

Grupo ARCOS

uc3m | Universidad **Carlos III** de Madrid

Tema 4 (II) El procesador

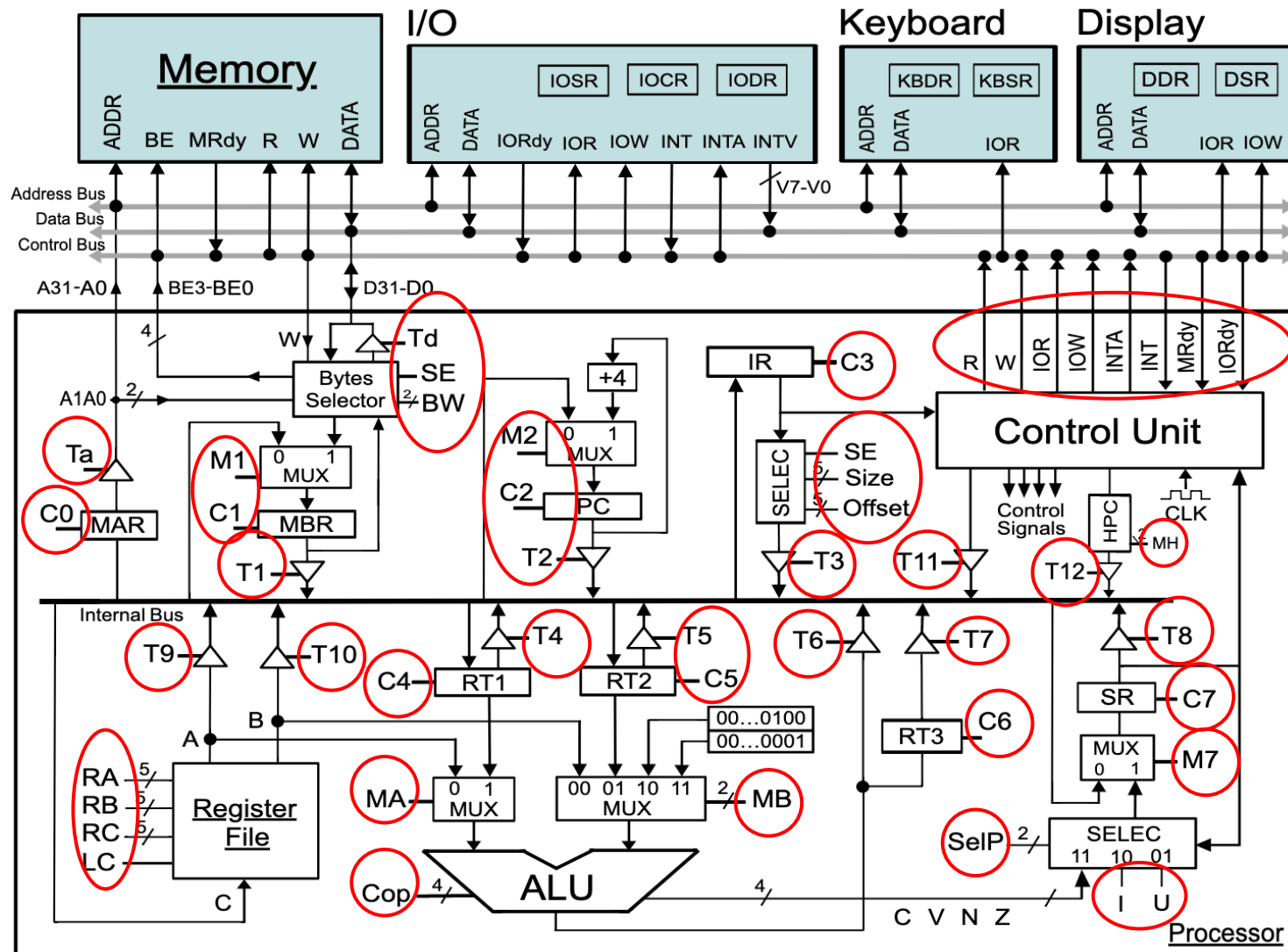
Estructura de Computadores
Grado en Ingeniería Informática

Contenidos

1. Elementos de un computador
2. Organización del procesador
3. La unidad de control
4. Ejecución de instrucciones
5. **Diseño de la unidad de control**
 - a) **Tareas en el diseño de una unidad de control**
 - b) Unidad de control almacenada
 - c) Unidad de control en WepSIM
 - d) Ejemplo de juego de instrucciones microprogramado
6. Modos de ejecución
7. Interrupciones
8. Arranque de un computador
9. Prestaciones y paralelismo

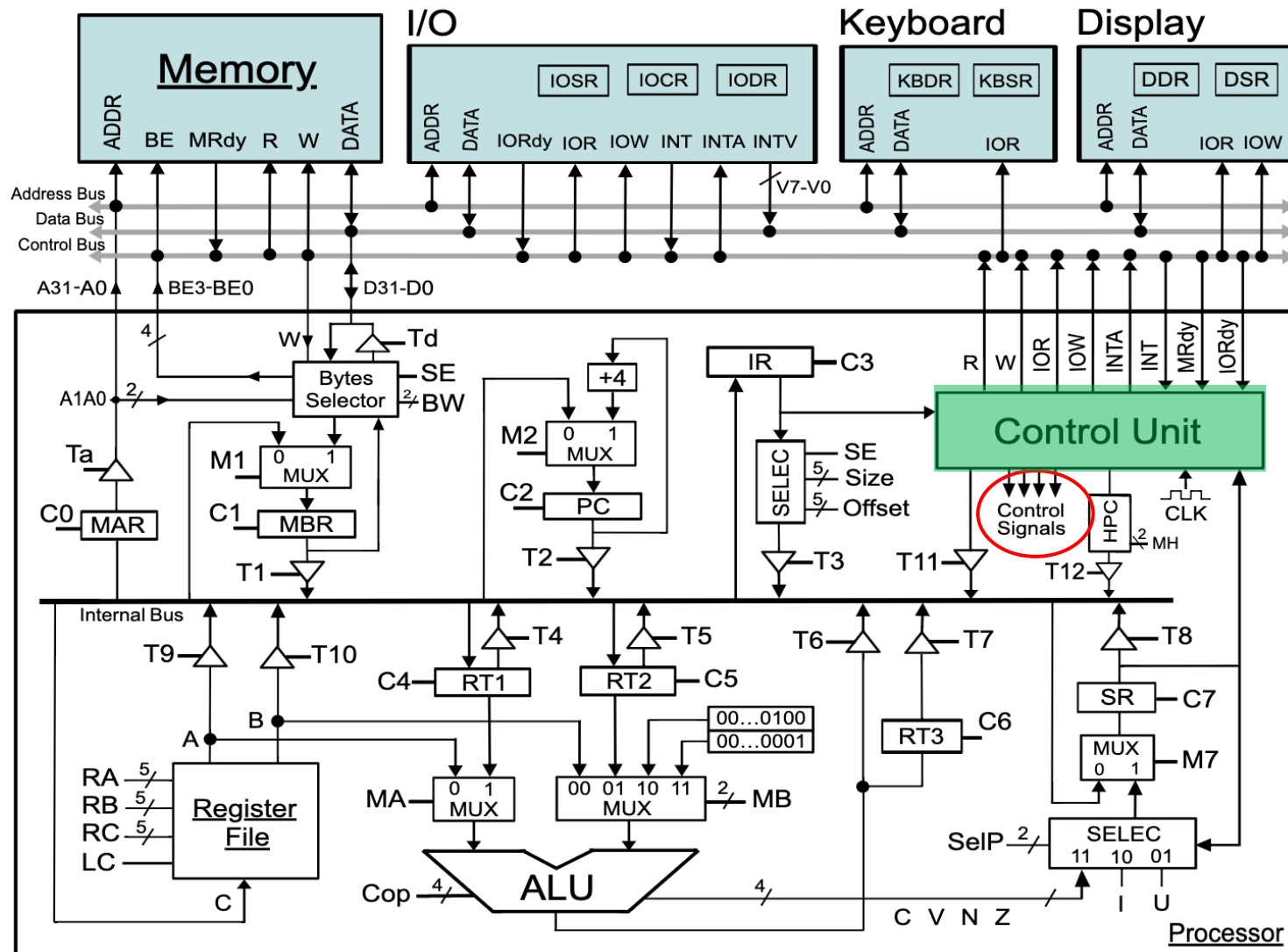
Señales de control

Recordatorio

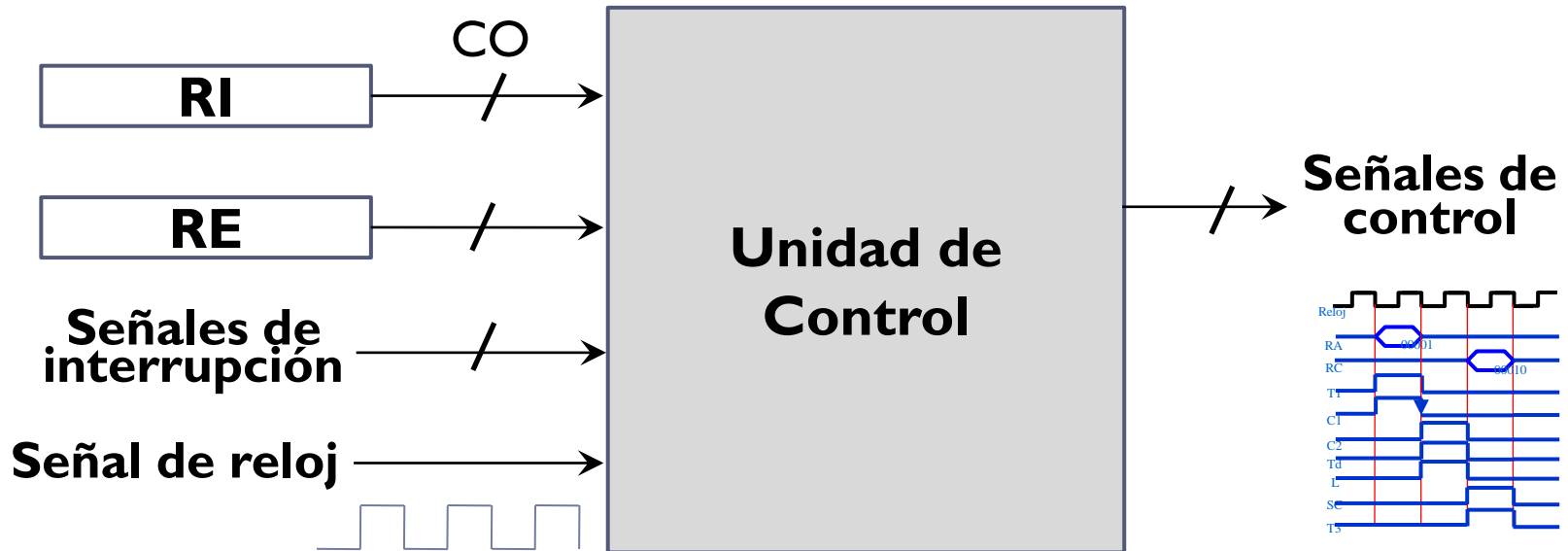
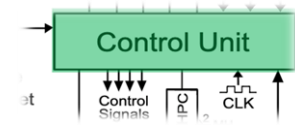


Unidad de control

Recordatorio



Unidad de control



- ▶ Cada una de las **señales de control** es **función** del valor de:
 - ▶ El contenido del **RI**
 - ▶ El contenido de **RE**
 - ▶ El **momento del tiempo**

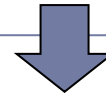
Diseño de la unidad de control

- Para cada instrucción máquina:

1. Definir el comportamiento en lenguaje de transferencia de registro (RT) en cada ciclo de reloj
2. Traducir el comportamiento a valores de cada señal de control en cada ciclo de reloj
3. Diseñar un circuito que genere el valor de cada señal de control en cada ciclo de reloj

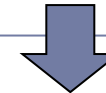
Instrucción

mv R0 R1

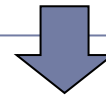
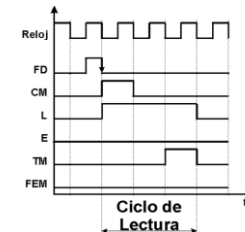


Secuencia de **operaciones elementales**

1. $RI \leftarrow [PC]$
2. $PC++$
3. decodificación
4. $R0 \leftarrow R1$



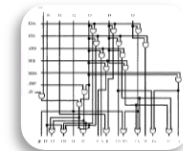
Secuencia de **señales de control** por cada operación elemental



Circuito que genera señales:

- a) Control cableado
- b) Control microprogramado

a)



b)



Ejemplo

- ▶ Diseño de una unidad de control para un juego de 4 instrucciones máquina:
- ▶ Instrucciones a considerar:
 - ▶ `add Rd, Rf:` `Rd <- Rd + Rf`
 - ▶ `lw Rd, dir:` `Rd <- MP[dir]`
 - ▶ `sw Rf, dir:` `MP[dir] <- Rf`
 - ▶ `bz R, dir:` `if (R==0) PC<- dir`

Diseño de la unidad de control

- Para cada instrucción máquina:

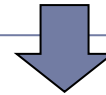
1. Definir el comportamiento en lenguaje de transferencia de registro (RT) en cada ciclo de reloj

2. Traducir el comportamiento a valores de cada señal de control en cada ciclo de reloj

3. Diseñar un circuito que genere el valor de cada señal de control en cada ciclo de reloj

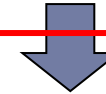
Instrucción

mv R0 R1

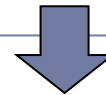
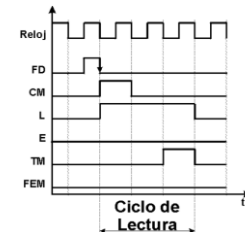


Secuencia de **operaciones elementales**

1. $RI \leftarrow [PC]$
2. $PC++$
3. decodificación
4. $R0 \leftarrow R1$



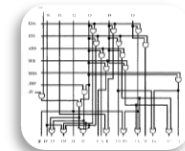
Secuencia de **señales de control** por cada operación elemental



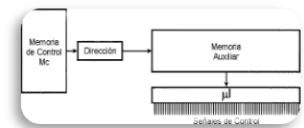
Circuito que genera señales:

- a) Control cableado
- b) Control microprogramado

a)

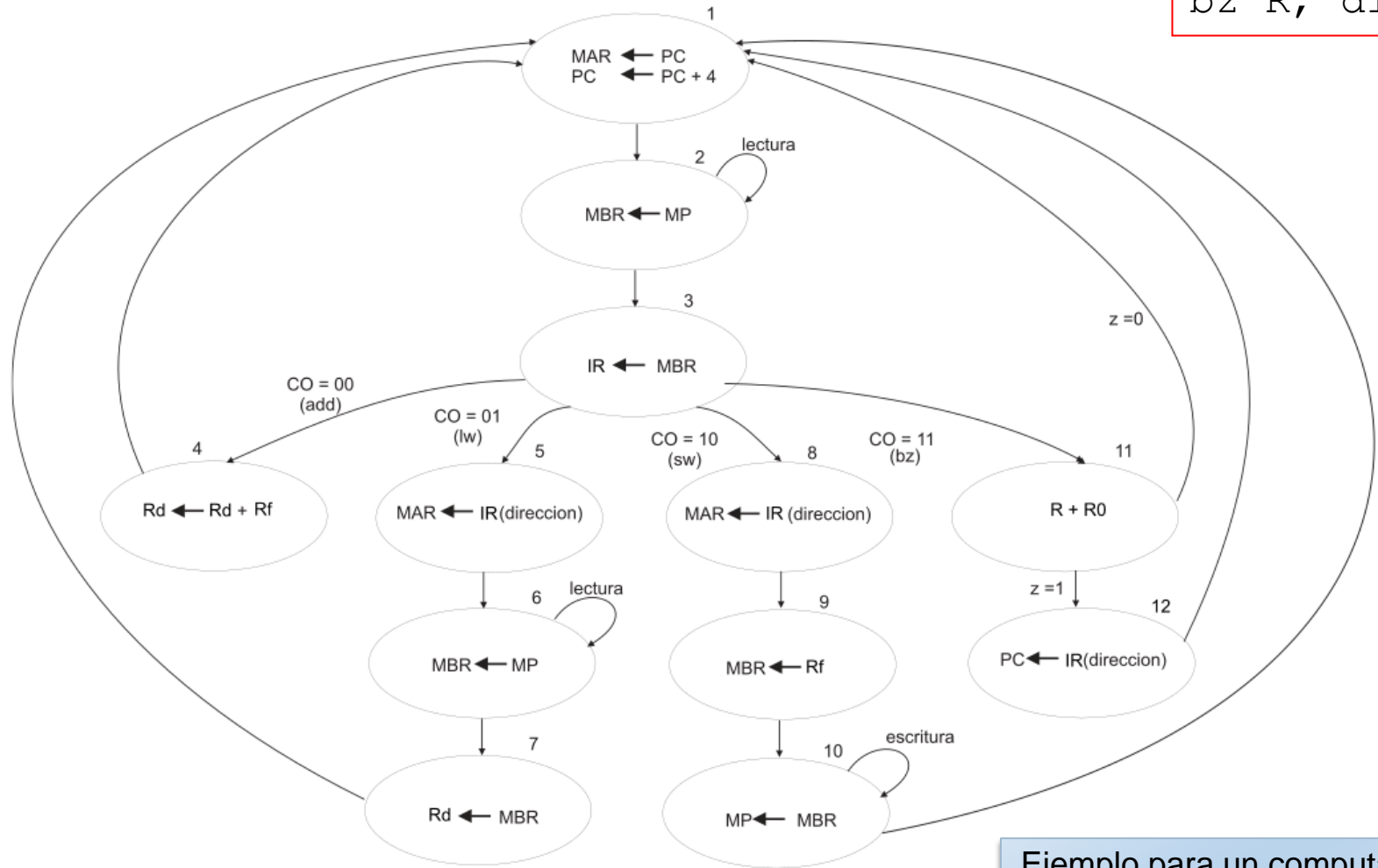


b)



Máquina de estados para ejemplo

add rd, rf
lw rd, dir
sw Rf, dir
bz R, dir



Ejemplo para un computador
con solo 4 instr. máquina

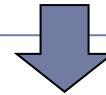
Diseño de la unidad de control

- Para cada instrucción máquina:

1. Definir el comportamiento en lenguaje de transferencia de registro (RT) en cada ciclo de reloj
2. Traducir el comportamiento a valores de cada señal de control en cada ciclo de reloj
3. Diseñar un circuito que genere el valor de cada señal de control en cada ciclo de reloj

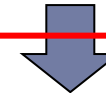
Instrucción

mv R0 R1

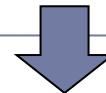
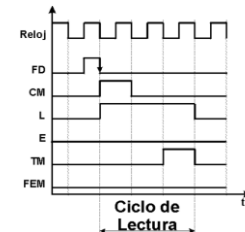


Secuencia de **operaciones elementales**

1. $RI \leftarrow [PC]$
2. $PC++$
3. decodificación
4. $R0 \leftarrow R1$

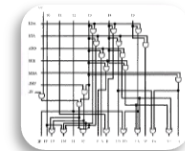


Secuencia de **señales de control** por cada operación elemental

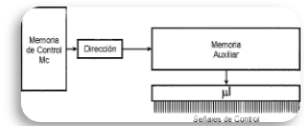


Circuito que genera señales:
a) Control cableado
b) Control microprogramado

a)



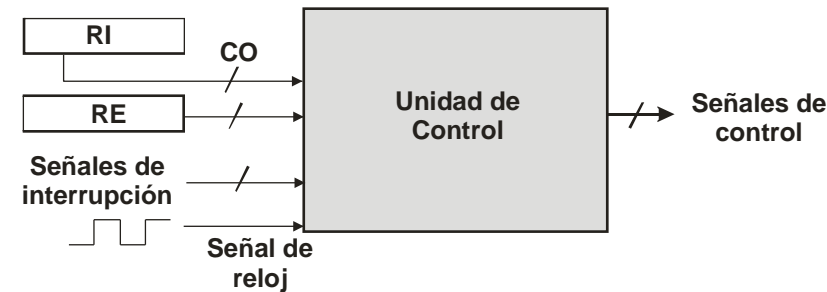
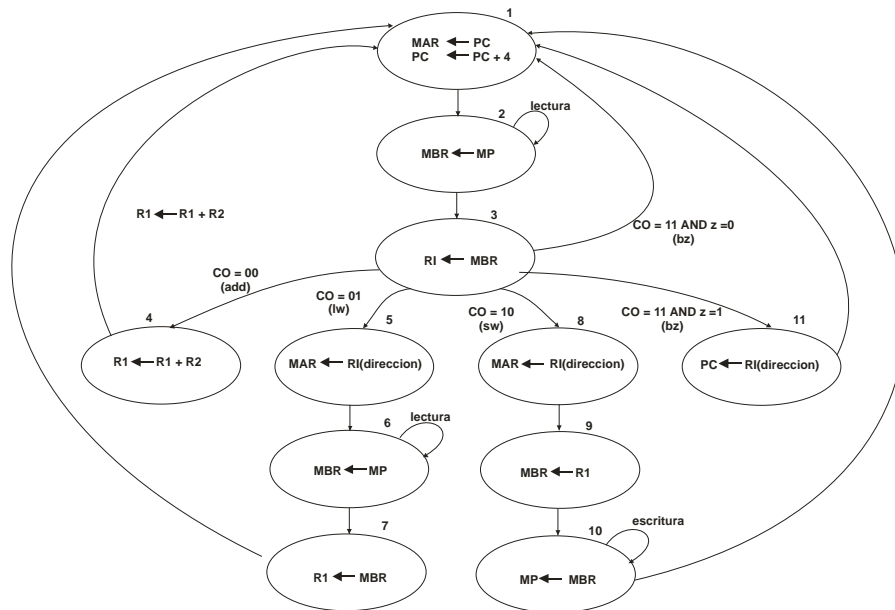
b)



Técnicas de control

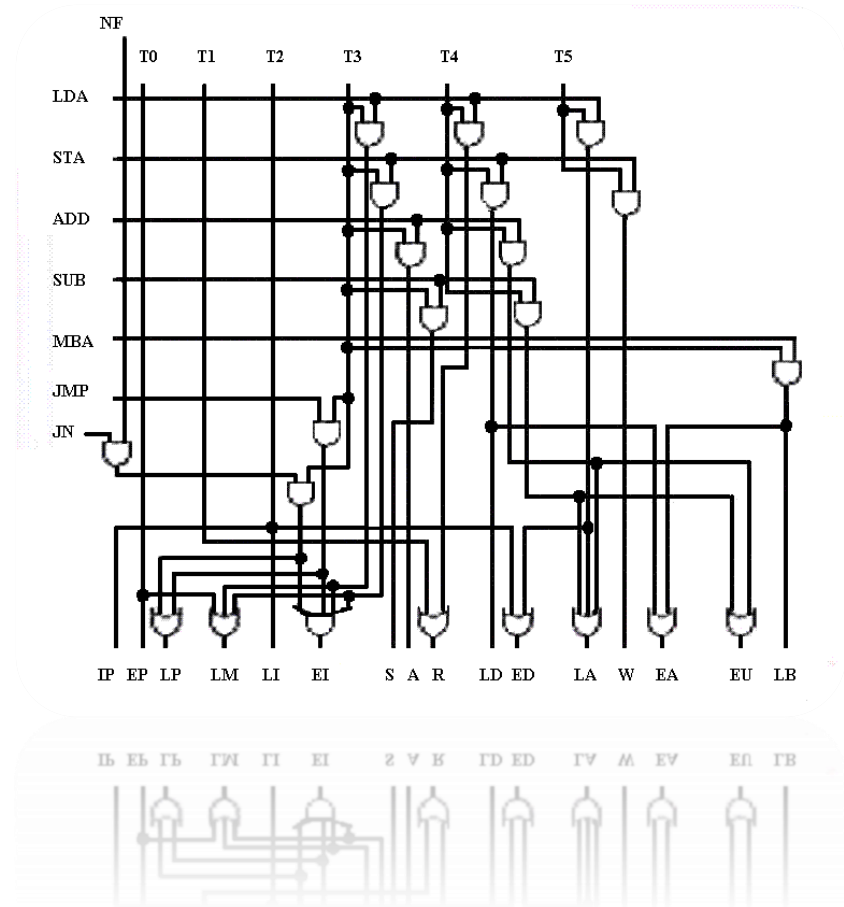
► Dos técnicas de diseñar y construir una unidad de control:

- a) Lógica cableada
- b) Lógica almacenada (microprogramación)



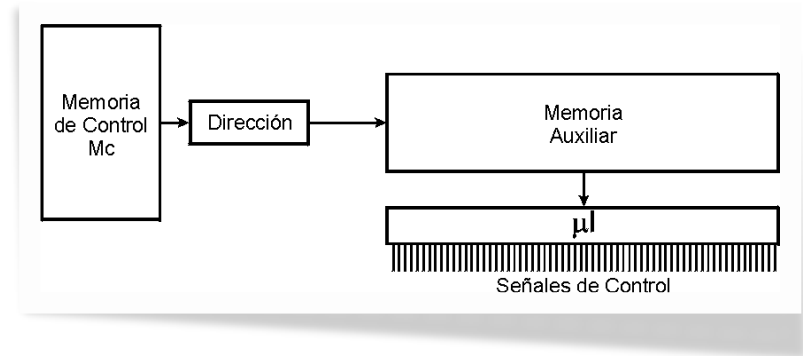
Unidad de control cableada

- ▶ Construcción mediante puertas lógicas, siguiendo los métodos de diseño lógico.
- ▶ Características:
 - ▶ Laborioso y costoso el diseño y puesta a punto del circuito
 - ▶ Difícil de modificar:
 - ▶ rediseño completo.
 - ▶ Muy rápida (usado en computadores RISC)



