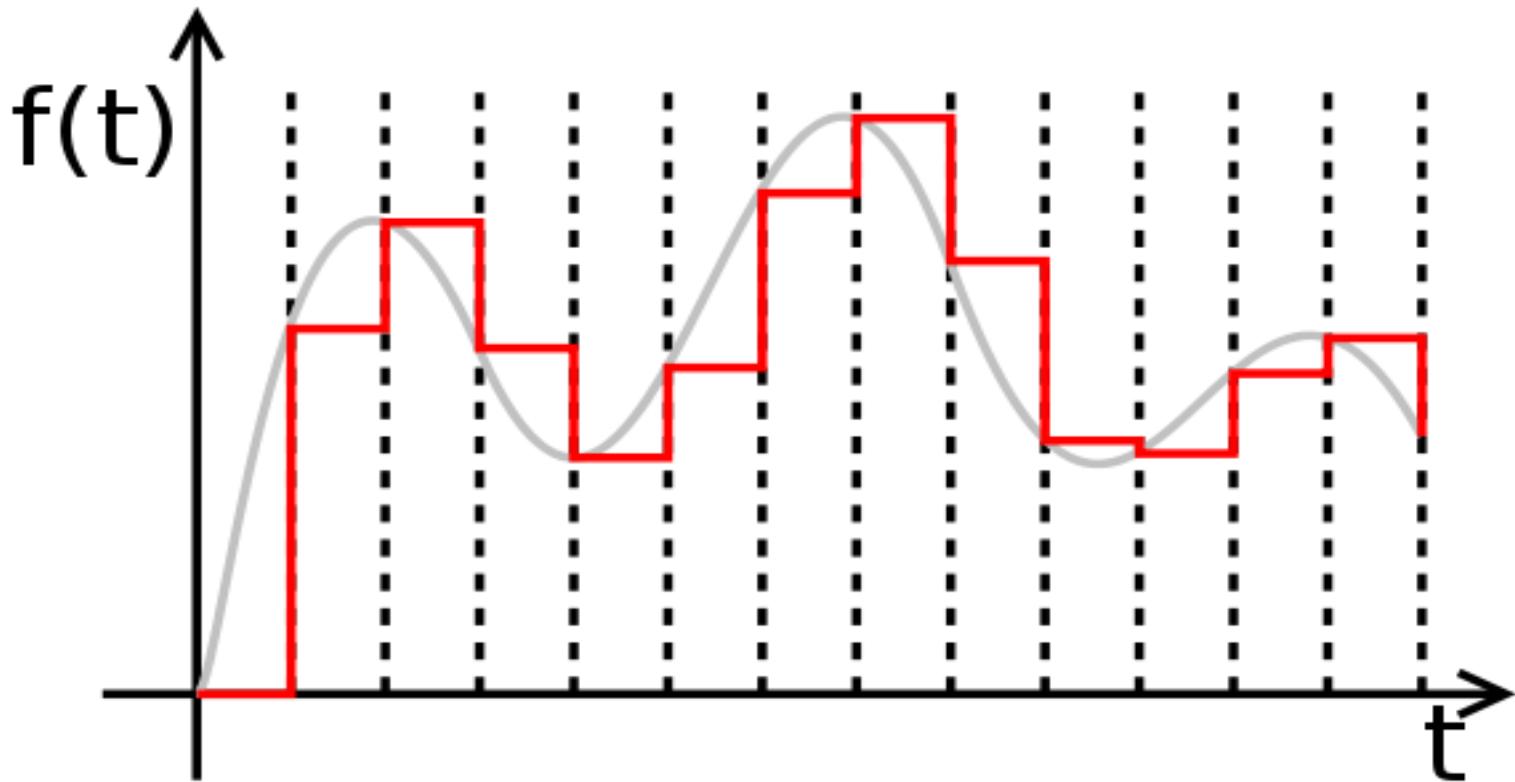


- The **larger the duty cycle, the more energy applied.**
- **Cycle period duration should suit the load dynamic.**
- Waveforms can be generated by software and digital outputs or by PWM capable hardware timers.

Digital to Analog Converter (DAC)

- Widely used peripheral device.
 - Audio and video devices use them to reproduce digital files.
- The simplest DAC type are based on PWM.
- UPMSat-2 does not have DAC. However, the 40-Channel,16-Bit, AD5370 was selected for the first version of OBC.
- Resolution: the number of output levels. 16 bit means 65536 different levels.
- Maximum sampling rate: the speed at what the DAC is able to produce outputs.
- They could have parallel or serial interface.

