Tema 1

La CPU:
la unidad de control

- **¿Cómo construir la CPU?**

sistema digital

Bloques elementales

Combinacionales:

puertas

multiplexores

comparadores

descodificadores

sumadores

unidades aritmético-lógicas

memorias

Secuenciales síncronos (clk):

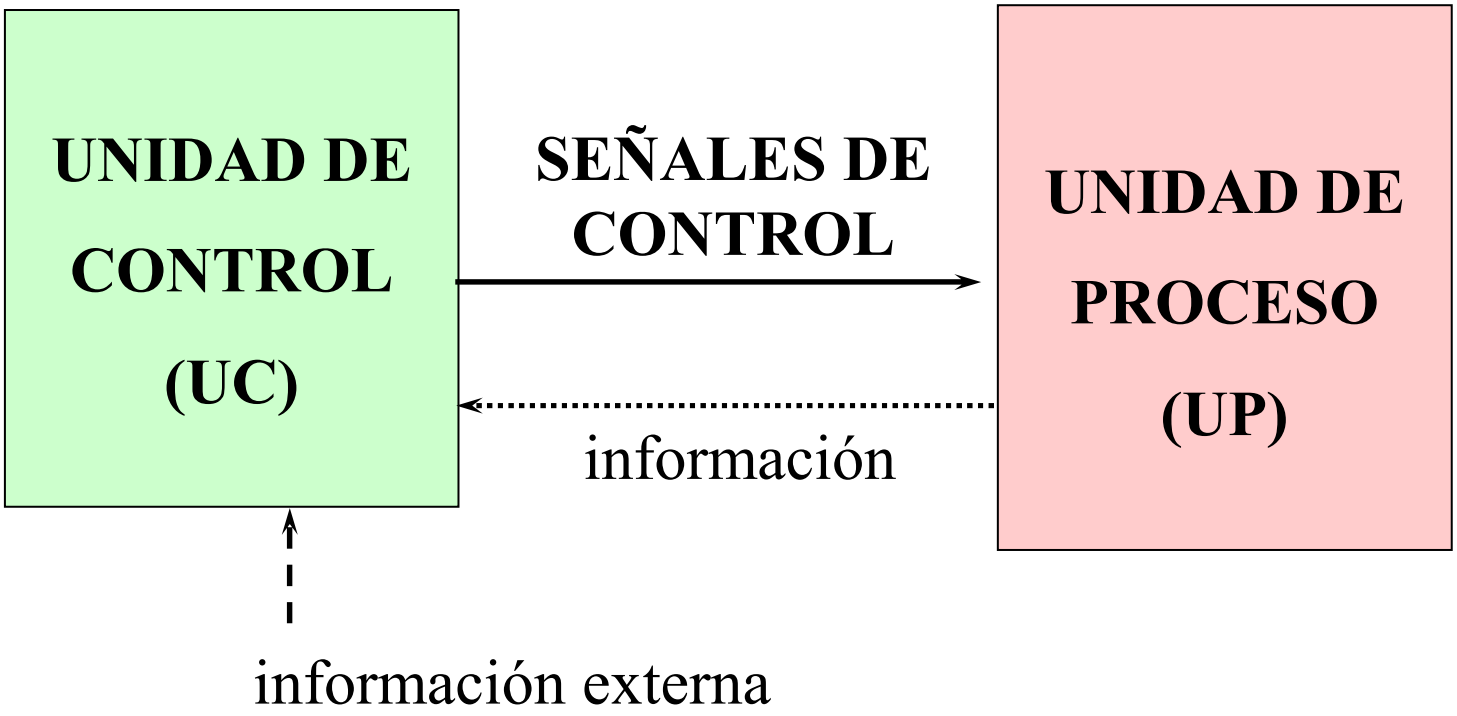
biestables

registros

contadores

Organización de la CPU

estructura genérica de un sistema digital



Estructura genérica de un sistema digital

La Unidad de Control (UC): tema 1

estructura estándar

funcionamiento específico dado por el algoritmo de control

La Unidad de Proceso (UP):

1) sistema de aplicación específica

estructura dependiente de la aplicación concreta

2) sistema de propósito general: procesador

estructura: ruta de datos (*data path*)

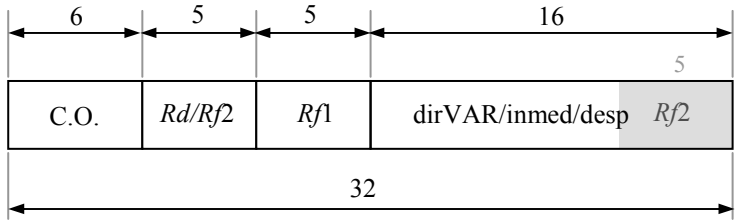
registros + Unidad Aritmético-Lógica: tema 2

Circuitos aritméticos (algoritmos)

Un procesador simple: BIRD

- 42 instrucciones de 32 bits
  - código de operación: 6 bits
  - 4 modos de direccionamiento:
    - inmediato
      - tamaño de los datos: 16 bits
    - absoluto
      - tamaño de la memoria: 64K x 16 bit → direcciones: 16 bits
    - directo de registro
      - conjunto de registros de 32 registros: 5 bits para direccionar los registros
    - indexado
      - dirección base de memoria: 16 bits; índice: contenido de un registro

Formato genérico de las instrucciones:

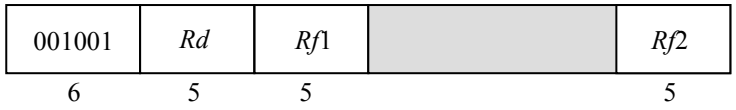


Algunas instrucciones de BIRD (1)

Instrucciones aritméticas y lógicas:

Formato	Lenguaje ensamblador	Operación

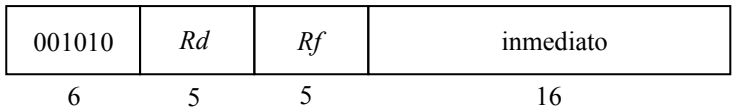
`add`



`add rd,rf1,rf2` `rd:= rf1 + rf2`

(instrucciones similares: `sub`, `mul`, `div`, `or`, `and`, `xor`)

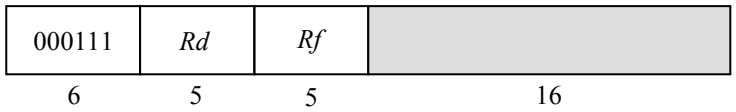
`addi`



`addi rd,rf,#inmed` `rd:= rf + inmed`

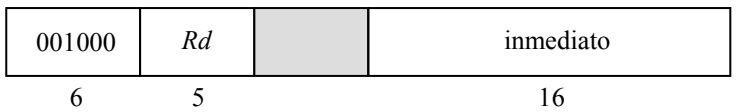
(instrucciones similares: `subi`, `muli`, `divi`, `ori`, `andi`, `xori`)

`mov`



`mov rd,rf` `rd:= rf`

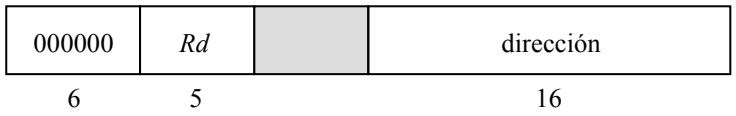
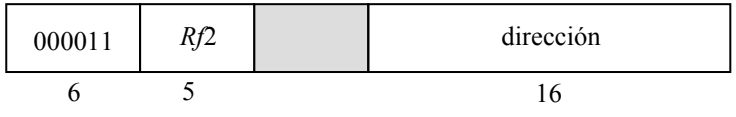
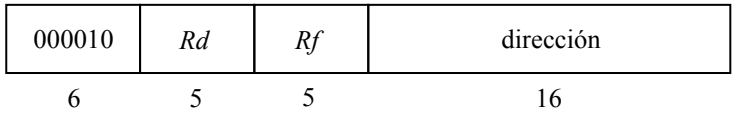
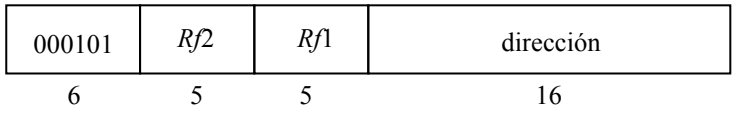
`movi`



`movi rd,#inmed` `rd:= inmed`

Algunas instrucciones de BIRD (2)

Instrucciones que operan con la memoria:

Formato	Lenguaje ensamblador	Operación
ld 	`ld rd,VARIABLE`	`rd:= MEM[dirección]`
st 	`st rf2,VARIABLE`	`MEM[dirección]:= rf2`
ldx 	`ldx rd,VAR[rf]`	`rd:= MEM[dirección+rf]`
stx 	`stx rf2,VAR[rf1]`	`MEM[dirección+rf1]:= rf2`

Algunas instrucciones de BIRD (3)

