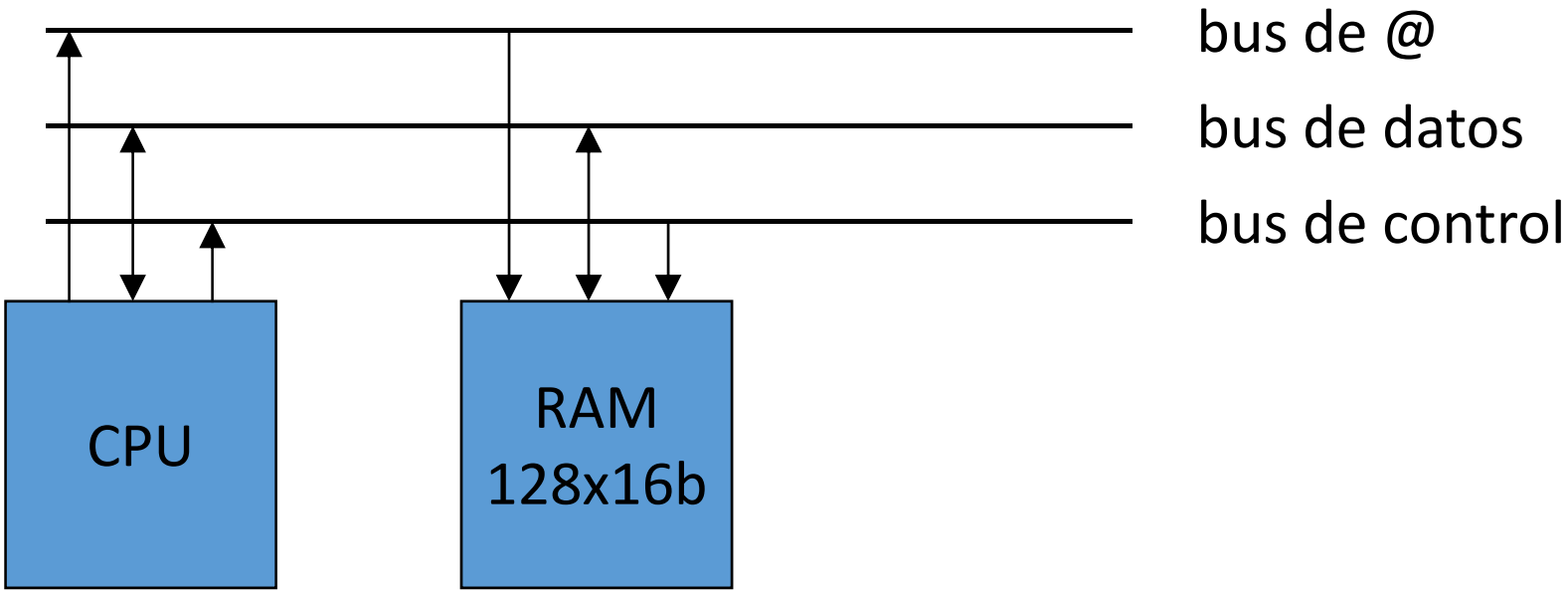
Diseño de la Máquina Sencilla

Introducción a los Computadores

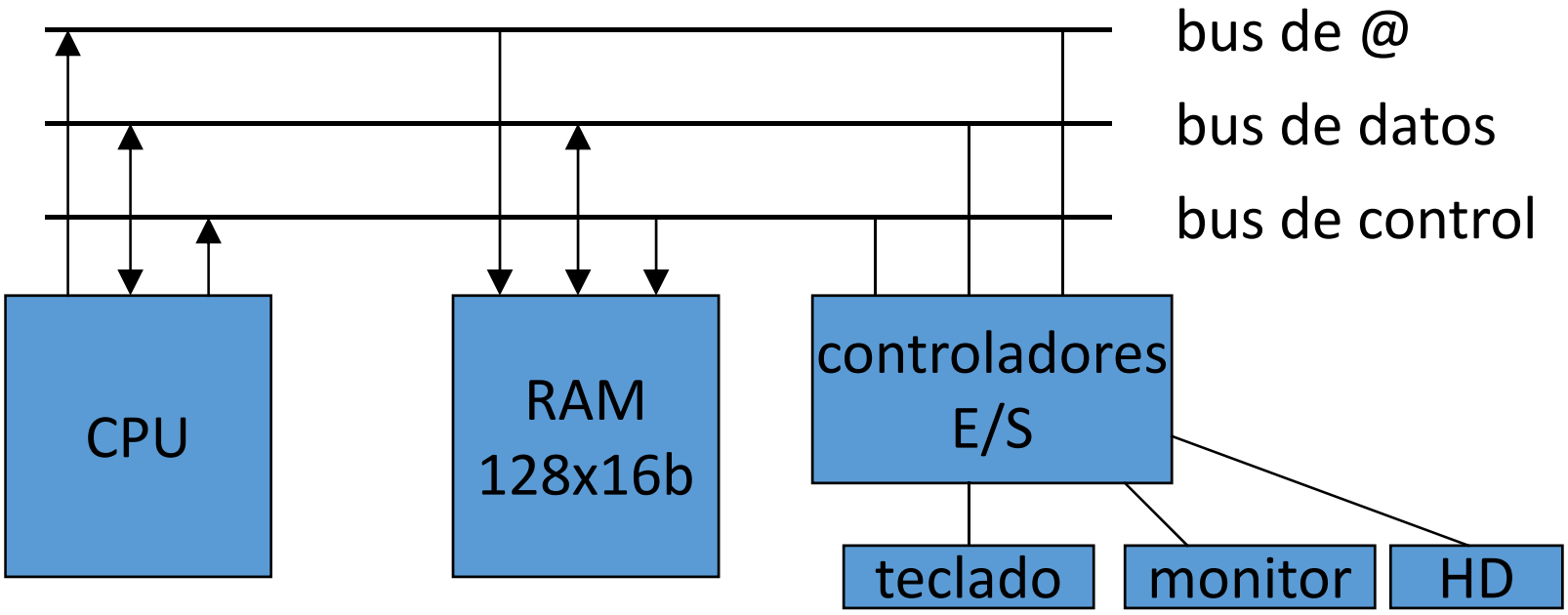
1º Ingeniería Informática, EINA, U. de Zaragoza

Luis M. Ramos, Víctor Viñals

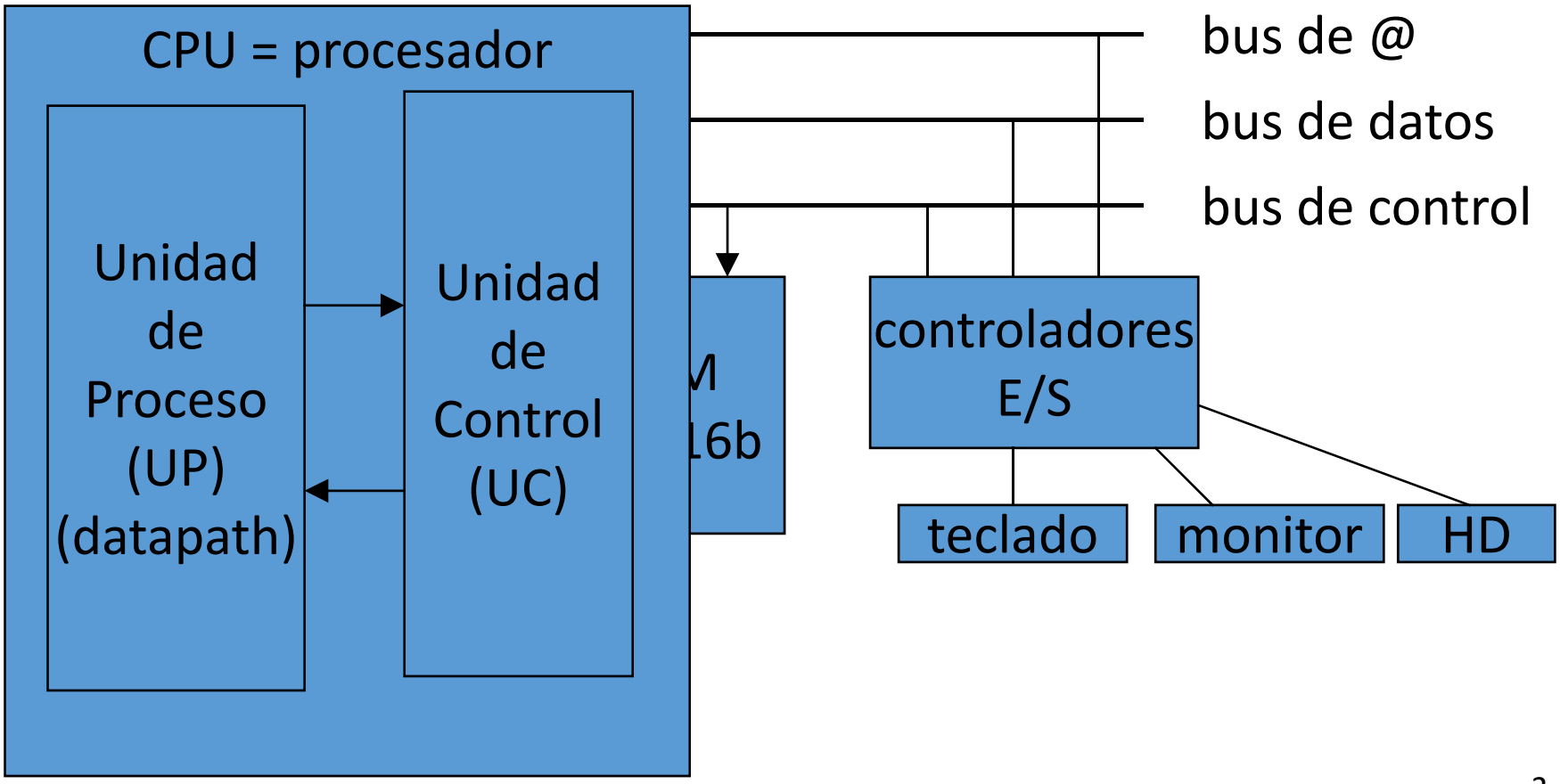
Organización de un computador sencillo



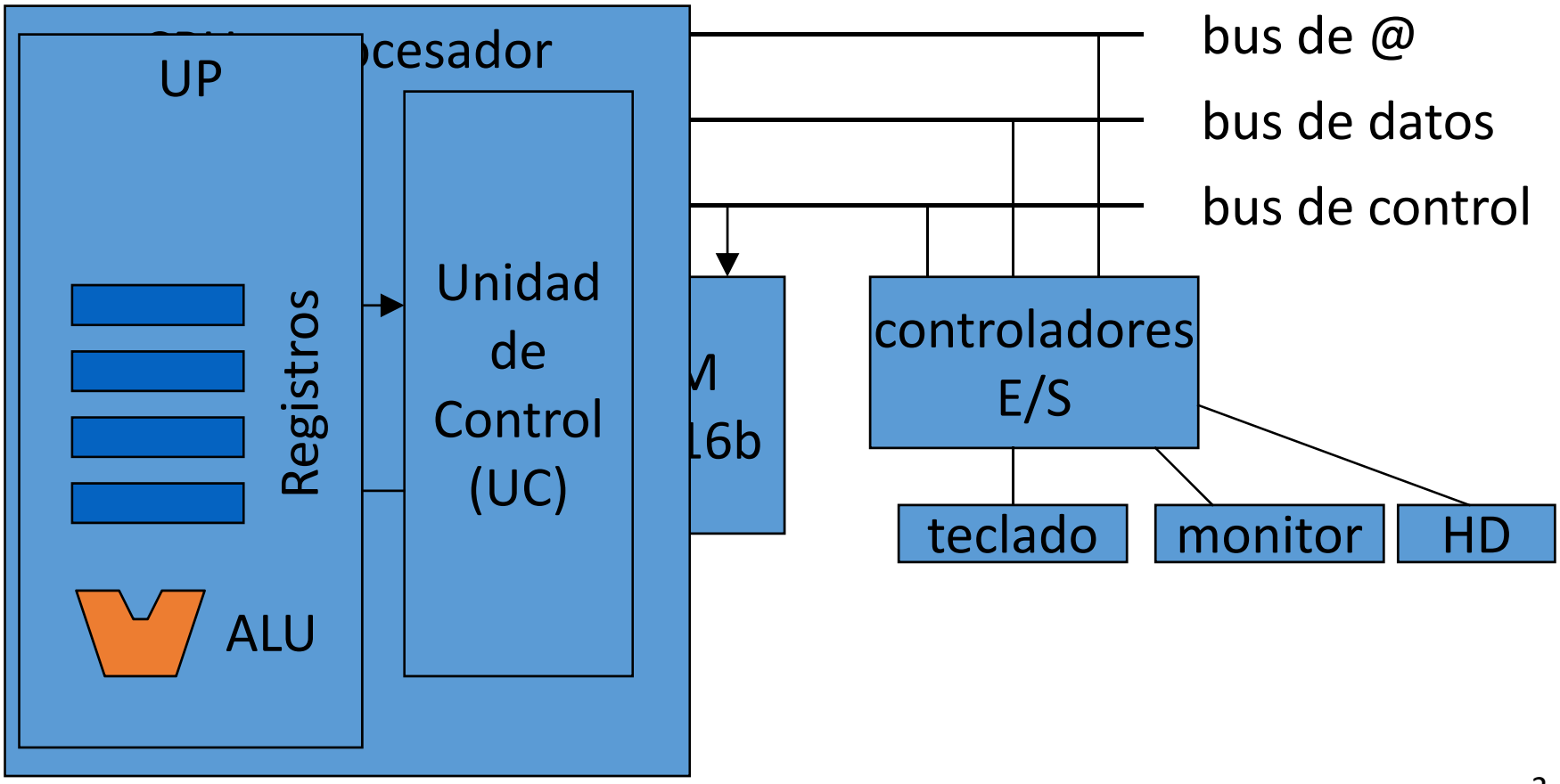
Organización de un computador sencillo



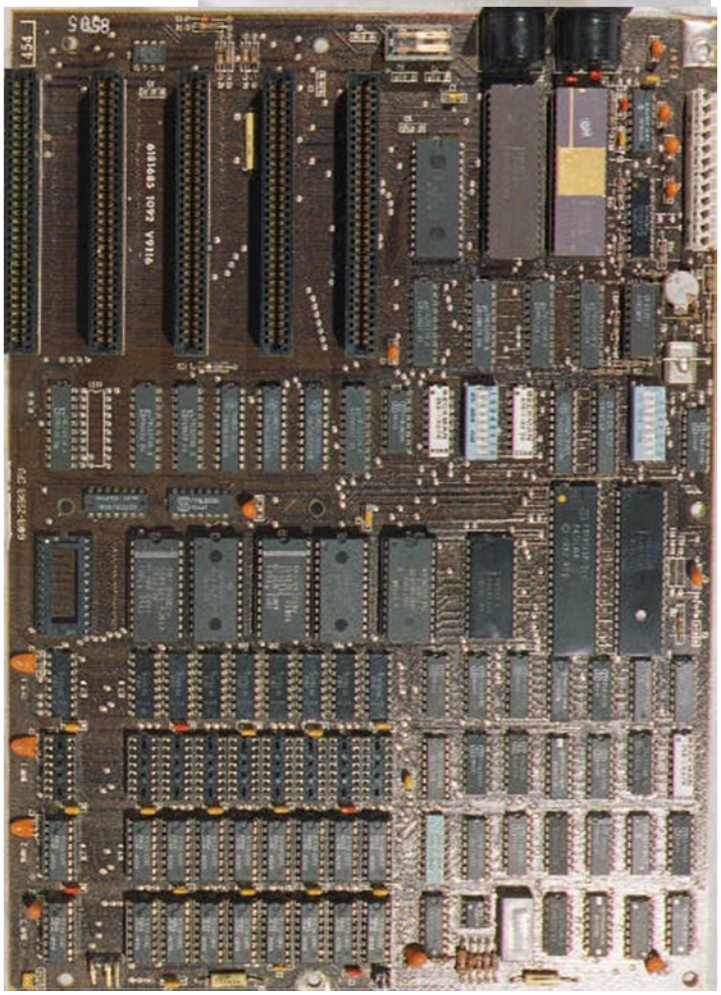
Organización de un computador sencillo

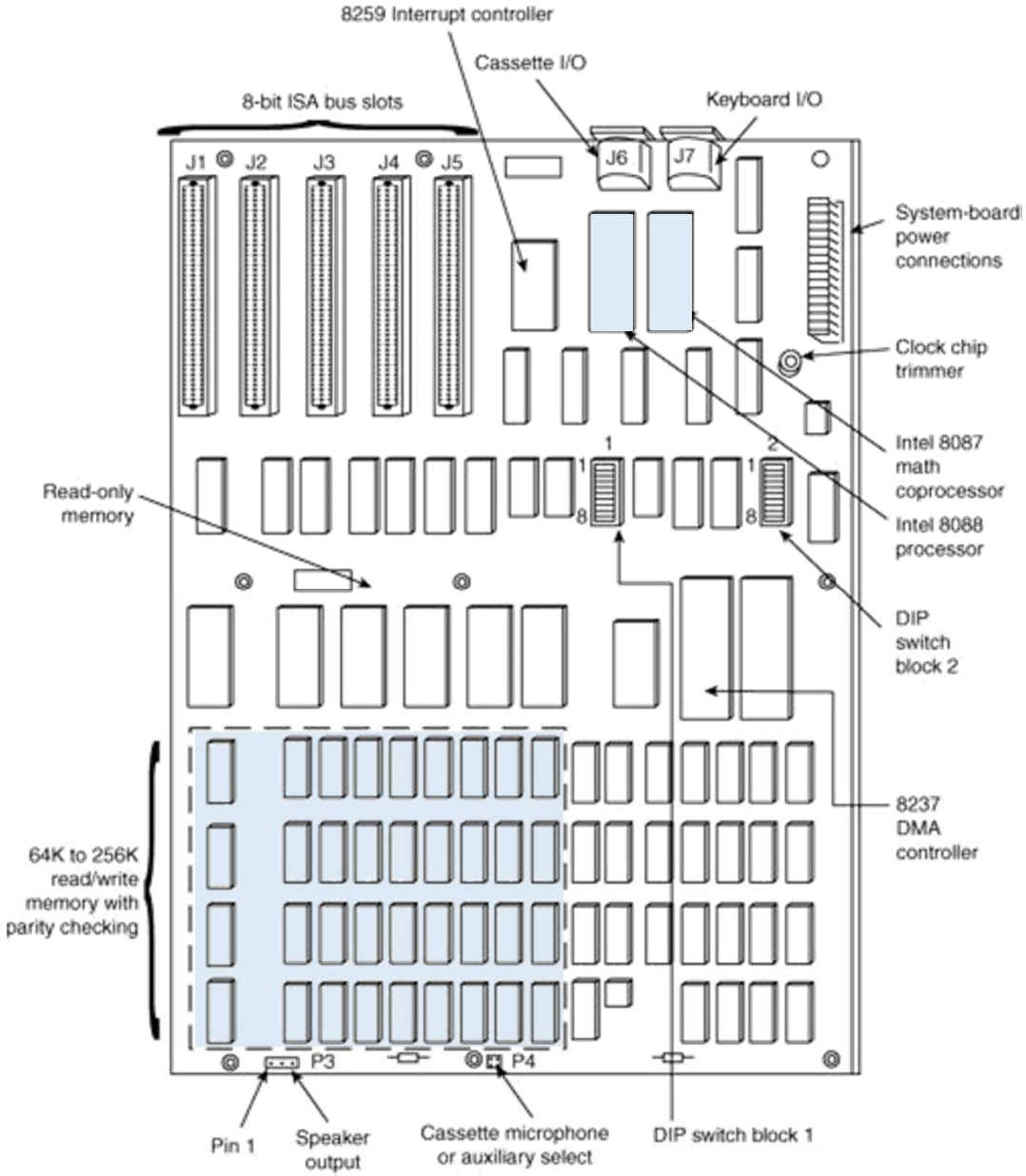


Organización de un computador sencillo

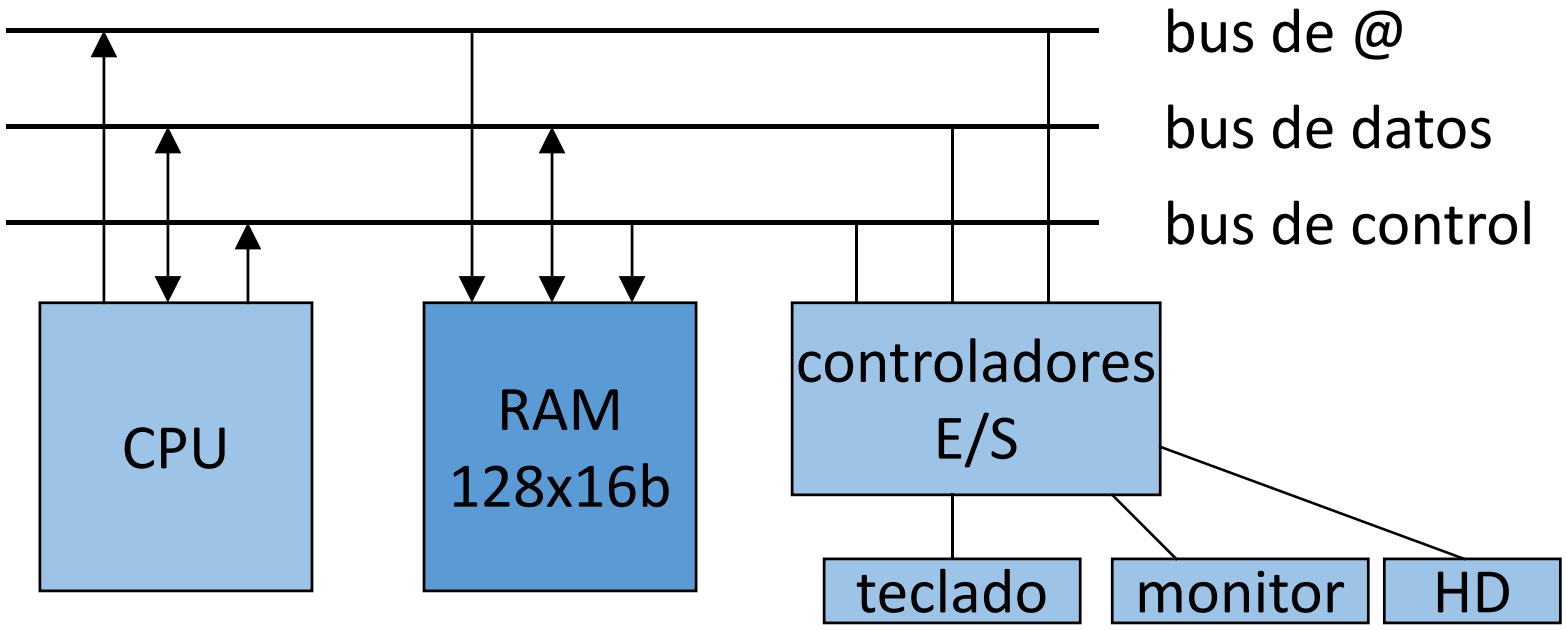


IBM PC XT, 1981

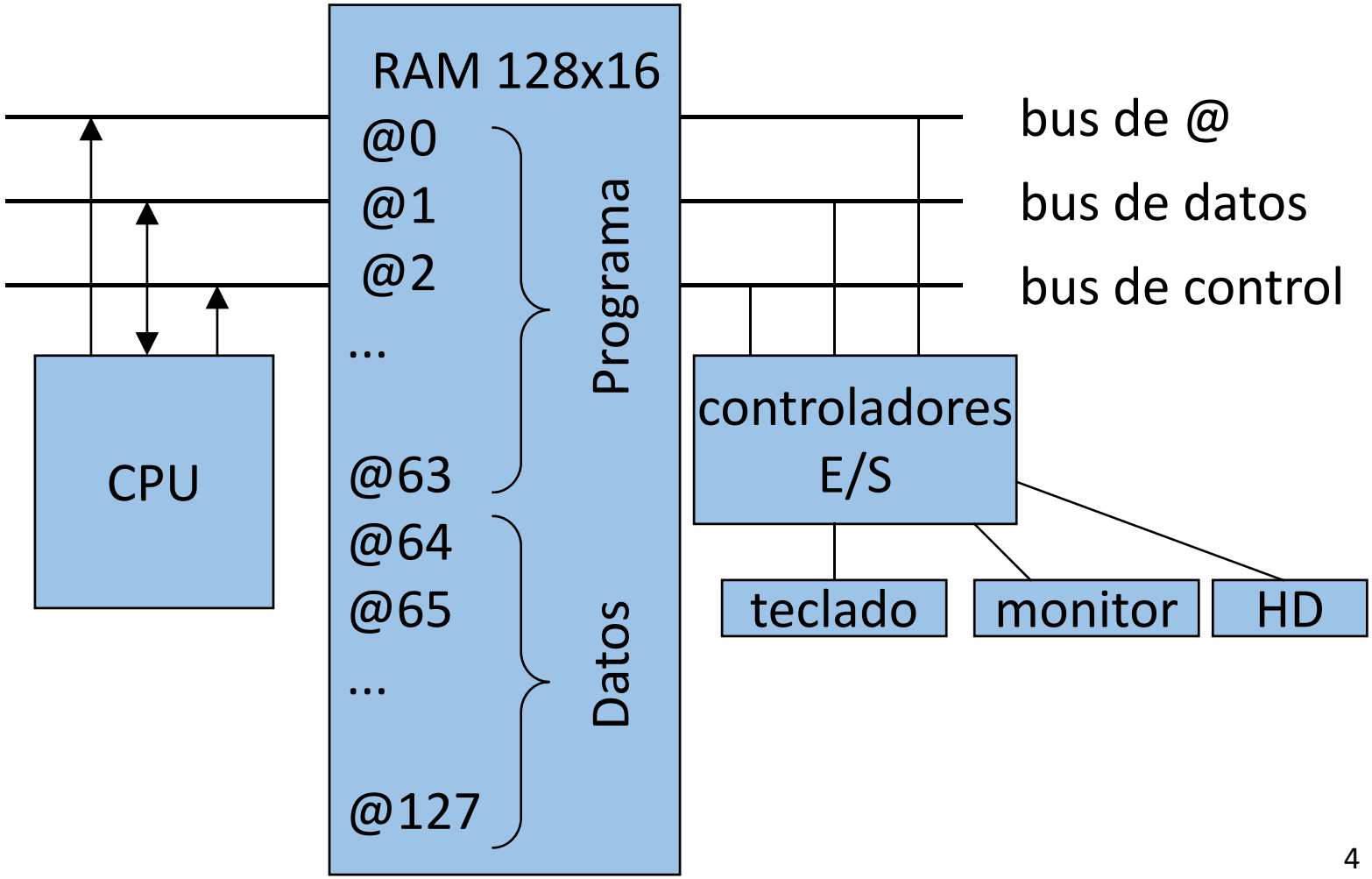




Organización de un computador sencillo



Organización de un computador sencillo



¿Qué hacen la UP y la UC?

¿Qué hacen la UP y la UC?

Unidad de Proceso

¿Qué hacen la UP y la UC?

Unidad de Proceso

- consigue siguiente instrucción a ejecutar (leer RAM)
  operación + localización de operandos → registro de instrucción

¿Qué hacen la UP y la UC?

Unidad de Proceso

- consigue siguiente instrucción a ejecutar (leer RAM)
  operación + localización de operandos → registro de instrucción
- decodifica la instrucción

¿Qué hacen la UP y la UC?

Unidad de Proceso

- consigue siguiente instrucción a ejecutar (leer RAM)
  operación + localización de operandos → registro de instrucción
- decodifica la instrucción
- consigue los operandos fuente (leer RAM dos veces)
  dato → registro de datos

¿Qué hacen la UP y la UC?

Unidad de Proceso

- consigue siguiente instrucción a ejecutar (leer RAM)
  operación + localización de operandos → registro de instrucción
- decodifica la instrucción
- consigue los operandos fuente (leer RAM dos veces)
  dato → registro de datos
- opera y guarda resultado (escribir RAM)
  resultado → RAM

¿Qué hacen la UP y la UC?

Unidad de Proceso

- consigue siguiente instrucción a ejecutar (leer RAM)
  operación + localización de operandos → registro de instrucción
- decodifica la instrucción
- consigue los operandos fuente (leer RAM dos veces)
  dato → registro de datos
- opera y guarda resultado (escribir RAM)
  resultado → RAM

Unidad de Control

¿Qué hacen la UP y la UC?

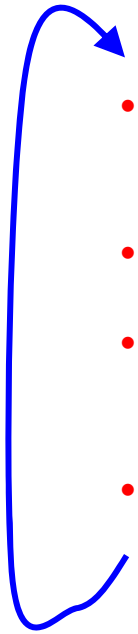
Unidad de Proceso

- consigue siguiente instrucción a ejecutar (leer RAM)
  operación + localización de operandos → registro de instrucción
- decodifica la instrucción
- consigue los operandos fuente (leer RAM dos veces)
  dato → registro de datos
- opera y guarda resultado (escribir RAM)
  resultado → RAM

Unidad de Control

- da órdenes a la UP,
  … porque la UP no sabe qué hacer ni cuándo hacerlo

¿Qué hacen la UP y la UC?



Unidad de Proceso

- consigue siguiente instrucción a ejecutar (leer RAM)
  operación + localización de operandos → registro de instrucción
- decodifica la instrucción
- consigue los operandos fuente (leer RAM dos veces)
  dato → registro de datos
- opera y guarda resultado (escribir RAM)
  resultado → RAM

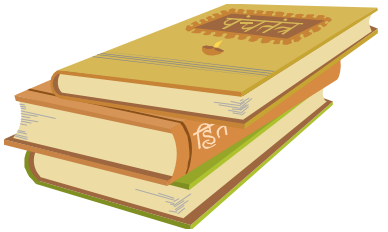
Unidad de Control

- da órdenes a la UP,
  … porque la UP no sabe qué hacer ni cuándo hacerlo

Arquitectura de la Máquina Sencilla

Almacenes

- Memoria
  - ✓ 128 palabras de 16 bits, numeradas desde la 0 hasta la 127
  - ✓ contiene instrucciones y datos
  - ✓ los datos son números naturales o enteros Ca2



- FZ (*flag zero*): 1 bit, registra igualdad cero
- PC (*program counter*): 7 bits, dirección de la instrucción a ejecutar

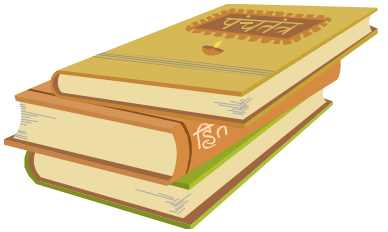
4 instrucciones de tamaño fijo: ADD, MOV, CMP, BEQ

- ADD, MOV y CMP usan el *modo de direccionamiento de datos* directo ó absoluto: la dirección del dato *está* en la propia instrucción

Arquitectura de la Máquina Sencilla

Almacenes

- Memoria
  - ✓ 128 palabras de 16 bits, numeradas desde la 0 hasta la 127
  - ✓ contiene instrucciones y datos
  - ✓ los datos son números naturales o enteros Ca2



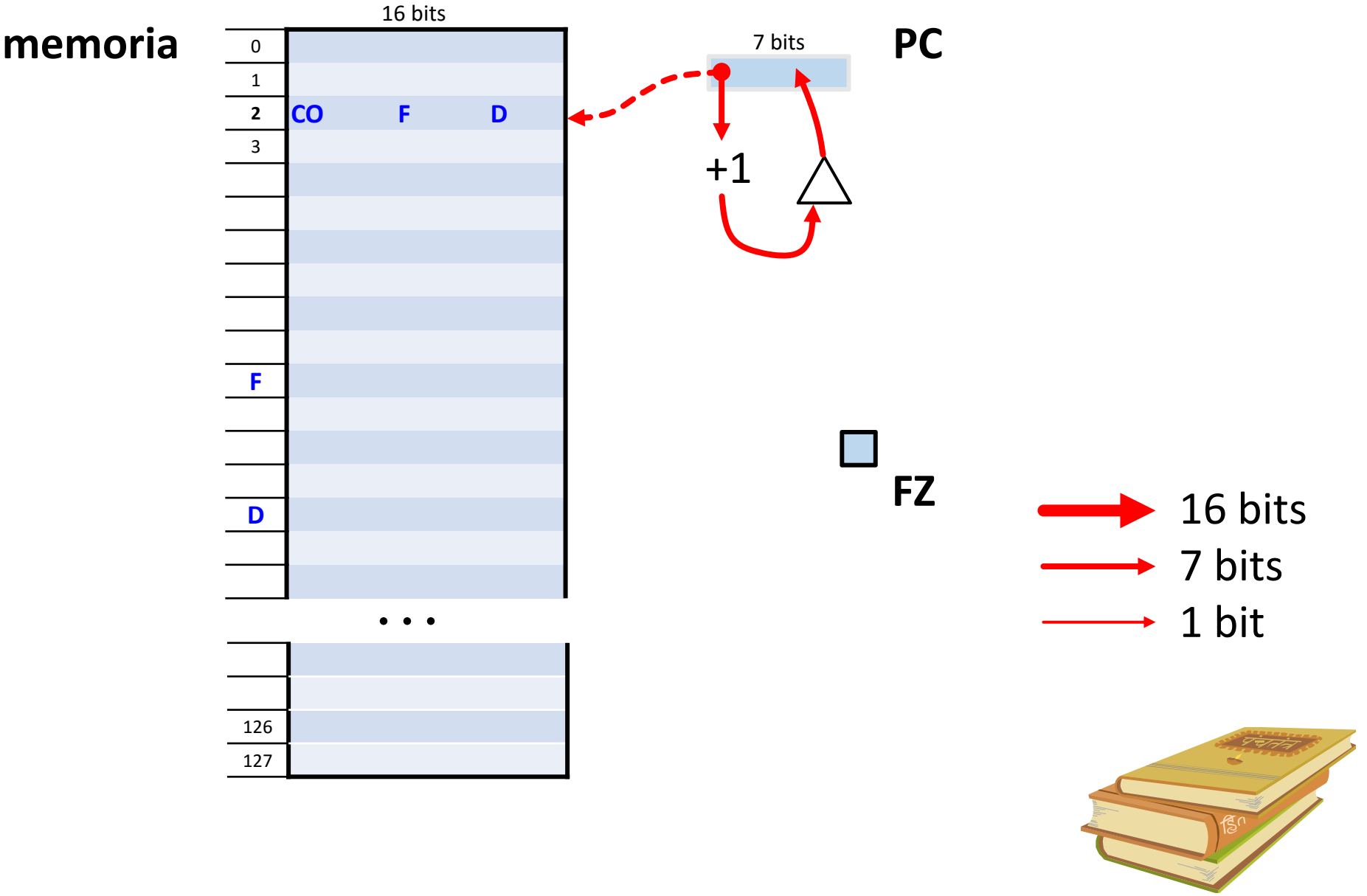
- FZ (*flag zero*): 1 bit, registra igualdad cero
- PC (*program counter*): 7 bits, dirección de la instrucción a ejecutar