Unidad de control almacenada. Microprogramación

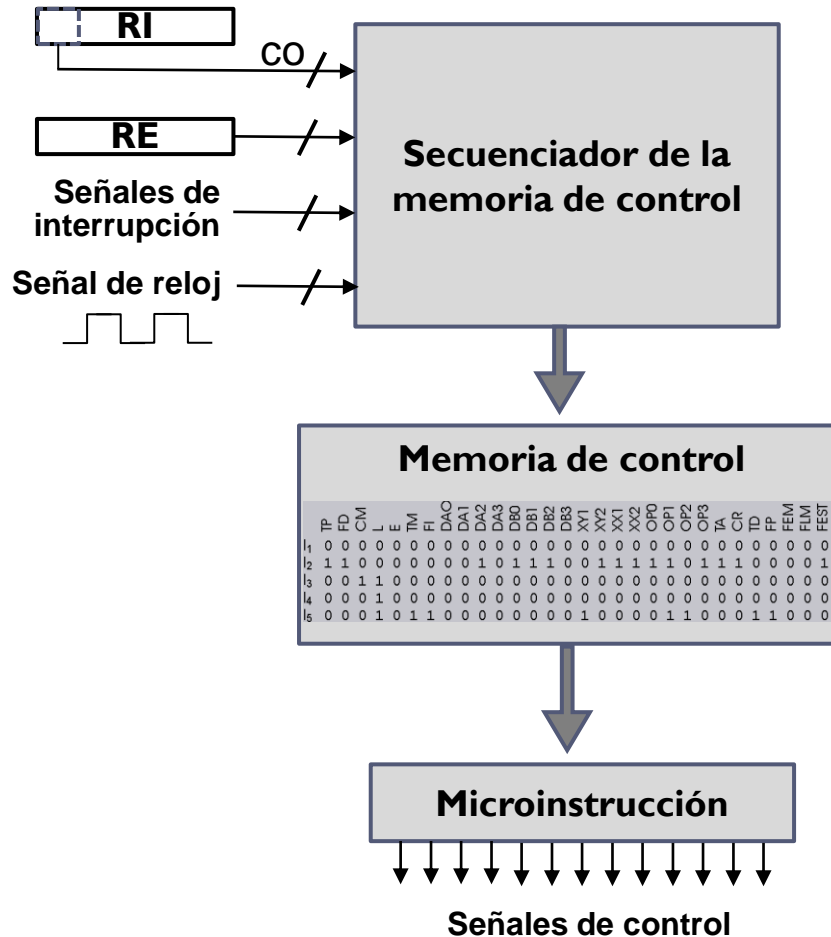
- ▶ Idea básica:
Emplear una memoria (**memoria de control**) donde almacenar las señales de cada ciclo de cada instrucción.
- ▶ Características:
 - ▶ Fácil modificación
 - ▶ Actualización, ampliación, etc..
 - ▶ Ej.: Ciertas consolas, *routers*, etc.
 - ▶ Fácil tener instrucciones complejas
 - ▶ Ej.: Rutinas de diagnóstico, etc.
 - ▶ Fácil tener varios juegos de instrucciones
 - ▶ Se pueden emular otros computadores.
 - ▶ HW simple \Rightarrow difícil microcódigo



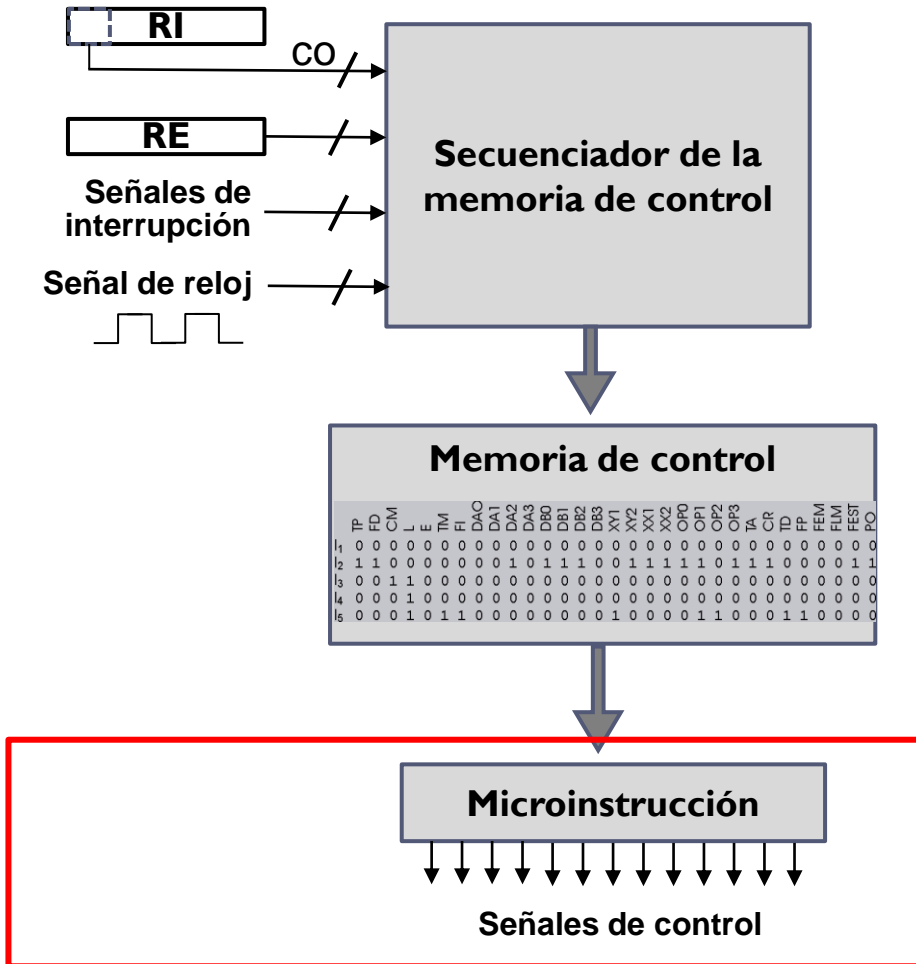
Contenidos

1. Elementos de un computador
2. Organización del procesador
3. La unidad de control
4. Ejecución de instrucciones
5. **Diseño de la unidad de control**
 - a) Tareas en el diseño de una unidad de control
 - b) **Unidad de control almacenada**
 - c) Unidad de control en WepSIM
 - d) Ejemplo de juego de instrucciones microprogramado
6. Modos de ejecución
7. Interrupciones
8. Arranque de un computador
9. Prestaciones y paralelismo

Estructura general de una unidad de control microprogramada



Estructura general de una unidad de control microprogramada



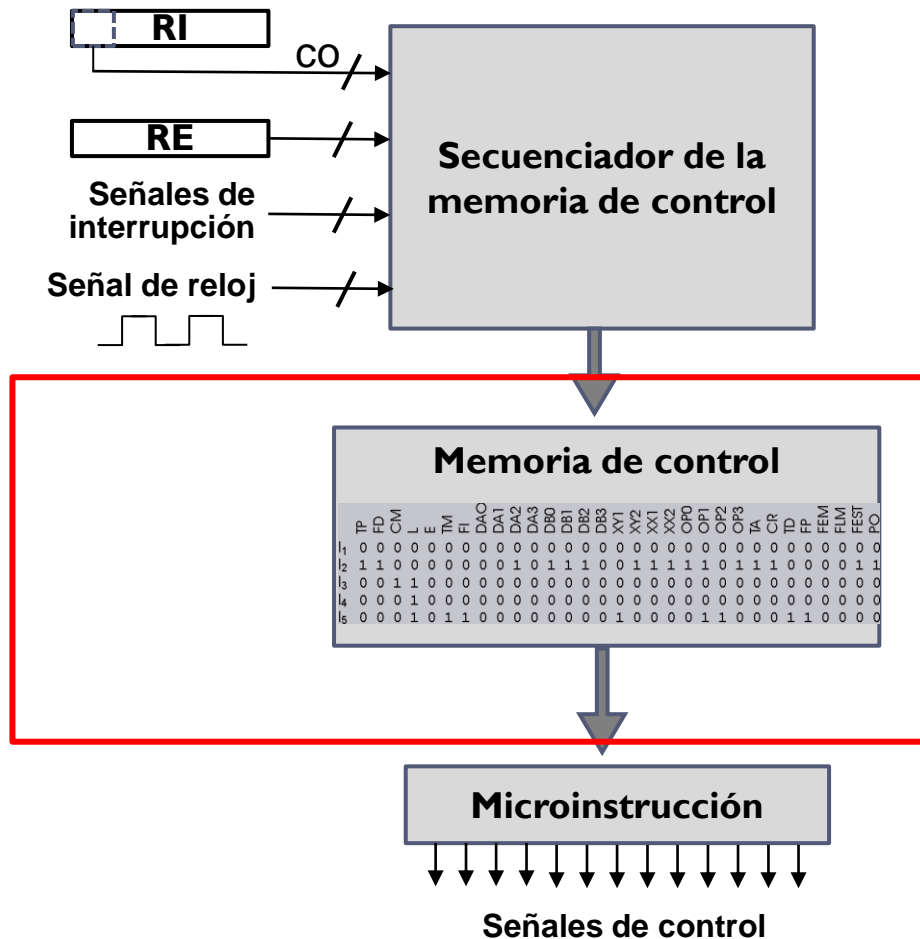
Formato de las microinstrucciones

- ▶ **Formato de la microinstrucción:**
especifica el n° de bits y el significado de cada uno de ellos.



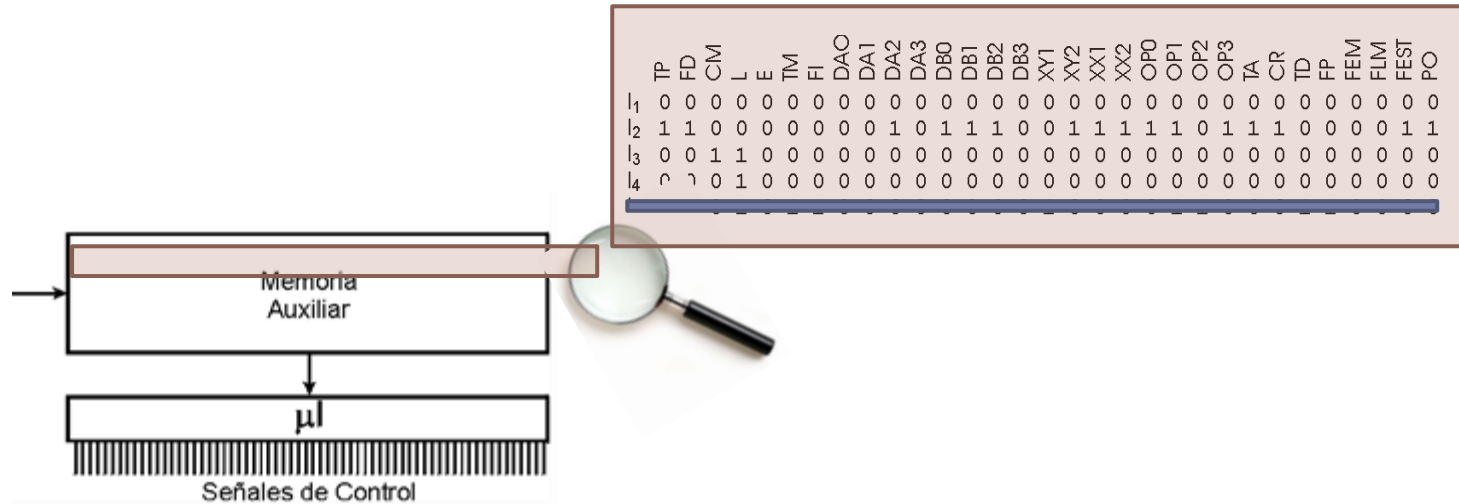
- ▶ Las señales se agrupan por **campos**:
 - ▶ Señales triestado de acceso a bus
 - ▶ Señales de gobierno de la ALU
 - ▶ Señales de gobierno del banco de registros
 - ▶ Señales de gobierno de la memoria
 - ▶ Señales de control de los multiplexores

Estructura general de una unidad de control microprogramada



Unidad de control almacenada.

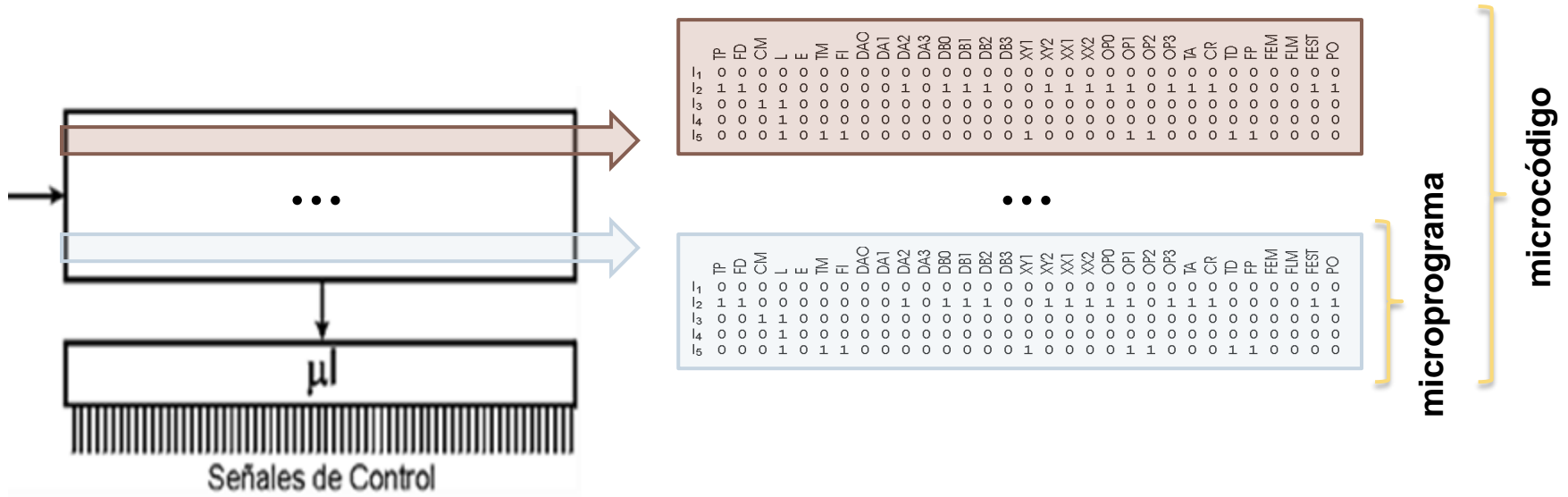
Microinstrucciones



- ▶ **Microinstrucción:** A cada palabra que define el valor de cada señal de control en un ciclo de una instrucción/fetch/CRI
- ▶ **Las microinstrucciones**
 - ▶ tienen un bit por cada señal de control.
 - ▶ cadena de 1's y 0's que representa el estado de cada señal de control durante un período de una instrucción.

Unidad de control almacenada.

Microprograma y microcódigo

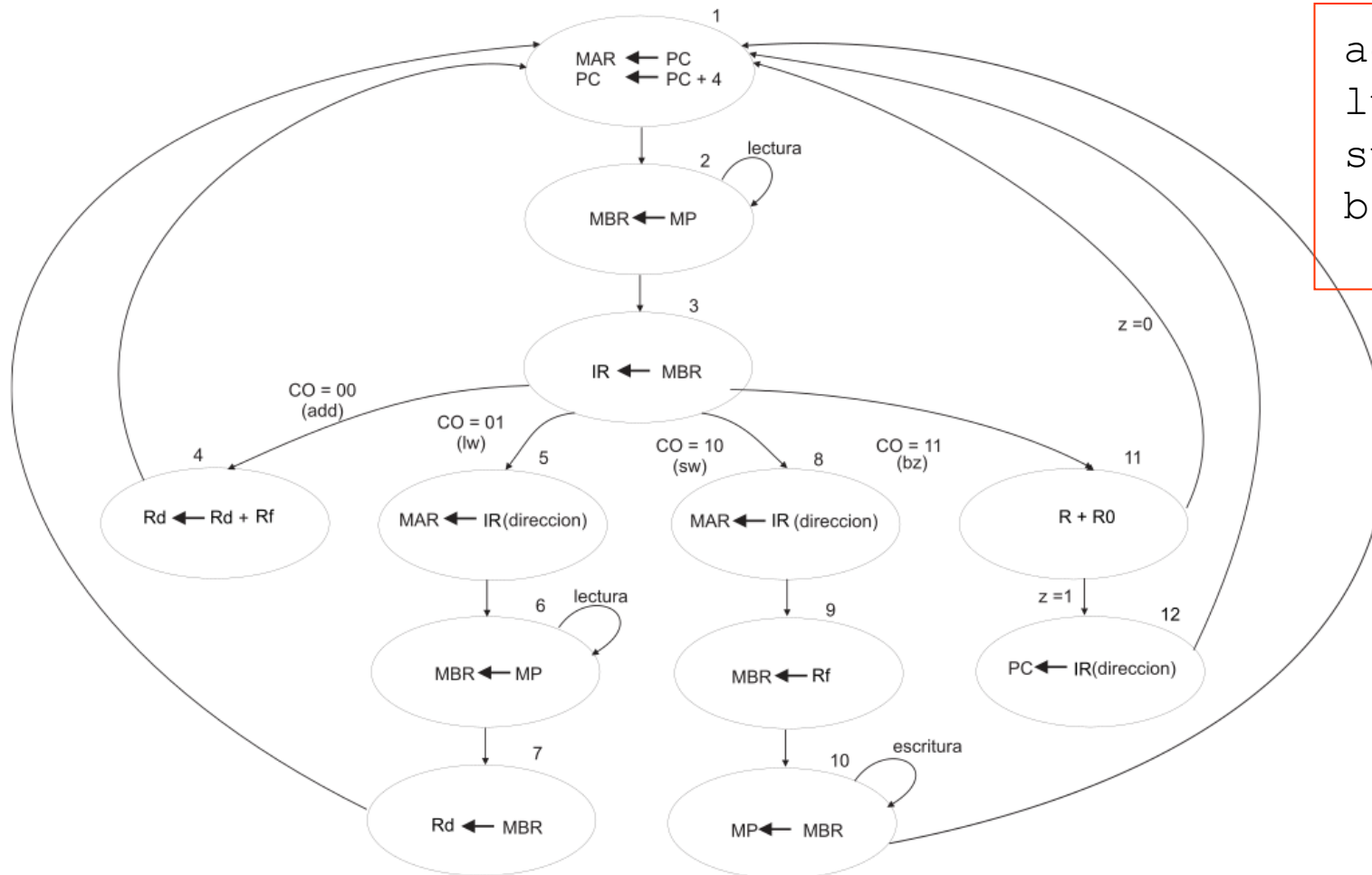


- ▶ **Microprograma:** conjunto ordenado de microinstrucciones, que representan el cronograma de una instrucción máquina.
- ▶ **Microcódigo:** conjunto de los microprogramas de una máquina.

Ejemplo: Máquina de estados

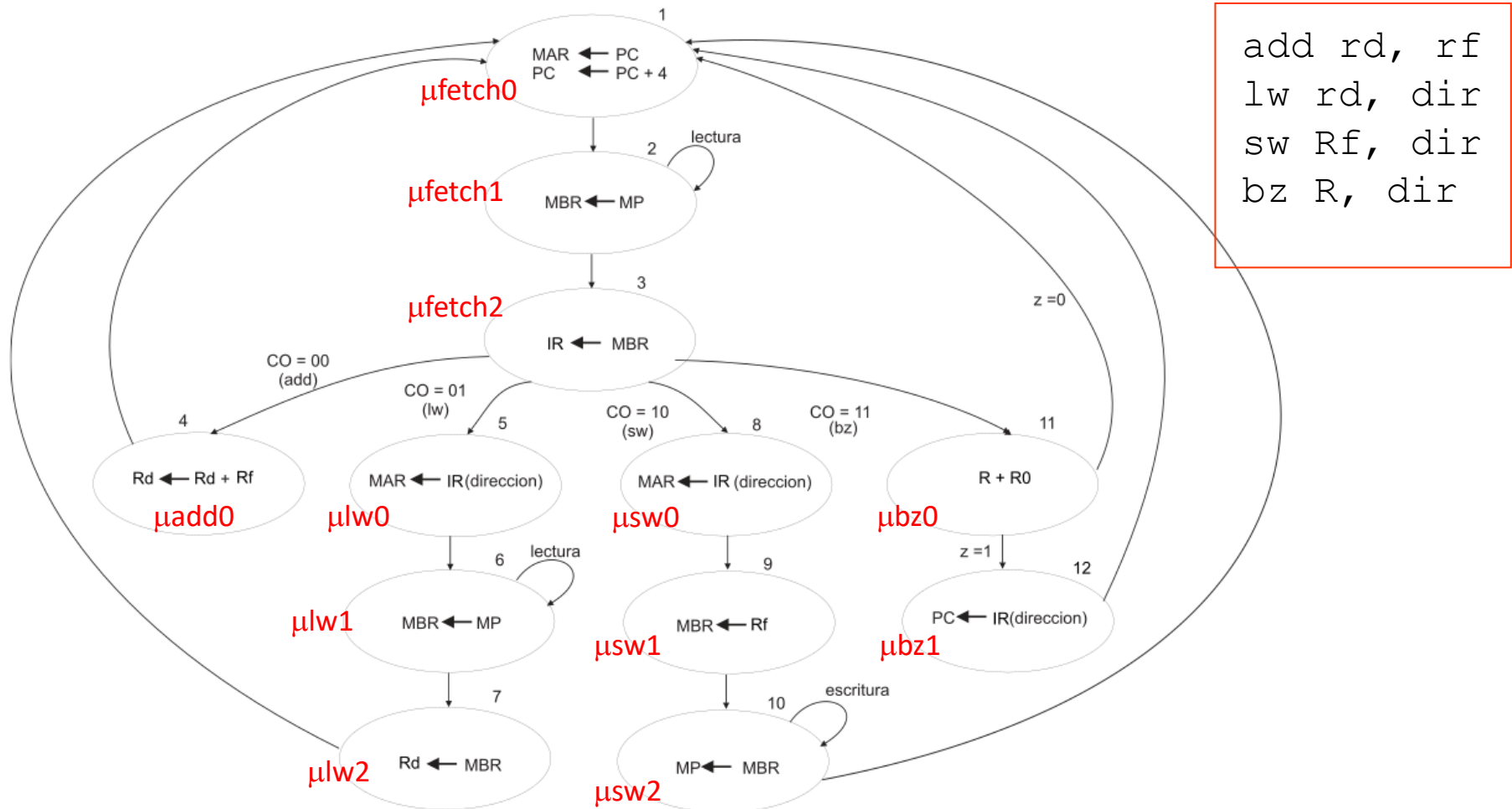
Ejemplo para un computador
con solo 4 instruc. máquina

```
add rd, rf
lw rd, dir
sw Rf, dir
bz R, dir
```



Ejemplo: microinstrucciones asociadas

Ejemplo para un computador con solo 4 instruc. máquina

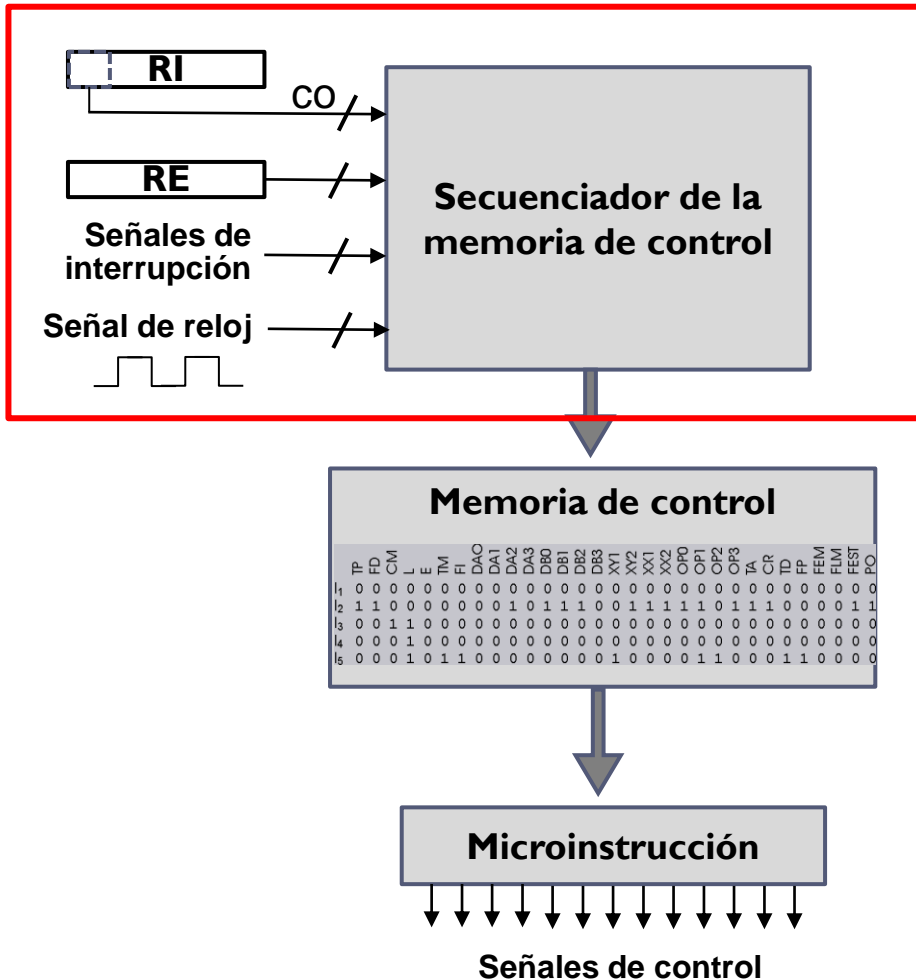


Ejemplo: Microcódigo

```
add r1, r2
lw r1, dir
bz dir
sw r1
```

	C0	C1	C2	C3	C4	C5	C6	C7	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	LE	MA	MB1	MB0	M1	M2	M7	R	W	Ta	Td		
μfetch0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	fetch
μfetch1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0		
μfetch2	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
μadd0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	add
μlw0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	lw
μlw1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0		
μlw2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
μsw0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	sw
μsw1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	
μsw3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1		
μbz0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	bz
μbz1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Estructura general de una unidad de control microprogramada