Instrucciones para controlar el flujo de los programas (saltos):

Formato	Lenguaje ensamblador	Operación

`beq`

`beq rf,etiqueta`

`if (rf=0) then`

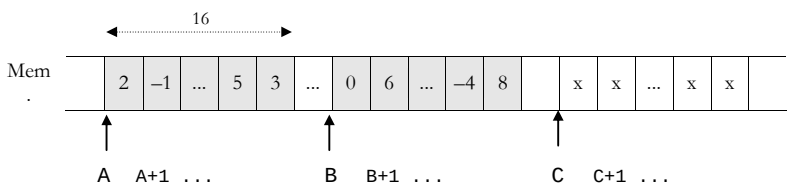
`PC:=` PC_{beq} `+ desplazamiento`

(desplazamiento: distancia entre la instrucción `beq` y la instrucción indicada por la `etiqueta`)

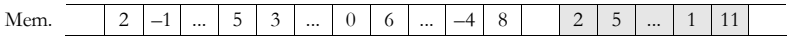
Un programa ejemplo

Sumar dos vectores A y B y dejar el resultado en otro vector, C. Todos ellos de 16 elementos.

inicialmente



al finalizar

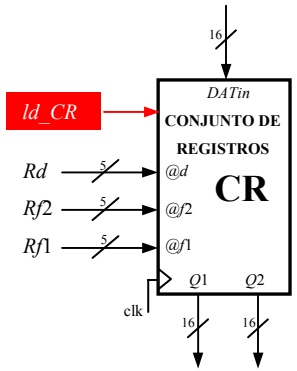


programa para BIRD:

```
        ...
        movi  r1, #0          ; inicializa el registro índice
        movi  r2, #16         ; 16 componentes cada vector
sigue:  ldx   r3, A[r1]       ; lee un elemento del vector A
        ldx   r4, B[r1]       ; lee un elemento del vector B
        add   r5, r4, r3      ; suma ambos elementos (deja el resultado en r5)
        stx   r5, C[r1]       ; guarda el resultado en el elemento correspondiente del vector C
        addi  r1, r1, #1      ; incrementa r1, para obtener el índice del siguiente elemento de los vectores
        subi  r2, r2, #1      ; falta un elemento menos por sumar
        beq   r2, fin         ; si es r2 = 0, se ha terminado la suma de los dos vectores (salta a fin)
        beq   r0, sigue       ; en caso contrario, sigue con la suma (r0 = 0 → salta a sigue)
fin:    ...
```

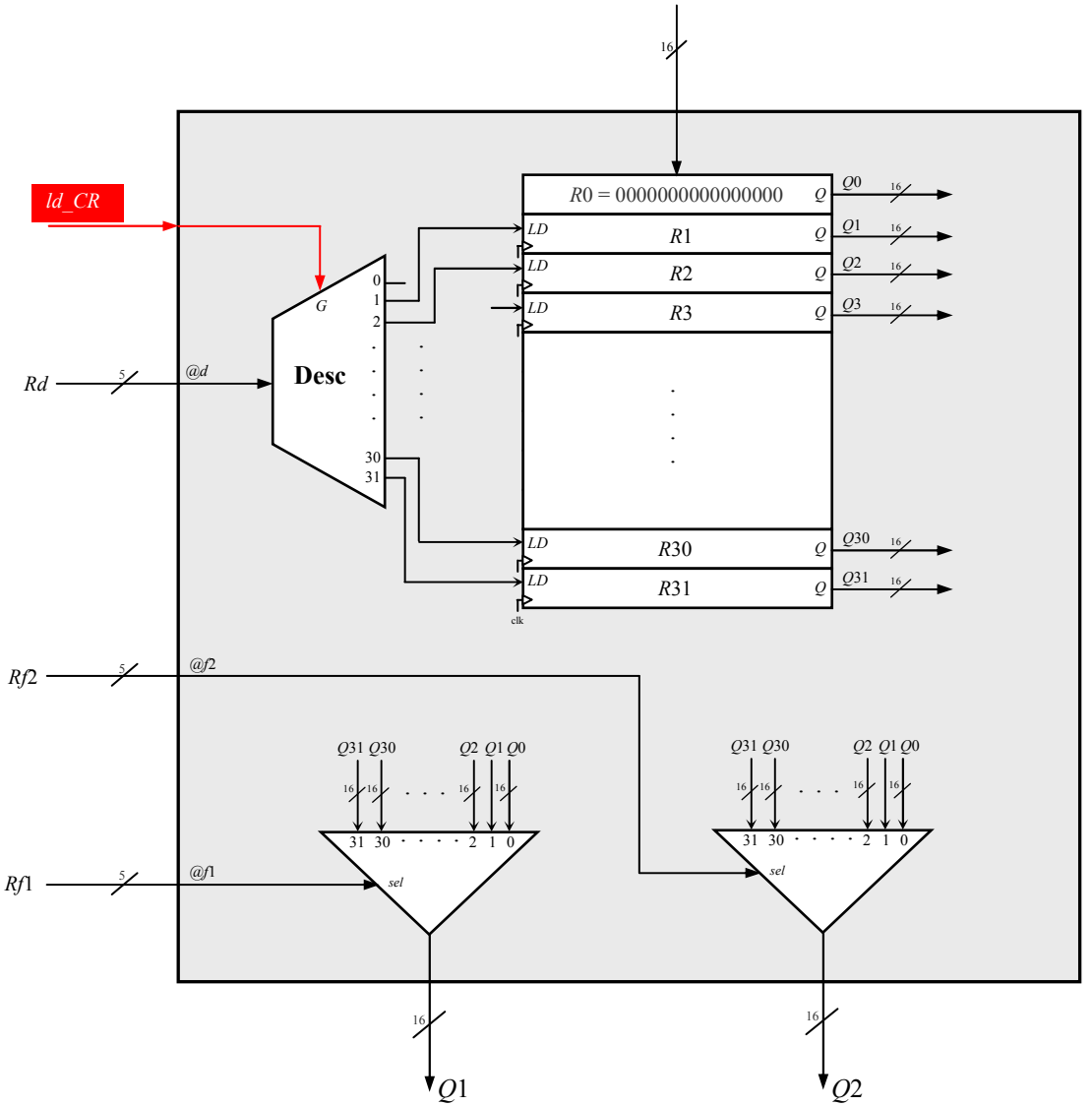
Componentes de la UP de BIRD

CONJUNTO DE REGISTROS

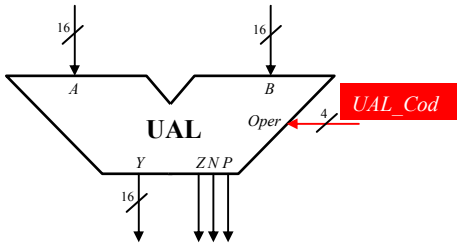



```
CR[@d]:= DATin  (ld_CR & clk ↑)
Q1:= CR[@f1]
Q2:= CR[@f2]
```