DAC channel output



By image source obtained from en:User:Petr.adamek (with permission) and previously saved as PD in PNG format.
touched up a little and converted to SVG by en:User:Rbj - en:Zeroorderhold.signal.svg, Public Domain,
<https://commons.wikimedia.org/w/index.php?curid=870310>

Theory of operation (AD5370 manual)

The AD5370 contains 40 DAC channels and 40 output amplifiers in a single package. The architecture of a single DAC channel consists of a 16-bit resistor-string DAC followed by an output buffer amplifier. The resistor-string section is simply a string of resistors, of equal value, from VREF to AGND. This type of architecture guarantees DAC monotonicity. The 16-bit binary digital code loaded to the DAC register determines at which node on the string the voltage is tapped off before being fed into the output amplifier. The output amplifier multiplies the DAC output voltage by 4. The nominal output span is 12 V with a 3 V reference and 20 V with a 5 V reference.

Analog to Digital Converter (ADC)

- They convert a continuous signal to digital numbers that represent signal amplitude.
- The input signal is usually a voltage.
- Therefore, transducer are needed to convert temperature, magnetic field, etc. to voltage.
- The common type successive-approximation ADC are based on DAC.

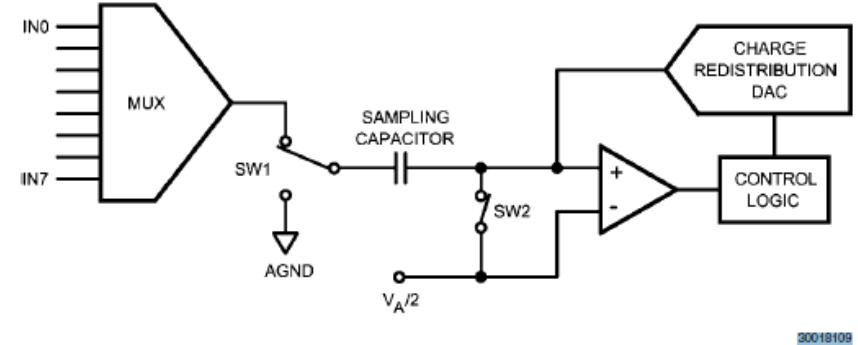


FIGURE 4. ADC128S102 in Track Mode

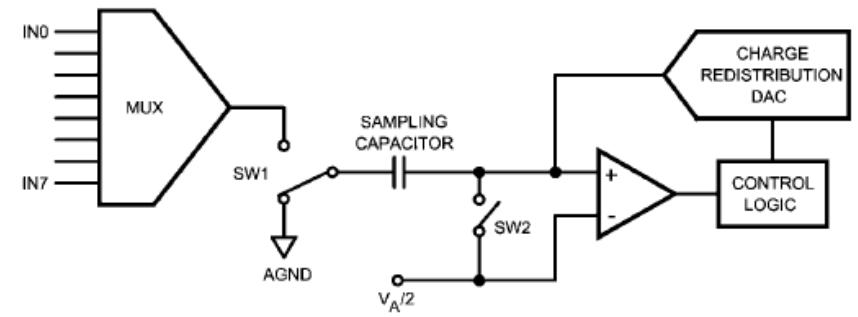


FIGURE 5. ADC128S102 in Hold Mode

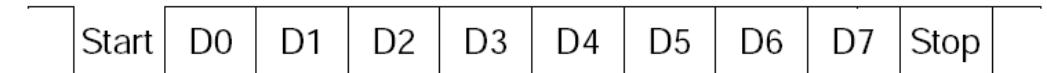
Analog to Digital Converter (ADC)

- The whole chain (transducer, signal adaptation and ADC) must be calibrated.
- UPMSat-2 has a ADC128S102QML 8-Channel, 50 kSPS to 1 MSPS, 12-Bit A/D Converter.
- It has a serial (SPI) interface.
 - Parallel ADC are usually interrupt capable.
- The ADC range (0..4095) must be translated to engineering units. For example, ADC range to teslas for magnetometers or to kelvin degrees for temperatures.
- There are up to 64 analog signals in UPMSat-2.
 - There is also a 64:8 multiplexer that allows to select among 8 set of 8 signals by means of 3 digital outputs.

