Grupo ARCOS

uc3m | Universidad Carlos III de Madrid

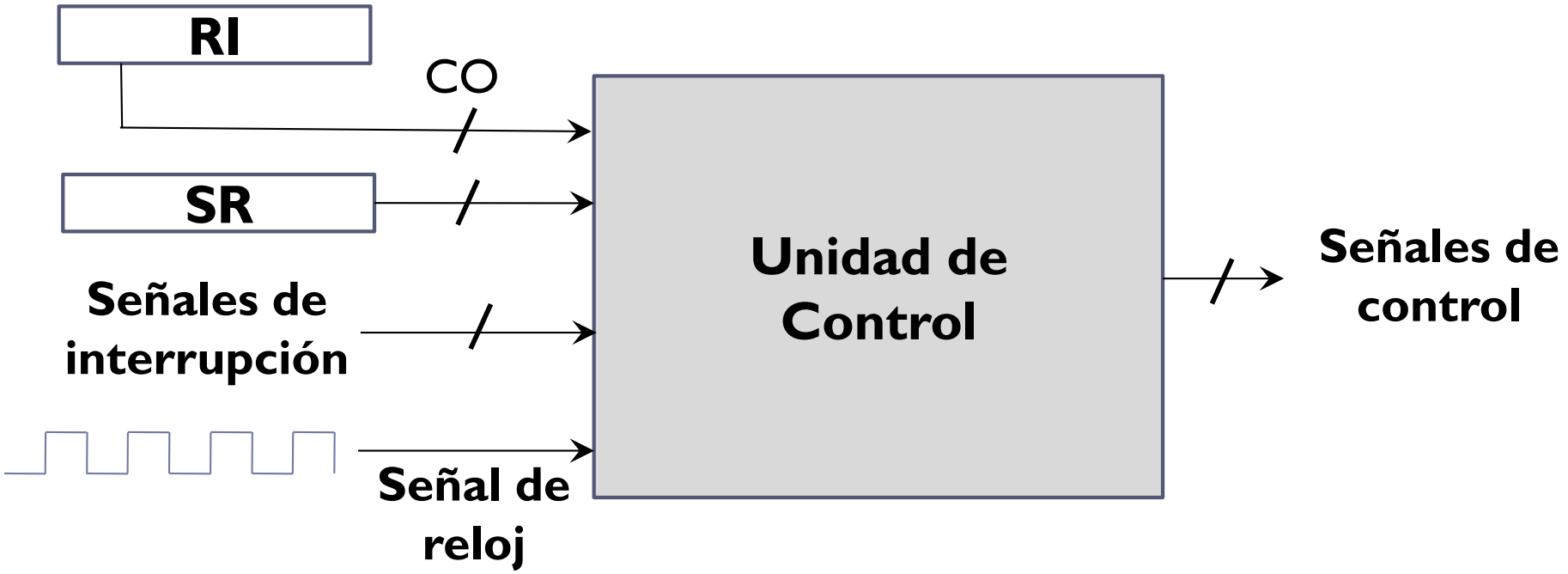
Tema 4 (II)
El procesador

Estructura de Computadores
Grado en Ingeniería Informática

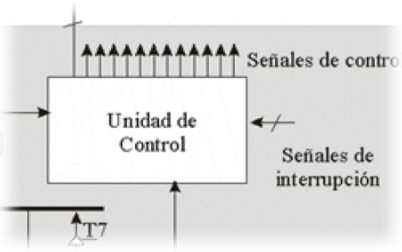
Contenidos

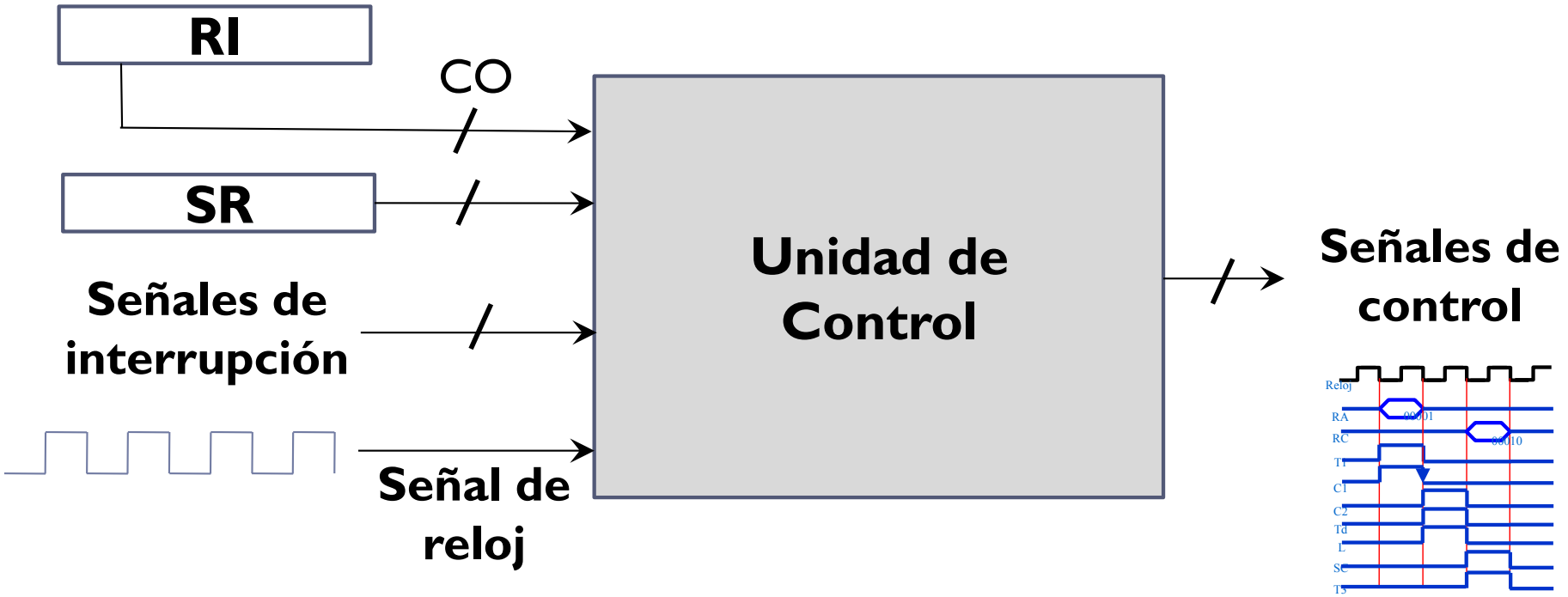
1. Elementos de un computador
2. Organización del procesador
3. La unidad de control
4. Ejecución de instrucciones
5. Modos de ejecución
6. Interrupciones
7. Diseño de la unidad de control
8. Arranque de un computador
9. Prestaciones y paralelismo

Unidad de control



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

ARCOS @ UC3M
Félix García-Carballeira, Alejandro Calderón Mateos

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

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

Diseño de la unidad de control

Instrucción

mv R0 R1

Secuencia de **operaciones elementales**

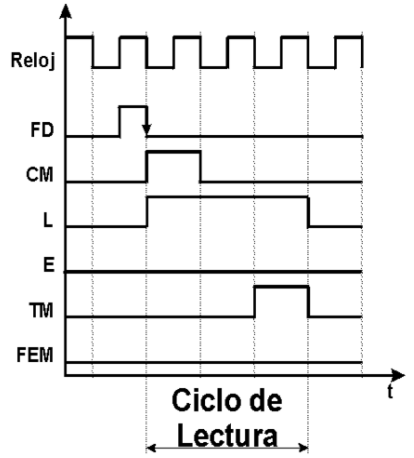
RI <- [PC]

PC++

decodificación

R0 <- R1

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



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

Técnicas de control

- Unidad de control cableada

- Unidad de control microprogramada

Ejemplo

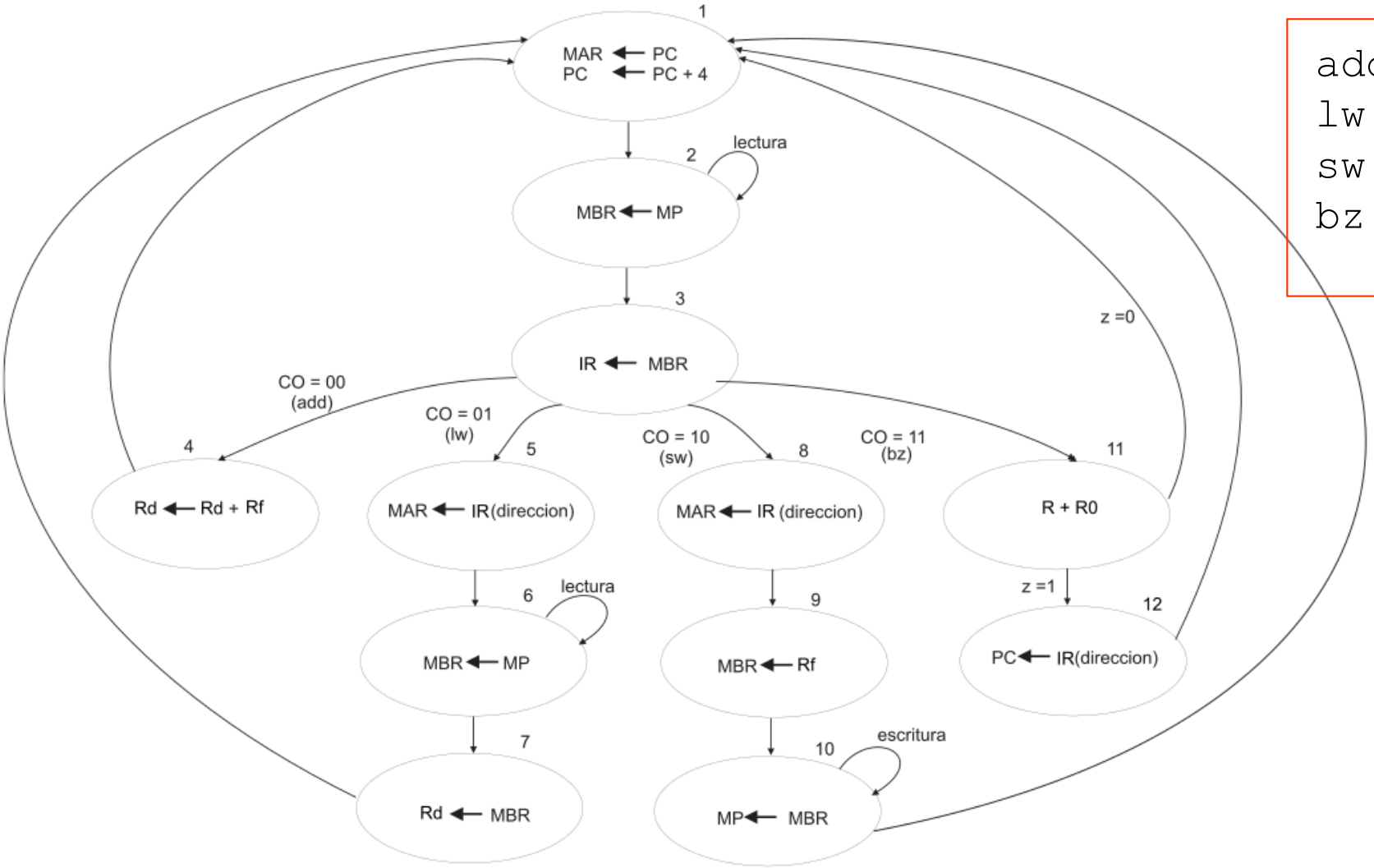
- Diseño de una unidad de control para un juego de 4 instrucciones máquina:

- 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`

Máquina de estados para el ejemplo

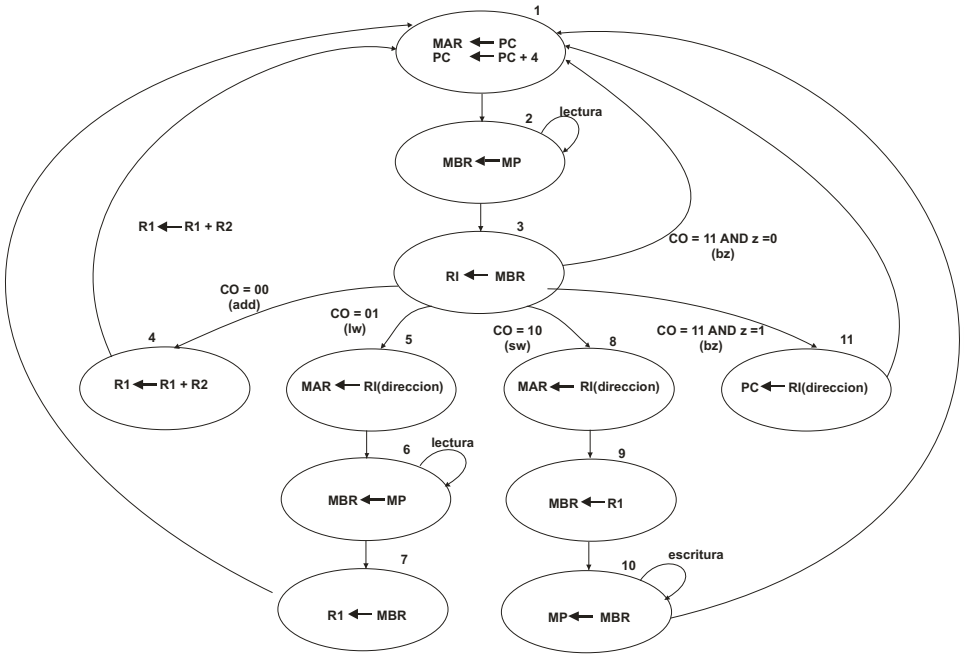
Ejemplo para un computador con solo 4 instruc. máquina

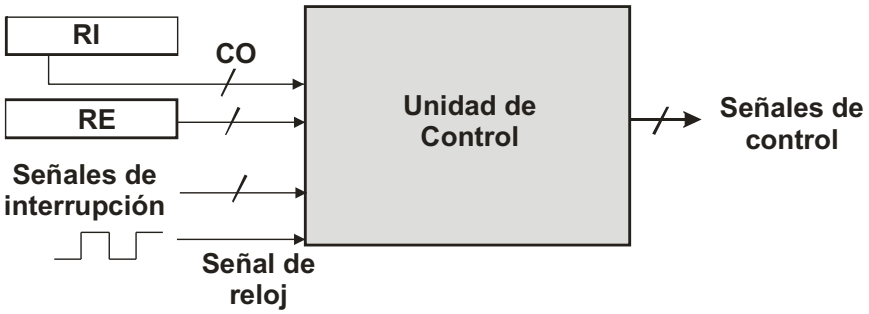
```
add rd, rf
lw rd, dir
sw Rf, dir
bz R, dir
```



Técnicas de control

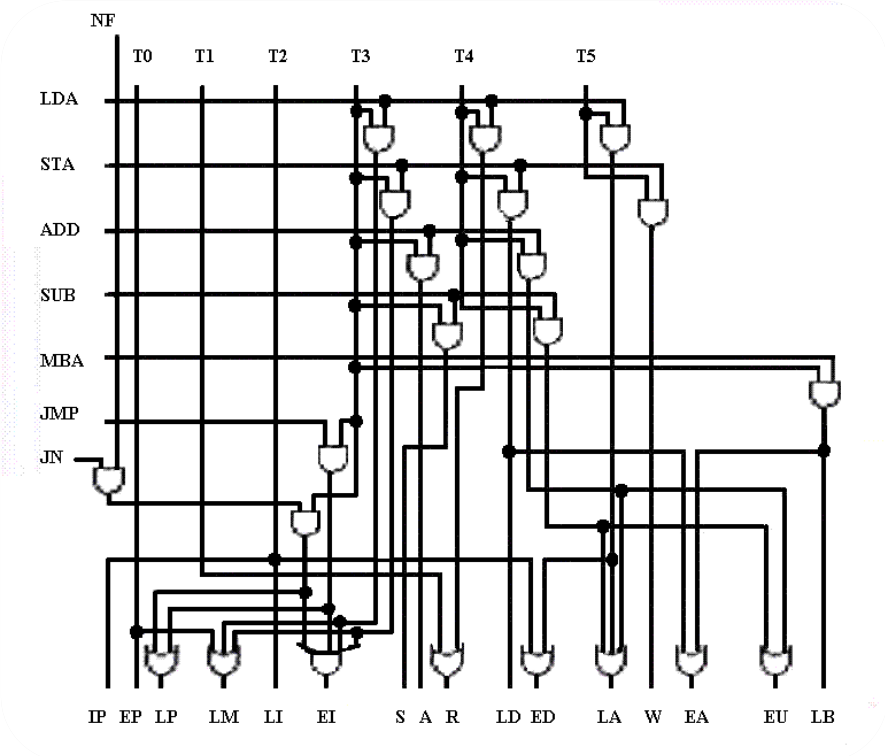
- **Dos técnicas** de diseñar y construir una unidad de control:
  a) **Lógica cableada**
  b) **Lógica almacenada (microprogramación)**





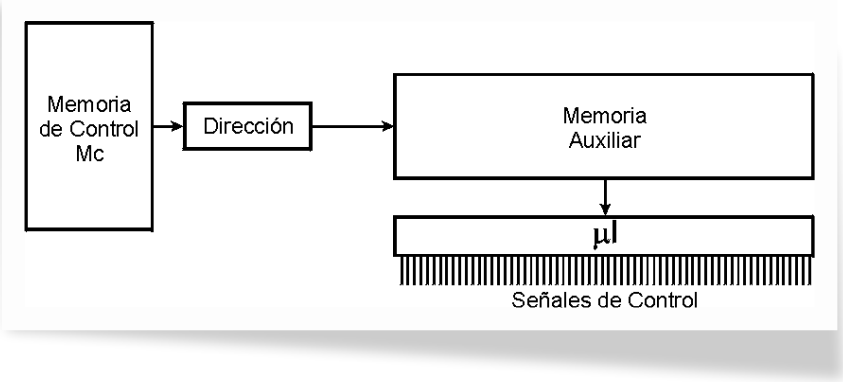
Unidad de control cableada

- 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)



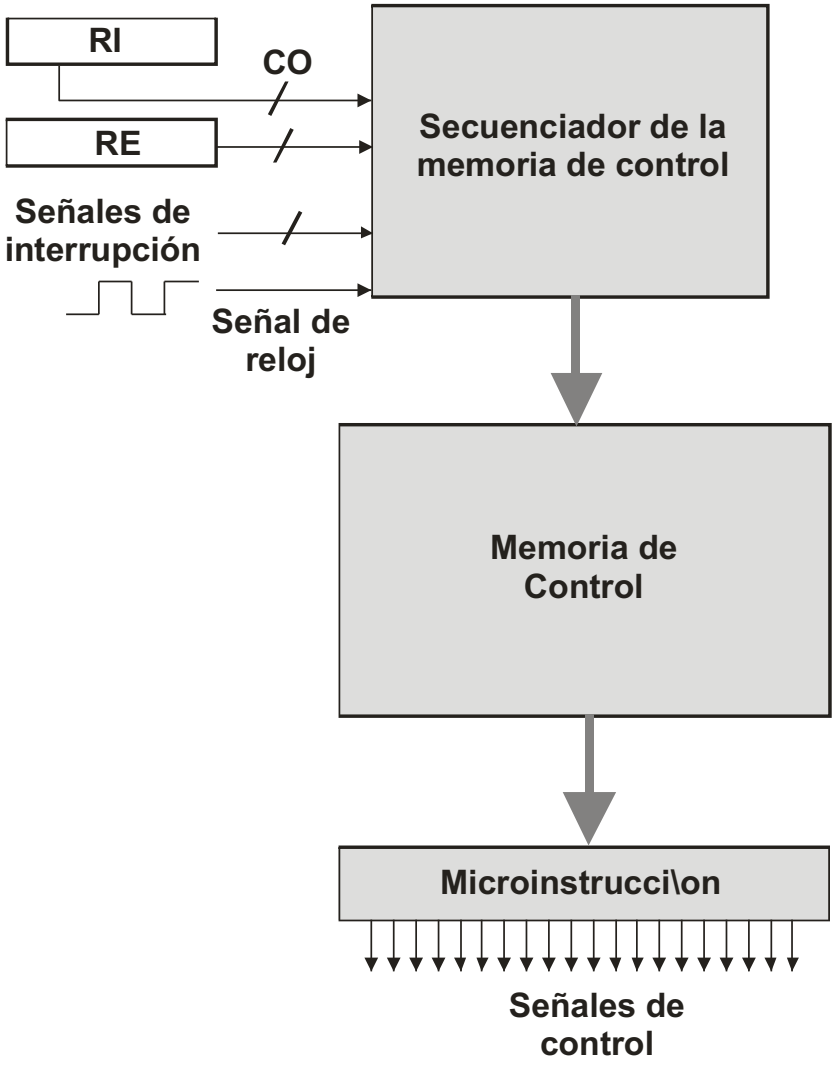
Unidad de control almacenada. Microprogramación