ZOOM



UNIDAD ARITMÉTICO-LÓGICA



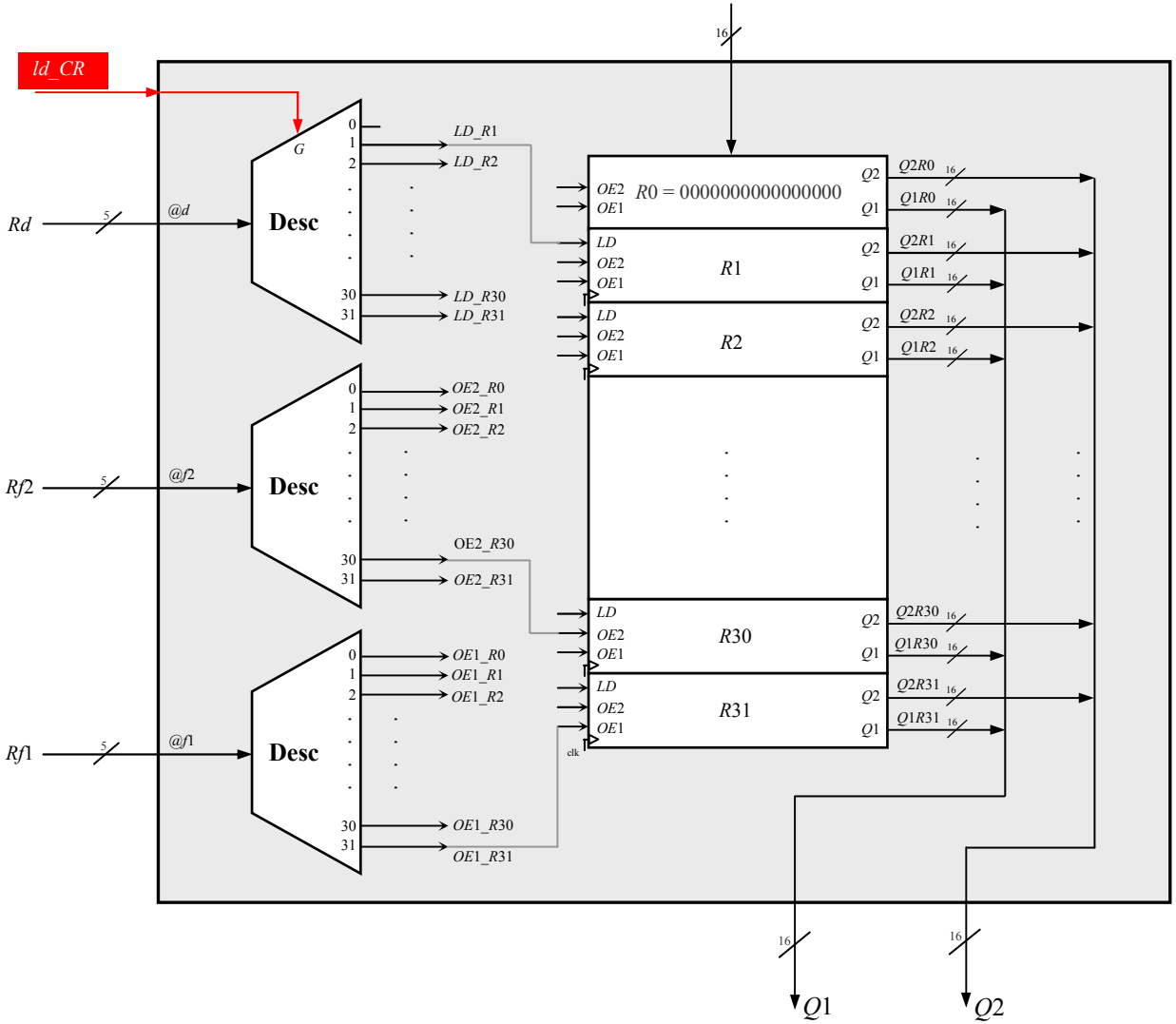
UAL de 16 operaciones:

UAL_Cod	0000	0001	0010	0011	0100	0101	0110
Y	$A + B$	$A - B$	$A \times B$	A/B	A	B	$A + 1$

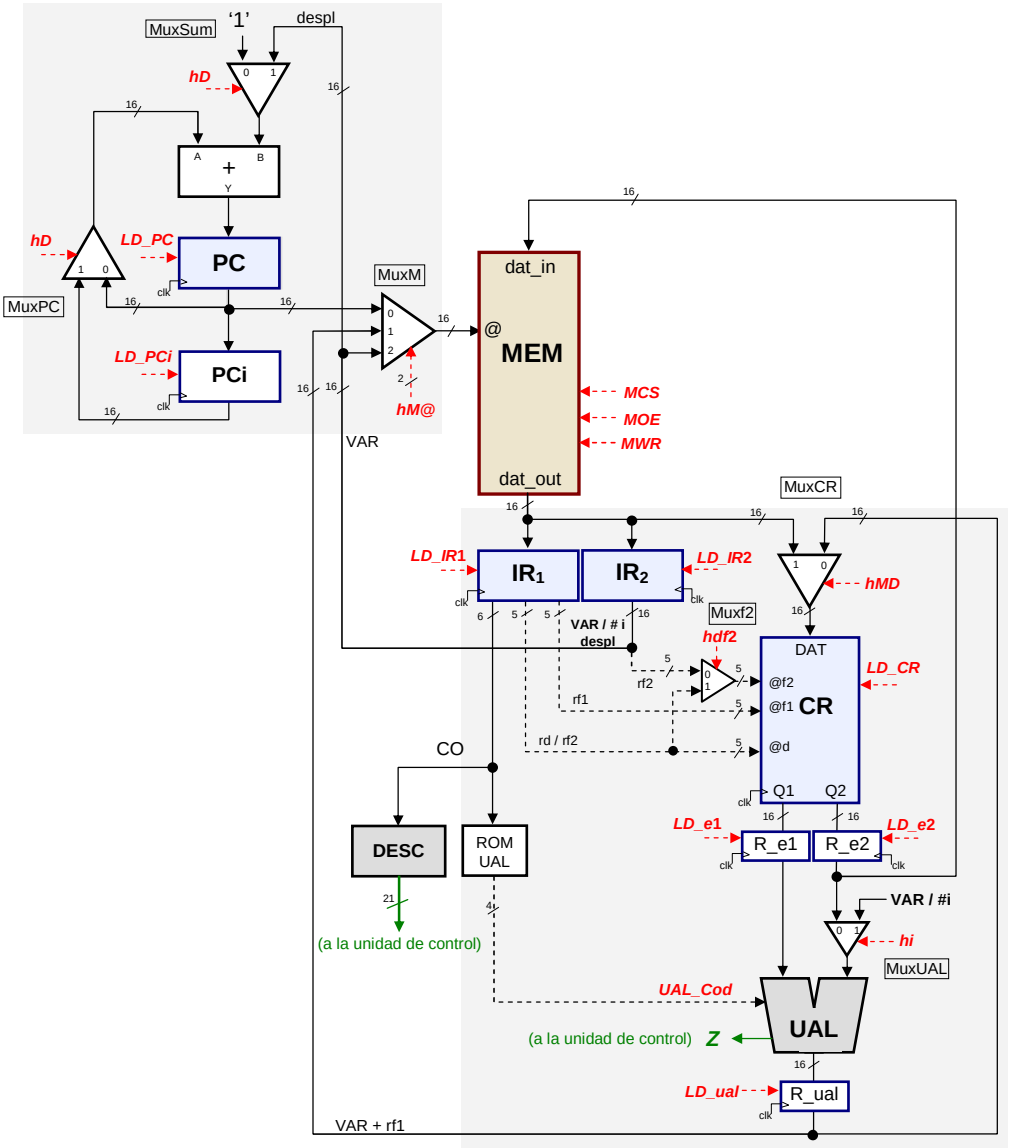
UAL_Cod	0111	1000	1001	1010	1011	1100	...
Y	$A - 1$	*A or B*	*A and B*	*A xor B*	$\leftarrow A$	$\rightarrow A$	...

Componentes de la UP de BIRD

CONJUNTO DE REGISTROS: registros con salida tri-estado

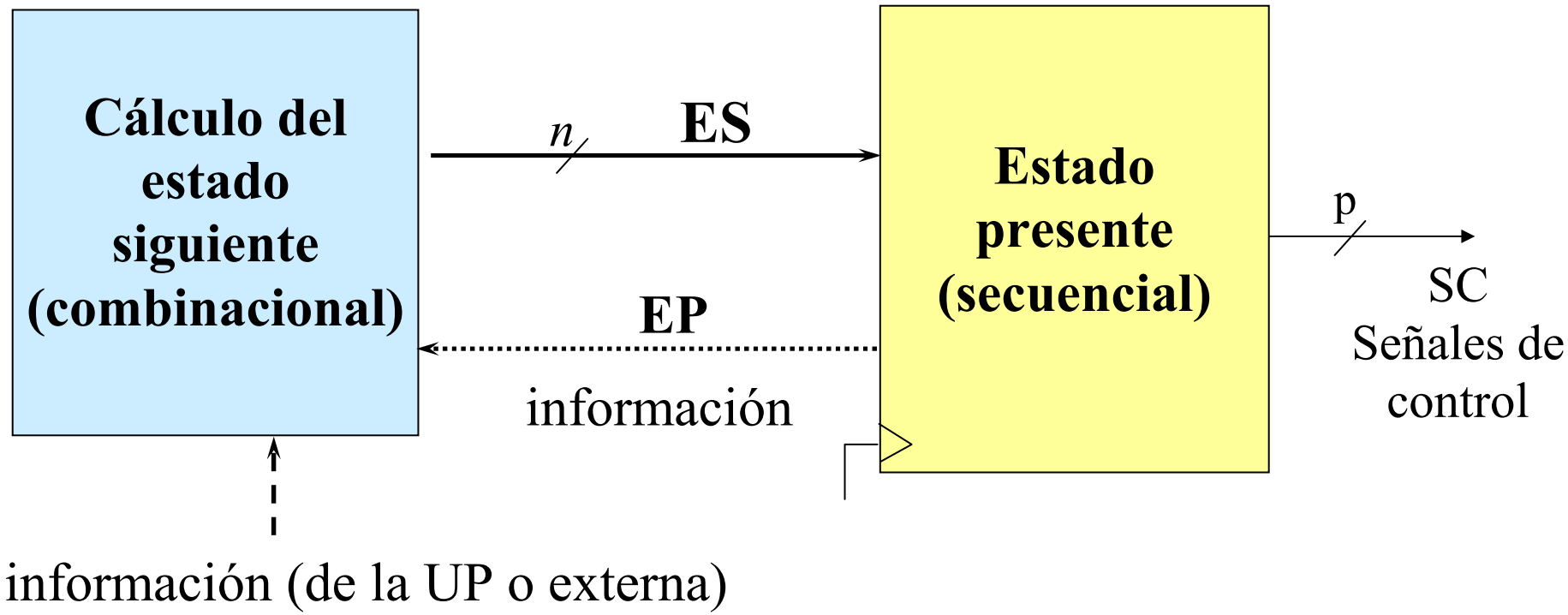


UP de BIRD


MuxSum
'1'
despl
hD
A
B
+
Y
hD
LD_PC
PC
MuxPC
MuxM
LD_PCi
PCi
hM@
VAR
dat_in
@
MEM
dat_out
MCS
MOE
MWR
MuxCR
LD_IR1
IR1
IR2
LD_IR2
hMD
Muxf2
VAR / # i
despl
hdf2
DAT
LD_CR
rf2
@f2
rf1
@f1
CR
rd / rf2
@d
CO
Q1
Q2
LD_e1
R_e1
R_e2
LD_e2
DESC
ROM
UAL
(a la unidad de control)
VAR / #i
hi
MuxUAL
UAL_Cod
UAL
(a la unidad de control)
Z
LD_ual
R_ual
VAR + rf1