4 instrucciones de tamaño fijo: ADD, MOV, CMP, BEQ

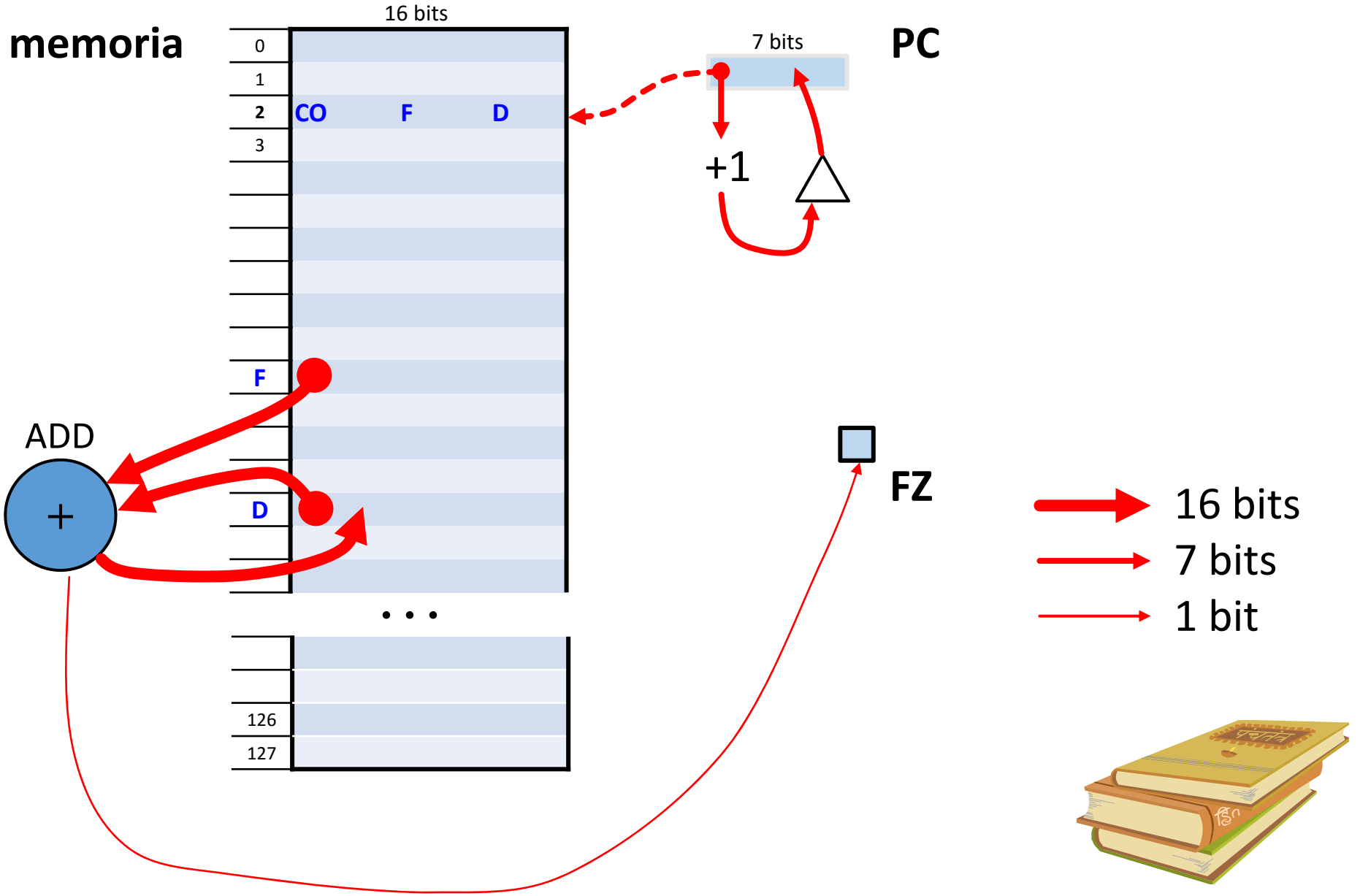
- ADD, MOV y CMP usan el *modo de direccionamiento de datos* directo ó absoluto: la dirección del dato *está* en la propia instrucción

Ensamblador	Descripción	Tipo	Codificación (16b)
ADD F,D	@D ← (D)+(F) FZ ← 1 si (D)+(F)=0	aritmética	`00 fffffff ddddddd`
CMP F,D	FZ ← 1 si (F)⊕(D)=0	aritmético-lógica	`01 fffffff ddddddd`
MOV F,D	@D ← (F) FZ ← 1 si (F)=0	movimiento	`10 fffffff ddddddd`
BEQ dst	si FZ=1 salta a dst sino ejecuta siguiente	salto condicional	`11 _______ ddddddd`

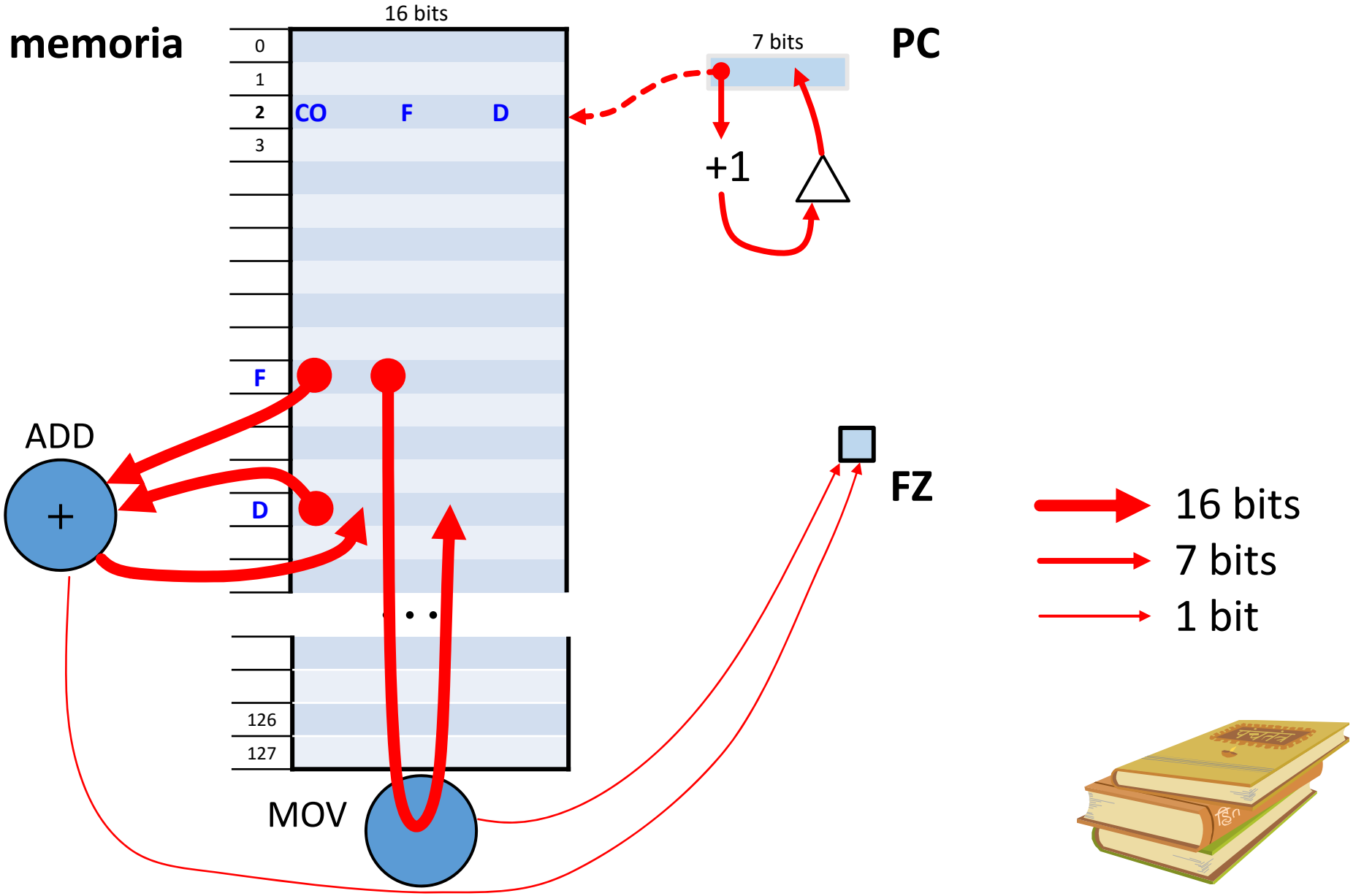
Arquitectura de la M.S. representación gráfica



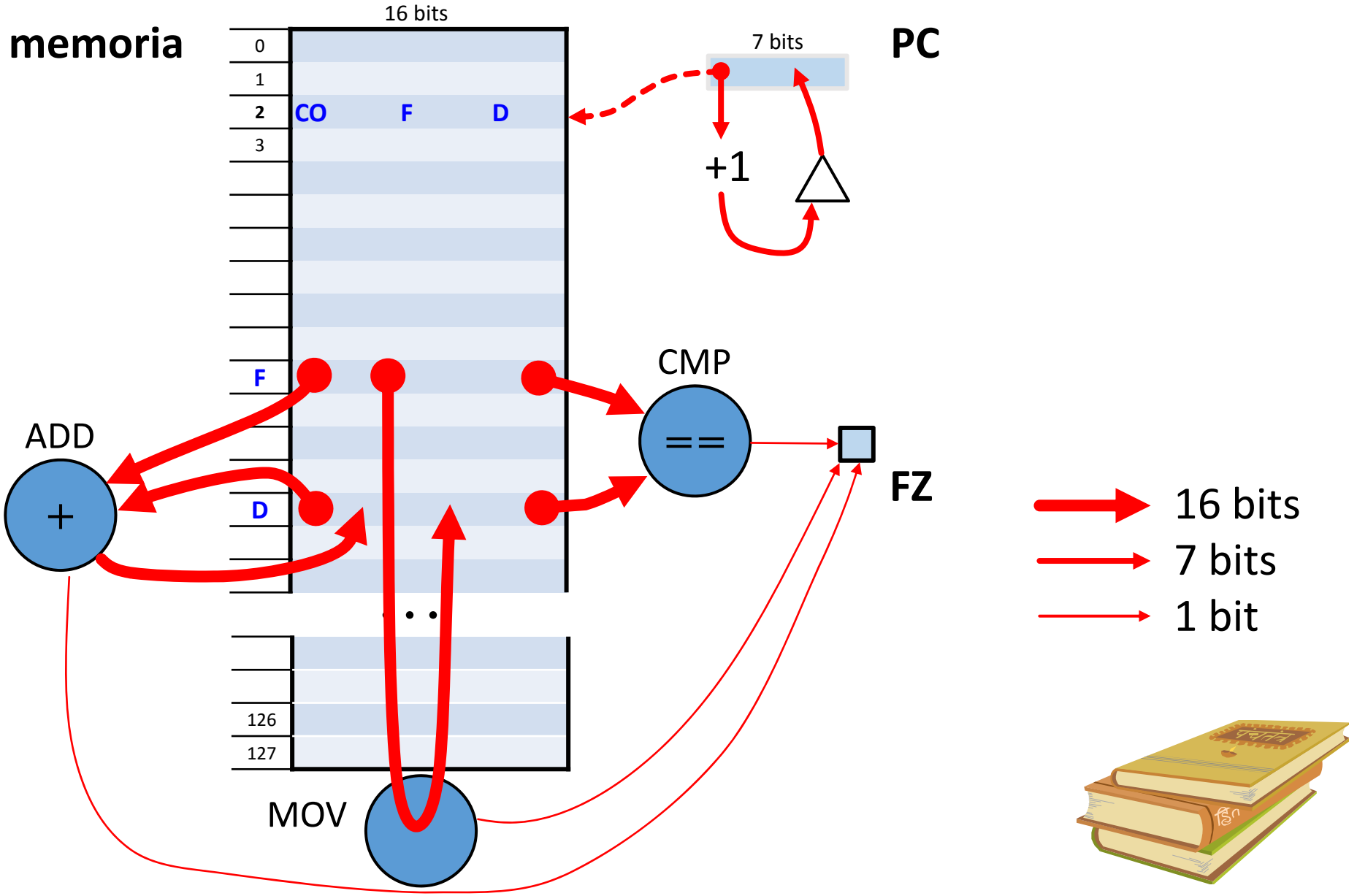
Arquitectura de la M.S. representación gráfica



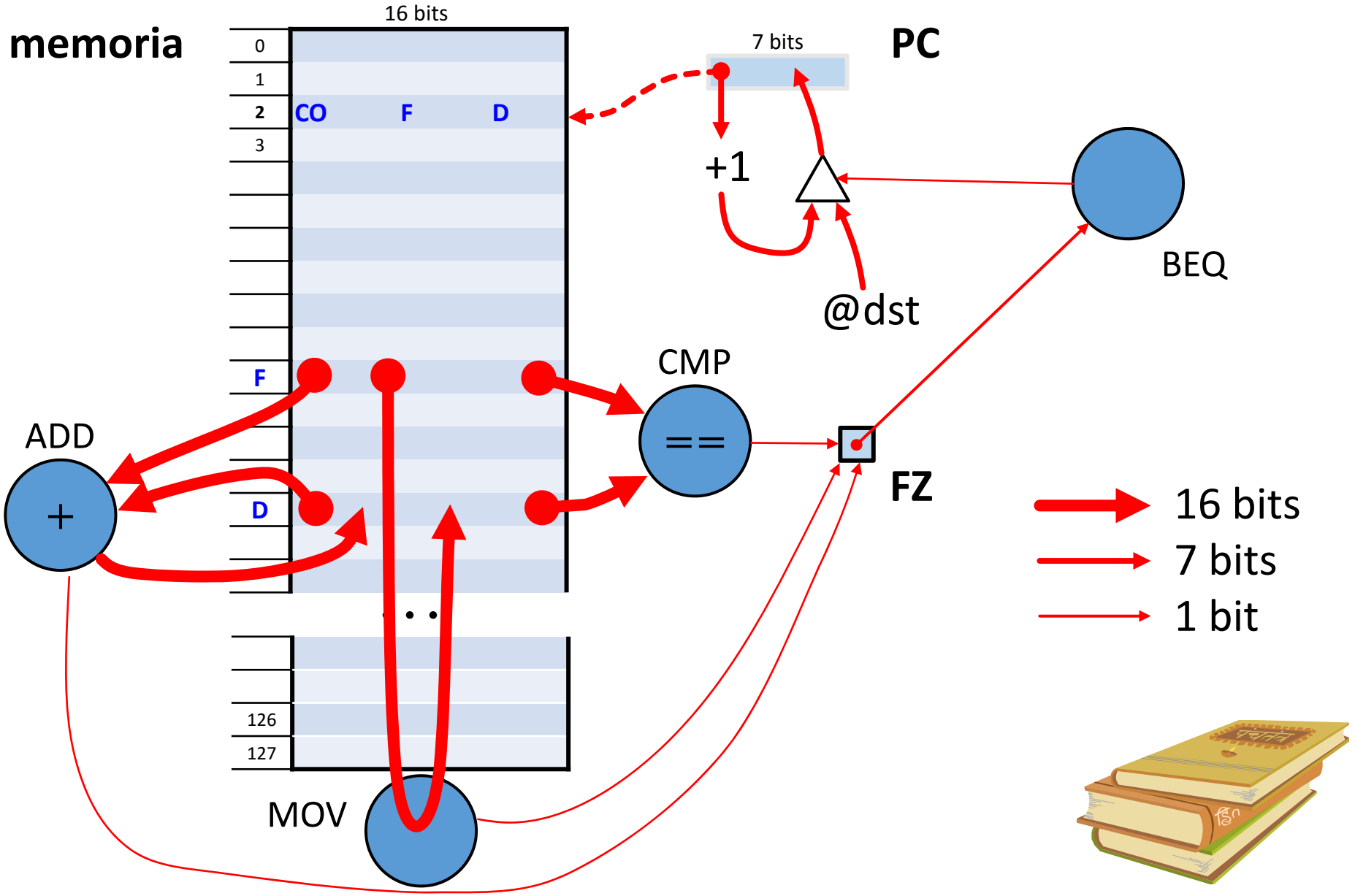
Arquitectura de la M.S. representación gráfica



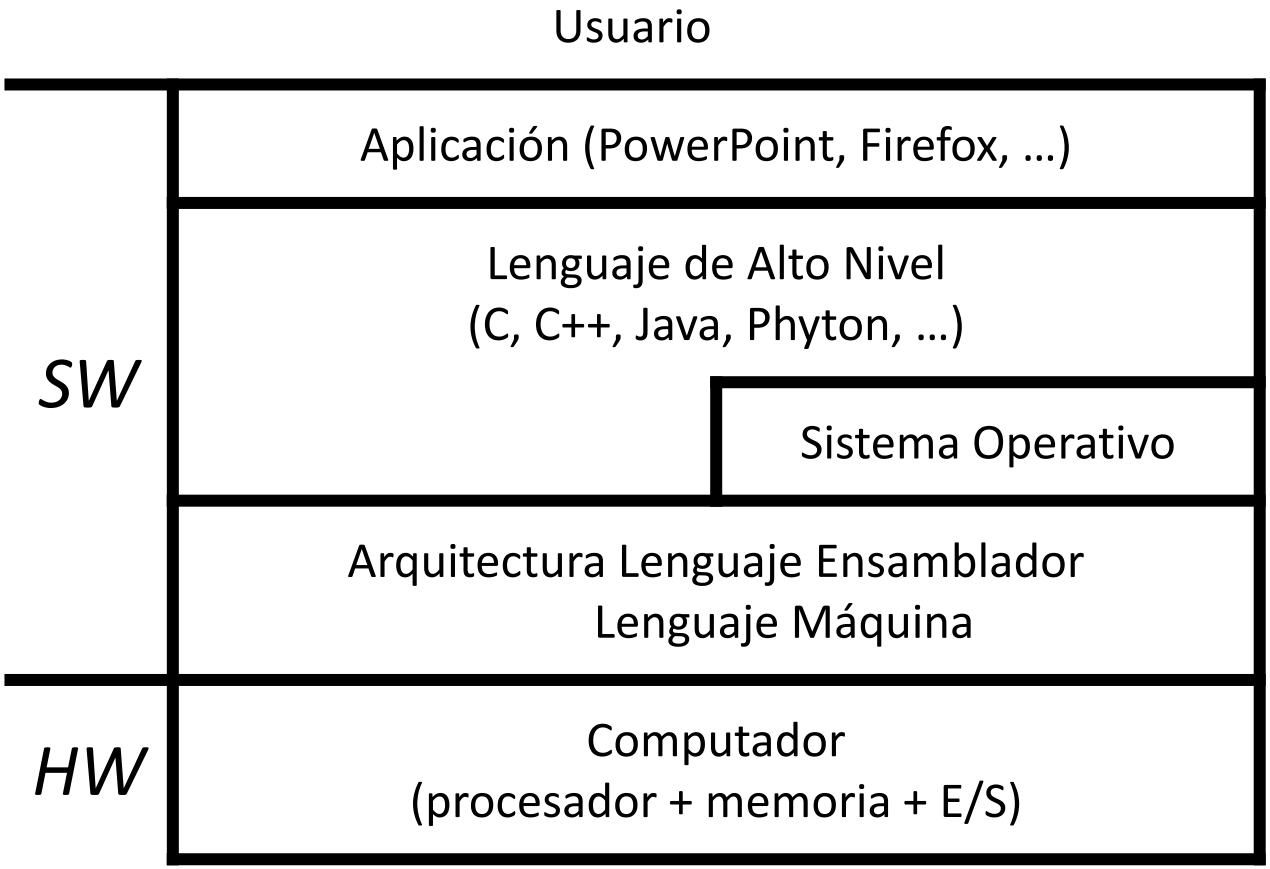
Arquitectura de la M.S. representación gráfica



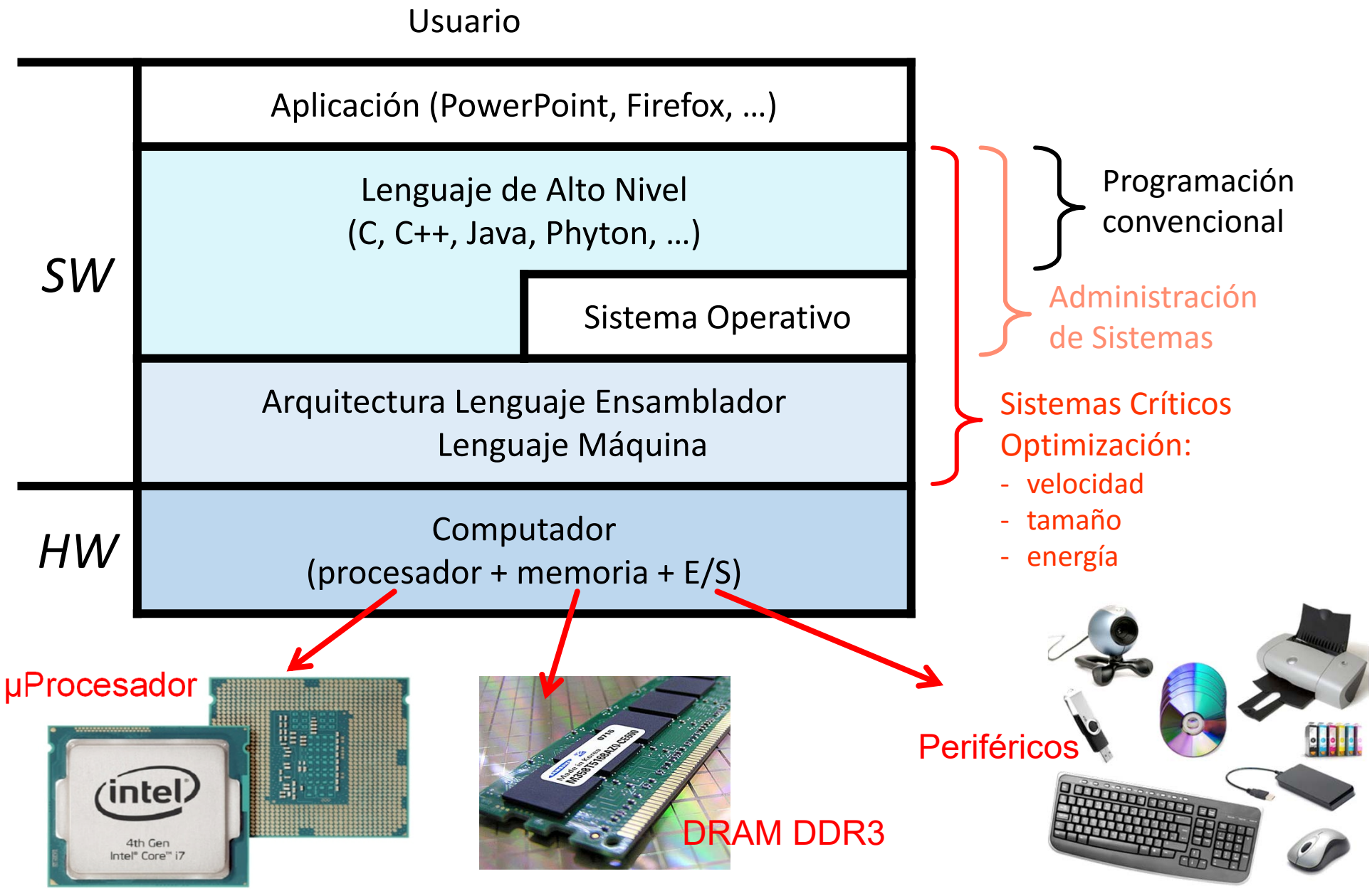
Arquitectura de la M.S. representación gráfica