Línea serie (UART)

- Tamaño de los datos: byte
- V_{transf} : 110-115.200 bits/s
- A **universal asynchronous receiver/transmitter** (usually abbreviated UART) is a type of "asynchronous receiver/transmitter", a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with other communication standards such as EIA RS-232.
(<http://en.wikipedia.org/wiki/UART>)

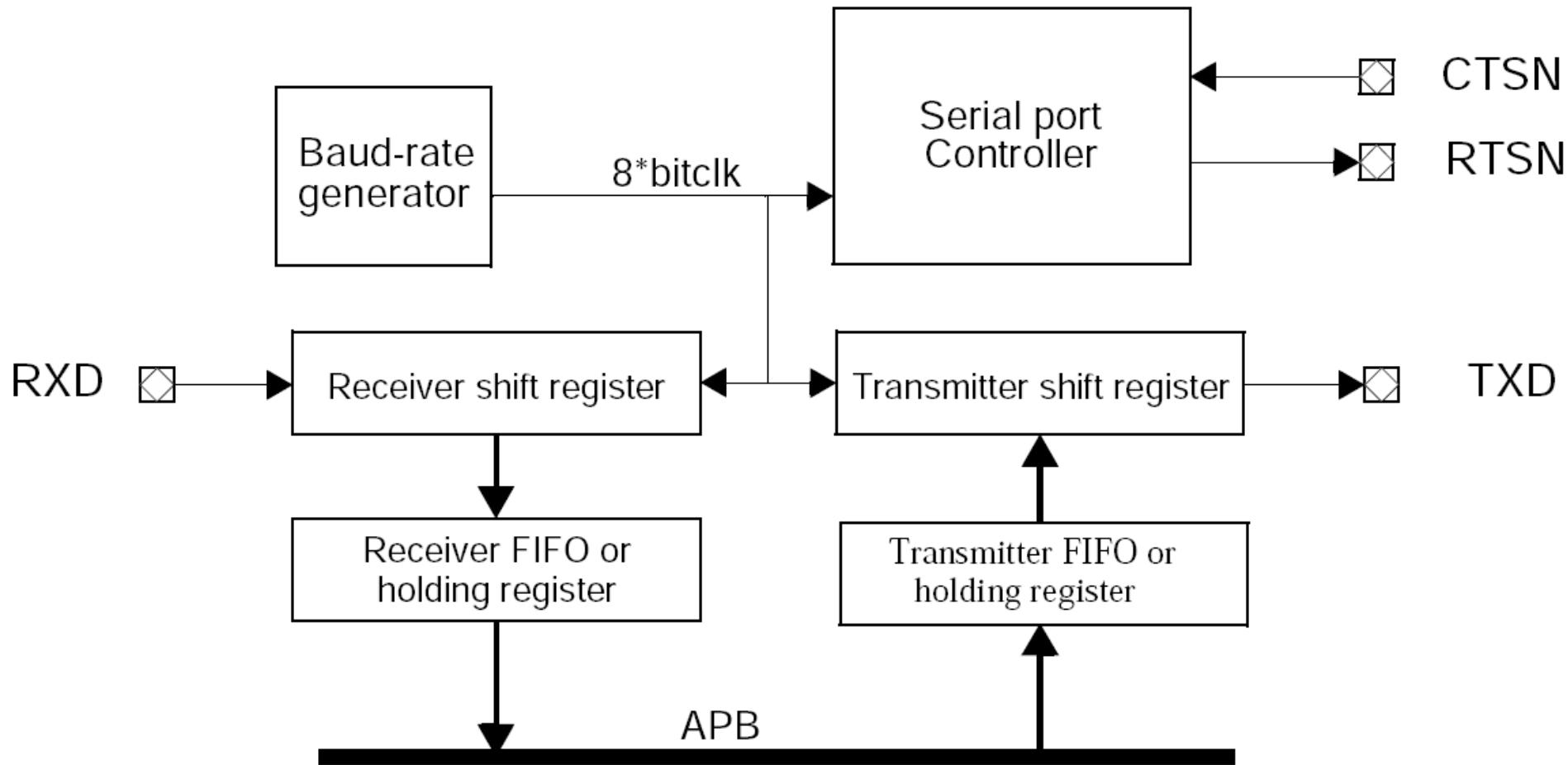
Data frame, no parity:



Data frame with parity:



Estructura UART



Control de flujo

Adecuar las velocidades de procesamiento de los dispositivos conectados.