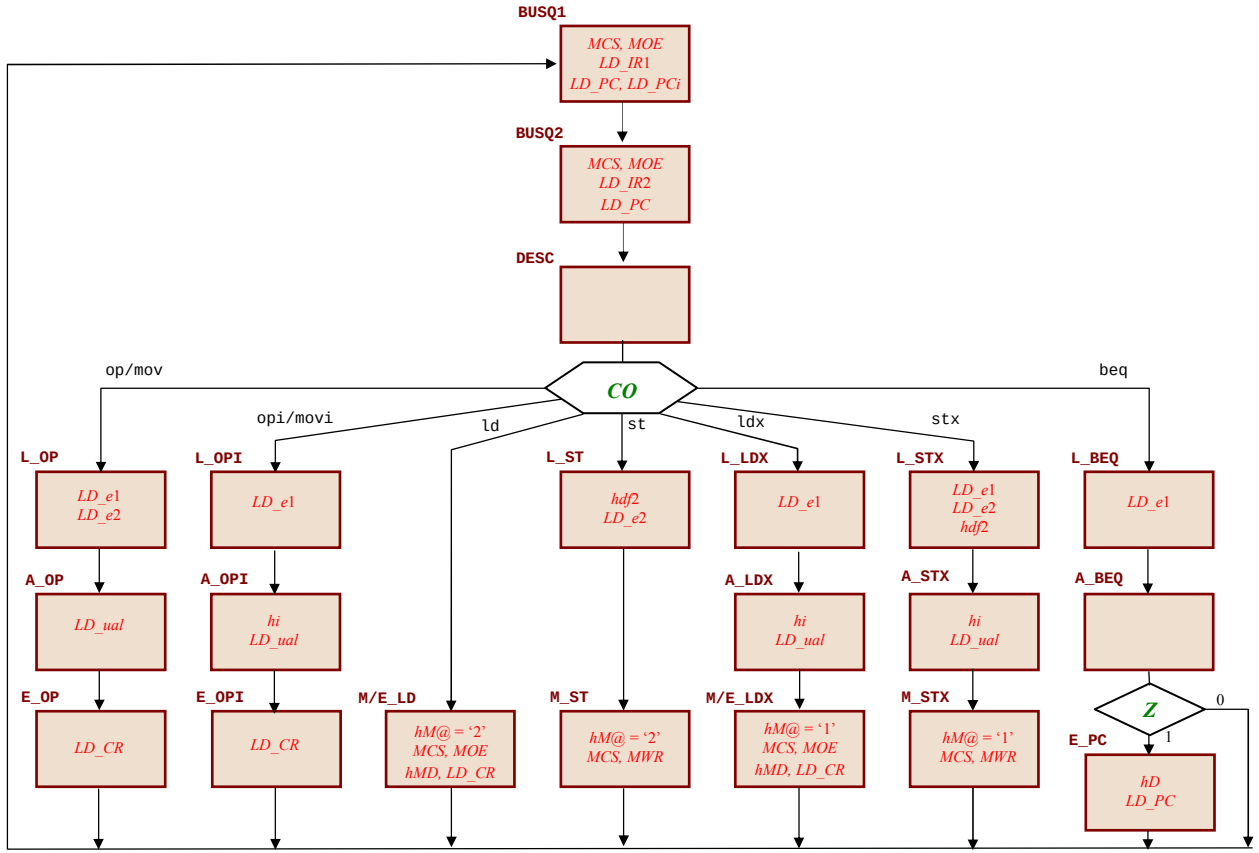
LA UNIDAD DE CONTROL

Estructura genérica: secuenciamiento de estados



Especificación del funcionamiento: algoritmo de control

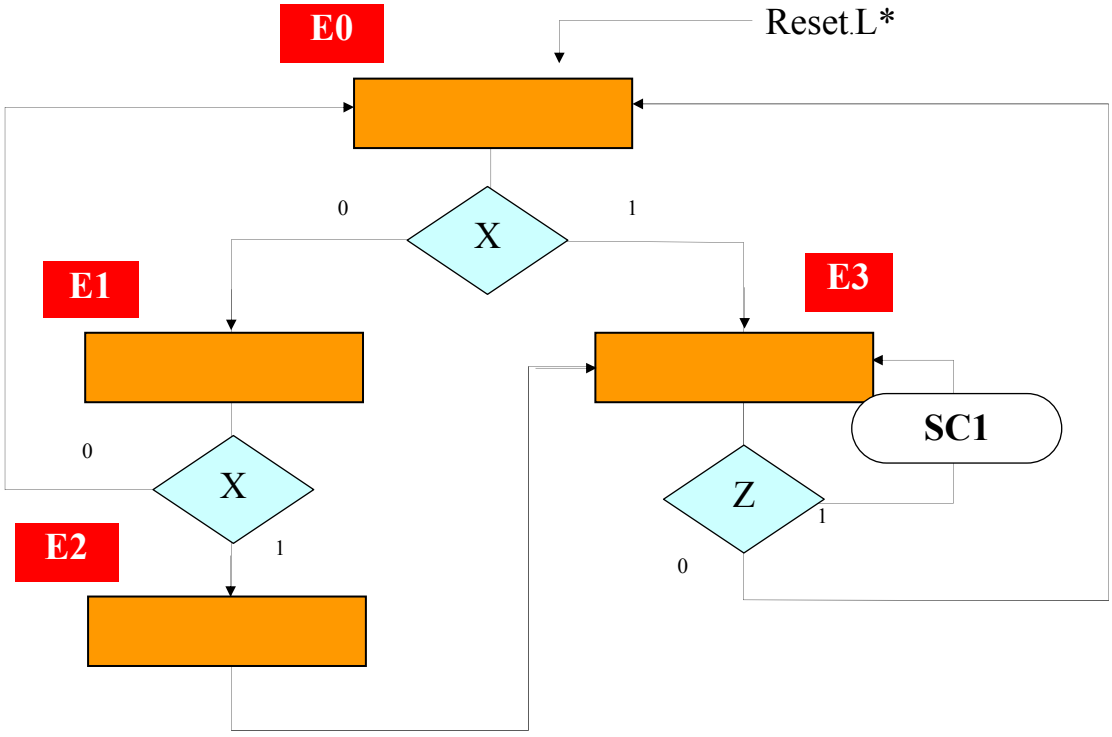
Algoritmo de control de BIRD



- Método para construir el algoritmo: método de los multiplexores
  - Biestables D para guardar el estado presente
  - Multiplexores (y puertas) para obtener el estado siguiente
  - Descodificador (y puertas) para generar las señales de control

- Método cableado (no es único)