- 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 ⇒ difícil microcódigo

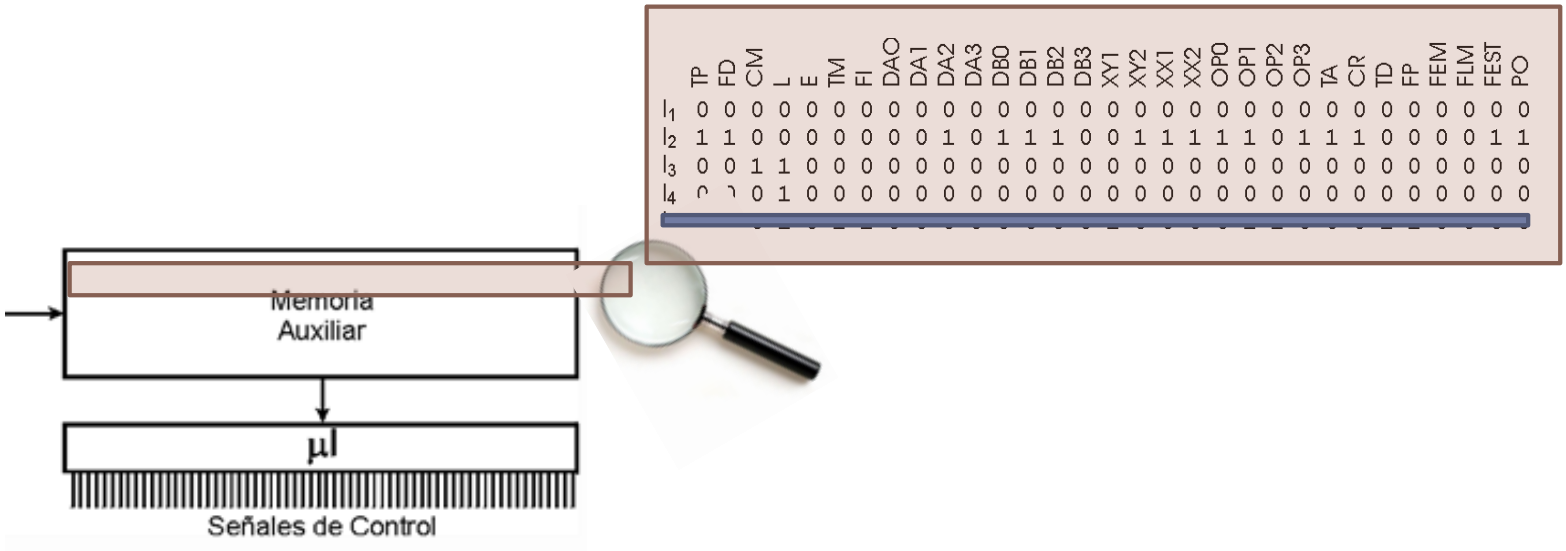
Estructura general de una unidad de control microprogramada



C1	C2	C3	C4	R5	C6	C7	C8	C9	C10	C11	Td	Ta	T1	T2	T3	T4	T5	T6	T7	T8	RA4	RA3	RA2	Ra1
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

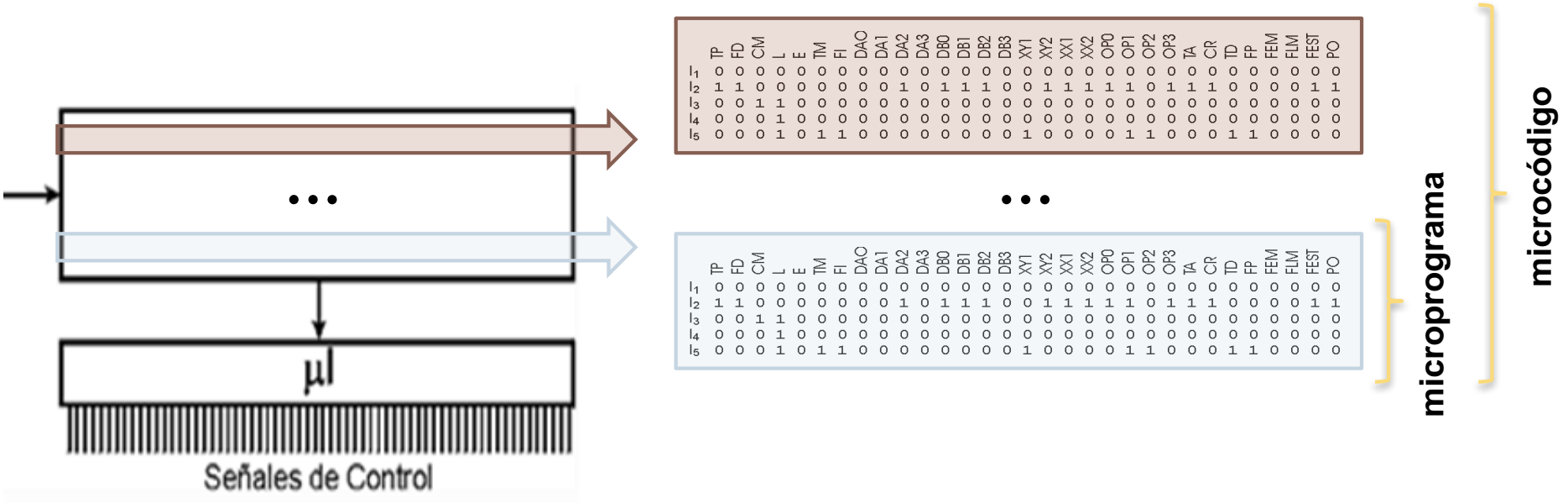
RB4	Rb3	RB2	RB1	RB0	RC4	RC3	RC2	RC1	RC0	SC	L	E	Cop3	Cop2	Cop1	Cop0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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



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

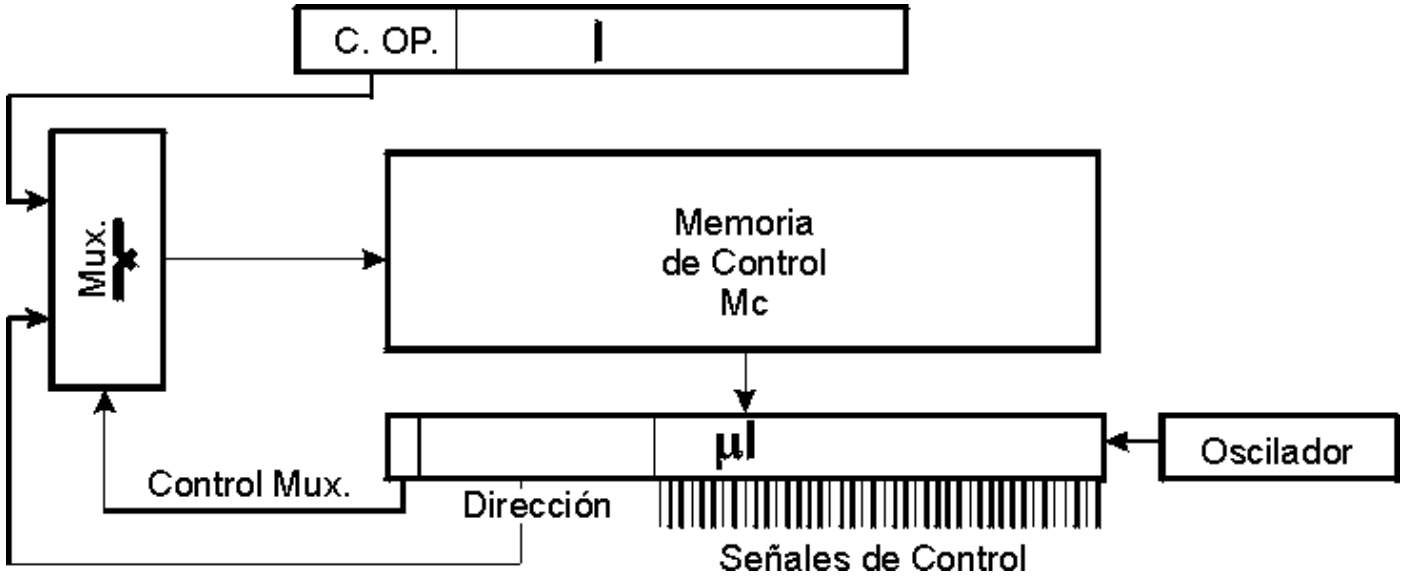
Contenido de la memoria de control

- FETCH: traer sig. Instrucción
  - Ciclo Reconocimiento Int.
  - IR<- Mem[PC], 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

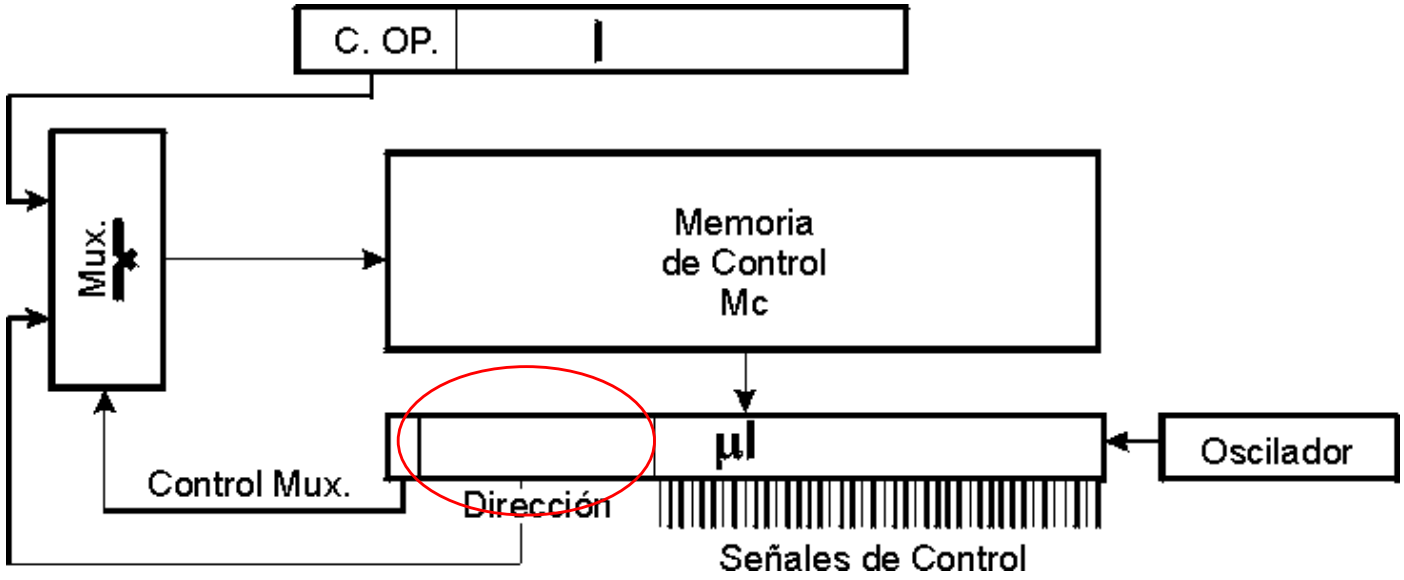
- Tres condiciones básicas:
  1. Memoria de control suficiente para almacenar todos los microprogramas correspondientes a todas las instrucciones.
  2. Procedimiento para asociar a cada instrucción su microprograma
     - Procedimiento que convierta el código de operación de la instrucción en la dirección 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



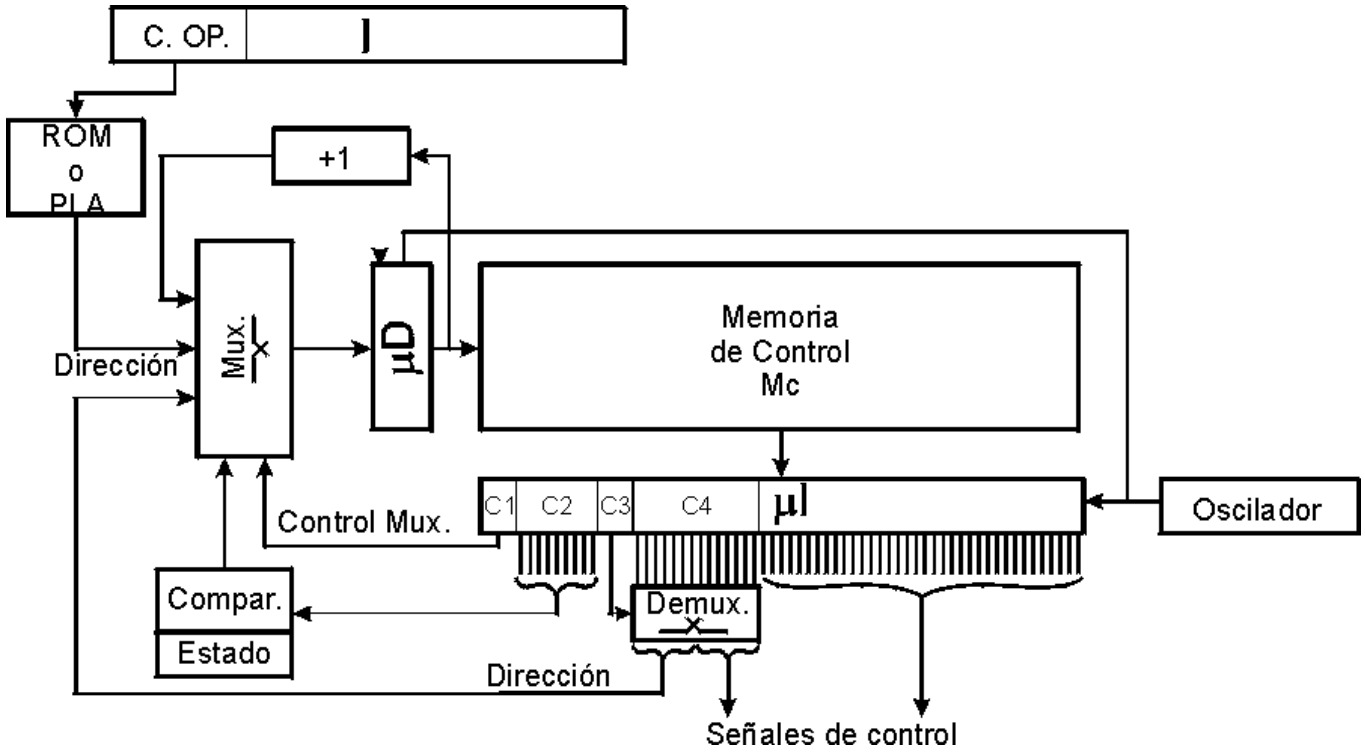
- Memoria de control guarda todos los µprogramas, donde cada µinstrucción proporciona la µdirección de la µinstrucción siguiente
- El CO representa la µDirección de la primera µinstrucción asociado a la instrucción máquina

Estructura de UC microprogramada con secuenciamiento explícito



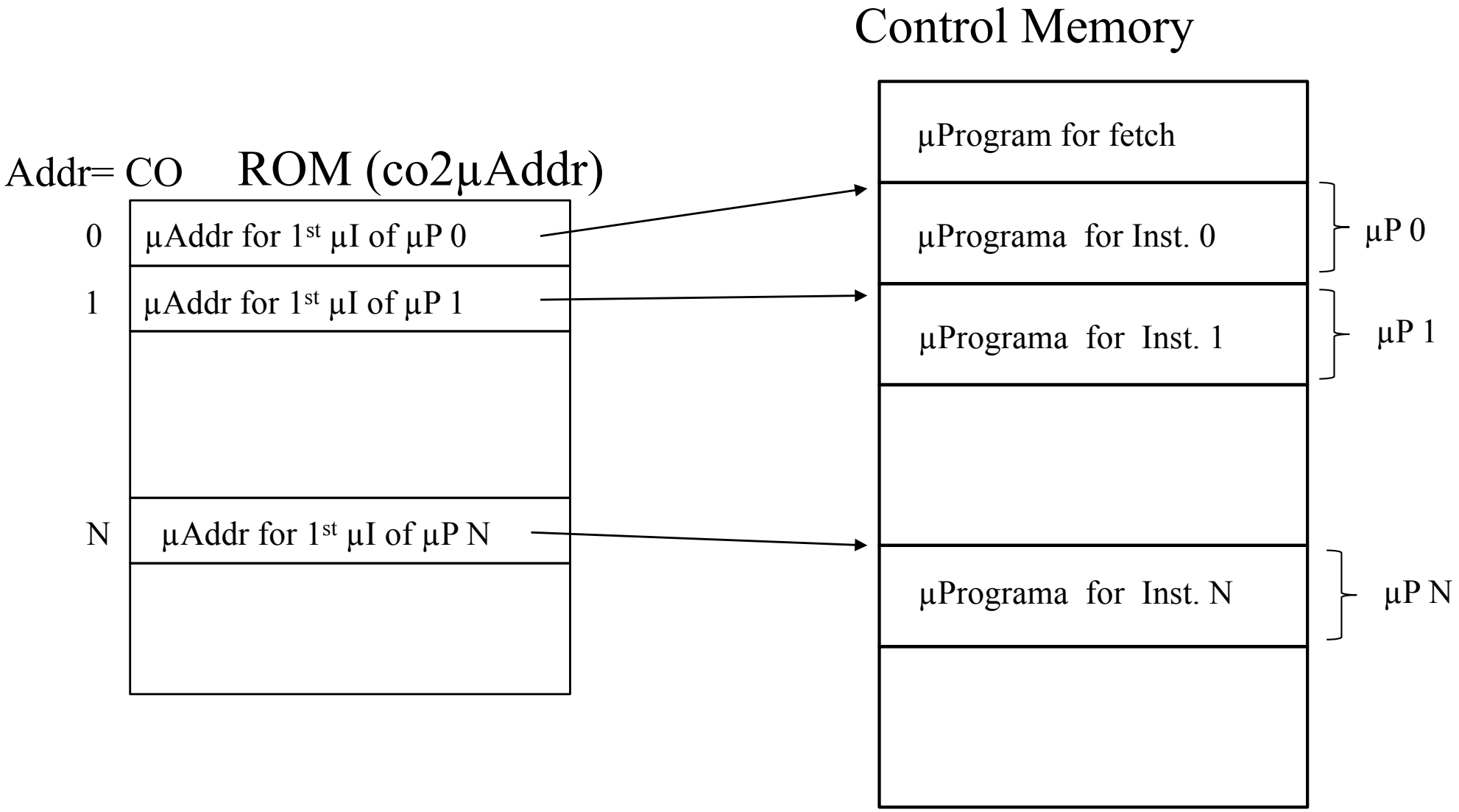
- Memoria de control guarda todos los µprogramas, donde cada µinstrucción proporciona la µdirección de la µinstrucción siguiente
- Problema: gran cantidad de memoria de control para el secuenciamiento de instrucciones, necesario almacena la µdirección siguiente

Estructura de U.C. microprogramada con secuenciamiento implícito

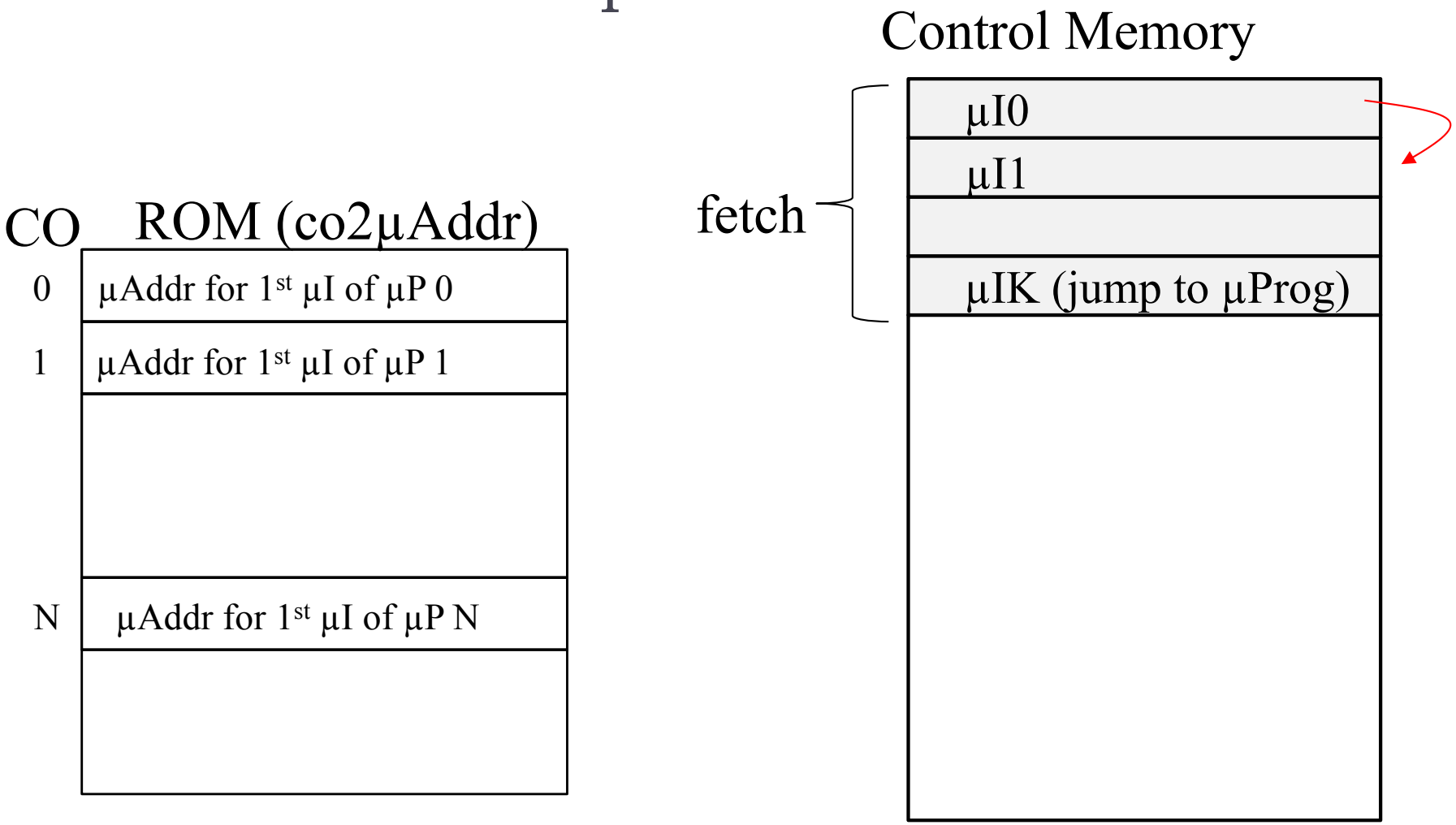


- Memoria de control guarda todos los microprogramas de forma consecutiva en la memoria de control
- La ROM/PLA asocia a cada instrucción su microprograma (primera µdirección
- Siguiente µinstrucción (+1), µbifurcaciones condicionales o µbucles