- Hardware:
 - RTS/CTS: Request To Send/Clear To Send
 - Controla el buffer de recepción de la UART
- Software:
 - XON/XOFF: Transmission ON/OFF
 - Códigos ASCII (17 y 19)
 - Controla el buffer de la aplicación en memoria principal

Características de la comunicación

- Gestión: Simplex, Half-Duplex y Full-Duplex
- Velocidad/distancia:
 - 19.200 bits/s: 15 m
 - 2.400 bits/s: 1.000 m
- Niveles RS-232:
 - 1 lógico: -3..-15V
 - 0 lógico: +3..+15V
- RS-422: Señales en modo diferencial.
- RS-485: Conexión en bus
 - Configuración maestro/esclavo
- NMEA 0183: receptores GPS y equipos “marinos”
 - 4.800 bits/s, 8N1 y niveles RS-422

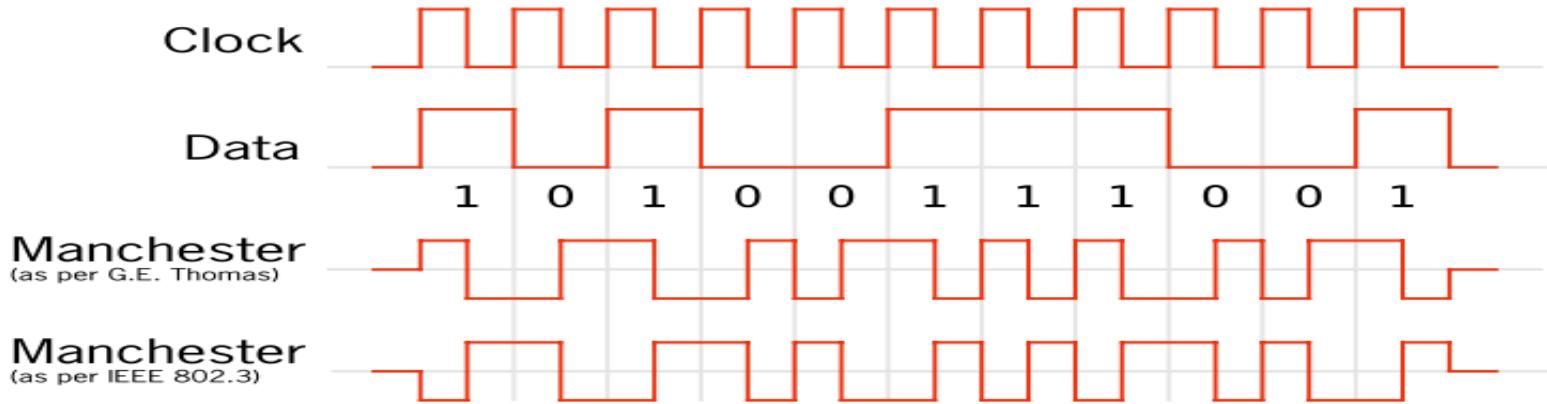
Ejemplo UPMSat-2

- En el UPMSat-2 hay 4 UARTs operadas mediante interrupciones:
 - Comunicación con la radio (RS-422).
 - Comunicación con la rueda de inercia (niveles TTL).
 - Comunicación con el computador de desarrollo (RS-422).
 - Línea auxiliar para mensajes de depuración (RS-422).
- Además existen otras 2 conexiones serie operadas mediante interrupciones:
 - I2C para comunicación con el reloj de misión.
 - SPI para comunicación con el ADC.