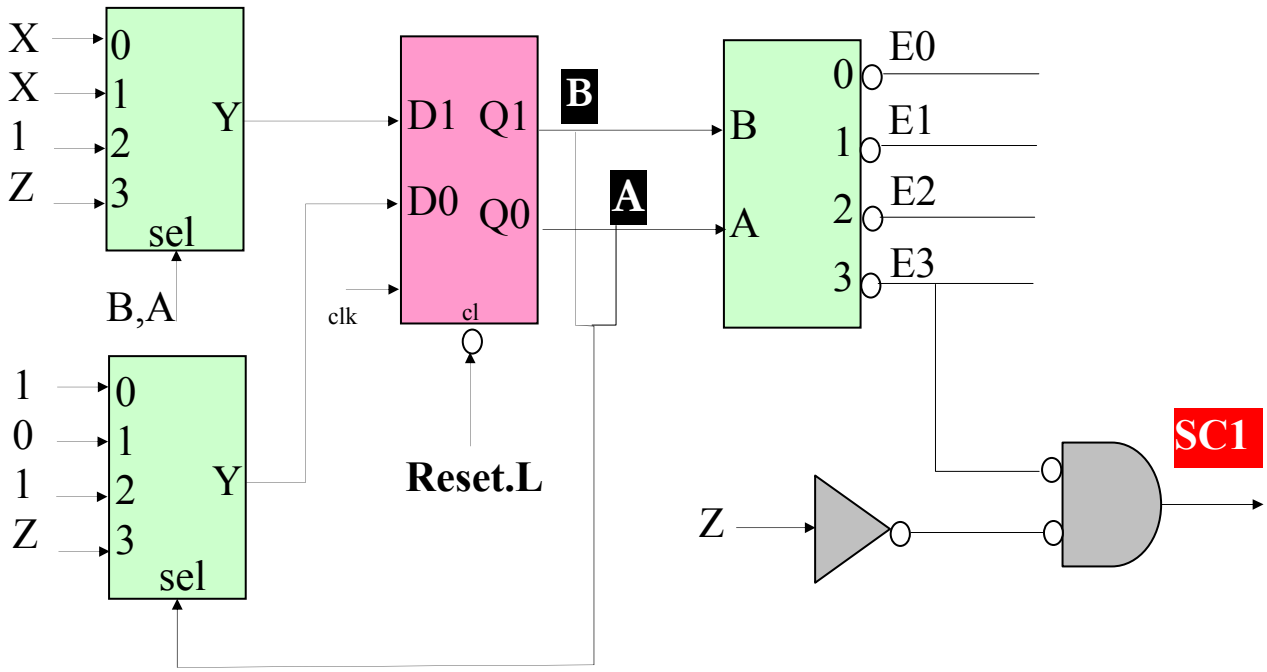
Ejemplo



Multiplexores 4:1

Biestables D

Descodificador 2:4



Ventajas y desventajas del diseño cableado

- ✔ Ejecución rápida
- ✘ lógica de secuenciamiento y ejecución compleja
- ✘ difícil de diseñar y de comprobar su funcionamiento
- ✘ diseño no flexible, no adaptable
- ✘ dificultad de añadir nuevas instrucciones

Técnica empleada básicamente en procesadores RISC

UNIDAD DE CONTROL microprogramada

- Estado → un **contador**

En el flanco: contar $E_i \rightarrow E_{i+1}$

o

cargar (salto) $E_i \rightarrow E_j$

- Información relacionada con los estados: en una **MEMORIA ROM**
  tabla de transición de estados + señales de control

- Funcionamiento:
  secuenciamiento de estados y ejecución de ciertas operaciones en cada estado

- Por analogía: microprograma (*firmware*)

  secuencia de microinstrucciones

- Una microinstrucción = 1 estado del algoritmo de control

  no se admiten salidas condicionales y sólo un cualificador por estado

- Formato de las microinstrucciones

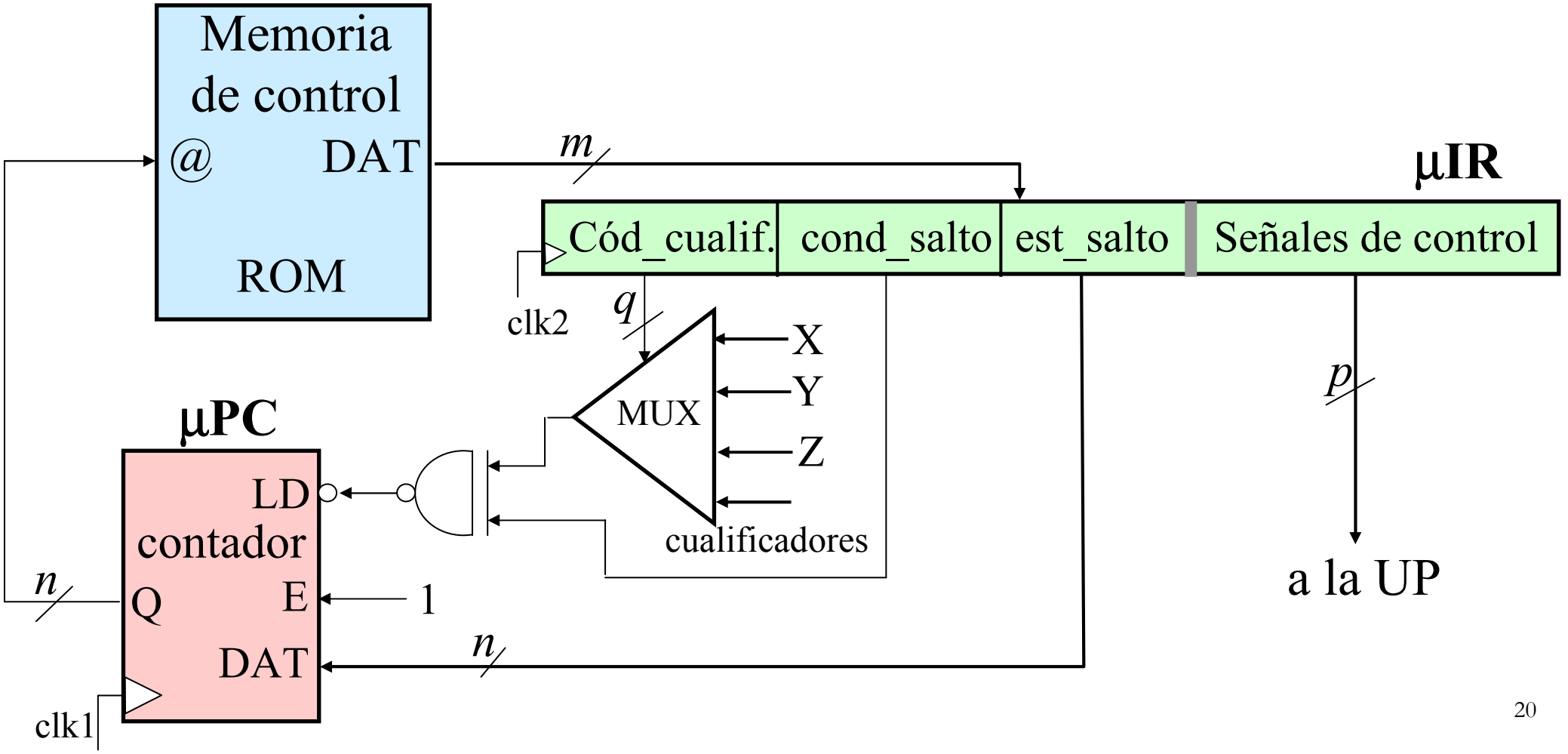
Una opción posible (1):

* Un bit para cada SC (valores 0/1)
* El "código" del cualificador o variable de control que se analiza en el EP
* Valor del cualificador que propicia el salto: condición de salto
* Estado al que se salta (único)

formato de las microinstrucciones (1)

Cód_cualif	cond_salto	est_salto	Señales de control

estructura de la UC microprogramada (1)