Contenido de la memoria de control



- ▶ **FETCH: traer sig. Instrucción**
 - ▶ Ciclo Reconocimiento Int.
 - ▶ $IR \leftarrow \text{Mem}[\text{PC}], \text{PC}++$, salto-a-C.O.
- ▶ **Microprograma:**
uno por instrucción de ensamblador
 - ▶ Traer resto de operandos (si hay)
 - ▶ Actualizar PC en caso de más operandos
 - ▶ Realizar la instrucción
 - ▶ Salto a FETCH

Estructura de la unidad de control microprogramada

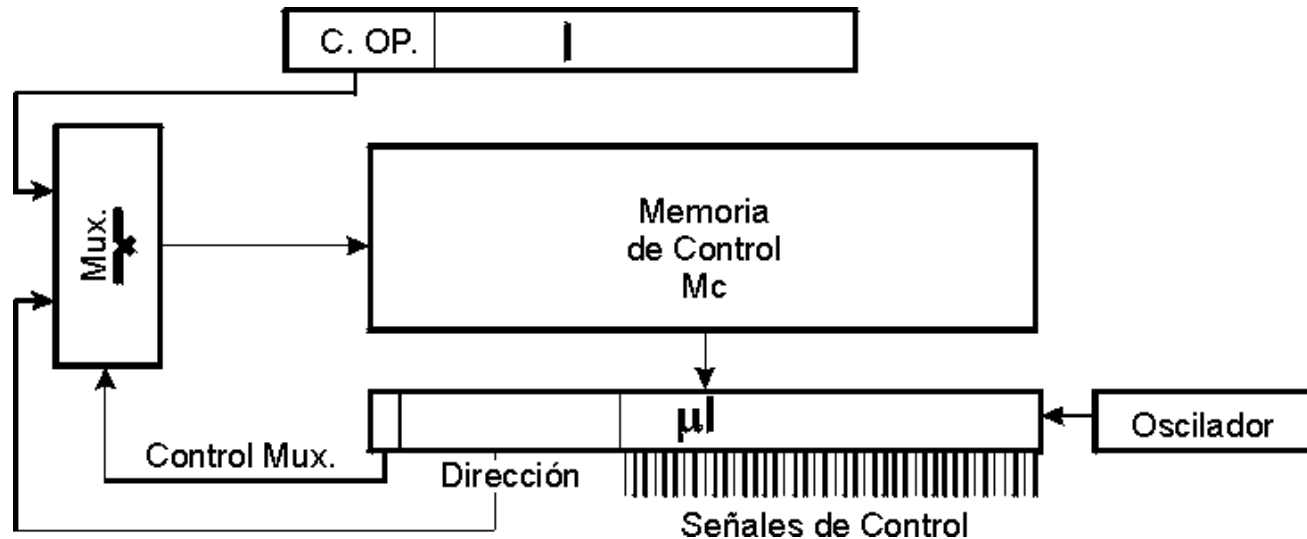
► Tres condiciones básicas:

1. Memoria de control suficiente para almacenar todos los microprogramas correspondientes a todas las instrucciones.
2. Procedimiento para asociar a cada instrucción su microprograma
 - Procedimiento que convierta el código de operación de la instrucción en la dirección de la memoria de control donde empieza su microprograma.
3. Mecanismo de secuenciación para ir leyendo las sucesivas microinstrucciones, y para bifurcar a otro microprograma cuando termina el que se está ejecutando.

► Dos alternativas:

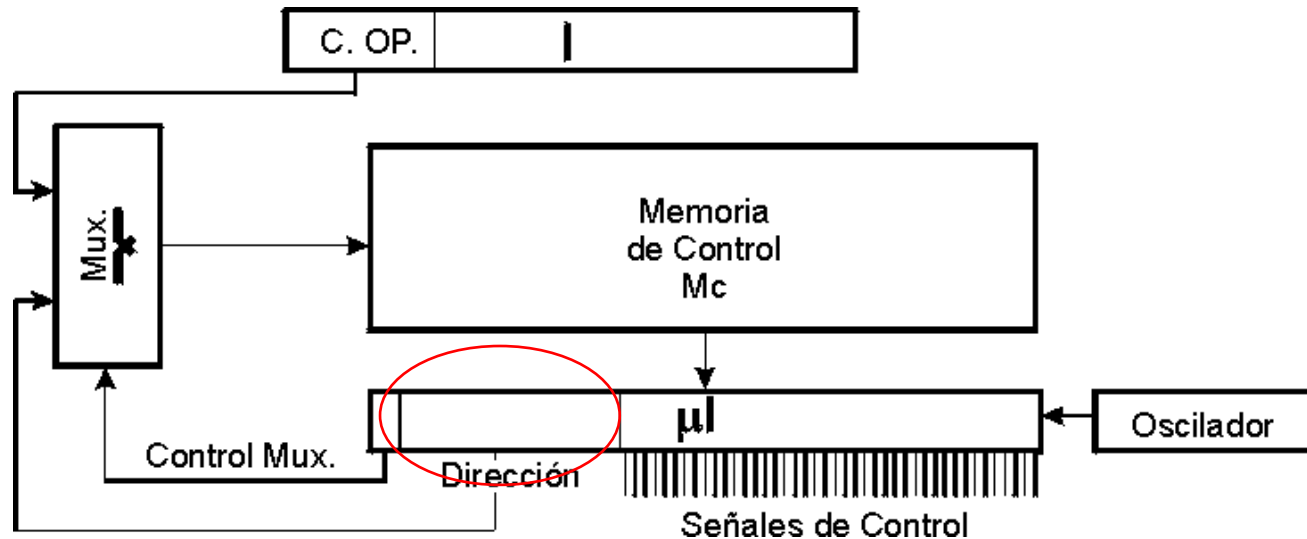
1. Secuenciamiento explícito.
2. Secuenciamiento implícito.

Estructura de UC microprogramada con secuenciamiento **explícito**



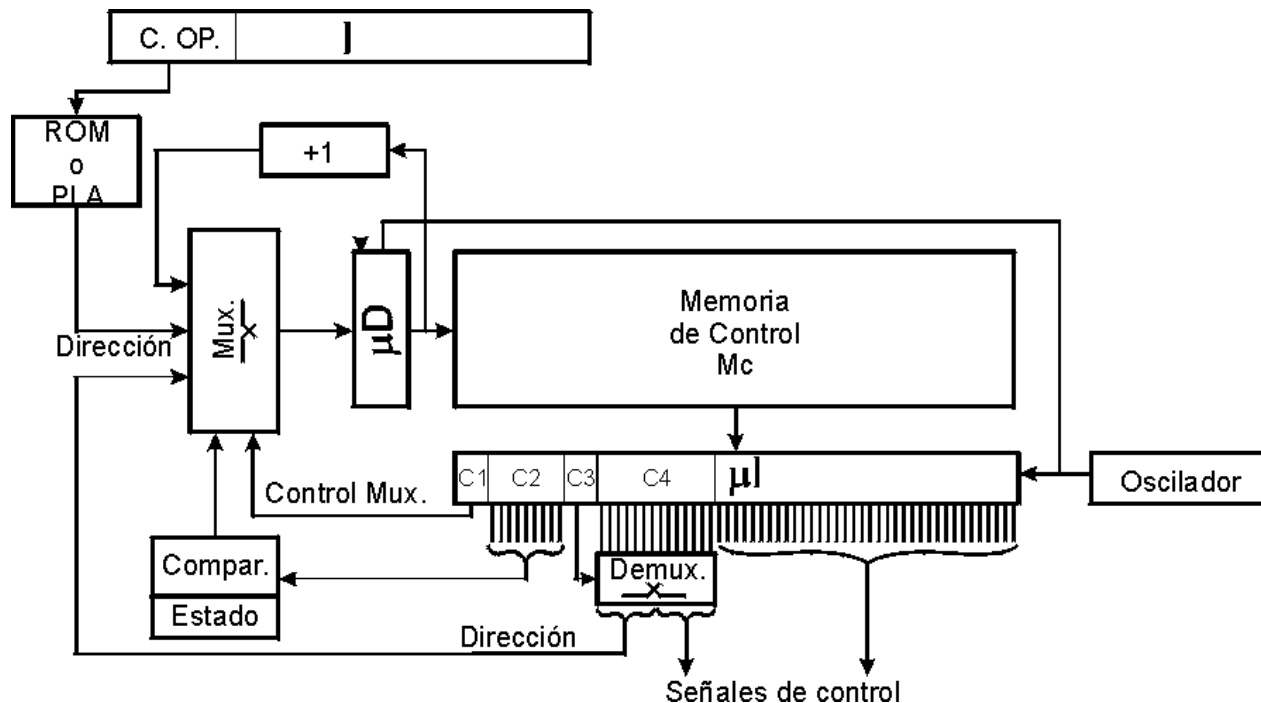
- ▶ Memoria de control guarda todos los μ programas, donde cada μ instrucción proporciona la μ dirección de la μ instrucción siguiente
- ▶ El CO representa la μ Dirección de la primera μ instrucción asociado a la instrucción máquina

Estructura de UC microprogramada con secuenciamiento **explícito**



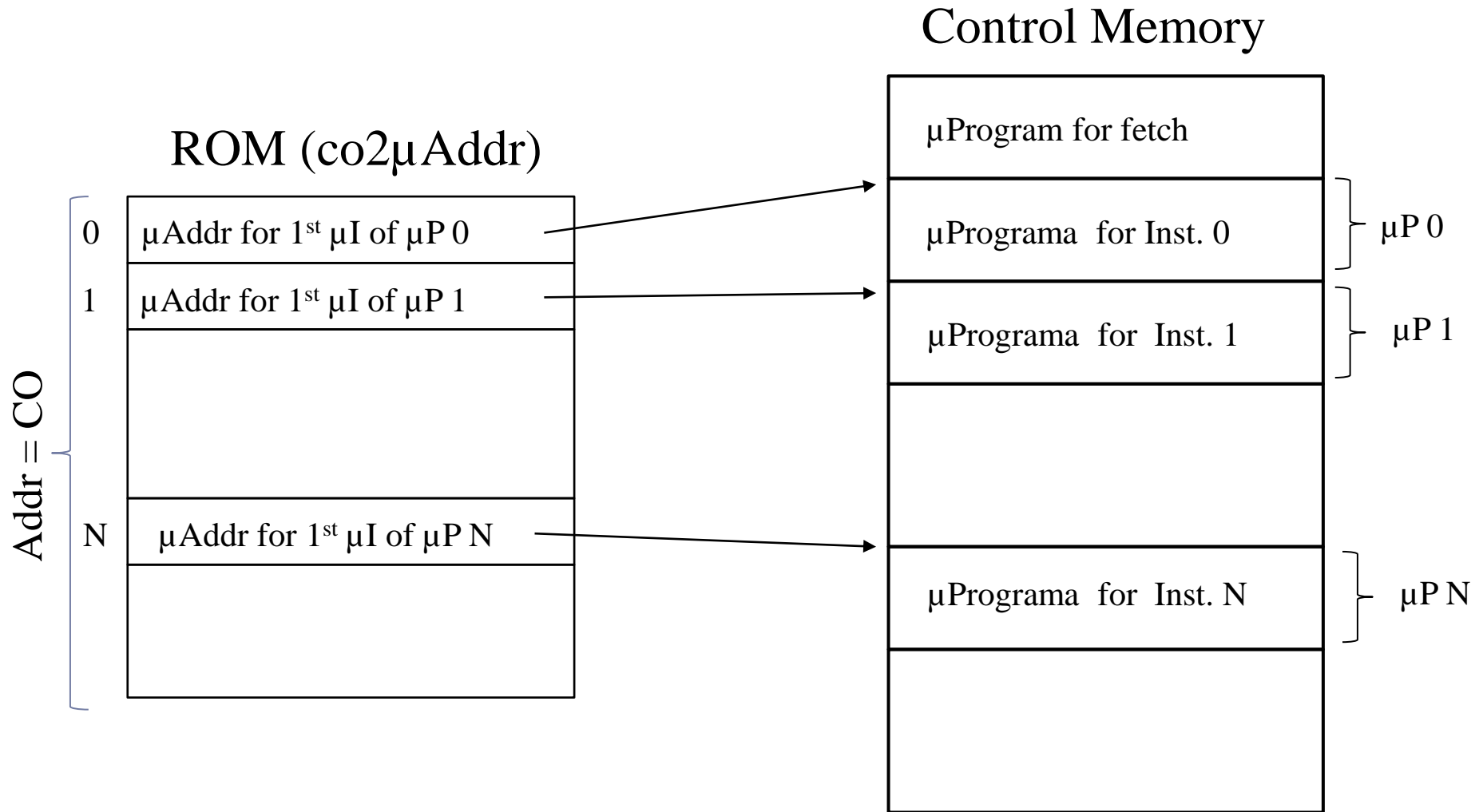
- ▶ Memoria de control guarda todos los μ programas, donde cada μ instrucción proporciona la μ dirección de la μ instrucción siguiente
- ▶ **Problema:** gran cantidad de memoria de control para el secuenciamiento de instrucciones, necesario almacena la μ dirección siguiente

Estructura de U.C. microprogramada con secuenciamiento **implícito**



- ▶ Memoria de control guarda todos los microprogramas de forma consecutiva en la memoria de control
- ▶ La ROM/PLA asocia a cada instrucción su microprograma (primera μ dirección
- ▶ Siguiendo μ instrucción (+1), μ bifurcaciones condicionales o μ bucles

Ejemplo de funcionamiento de una UC con secuenciación **implícito**

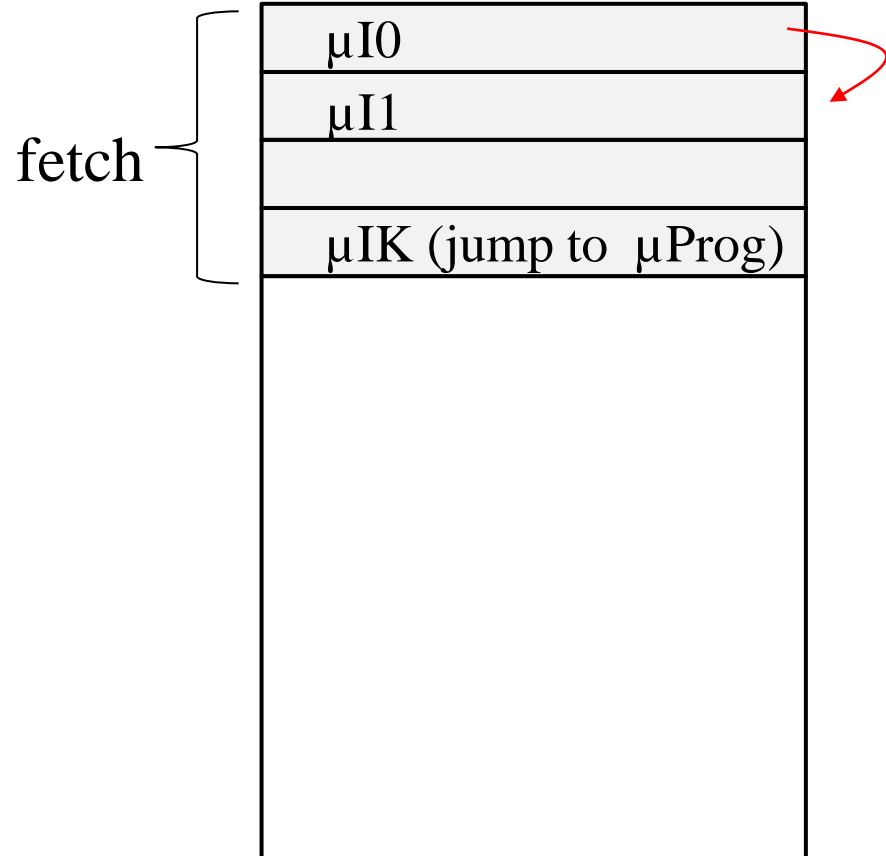


Ejemplo de funcionamiento de una UC con secuenciación **implícito**

CO ROM (co2 μ Addr)

0	μ Addr for 1 st μ I of μ P 0
1	μ Addr for 1 st μ I of μ P 1
N	μ Addr for 1 st μ I of μ P N

Control Memory

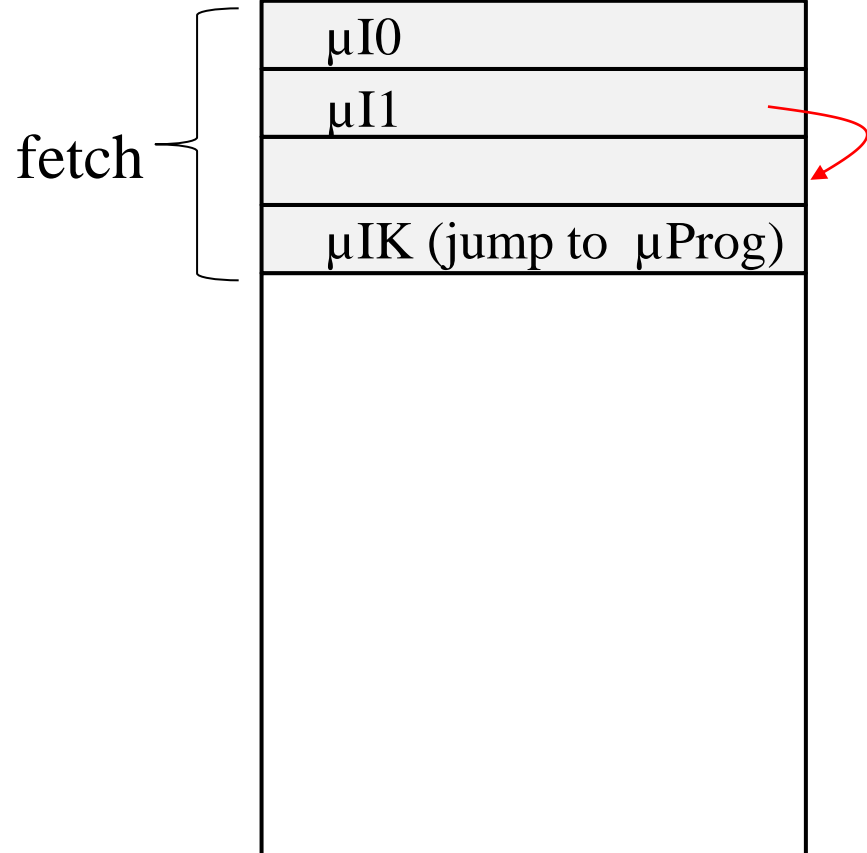


Ejemplo de funcionamiento de una UC con secuenciación **implícito**

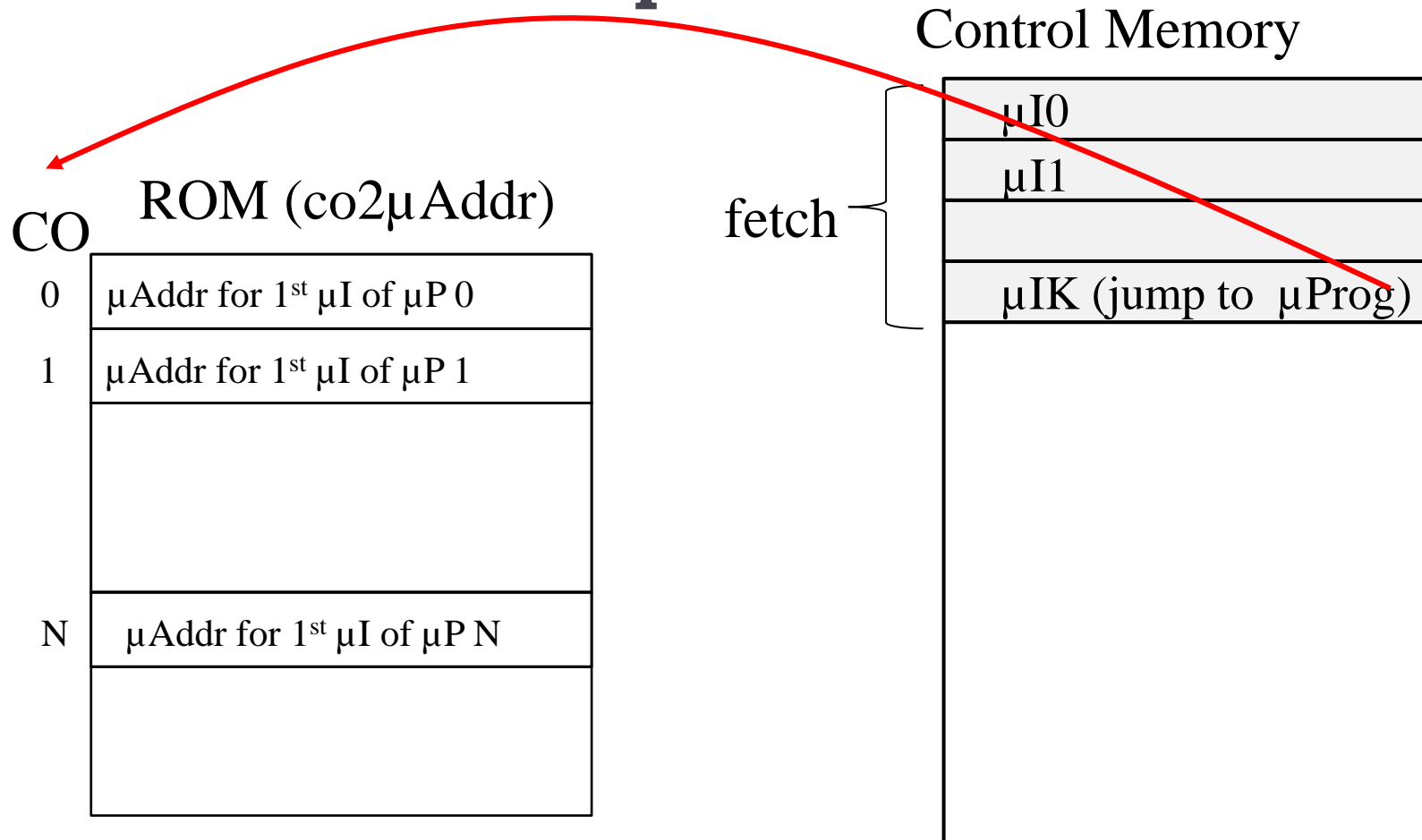
CO ROM (co2 μ Addr)

0	μ Addr for 1 st μ I of μ P 0
1	μ Addr for 1 st μ I of μ P 1
N	μ Addr for 1 st μ I of μ P N