Ethernet

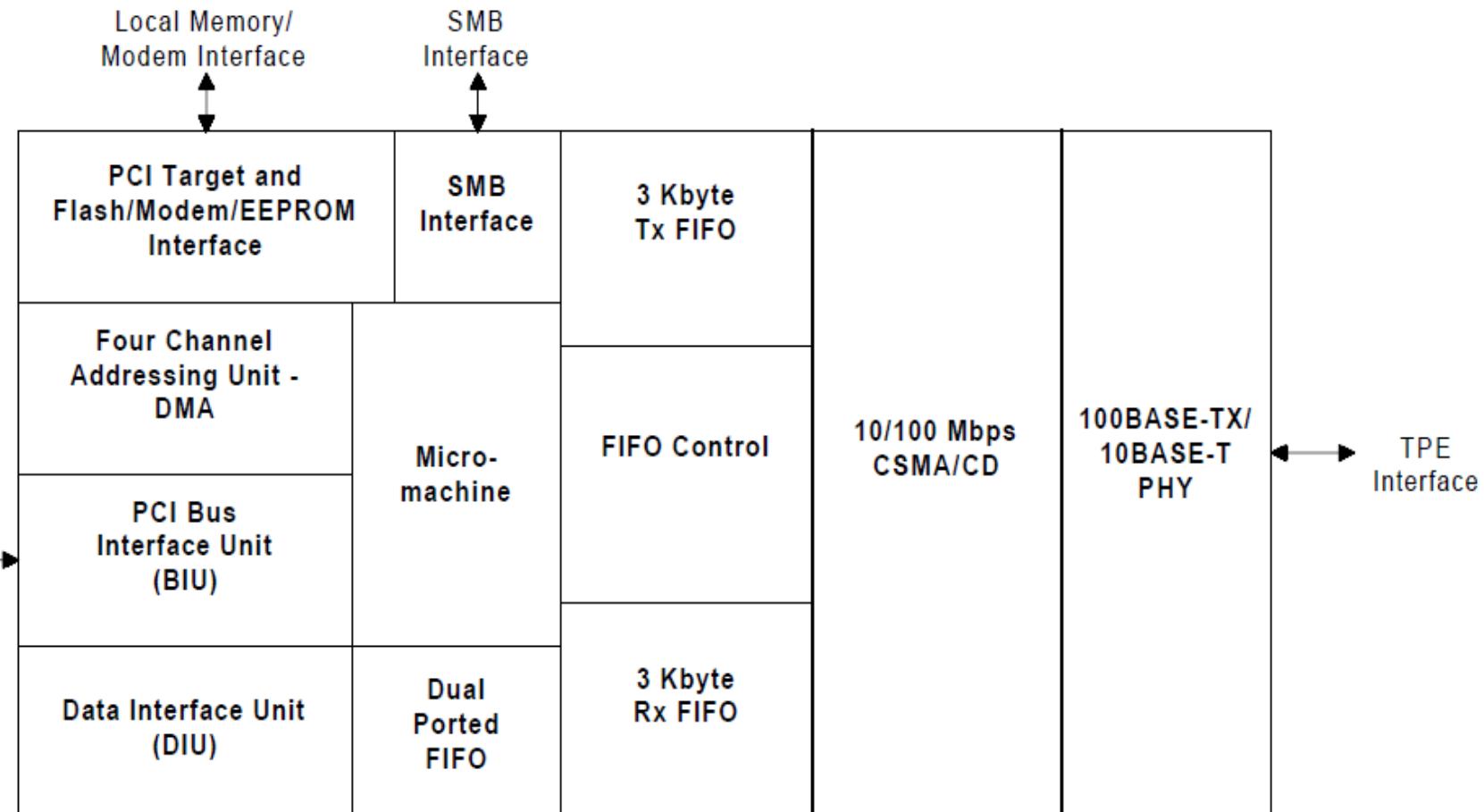
- Dispositivo de bloque
 - Habitualmente operado por DMA
- Tamaño de los bloques variable
- V_{transf} : 10-10.000 Mbps (10 Gbits/s)
- Modo de funcionamiento:
 - Codificación de datos
 - Formato de paquete
 - Control de flujo
 - Medio físico

Codificación de datos



- Modulación por desplazamiento de fase.
- Reloj autocontenido.
- Se necesita doble ancho de banda.

LANCE: Local Area Network Controller for Ethernet



Formato de paquete

802.3 Ethernet frame structure									
Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype or length	Payload	Frame check sequence (32-bit CRC)	Interframe gap	
7 octets of 10101010	1 octet of 10101011	6 octets	6 octets	(4 octets)	2 octets	46-1500 octets	4 octets	12 octets	
64-1522 octets									
72-1530 octets									
84-1542 octets									

Figura de <https://es.wikipedia.org/wiki/Ethernet>

- La dirección MAC es única para cada LANCE:
\$ifconfig
eth0 Link encap: Ethernet dirección HW 20:cf:30:27:14:78
- Información neta: 46-1500 bytes
- Información bruta (incluyendo tag): 84-1542 bytes
- IFG permite a los LANCES prepararse para la recepción del siguiente paquete.

Medio físico

- Tiene estructura de bus: múltiples dispositivos conectados al mismo medio físico.
- Originalmente cable coaxial de 9.5 mm (thicknet) 1980-85. Bus de 500 m.
- Posteriormente cable coaxial fino RG-58 de 5 mm (thinnet). Bus de 185 m.
- Actualmente los medios más comunes son par trenzado con conectores 8P8C (Rj45) y transmisión Wireless LAN por radio (WiFi).

Control de flujo

- Topología en bus con protocolo CSMA/CD:
 1. CS (Carrier sense) los LANCES reciben por la red a la espera de que esté inactiva.
 2. MA (Multiple access) uno o varios LANCES empiezan la transmisión a la vez que reciben lo que se manda por la red.
 3. CD (Collision Detect) si lo que reciben difiere de lo que están enviando quiere decir que se ha producido una colisión.
 4. Se aborta la transmisión y se espera un tiempo aleatorio para volver al paso 1. El número de reintentos es configurable por software.