Niveles (o capas) del computador: HW y SW



Niveles (o capas) del computador: HW



Ejemplo Máquina Sencilla

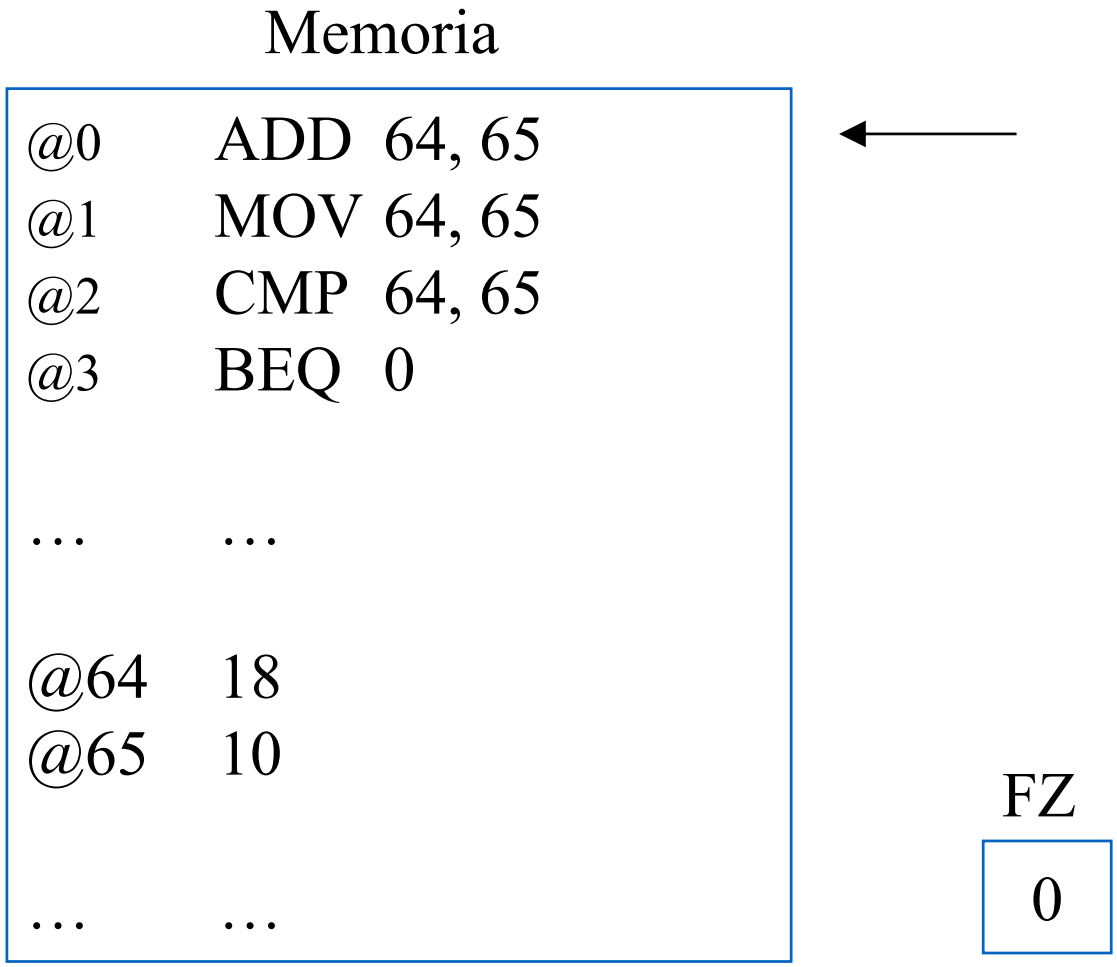
Memoria

@0	ADD 64, 65
@1	MOV 64, 65
@2	CMP 64, 65
@3	BEQ 0
…	…
@64	18
@65	10
…	…

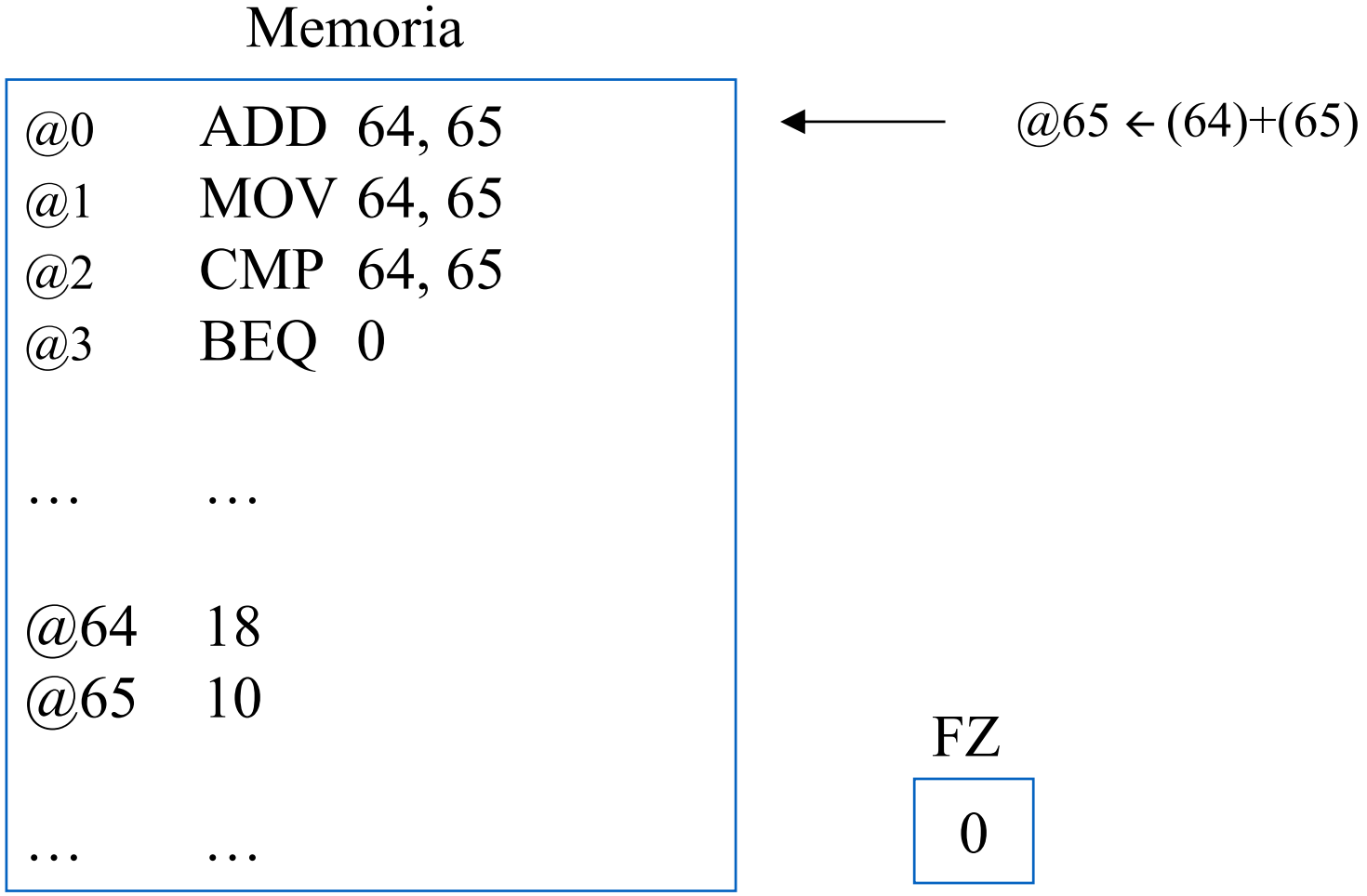
FZ

0

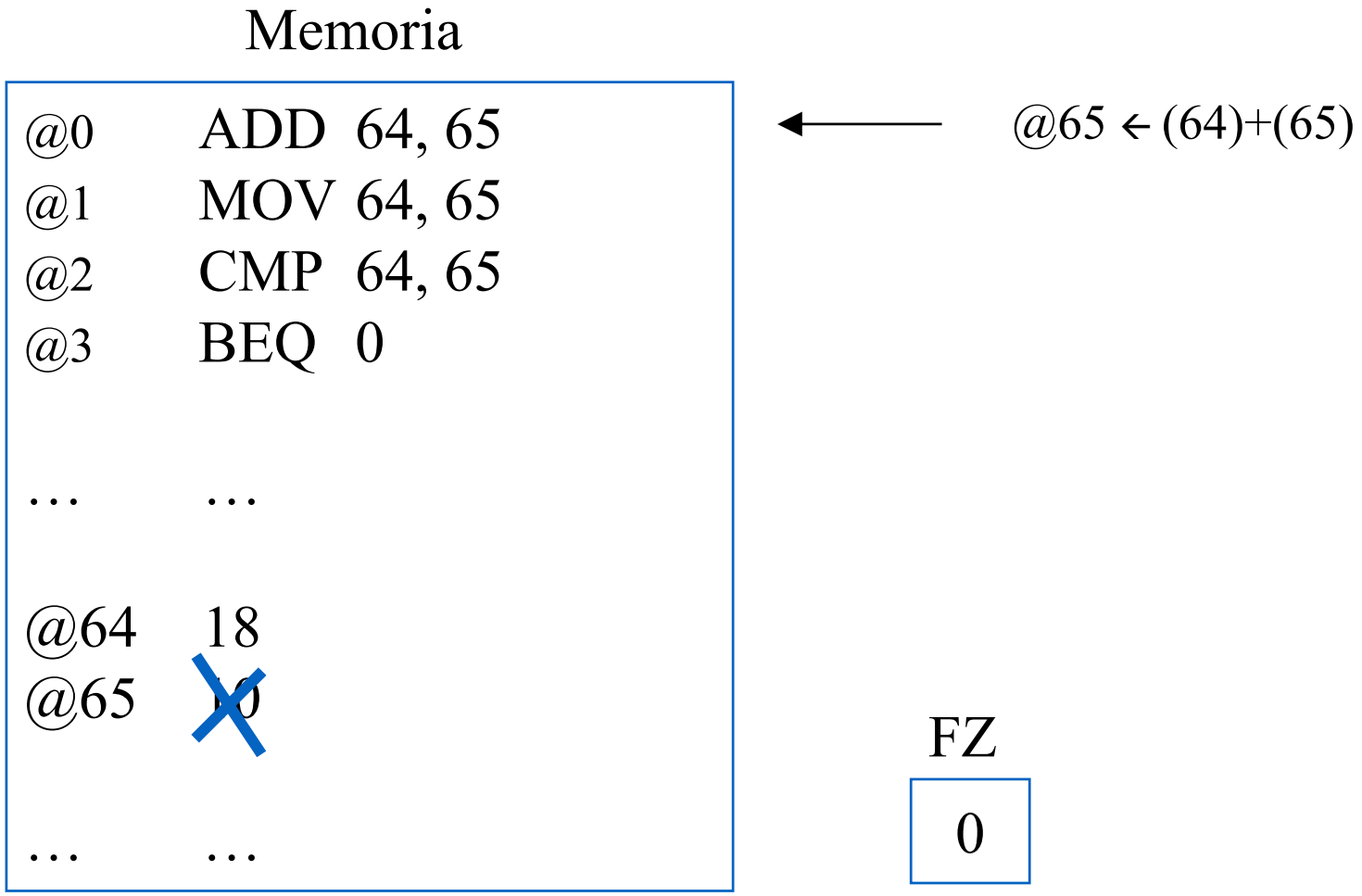
Ejemplo Máquina Sencilla



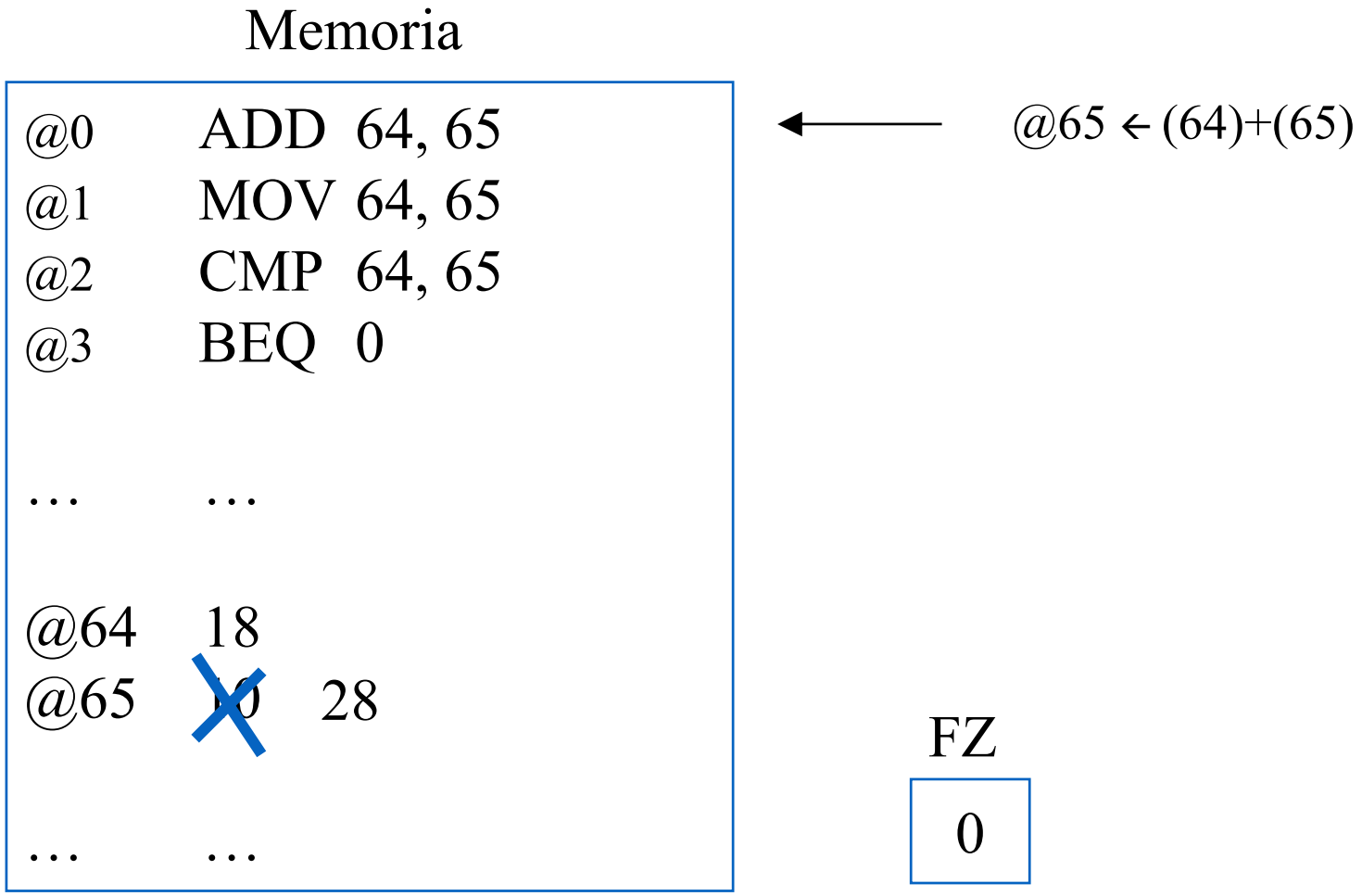
Ejemplo Máquina Sencilla



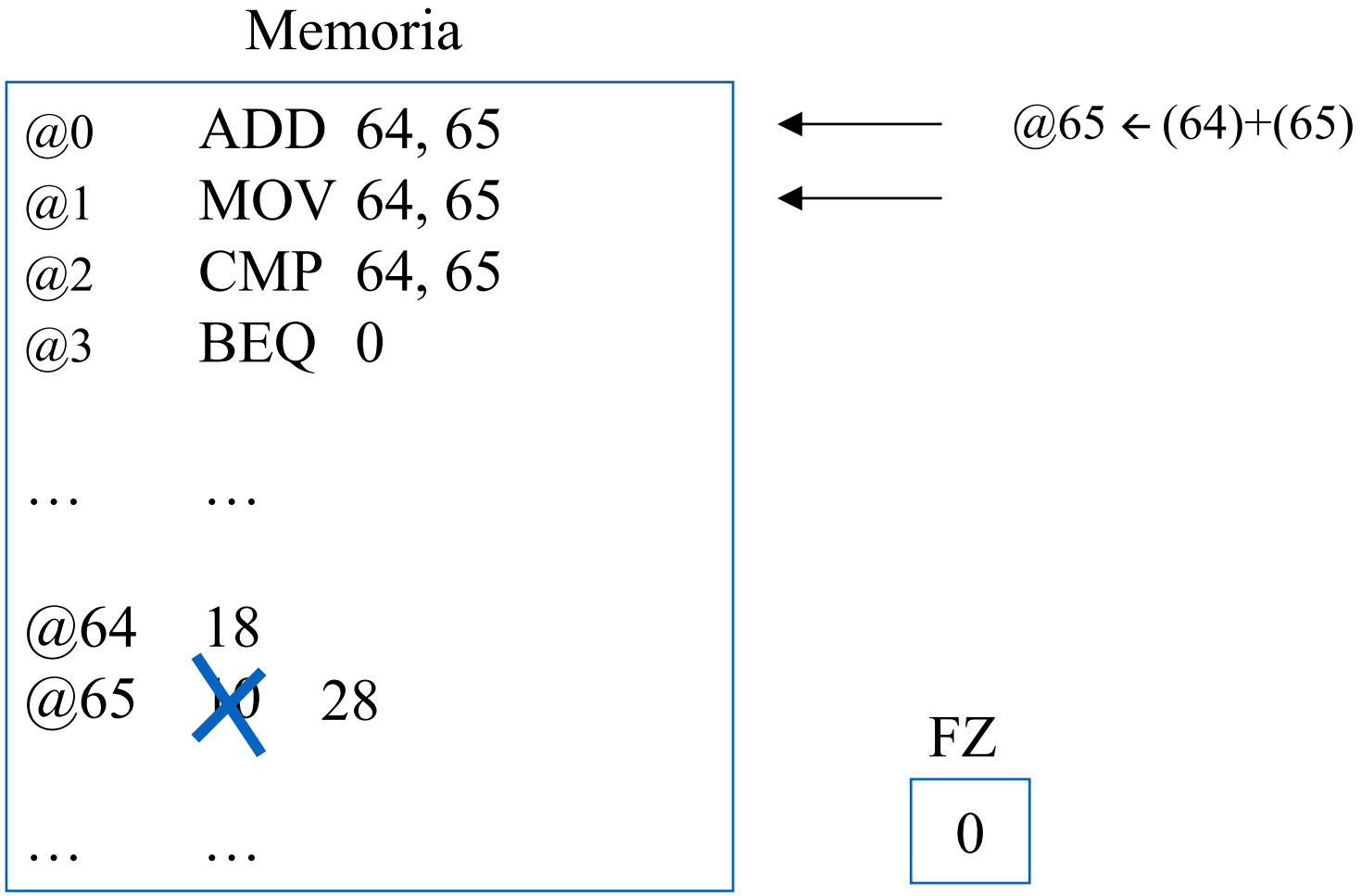
Ejemplo Máquina Sencilla



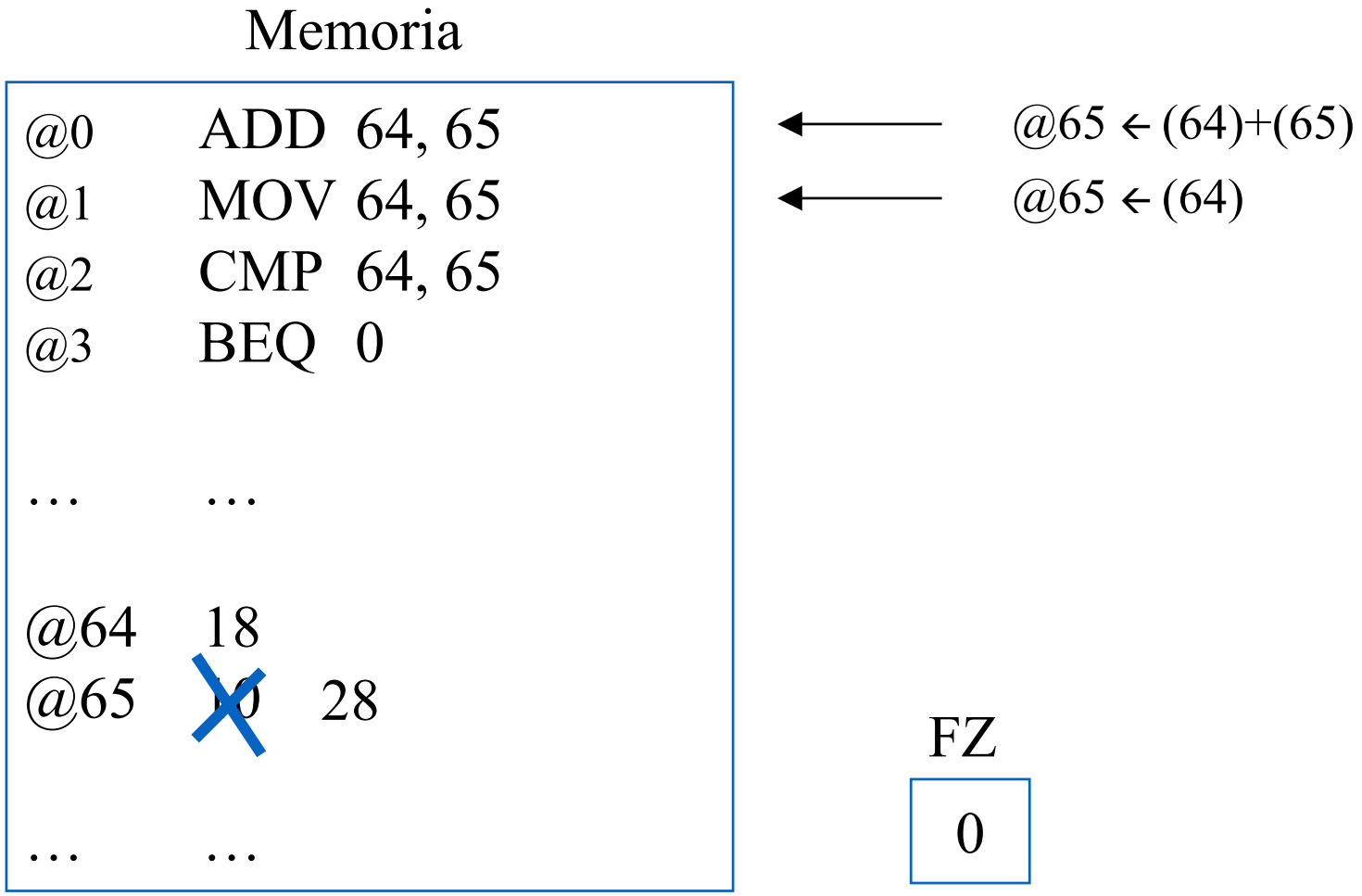
Ejemplo Máquina Sencilla



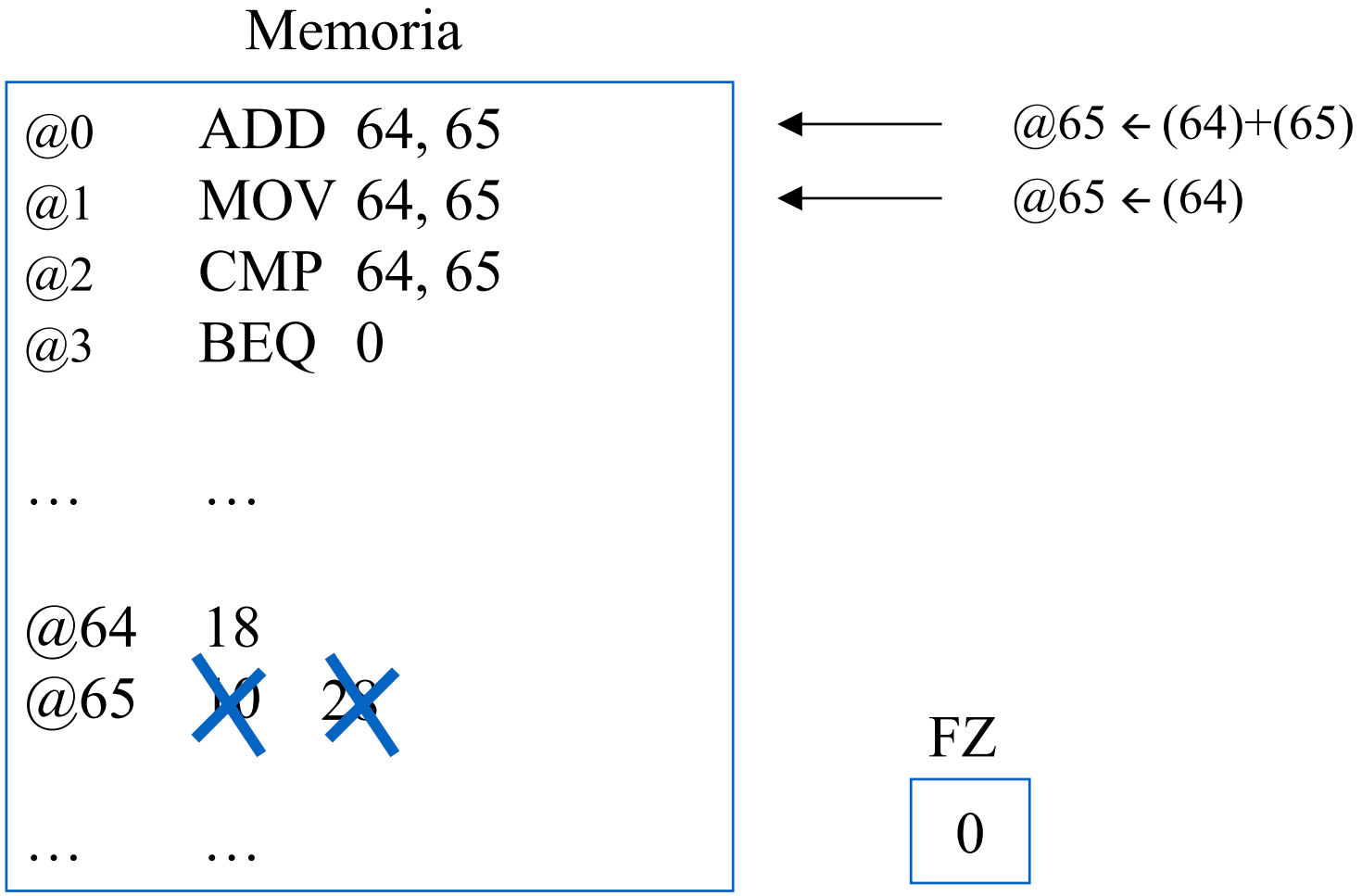
Ejemplo Máquina Sencilla



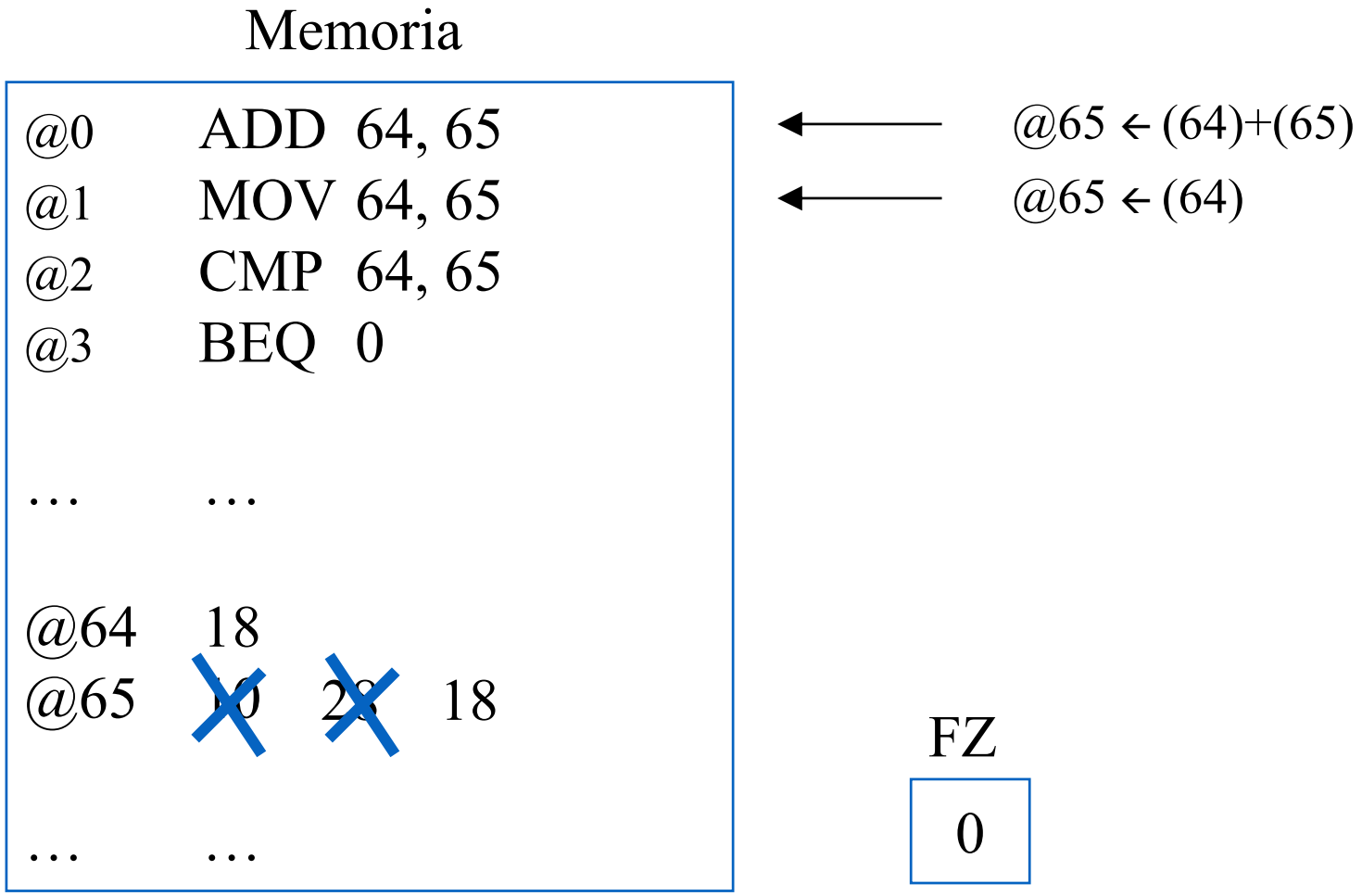
Ejemplo Máquina Sencilla



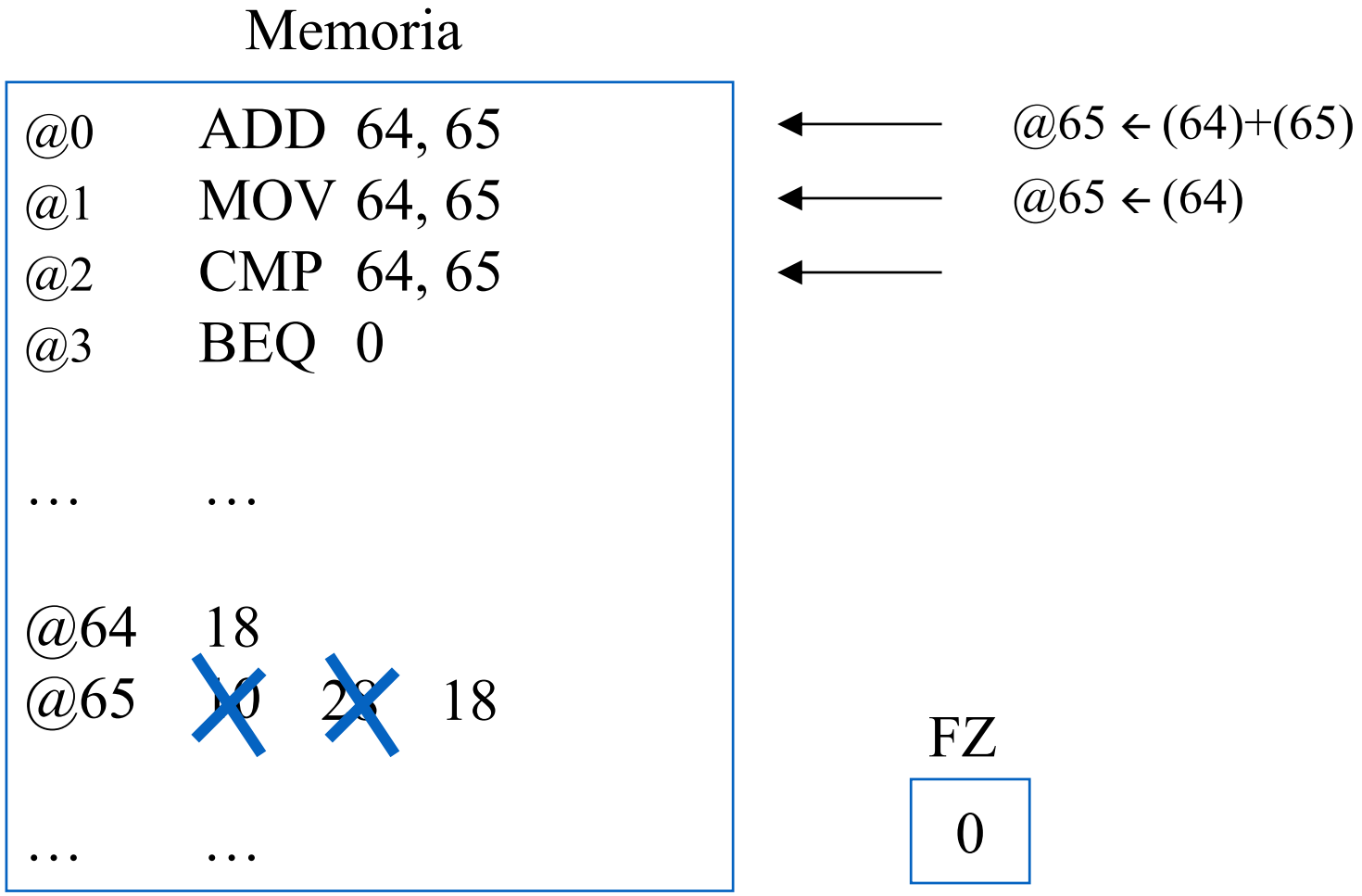
Ejemplo Máquina Sencilla



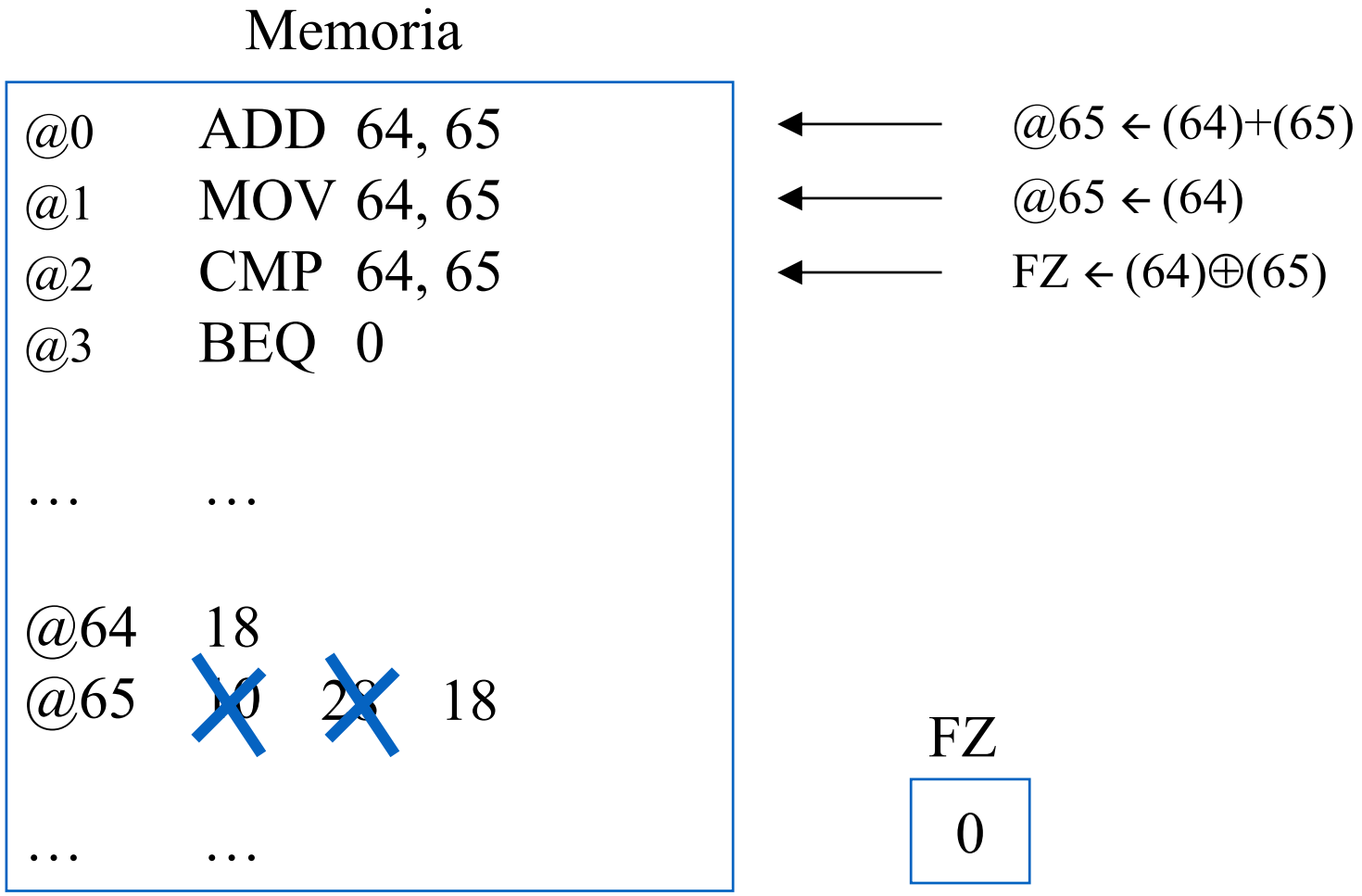
Ejemplo Máquina Sencilla



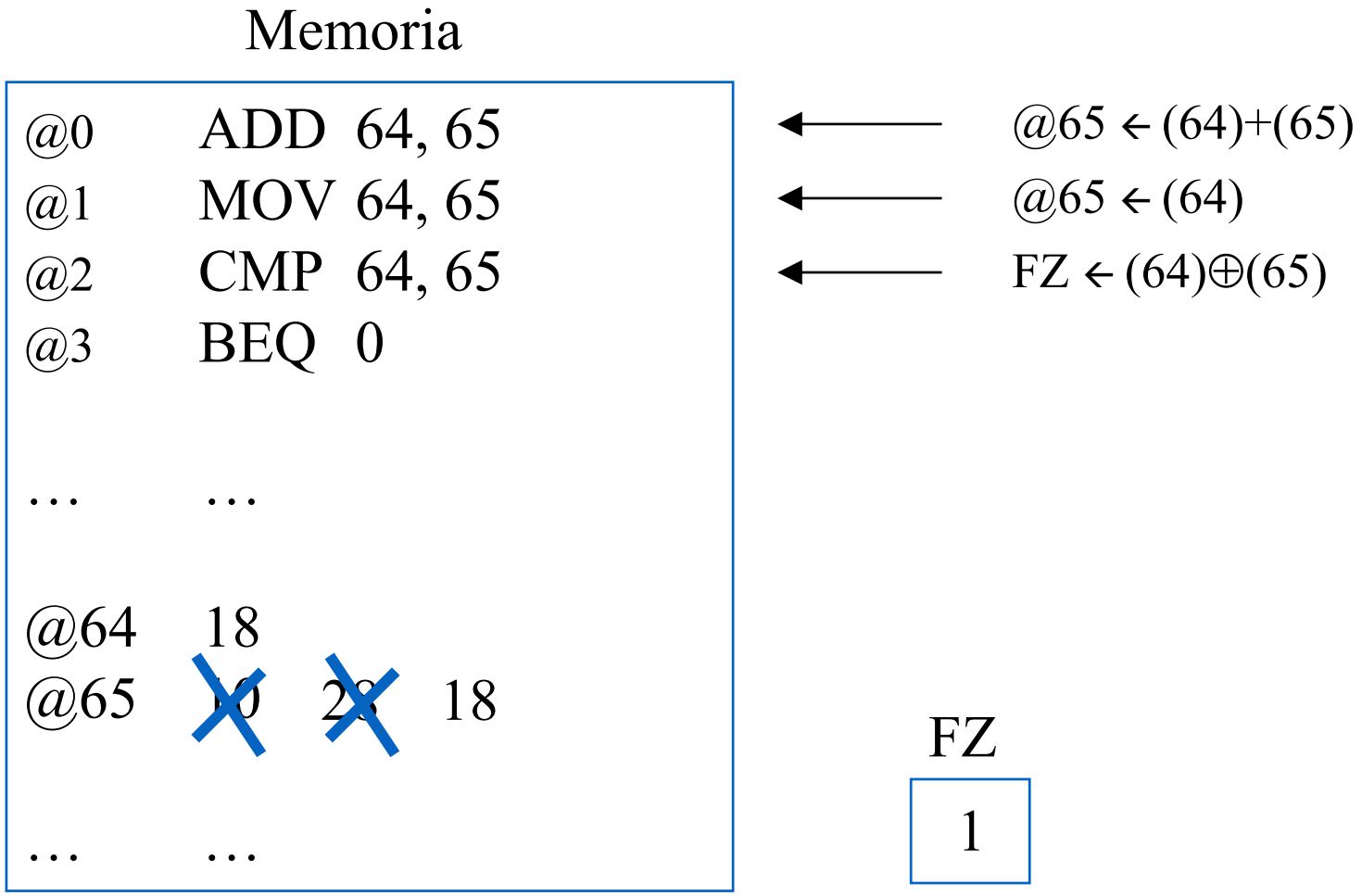
Ejemplo Máquina Sencilla



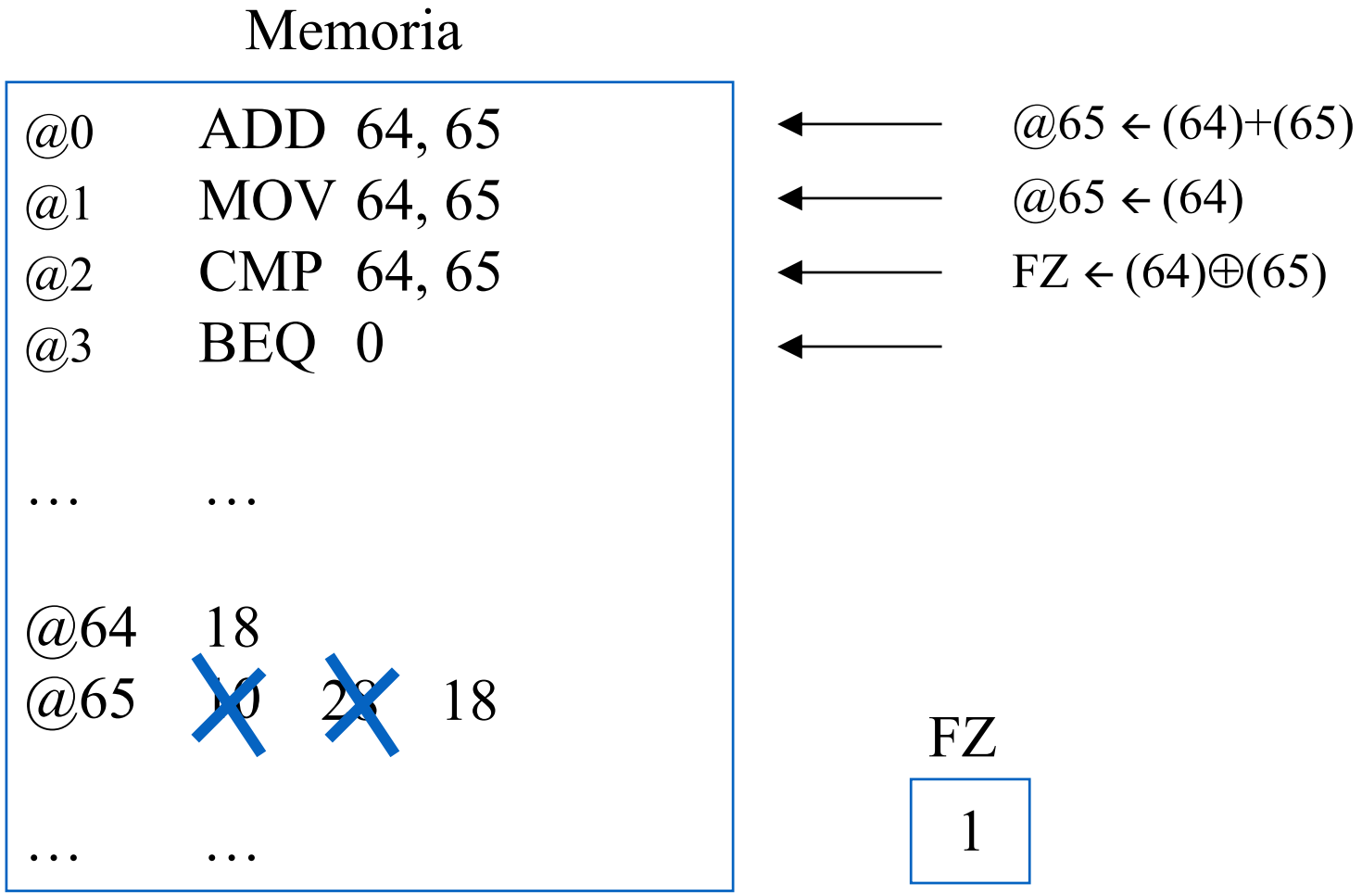
Ejemplo Máquina Sencilla



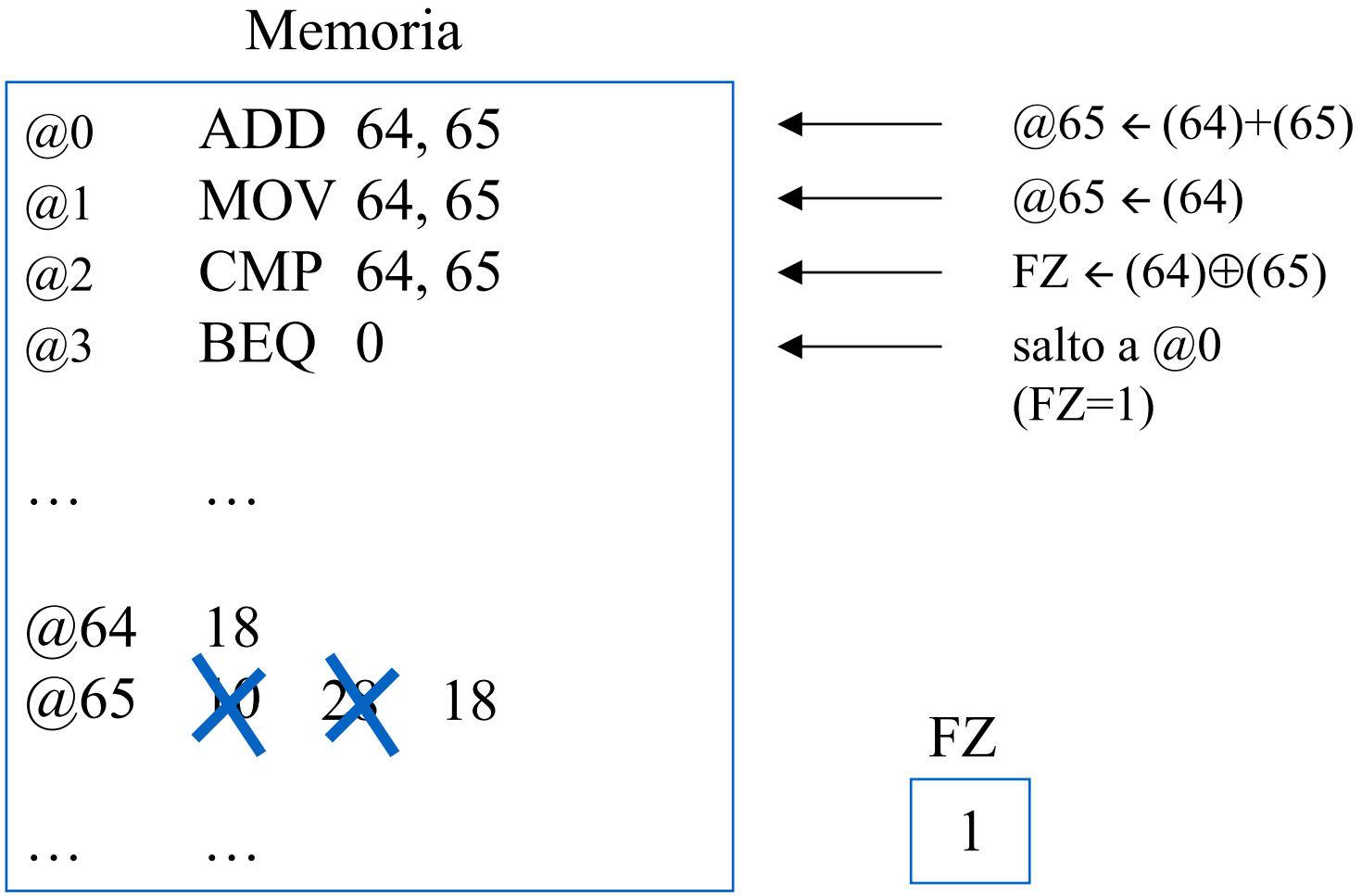
Ejemplo Máquina Sencilla



Ejemplo Máquina Sencilla



Ejemplo Máquina Sencilla



¿4 son pocas instrucciones?

```
uint16_t   a, b;
…
c= a*b
```

¿4 son pocas instrucciones?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

¿4 son pocas instrucciones?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

```
        MOV CERO, c
        MOV CERO, i
while:  CMP i, b
        BEQ fin
        ADD a, c
        ADD UNO, i
        CMP i, i
        BEQ while
fin:
```

¿4 son pocas instrucciones?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

```
            MOV CERO, c
            MOV CERO, i
while:      CMP i, b
            BEQ fin
            ADD a, c
            ADD UNO, i
            CMP i, i
            BEQ while
fin:
```

¿4 son pocas instrucciones?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

```
        MOV CERO, c
        MOV CERO, i
while:  CMP i, b      } salto
        BEQ fin       } condicional
        ADD a, c
        ADD UNO, i
        CMP i, i
        BEQ while
fin:
```

¿4 son pocas instrucciones?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

```
        MOV CERO, c
        MOV CERO, i
while:  CMP i, b      } salto
        BEQ fin       } condicional
        ADD a, c
        ADD UNO, i
        CMP i, i      }
        BEQ while     }
fin:
```

¿4 son pocas instrucciones?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

```
        MOV CERO, c
        MOV CERO, i
while:  CMP i, b      } salto
        BEQ fin       } condicional
        ADD a, c
        ADD UNO, i
        CMP i, i      } salto
        BEQ while     } incondicional
fin:
```

¿4 son pocas instrucciones?