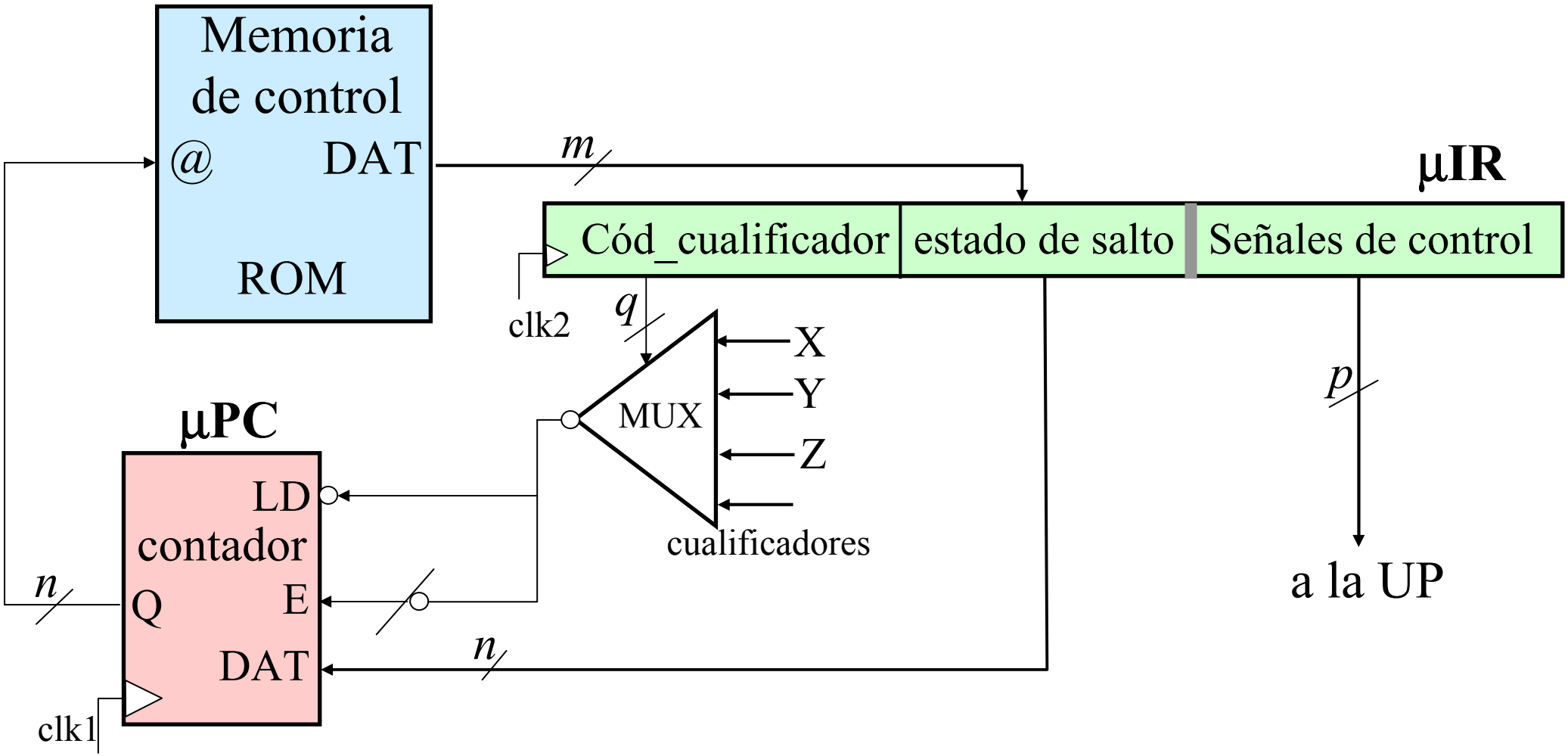


Otra opción posible (2):

Eliminar la condición de salto de la microinstrucción y recodificar los estados del algoritmo de manera que siempre se salte cuando el valor del cualificador sea 1.

formato de las microinstrucciones (2)

estructura de la UC microprogramada (2)



¿Qué ocurre cuando entre dos estados no hay ningún cualificador? ¿Es decir, cuando el paso de un estado a otro es fijo?

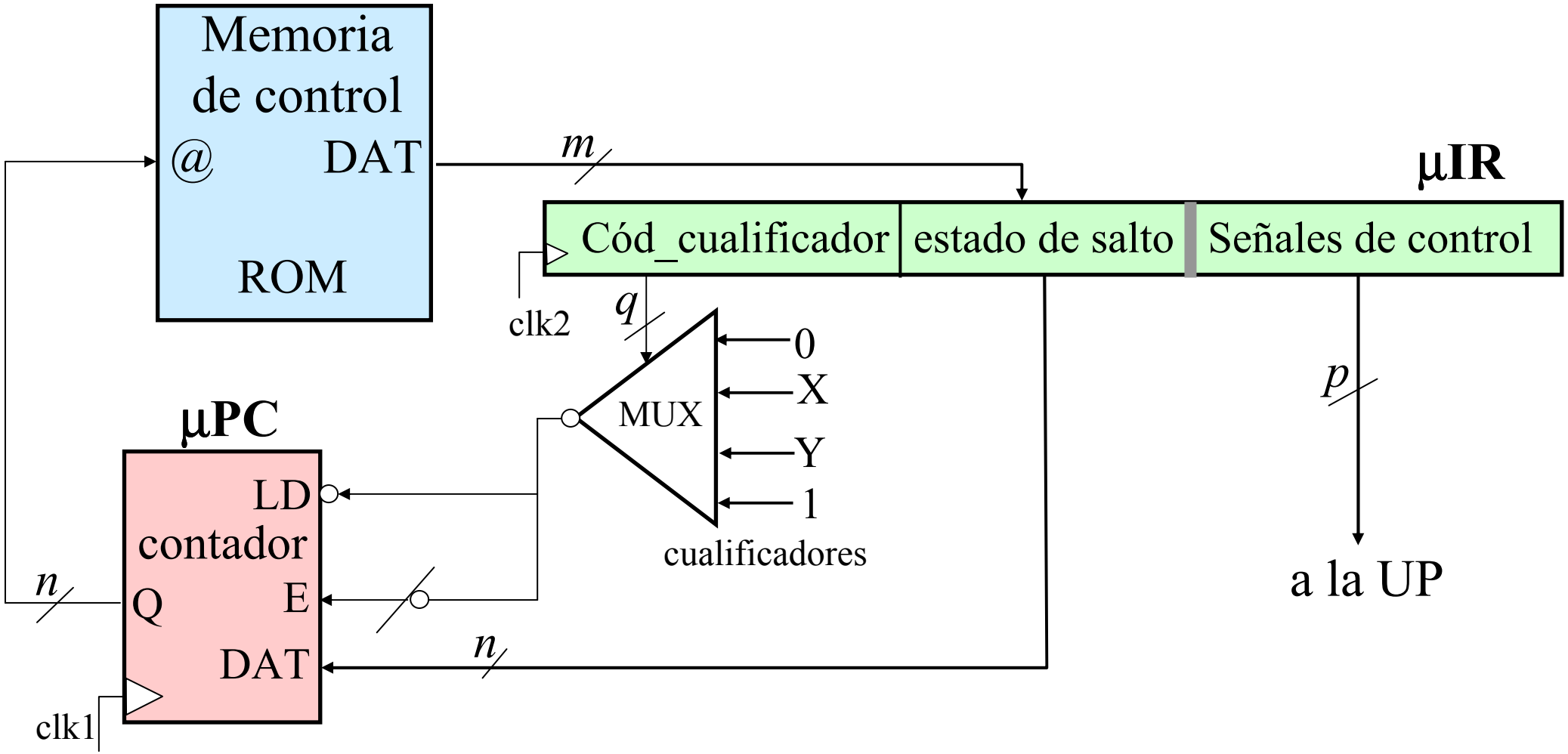
Dos posibilidades:

que el estado siguiente sea el consecutivo, con lo que no se debe producir el salto;

o que no lo sea, con lo que se deberá producir el salto.

Posible solución: añadir a la entrada del multiplexor de la UC dos "cualificadores" nuevos de valores constantes, 0 (no salto) y 1 (salto).

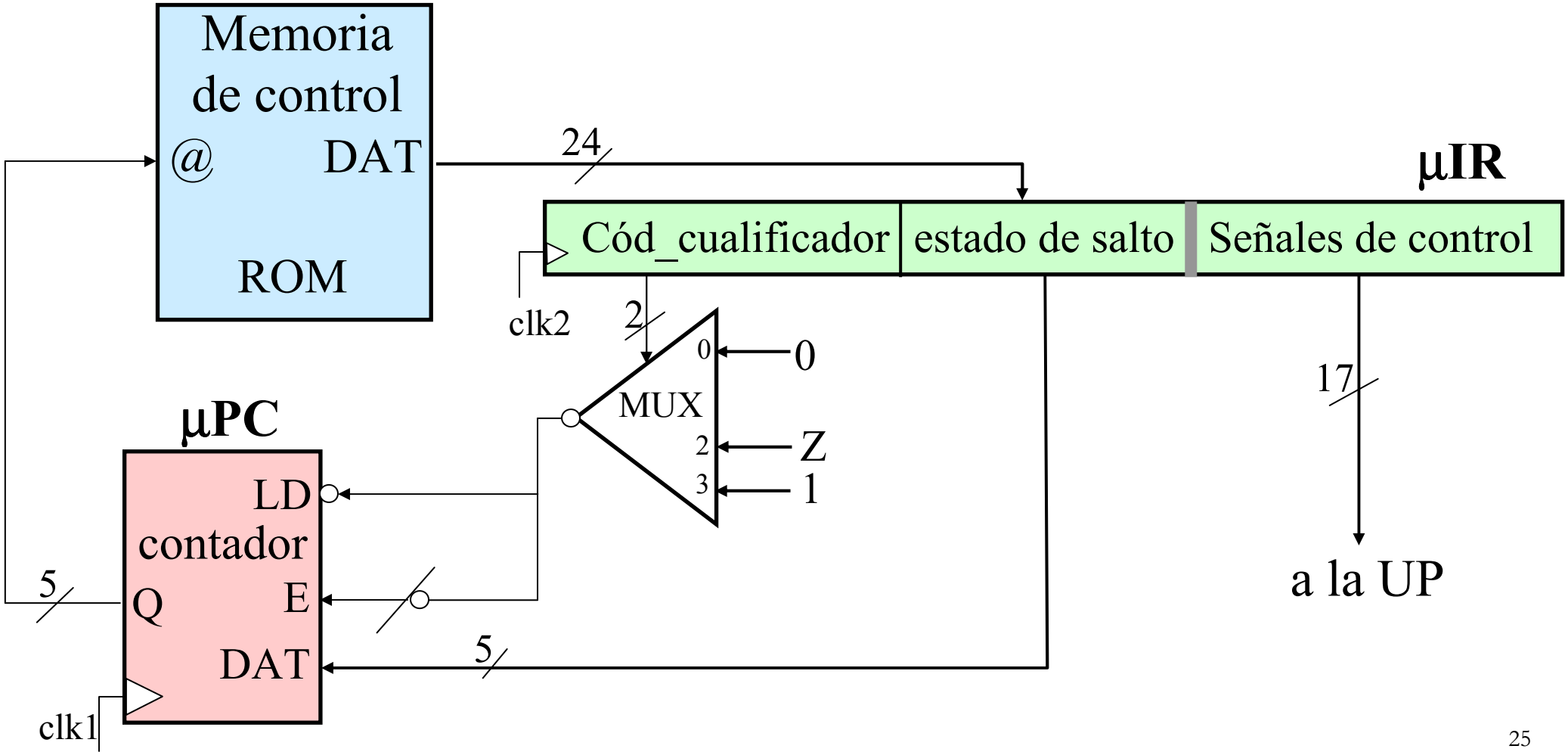
estructura de la UC microprogramada (3)