Análisis

- Como consecuencia del control de flujo el tiempo de acceso al medio no está acotado.
 - El ancho de banda efectivo es muy bueno y en ofimática es prácticamente la única que se usa.
- Por lo tanto no se usa en sistemas empotrados de control.
 - Existen variantes que mediante software acotan este tiempo máximo. Ejemplo EtherCat.
- CAN Bus (Controller Area Network) se desarrolló para automoción y se usa en sistemas espaciales.
 - Control de flujo CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance).
- Para velocidades de transmisión mayores se usa SpaceWire.

Diseño e implementación

Codiseño Software/Hardware

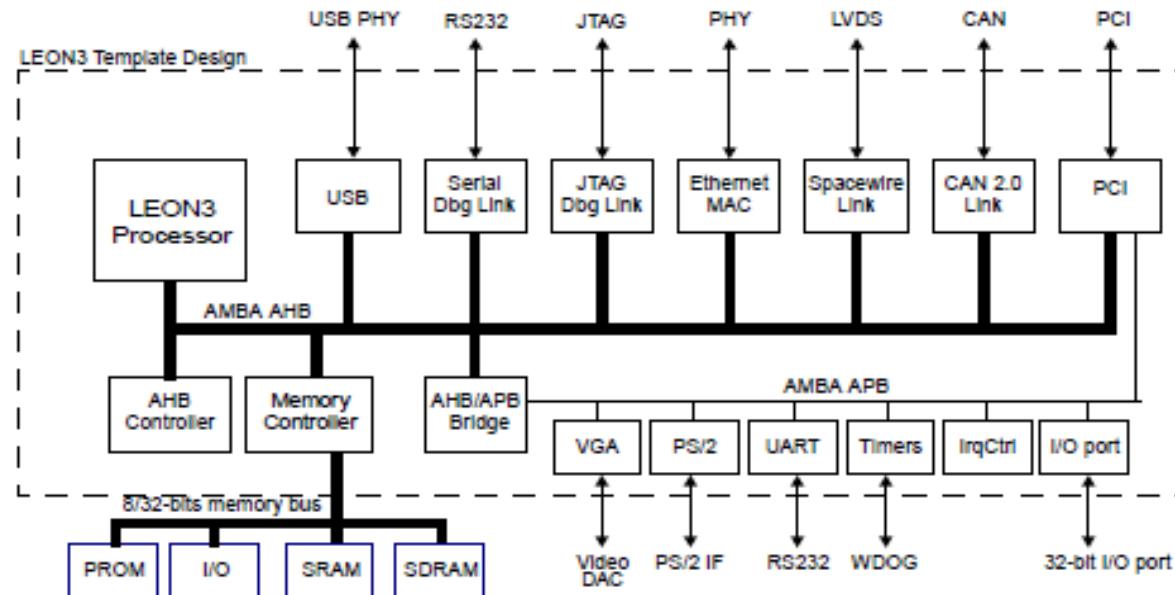
- El desarrollo “clásico” de sistemas empotrados consiste en desarrollar el software para un computador previamente seleccionado o diseñado.
- Actualmente, las plataformas de hardware configurable (ASIC, FPGA, ...) pueden integrar un gran número de componentes.
 - Procesadores, memorias, periféricos, etc.
 - Así como componentes hardware a medida.
 - Y componentes software escritos en un lenguaje de definición de hardware.
- De este modo, el software y el hardware no se diseñan por separado sino conjuntamente.