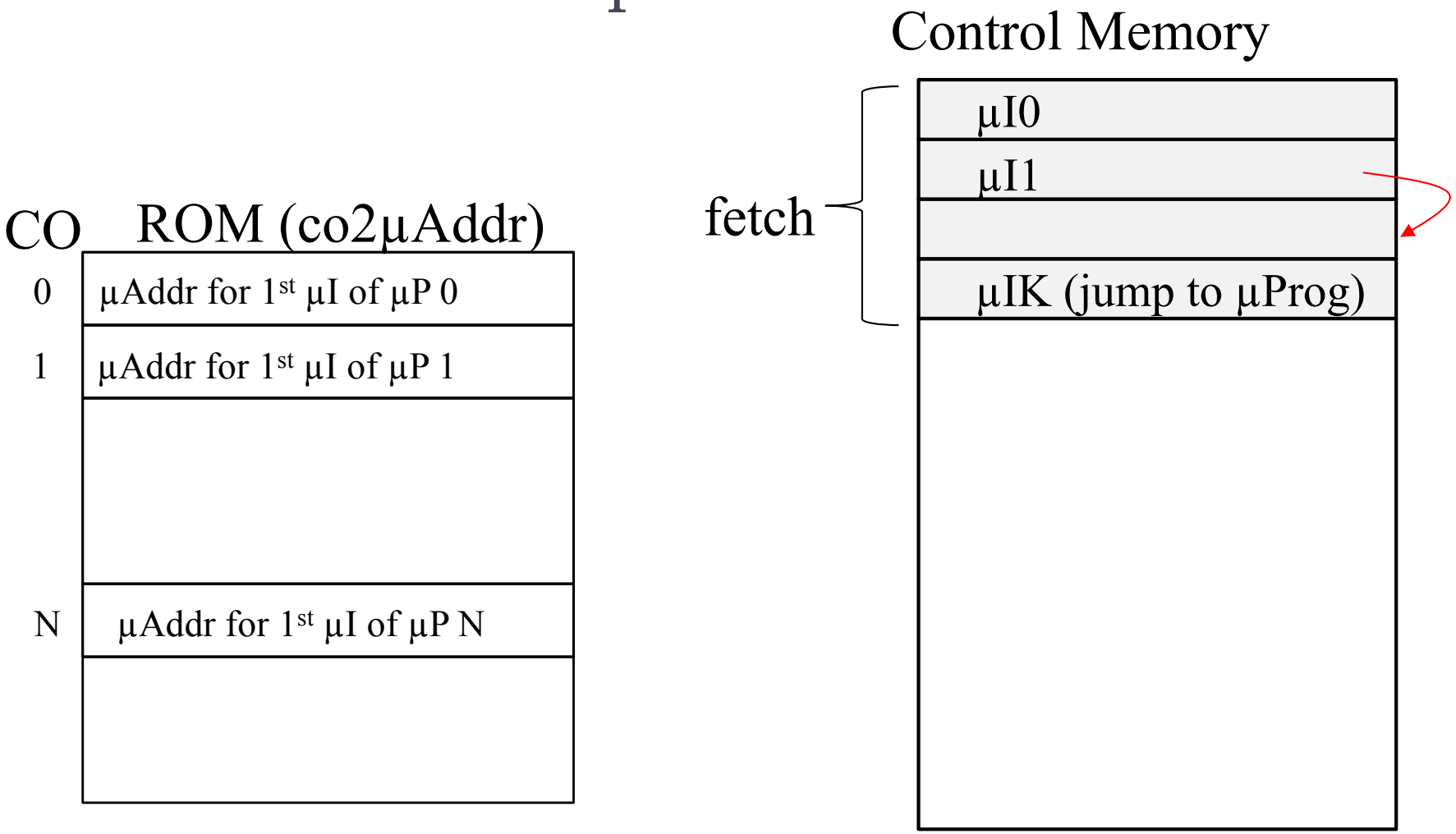
Ejemplo de funcionamiento de una UC con secuenciación impícito



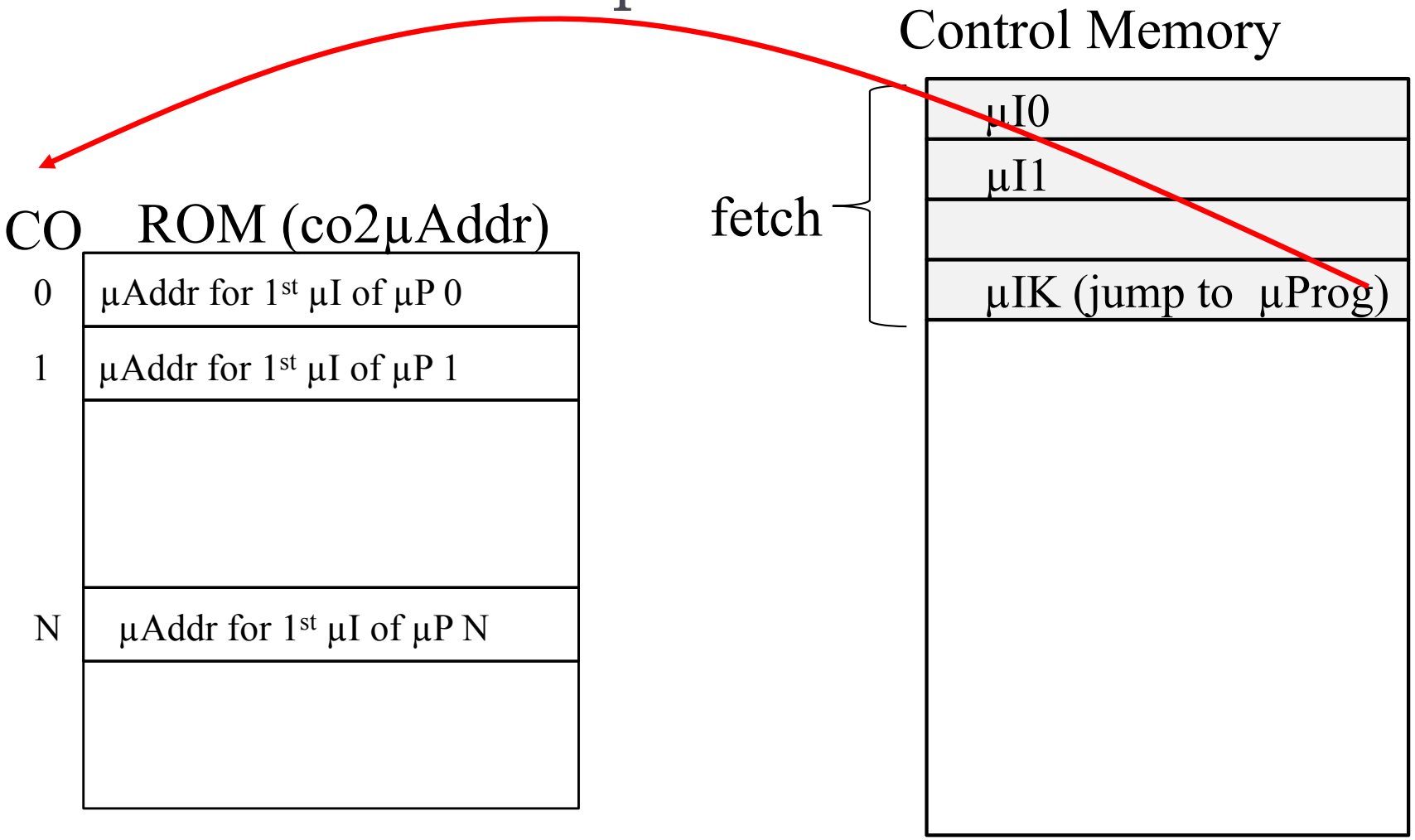
Ejemplo de funcionamiento de una UC con secuenciación impícito



Ejemplo de funcionamiento de una UC con secuenciación impícito

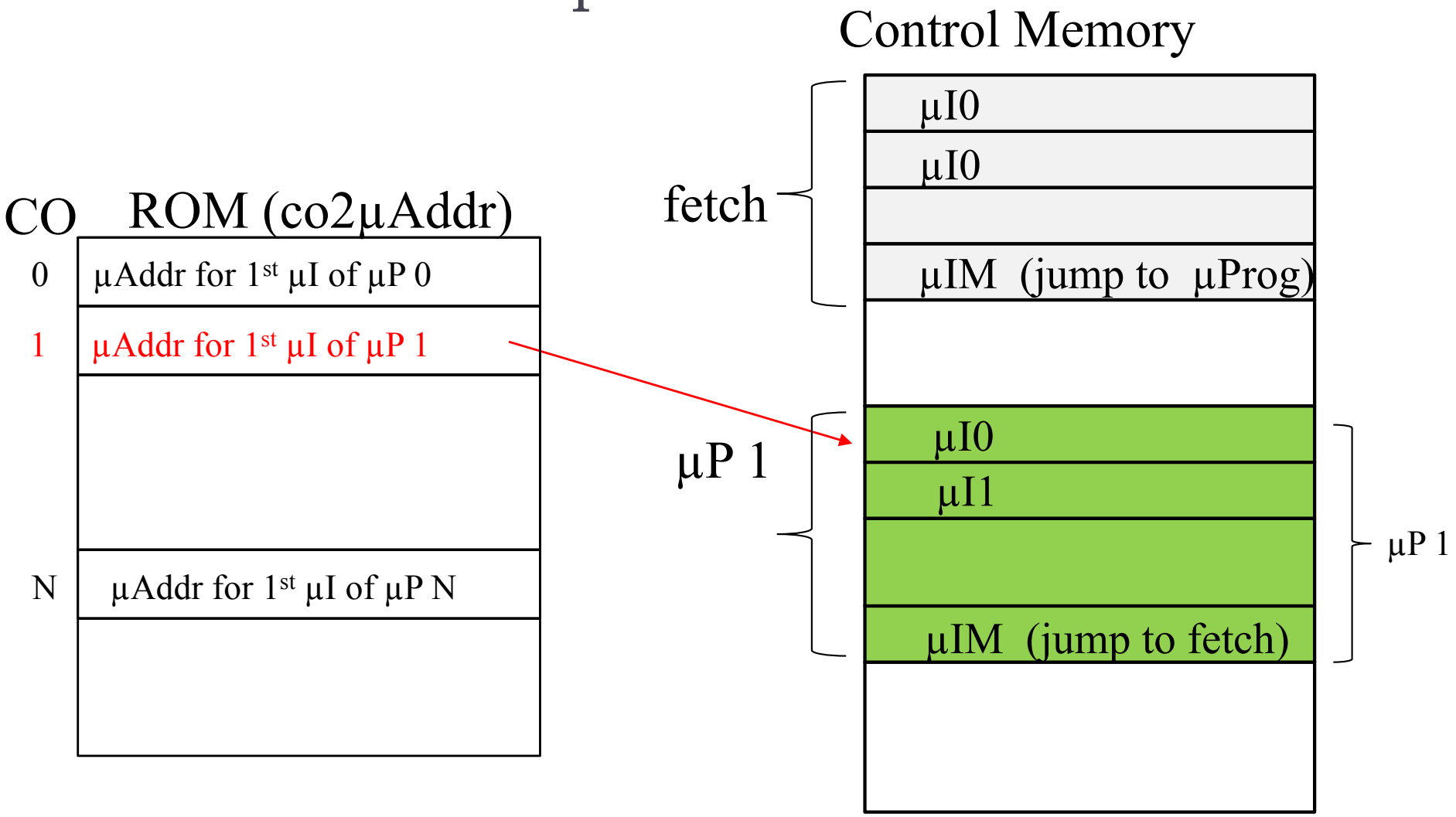


Ejemplo de funcionamiento de una UC con secuenciación impícito

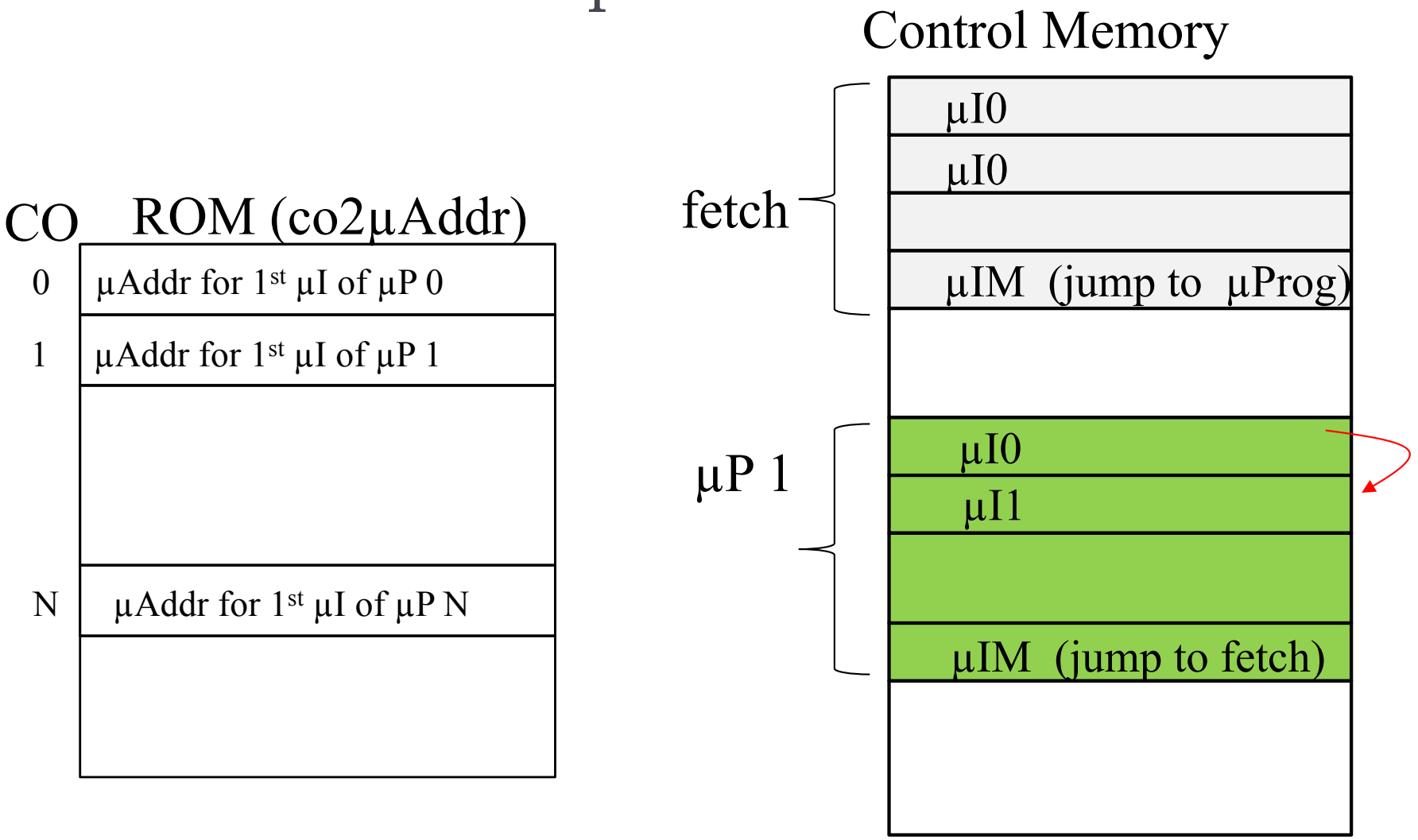


En el Registro de Instrucción el CO

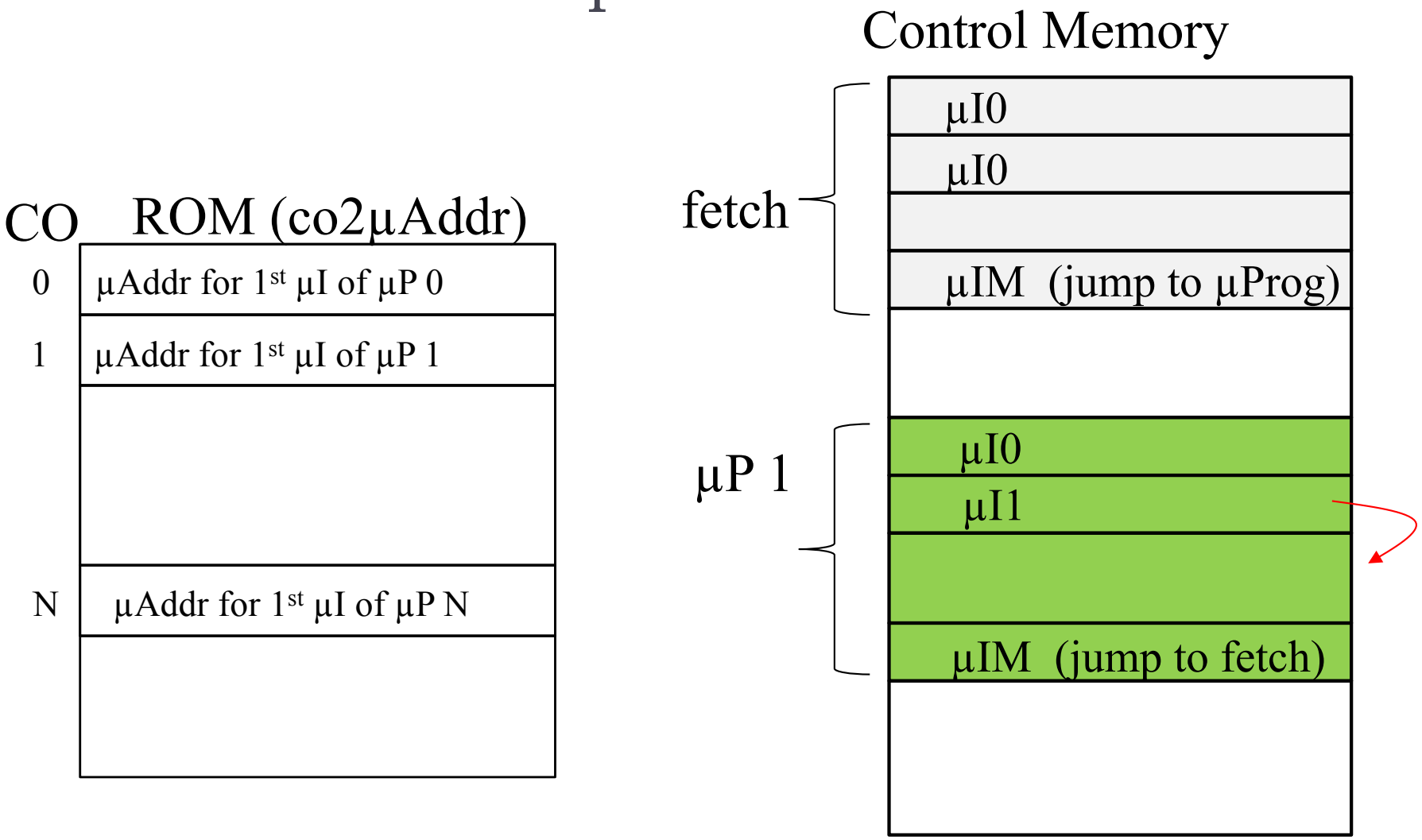
Ejemplo de funcionamiento de una UC con secuenciación impícito