Control Memory

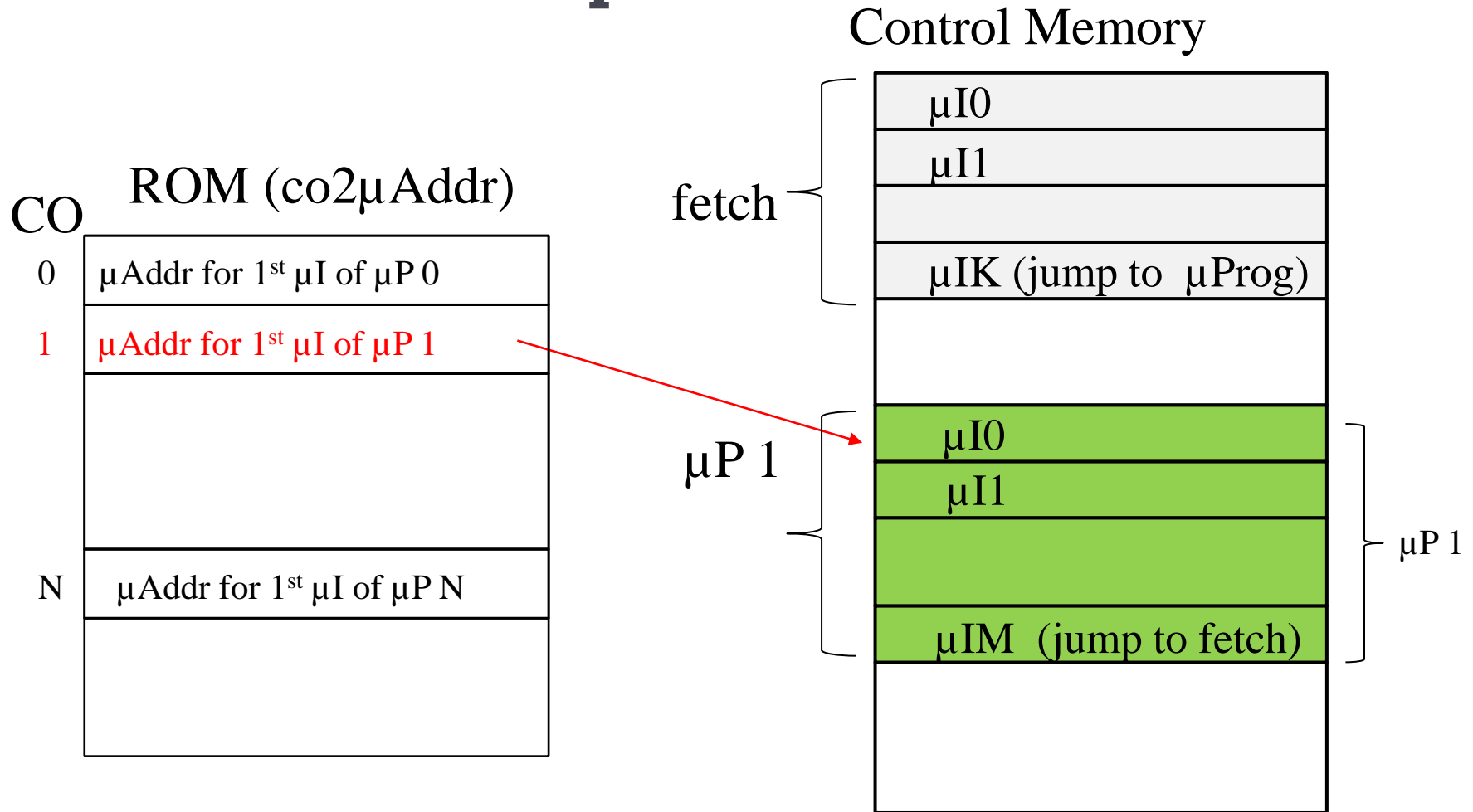


Ejemplo de funcionamiento de una UC con secuenciación **implícito**



En el Registro de Instrucción el CO

Ejemplo de funcionamiento de una UC con secuenciación **implícito**

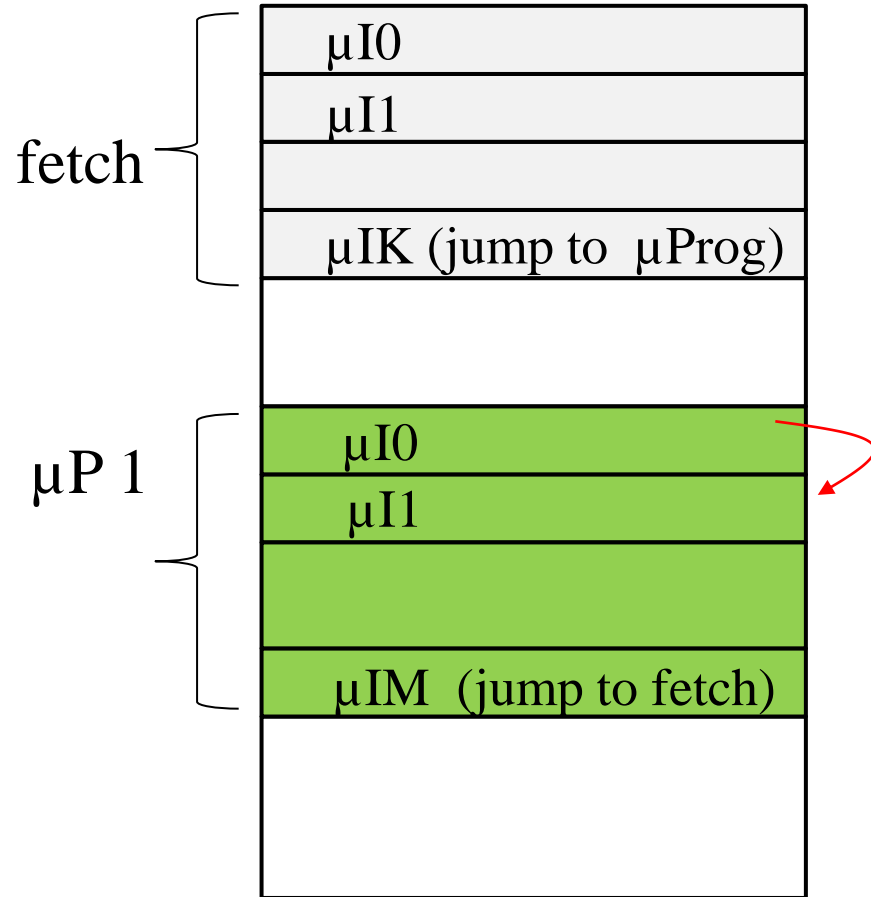


Ejemplo de funcionamiento de una UC con secuenciación **implícito**

CO ROM (co2 μ Addr)

0	μ Addr for 1 st μ I of μ P 0
1	μ Addr for 1 st μ I of μ P 1
N	μ Addr for 1 st μ I of μ P N

Control Memory

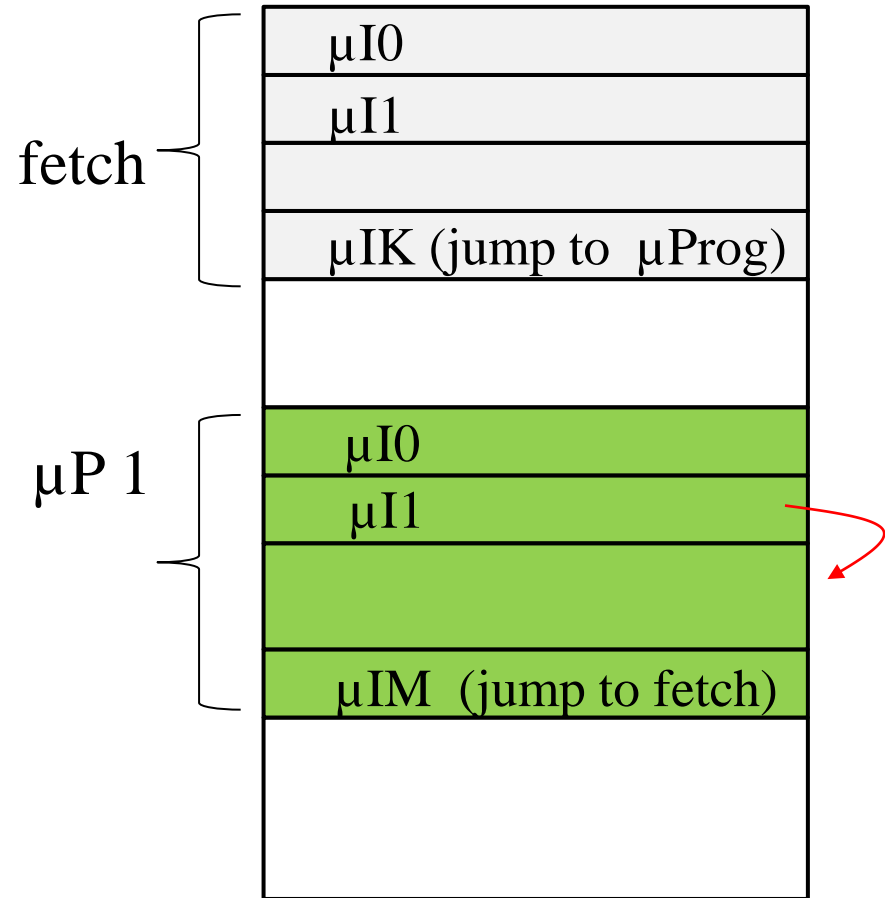


Ejemplo de funcionamiento de una UC con secuenciación **implícito**

CO ROM (co2 μ Addr)

0	μ Addr for 1 st μ I of μ P 0
1	μ Addr for 1 st μ I of μ P 1
N	μ Addr for 1 st μ I of μ P N

Control Memory

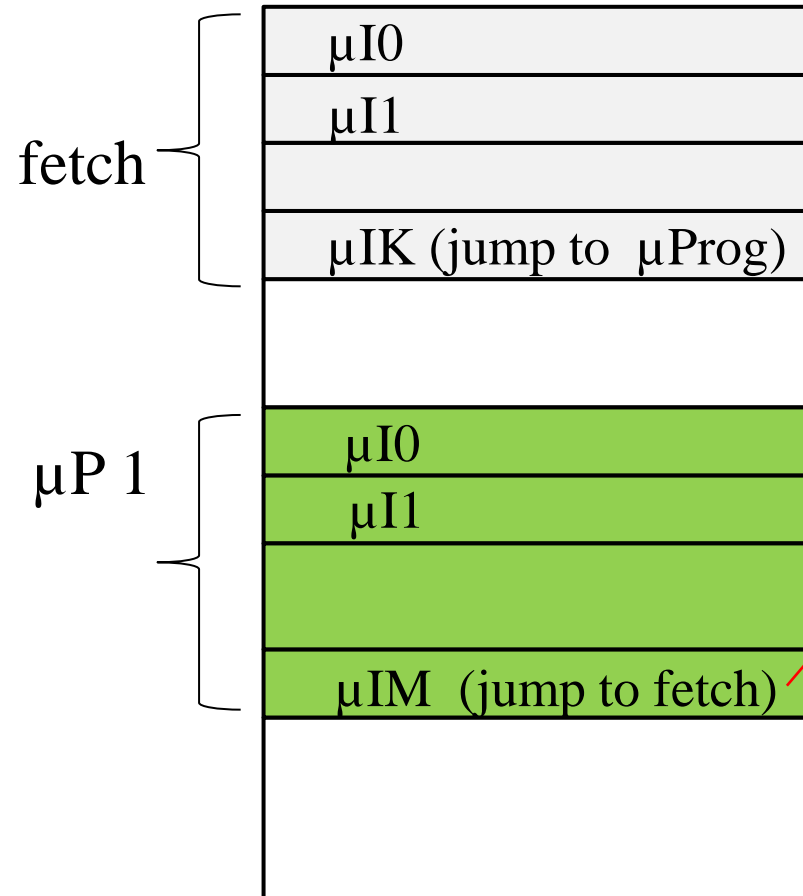


Ejemplo de funcionamiento de una UC con secuenciación **implícito**

CO ROM (co2 μ Addr)

0	μ Addr for 1 st μ I of μ P 0
1	μ Addr for 1 st μ I of μ P 1
N	μ Addr for 1 st μ I of μ P N

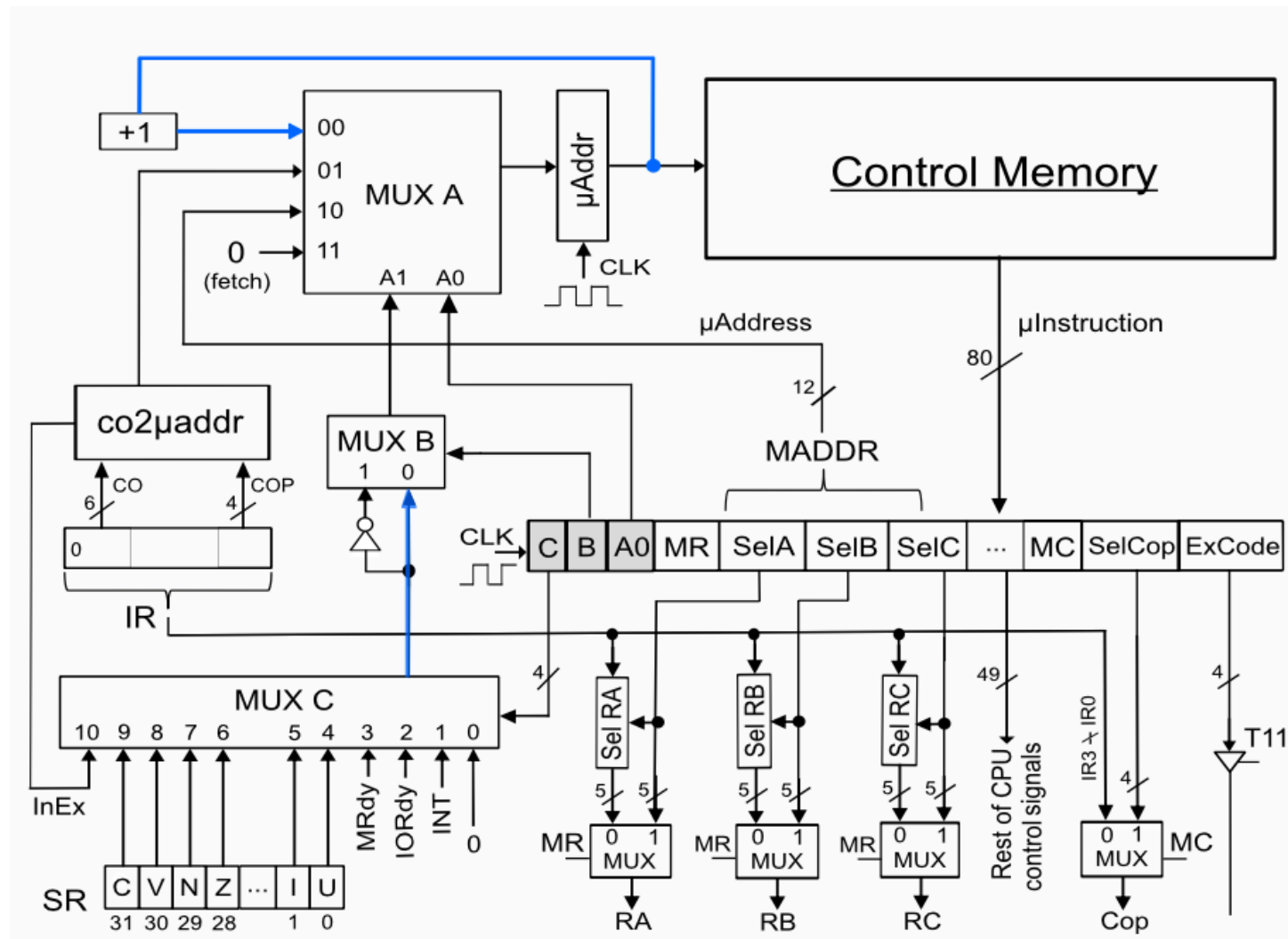
Control Memory



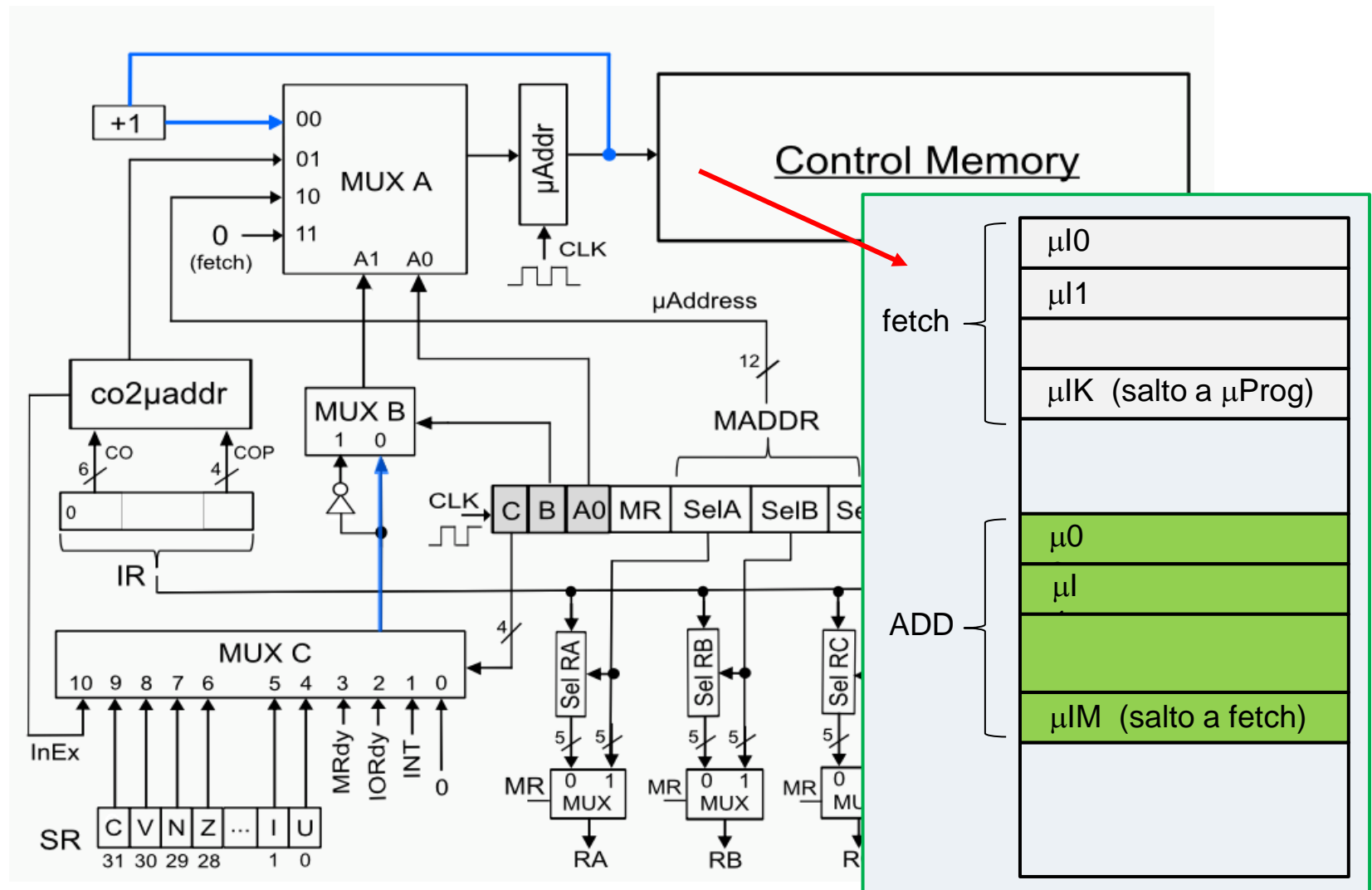
Contenidos

1. Elementos de un computador
2. Organización del procesador
3. La unidad de control
4. Ejecución de instrucciones
5. **Diseño de la unidad de control**
 - a) Tareas en el diseño de una unidad de control
 - b) Unidad de control almacenada
 - c) **Unidad de control en WepSIM**
 - d) Ejemplo de juego de instrucciones microprogramado
6. Modos de ejecución
7. Interrupciones
8. Arranque de un computador
9. Prestaciones y paralelismo