¿Qué son a, b, c, i? ¿CERO y UNO? ¿fin y while?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

```
        MOV CERO, c
        MOV CERO, i
while:  CMP i, b      } salto
        BEQ fin       } condicional
        ADD a, c
        ADD UNO, i
        CMP i, i      } salto
        BEQ while     } incondicional
fin:
```

¿4 son pocas instrucciones?

¿Qué son a, b, c, i? ¿CERO y UNO? ¿fin y while?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

```
        MOV CERO, c
        MOV CERO, i
while:  CMP i, b      } salto
        BEQ fin       } condicional
        ADD a, c
        ADD UNO, i
        CMP i, i      } salto
        BEQ while     } incondicional
fin:
```

Simbolos
a → @100
b → @101
c → @102
i → @103

¿4 son pocas instrucciones?

¿Qué son a, b, c, i? ¿CERO y UNO? ¿fin y while?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

```
            MOV CERO, c
            MOV CERO, i
while:      CMP i, b      } salto
            BEQ fin       } condicional
            ADD a, c
            ADD UNO, i
            CMP i, i      } salto
            BEQ while     } incondicional
fin:
```

Simbolos
a → @100
b → @101
c → @102
i → @103

UNO → @104
CERO → @105

¿4 son pocas instrucciones?

¿Qué son a, b, c, i? ¿CERO y UNO? ¿fin y while?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

```
@0        MOV CERO, c
@1        MOV CERO, i
@2  while: CMP i, b      } salto
@3        BEQ fin        } condicional
@4        ADD a, c
@5        ADD UNO, i
@6        CMP i, i       } salto
@7        BEQ while      } incondicional
@8  fin:
```

Simbolos
a → @100
b → @101
c → @102
i → @103

UNO → @104
CERO → @105

¿4 son pocas instrucciones?

¿Qué son a, b, c, i? ¿CERO y UNO? ¿fin y while?

```
uint16_t   a, b;
…
c= a*b
```

```
c= 0;
i= 0;
while (i!=b){

  c= c+a;
  i= i+1;
}
```

```
@0        MOV CERO, c
@1        MOV CERO, i
@2  while: CMP i, b      } salto
@3        BEQ fin        } condicional
@4        ADD a, c
@5        ADD UNO, i
@6        CMP i, i       } salto
@7        BEQ while      } incondicional
@8  fin:
```

Simbolos
a → @100
b → @101
c → @102
i → @103

UNO → @104
CERO → @105

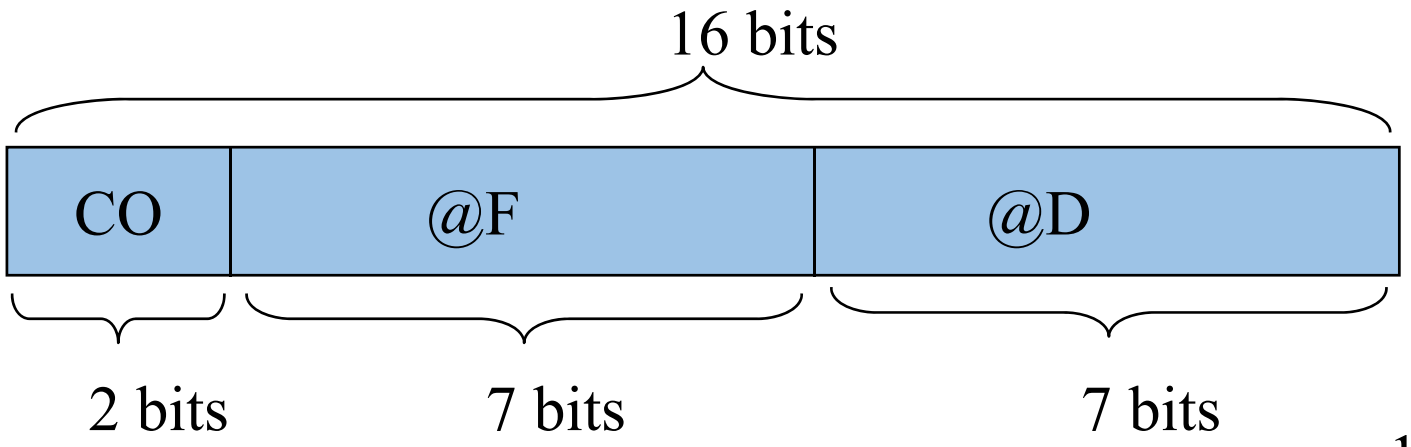
fin → @8
while → @2

¿Cómo representar instrucciones?

- 4 instrucciones → 2 bits para codificar instrucción
- RAM de 128 → 7 bits para cada operando

¿Cómo representar instrucciones?

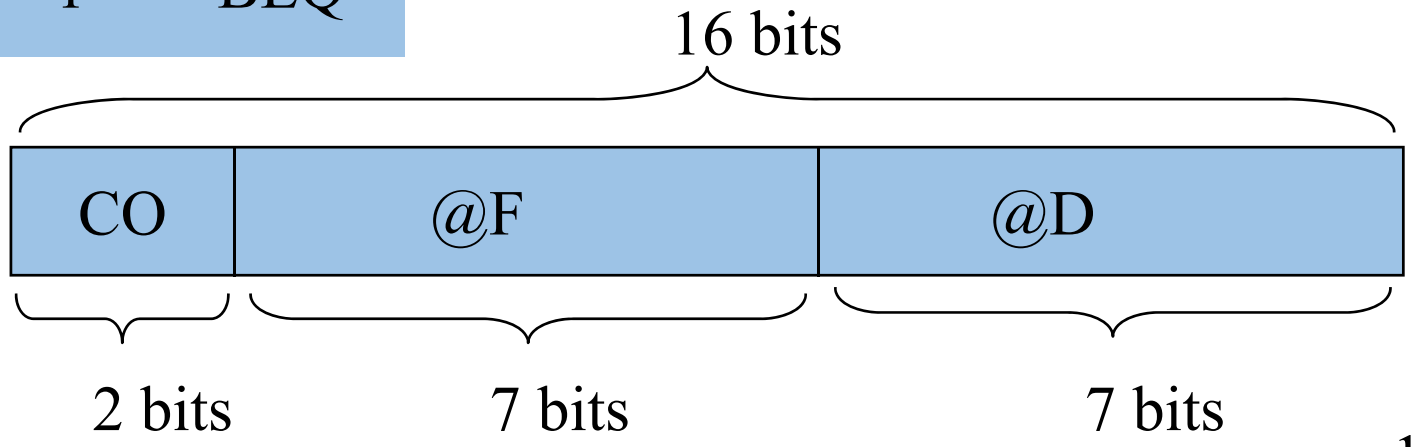
- 4 instrucciones → 2 bits para codificar instrucción
- RAM de 128 → 7 bits para cada operando



¿Cómo representar instrucciones?

- 4 instrucciones → 2 bits para codificar instrucción
- RAM de 128 → 7 bits para cada operando

CO_1	CO_0	Inst
0	0	ADD
0	1	CMP
1	0	MOV
1	1	BEQ

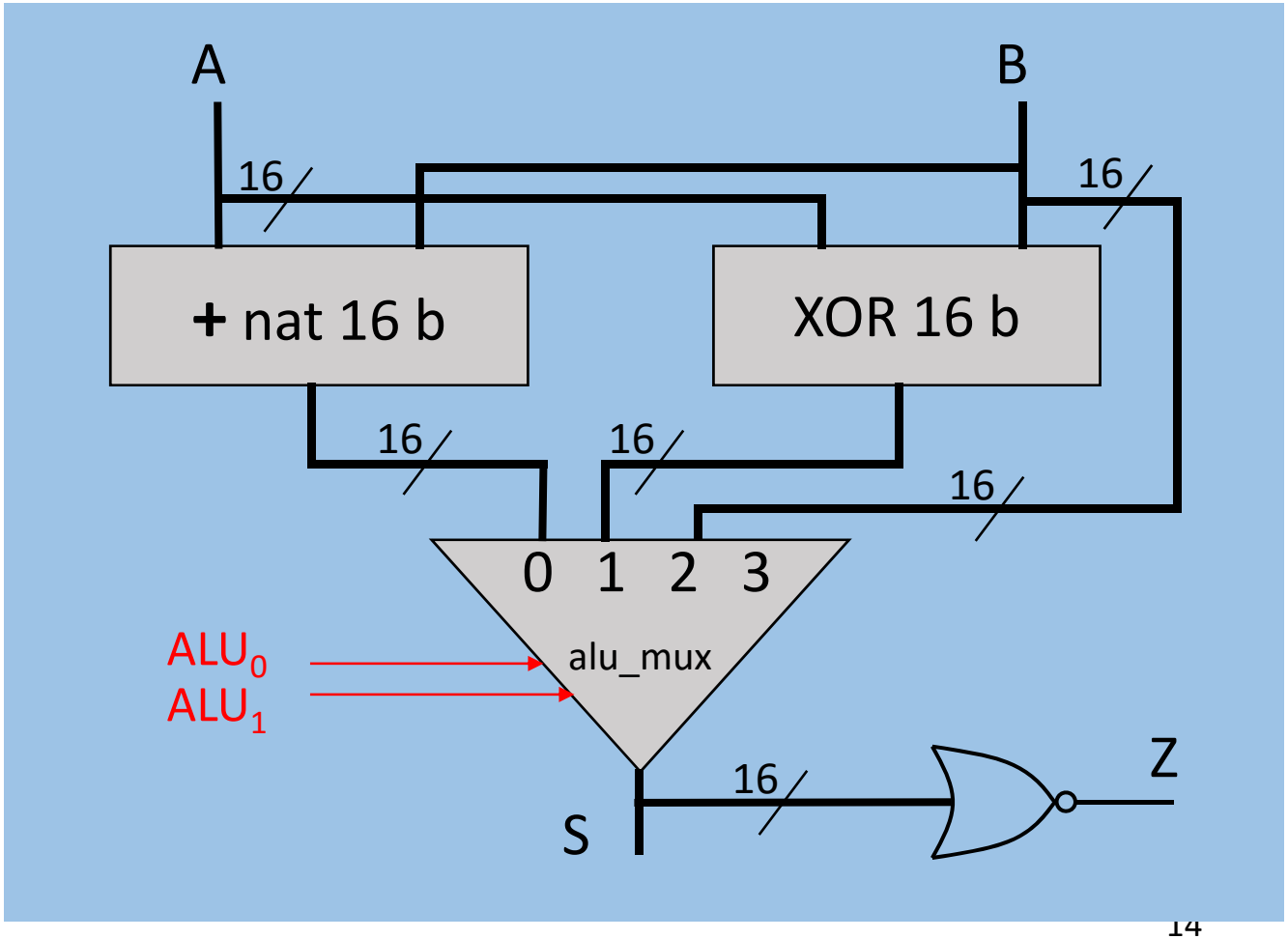


¿Cómo queda la RAM una vez cargado el programa ?

@0	2	105	102
@1	2	105	103
@2	1	103	101
@3	3	X	8
@4	0	100	102
@5	0	104	103
@6	1	103	103
@7	3	X	2
...			
@100	valor de a (16 bits)		
@101	valor de b (16 bits)		
@102	valor de c (16 bits)		
@103	valor de i (16 bits)		
@104	0000000000000001		
@105	0000000000000000		

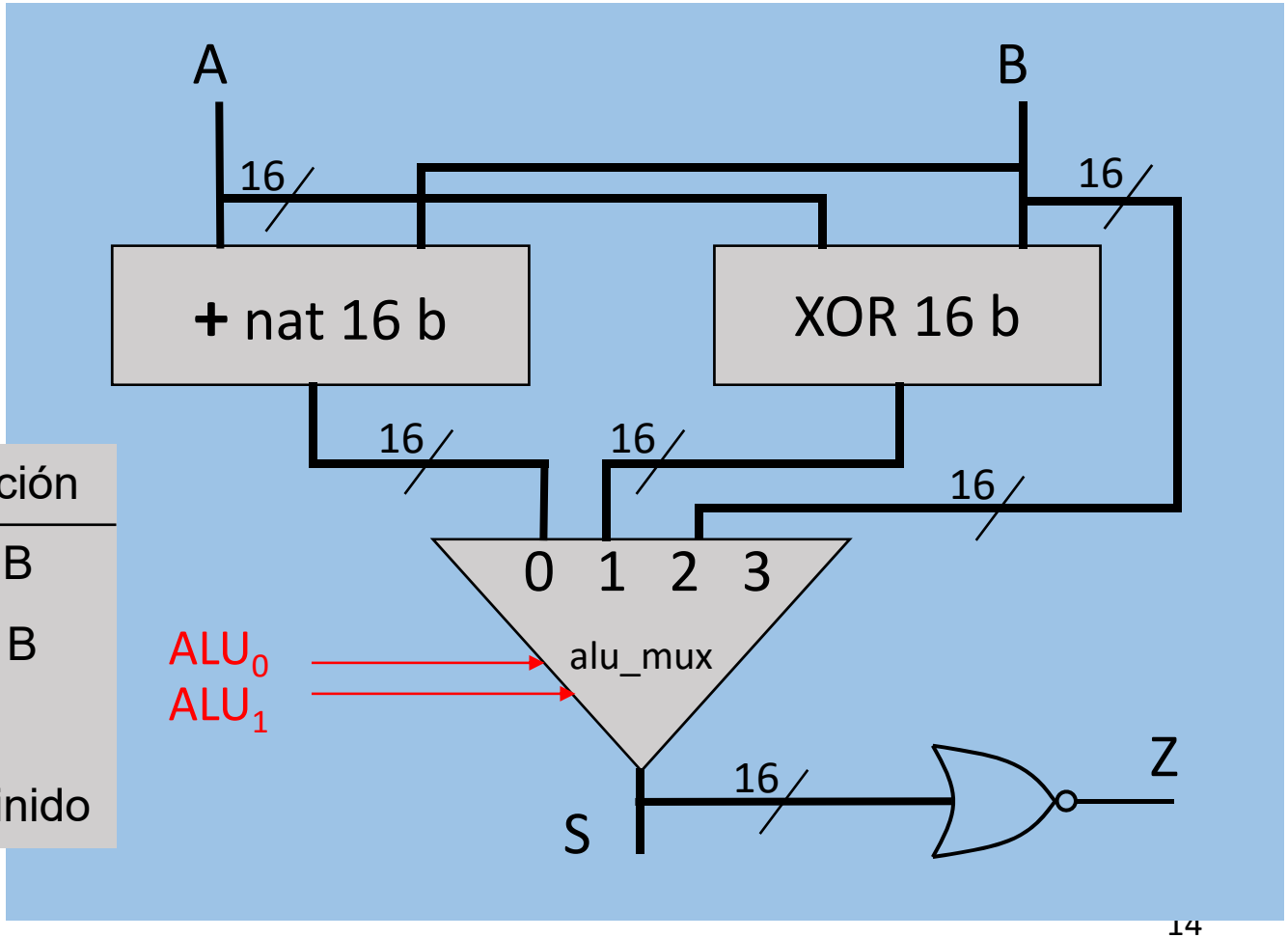
```
        MOV   CERO, c
        MOV   CERO, i
while:  CMP   i, b
        BEQ   fin
        ADD   a, c
        ADD   UNO, i
        CMP   i, i
        BEQ   while
fin:
```

Diseño de la Unidad de Proceso (o *datapath*)



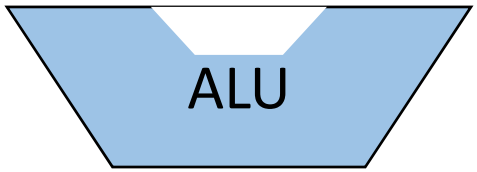
Diseño de la Unidad de Proceso (o *datapath*)

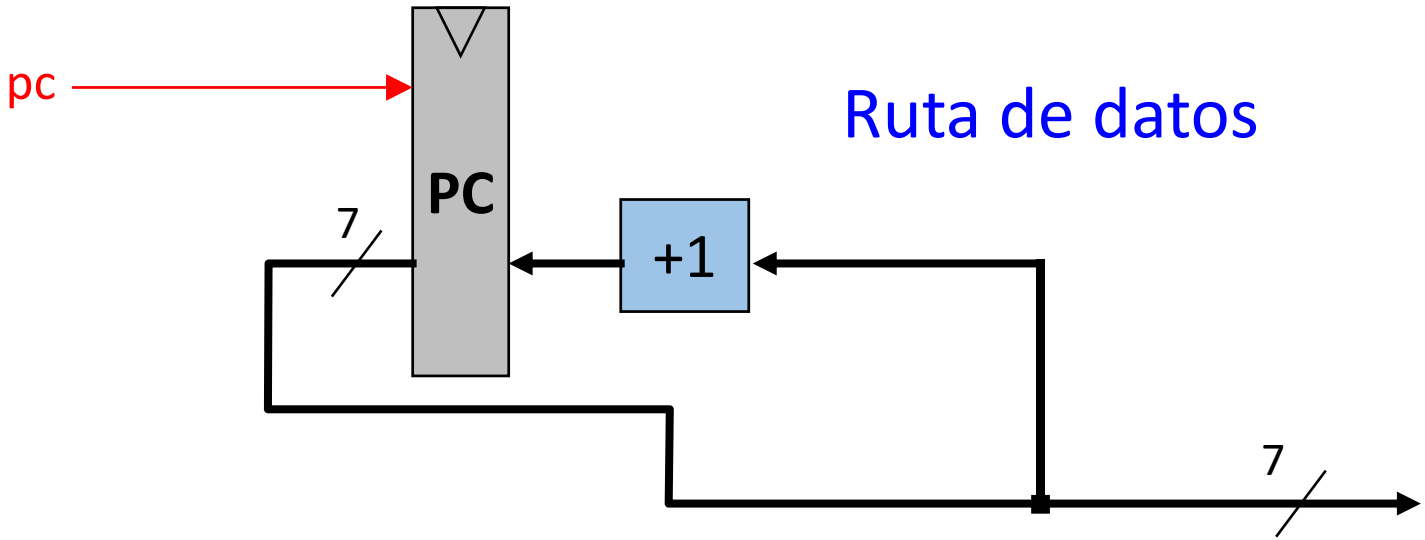
ALU_1	ALU_0	operación
0	0	A + B
0	1	A ⊕ B
1	0	B
1	1	no definido