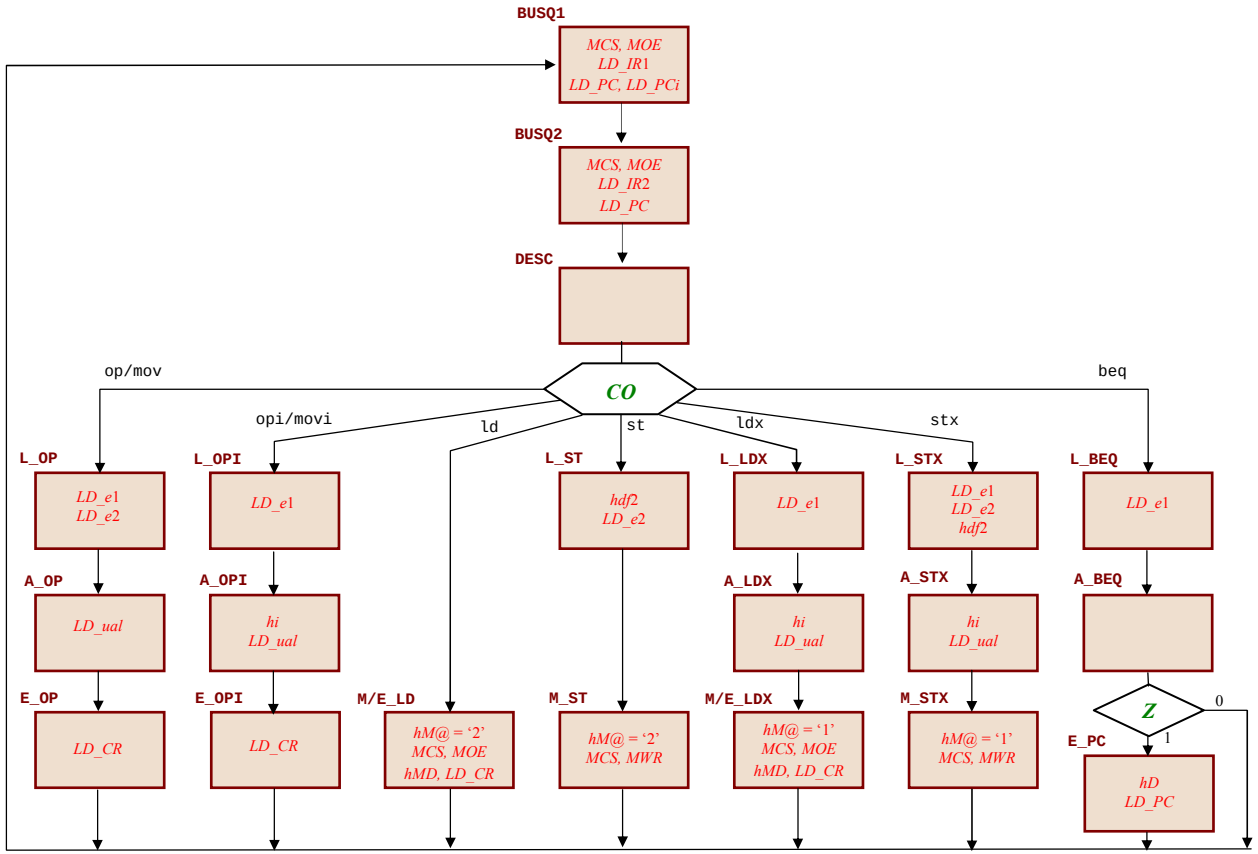
La UC de BIRD microprogramada

1 cualificador, Z (código cualificador: 2 bits / Z, 0, 1)

21 estados (5 bits)



Algoritmo de control de BIRD



Microprograma

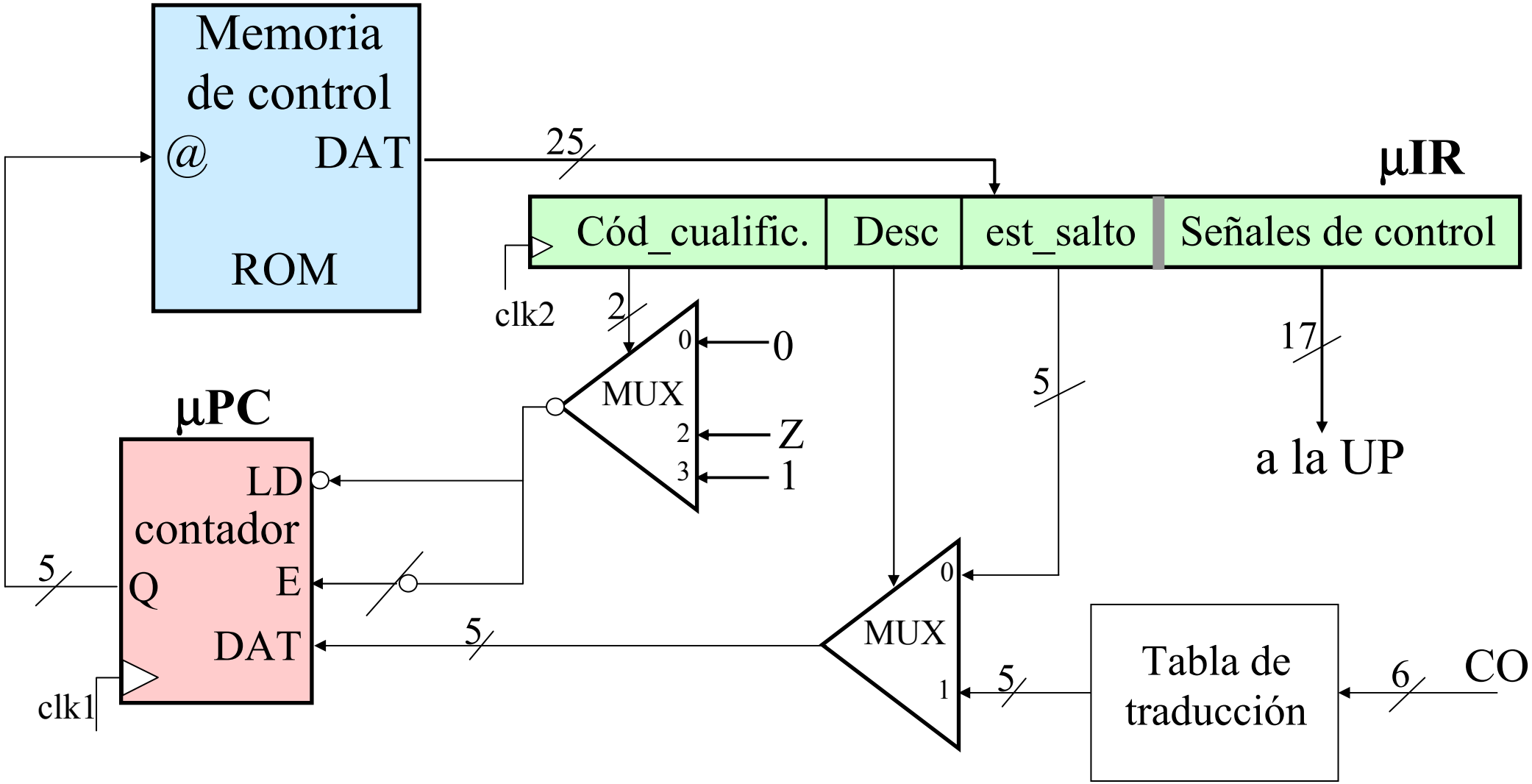
- Lo que hay que decir en cada microinstrucción: Los valores de las señales de control y el estado siguiente: salto al estado consecutivo o a otro estado.

  En binario: latoso

  Lenguaje microensamblador

- El estado de descodificación es especial. No hay un solo estado de salto. Para conseguir eso, más hardware en la unidad de control: una tabla de traducción:

  CO → en qué estado empieza su ejecución.