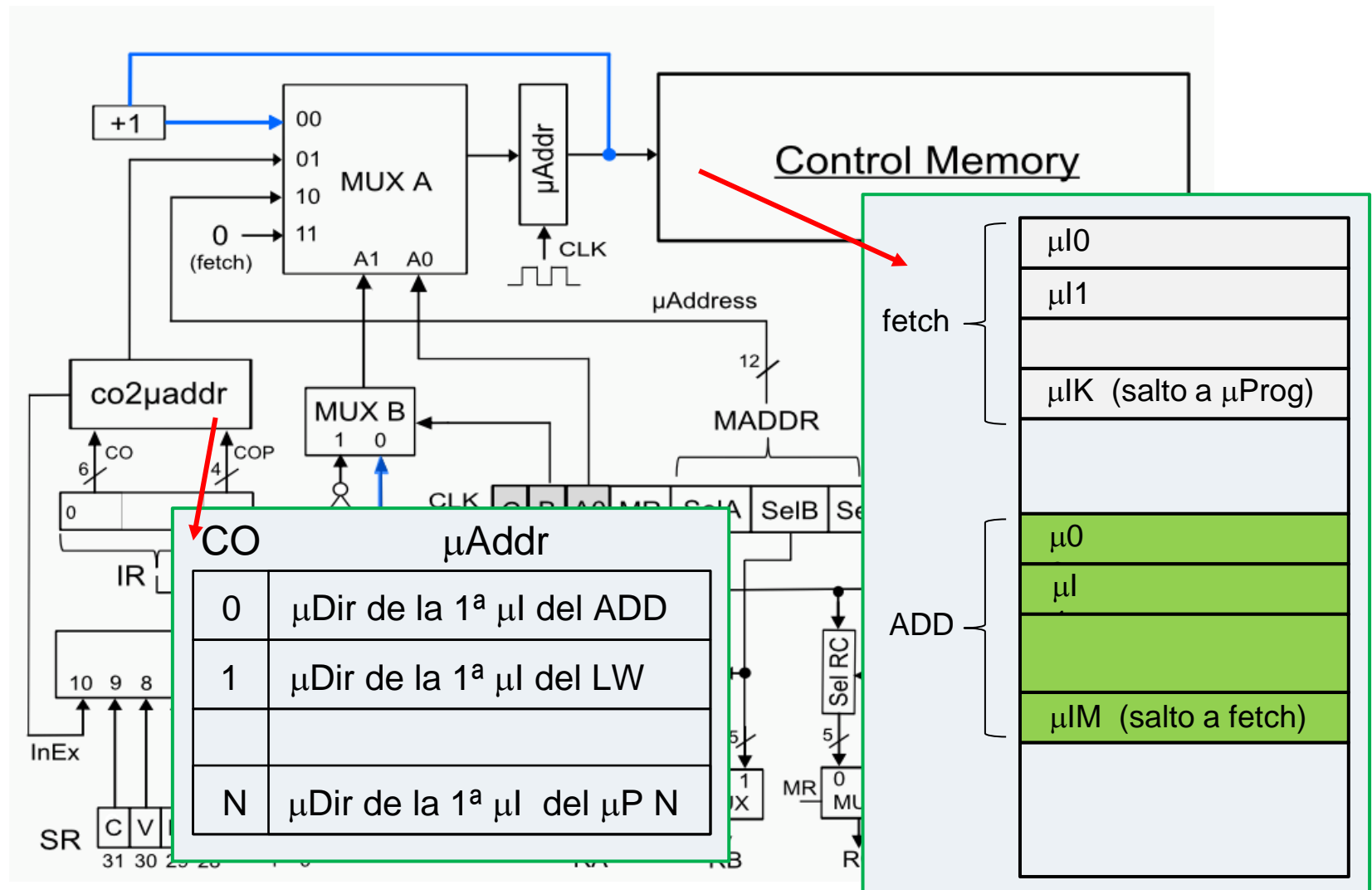
Unidad de control de WepSIM



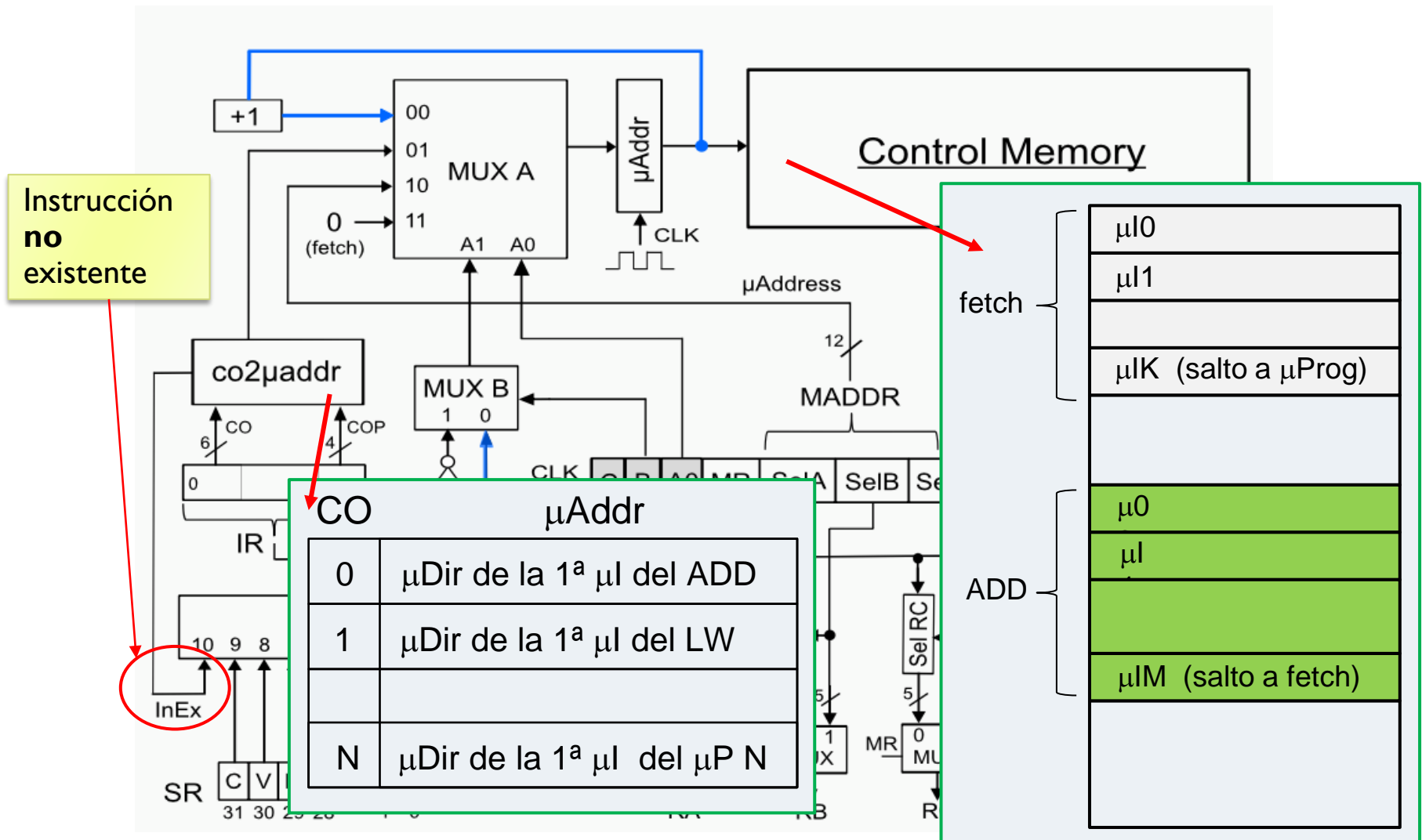
Unidad de control de WepSIM



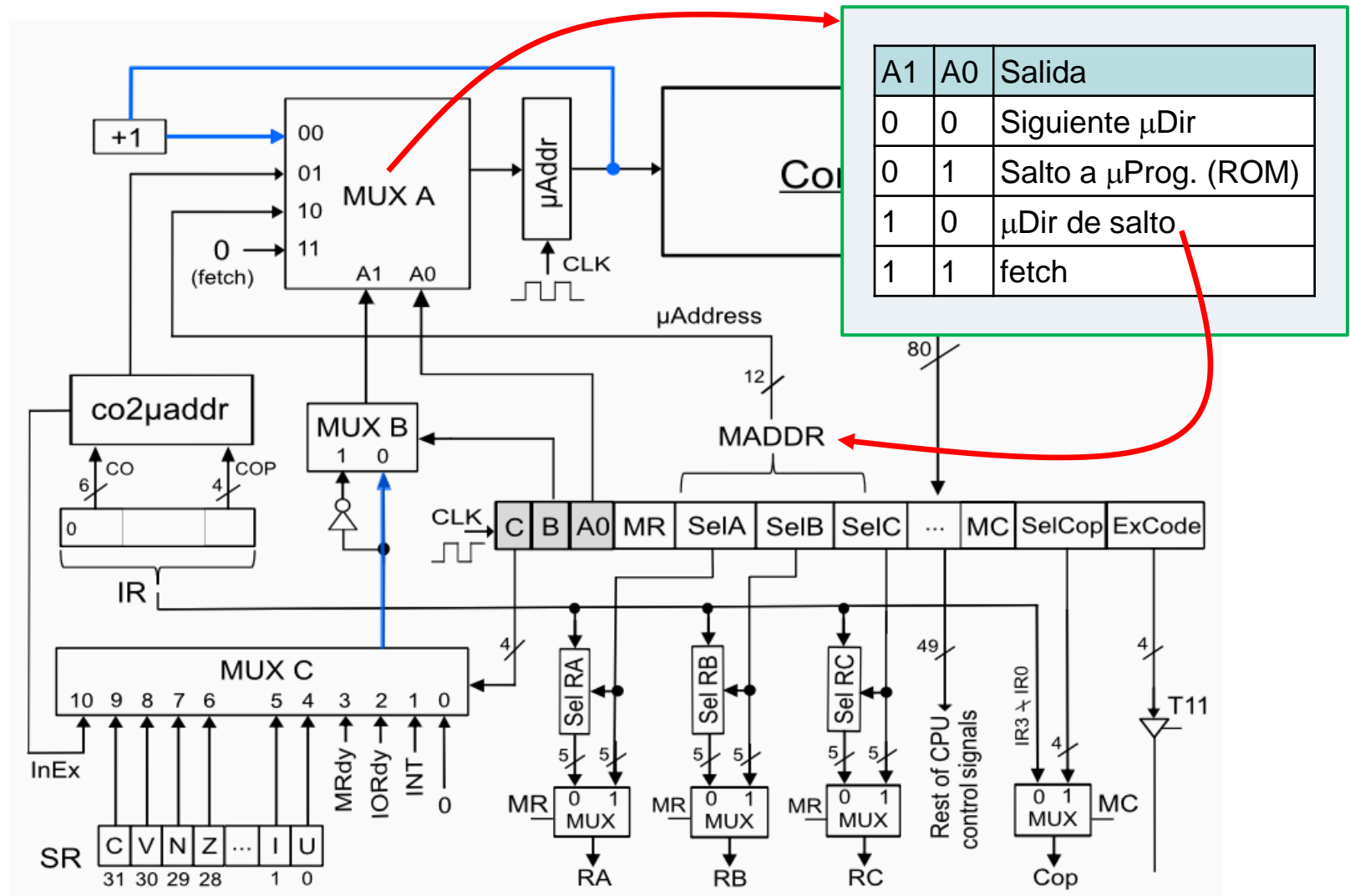
Unidad de control de WepSIM



Unidad de control de WepSIM



Unidad de control de WepSIM



Ejemplos de saltos más frecuentes

operaciones elementales con la UC

- ▶ **Salto a la dirección 000100011100 (12 bits) si Z = 1. En caso contrario se salta a la siguiente.**

O. Elemental	Señales
Si (Z) $\mu\text{PC}=000100011100$	$A0=0, B=0, C=0110_2, m\text{ADDR}=000100011100_2$

- ▶ **Salto incondicional a la dirección 000100011111**

O. Elemental	Señales
$\mu\text{PC}=000100011111$	$A0=0, B=1, C=0000_2, m\text{ADDR}=000100011111_2$

- ▶ **Salto a la primera μ dirección del μ programa asociado al CO**

O. Elemental	Señales
Salto a CO	$A0=1, B=0, C=0000_2$

Unidad de control de WepSIM

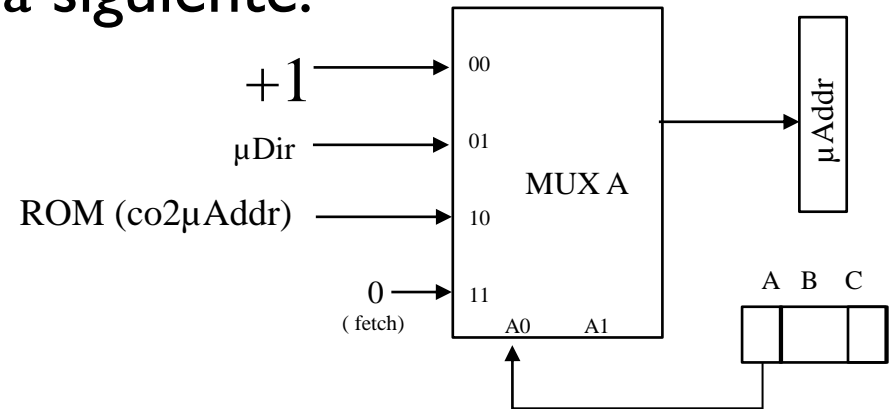
A0	B	C3	C2	C1	C0	Acción
0	0	0	0	0	0	Siguiente μ Dirección
0	1	0	0	0	0	Salto incondicional a MADDR
0	0	0	0	0	1	Salto condicional a MADDR si INT = 1 (*)
0	1	0	0	1	0	Salto condicional a MADDR si IORdy = 0 (*)
0	1	0	0	1	1	Salto condicional a MADDR si MRdy = 0 (*)
0	0	0	1	0	0	Salto condicional a MADDR si U = 1 (*)
0	0	0	1	0	1	Salto condicional a MADDR si I = 1 (*)
0	0	0	1	1	0	Salto condicional a MADDR si Z = 1 (*)
0	0	0	1	1	1	Salto condicional a MADDR si N = 1 (*)
0	0	1	0	0	0	Salto condicional a MADDR si O = 1 (*)
1	0	0	0	0	0	Salto a μ Prog. (ROM c02 μ addr)
1	1	0	0	0	0	Salto a fetch (μ Dir = 0)

- ▶ (*) Si no se cumple la condición \rightarrow Siguiente μ Dirección
- ▶ Resto de entradas \rightarrow funcionamiento indefinido

Ejemplo

- ▶ Salto a la μ Dirección 000100011100 (12 bits) si $Z = 1$.
En caso contrario se salta a la siguiente:

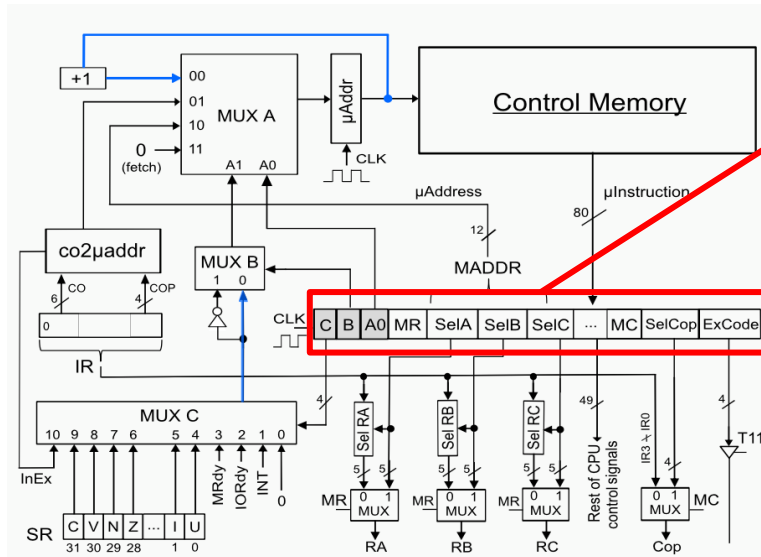
- ▶ $A0 = 0$
- ▶ $B = 0$
- ▶ $C = 0110$
- ▶ $\mu\text{Addr} = 000100011100$



- ▶ Salto incondicional a la μ Dirección 000100011111
- ▶ $A0 = 0$
- ▶ $B = 1$
- ▶ $C = 0000$
- ▶ $\mu\text{Addr} = 000100011111$

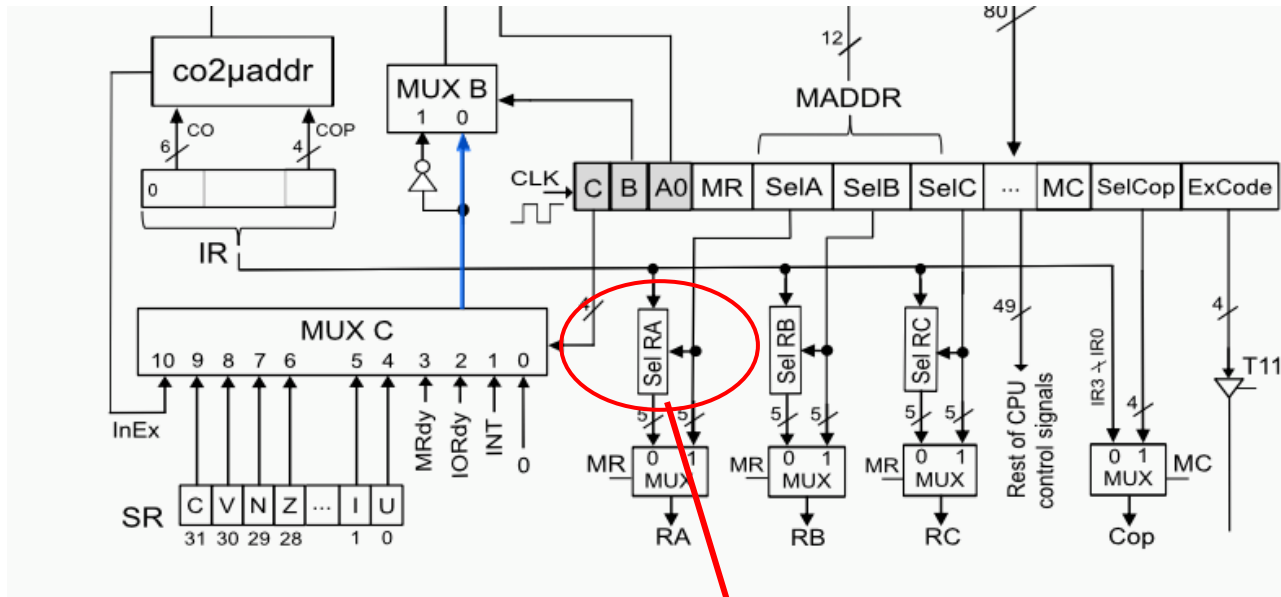
μ Dirección codificada
en los bits 72-61 de la
 μ Instrucción

Formato de la microinstrucción

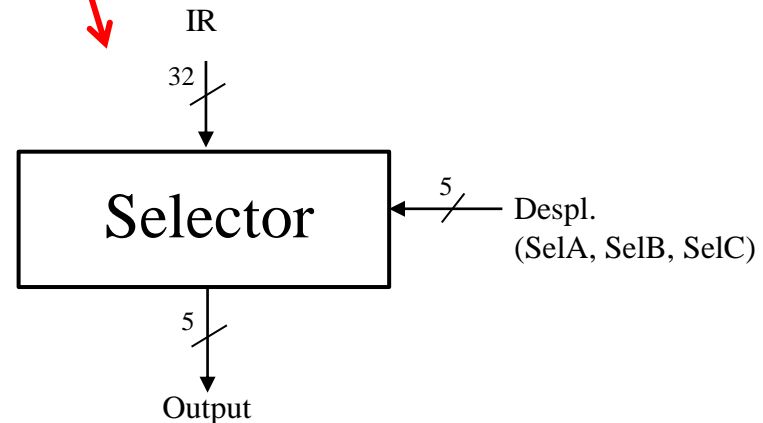


C0 .. C7	Carga en registros
Ta,Td	Triestados a buses
T1..T10	Puertas triestado
M1,M2, M7, MA, MB	Multiplexores
SelP	Selector Registro estado
LC	Carga en Register File
SE	Extensión de signo
Size, Offset	Selector del registro IR
BW	Tamaño de operación en memoria
R,W	Operación de memoria
IOR, IOW	Operación de E/S
INTA	Reconocimiento INT
I	Habilitar interrupciones
U	Usuario/núcleo

Selector de registros

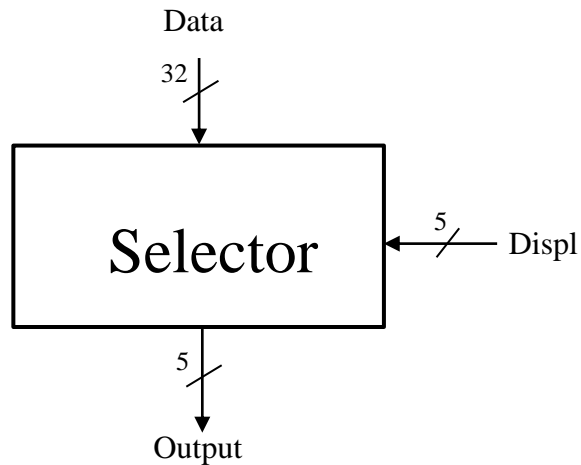
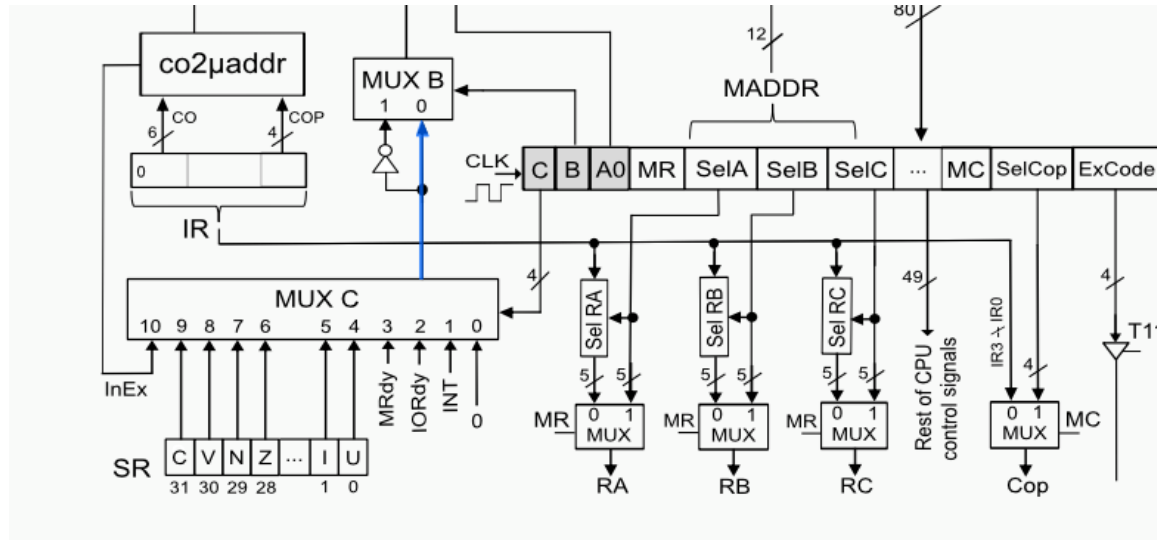


Selecciona 5 bits de un conjunto de 32 bits desde la posición indicada en Despl. (bit inferior)



Selector de registros

Ejemplo



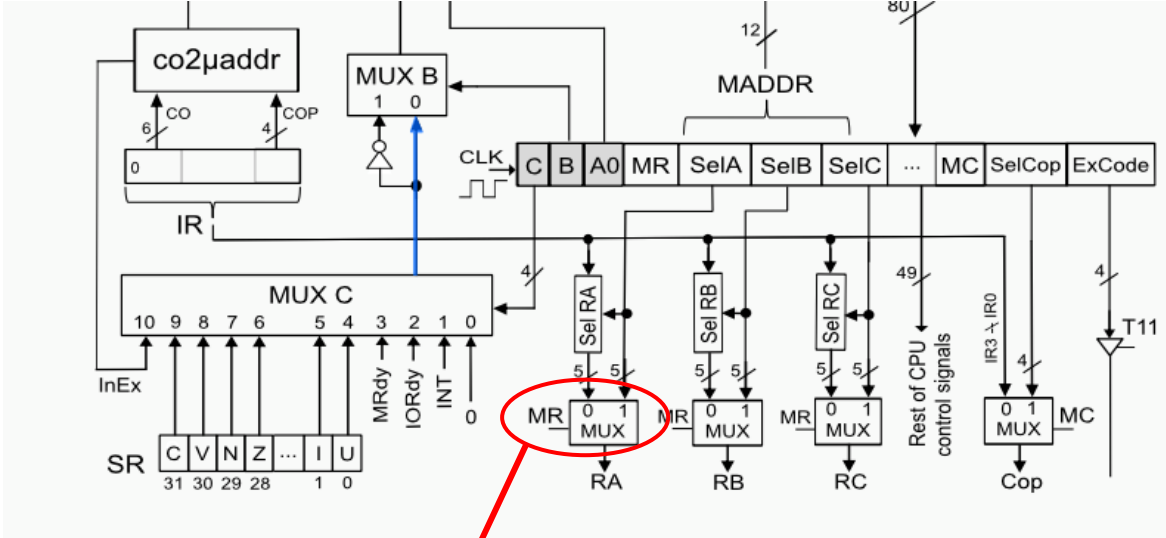
RI: $D_{31}D_{30}D_{29}D_{28}D_{27}D_{26}D_{25} \dots D_4D_3D_2D_1D_0$

Si Displ = 11011 \rightarrow Output = $D_{31}D_{30}D_{29}D_{28}D_{27}$

Si Displ = 00000 → Output = D₄D₃D₂D₁D₀

Si Displ = 10011 \rightarrow Output = $D_{23}D_{22}D_{21}D_{20}D_{19}$

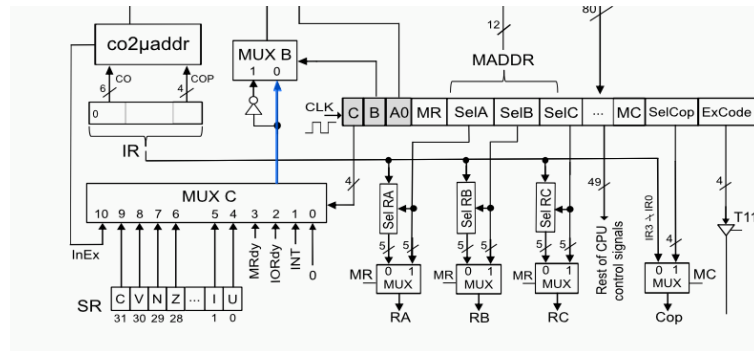
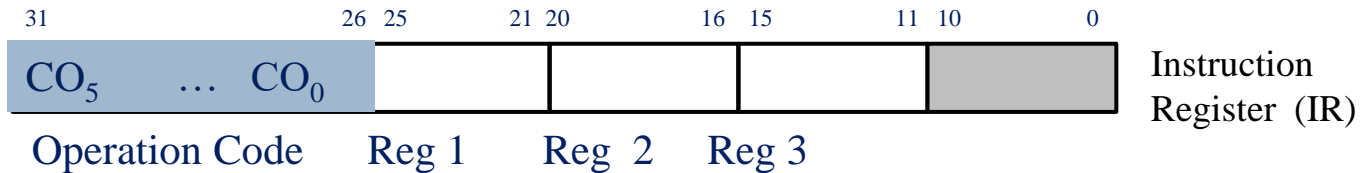
Si Displ = 01011 → Output = D₁₅D₁₄D₁₃D₁₂D₁₁



- Si $MR = 1$, RA se obtiene directamente de la μ Instrucción
- Si $MR = 0$, RA se obtiene de un campo de la instrucción (en IR)

Selector de registros

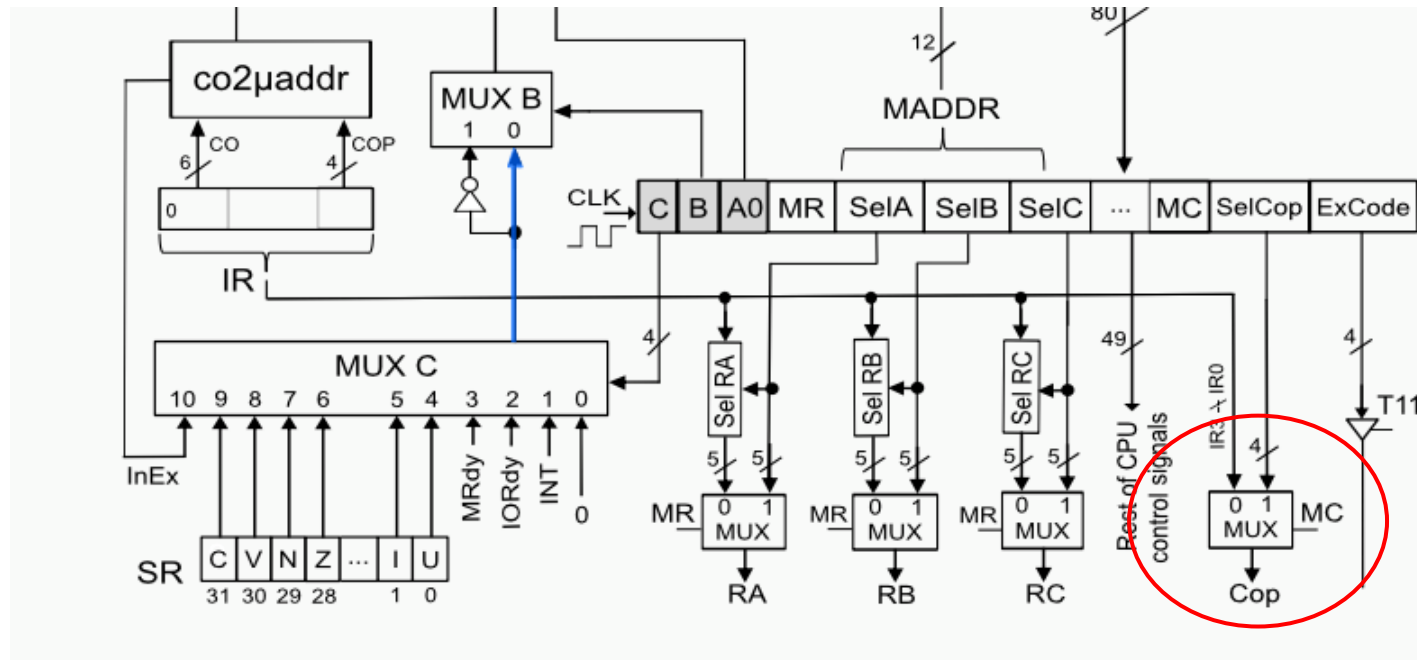
- Si el formato de una instrucción almacenada en IR es:



MR = 0

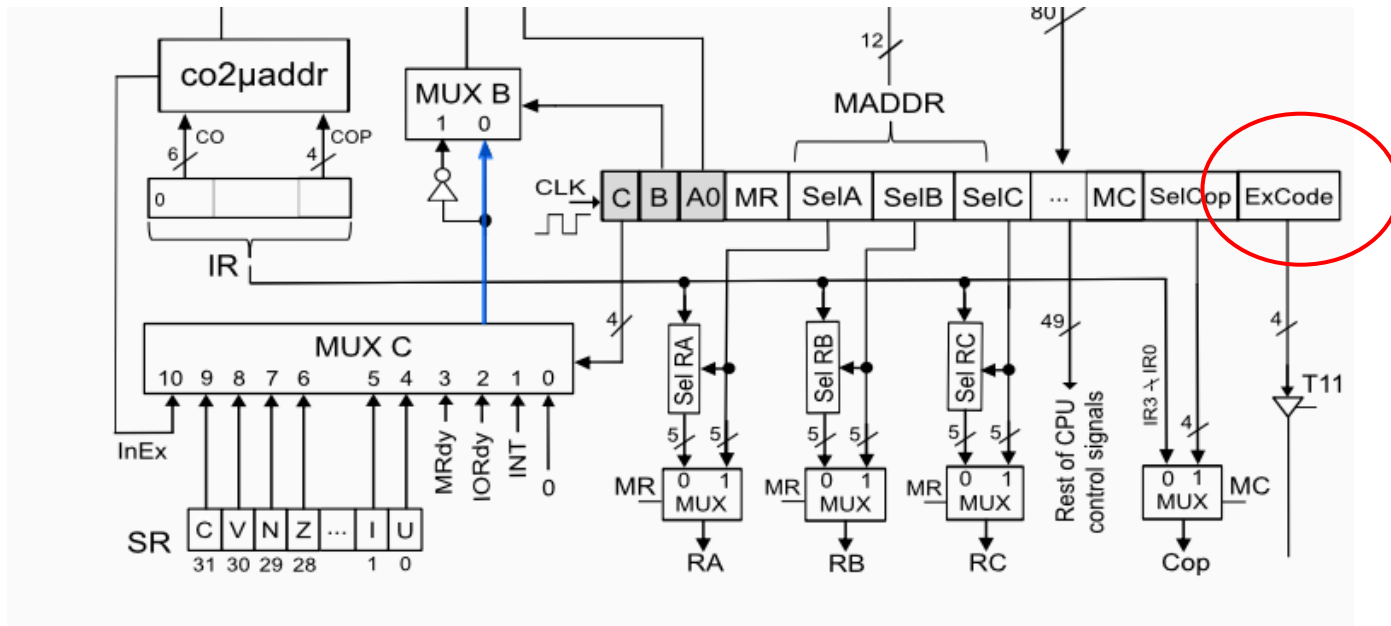
- Si se quiere seleccionar el campo con el Reg 2 en la puerta B del banco de registros → SelB = 10000 (RB se obtiene de los bits 20...16 del IR)
- Si se quiere seleccionar el campo con el Reg 3 en la puerta A del banco de registros → SelA = 01011 (RA se obtiene de los bits 15...11 del IR)
- Si se quiere seleccionar el campo con el Reg 1 en la puerta C del banco de registros → SelC = 10101 (RC se obtiene de los bits 25...21 del IR)

Selección del código de operación de la ALU



- Si $MC = 1$, el código de operación de la ALU se obtiene directamente de la microinstrucción (SelCop)
- Si $MC = 0$, el código de operación de la ALU se obtiene de los cuatro últimos bits almacenados en el registro de instrucción

Código de excepción



- ExCode:

- Permite tener un valor inmediato cualquiera de 4 bits,
- Especialmente útil para generar el vector de interrupción a utilizar cuando se produce una excepción en la instrucción.