IP Core Libraries

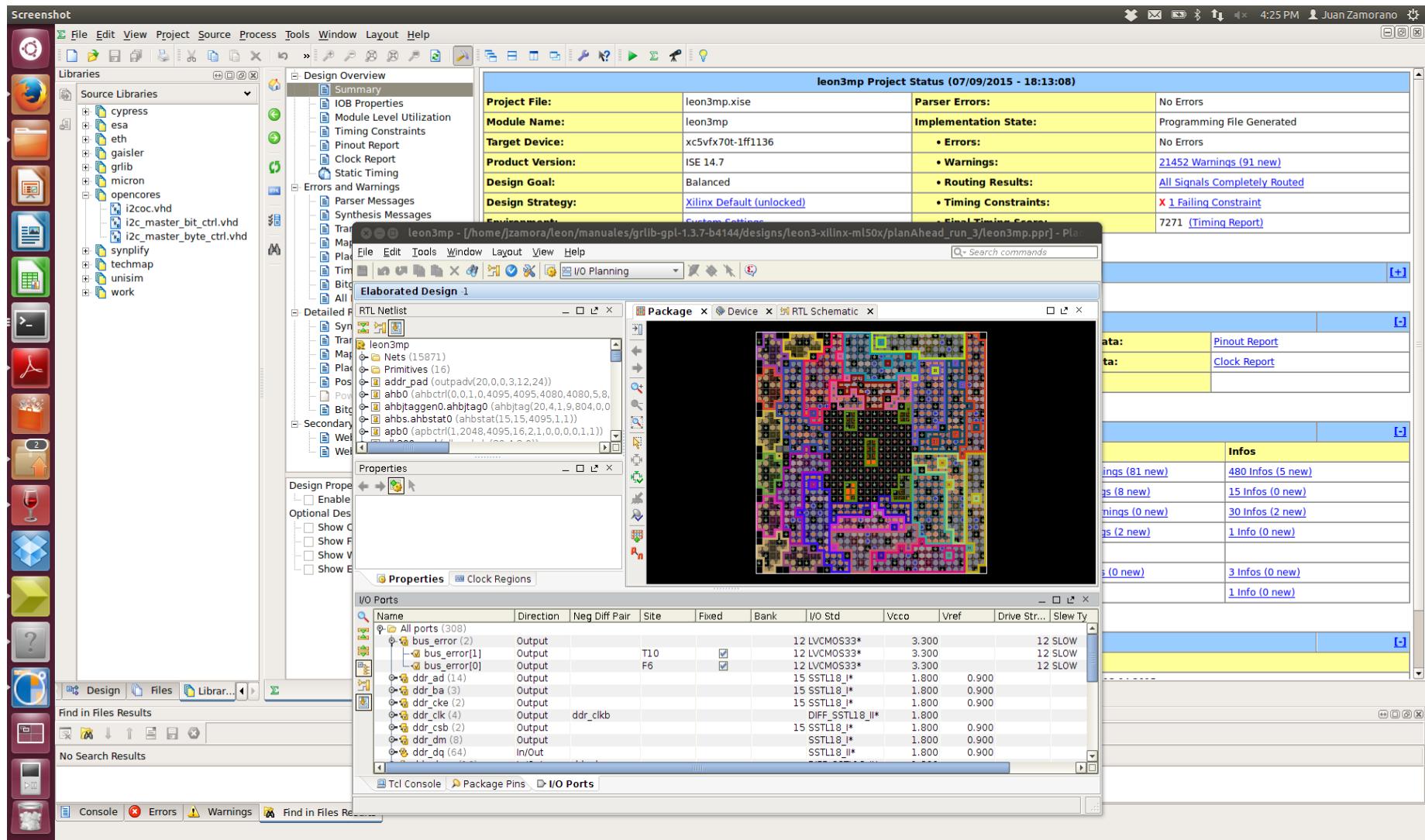
- They contain generic IP cores that can be instantiated by providing the needed parameters. e.g. bus base address, chip select line, interrupt line, etc.
- It is also needed to provide the mapping between FPGA pins and external signals of the SoC. e.g. connect clock line to the FPGA pin that is connected with the external clock.
- These files are synthesised to produce a binary file that programs logic blocks and configures interconnections of the FPGA device.
- The binary file must be recorded into the FPGA non-volatile memory. So that the FPGA is properly reconfigured at start up.