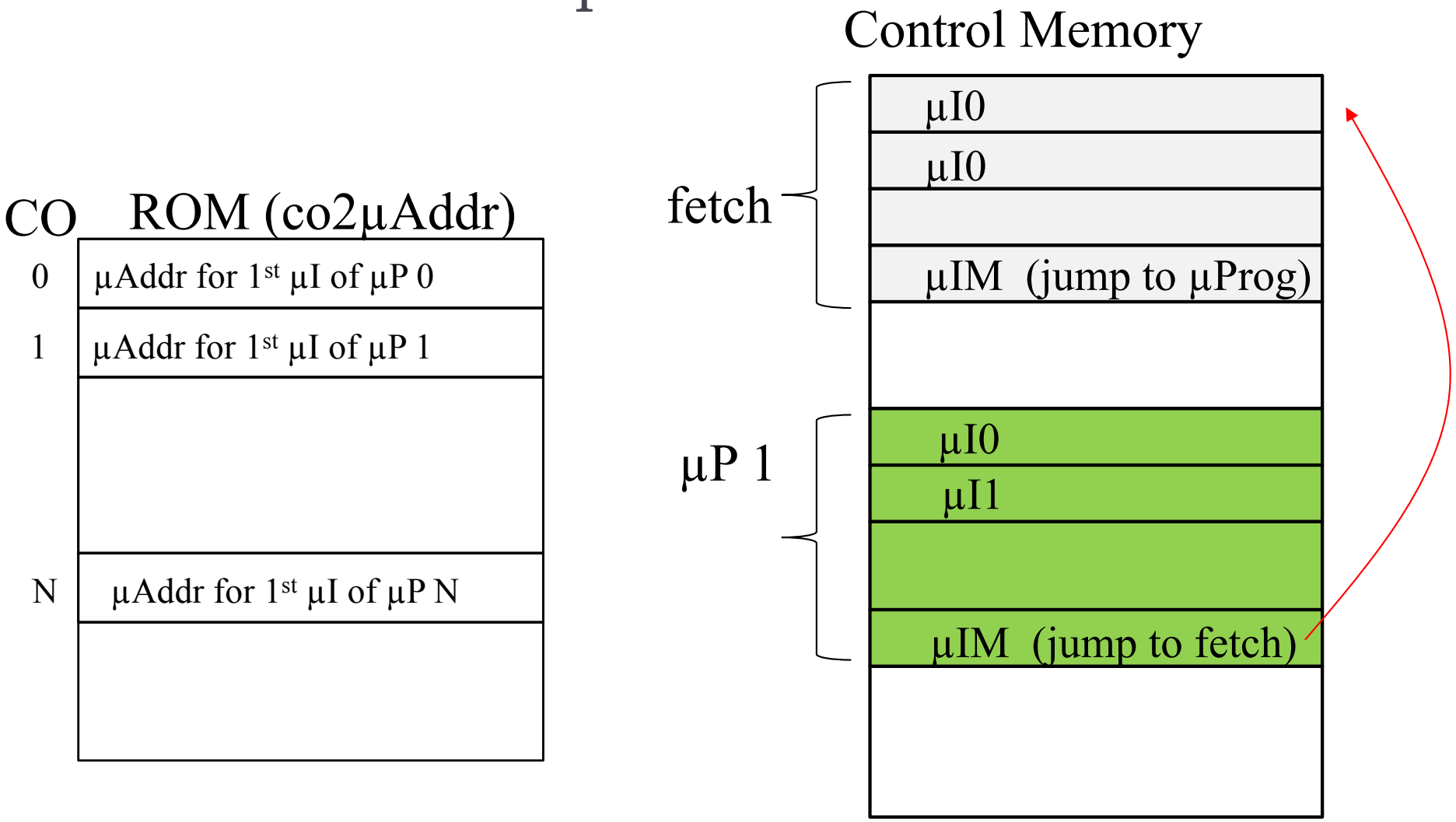
Ejemplo de funcionamiento de una UC con secuenciación impícito



Ejemplo de funcionamiento de una UC con secuenciación impícito



Ejemplo de funcionamiento de una UC con secuenciación impícito



Formato de las microinstrucciones

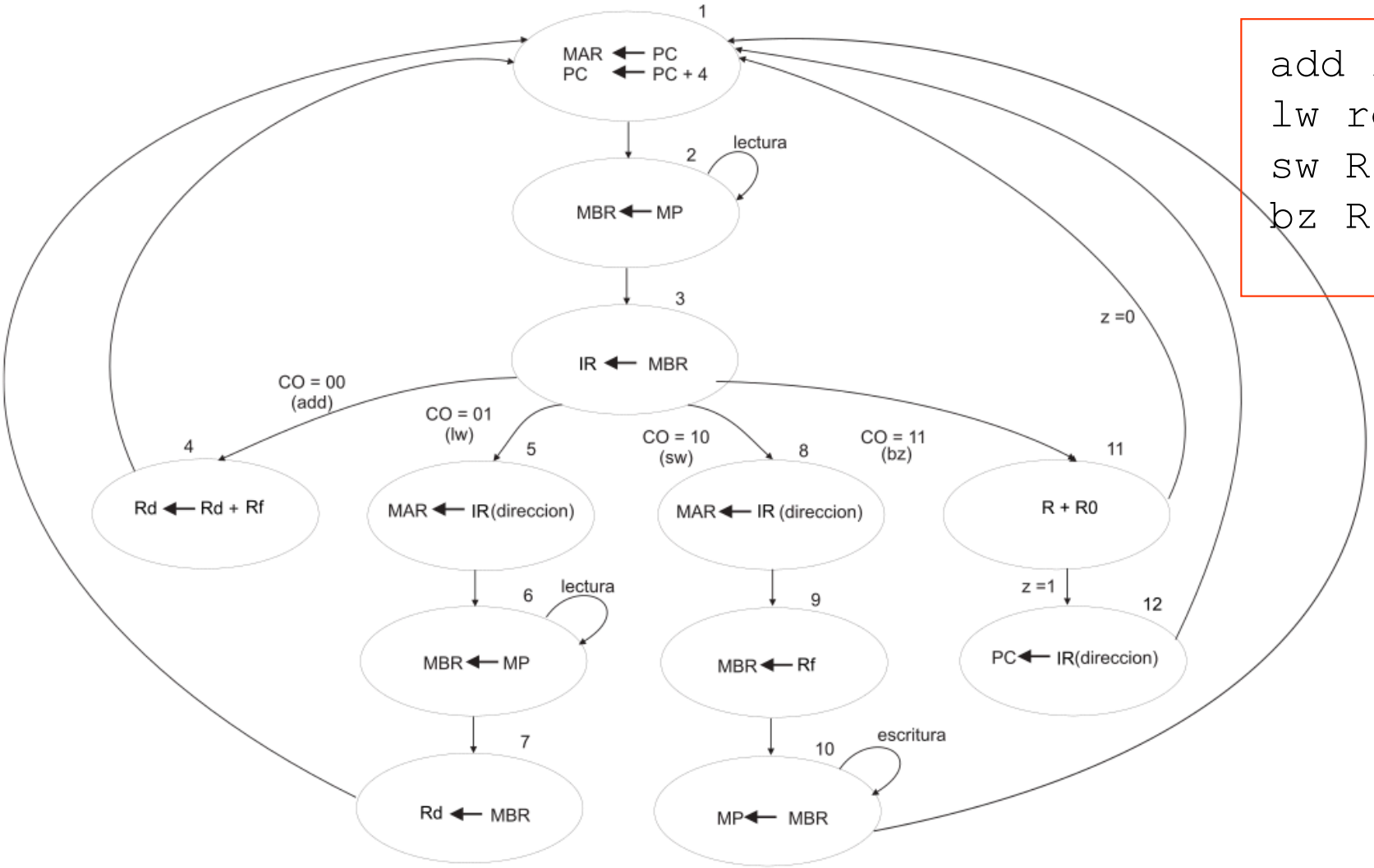
- **Formato de la microinstrucción**:
  especifica el nº de bits y el significado de cada uno de ellos.

- 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

Máquina de estados del ejemplo

Ejemplo para un computador con solo 4 instruc. máquina

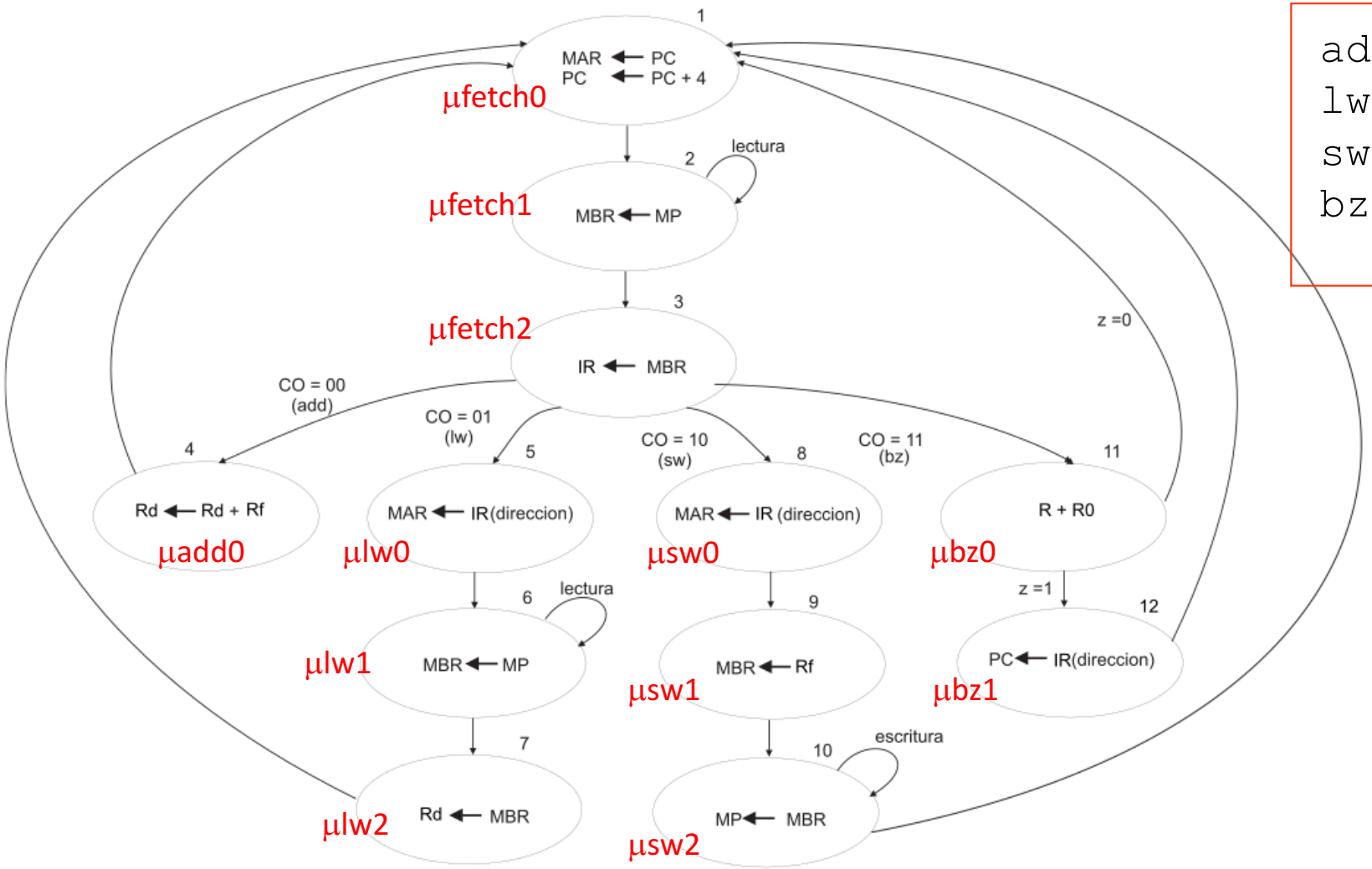
```
add rd, rf
lw rd, dir
sw Rf, dir
bz R, dir
```



Microinstrucciones para el ejemplo

Ejemplo para un computador con solo 4 instruc. máquina

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

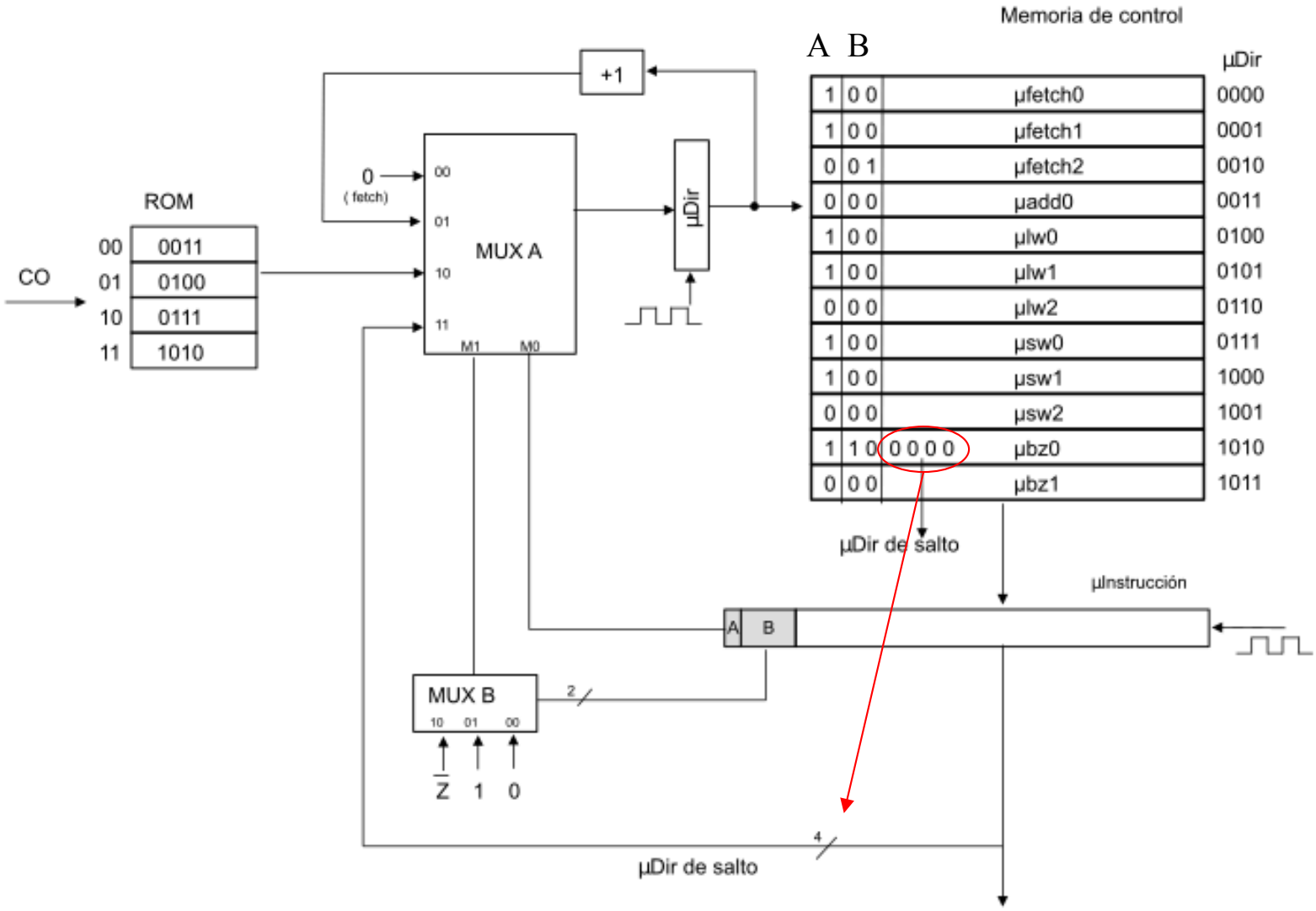


Microdódigo para el ejemplo

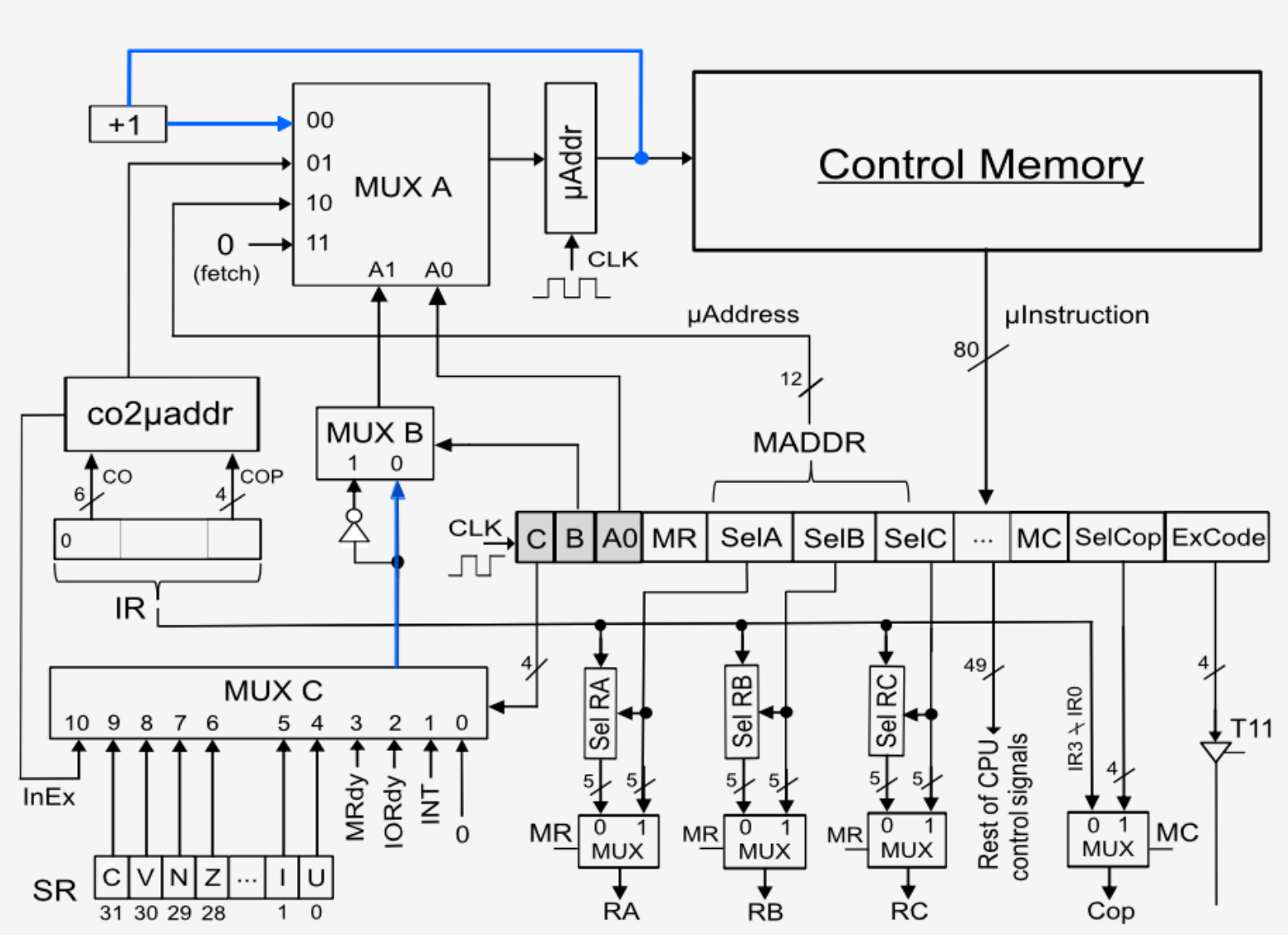
```
add r1, r2
lw r1, dir
bz dir
sw r1
```

	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	
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	fetch
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	
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	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	
µw1	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	lw
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	
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	
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	sw
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	0	1	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	