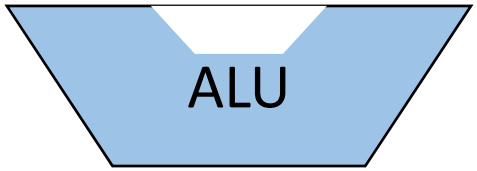
Ruta de datos

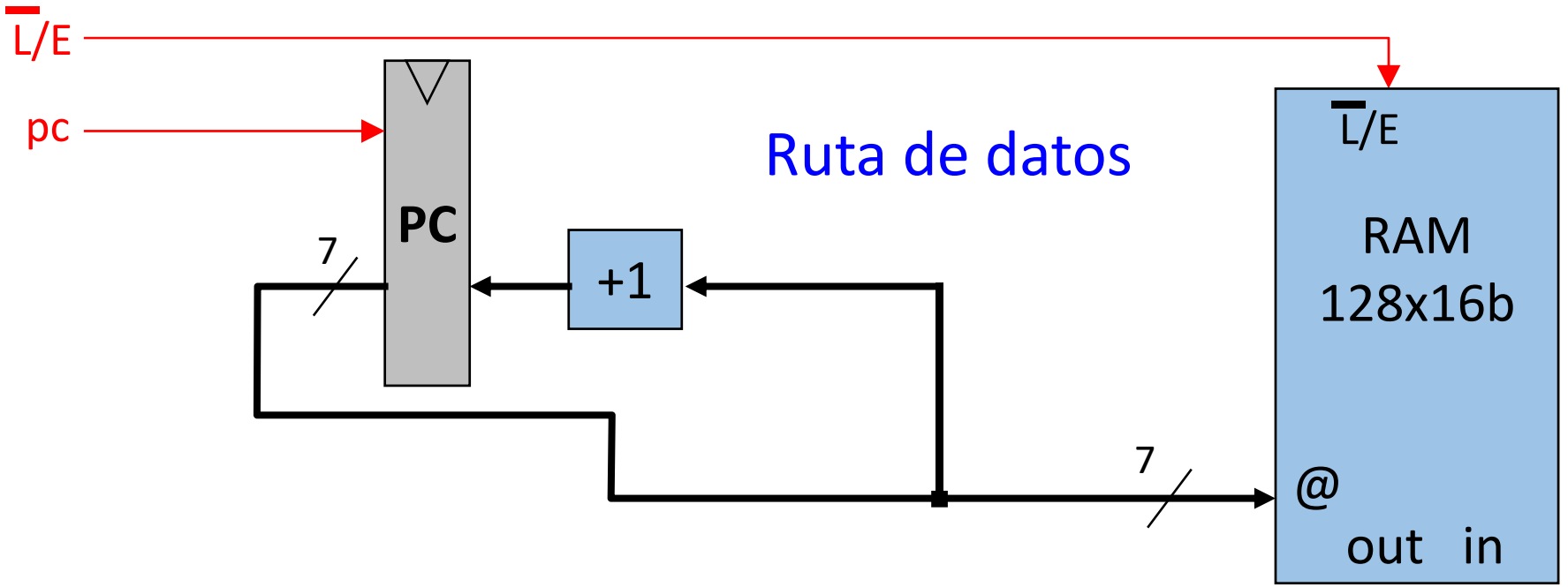
Ruta de datos

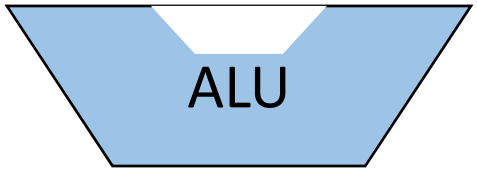


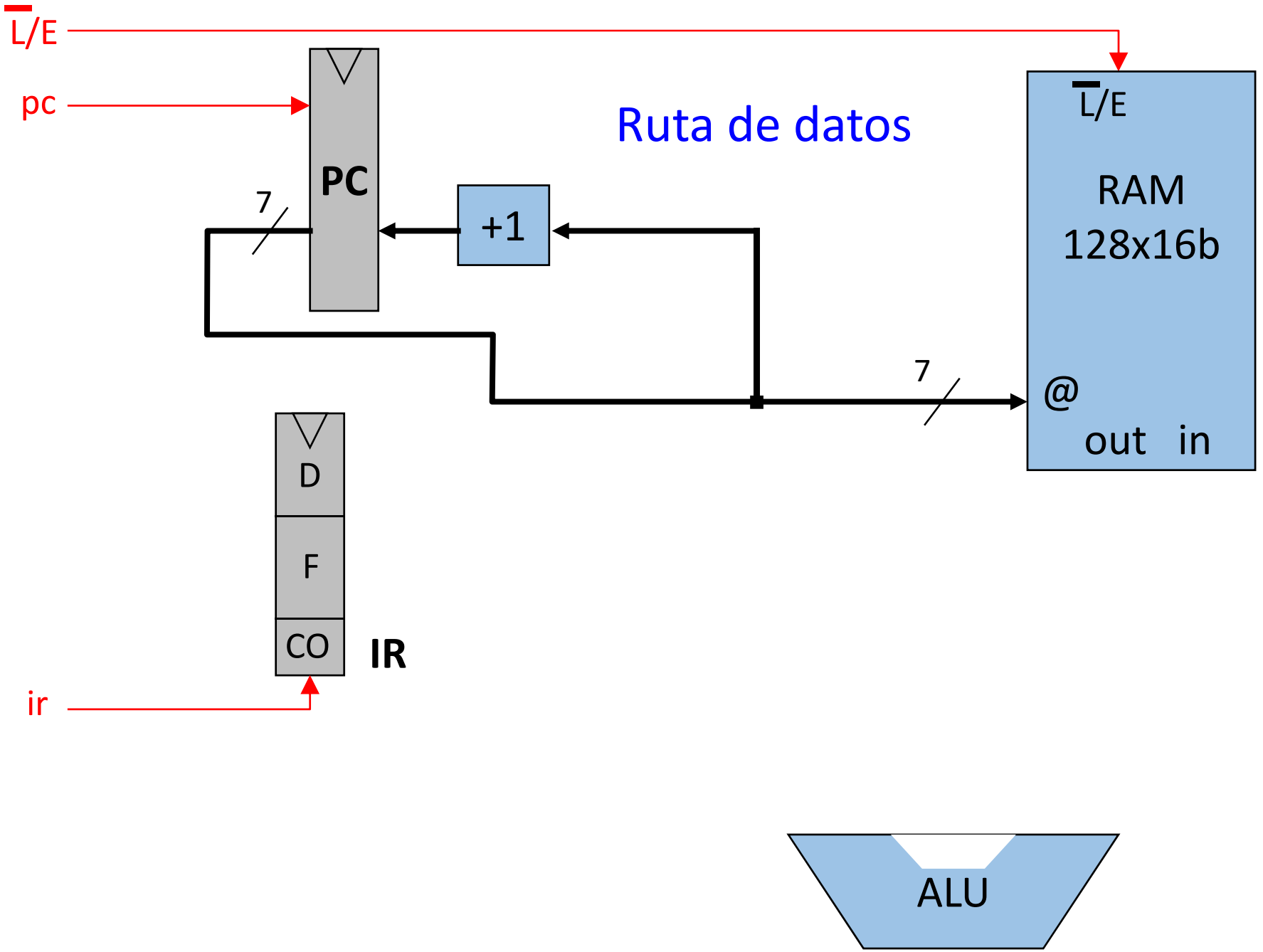


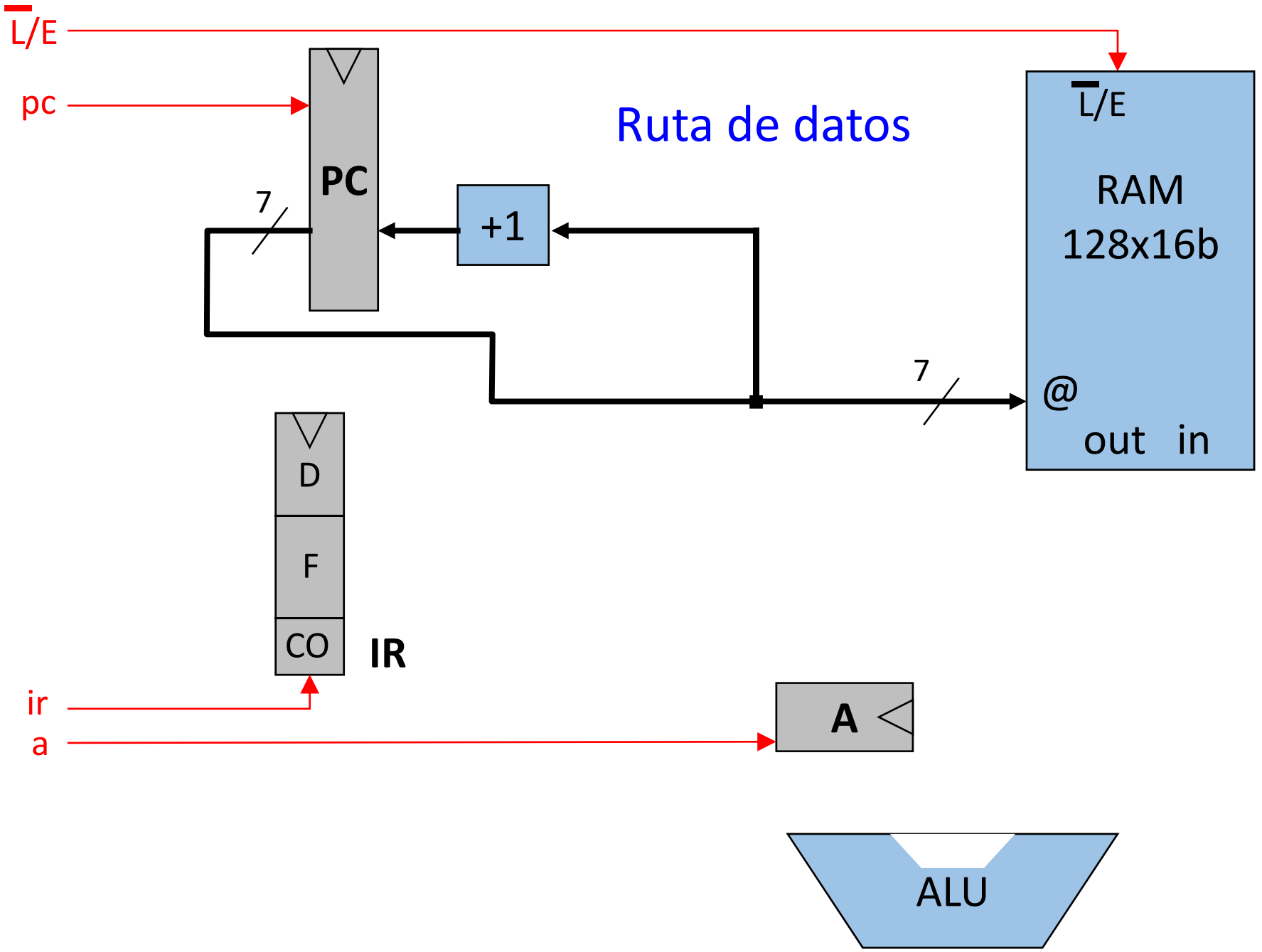
Ruta de datos

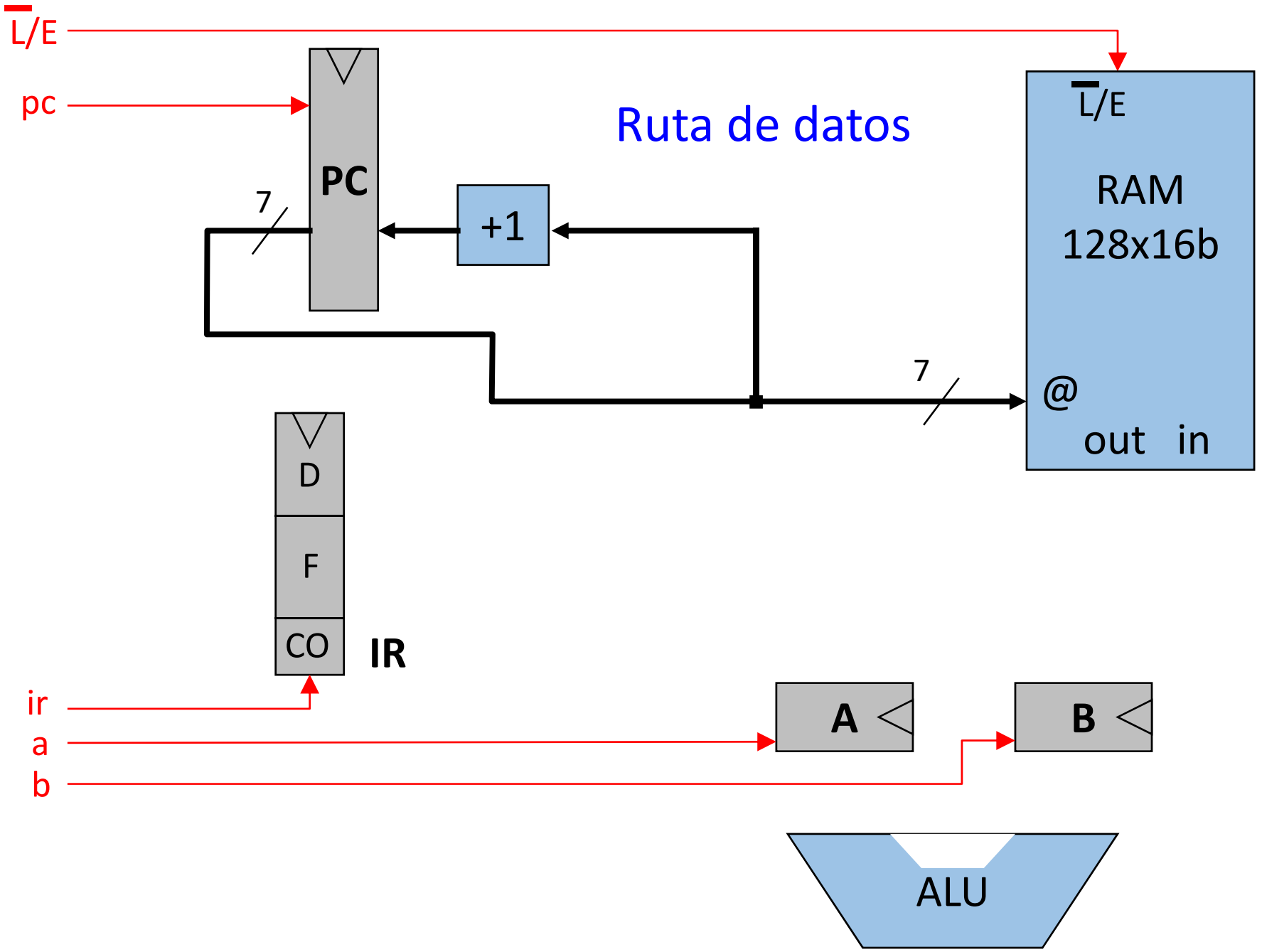


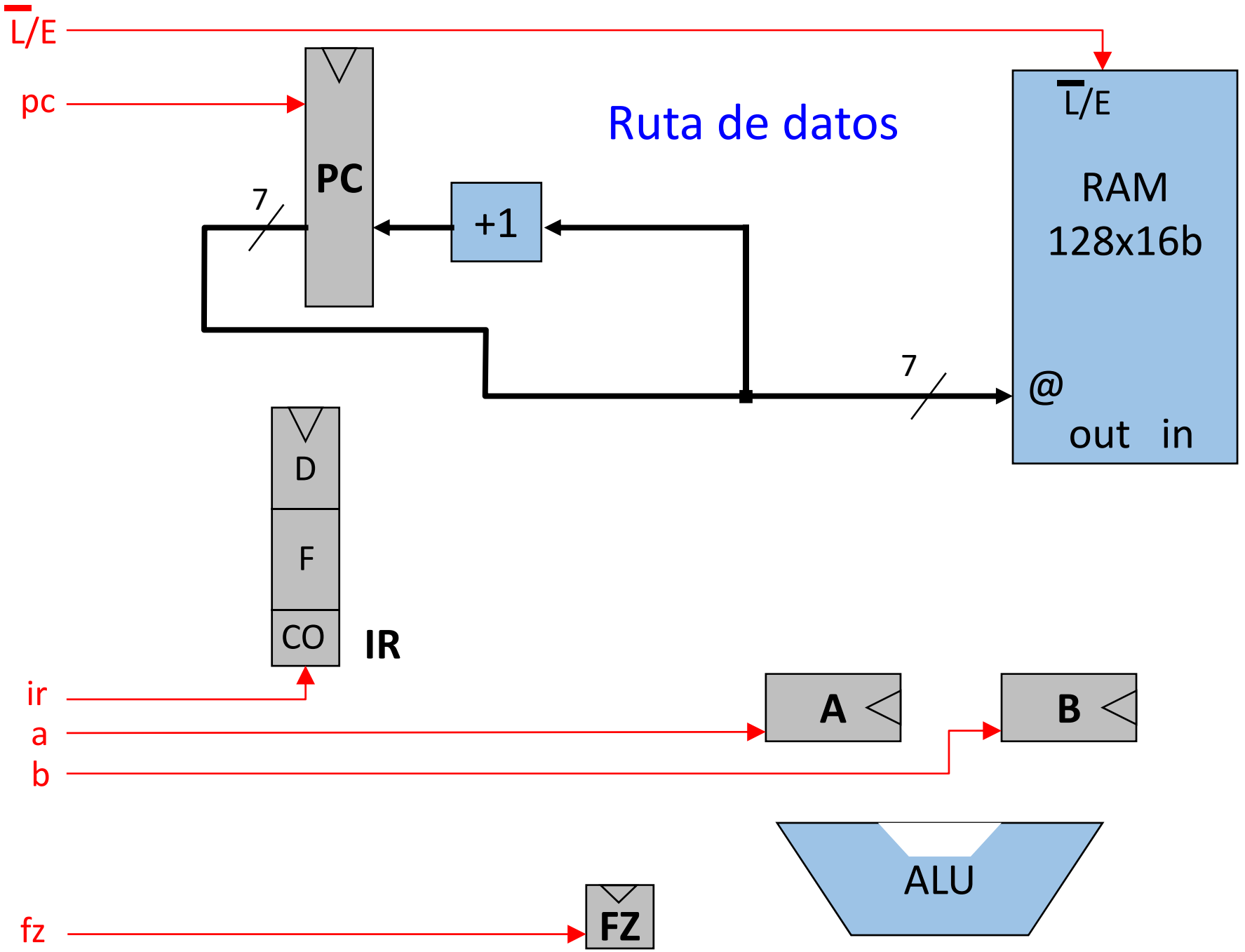

L/E
pc
PC
7
+1
Ruta de datos
L/E
RAM
128x16b
7
@
out in

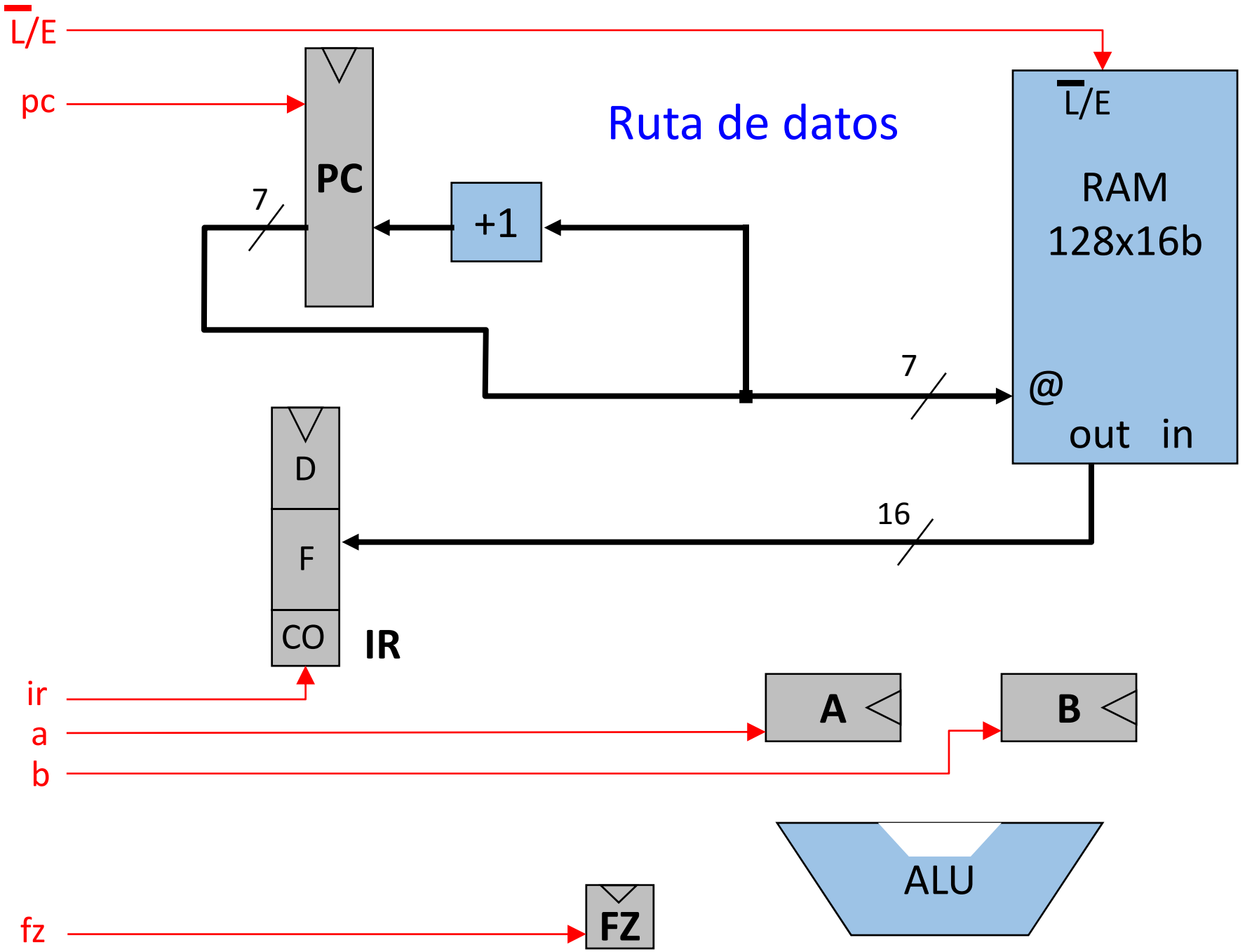

ALU

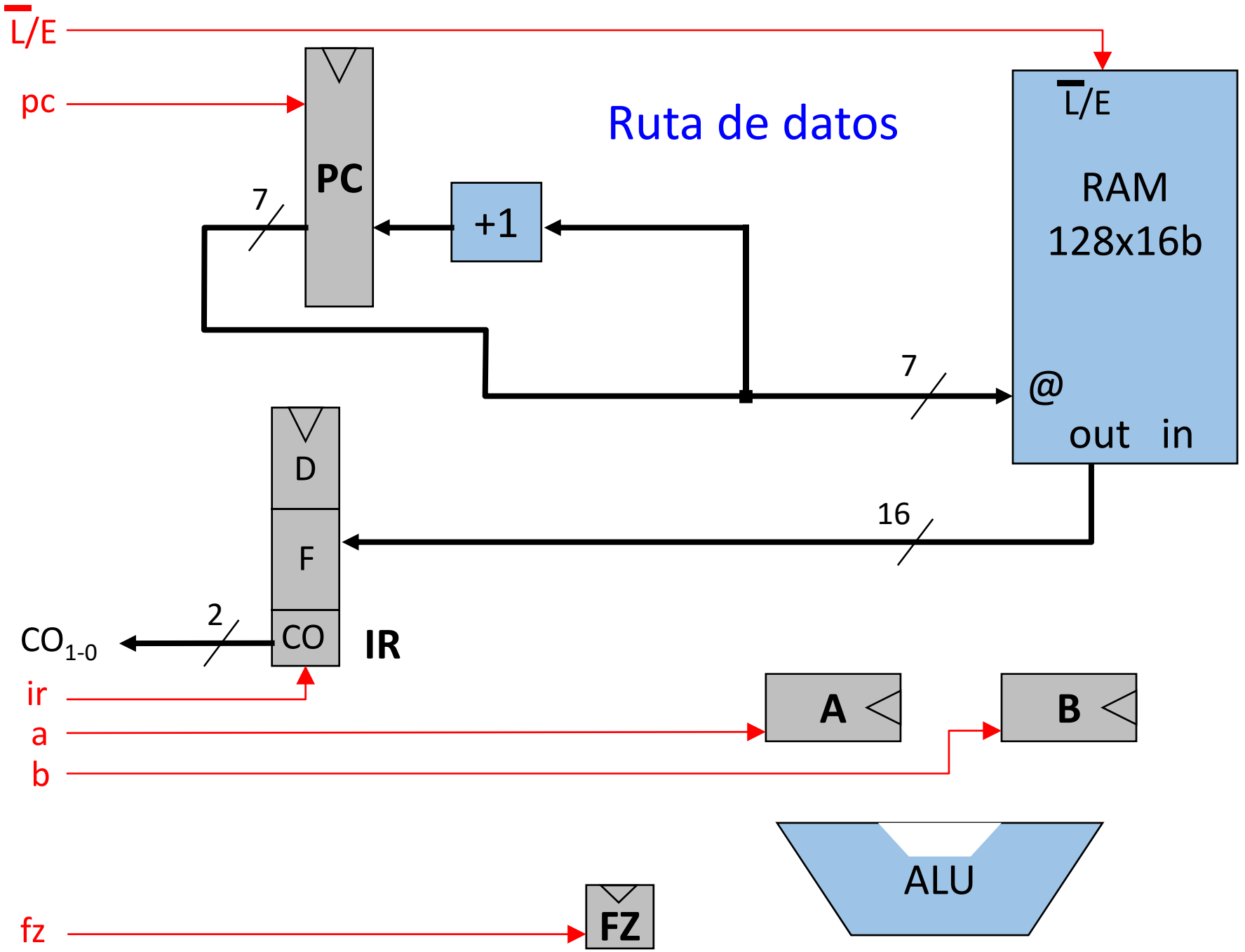

L/E
pc
PC
7
+1
Ruta de datos
L/E
RAM
128x16b
7
@
out in
D
F
CO
IR
ir
ALU

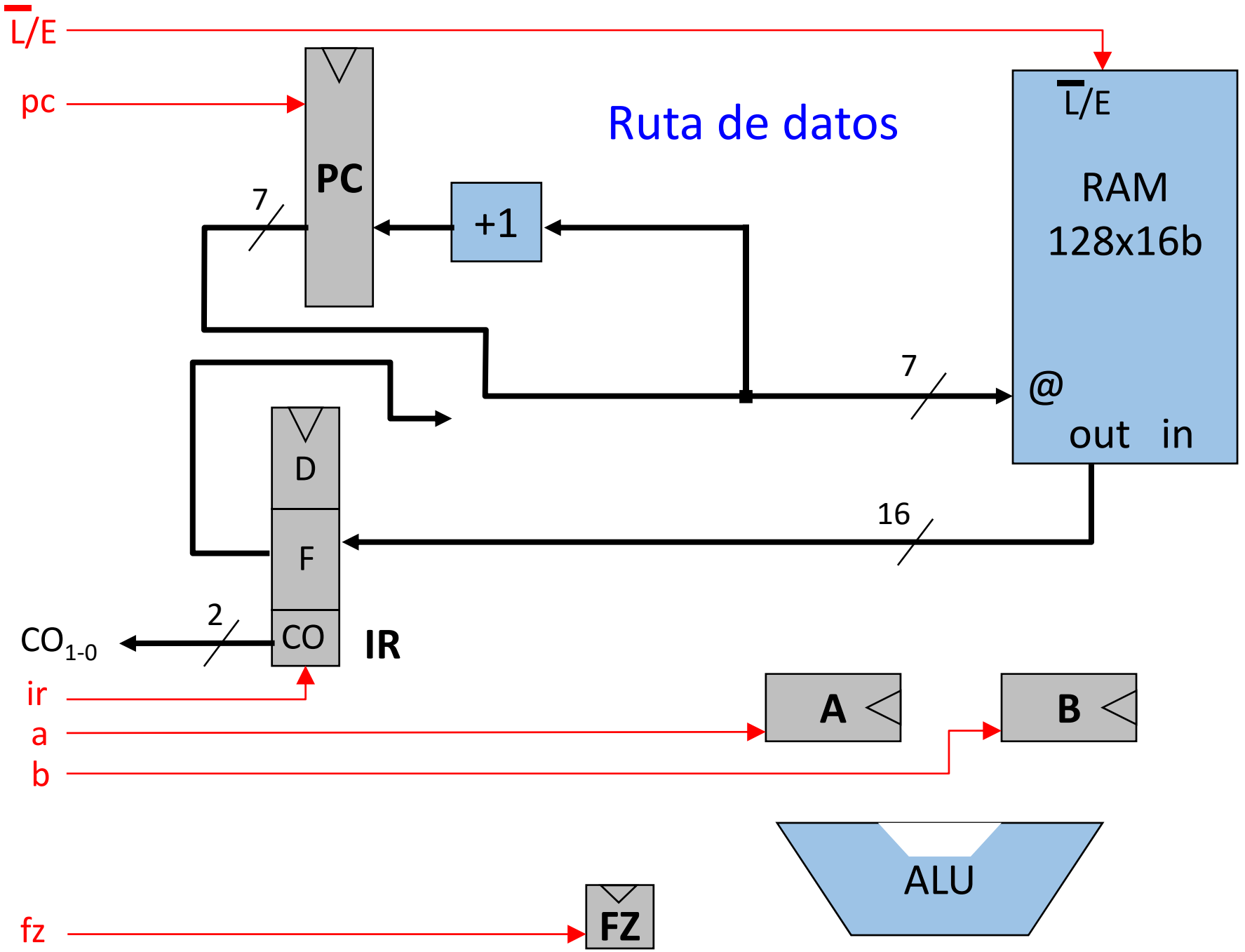

L/E
pc
PC
7
+1
Ruta de datos
L/E
RAM
128x16b
7
@
out in
D
F
CO
IR
ir
a
A
ALU

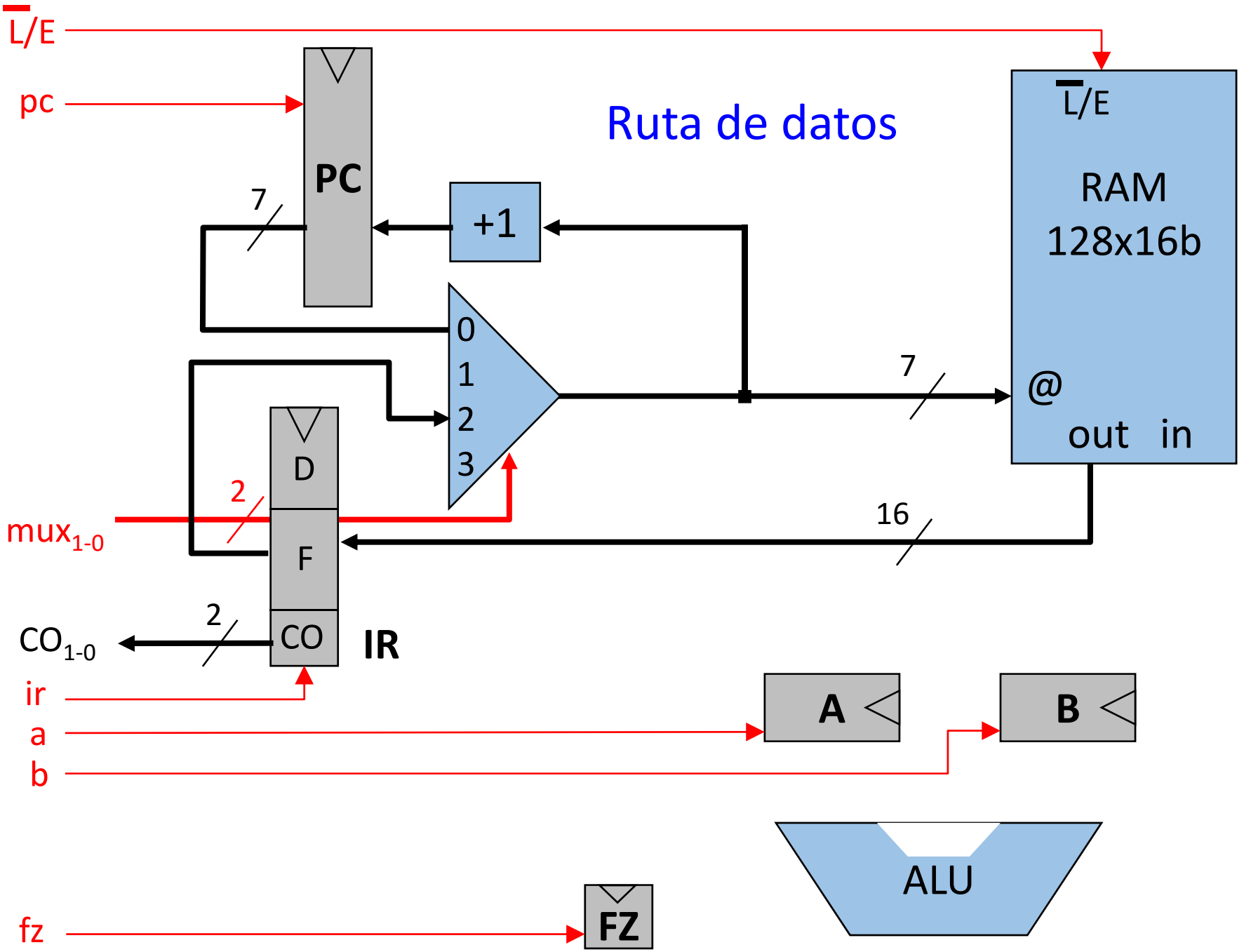

L/E
pc
PC
7
+1
Ruta de datos
L/E
RAM
128x16b
7
@
out in
D
F
CO
IR
ir
a
b
A
B
ALU

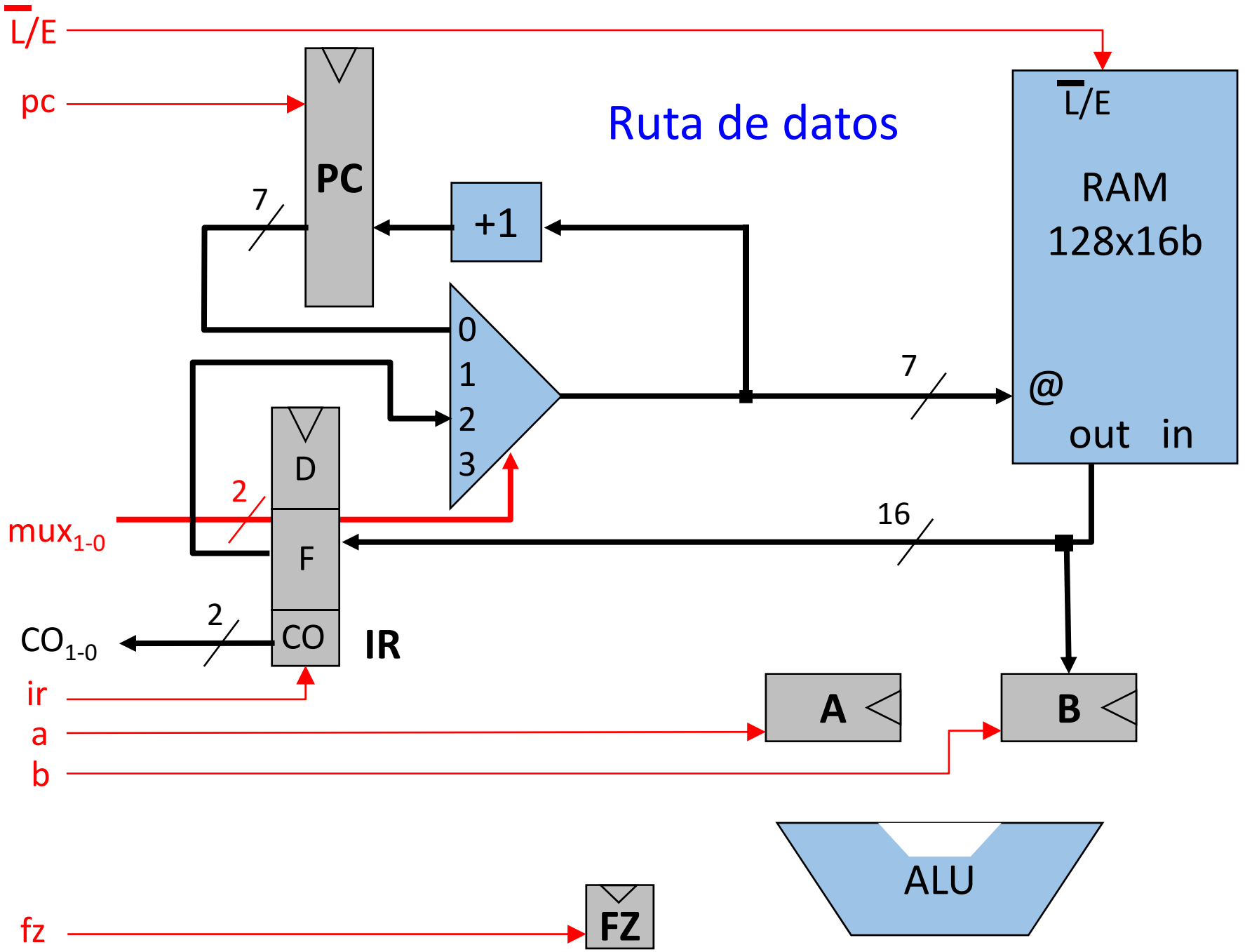

L/E
pc
PC
7
+1
Ruta de datos
L/E
RAM
128x16b
7
@
out in
D
F
CO
IR
ir
a
b
A
B
ALU
fz
FZ

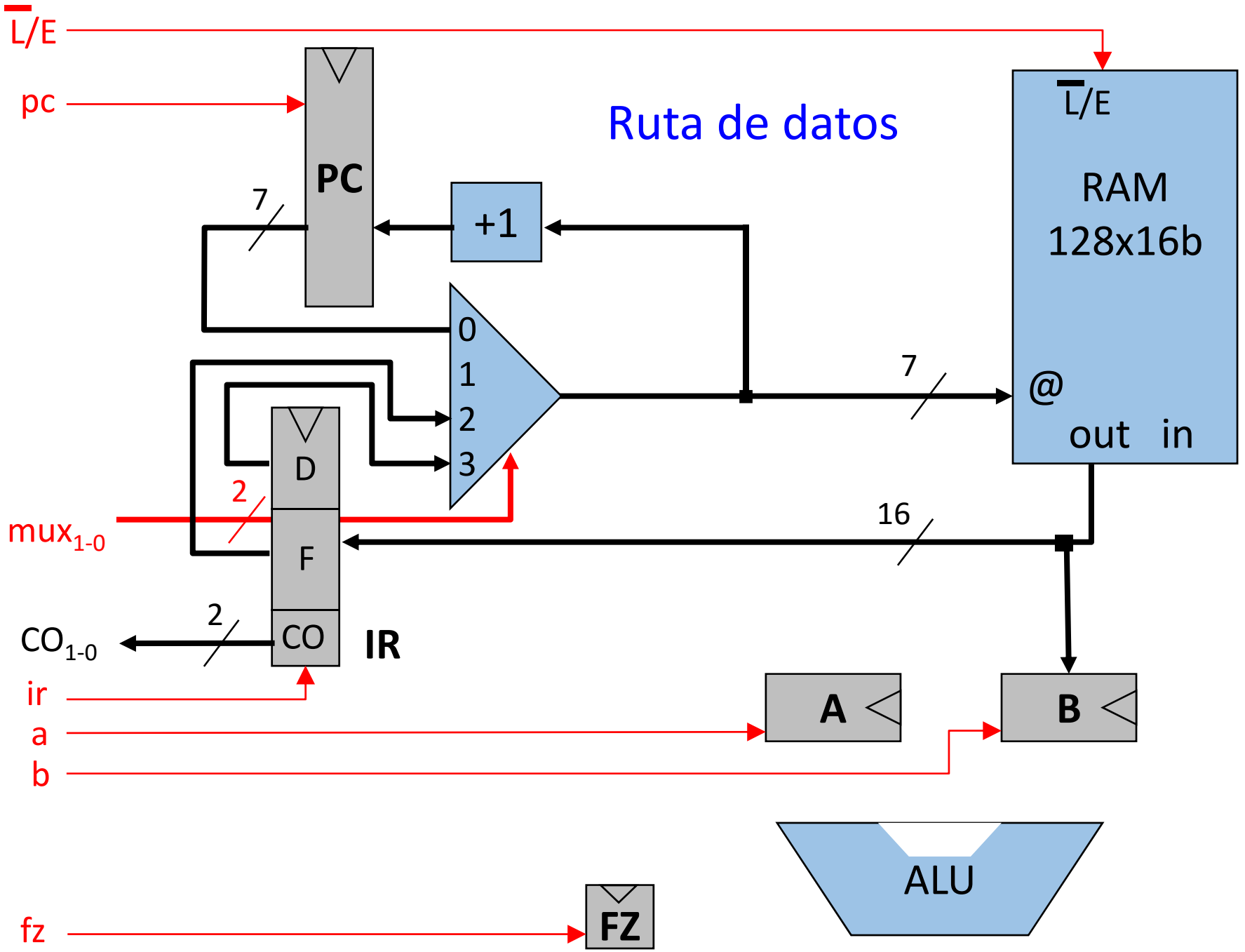

L/E
pc
Ruta de datos
PC
7
+1
L/E
RAM
128x16b
7
@
out in
D
F
CO
IR
16
ir
a
b
A
B
ALU
fz
FZ

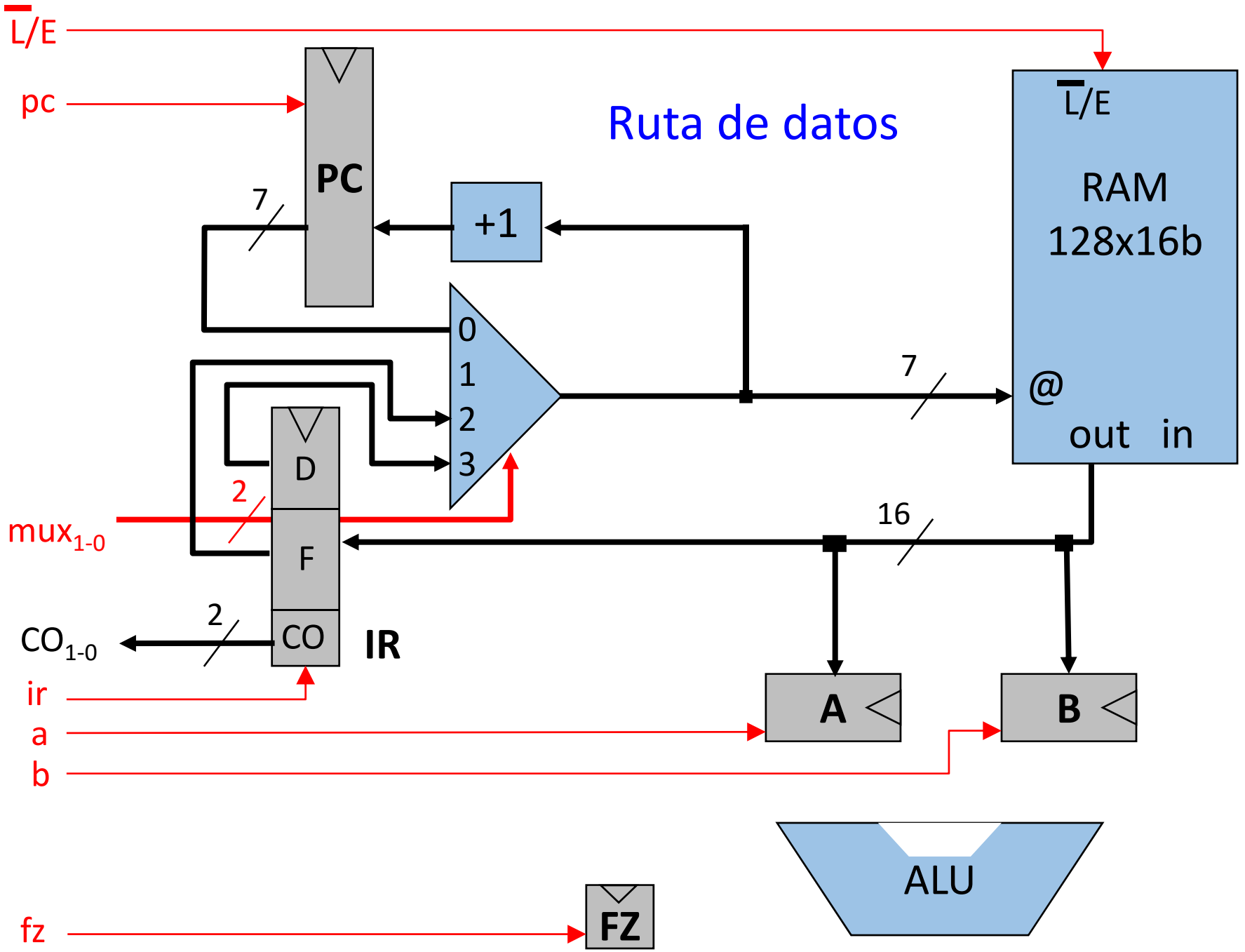

L/E
pc
Ruta de datos
PC
7
+1
L/E
RAM
128x16b
7
@
out in
D
F
16
2
CO
IR
CO1-0
ir
a
b
A
B
ALU
fz
FZ

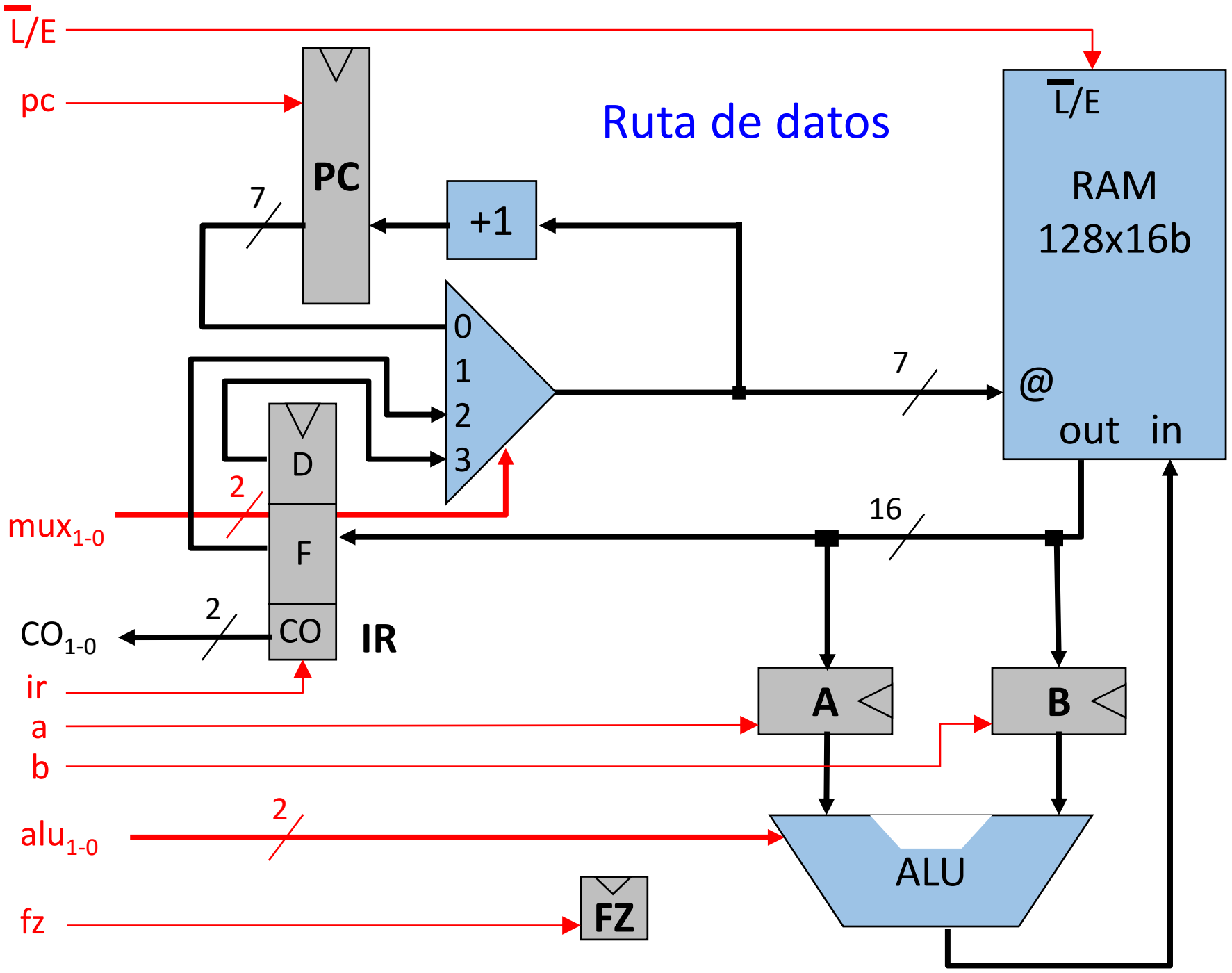

L/E
pc
Ruta de datos
PC
7
+1
L/E
RAM
128x16b
7
@
out in
D
F
16
2
CO
IR
CO1-0
ir
A
B
a
b
ALU
fz
FZ

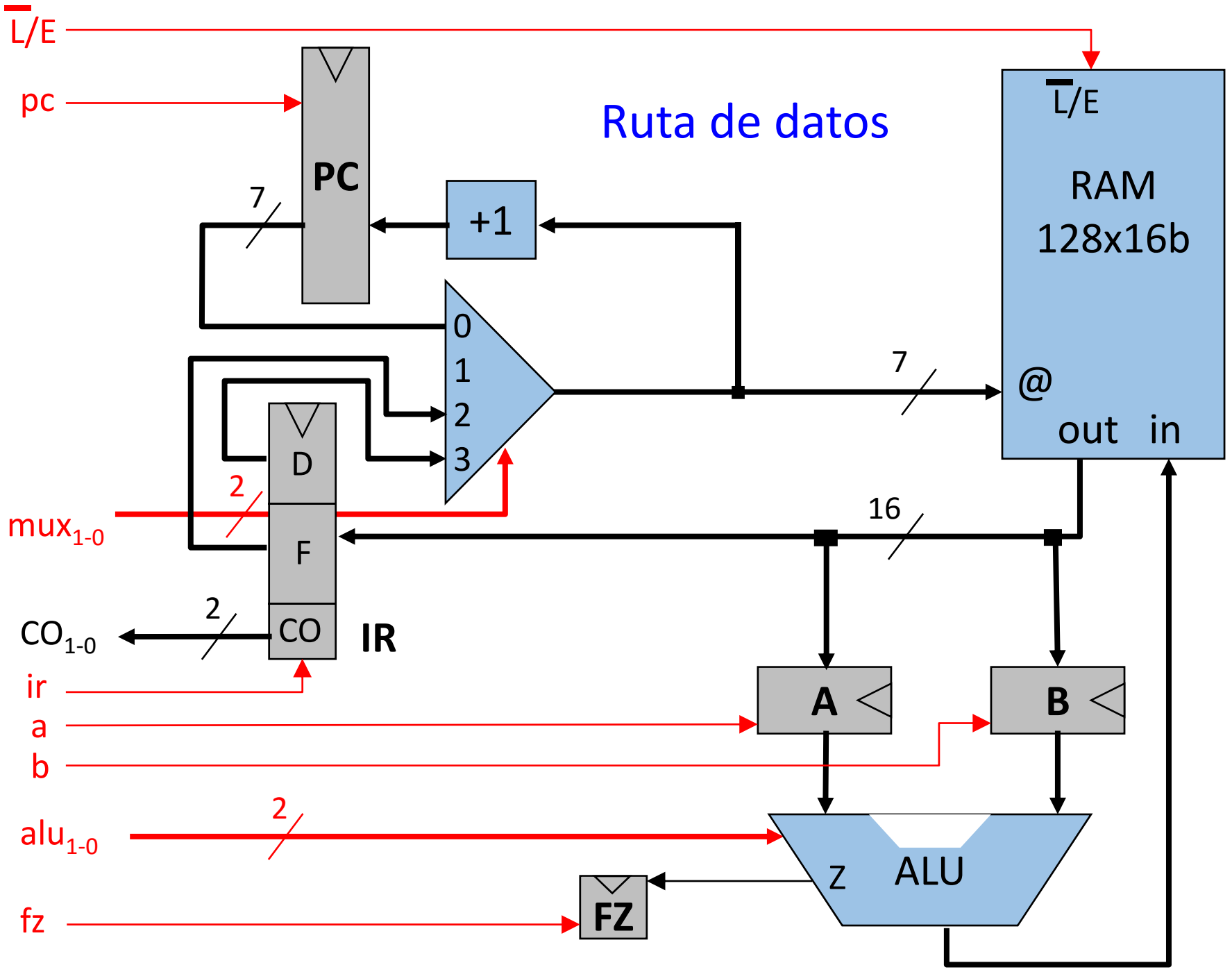

L/E
pc
Ruta de datos
PC
7
+1
L/E
RAM
128x16b
0
1
2
3
7
@
out in
D
2
mux1-0
F
16
2
CO1-0
CO
IR
ir
A
B
a
b
ALU
fz
FZ