Contenidos

1. Elementos de un computador
2. Organización del procesador
3. La unidad de control
4. Ejecución de instrucciones
5. **Diseño de la unidad de control**
 - a) Tareas en el diseño de una unidad de control
 - b) Unidad de control almacenada
 - c) Unidad de control en WepSIM
 - d) **Ejemplo de juego de instrucciones microprogramado**
6. Modos de ejecución
7. Interrupciones
8. Arranque de un computador
9. Prestaciones y paralelismo

Ejemplo

► Instrucciones a microprogramar con WepSIM*:

Instrucción	Cód. Oper.	Significado
ADD Rd, Rf1, Rf2	000000	$Rd \leftarrow Rf1 + Rf2$
LI R, valor	000001	$R \leftarrow \text{valor}$
LW R, dir	000010	$R \leftarrow MP[\text{dir}]$
SW R, dir	000011	$MP[\text{dir}] \leftarrow R$
BEQ Rf1, Rf2, displ	000100	if ($Rf1 == Rf2$) $PC \leftarrow PC + \text{desp}$
J dir	000101	$PC \leftarrow \text{dir}$
HALT	000110	Parada, bucle infinito

* Memoria de un ciclo

Microprograma de las instrucciones

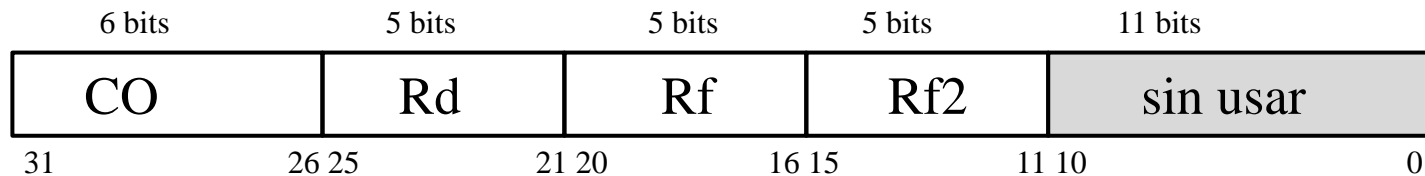
► FETCH

Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	$MAR \leftarrow PC$	T2, C0	0000	0	0
1	$MBR \leftarrow MP$	Ta, R, BW = 11, CI, MI	0000	0	0
	$PC \leftarrow PC + 4$	M2, C2	0000	0	0
2	$IR \leftarrow MBR$	T1, C3	0000	0	0
3	Decodificación		0000	0	1

Microprograma de las instrucciones

► ADD Rd, Rf1, Rf2

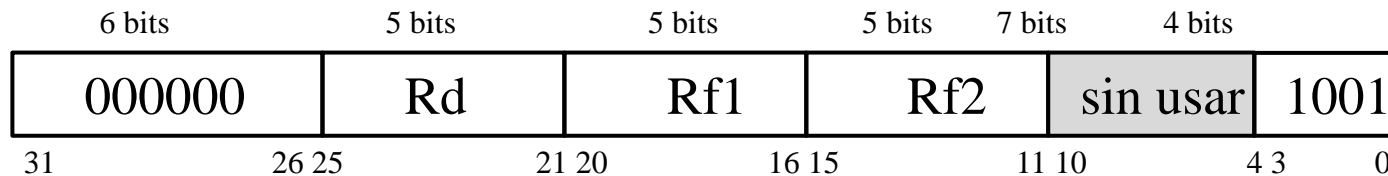
Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	$Rd \leftarrow Rf1 + Rf2$	Cop = 1010 SelP = 11, C7, M7 T6, LC SelA = 10000 (16) SelB = 01011 (11) SelC = 10101 (21)	0000	1	1



Microprograma de las instrucciones (otra)

► ADD Rd, Rf1, Rf2

Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	$Rd \leftarrow Rf1 + Rf2$	SelCop = 1010, MC SelP=11, C7, M7 T6, LC SelA = 10000 (16) SelB = 01011 (11) SelC = 10101 (21)	0000	1	1



Microprograma de las instrucciones

► LI R, valor

Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	$R \leftarrow IR$ (valor)	LC SelC = 10101 (21) T3, Size = 10000 Offset= 00000 SE=1	0000	1	1

6 bits

5 bits

5 bits

16 bits



31

26 25

21 20

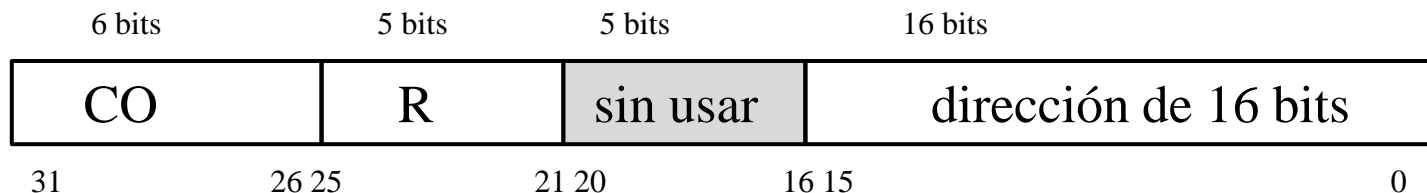
16 15

0

Microprograma de las instrucciones

- LW R dir, con memoria síncrona de un ciclo

Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	MAR \leftarrow IR (dir)	T3, C0 Size = 10000, Offset= 00000	0000	0	0
1	MBR \leftarrow MP[MAR]	Ta, R, BW = 11, CI, MI	0000	0	0
2	R \leftarrow MBR	TI, LC, SelC = 10101	0000	1	1



Microprograma de las instrucciones

- LW R dir, con memoria asíncrona (MRdy=1 indica el fin)

Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	MAR \leftarrow IR (dir)	T3, C0 Size = 10000, Offset= 00000	0000	0	0
1	while (!MRdy) MBR \leftarrow MP[MAR]	Ta, R, BW = 11, C1, M1, MADDR= μ Add de esta μ instrucción	0011	1	0
2	R \leftarrow MBR	T1, LC, SelC = 10101	0000	1	1

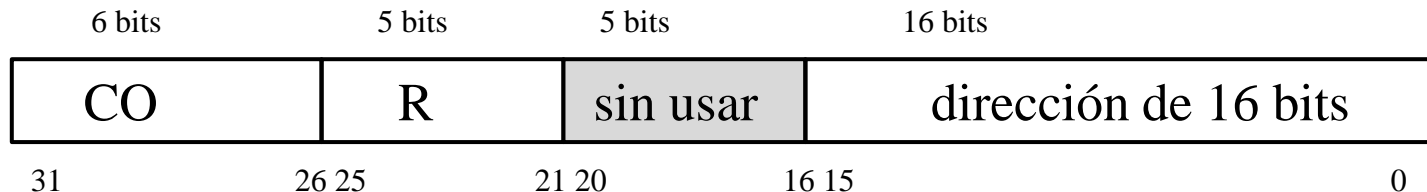
Se ejecuta esta microinstrucción mientras MRdy==0



Microprograma de las instrucciones

- SW R dir, con memoria síncrona de un ciclo

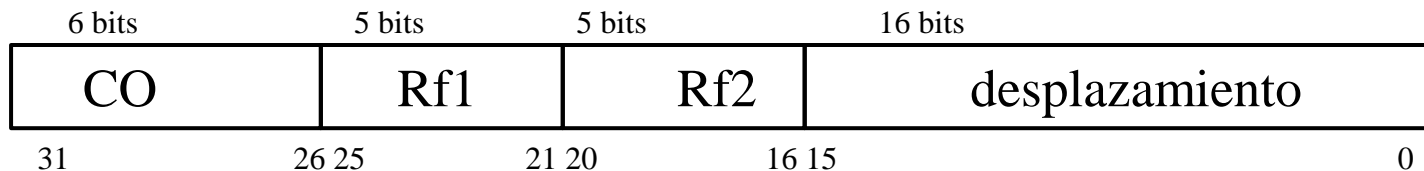
Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	$MBR \leftarrow R$	$T9, CI, SelA = 10101$	0000	0	0
1	$MAR \leftarrow IR(dir)$	$T3, C0,$ Size = 10000, offset= 00000	0000	0	0
2	$MP[dir] \leftarrow MBR$	$Td, Ta, BW = 11, W$	0000	1	1



Microprograma de las instrucciones

► BEQ Rf1, Rf2, desp

Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	Rf1 - Rf2	SelCop = 1011, MC, C7, M7 SelP = 11, SelA = 10101 SelB = 10000	0000	0	0
11	If (Z == 0) goto fetch else next	MADDR = 0	0110	1	0
2	RT1 ← PC	T2, C4	0000	0	0
3	RT2 ← IR (dir)	Size = 10000 Offset = 00000, T3, C5	0000	0	0
4	PC ← RT1 + RT2	SelCop = 1010, MC, MA, MB=01, T6, C2,	0000	1	1



Microprograma de las instrucciones

► J dir

Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	$PC \leftarrow IR \text{ (dir)}$	C2,T3, size = 10000, offset= 00000	0000	I	I



Especificación de los microprogramas en WepSIM

<Lista de microcódigos>

<Especificación de registros>

<Pseudoinstrucciones>

Especificación de los microprogramas en WepSIM

begin

{

 fetch: (T2, C0=1),
 (Ta, R, BW=11, C1, M1),
 (M2, C2, T1, C3),
 (A0, B=0, C=0)

}

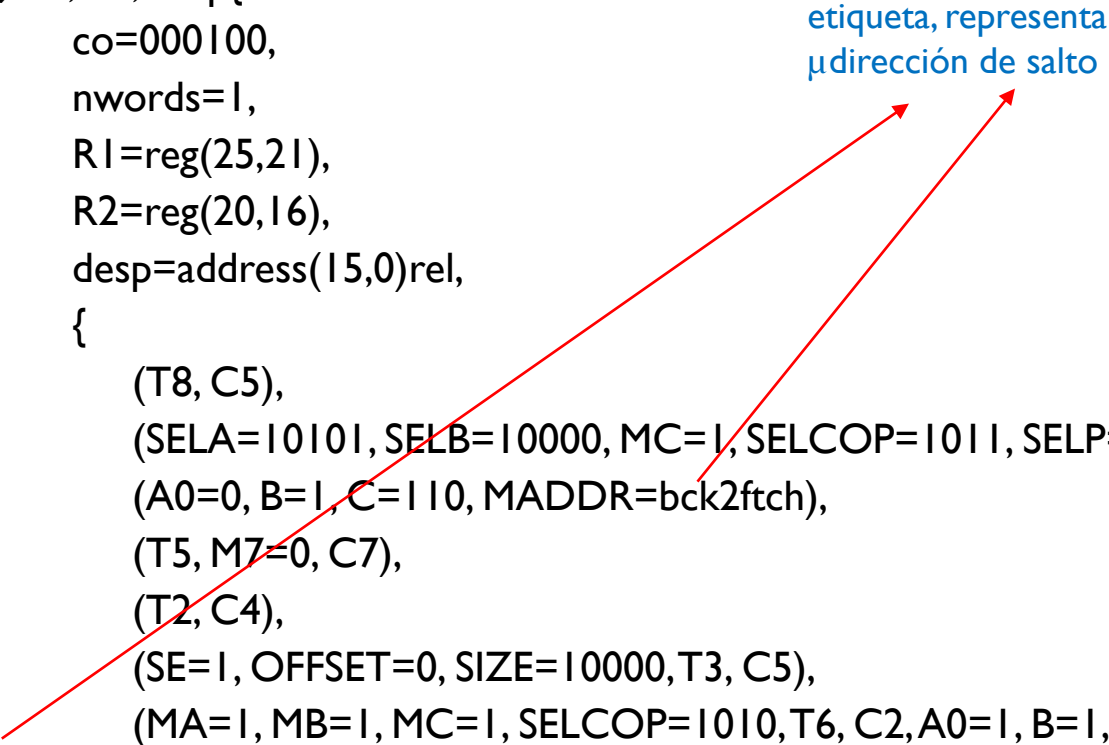
Especificación de los microprogramas en WepSIM

```
ADD R1,R2, R3{  
    co=000000,  
    nwords=1,  
    R1=reg(25,21),  
    R2=reg(20,16),  
    R3=reg(15,11),  
    {  
        (SelCop=1010, MC, SelP=11, M7,C7,T6, LC,  
        SelA=01011, SelB=10000, SelC=10101,  
        A0=1, B=1, C=0)  
    }  
}
```

Especificación de los microprogramas en WepSIM

```
BEQ R1, R2, desp{
    co=000100,
    nwords=1,
    R1=reg(25,21),
    R2=reg(20,16),
    desp=address(15,0)rel,
    {
        (T8, C5),
        (SELA=10101, SELB=10000, MC=1, SELCOP=1011, SELP=11, M7, C7),
        (A0=0, B=1, C=110, MADDR=bck2ftch),
        (T5, M7=0, C7),
        (T2, C4),
        (SE=1, OFFSET=0, SIZE=10000, T3, C5),
        (MA=1, MB=1, MC=1, SELCOP=1010, T6, C2, A0=1, B=1, C=0),
        bck2ftch: (T5, M7=0, C7),
        (A0=1, B=1, C=0)
    }
}
```

etiqueta, representa
μdirección de salto



Especificación de registros

```
registers {  
    0=(zero, x0),  
    2=(sp, x2) (stack_pointer),  
    4=(tp, x4),  
    6=(t1, x6),  
    8=(s0, x8),  
    10=(a0, x10),  
    12=(a2, x12),  
    14=(a4, x14),  
    16=(a6, x16),  
    18=(s2, x18),  
    20=(s4, x20),  
    22=(s6, x22),  
    24=(s8, x24),  
    26=(s10, x26),  
    28=(t3, x28),  
    30=(t5, x30),  
    1=(ra, x1),  
    3=(gp, x3),  
    5=(t0, x5),  
    7=(t2, x7),  
    9=(s1, x9),  
    11=(a1, x11),  
    13=(a3, x13),  
    15=(a5, x15),  
    17=(a7, x17),  
    19=(s3, x19),  
    21=(s5, x21),  
    23=(s7, x23),  
    25=(s9, x25),  
    27=(s11, x27),  
    29=(t4, x29),  
    31=(t6, x31)  
}
```

Contenido

1. Elementos de un computador
2. Organización del procesador
3. La unidad de control
4. Ejecución de instrucciones
5. Diseño de la unidad de control
6. **Modos de ejecución**
7. Interrupciones
8. Arranque de un computador
9. Prestaciones y paralelismo

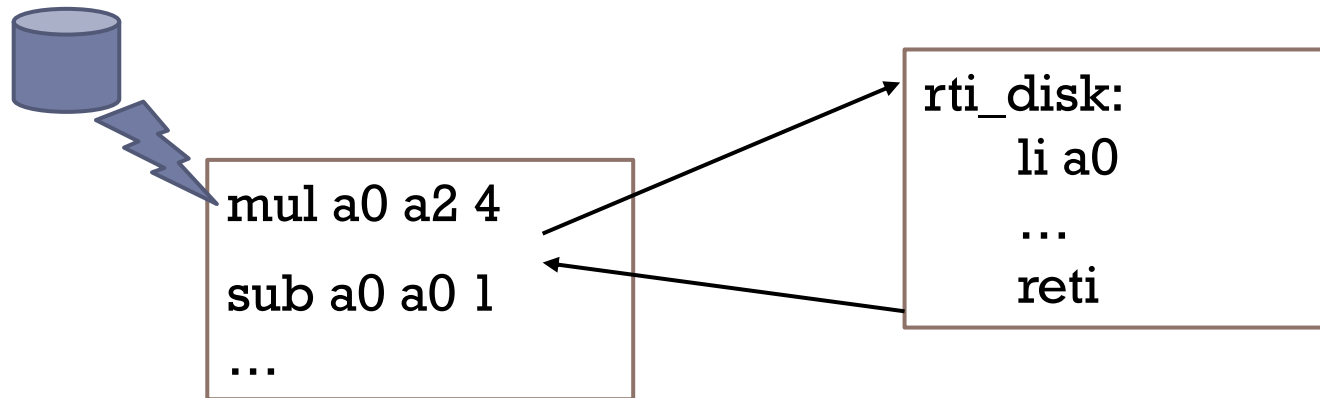
Modos de ejecución

- ▶ Se indica con un bit situado en el **registro de estado (U)**
- ▶ Al menos 2 modos:
 - ▶ **Modo usuario**
 - ▶ El procesador no puede **ejecutar instrucciones privilegiadas** (ejemplo: instrucciones de E/S, de habilitación de interrupciones, ...)
 - ▶ Si un proceso de usuario ejecuta una instrucción privilegiada se produce una interrupción
 - ▶ **Modo núcleo**
 - ▶ Reservado al sistema operativo
 - ▶ El procesador puede ejecutar todo el repertorio de instrucciones

Contenido

1. Elementos de un computador
2. Organización del procesador
3. La unidad de control
4. Ejecución de instrucciones
5. Diseño de la unidad de control
6. Modos de ejecución
7. **Interrupciones**
8. Arranque de un computador
9. Prestaciones y paralelismo

Idea de interrupción

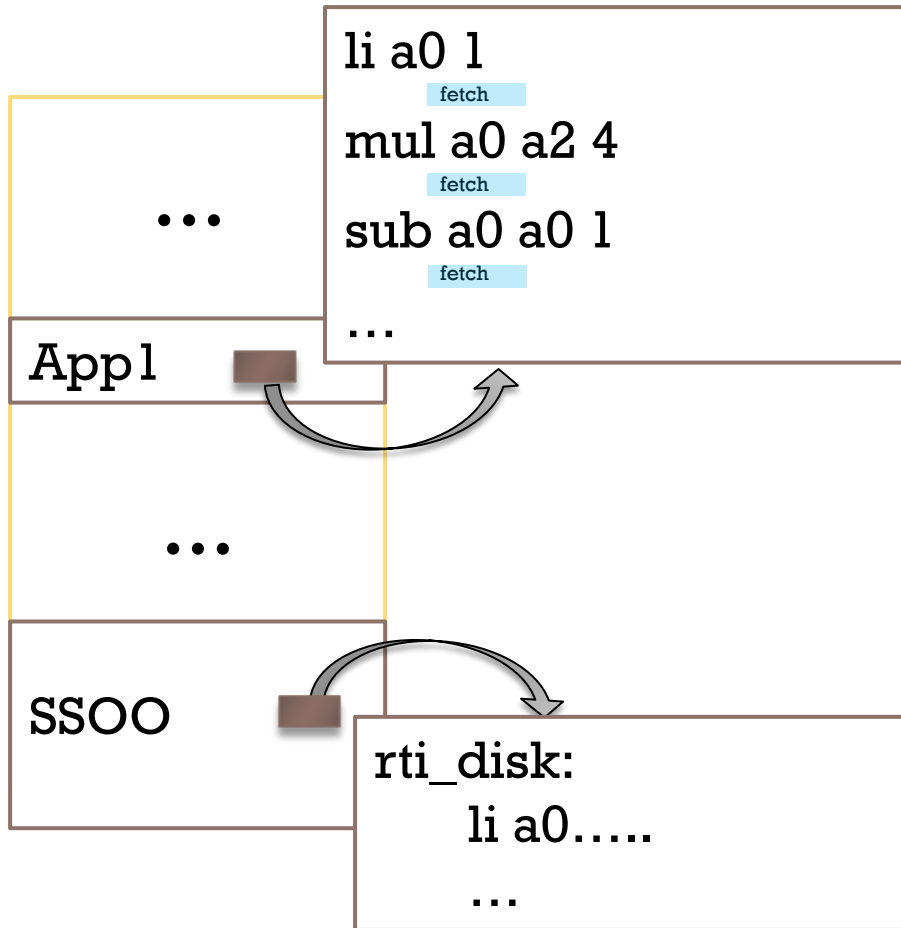


- ▶ Señal que llega a la U.C. y que rompe la secuencia normal de ejecución:
 - ▶ Se pausa la ejecución del programa actual y se transfiere la ejecución a otro programa que atiende la interrupción (ISR).
 - ▶ Al terminar el ISR la ejecución del programa se reanuda.
- ▶ Ejemplo de causas:
 - ▶ Cuando un periférico solicita la atención del procesador,
 - ▶ Cuando ocurre un error en la ejecución de la instrucción,
 - ▶ Etc.