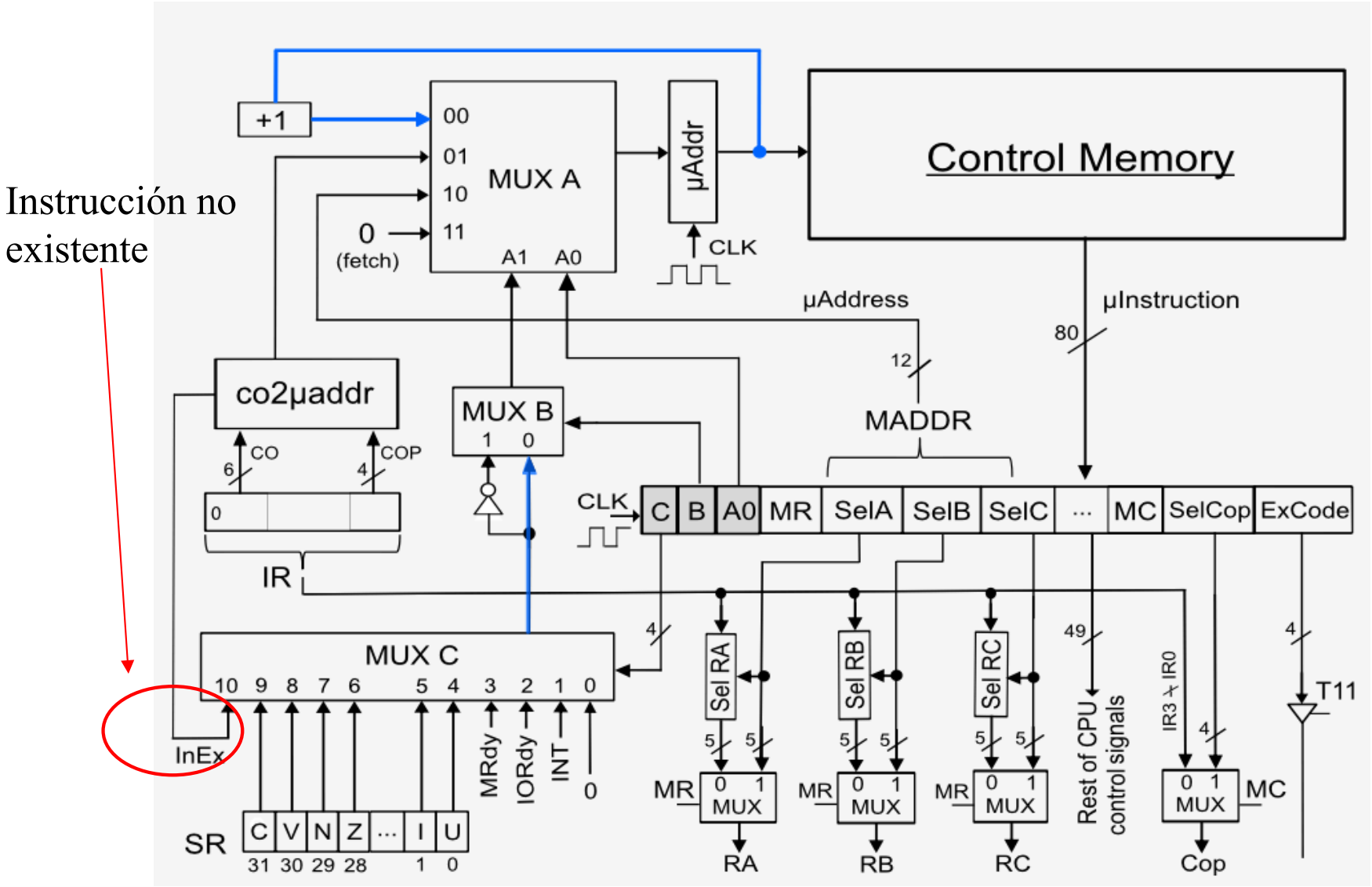
Ejemplo de unidad de control microprogramada para el ejemplo



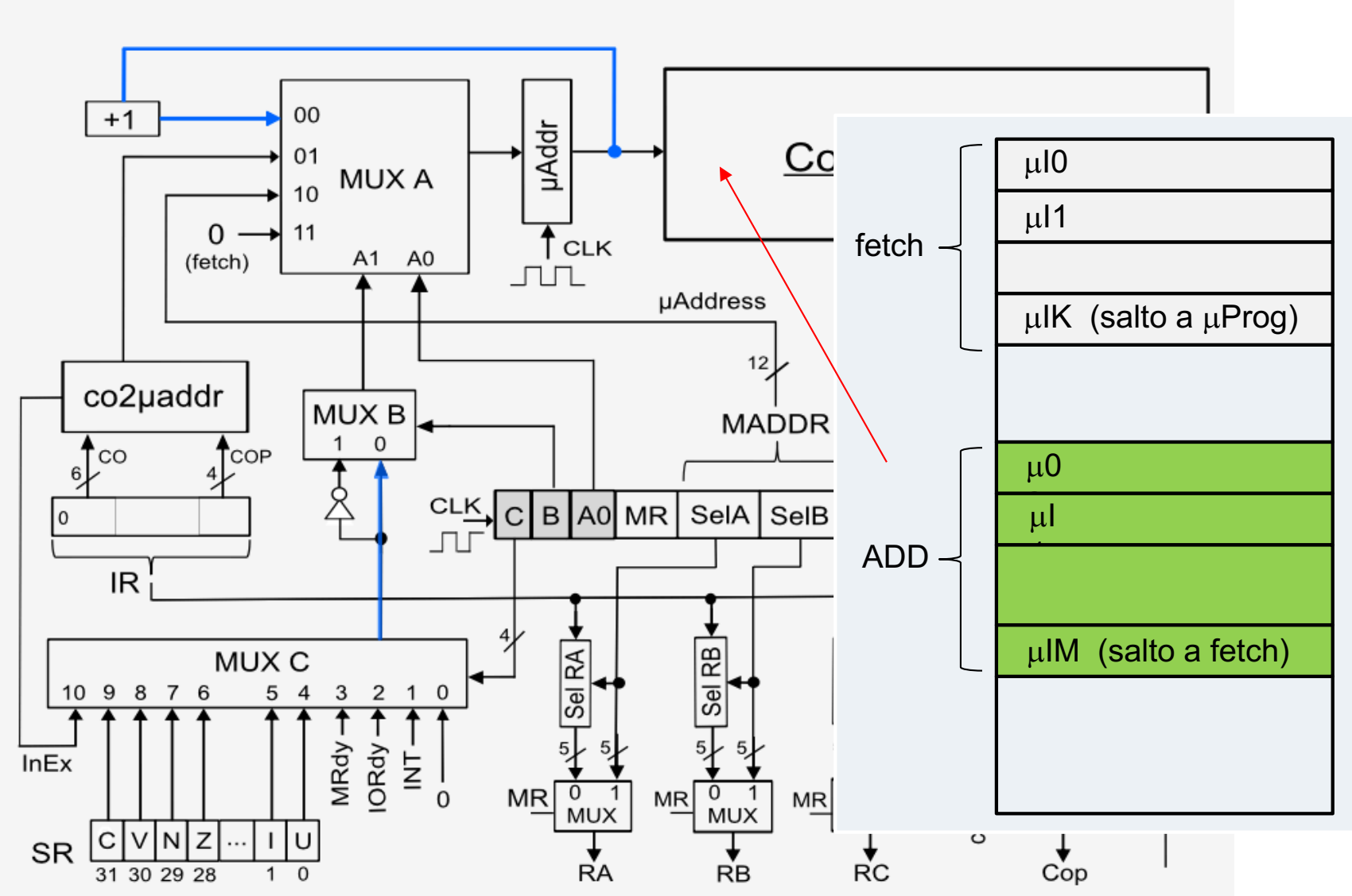
Unidad de control de WepSIM



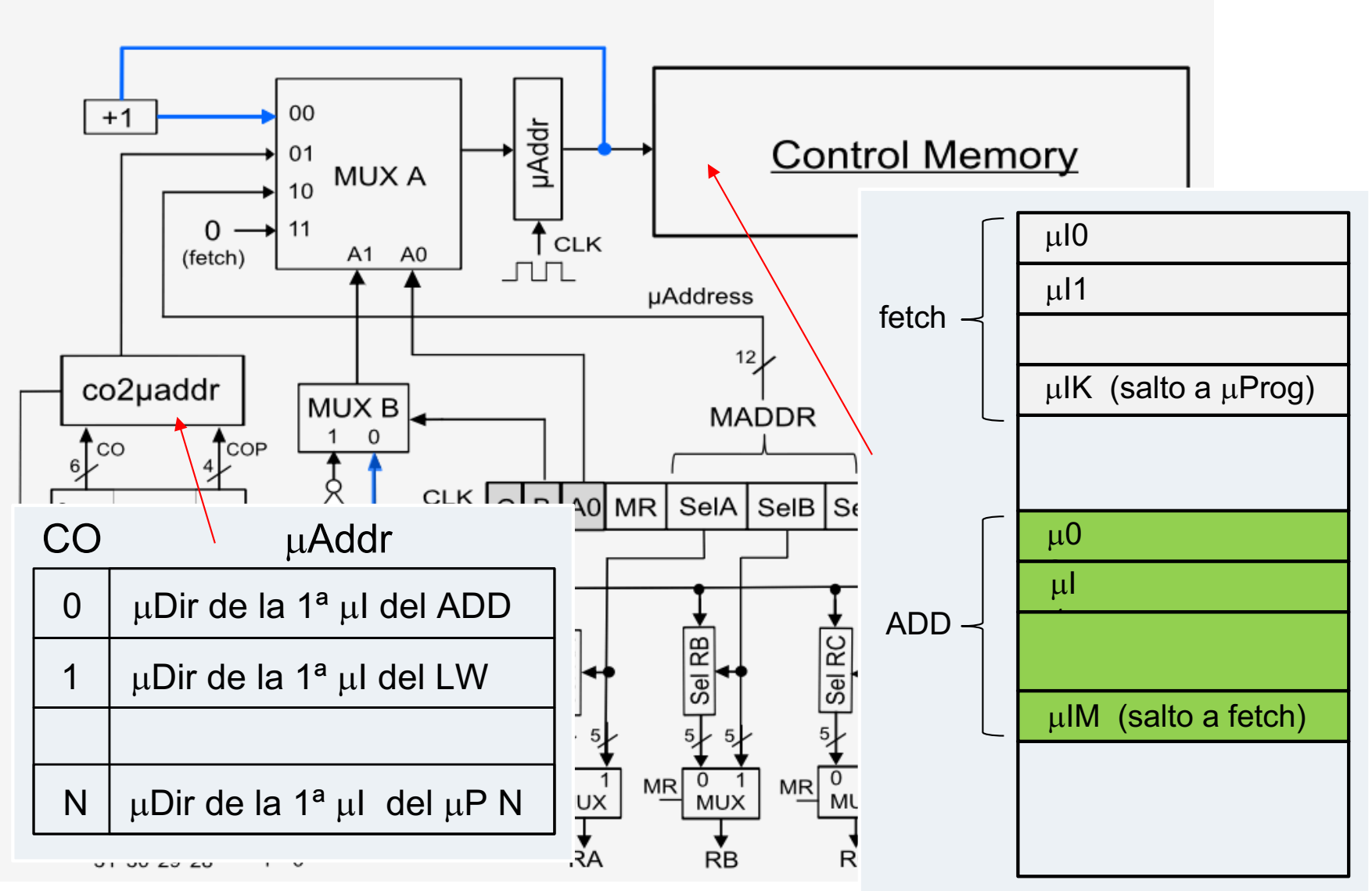
Unidad de control de WepSIM



Unidad de control de WepSIM



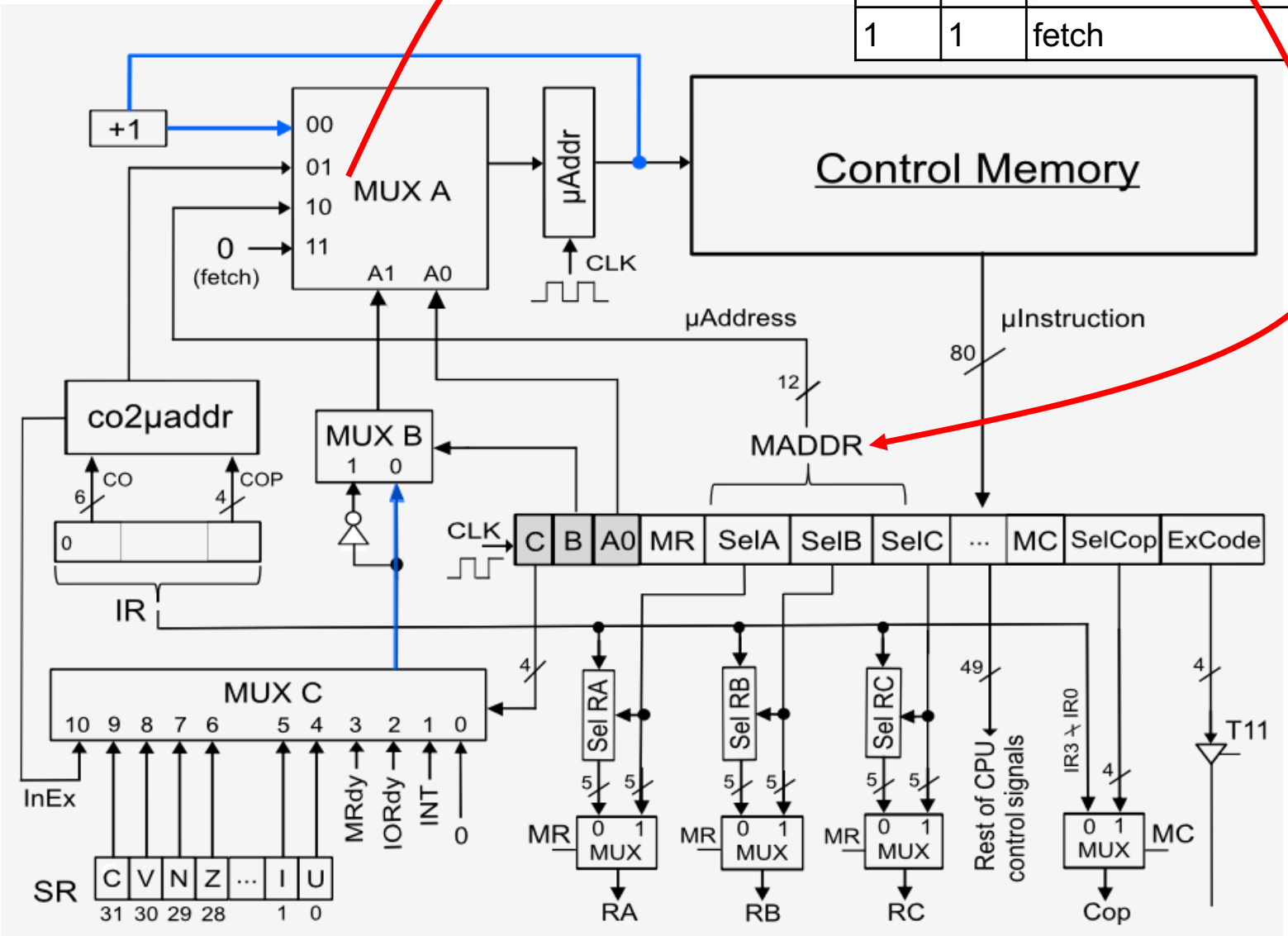
Unidad de control de WepSIM



CO	µAddr
0	µDir de la 1ª µI del ADD
1	µDir de la 1ª µI del LW
N	µDir de la 1ª µI del µP N

Unidad de control de WepSIM

A1	A0	Salida
0	0	Sig μDir
0	1	Salto a μProg. (ROM)
1	0	μDir de salto
1	1	fetch



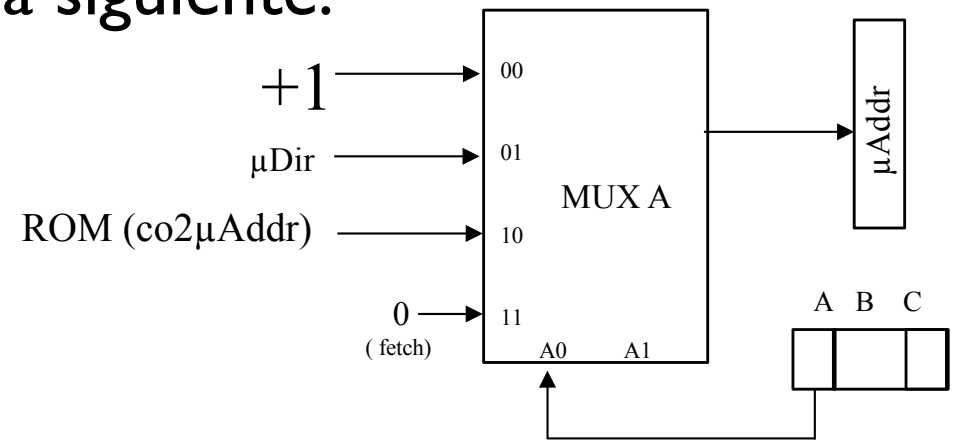
Unidad de control de WepSIM

A0	B	C3	C2	C1	C0	Acción
0	0	0	0	0	0	Siguiente μDirección
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 μProg. (ROM c02μaddr)
1	1	0	0	0	0	Salto a fetch (μDir = 0)

- (*) Si no se cumple la condición → Siguiente μDirección
- Resto de entradas → funcionamiento indefinido

Ejemplo

- Salto a la µDirección 000100011100 (12 bits) si Z = 1.
  En caso contrario se salta a la siguiente:
  - A0 = 0
  - B= 0
  - C = 0110
  - µAddr = 000100011100

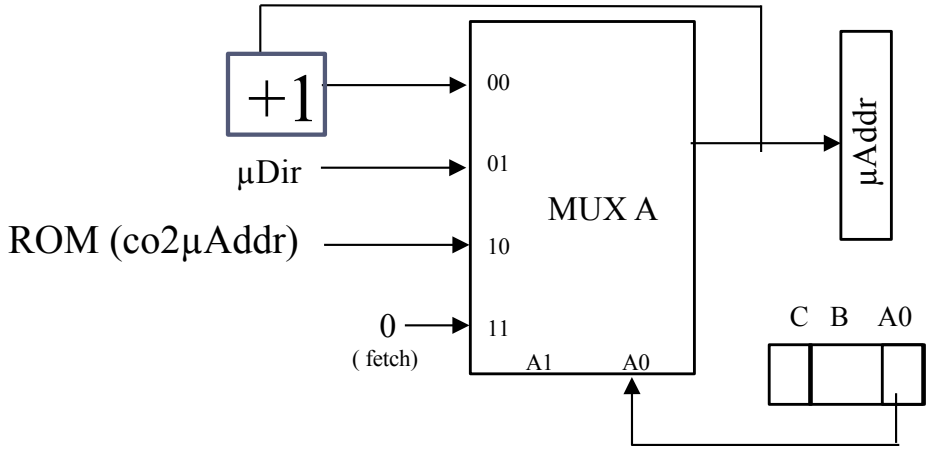


- Salto incondicional a la µDirección 000100011111
  - A0 = 0
  - B= 1
  - C = 0000
  - µAddr = 000100011111

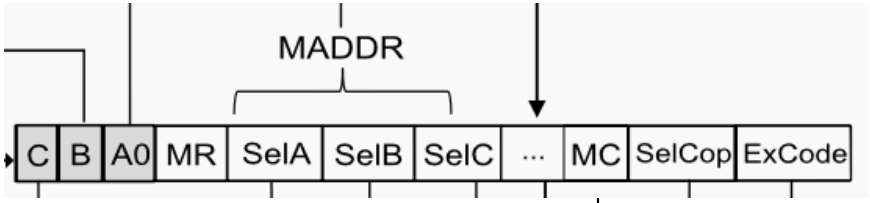
µDirección codificada en los bits 72-61 de la µInstrucción

Ejemplo

- En la última μInstrucción del fetch saltar a la primera μDirección del μPrograma asociado al código de operación leído de memoria
  - A0 = 1
  - B= 0
  - C = 0000



Formato de la microinstrucción



MicroAddress with 80bits

C0 .. C7	Carga en registros
Ta, Td	Triestados a buses
T1..T10	Puertas triestado
M1, M2, M7, MA, MB	Multiplexores
SelP	Selector Registro estado
LE	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

Ejemplo
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) μPC=000100011100	A0=0, B=0, C=0110_2, mADDR=000100011100_2

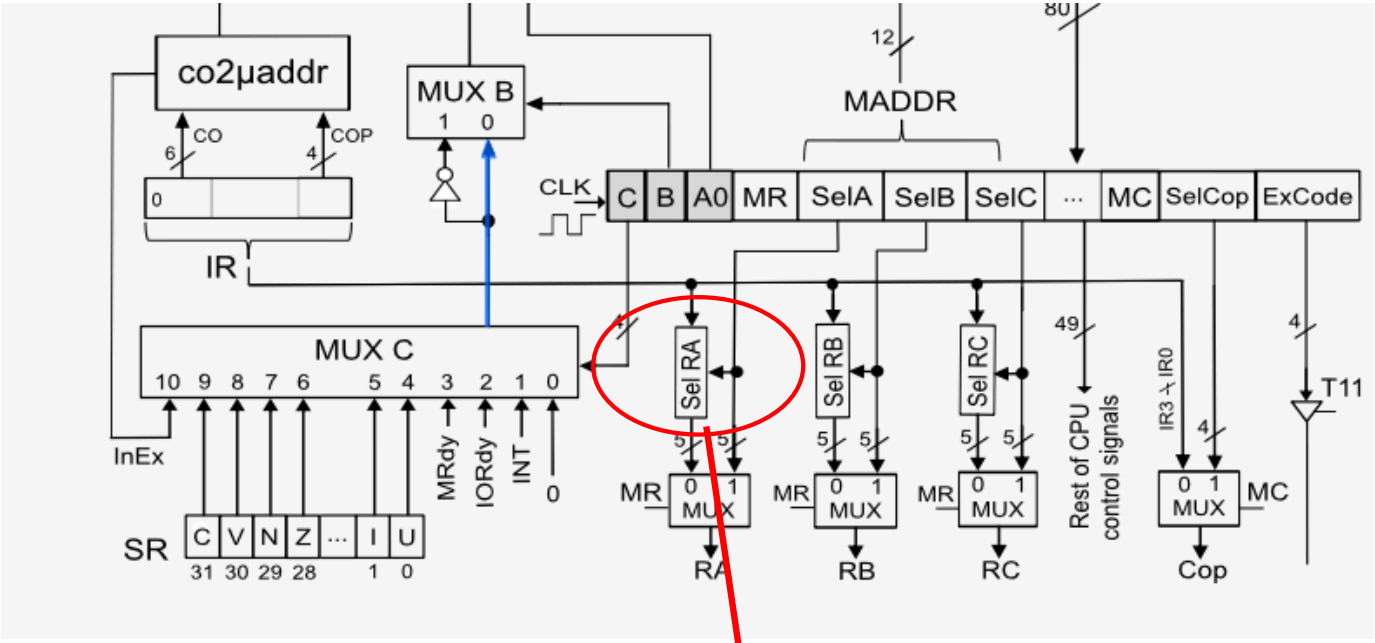
- **Salto incondicional a la dirección 000100011111**