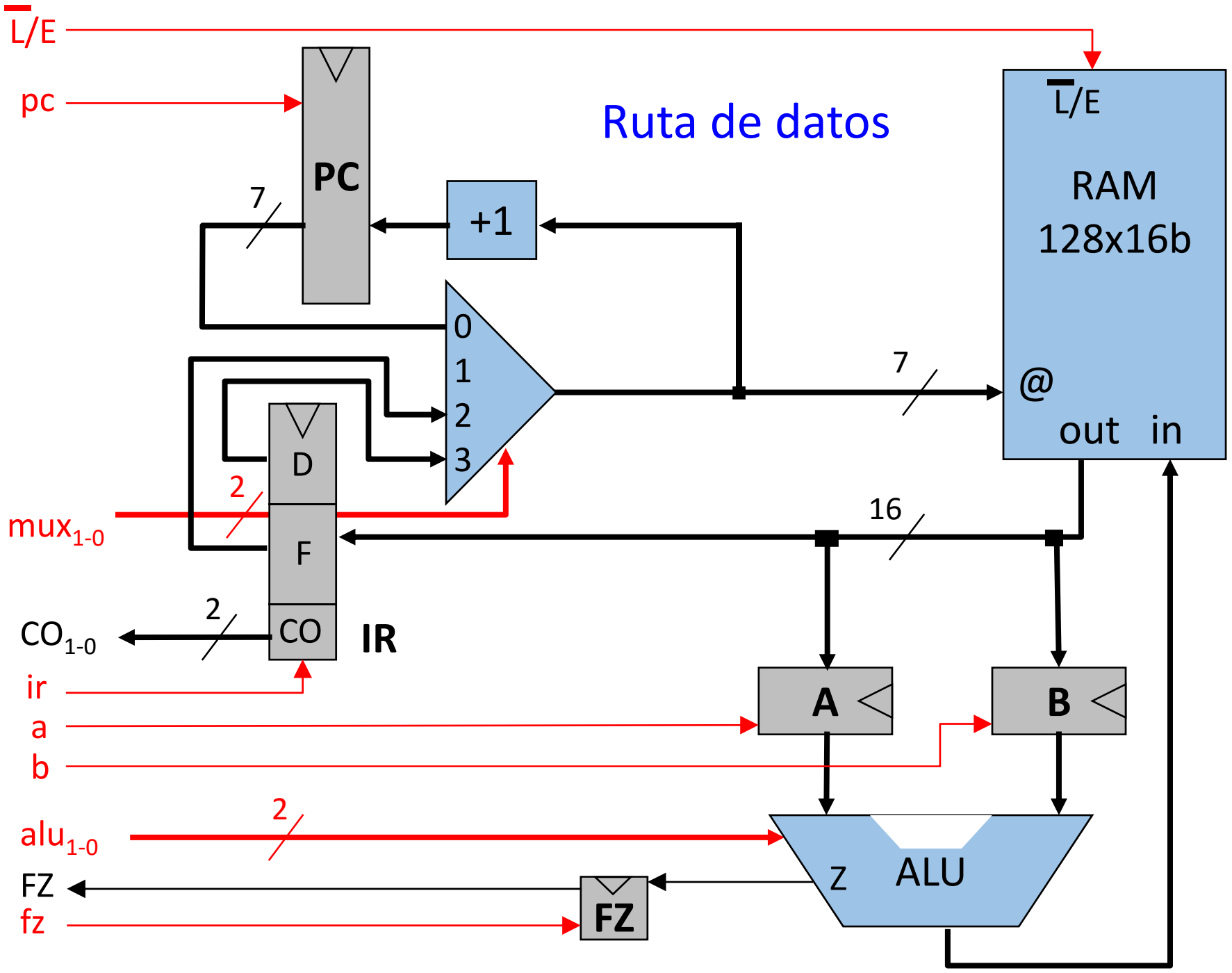

L/E
pc
PC
7
+1
Ruta de datos
L/E
RAM
128x16b
@
out in
0
1
2
3
7
D
2
mux1-0
F
16
2
CO1-0
CO
IR
ir
A
B
a
b
ALU
fz
FZ


L/E
pc
PC
7
+1
Ruta de datos
L/E
RAM
128x16b
@
out in
7
0
1
2
3
D
2
mux1-0
F
16
2
CO1-0
CO
IR
ir
A
B
a
b
ALU
fz
FZ


L/E
pc
PC
7
+1
Ruta de datos
L/E
RAM
128x16b
@
out in
7
0
1
2
3
D
2
mux1-0
F
16
CO1-0
2
CO
IR
ir
A
B
a
b
ALU
fz
FZ


Ruta de datos
L/E
pc
PC
7
+1
0
1
2
3
7
L/E
RAM
128x16b
@
out in
D
2
mux1-0
F
16
2
CO1-0
CO
IR
ir
A
B
a
b
2
alu1-0
ALU
fz
FZ


L/E
pc
PC
7
+1
Ruta de datos
L/E
RAM
128x16b
@
out in
7
0
1
2
3
D
2
mux1-0
F
16
2
CO1-0
CO
IR
ir
A
B
a
b
2
alu1-0
Z
ALU
FZ
fz


Ruta de datos
L/E
pc
PC
7
+1
0
1
2
3
7
L/E
RAM
128x16b
@
out in
D
2
mux1-0
F
16
2
CO1-0
CO
IR
ir
A
B
a
b
2
alu1-0
Z
ALU
FZ
fz
FZ

Cuatro fases de ejecución: interpretación

F1 Buscar instrucción (*fetch*)
RAM → IR

F2 Decodificar (Unidad de Control)

F3 Buscar operandos
RAM → A, RAM → B
o evaluar FZ

F4 Calcular resultado y Z (ALU), escribir RAM y FZ
resultado → RAM
ALU_Z → FZ

Cuatro fases de ejecución: interpretación

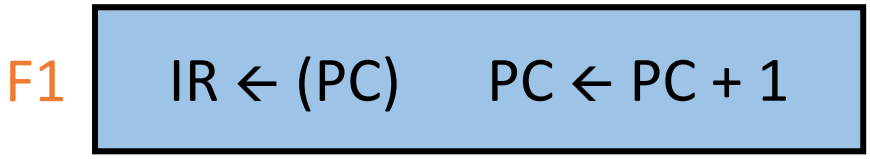
F1 Buscar instrucción (*fetch*)
RAM → IR

F2 Decodificar (Unidad de Control)

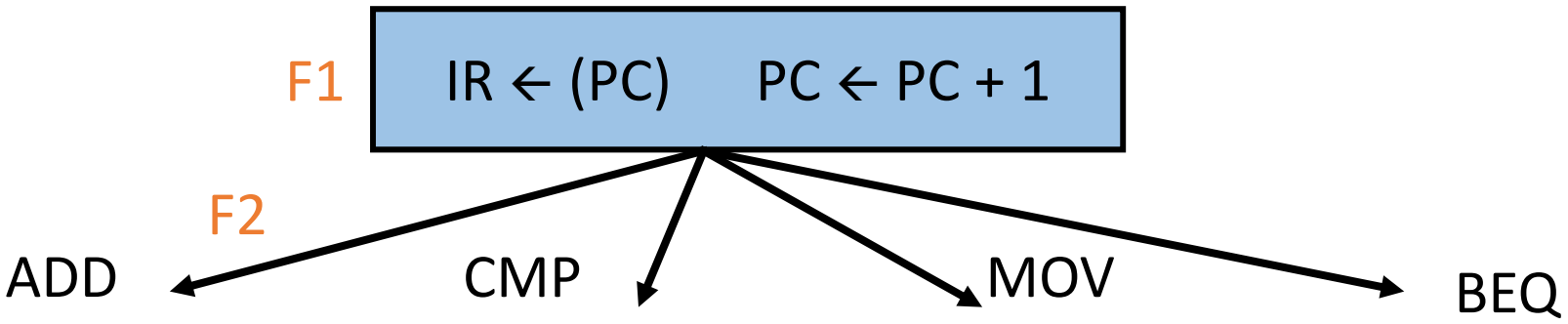
F3 Buscar operandos
RAM → A, RAM → B
o evaluar FZ

F4 Calcular resultado y Z (ALU), escribir RAM y FZ
resultado → RAM
ALU_Z → FZ

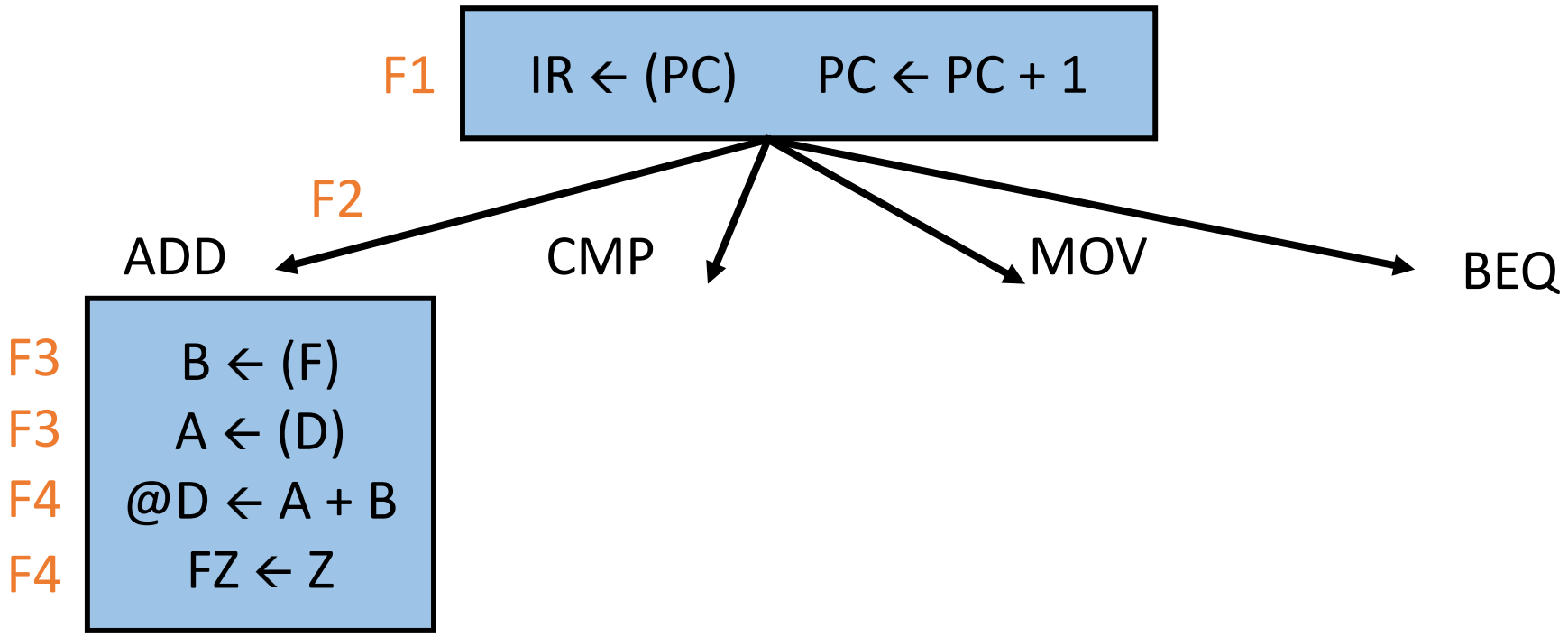
Fases y acciones RTL *(Register Transfer Level)*



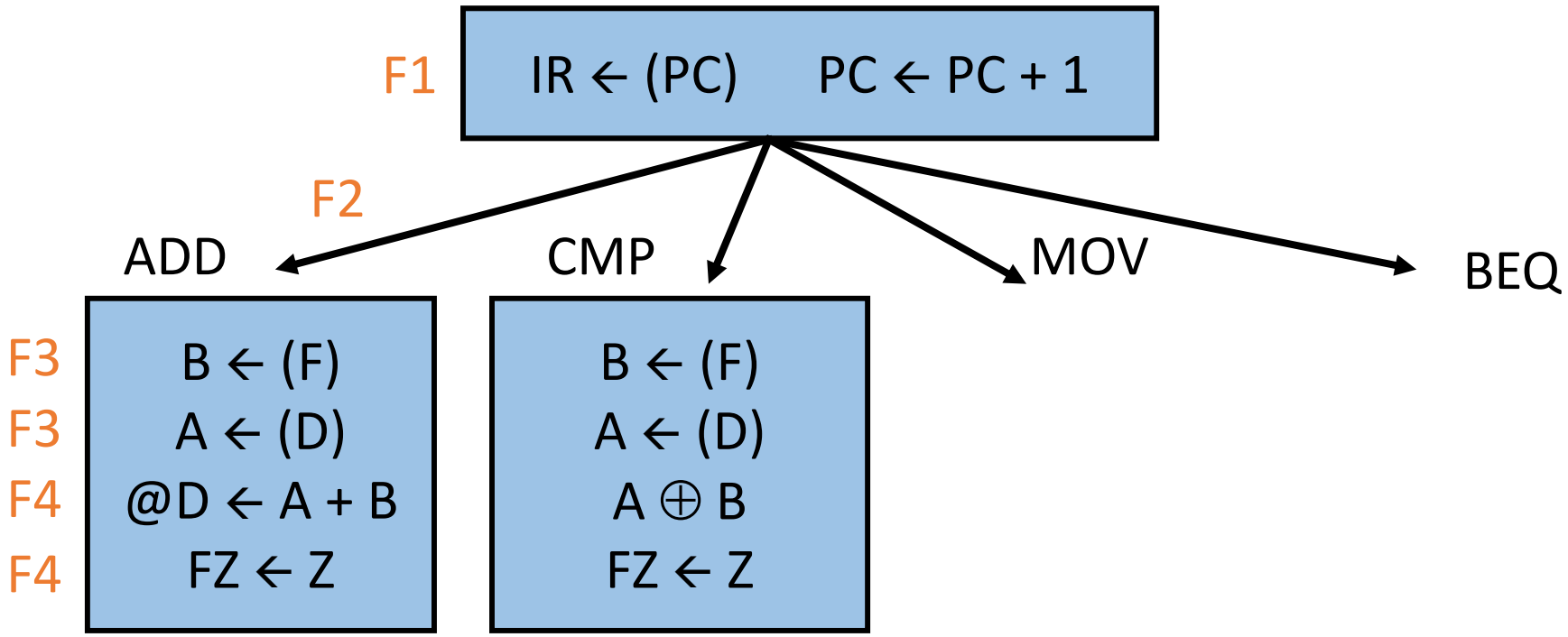
Fases y acciones RTL *(Register Transfer Level)*



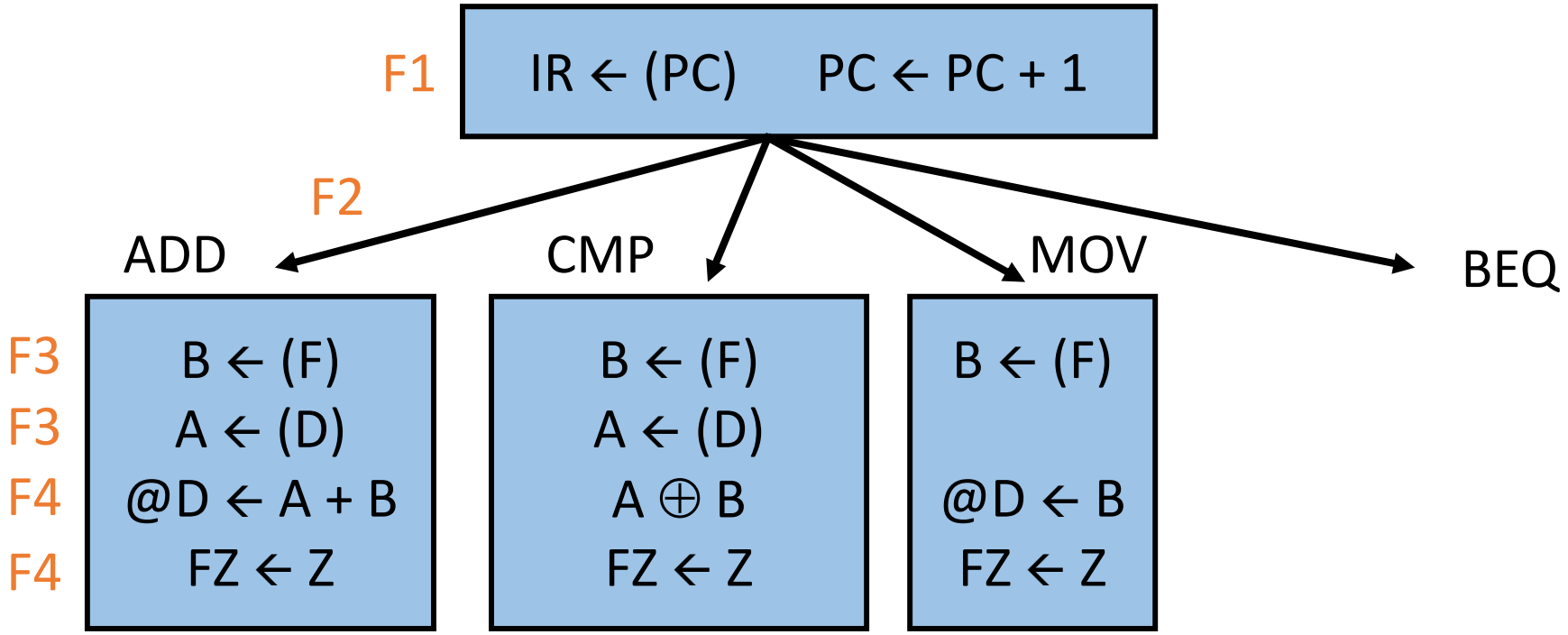
Fases y acciones RTL *(Register Transfer Level)*



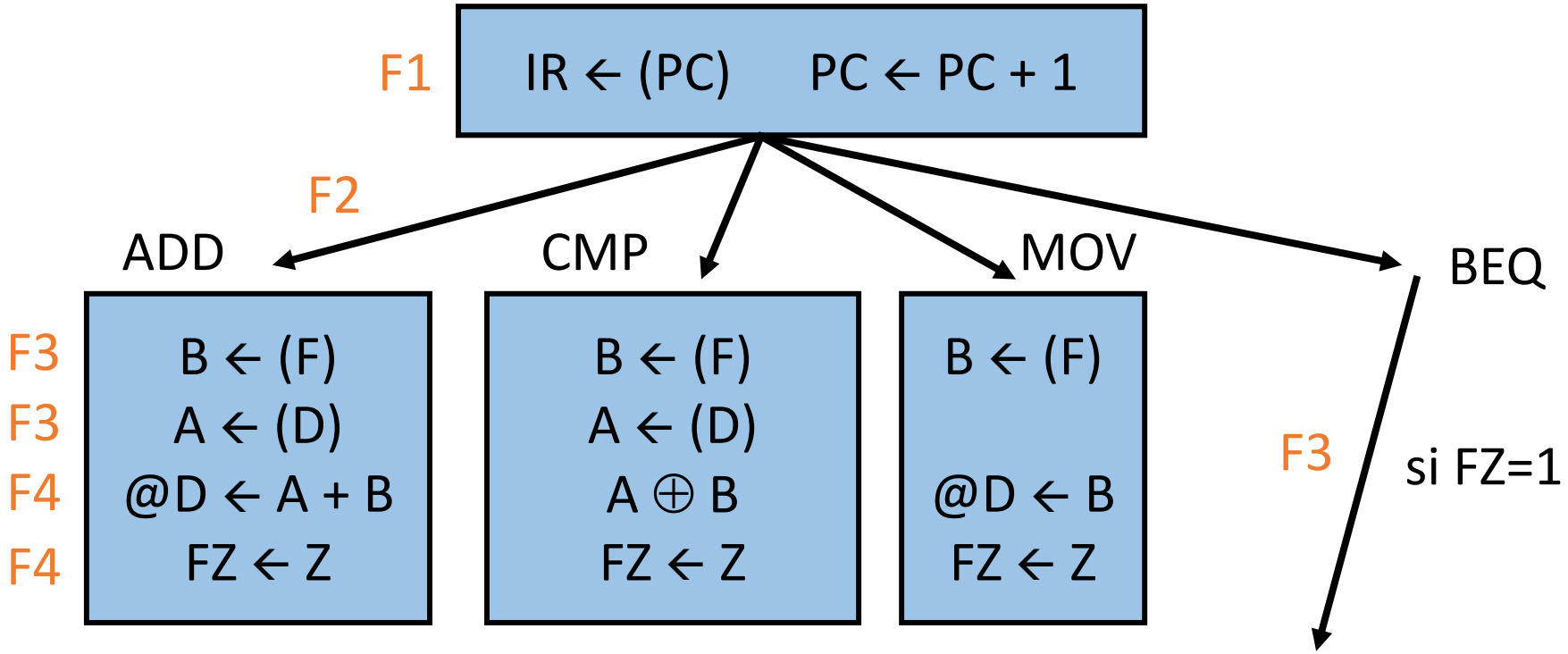
Fases y acciones RTL *(Register Transfer Level)*



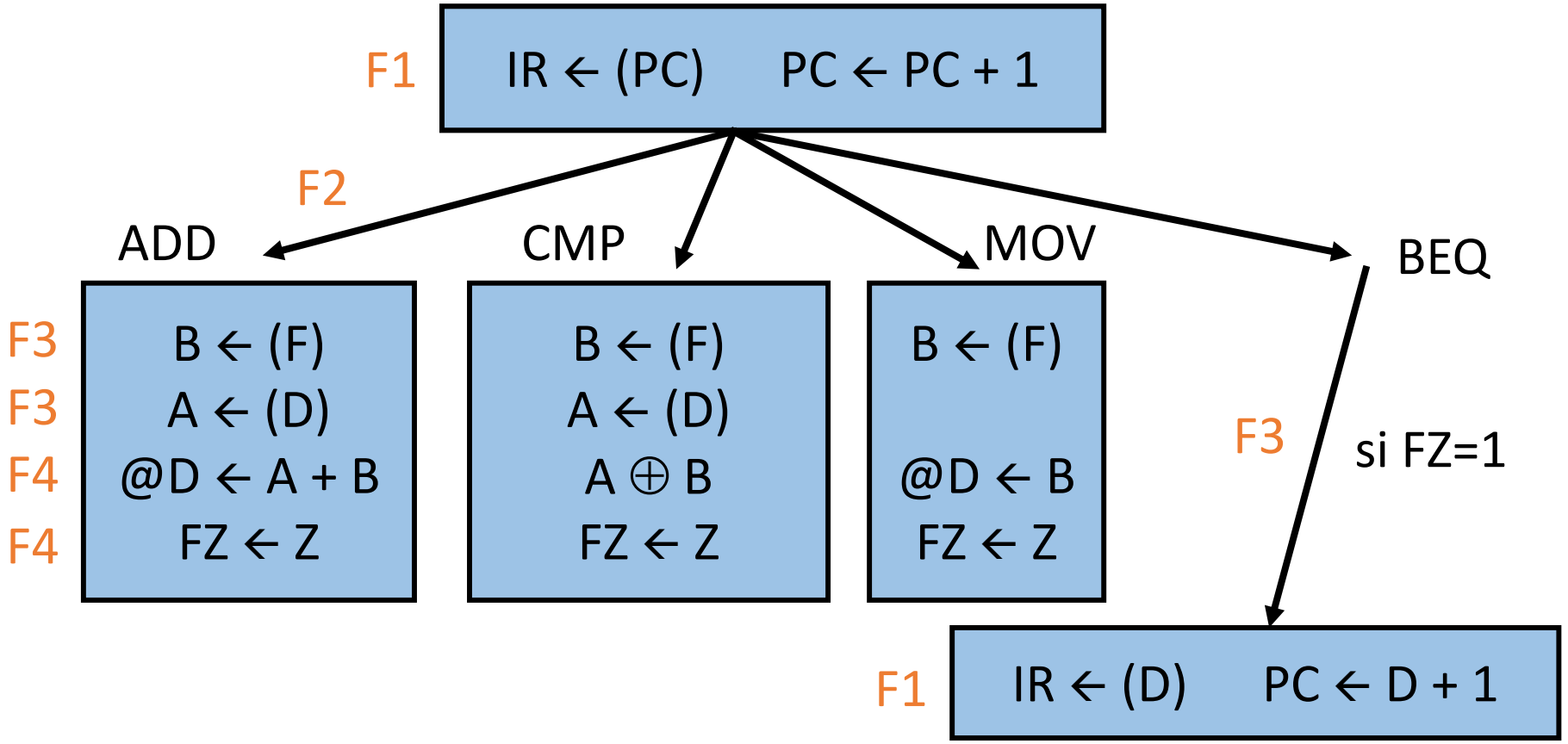
Fases y acciones RTL *(Register Transfer Level)*



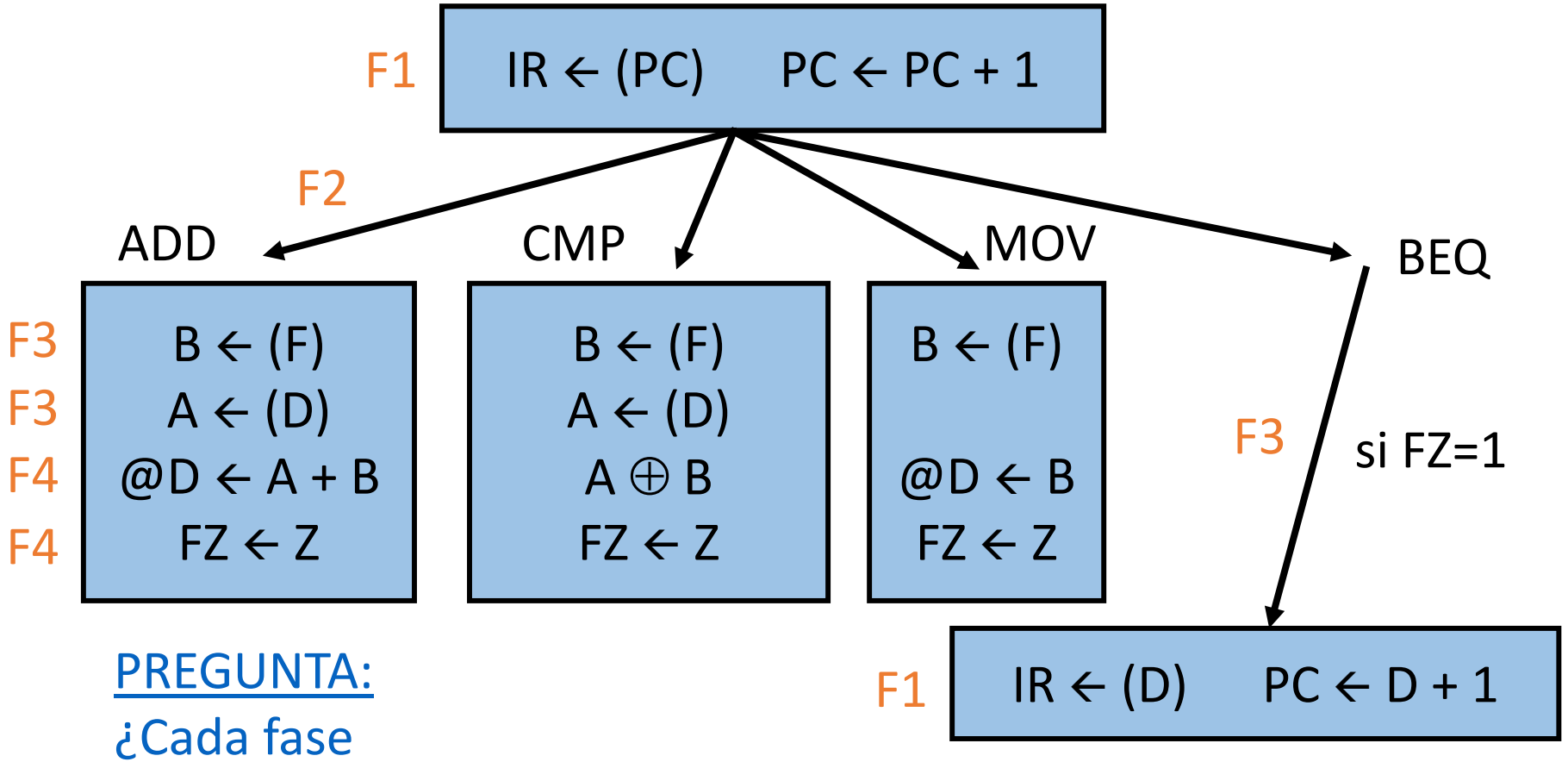
Fases y acciones RTL *(Register Transfer Level)*



Fases y acciones RTL *(Register Transfer Level)*

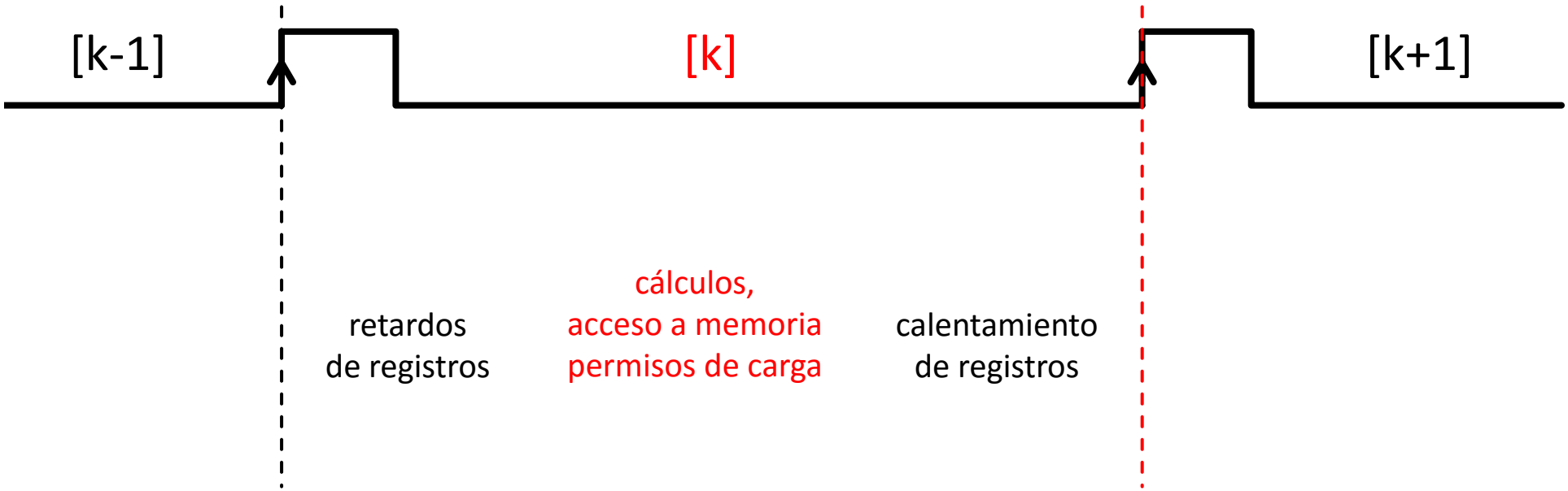


Fases y acciones RTL *(Register Transfer Level)*



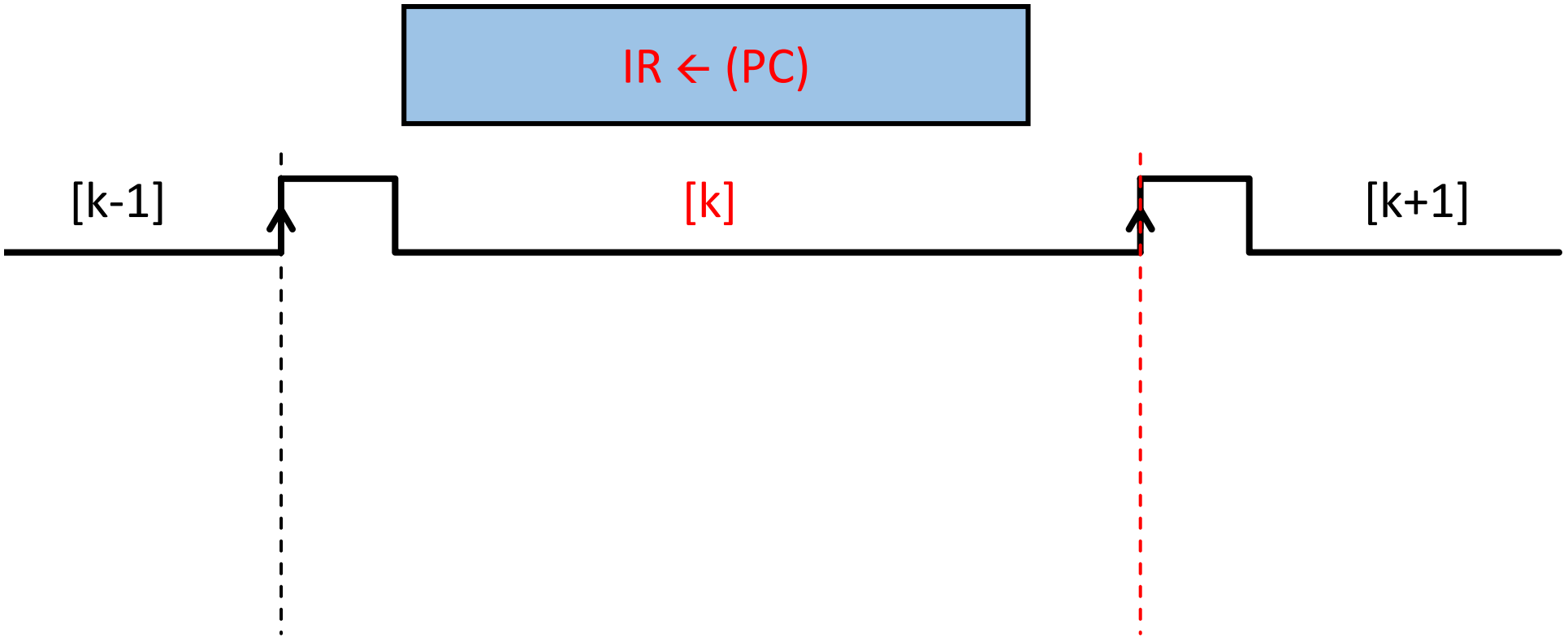
PREGUNTA:
¿Cada fase
es un ciclo de reloj?