Clasificación de las interrupciones

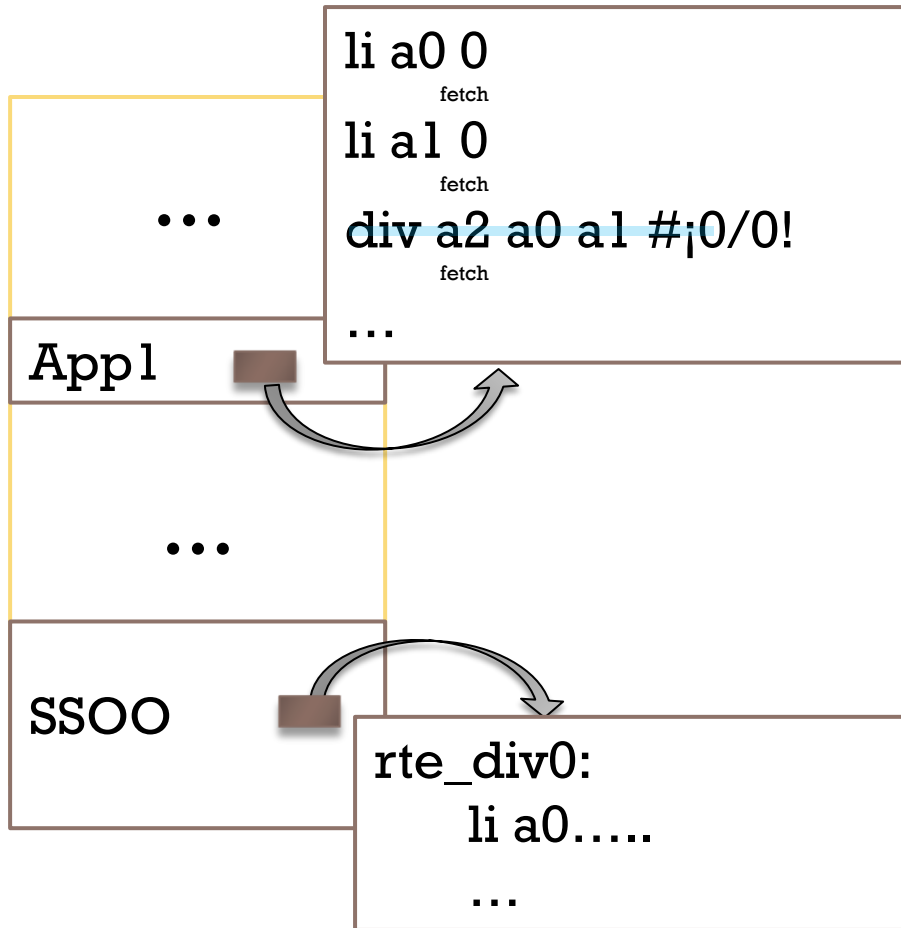
- ▶ **Excepciones hardware *síncronas***
 - ▶ Cuando un *error* ocurre *en la ejecución* de la *instrucción en curso*: División por cero, acceso a una posición de memoria ilegal, etc.
- ▶ **Excepciones hardware *asíncronas***
 - ▶ Fallos o *errores* en el hardware *no relacionados* con la *instrucción en curso*: impresora sin papel, etc.
- ▶ **Interrupciones externas**
 - ▶ Cuando un periférico precisa de atención por parte de la CPU: periféricos, interrupción del reloj
- ▶ **Llamadas al sistema**
 - ▶ Petición de servicio del sistema operativo
 - ▶ Instrucciones máquina especiales que generan una interrupción para activar al sistema operativo

Excepciones hardware asíncronas e Interrupciones externas



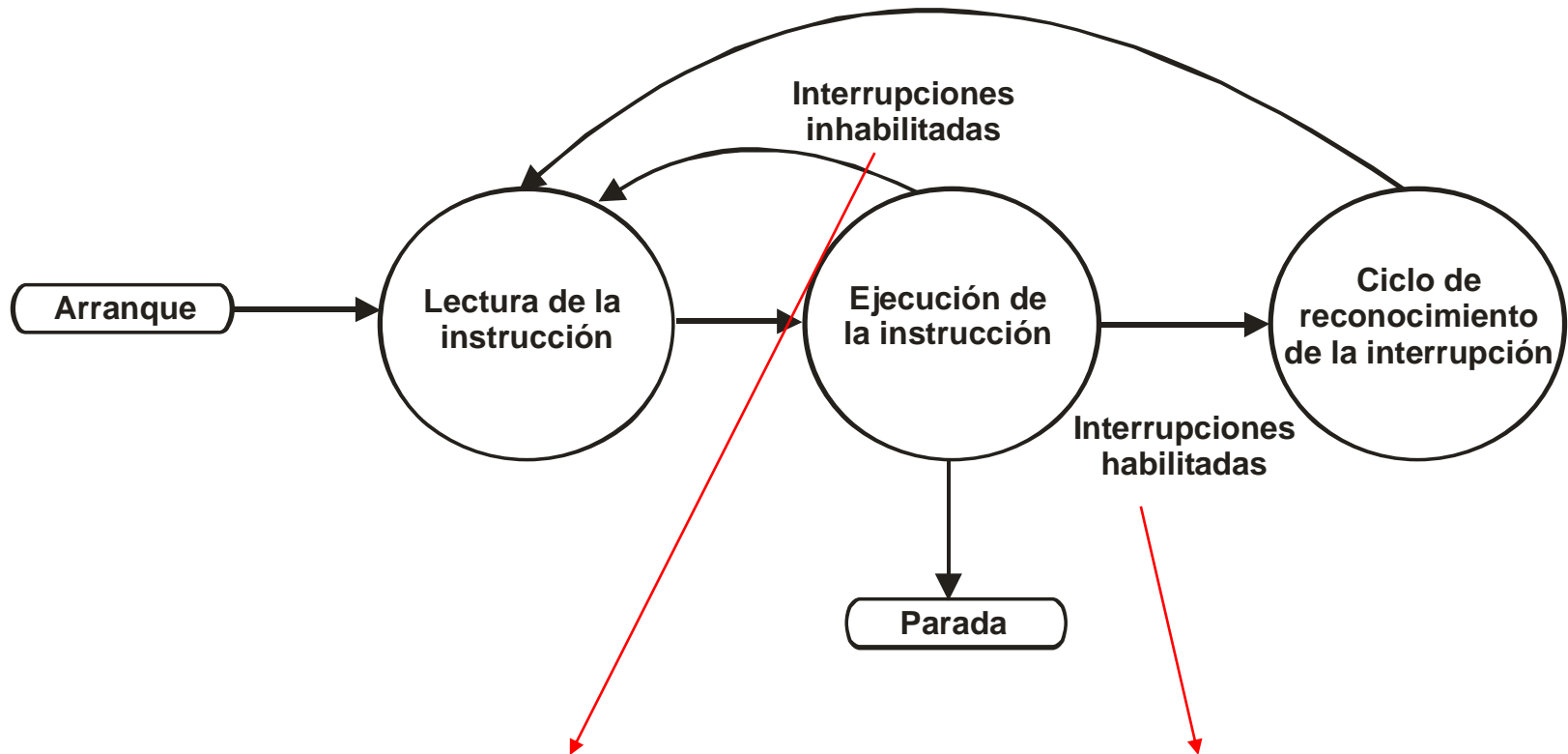
- ▶ Originan una ruptura de secuencia no programada
 - ▶ **Antes del ciclo de fetch de la siguiente instrucción, ver si hay interrupción pendiente, y si la hay...**
 - ▶ ...Bifurcación a subrutina del S.O. que la trata
- ▶ Posteriormente, restituye el estado y devuelve el control al programa interrumpido.
- **Causa asíncrona a la ejecución del programa en curso**
 - ▶ Atención a periférico
 - ▶ Etc.

Excepciones hardware síncronas



- ▶ Originan una ruptura de secuencia no programada
 - ▶ **Dentro del microprograma de la instrucción en curso...**
 - ▶ ...Bifurcación a subrutina del S.O. que la trata
- ▶ Posteriormente, restituye el estado y devuelve el control al programa interrumpido **o finaliza su ejecución**
- **Causa síncrona a la ejecución del programa en curso**
 - ▶ División entre cero
 - ▶ Etc.

Activación de interrupción en el registro de estado



Se indica con un bit situado en el **registro de estado (I)**

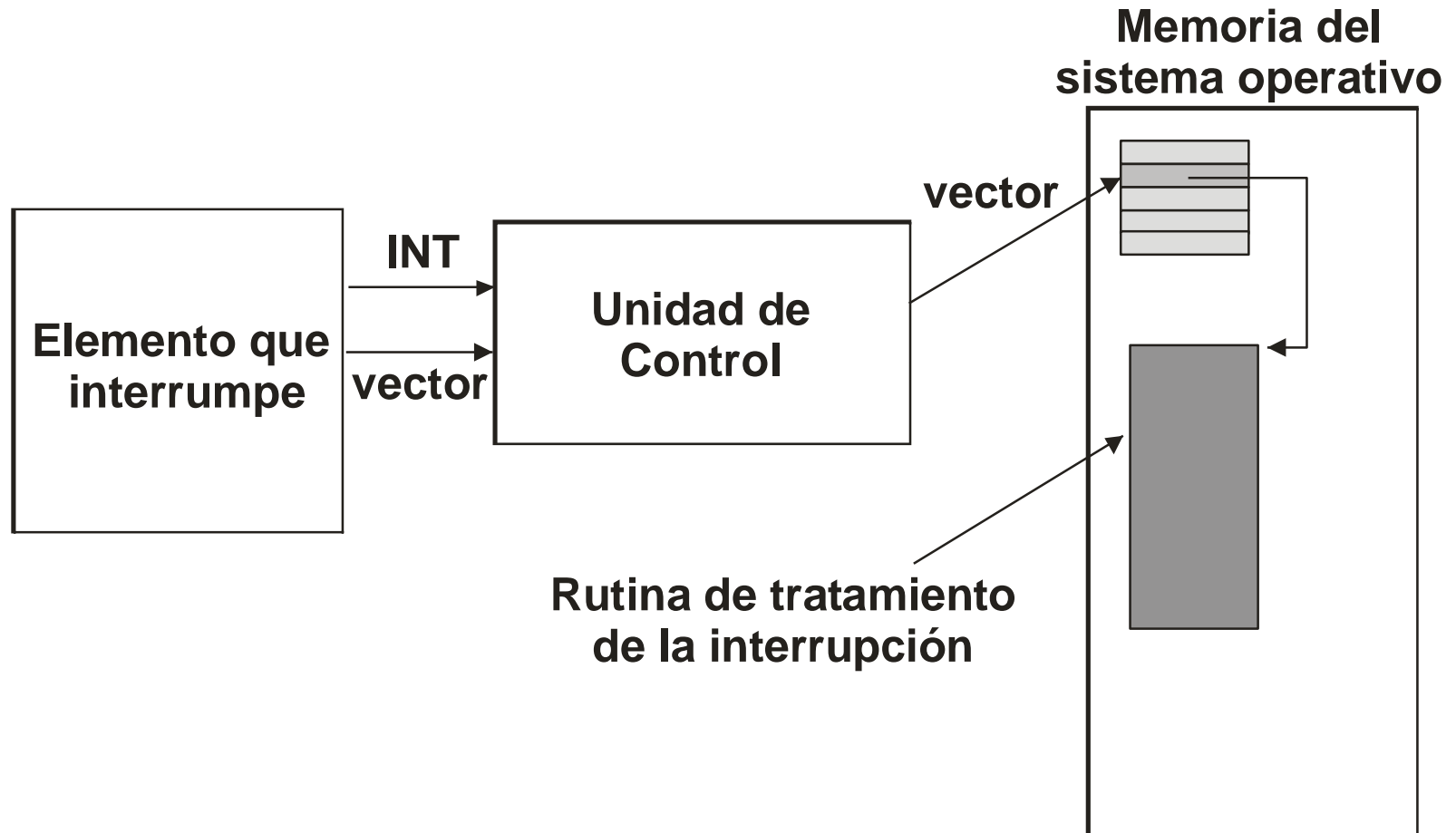
Ciclo de reconocimiento de la interrupción (CRI)

- ▶ Forma parte del microcódigo antes del ciclo de fetch
 - ▶ Trata especialmente las interrupciones asíncronas
- ▶ Estructura general del CRI:
 1. Comprueba se hay activada una señal de interrupción.
 2. Si está activada:
 1. Salva PC y RE (el contador de programa y el registro de estado)
 - Equivalent to “push pc, push sr”
 2. Pasa de modo usuario a modo núcleo
 - Equivalent to “SR.U = 0”
 3. Obtiene la dirección de la rutina de tratamiento de la interrupción
 - Equivalent to “isr_addr = Vector_interrupts[id_interrupt]”
 4. Almacena en el contador de programa la dirección obtenida (de esta forma la siguiente instrucción será la de la rutina de tratamiento)
 - Equivalent to “PC = isr_addr”

Rutina de tratamiento de la interrupción (RTI)

- ▶ Forma parte del código del sistema operativo
 - ▶ Hay una RTI por cada interrupción que pueda darse
- ▶ Estructura general de las RTI:
 1. Salva el resto de registros del procesador (que precise)
 2. Atiende la interrupción
 3. Restaura los registros del procesador guardados en (2)
 4. Ejecuta una instrucción máquina especial: RETI
 - ▶ Restaura el registro de estado del programa interrumpido (fijando de nuevo el modo del procesador a modo usuario)
 - ▶ Restaura el contador de programa (de forma que la siguiente instrucción es la del programa interrumpido).

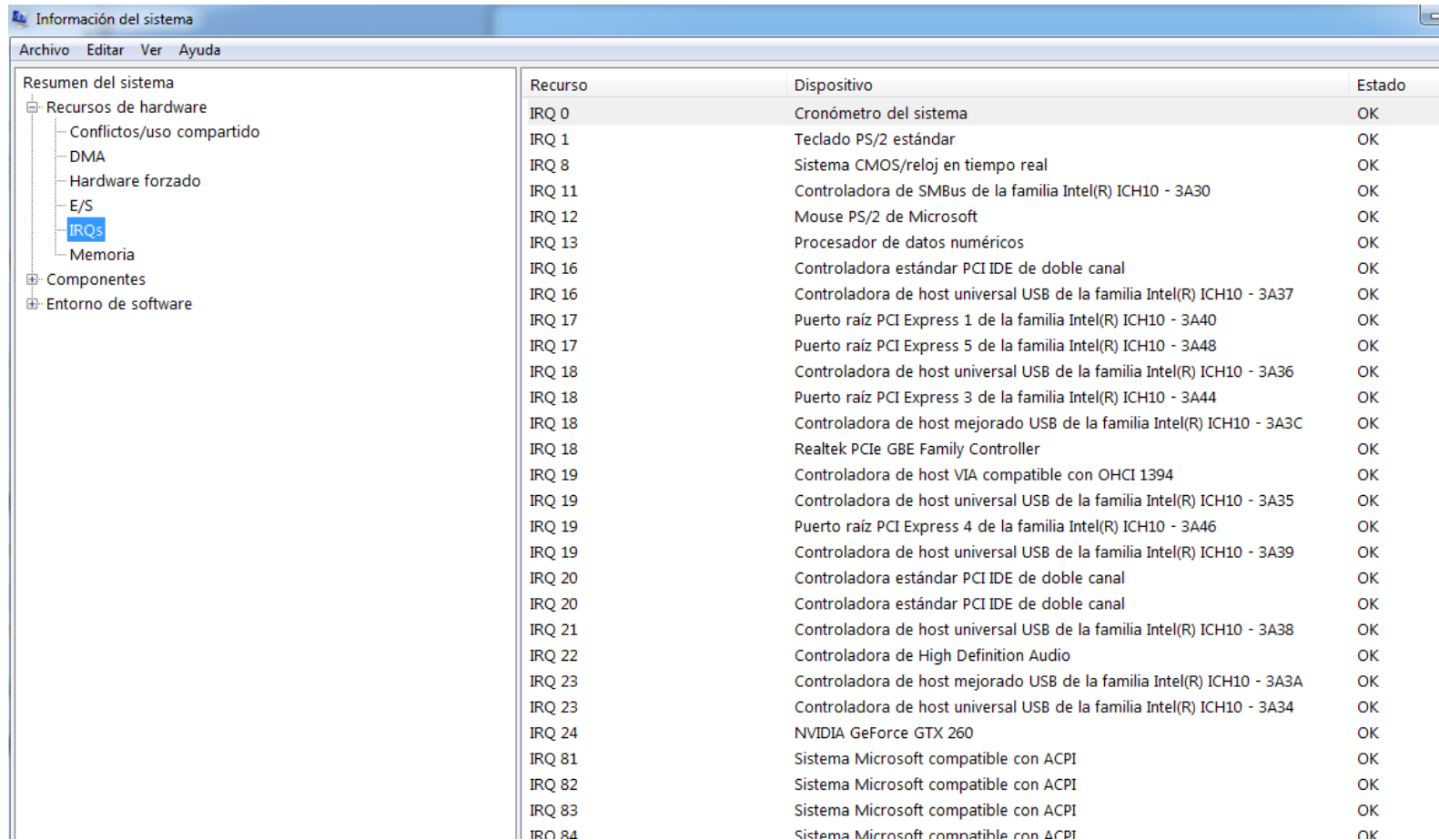
Interrupciones vectorizadas



Interrupciones vectorizadas

- ▶ El elemento que interrumpe suministra el **vector de interrupción**
- ▶ Este **vector** es un índice en una tabla que contiene la dirección de la rutina de tratamiento de la interrupción.
- ▶ La UC lee el contenido de esta entrada y carga el valor en el PC
- ▶ Cada sistema operativo rellena esta tabla con las direcciones de cada una de las rutinas de tratamiento, que son dependientes de cada sistema operativo.

Interrupciones en un PC con Windows



The screenshot shows the 'Información del sistema' window in Windows. The left sidebar has a tree view with 'Recursos de hardware' expanded, and 'IRQs' selected. The main area displays a table of system resources and their interrupt request (IRQ) numbers.

Recurso	Dispositivo	Estado
IRQ 0	Cronómetro del sistema	OK
IRQ 1	Teclado PS/2 estándar	OK
IRQ 8	Sistema CMOS/reloj en tiempo real	OK
IRQ 11	Controladora de SMBus de la familia Intel(R) ICH10 - 3A30	OK
IRQ 12	Mouse PS/2 de Microsoft	OK
IRQ 13	Procesador de datos numéricos	OK
IRQ 16	Controladora estándar PCI IDE de doble canal	OK
IRQ 16	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A37	OK
IRQ 17	Puerto raíz PCI Express 1 de la familia Intel(R) ICH10 - 3A40	OK
IRQ 17	Puerto raíz PCI Express 5 de la familia Intel(R) ICH10 - 3A48	OK
IRQ 18	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A36	OK
IRQ 18	Puerto raíz PCI Express 3 de la familia Intel(R) ICH10 - 3A44	OK
IRQ 18	Controladora de host mejorado USB de la familia Intel(R) ICH10 - 3A3C	OK
IRQ 18	Realtek PCIe GBE Family Controller	OK
IRQ 19	Controladora de host VIA compatible con OHCI 1394	OK
IRQ 19	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A35	OK
IRQ 19	Puerto raíz PCI Express 4 de la familia Intel(R) ICH10 - 3A46	OK
IRQ 19	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A39	OK
IRQ 20	Controladora estándar PCI IDE de doble canal	OK
IRQ 20	Controladora estándar PCI IDE de doble canal	OK
IRQ 21	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A38	OK
IRQ 22	Controladora de High Definition Audio	OK
IRQ 23	Controladora de host mejorado USB de la familia Intel(R) ICH10 - 3A3A	OK
IRQ 23	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A34	OK
IRQ 24	NVIDIA GeForce GTX 260	OK
IRQ 81	Sistema Microsoft compatible con ACPI	OK
IRQ 82	Sistema Microsoft compatible con ACPI	OK
IRQ 83	Sistema Microsoft compatible con ACPI	OK
IRQ 84	Sistema Microsoft compatible con ACPI	OK

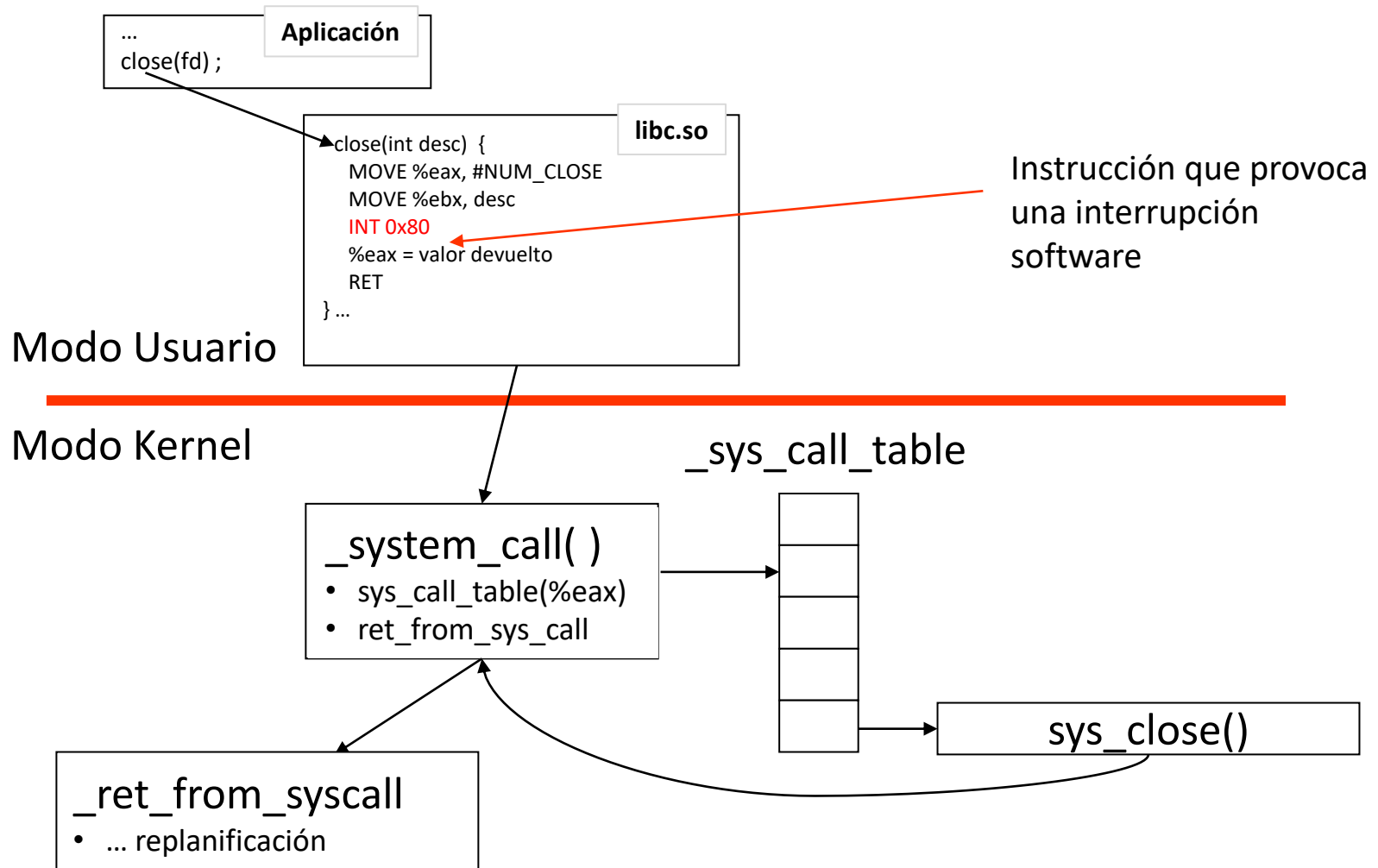
Interrupciones por software.

Llamadas al sistema y sistemas operativos

- ▶ El mecanismo de llamadas al sistema es el que permite que los programas de usuario puedan solicitar los servicios que ofrece el sistema operativo
 - ▶ Cargar programas en memoria para su ejecución
 - ▶ Acceso a los dispositivos periféricos
 - ▶ Etc.
- ▶ Similar a las llamadas al sistema que ofrece el simulador CREATOR
 - ▶ Hay ejemplos en WepSIM que muestran cómo internamente se puede implementar las llamadas al sistema

Interrupciones software

Llamadas al sistema (ejemplo: Linux)



Interrupciones del reloj y sistemas operativos

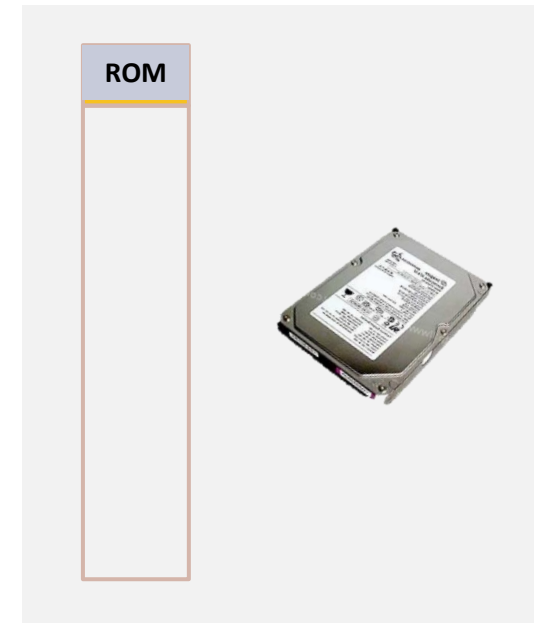
- ▶ La señal que gobierna la ejecución de las instrucciones máquina se divide mediante un divisor de frecuencia para generar una interrupción externa cada cierto intervalo de tiempo (pocos milisegundos)
- ▶ Estas **interrupciones de reloj** o tics son interrupciones periódicas que permite que el sistema operativo entre a ejecutar de forma periódica evitando que un programa de usuario monopolice la CPU
 - ▶ Permite alternar la ejecución de diversos programas en un sistema dado la apariencia de ejecución simultánea
 - ▶ Cada vez que llega una interrupción de reloj se suspende al programa y se salta al sistema operativo que ejecuta el **planificador** para decidir el **siguiente** programa a ejecutar

Contenido

1. Elementos de un computador
2. Organización del procesador
3. La unidad de control
4. Ejecución de instrucciones
5. Diseño de la unidad de control
6. Modos de ejecución
7. Interrupciones
8. Arranque de un computador
9. Prestaciones y paralelismo

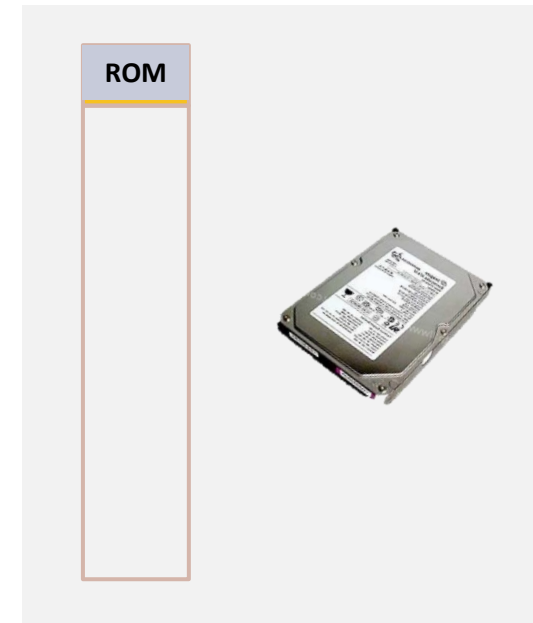
Arranque del computador

- ▶ El *Reset* carga en los registros sus valores predefinidos
 - ▶ $PC \leftarrow$ dirección de arranque del **programa iniciador** (en memoria ROM)