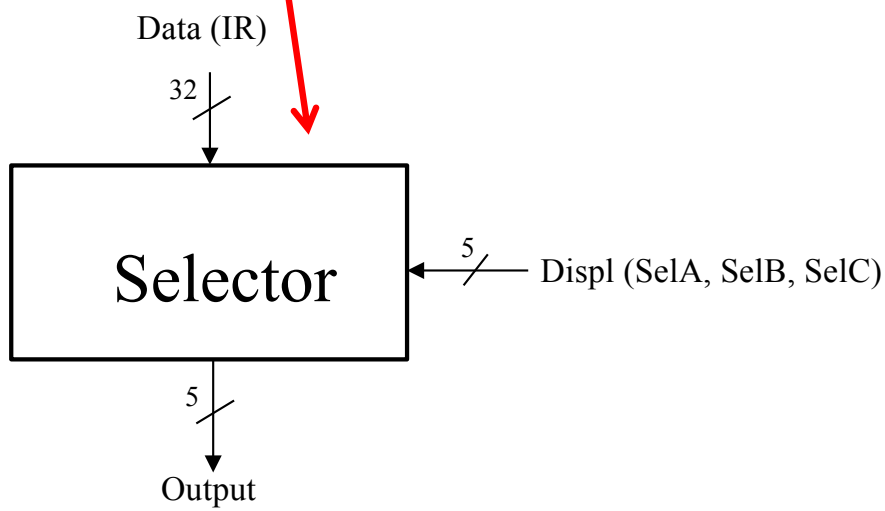
O. Elemental	Señales
μPC=000100011111	A0=0, B=1, C=0000_2, mADDR=000100011111_2

- **Salto a la primera μdirección del μprograma asociado al CO**

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

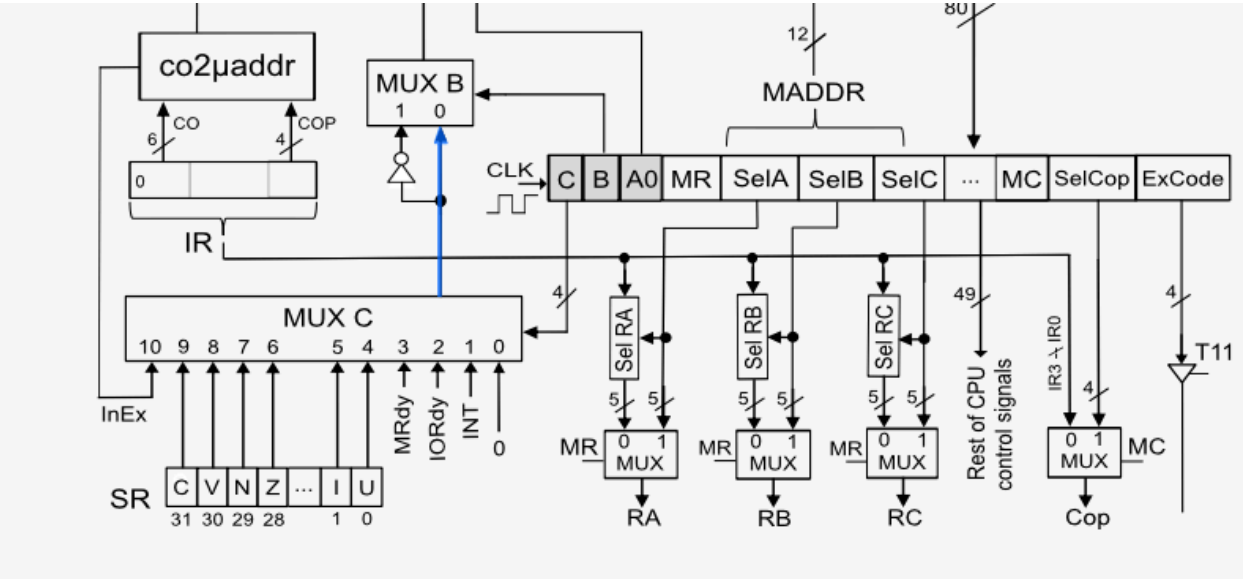
Selector de registros

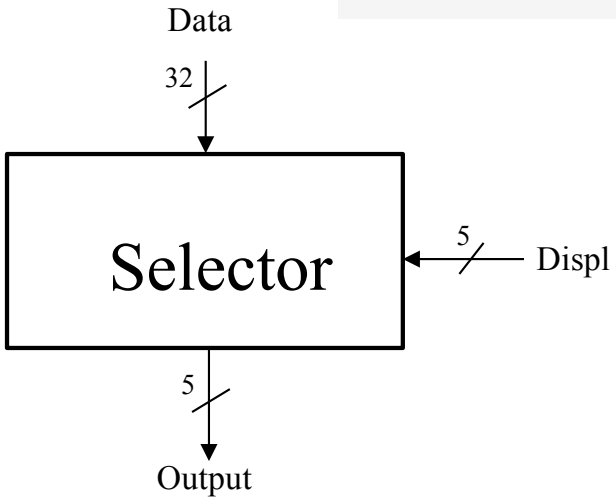




Selecciona 5 bits de un conjunto
de 32 bits desde la posición indicada
en Displ (bit inferior)

Selector de registros
Ejemplo





RI: $D_{31}D_{30}D_{29}D_{28}D_{27}D_{26}D_{25}$ ……. $D_4D_3D_2D_1D_0$

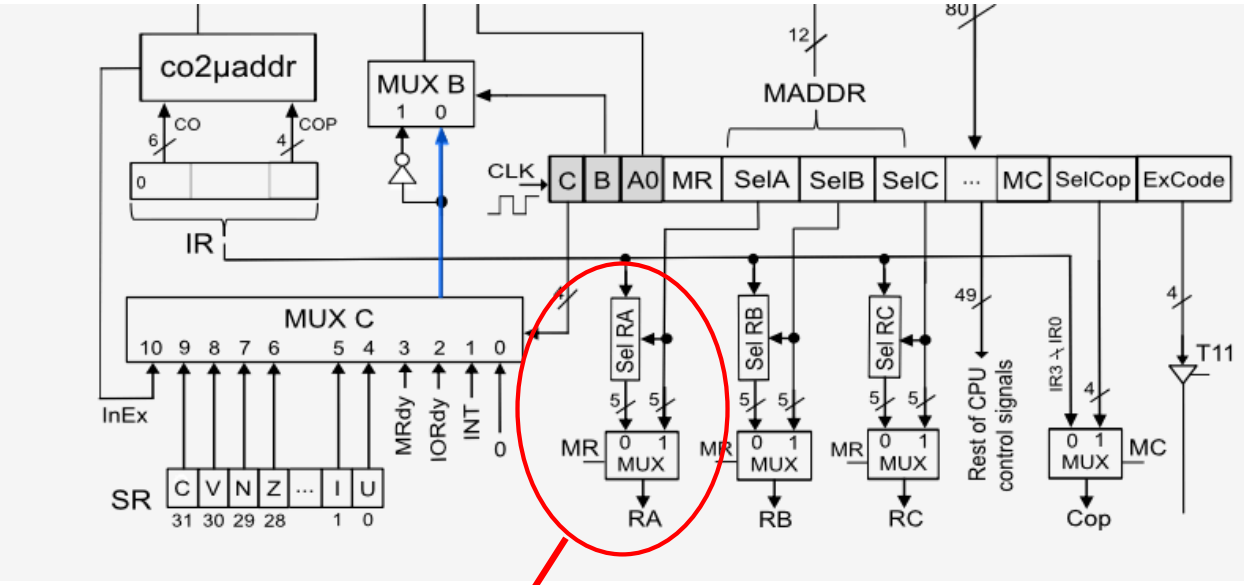
Si Displ = 11011 → Output = $D_{31}D_{30}D_{29}D_{28}D_{27}$

Si Displ = 00000 → Output = $D_4D_3D_2D_1D_0$

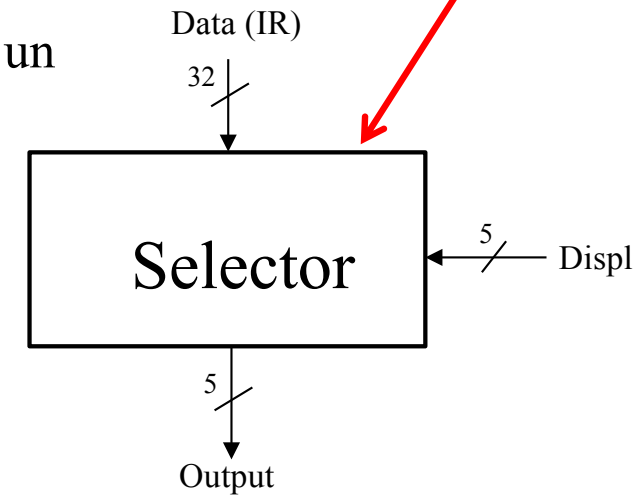
Si Displ = 10011 → Output = $D_{23}D_{22}D_{21}D_{20}D_{19}$

Si Displ = 01011 → Output = $D_{15}D_{14}D_{13}D_{12}D_{11}$

Selector de registros



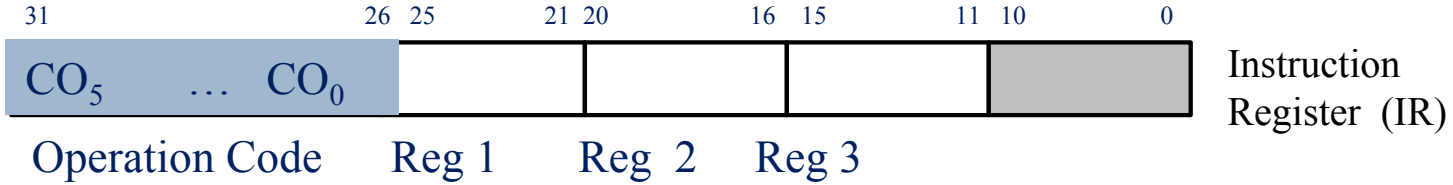
Selecciona 5 bits de un conjunto
de 32 bits desde la posición indicada en Displ

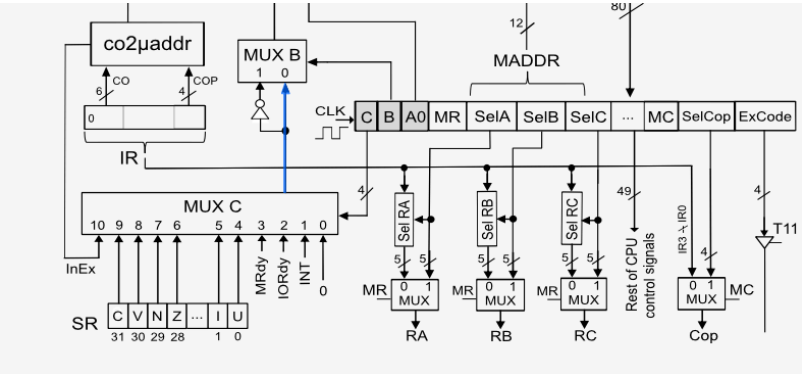


- Si MR = 1, RA se obtiene directamente de la µInstrucción
- Si MR = 0, RA se obtiene de un campo de la instrucción (en IR)

Selector de registros

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

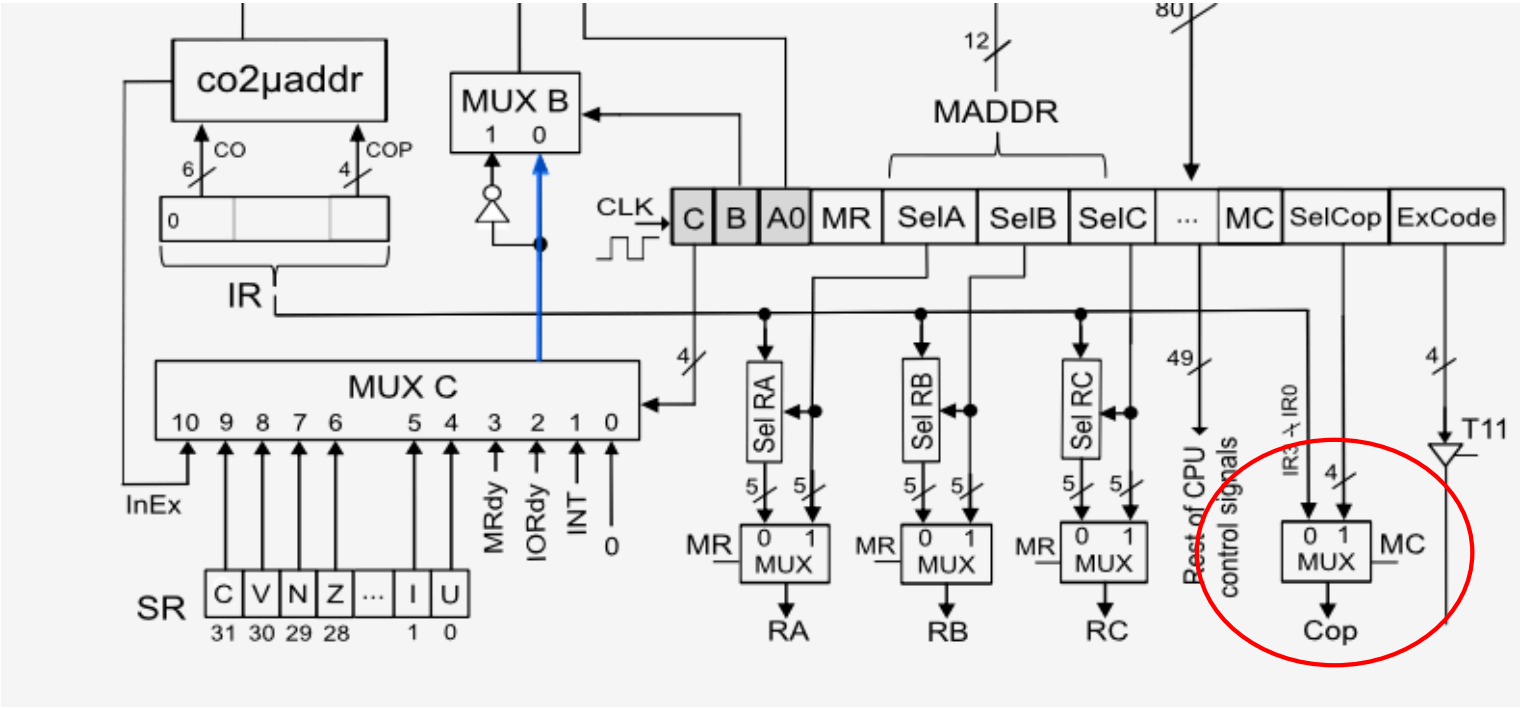




MR = 0

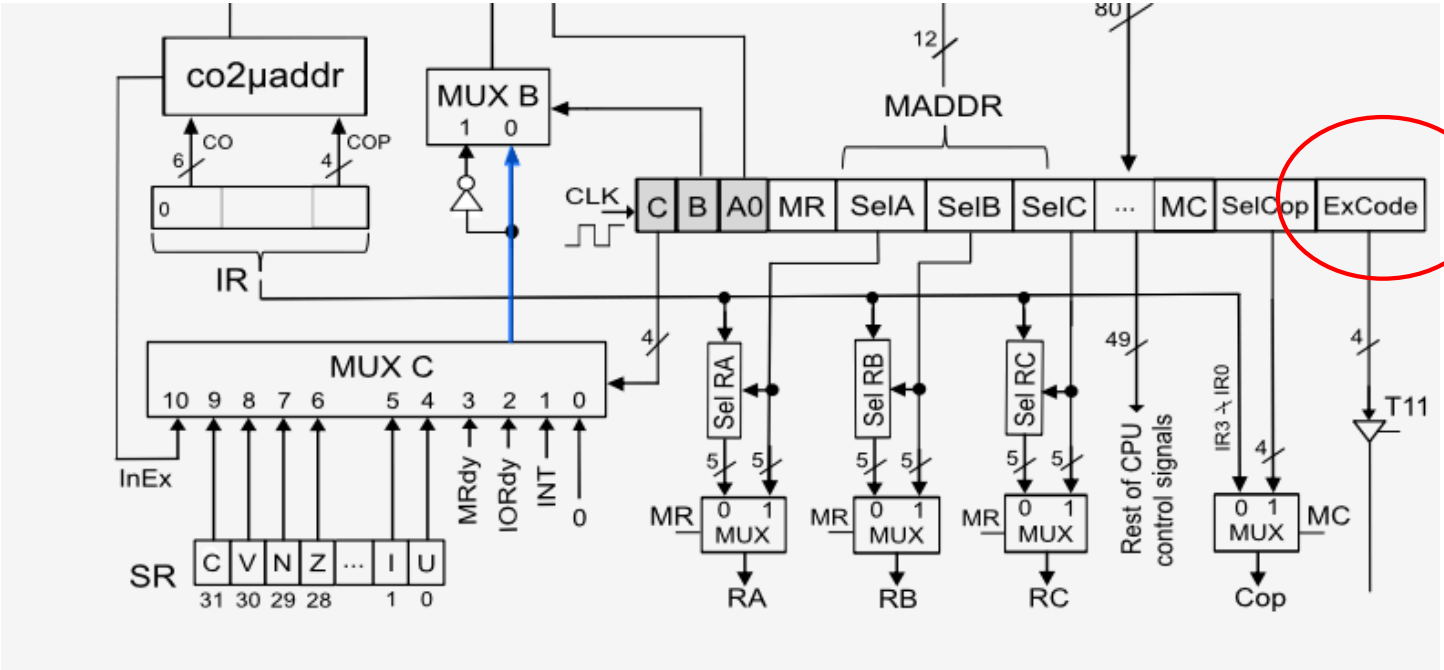
- Si se quiere seleccionar el campo con el Reg 2 en la puerta B del banco de registros → SelB = 10000 (RB se obtiene de los bits 20…16 del IR)
- Si se quiere seleccionar el campo con el Reg 3 en la puerta A del banco de registros → SelA = 01011 (RA se obtiene de los bits 15…11 del IR)
- Si se quiere 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: vector de interrupción a utilizar cuando se produce una excepción en la ejecución de la instrucción.