Fases y ciclo de procesador



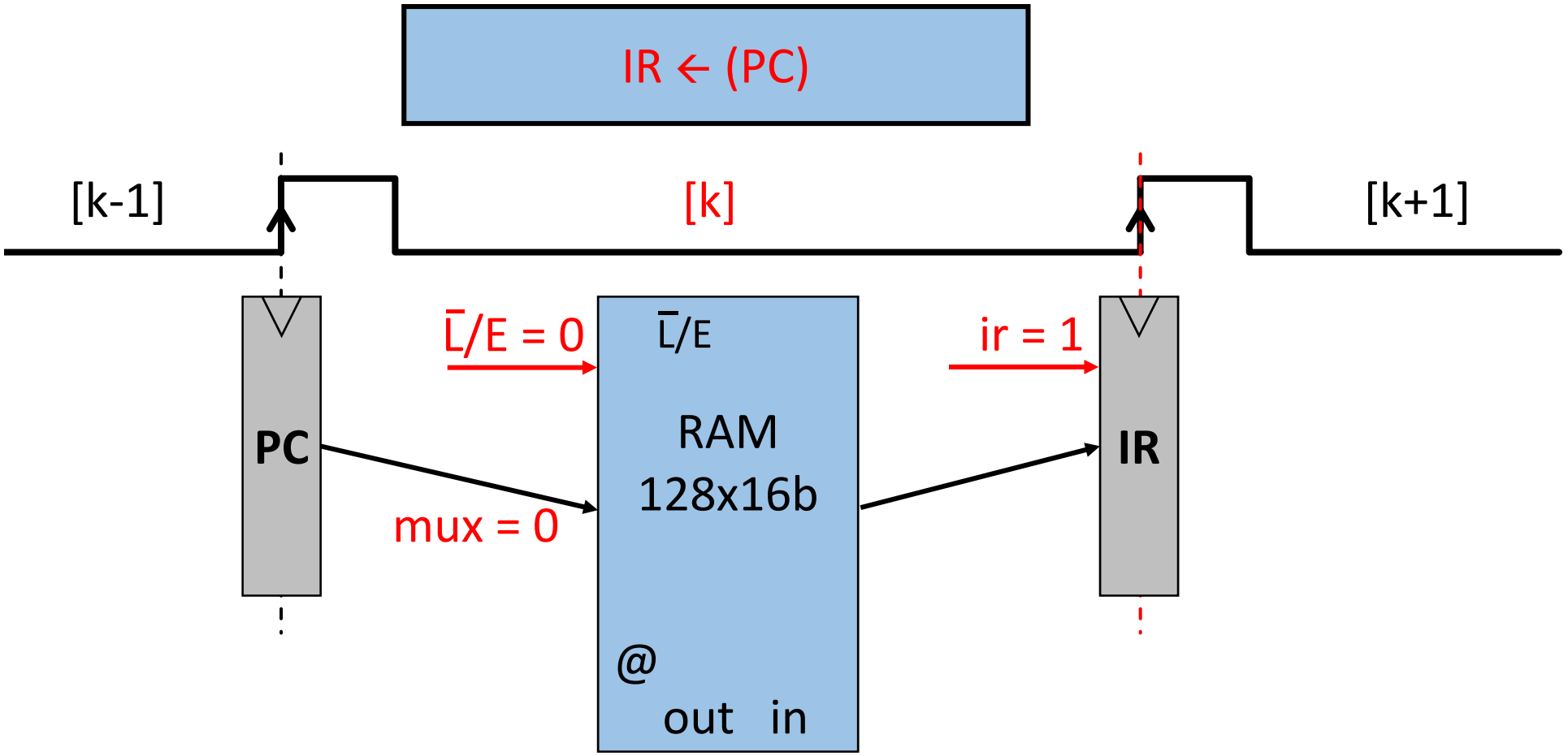
Fase: 1 o 2 ciclos

Fases y ciclo de procesador



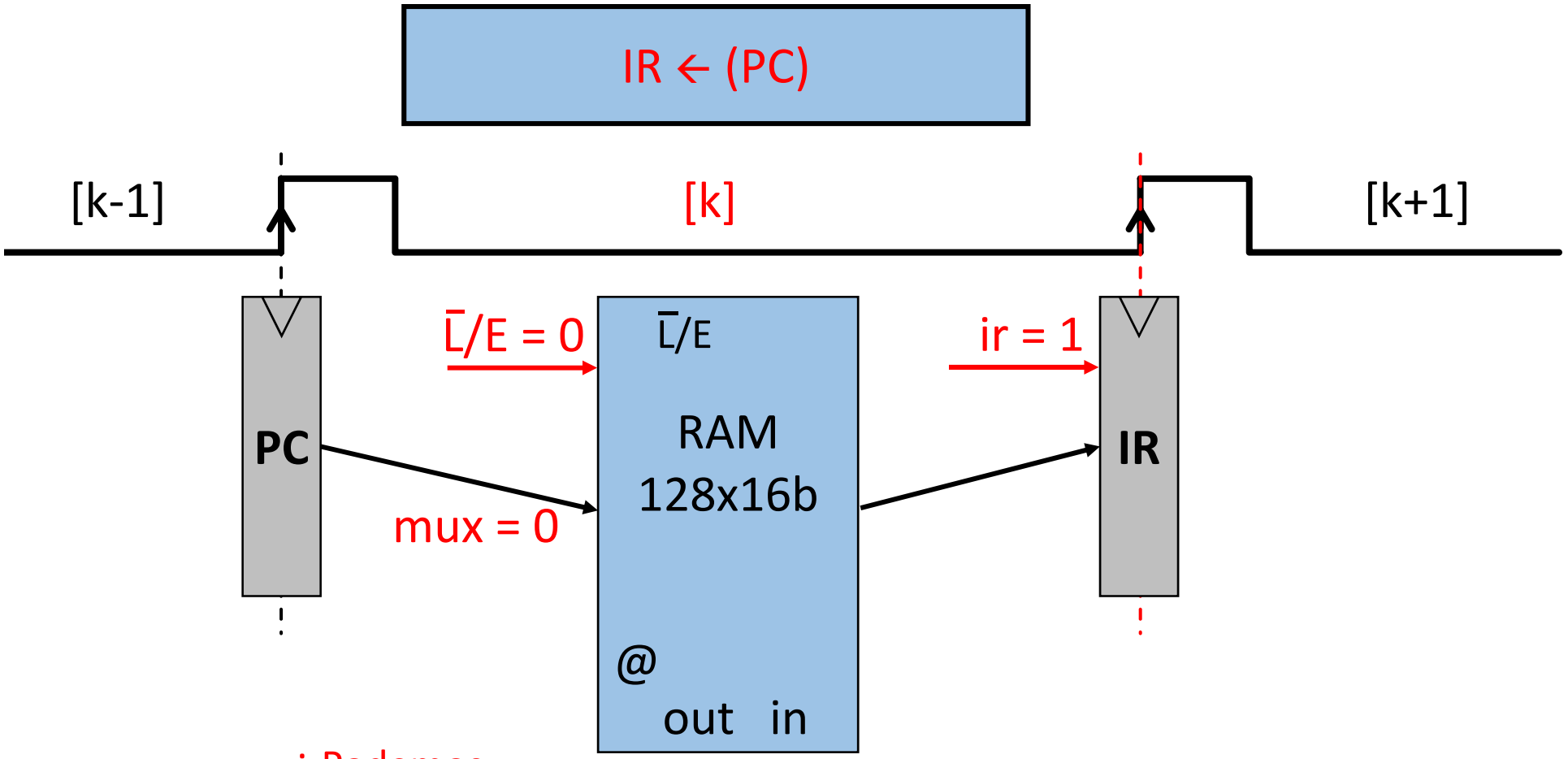
Fase: 1 o 2 ciclos

Fases y ciclo de procesador



Fase: 1 o 2 ciclos

Fases y ciclo de procesador



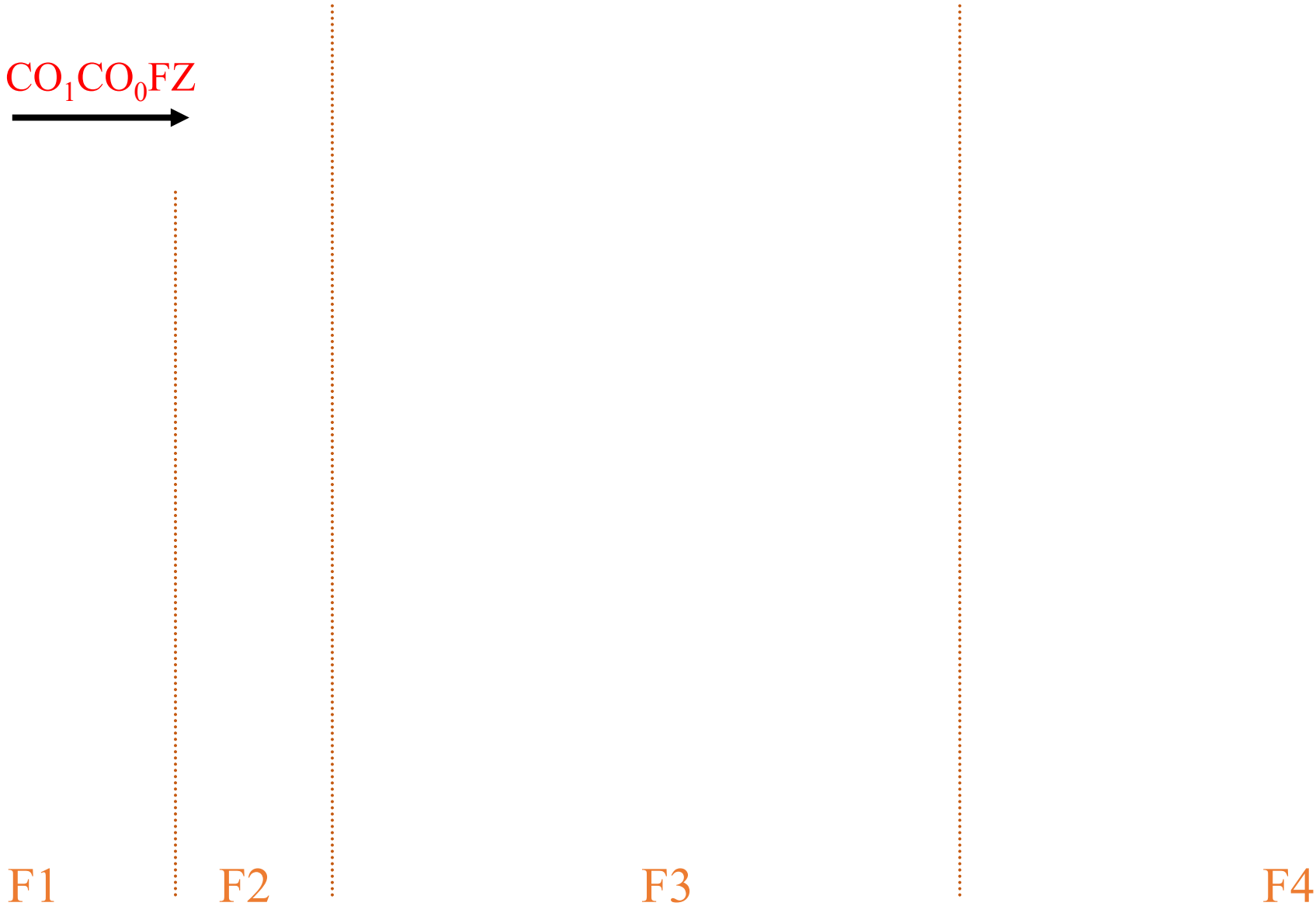
¿ Podemos
incrementar
PC en el mismo ciclo ?

Fase: 1 o 2 ciclos

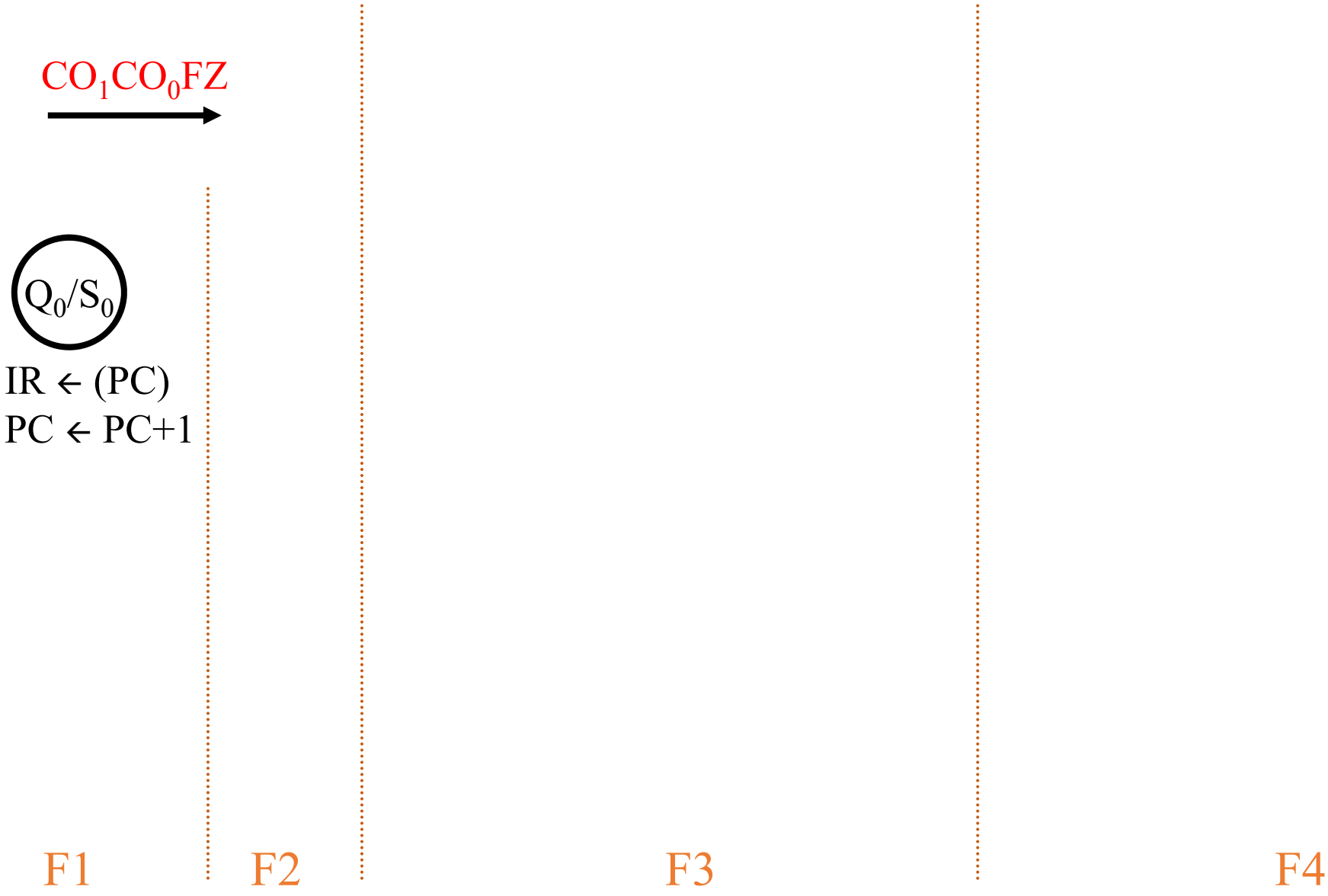
Diseño de la Unidad de Control

- U.C. cableada:
  - autómata Moore
  - cada estado → vector de salidas
- U.C. microprogramada:
  - microprograma
  - ejecutar instrucción → secuencia de microinstrucciones

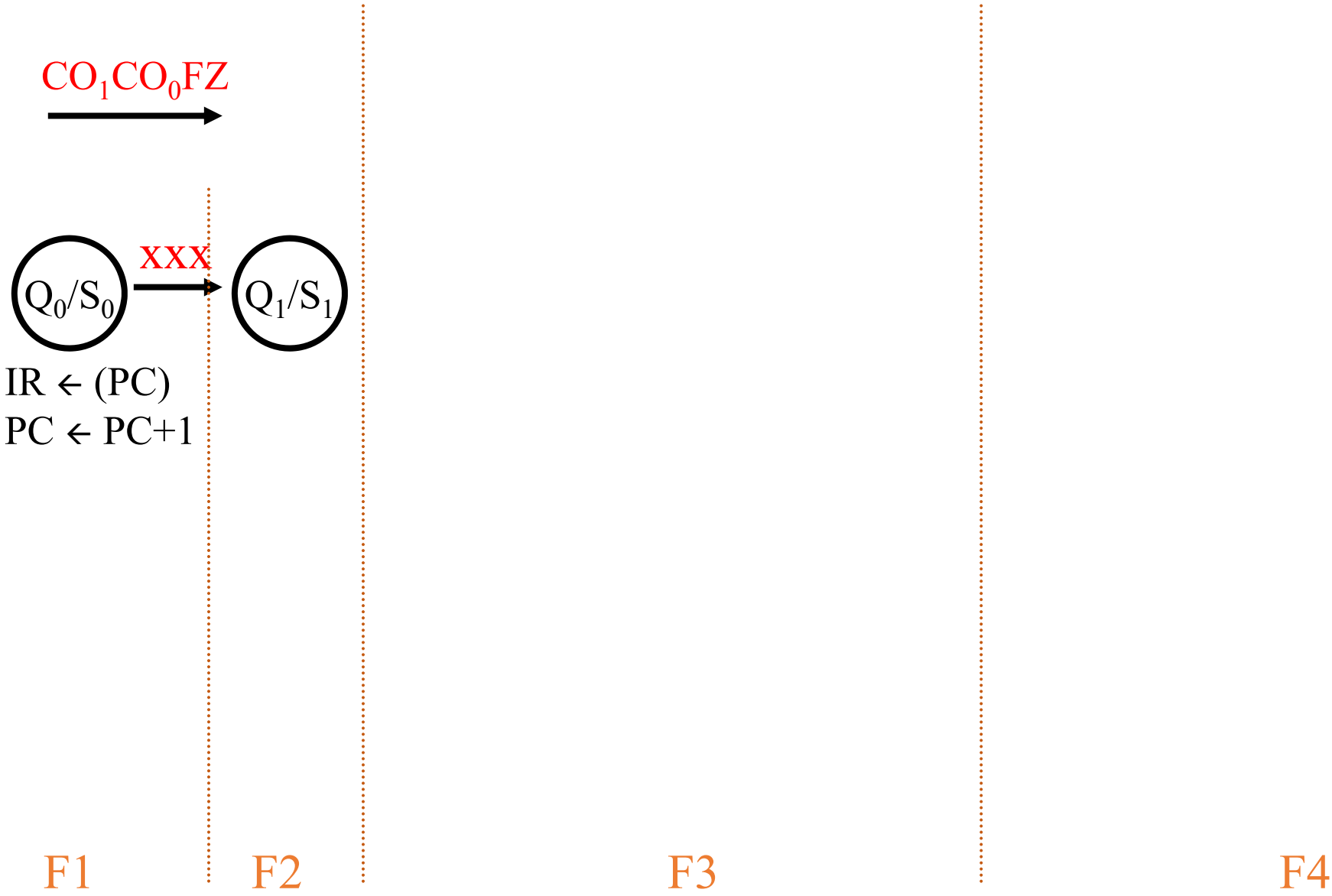
Autómata Moore de la U.C.



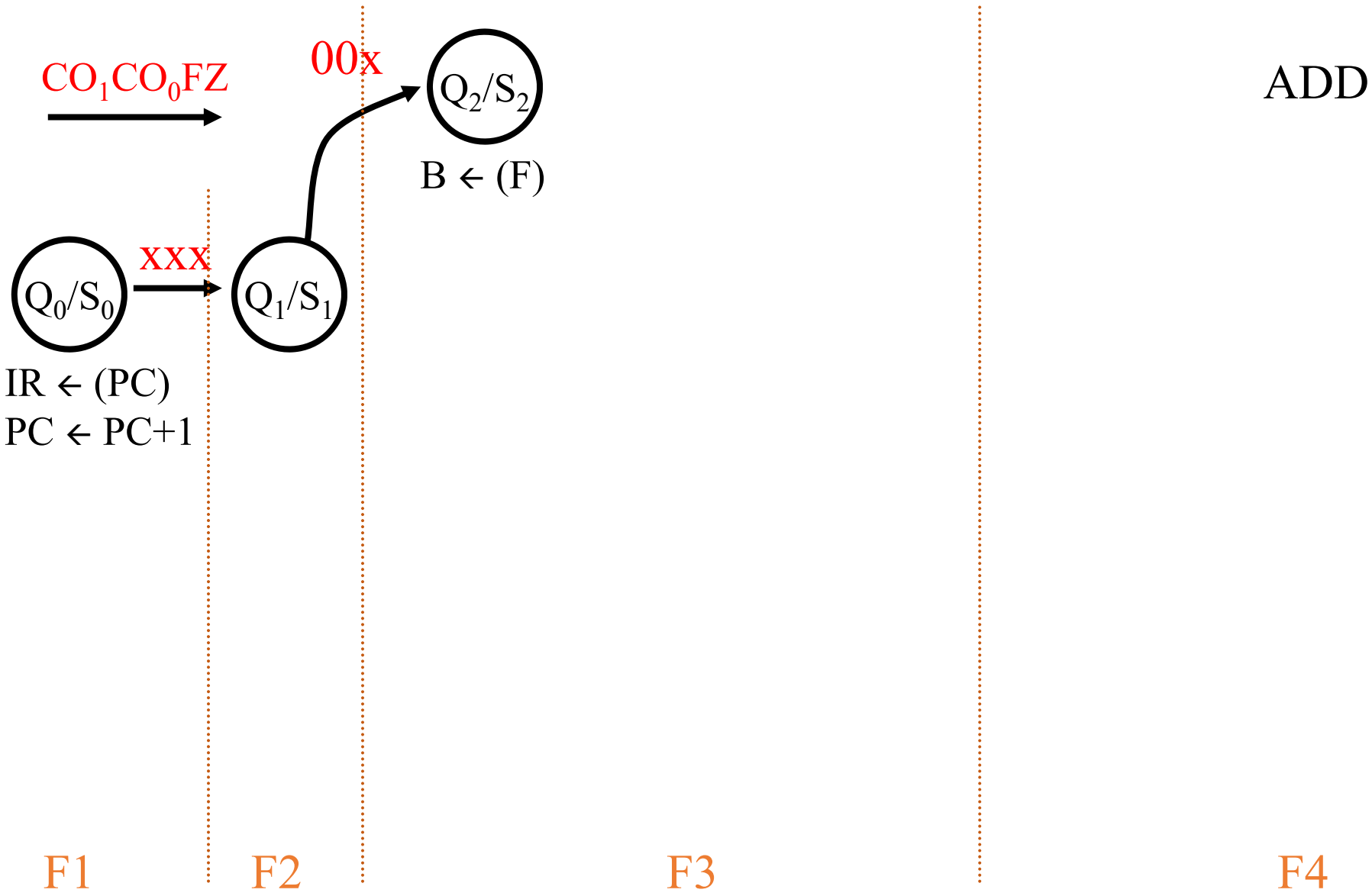
Autómata Moore de la U.C.



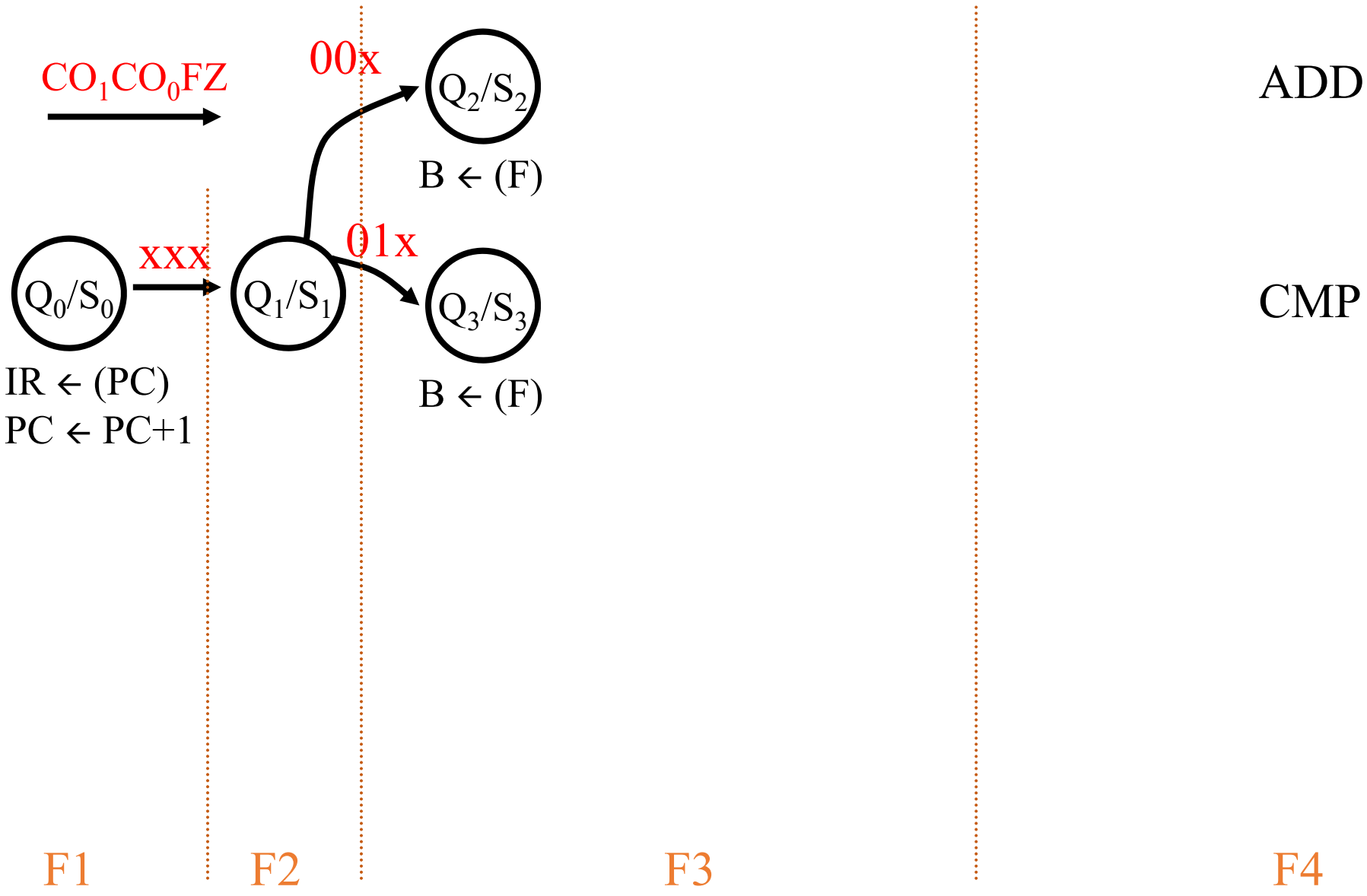
Autómata Moore de la U.C.



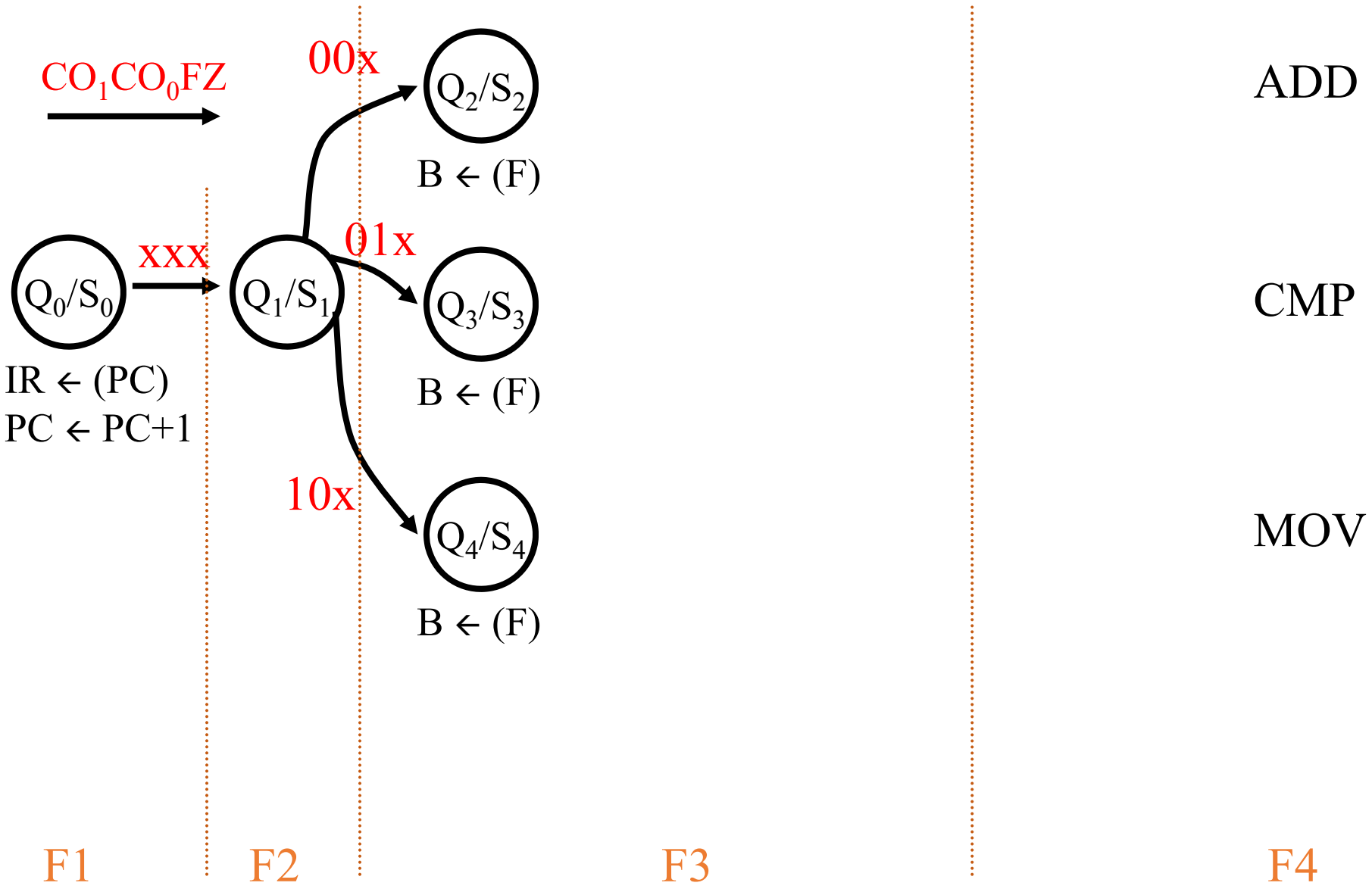
Autómata Moore de la U.C.



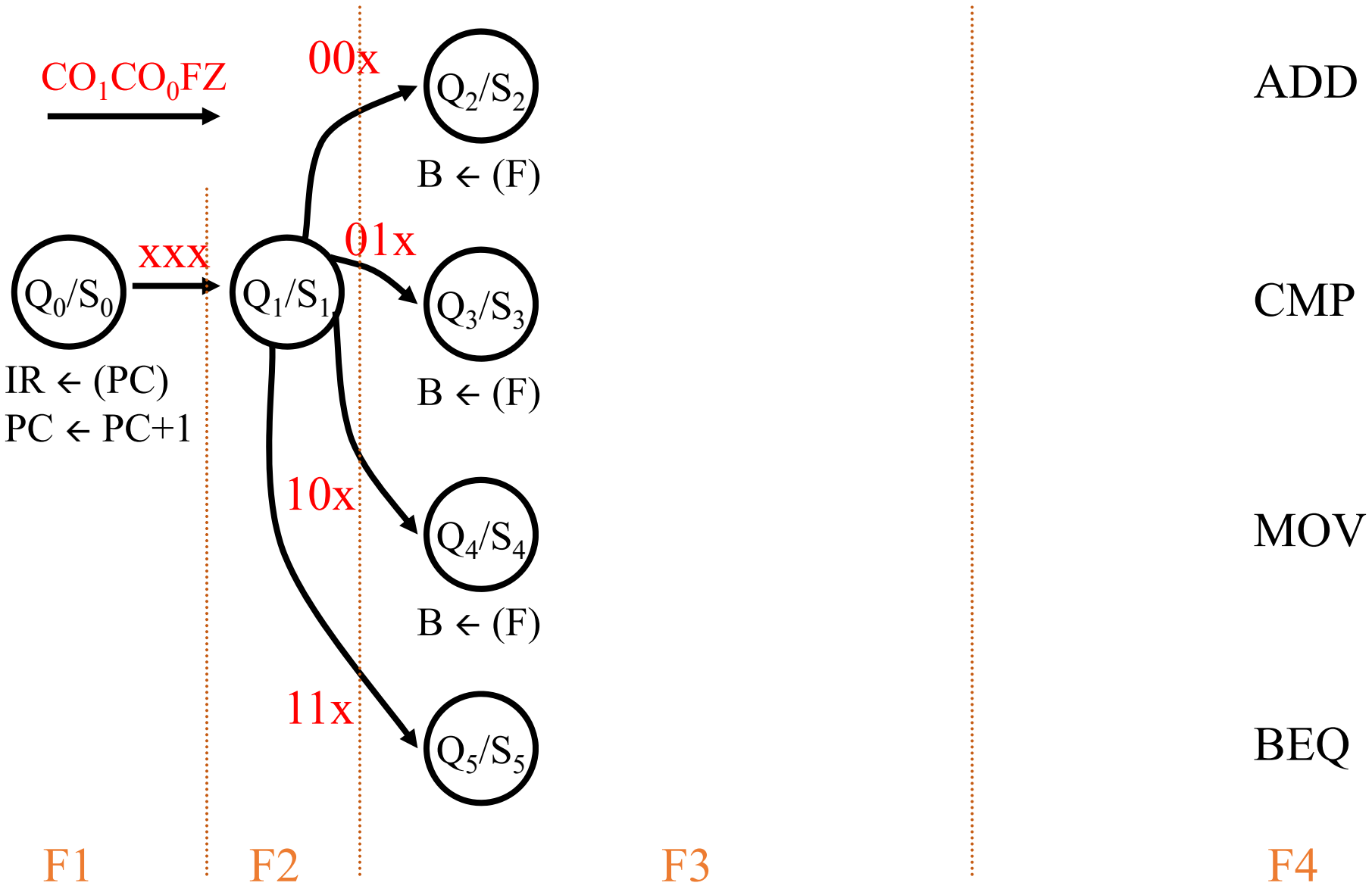
Autómata Moore de la U.C.



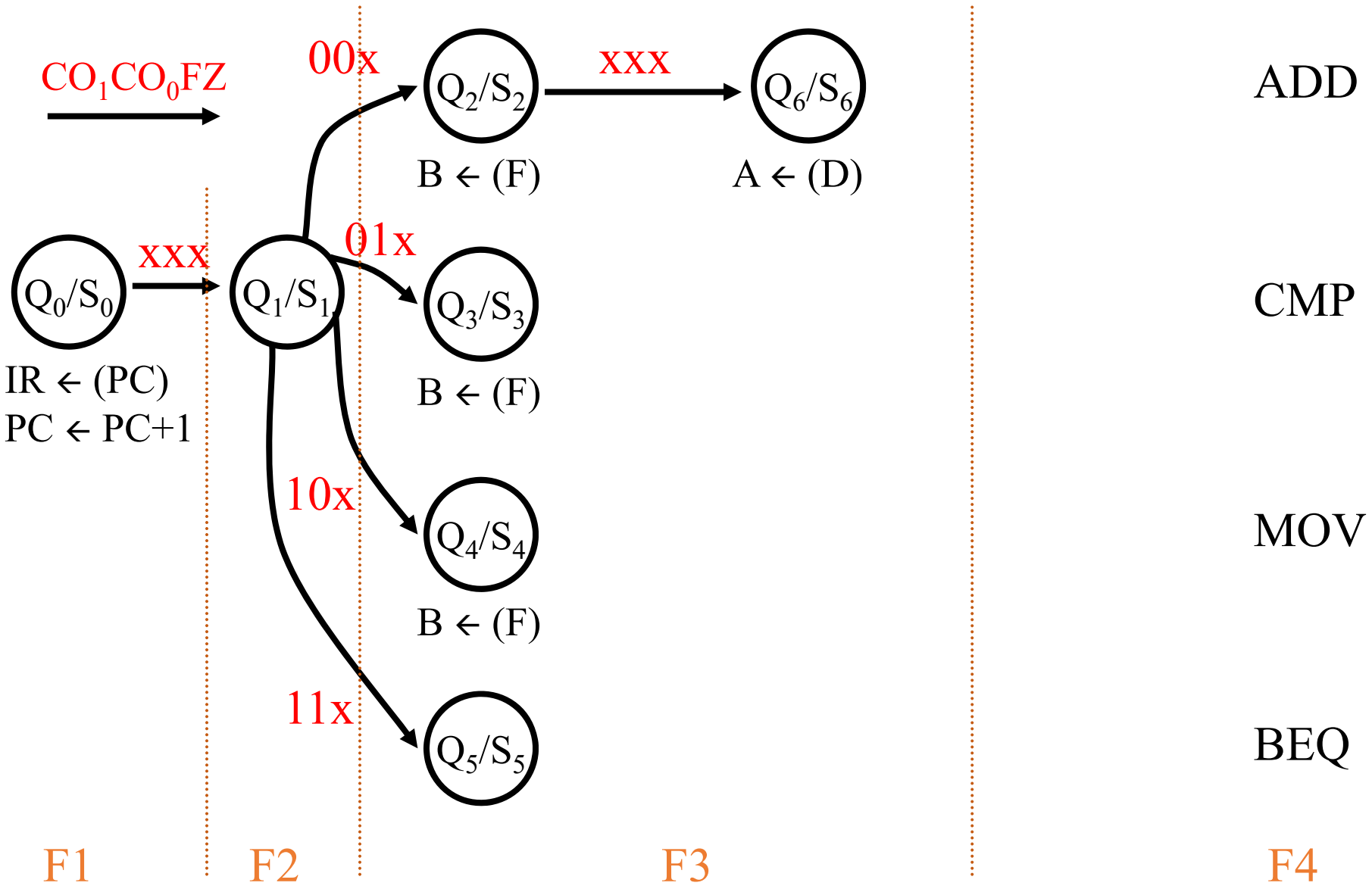
Autómata Moore de la U.C.