Arranque del computador

- ▶ El Reset carga en los registros sus valores predefinidos
 - ▶ PC ← dirección de arranque del programa iniciador (en memoria ROM)
- ▶ Se ejecuta el programa iniciador
 - ▶ Test del sistema (POST)



```
Award Modular BIOS v6.00PG, An Energy Star Ally
Copyright (C) 1984-2007, Award Software, Inc.

Intel X38 BIOS for X38-DQ6 F4

Main Processor : Intel(R) Core(TM)2 Extreme CPU X9650 @ 4.00GHz(333x12)
<CPUID:0676 Patch ID:0000>
Memory Testing : 2096064K OK

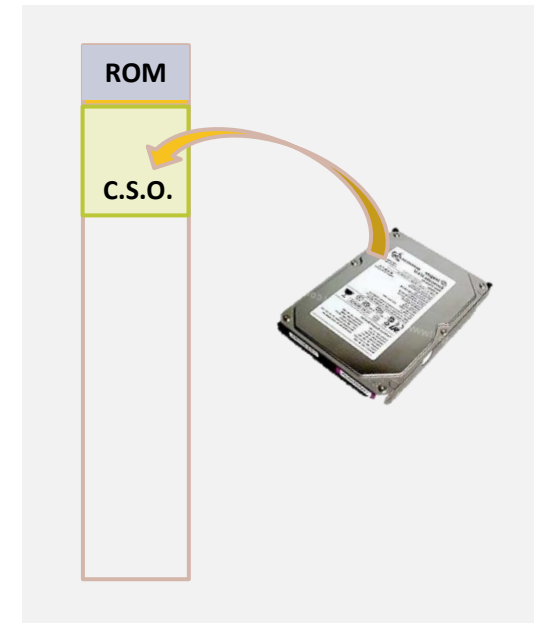
Memory Runs at Dual Channel Interleaved
IDE Channel 0 Slave : WDC WD3200AAJS-00RYA0 12.01B01
IDE Channel 1 Slave : WDC WD3200AAJS-00RYA0 12.01B01

Detecting IDE drives ...
IDE Channel 4 Master : None
IDE Channel 4 Slave : None
IDE Channel 5 Master : None
IDE Channel 5 Slave : None

<DEL>:BIOS Setup <F9>:XpressRecoveryZ <F12>:Boot Menu <End>:Quit
09/19/2007-X38-ICH9-6A79060QC-00
```

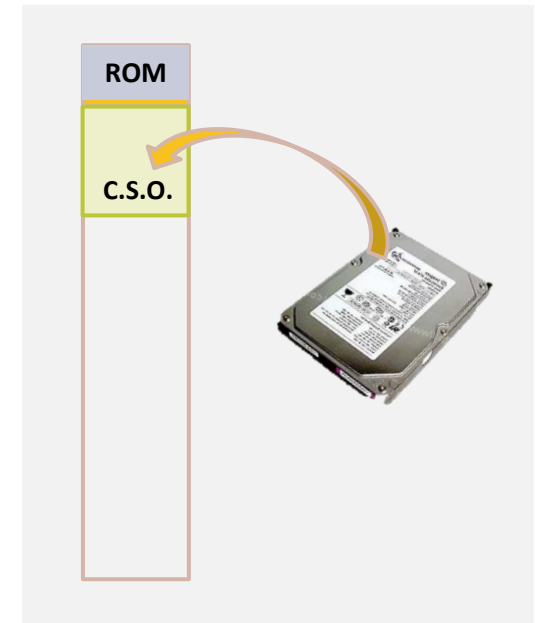
Arranque del computador

- ▶ El *Reset* carga en los registros sus valores predefinidos
 - ▶ PC ← dirección de arranque del **programa iniciador** (en memoria ROM)
- ▶ Se ejecuta el **programa iniciador**
 - ▶ Test del sistema (POST)
 - ▶ Carga en memoria el **cargador del sistema operativo (MBR)**



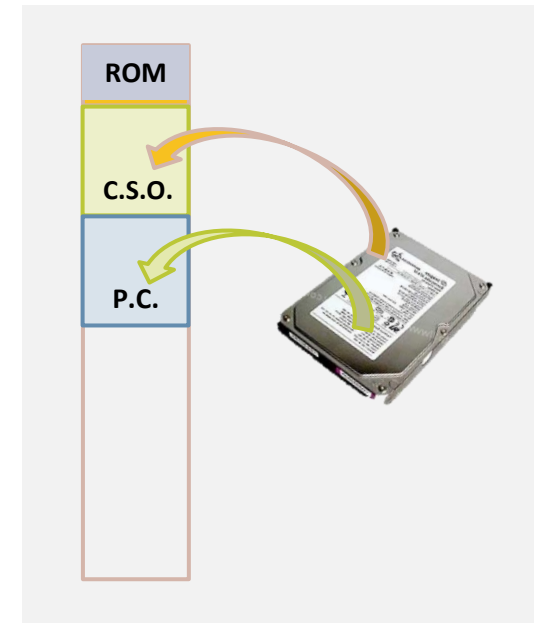
Arranque del computador

- ▶ El *Reset* carga en los registros sus valores predefinidos
 - ▶ PC ← dirección de arranque del **programa iniciador** (en memoria ROM)
- ▶ Se ejecuta el **programa iniciador**
 - ▶ Test del sistema (POST)
 - ▶ Carga en memoria el **cargador del sistema operativo (MBR)**
- ▶ Se ejecuta el **cargador del sistema operativo**
 - ▶ Establece opciones de arranque



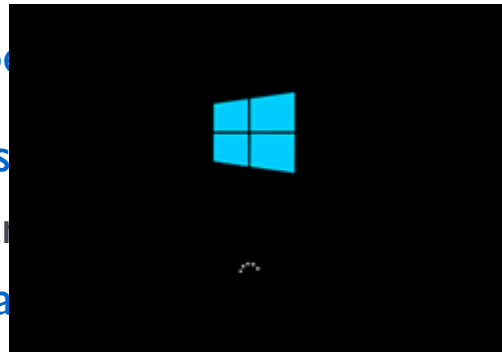
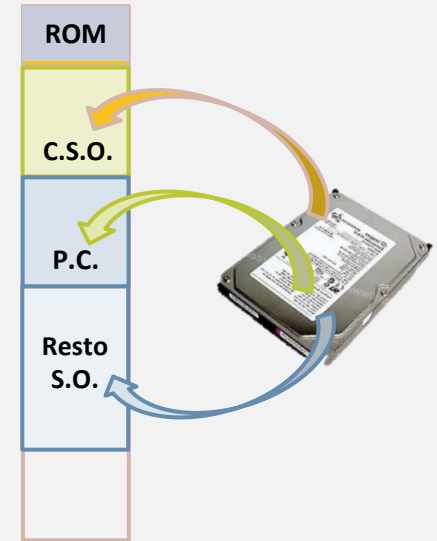
Arranque del computador

- ▶ El *Reset* carga en los registros sus valores predefinidos
 - ▶ PC ← dirección de arranque del **programa iniciador** (en memoria ROM)
- ▶ Se ejecuta el **programa iniciador**
 - ▶ Test del sistema (POST)
 - ▶ Carga en memoria el **cargador del sistema operativo (MBR)**
- ▶ Se ejecuta el **cargador del sistema operativo**
 - ▶ Establece opciones de arranque
 - ▶ Carga el **programa de carga**



Arranque del computador

- ▶ El **Reset** carga en los registros sus valores predefinidos
 - ▶ PC ← dirección de arranque del **programa iniciador** (en memoria ROM)
- ▶ Se ejecuta el **programa iniciador**
 - ▶ Test del sistema (POST)
 - ▶ Carga en memoria el **cargador del sistema operativo**
- ▶ Se ejecuta el **cargador del sistema operativo**
 - ▶ Establece opciones de arranque
 - ▶ Carga el **programa de carga**
- ▶ Se ejecuta el **programa de carga**
 - ▶ Establece estado inicial para el S.O.
 - ▶ Carga el sistema operativo y lo ejecuta

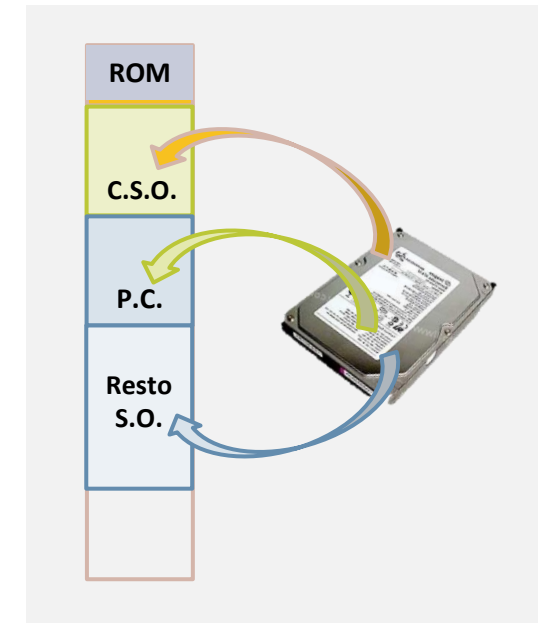


Una imagen de la pantalla de arranque de Linux, mostrando el logo de Tux (el pingüino) y un texto de configuración del sistema. El texto incluye comandos como 'Configuring ISA PNP', 'Setting system time from the hardware clock (localtime)', 'Using /etc/random-seed to initialize /dev/urandom.', 'Initializing basic system settings ...', 'Updating shared libraries', 'Setting hostname: enpc23.murdoch.edu.au', 'INIT: Entering runlevel: 4', 'rc.M ==> Going multiuser ...', 'Starting system logger ...', 'Initialising advanced hardware', 'Setting up modules ...', 'Initialising network', 'Setting up localhost ...', 'Setting up inet1 ...', 'Setting up route ...', 'Setting up fancy console and GUI', 'Loading fc-cache ...', 'rc.v1init ==> Going to runlevel 4', 'Starting services of runlevel 4', 'Starting dnsmasq ...', '==> rc.X Going to multiuser GUI mode ...', 'XFree86 Display Manager', 'Framebuffer /dev/fb0 is 307200 bytes.', 'Grabbing 640x480 ...'. Las palabras 'OK' aparecen al final de varias líneas.

Arranque del computador

resumen

- ▶ El *Reset* carga en los registros sus valores predefinidos
 - ▶ PC ← dirección de arranque del **programa iniciador** (en memoria ROM)
- ▶ Se ejecuta el **programa iniciador**
 - ▶ Test del sistema (POST)
 - ▶ Carga en memoria el **cargador del sistema operativo (MBR)**
- ▶ Se ejecuta el **cargador del sistema operativo**
 - ▶ Establece opciones de arranque
 - ▶ Carga el **programa de carga**
- ▶ Se ejecuta el **programa de carga**
 - ▶ Establece estado inicial para el S.O.
 - ▶ Carga el sistema operativo y lo ejecuta



Contenido

1. Elementos de un computador
2. Organización del procesador
3. La unidad de control
4. Ejecución de instrucciones
5. Diseño de la unidad de control
6. Modos de ejecución
7. Interrupciones
8. Arranque de un computador
9. Prestaciones y paralelismo

Tiempo de ejecución de un programa

Iron law of processor performance

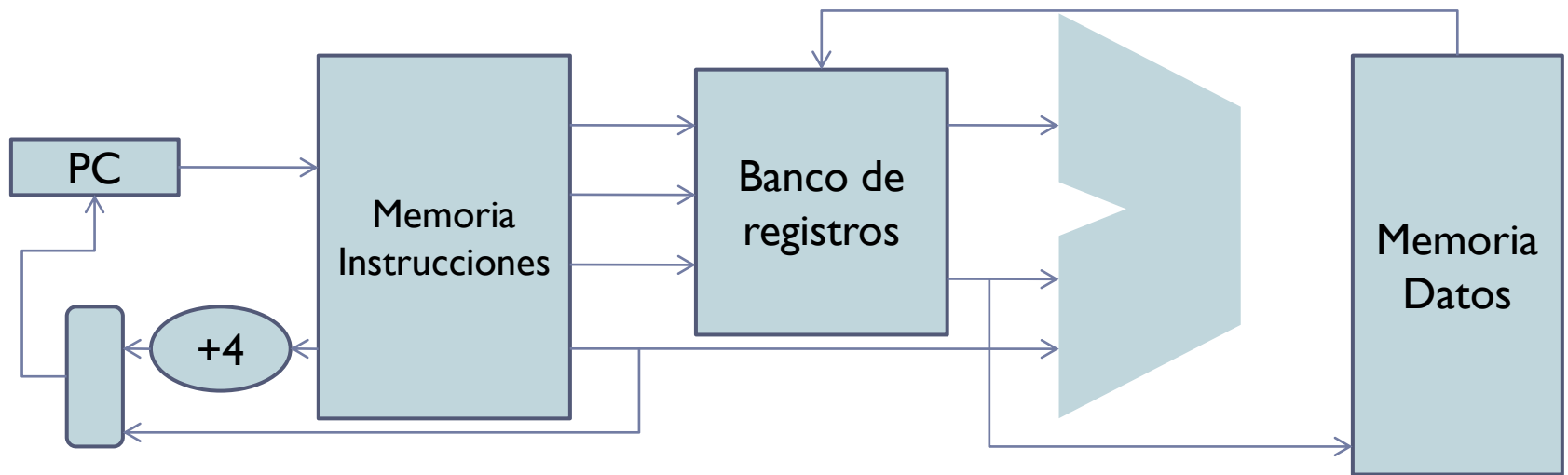
$$\text{Tiempo}_{\text{ejecución}} = \text{NI} \times \text{CPI} \times t_{\text{ciclo_CPU}} + \text{NI} \times \text{AMI} \times t_{\text{ciclo_mem}}$$

- ▶ **NI** es el número de instrucciones máquina del programa
- ▶ **CPI** es el número medio de ciclos de reloj necesario para ejecutar una instrucción
- ▶ **$t_{\text{ciclo_CPI}}$** es el tiempo que dura el ciclo de reloj del procesador
- ▶ **AMI** es el número medio de accesos a memoria por instrucción
- ▶ **$t_{\text{ciclo_mem}}$** es el tiempo de un acceso a memoria

Factores que afecta al tiempo de ejecución

	NI	CPI	$t_{\text{ciclo_CPI}}$	AMI	$t_{\text{ciclo_mem}}$
Programa	✓			✓	
Compilador	✓	✓		✓	
Juego de instrucciones (ISA)	✓	✓	✓	✓	
Organización		✓	✓		✓
Tecnología			✓		✓

Modelo de procesador basado en camino de datos (sin bus interno)



Paralelismo a nivel de instrucción

- ▶ Procesamiento concurrente de varias instrucciones
- ▶ Combinación de elementos que trabajan en paralelo:
 - ▶ **Procesadores segmentados**: utilizan técnicas de pipeline para procesar varias instrucciones simultáneamente
 - ▶ **Procesadores superescalares**: procesador segmentado que puede ejecutar varias instrucciones en paralelo cada una de ellas en una unidad segmentada diferente
 - ▶ **Procesadores multicore**: procesador que combina dos o más procesadores independientes en un solo empaquetado

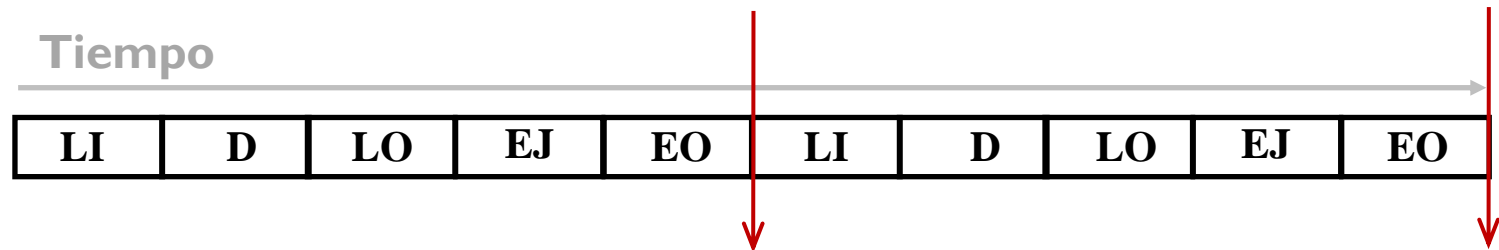
Segmentación de instrucciones



- ▶ Etapas de ejecución de una instrucción:
 - ▶ **LI**: Lectura de la instrucción e incremento del PC
 - ▶ **D**: Decodificación
 - ▶ **LO**: Lectura de Operandos
 - ▶ **Ej**: Ejecución de la instrucción
 - ▶ **EO**: Escritura de Operandos

Segmentación de instrucciones

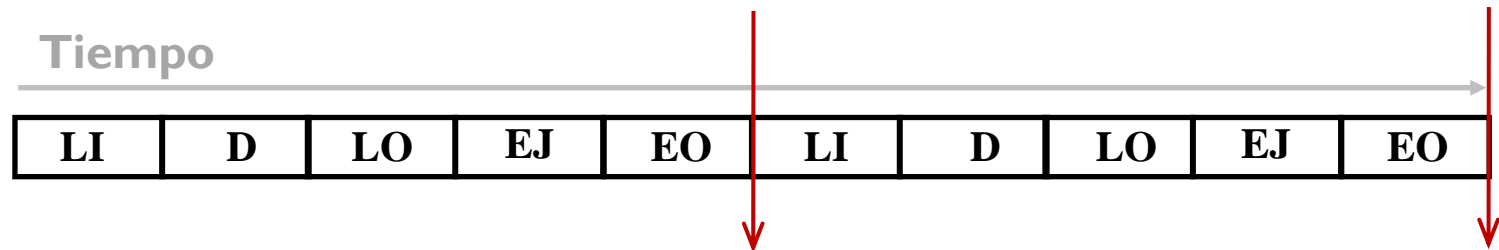
sin pipeline



- ▶ Etapas de ejecución de una instrucción:
 - ▶ **LI**: Lectura de la instrucción e incremento del PC
 - ▶ **D**: Decodificación
 - ▶ **LO**: Lectura de Operandos
 - ▶ **EJ**: Ejecución de la instrucción
 - ▶ **EO**: Escritura de Operandos

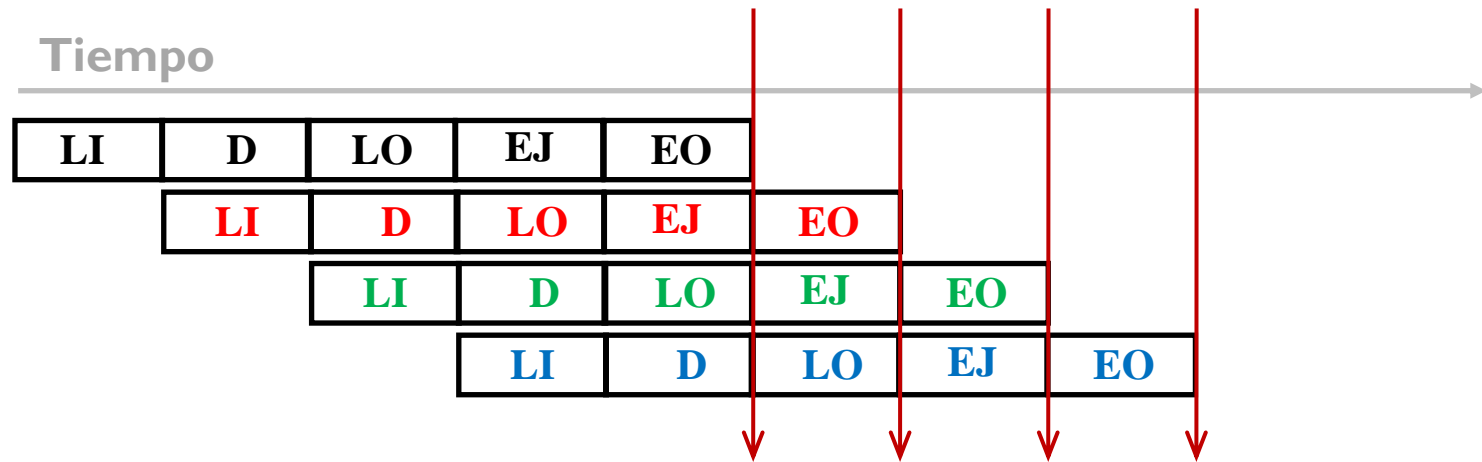
Segmentación de instrucciones

sin pipeline



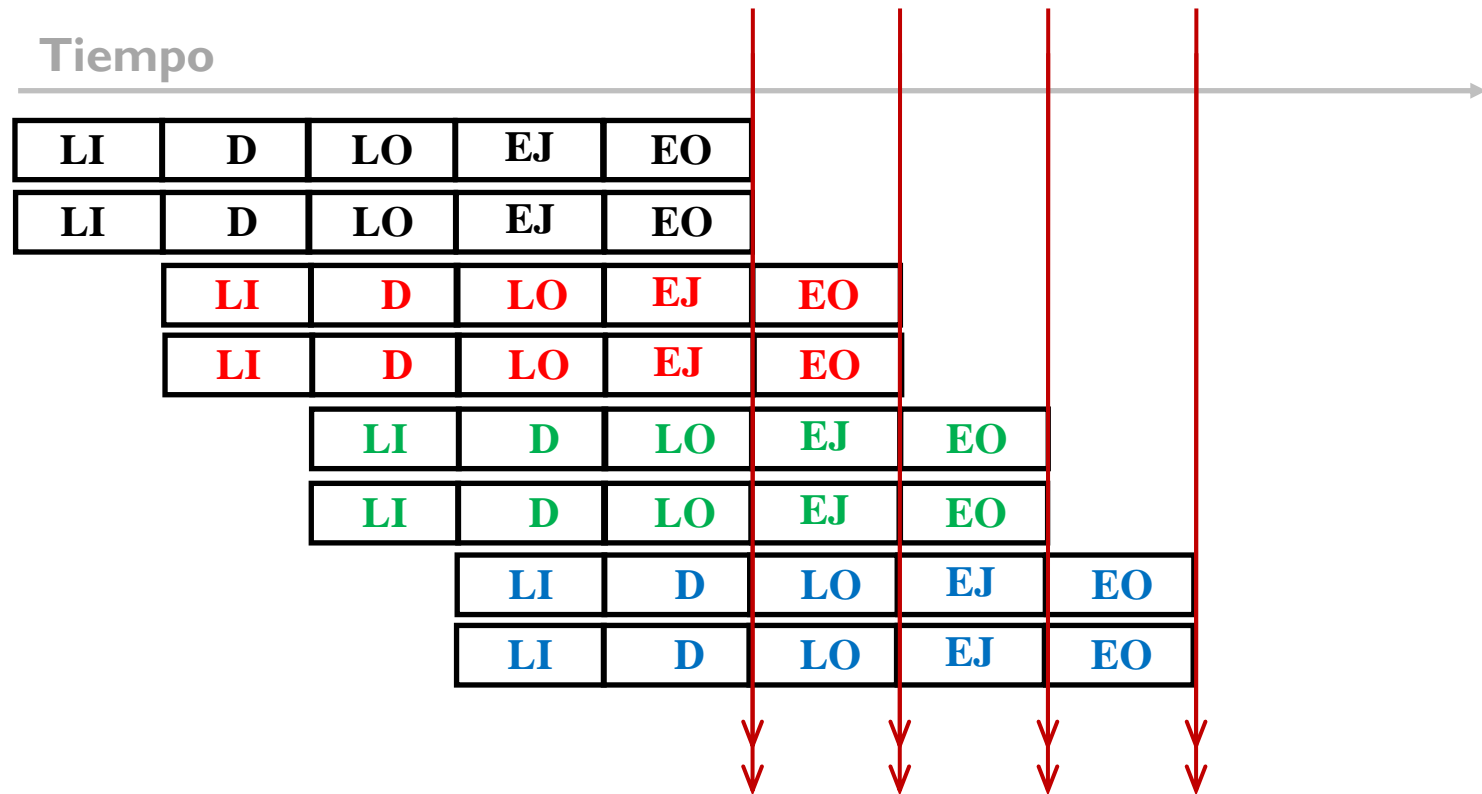
- ▶ Si cada fase dura N ciclos de reloj, entonces
 - ▶ Una instrucción se ejecuta en $5 \cdot N$ ciclos de reloj
 - ▶ Se ejecuta $1/5$ de instrucción cada N ciclos de reloj

Segmentación de instrucciones con pipeline



- ▶ Si cada fase dura N ciclos de reloj, entonces
 - ▶ Una instrucción se ejecuta en $5 \cdot N$ ciclos de reloj
 - ▶ Cada N ciclos de reloj termina 1 de instrucción

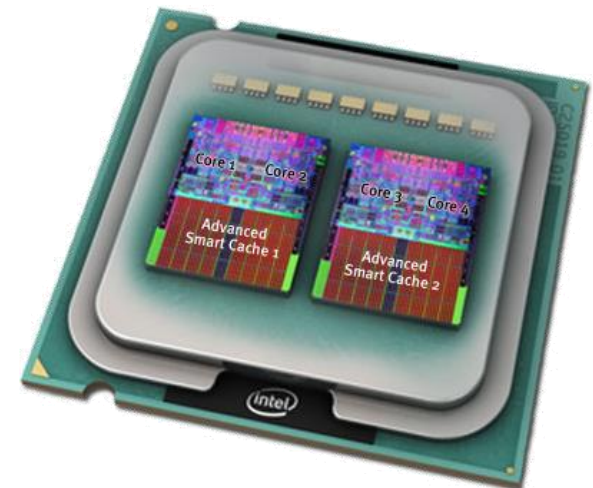
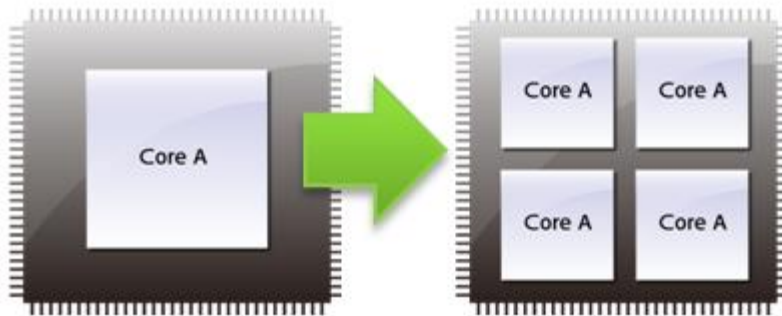
Superscalar



- Pipeline con varias unidades funcionales en paralelo

Multicore

- Múltiples procesadores en el mismo encapsulado



Grupo ARCOS

uc3m | Universidad **Carlos III** de Madrid

Tema 4 (II) El procesador

Estructura de Computadores
Grado en Ingeniería Informática