Ejemplo

- Instrucciones a microprogramar con WepSIM*:

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

* Memoria de un ciclo

Microprograma de las instrucciones

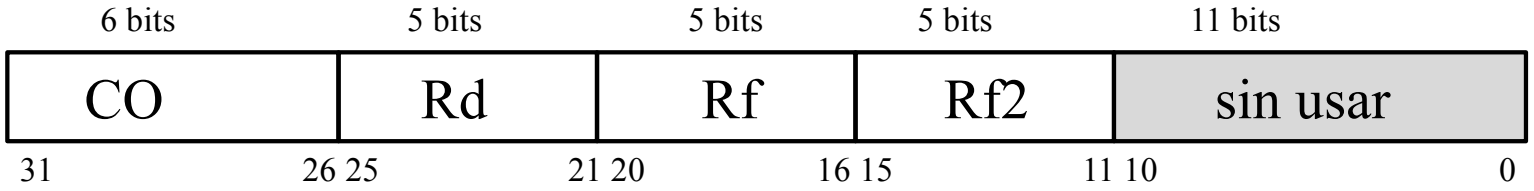
- FETCH

Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	MAR ← PC	T2, C0	0000	0	0
1	MBR ← MP	Ta, R, BW = 11, C1, M1	0000	0	0
	PC ← PC + 4	M2, C2	0000	0	0
2	IR ← MBR	T1, C3	0000	0	0
3	Decodificación		0000	0	1

Microprograma de las instrucciones

- ADD Rd, Rf1, Rf2

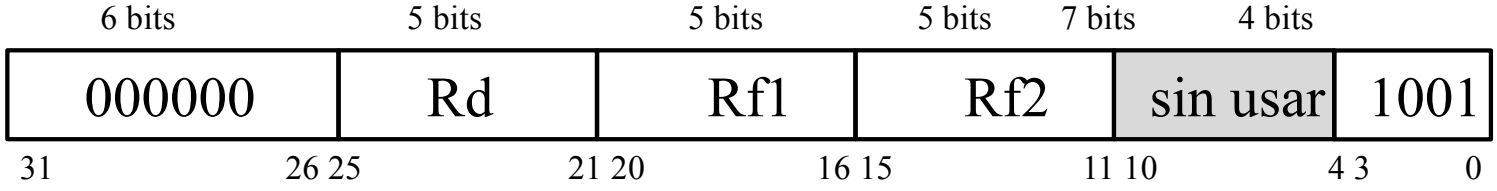
Ciclo	Op. Elemental	Señales activadas (resto a 0)	C B A0
0	Rd ← Rf1 + Rf2	Cop = 1010 SelP=11, C7, M7 T6, LC SelA = 10000 (16) SelB = 01011 (11) SelC = 10101 (21)	0000 1 1



Microprograma de las instrucciones (otra)

- ADD Rd, Rf1, Rf2

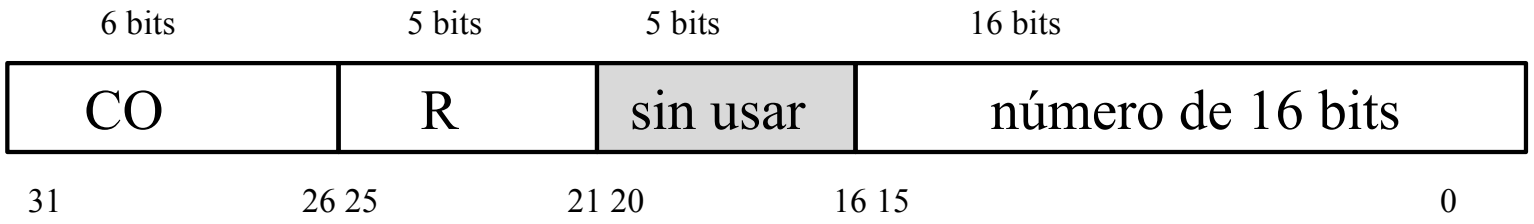
Ciclo	Op. Elemental	Señales activadas (resto a 0)	C B A0
0	Rd ← Rf1 + Rf2	SelCop = 1010, MC SelP=11, C7, M7 T6, LC SelA = 10000 (16) SelB = 01011 (11) SelC = 10101 (21)	0000 1 1



Microprograma de las instrucciones

- LI R, valor

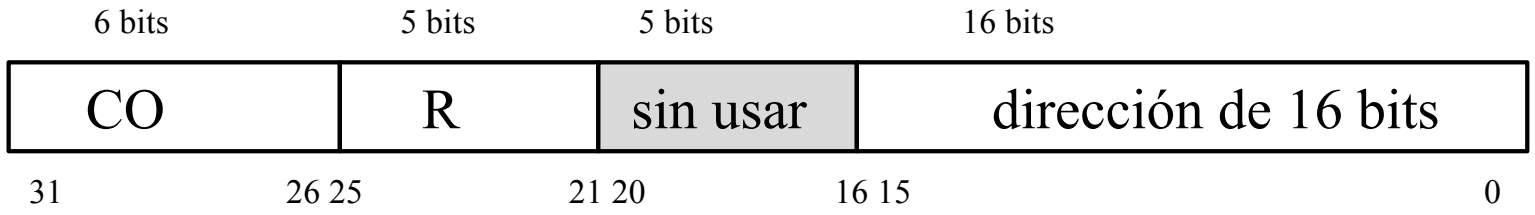
Ciclo	Op. Elemental	Señales activadas (resto a 0)	C B A0
0	R ← IR (valor)	LC SelC = 10101 (21) T3, Size = 10000 Offset= 00000 SE=1	0000 1 1



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 ← IR (dir)	T3, C0 Size = 10000, Offset= 00000	0000	0	0
1	MBR ← MP[MAR]	Ta, R, BW =11, C1, M1	0000	0	0
2	R ← MBR	T1, LC, SelC = 10101	0000	1	1



Microprograma de las instrucciones

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

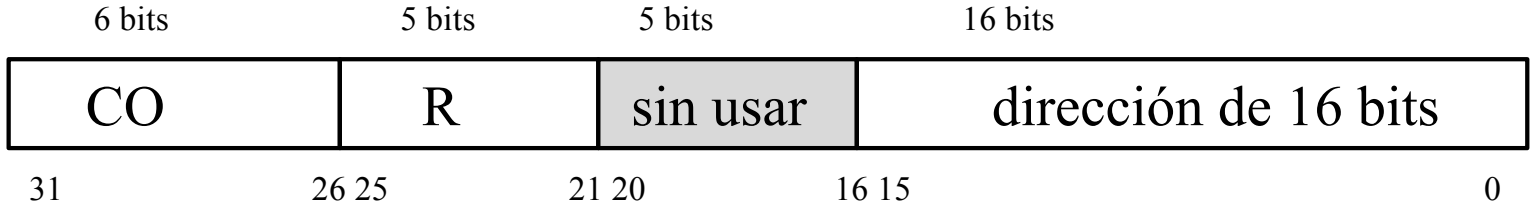
Ciclo	Op. Elemental	Señales activadas (resto a 0)	C B A0
0	MAR ← IR (dir)	T3, C0 Size = 10000, Offset= 00000	0000 0 0
1	while (!MRdy) MBR ← MP[MAR]	Ta, R, BW =11, C1, M1, MADDR=μAdd de esta μinstrucción	0011 1 0
2	R ← MBR	T1, LC, SelC = 10101	0000 1 1

Se ejecuta esta microinstrucción mientras MRdy==0

Microprograma de las instrucciones

- SW R dir, con memoria síncrona de un ciclo

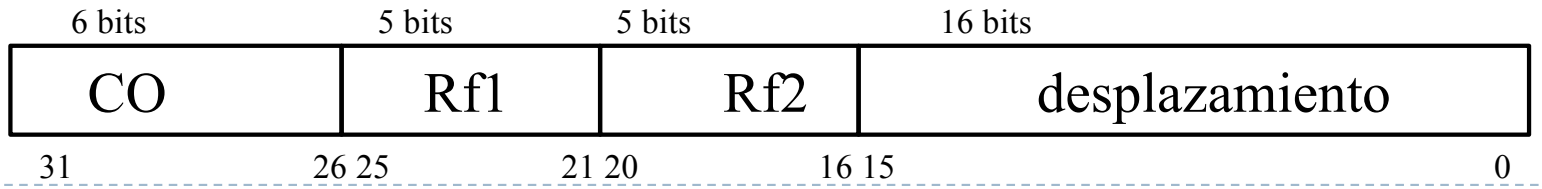
Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	MBR ← R	T9, C1, SelA=10101	0000	0	0
1	MAR ← IR(dir)	T3, C0, Size = 10000, offset= 00000	0000	0	0
2	MP[dir] ← MBR	Td, Ta, BW = 11, W	0000	1	1



Microprograma de las instrucciones

- BEQ Rf1, Rf2, desp

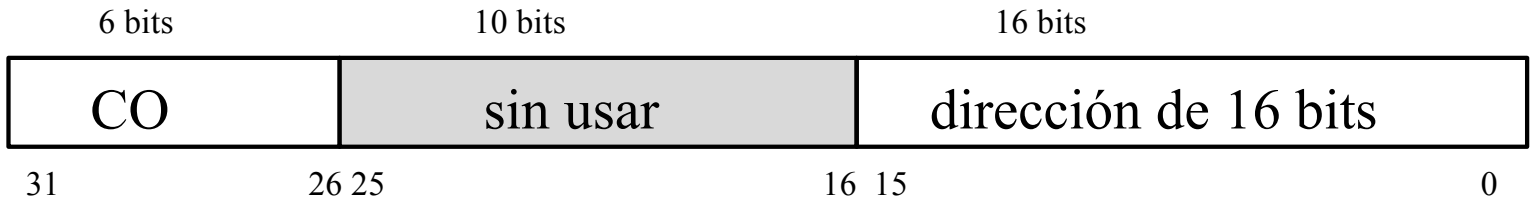
Ciclo	Op. Elemental	Señales activadas (resto a 0)	C B A0
0	Rf1 - Rf2	SelCop = 1011, MC, C7, M7 SelP = 11, SelA = 10101 SelB = 10000	0000 0 0
11	If (Z == 0) goto fetch else next	$MADDR = 0$	0110 1 0
2	RT1 ←PC	T2, C4	0000 0 0
3	RT2 ← IR (dir)	Size = 10000 Offset = 00000, T3,C5	0000 0 0
4	PC ← RT1 +RT2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



Especificación de los microprogramas en WepSIM

Lista de microcódigos

especificación de registros

pseudoinstruciones

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)
        }
}
```

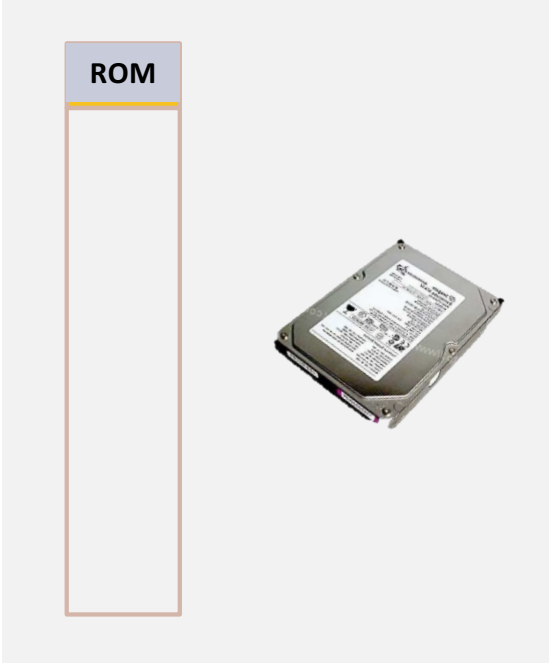
etiqueta, representa µdirección de salto

Especificación de registros