Example: GRLIB IP Core from Aeroflex Gaisler

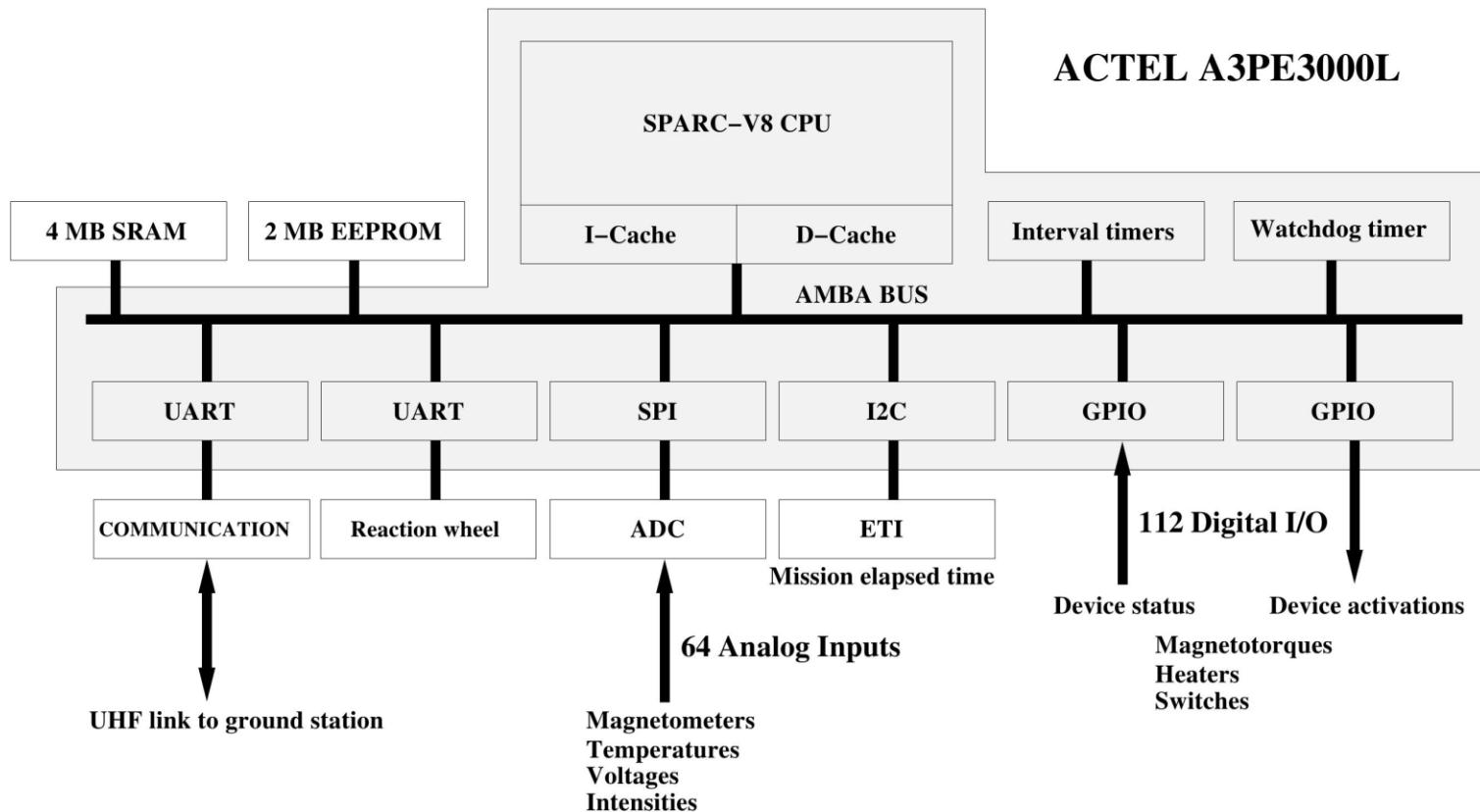
- GRLIB is organized around VHDL libraries, where each major IP (or IP vendor) is assigned a unique library name.
- The GRLIB is ‘bus-centric’, i.e. it is assumed that most of the IP cores will be connected through the AMBA-2.0 AHB/APB bus.
- The available processors are LEON3 and LEON4 that are SPARC v8 compatible.



Example of FPGA IDE



Example: UPMSAT-2 SoC of OBC



UPMSAT-2 SoC Cores

GRLIB build version: 4135

Detected frequency: 20 MHz

Component	Vendor
LEON3 SPARC V8 Processor	Cobham Gaisler
AHB Debug UART	Cobham Gaisler
LEON2 Memory Controller	European Space Agency
AHB/APB Bridge	Cobham Gaisler
LEON3 Debug Support Unit	Cobham Gaisler
Generic UART	Cobham Gaisler
Multi-processor Interrupt Ctrl.	Cobham Gaisler
Modular Timer Unit	Cobham Gaisler
SPI Controller	Cobham Gaisler
Generic UART	Cobham Gaisler
AMBA Wrapper for 0C I2C-master	Cobham Gaisler
General Purpose I/O port	Cobham Gaisler
General Purpose I/O port	Cobham Gaisler
General Purpose I/O port	Cobham Gaisler
General Purpose I/O port	Cobham Gaisler
Generic UART	Cobham Gaisler

Use command 'info sys' to print a detailed report of attached cores

grmon2> █

Conclusiones

- En base a lo aprendido en este tema, debéis elegir un computador de a bordo para la misión que habéis diseñado dentro de la asignatura Ingeniería de Sistemas y Gestión de Proyectos.
- Entrega de una memoria (jzamora@fi.upm.es).
- Componentes COTS
- Criterios:
 - Memoria necesaria: tipo y cantidad
 - Periféricos: tipo y cantidad