Memoria
de control
@
DAT
ROM
25
μIR
Cód_cualific.
Desc
est_salto
Señales de control
clk2
2
0
MUX
2
3
0
Z
1
17
a la UP
5
μPC
LD
contador
Q
E
DAT
clk1
5
5
0
MUX
1
5
Tabla de
traducción
6
CO

MICROPROGRAMA DE BIRD

0: ir1:=mem(pc); pc:=pc+1; pci:=pc; /*busq1*/

1: ir2:=mem(pc); pc:=pc+1; /*busq2*/

2: goto código_operación; /*desc*/

3: R_e1:=CR(Rf1); R_e2:=CR(Rf2); /***op**, **mov***/

4: R_ual:=ual_cod(R_e1,R_e2); /*ual-cod: ROM_UAL*/

5: CR(Rd):=R_ual; goto 0;

6: R_e1:=CR(Rf1); /***opi**, **movi** */

7: R_ual:=ual_cod(R_e1,ir2); /*en ir2 está el inmediato #i*/

8: CR(Rd):=R_ual; goto 0;

9: CR(Rd):=mem(ir2); goto 0; /***ld**, 000000*/

10: R_e2:=CR(Rdf2); /***st**, 000011*/

11: mem(ir2):=R_e2; goto 0;

12: R_e1:=CR(Rf1); /***ldx**, 000010*/

13: R_ual:=R_e1+ir2; /*ir2: dirección de la variable VAR*/

14: CR(Rd):=mem(R_ual); goto 0;

15: R_e1:=CR(Rf1); R_e2:=CR(Rdf2); /***stx**, 000101*/

16: R_ual:=R_e1+ir2; /*ir2: dirección de la variable VAR*/

17: mem(R_ual):=R_e2 ; goto 0;

18: R_e1:=CR(Rf1); /***beq**, 011010*/

19:Z:=(R_e1=0); if Z goto 21;

20: goto 0;

21: pc:=pci+ir2; goto 0; /*ir2: desplazamiento*/

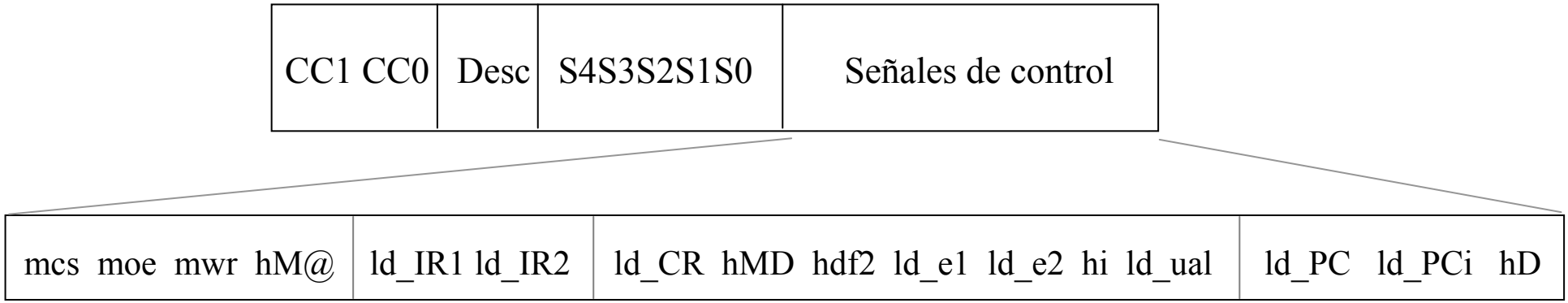
TABLA DE TRADUCCIÓN

Código de operación → dirección de memoria de control donde comienza la fase de ejecución de la instrucción

C.O.	@comienzo
000000	01001 (ld)
000010	01100 (ldx)
000011	01010 (st)
000101	01111 (stx)
000111	00011 (mov)
001000	00110 (movi)
001001	00011 (add)
001010	00110 (addi)
001011	00011 (sub)
001100	00110 (subi)
001101	00011 (mul)

C.O.	@comienzo
001110	00110 (muli)
001111	00011 (div)
010000	00110 (divi)
010011	00011 (and)
010100	00110 (andi)
010101	00011 (or)
010110	00110 (ori)
010111	00011 (xor)
011000	00110 (xori)
011010	10010 (beq)

formato de las microinstrucciones de BIRD



microprograma de BIRD en binario

dirección de memoria		contenido		
(busq1)	00000	00 0 xxxxx	1 1 0 00	1 0	0 x x 0 0 x 0	1 1 0
(busq2)	00001	00 0 xxxxx	1 1 0 00	0 1	0 x x 0 0 x 0	1 0 0
(desc)	00010	11 1 xxxxx	0 0 0 xx	0 0	0 x x 0 0 x 0	0 0 x
(op/mov)	00011	00 0 xxxxx	0 0 0 xx	0 0	0 x 0 1 1 x 0	0 0 x
	00100	00 0 xxxxx	0 0 0 xx	0 0	0 x x 0 0 0 1	0 0 x
	00101	11 0 00000	0 0 0 xx	0 0	1 0 x 0 0 x 0	0 0 x
(opi/movi)	00110	00 0 xxxxx	0 0 0 xx	0 0	0 x x 1 0 x 0	0 0 x
	.....	.....	.....	...	.....	

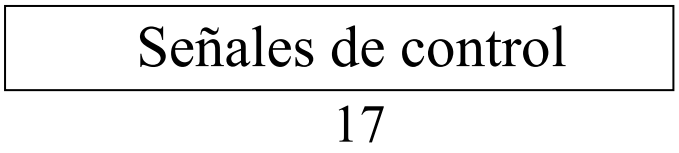
Ventajas y desventajas del diseño microprogramado

- ✔ simplifica el diseño de la UC
  - más barato
  - menos propenso a errores
  - más sistemático
  - fácilmente modificable
- ✔ posibilidad de emular diferentes conjuntos de instrucciones
- ✘ Ejecución más lenta

Empleado básicamente en procesadores CISC

métodos alternativos de microprogramación

microprogramación horizontal: cada microinstrucción especifica muchas microoperaciones diferentes que se deben realizar en paralelo

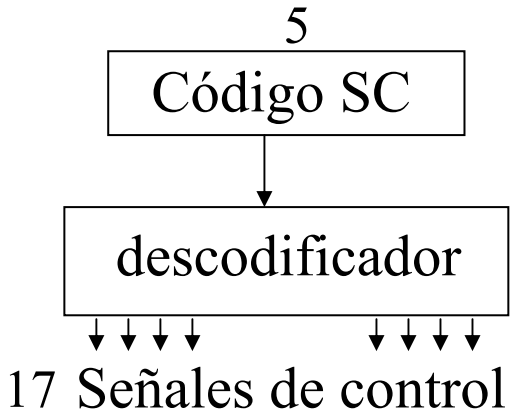


microprogramación vertical: cada microinstrucción especifica una sola microoperación o muy pocas

no todas las señales de control actúan simultáneamente: codificarlas

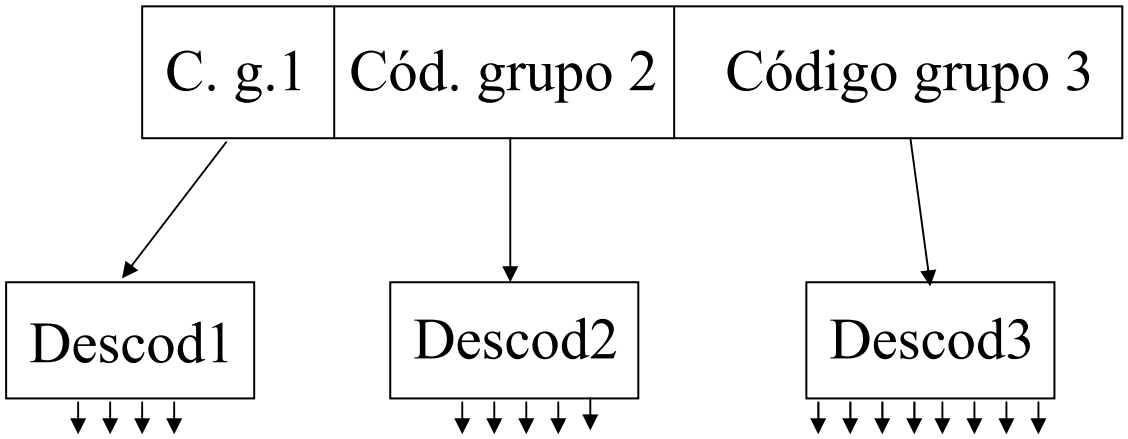
se alarga el microprograma

incapacidad de paralelismo



solución intermedia: cada microinstrucción especifica varias microoperaciones

las señales de control se codifican por grupos disjuntos



nanoprogramación: cada microinstrucción se ejecuta mediante una secuencia de pasos: nanoinstrucciones ⇨ nanoprograma

ventaja: ahorro en el tamaño total de la memoria de control

desventajas: mayor complejidad

operación más lenta