```
registers{
        0=$zero,
        1=$at,
        2=$v0,
        3=$v1,
        4=$a0,
        5=$a1,
        6=$a2,
        7=$a3,
        8=$t0,
        9=$t1,
        10=$t2,
        11=$t3,
        12=$t4,
        13=$t5,
        14=$t6
        15=$t7,
        16=$s0,
        17=$s1,
        18=$s2,
        19=$s3,
        20=$s4,
        21=$s5,
        22=$s6,
        23=$s7,
        24=$t8,
        25=$t9,
        26=$k0,
        27=$k1,
        28=$gp,
        29=$sp (stack_pointer),
        30=$fp,
        31=$ra
}
```

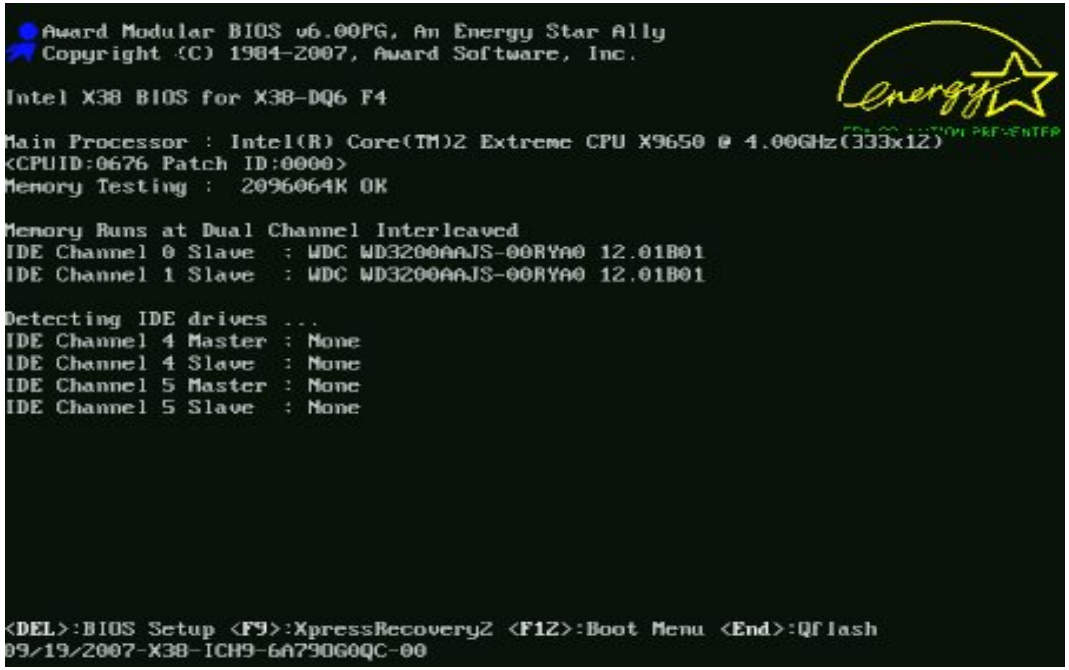
Arranque del computador

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

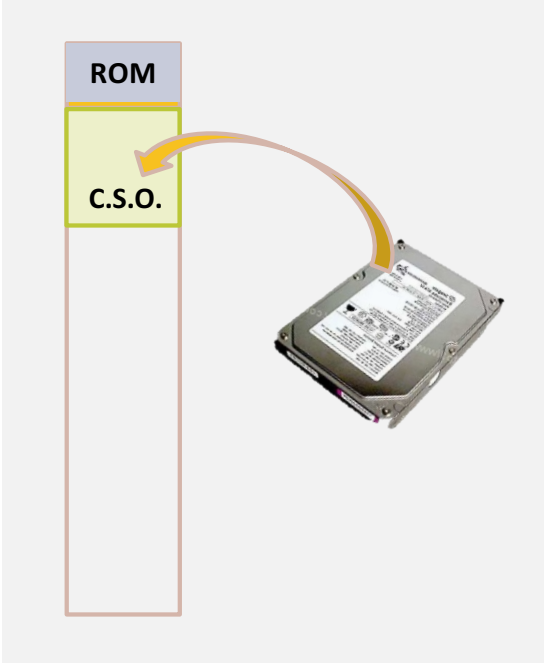


Arranque del computador

- 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)

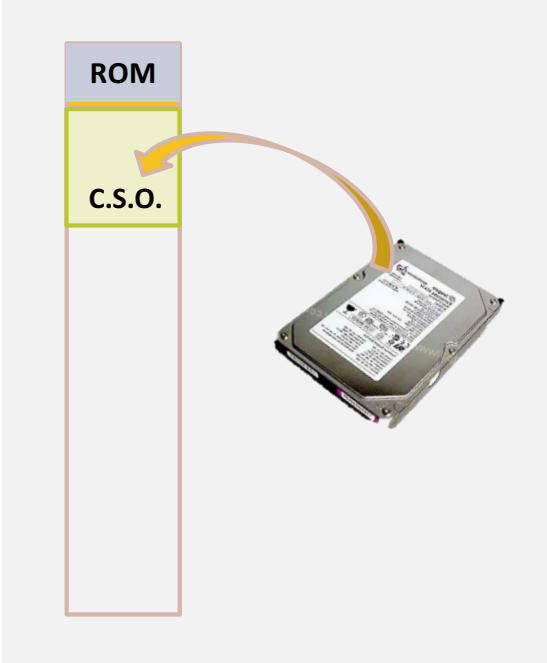


Arranque del computador



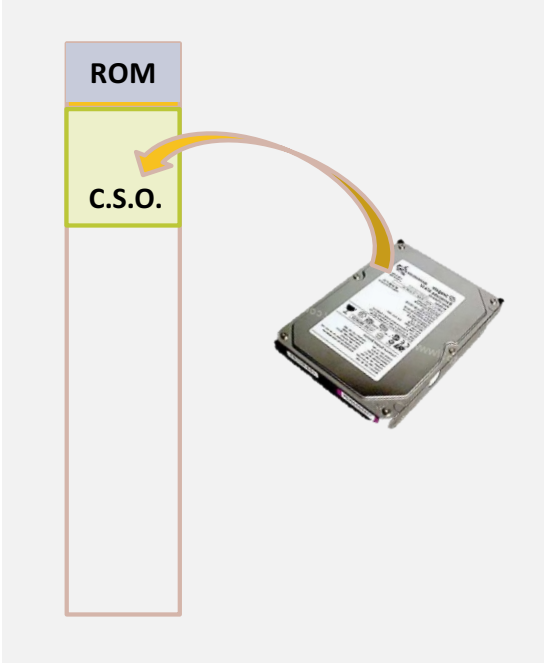
- El *Reset* carga en los registros sus valores predefinidos
  - PC ← dirección de arranque del programa iniciador (en memoria ROM)
- Se ejecuta el programa iniciador
  - Test del sistema (POST)
  - Carga en memoria el cargador del sistema operativo (MBR)

Arranque del computador

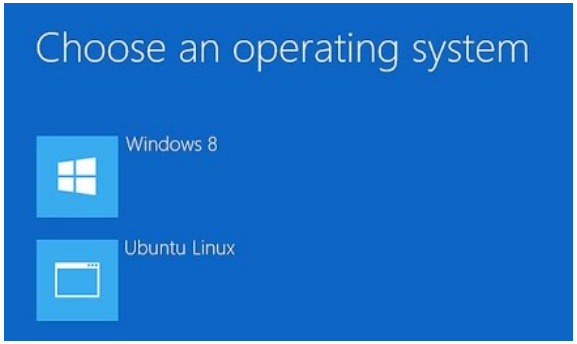


- El *Reset* carga en los registros sus valores predefinidos
  - PC ← dirección de arranque del programa iniciador (en memoria ROM)
- Se ejecuta el programa iniciador
  - Test del sistema (POST)
  - Carga en memoria el cargador del sistema operativo (MBR)

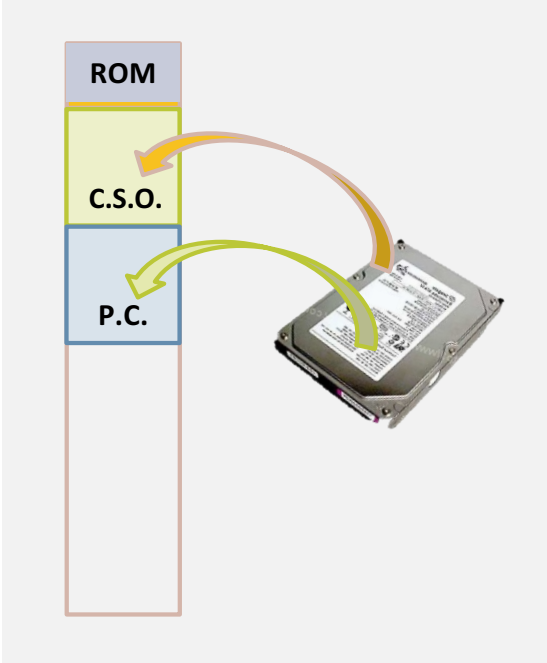
Arranque del computador



- 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

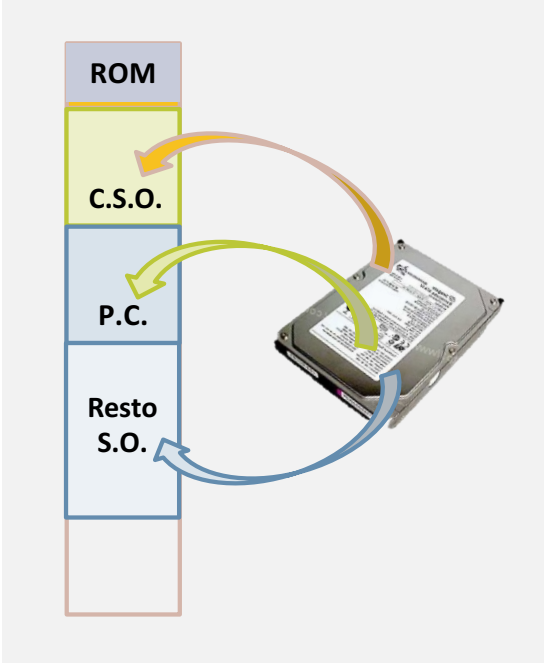


Arranque del computador

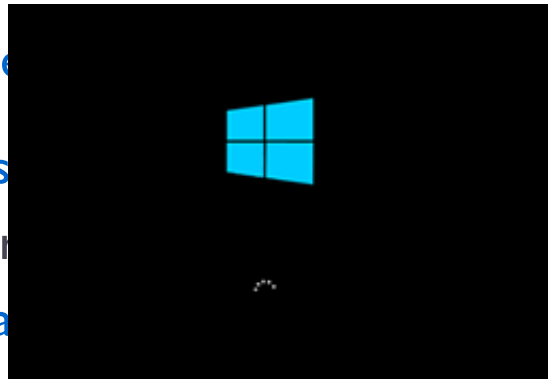


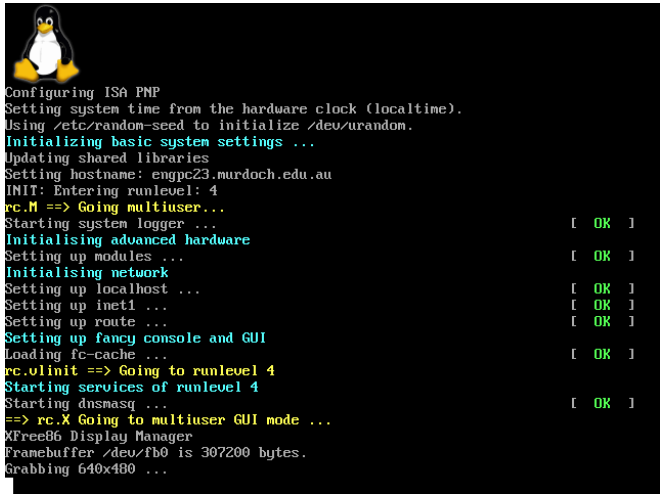
- El *Reset* carga en los registros sus valores predefinidos
  - PC ← dirección de arranque del programa iniciador (en memoria ROM)
- Se ejecuta el programa iniciador
  - Test del sistema (POST)
  - Carga en memoria el cargador del sistema operativo (MBR)
- Se ejecuta el cargador del sistema operativo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

Arranque del computador

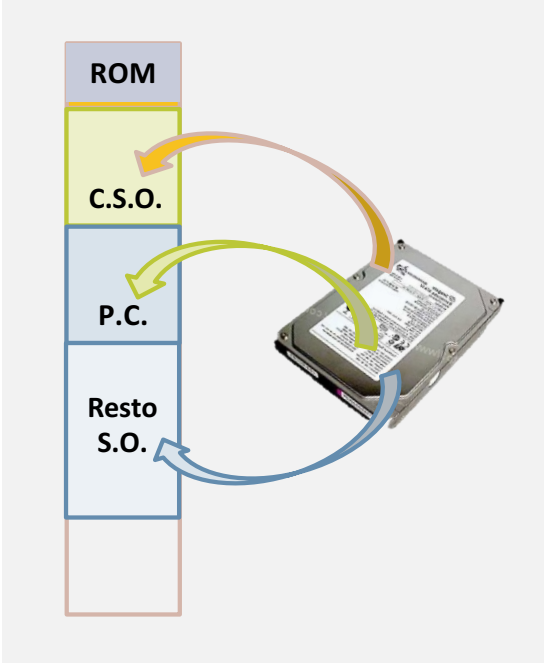


- El *Reset* carga en los registros sus valores predefinidos
  - PC ← dirección de arranque del programa iniciador (en memoria ROM)
- Se ejecuta el programa iniciador
  - Test del sistema (POST)
  - Carga en memoria el cargador del sistema ope
- Se ejecuta el cargador del s
  - Establece opciones de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





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

Tiempo de ejecución de un programa

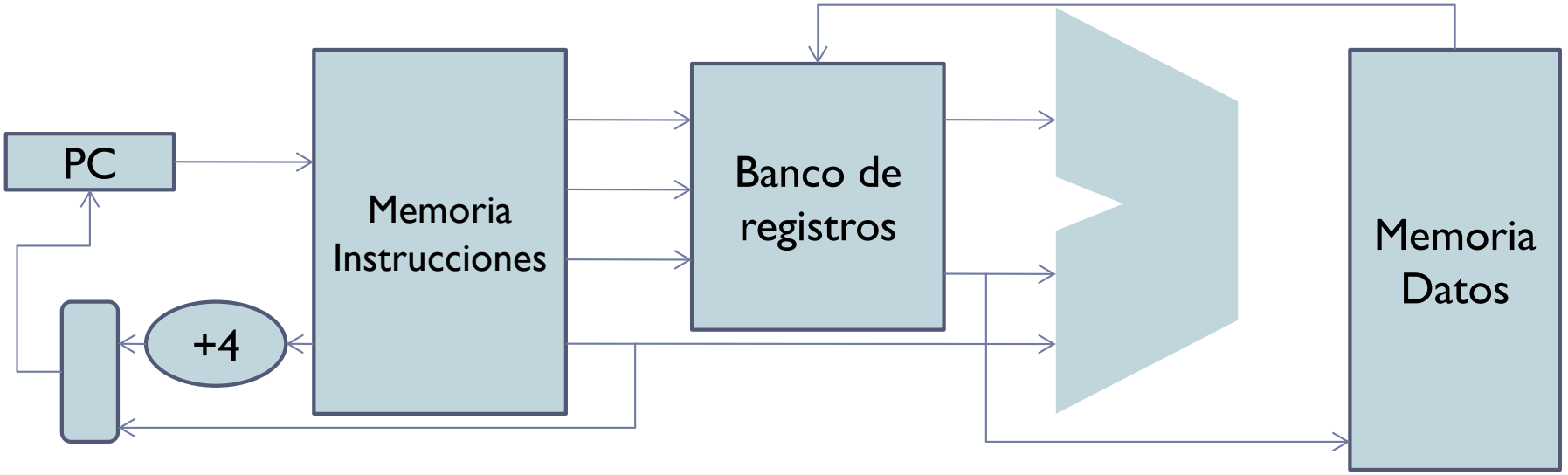
$$\text{Tiempo}_{\text{ejecución}} = NI \times CPI \times t_{ciclo\_CPU} + NI \times AMI \times t_{ciclo\_mem}$$

- NI es el número de instrucciones máquina del programa
- CPI es el número medio de ciclos de reloj necesario para ejecutar una instrucción
- t_{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_{ciclo_mem} es el tiempo de un acceso a memoria

Factores que afecta al tiempo de ejecución

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

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



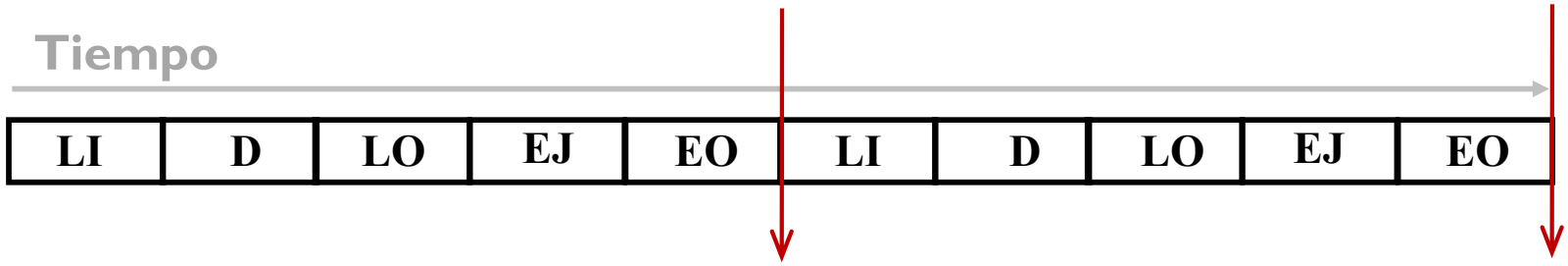
Paralelismo a nivel de instrucción

- 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

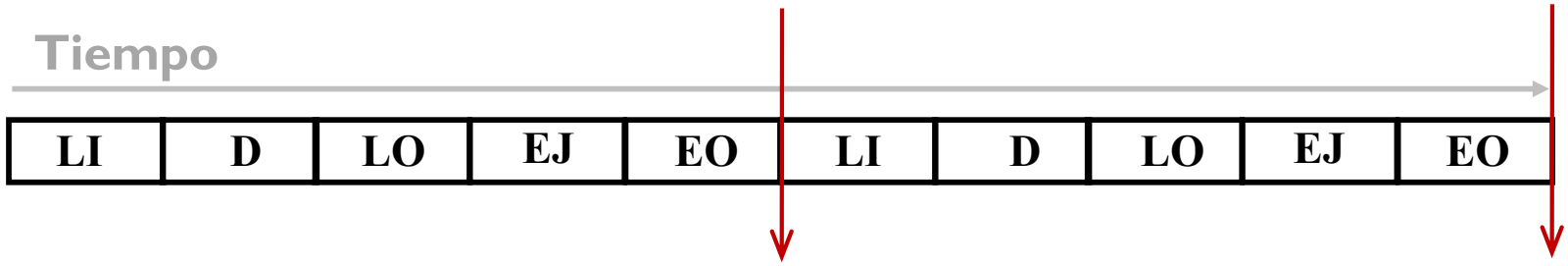
- Etapas de ejecución de una instrucción:
  - LI: Lectura de la instrucción e incremento del PC
  - D: Decodificación
  - LO: Lectura de Operandos
  - EJ: EJecución de la instrucción
  - EO: Escritura de Operandos

Segmentación de instrucciones
sin pipeline



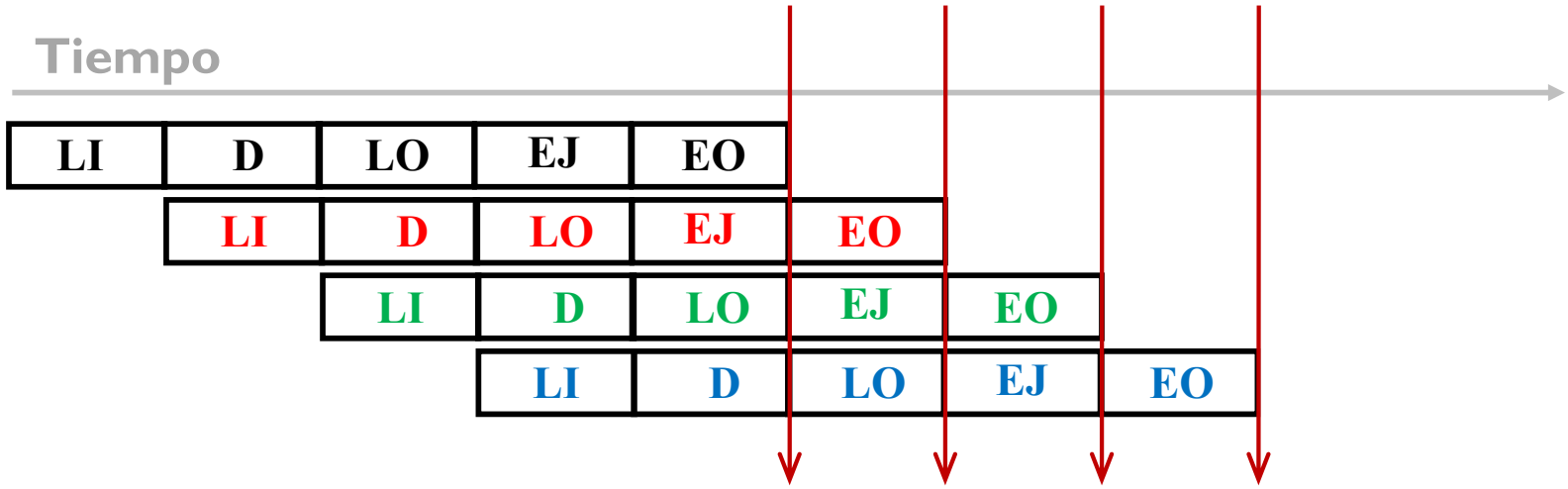
- Etapas de ejecución de una instrucción:
  - LI: Lectura de la instrucción e incremento del PC
  - D: Decodificación
  - LO: Lectura de Operandos
  - EJ: EJecución de la instrucción
  - EO: Escritura de Operandos

Segmentación de instrucciones
sin pipeline



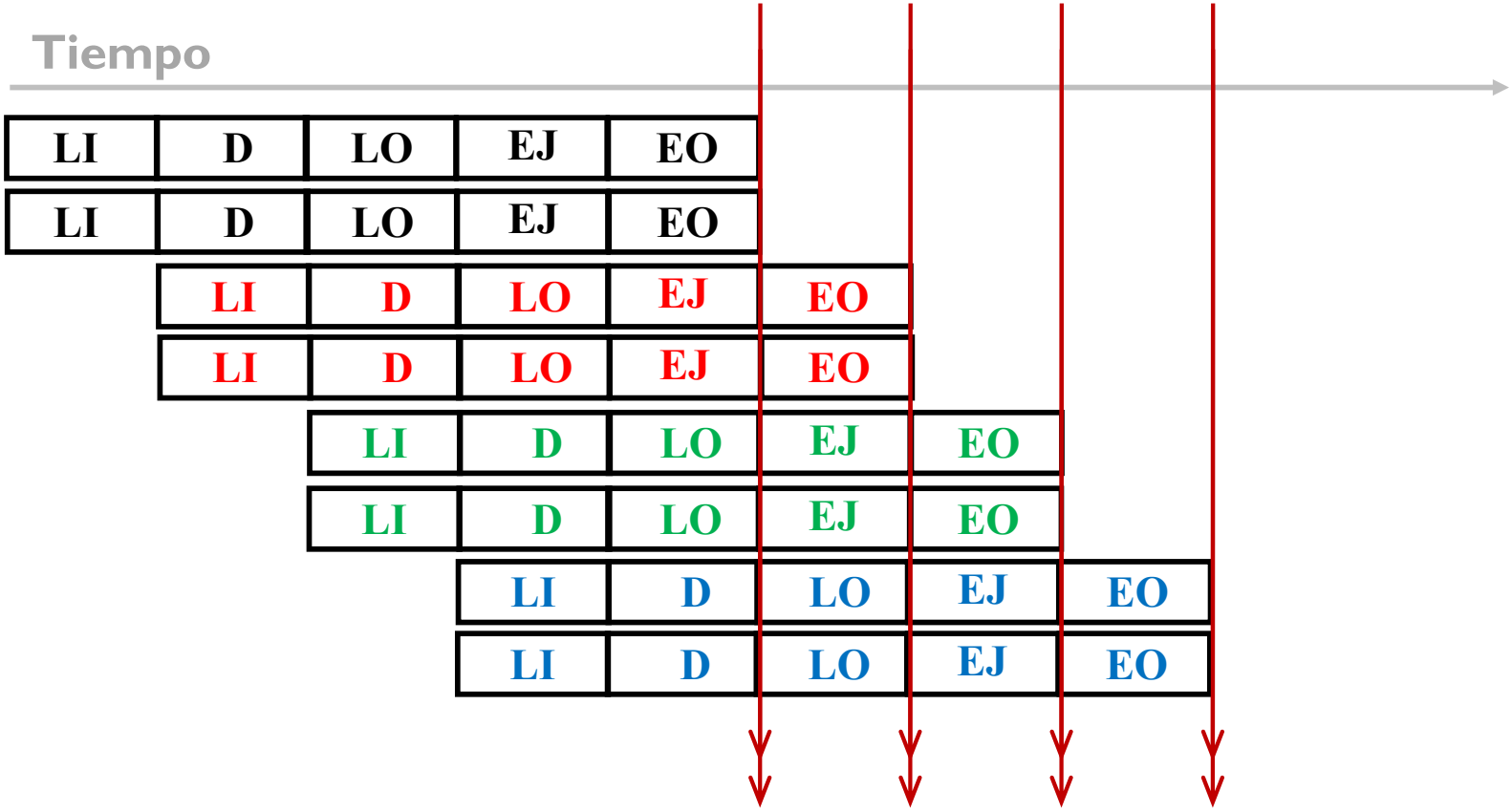
- Si cada fase dura N ciclos de reloj, entonces
  - Una instrucción se ejecuta en 5*N ciclos de reloj
  - Cada N ciclos de reloj se ejecuta 1/5 de instrucción

Segmentación de instrucciones
con pipeline



- Si cada fase dura N ciclos de reloj, entonces
  - Una instrucción se ejecuta en 5*N ciclos de reloj
  - Cada N ciclos de reloj se ejecuta 1 de instrucción

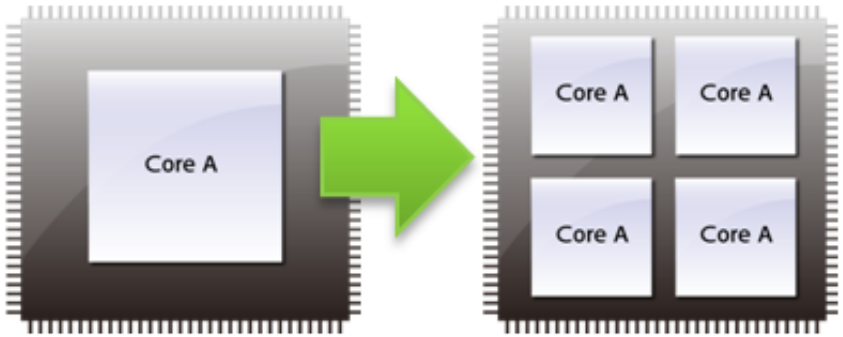
Superescalar

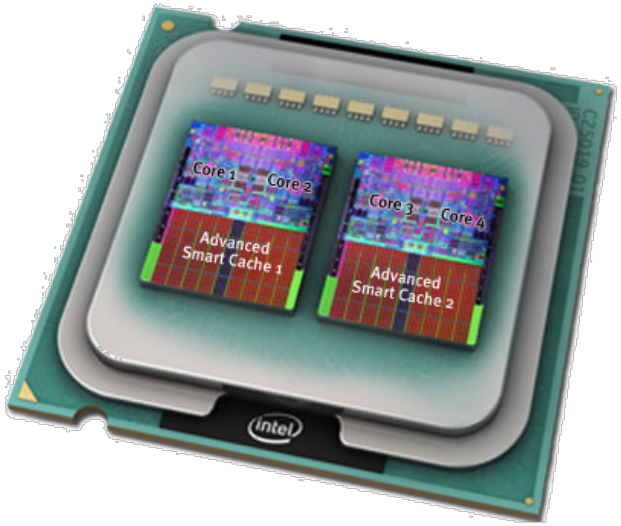


- Pipeline con varias unidades funcionales en paralelo

Multicore

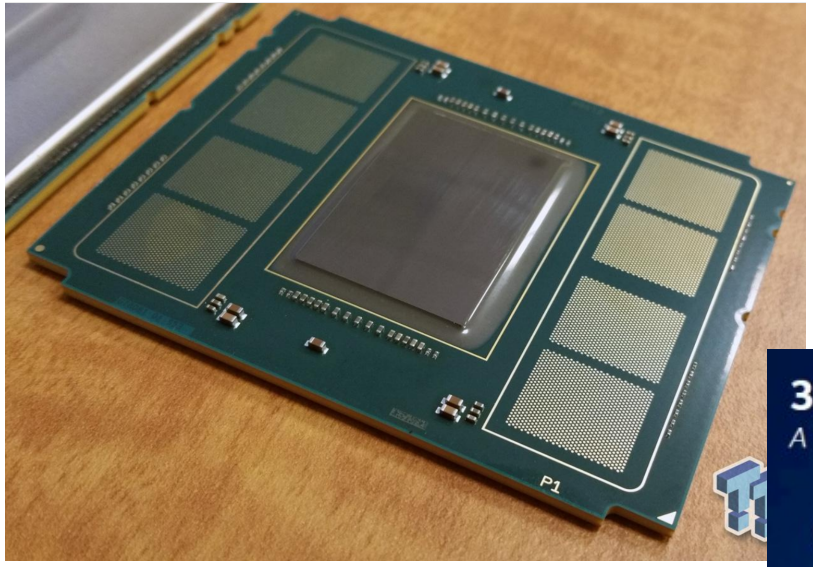
- Múltiples procesadores en el mismo encapsulado

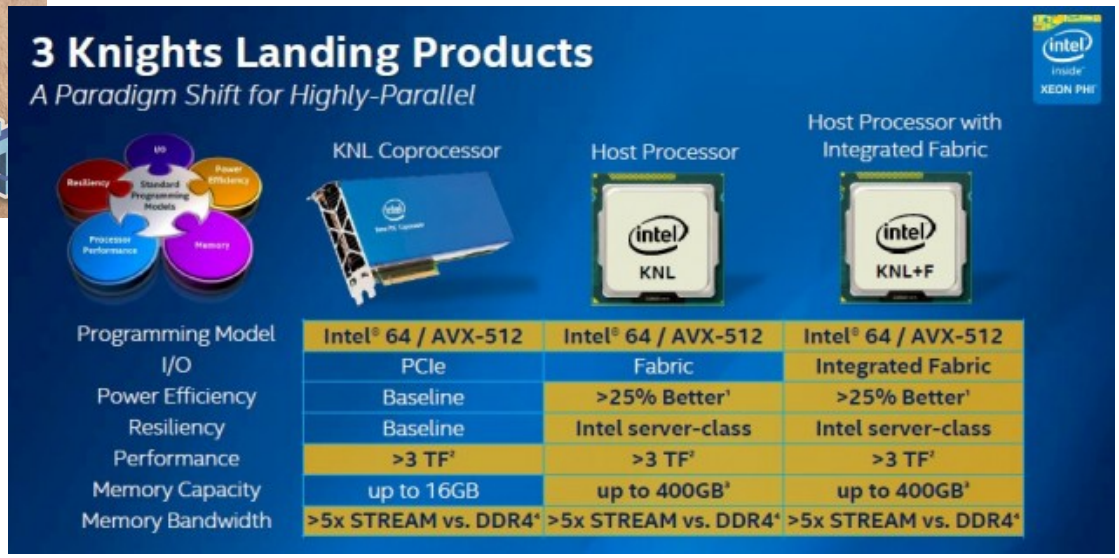




Multicore

- Múltiples procesadores en el mismo encapsulado





http://wccftech.com/intel-knights-landing-detailed-16-gb-highbandwidth-ondie-memory-384-gb-ddr4-system-memory-support-8-billion-transistors/