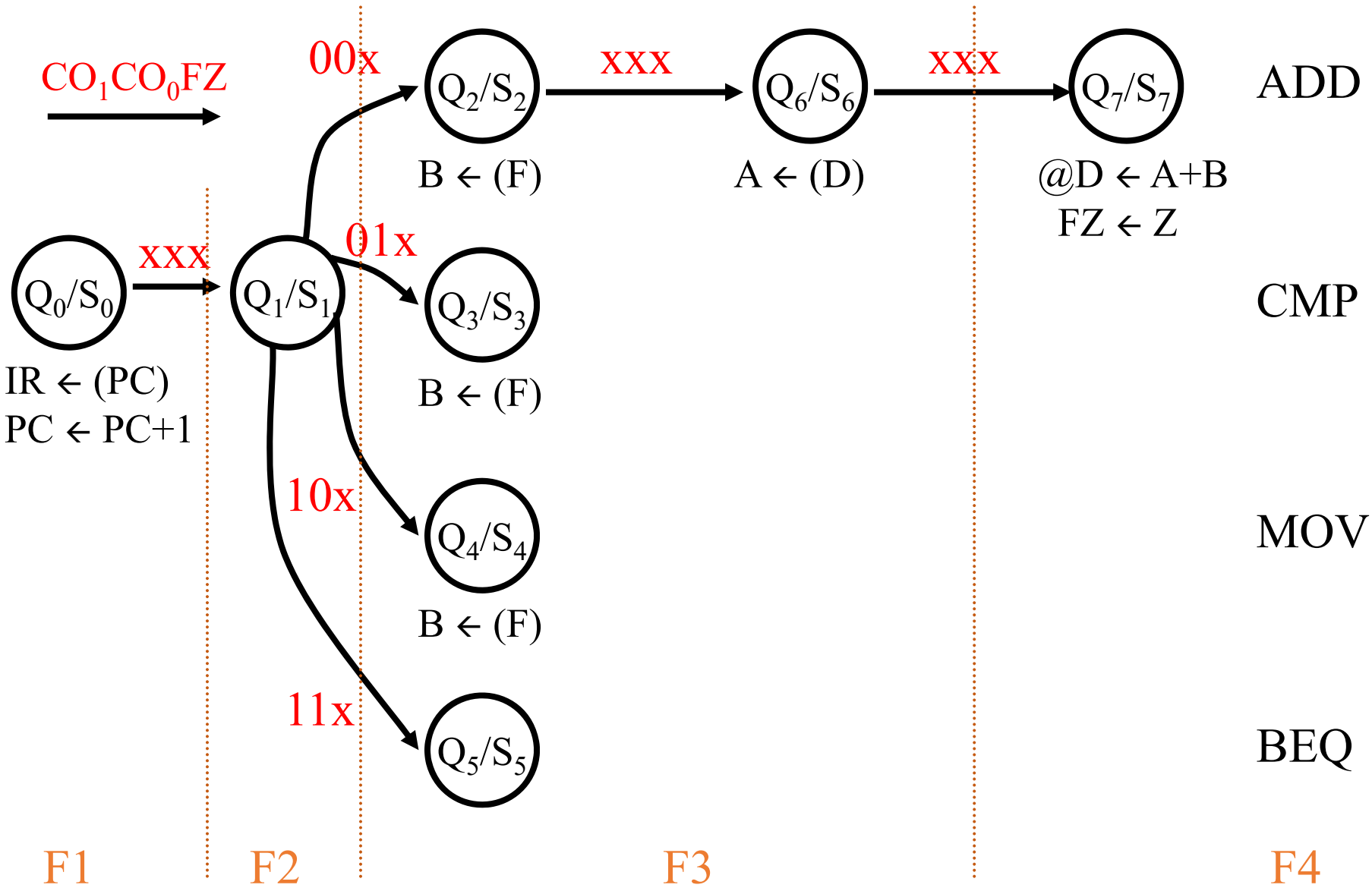
Autómata Moore de la U.C.



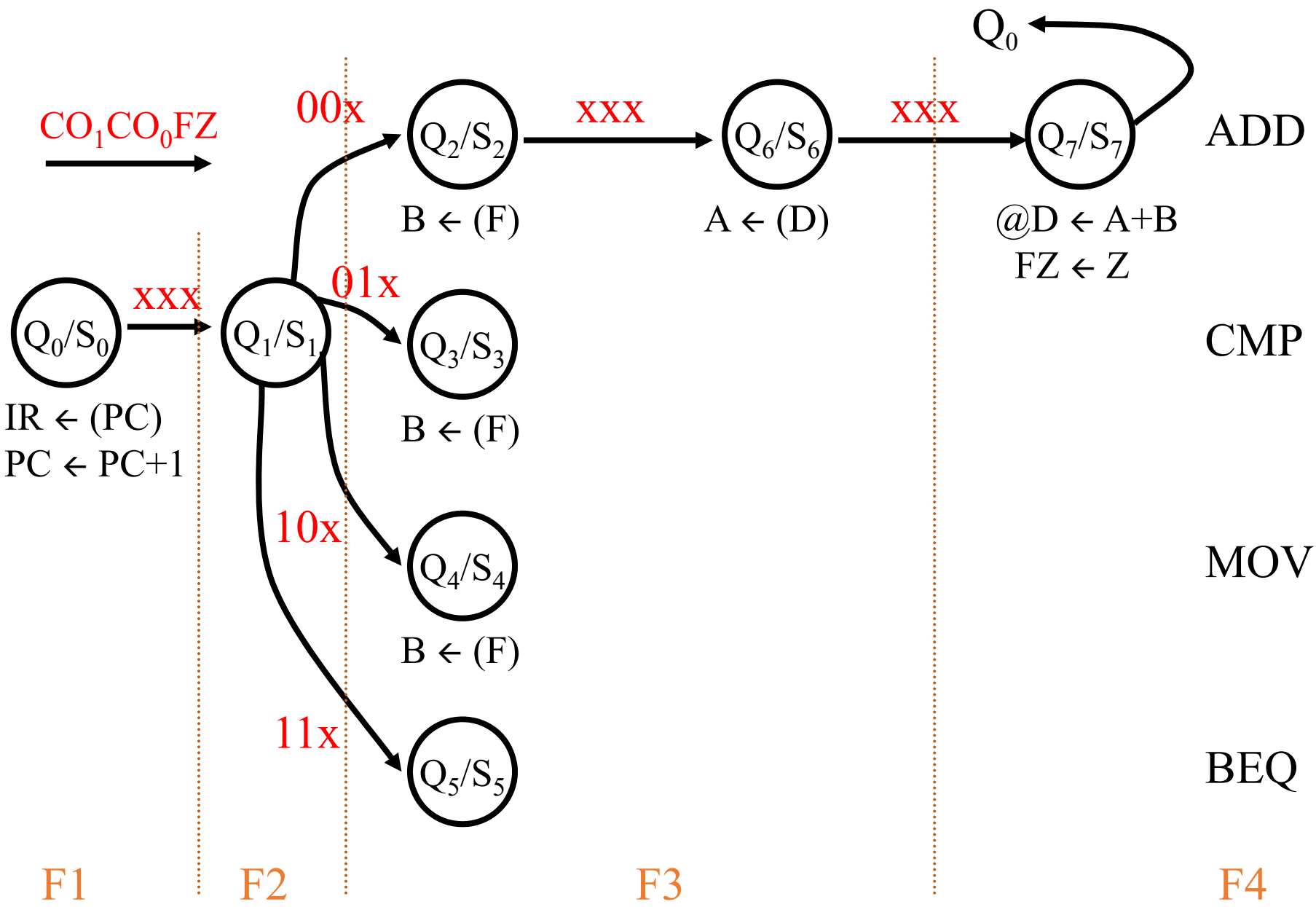
Autómata Moore de la U.C.



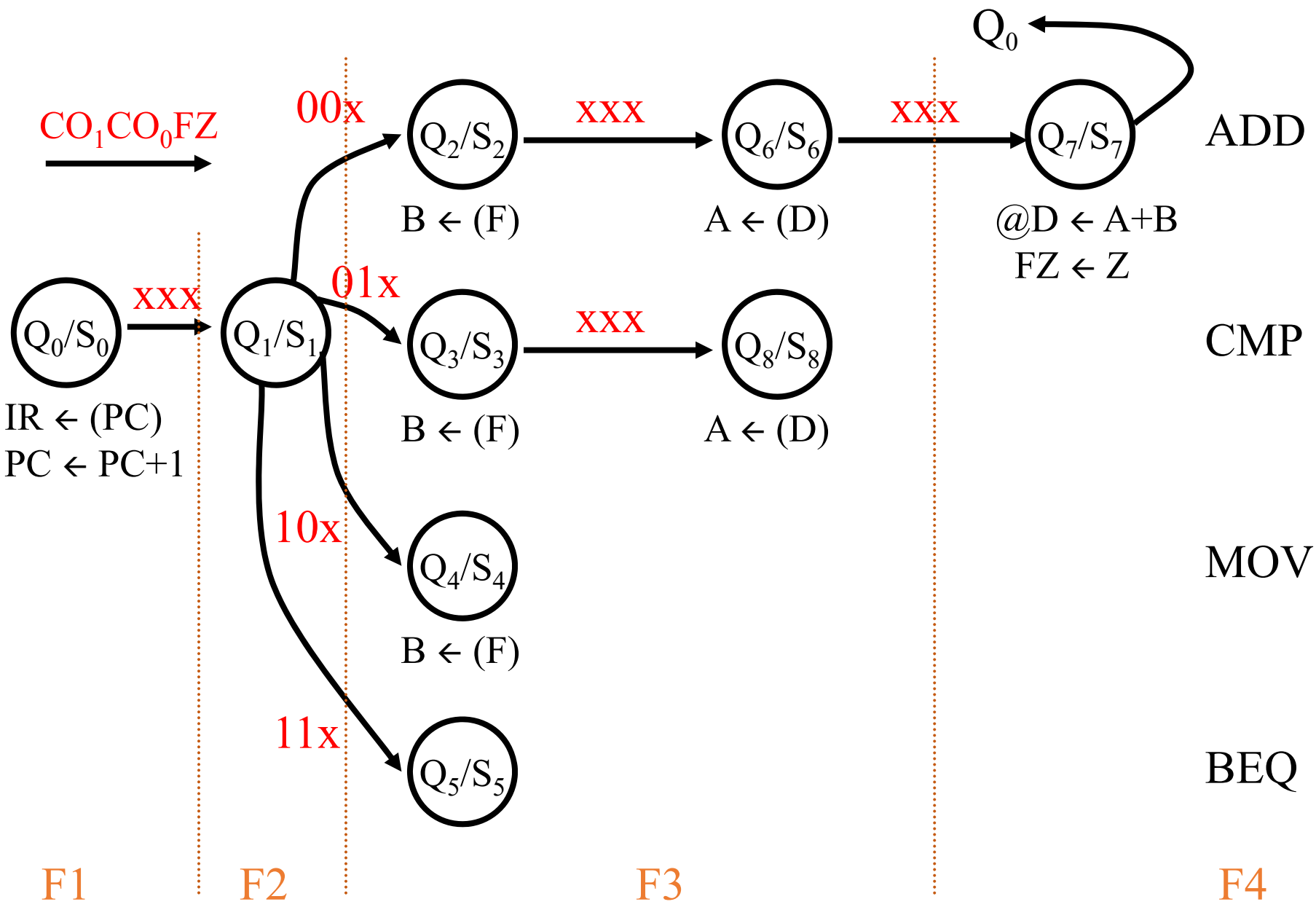
Autómata Moore de la U.C.



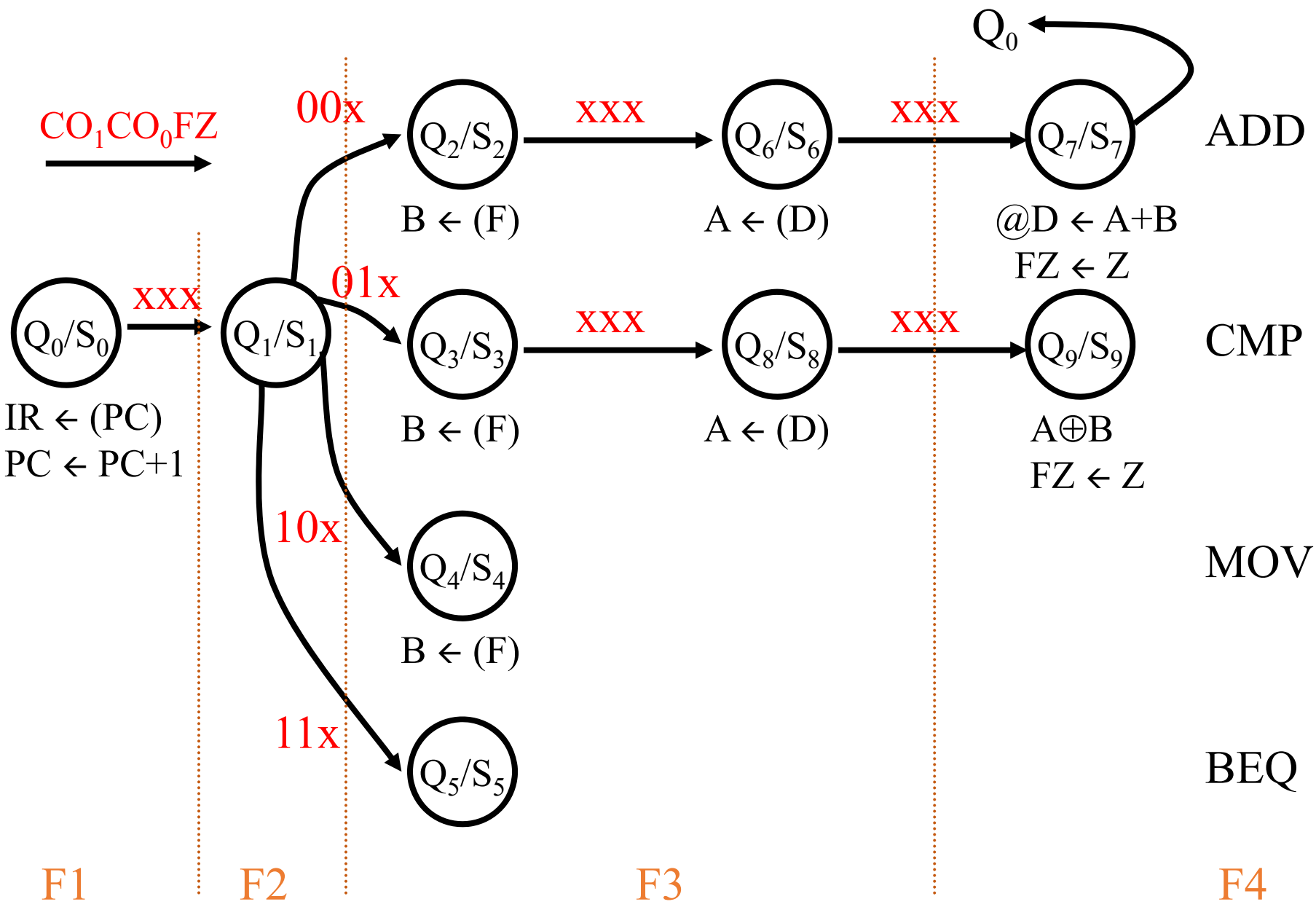
Autómata Moore de la U.C.



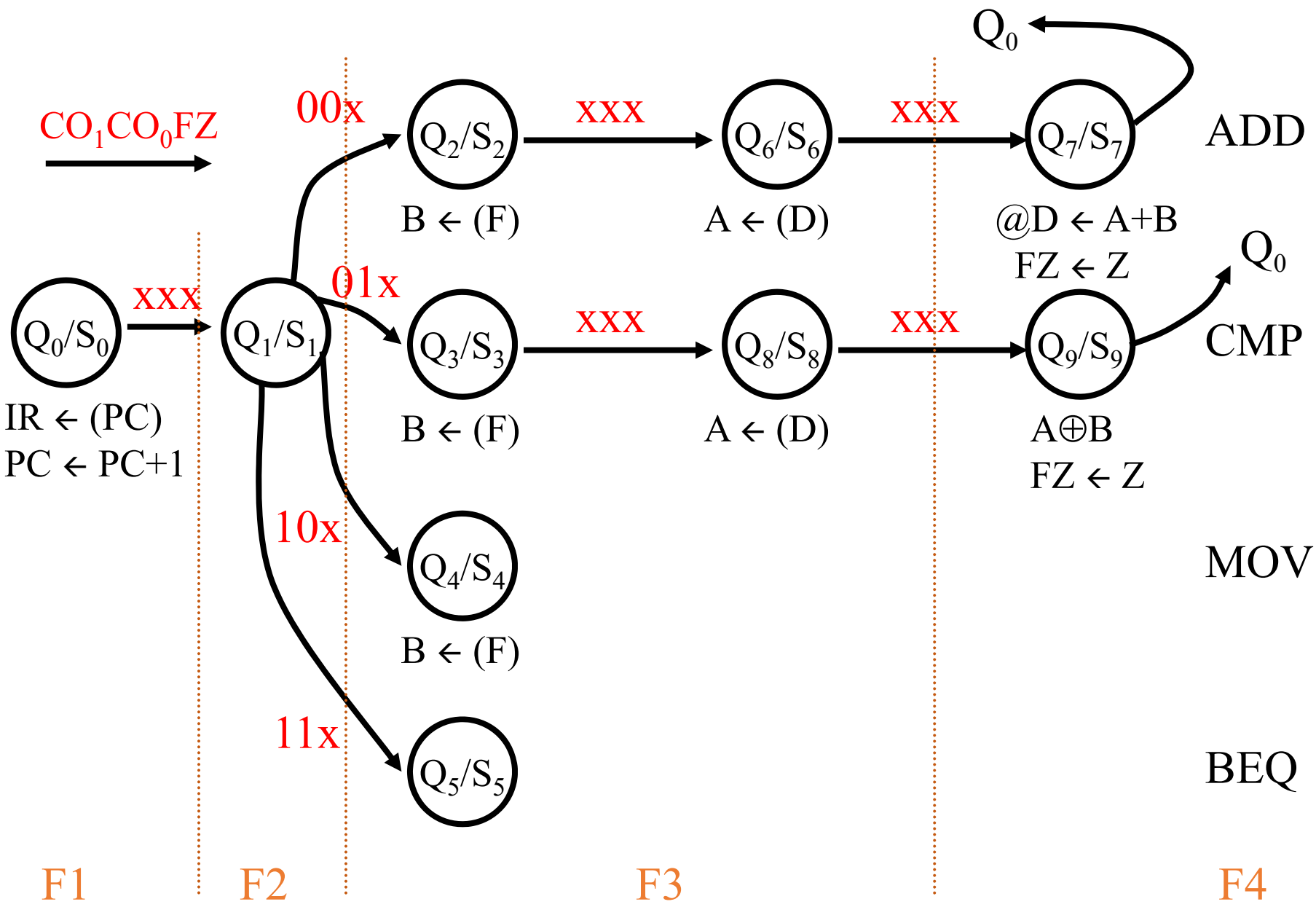
Autómata Moore de la U.C.



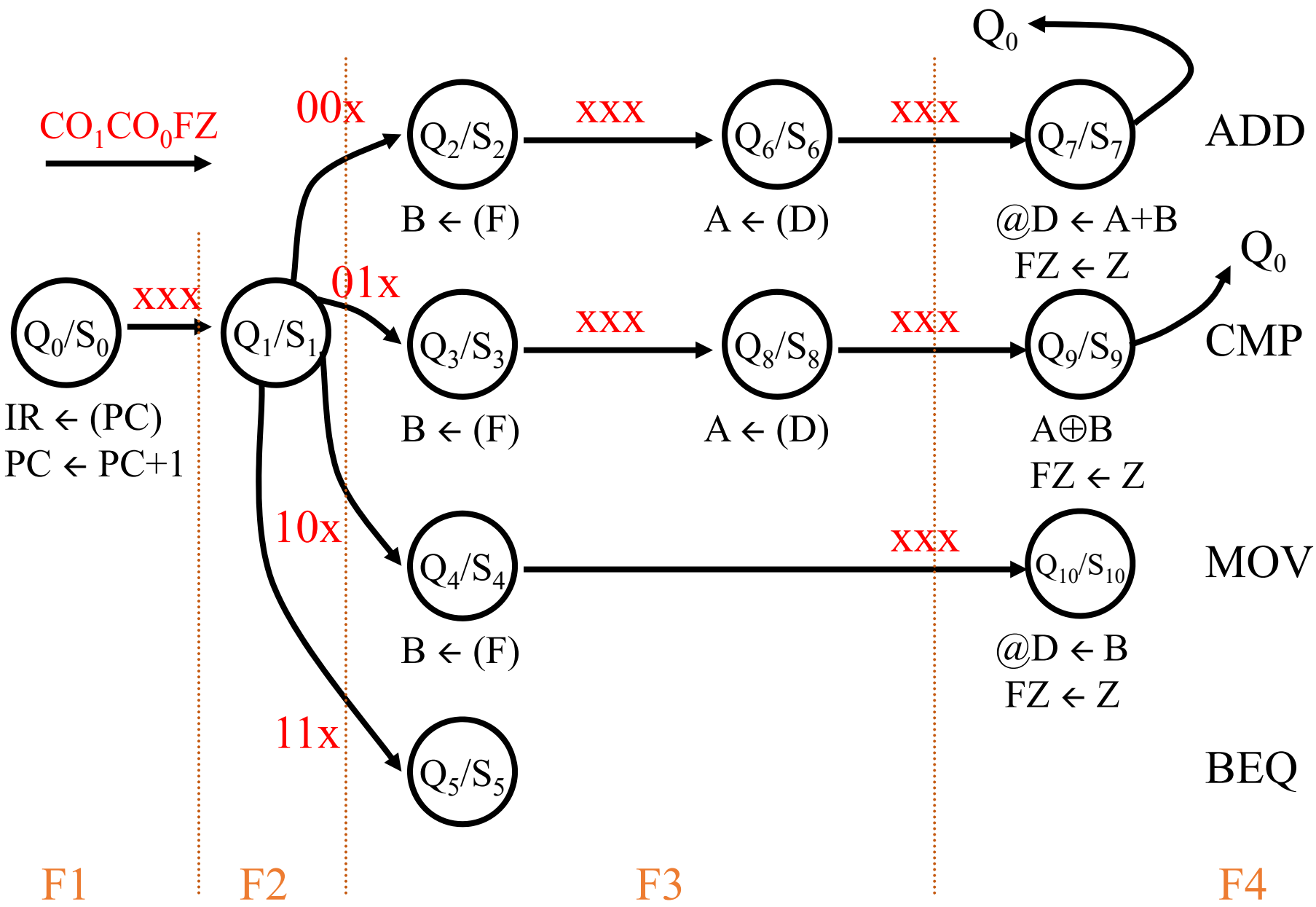
Autómata Moore de la U.C.



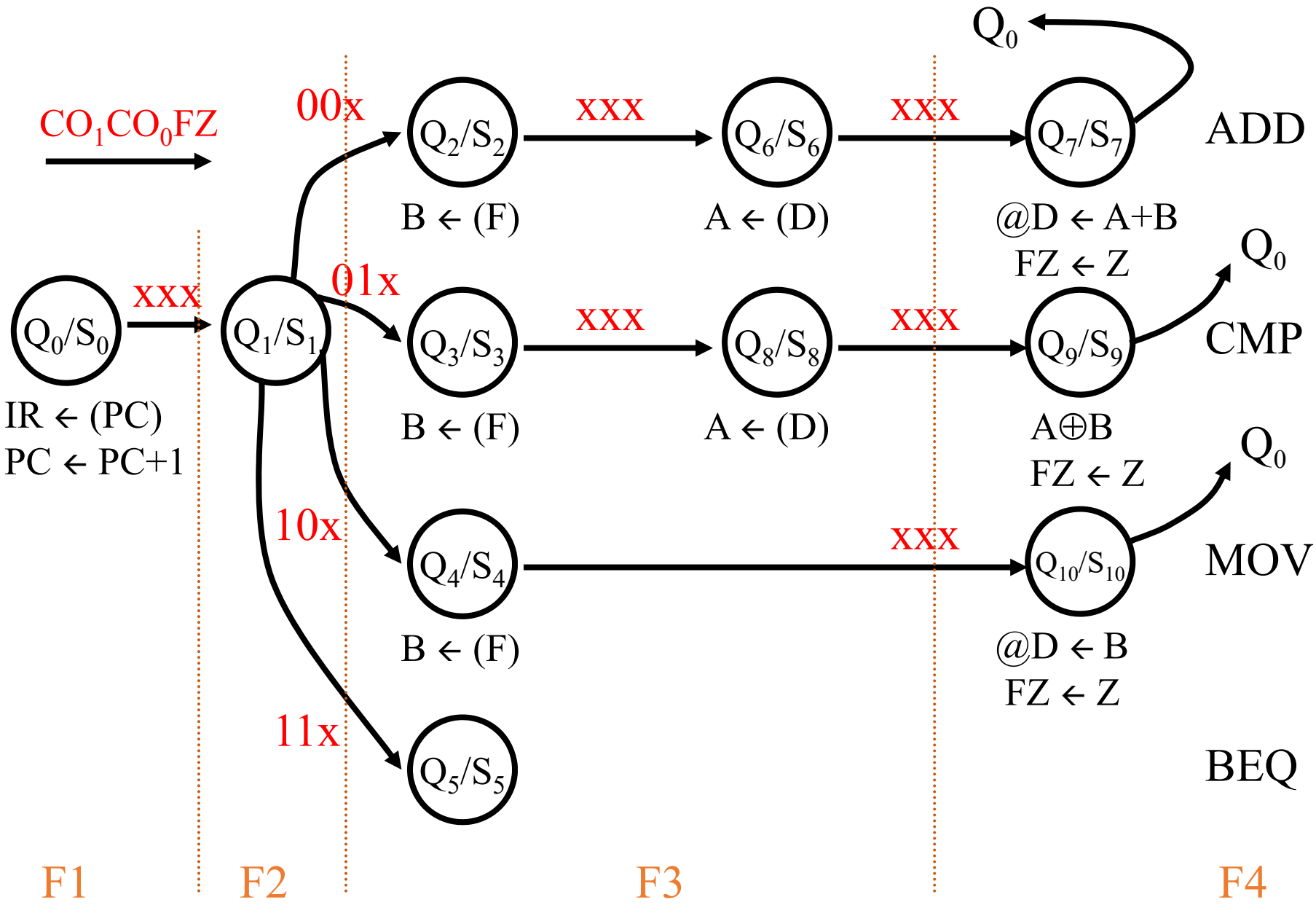
Autómata Moore de la U.C.



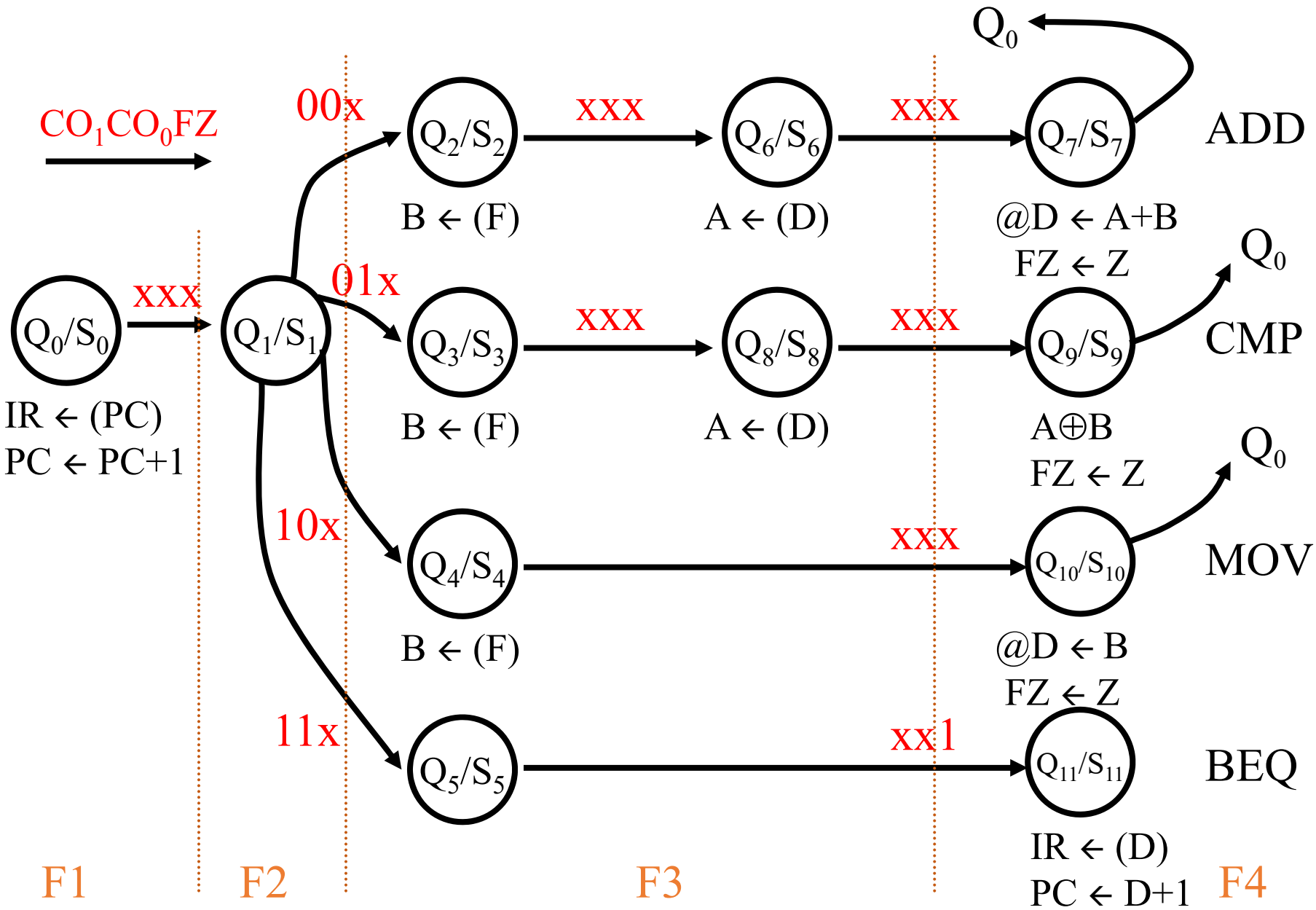
Autómata Moore de la U.C.



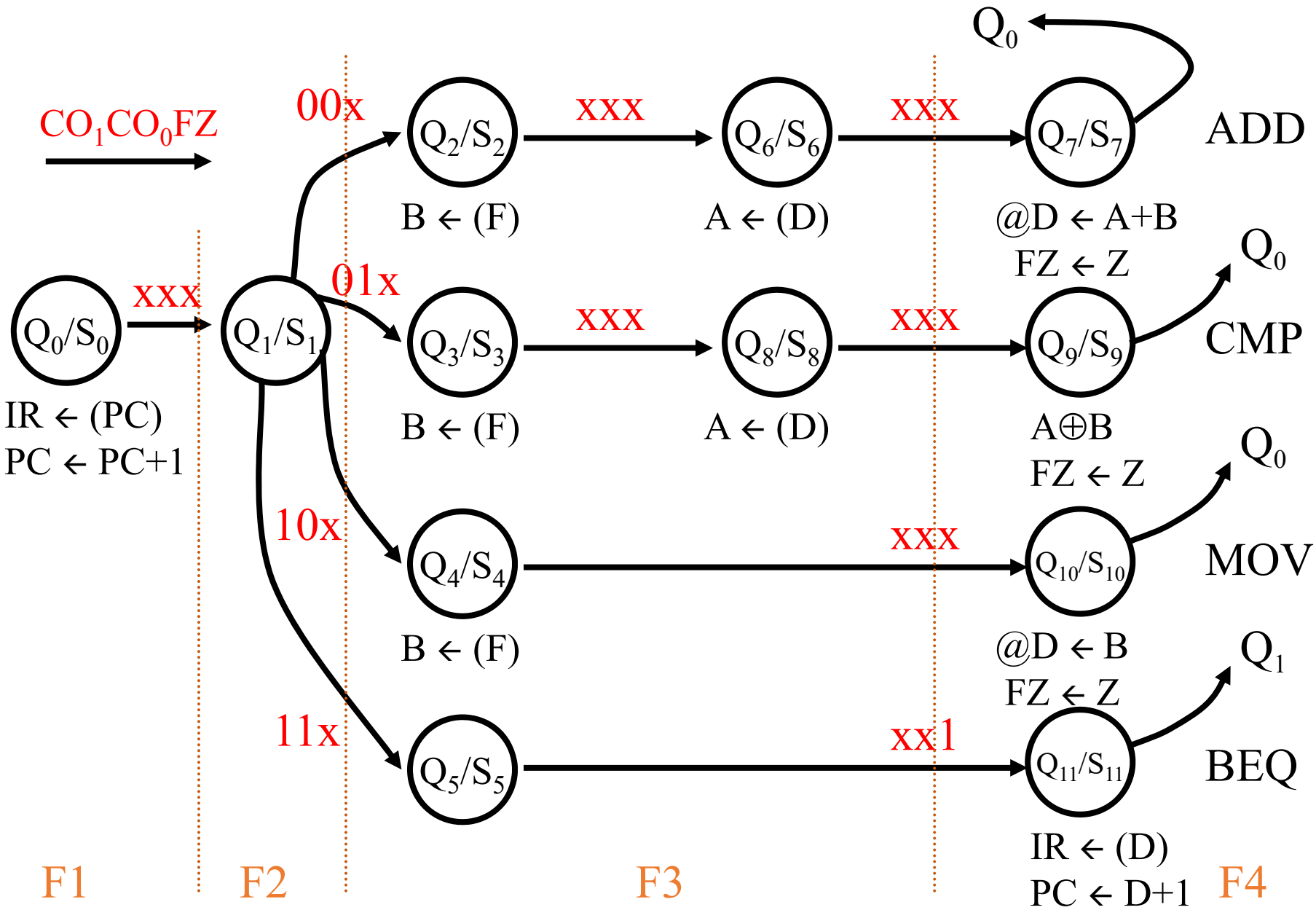
Autómata Moore de la U.C.



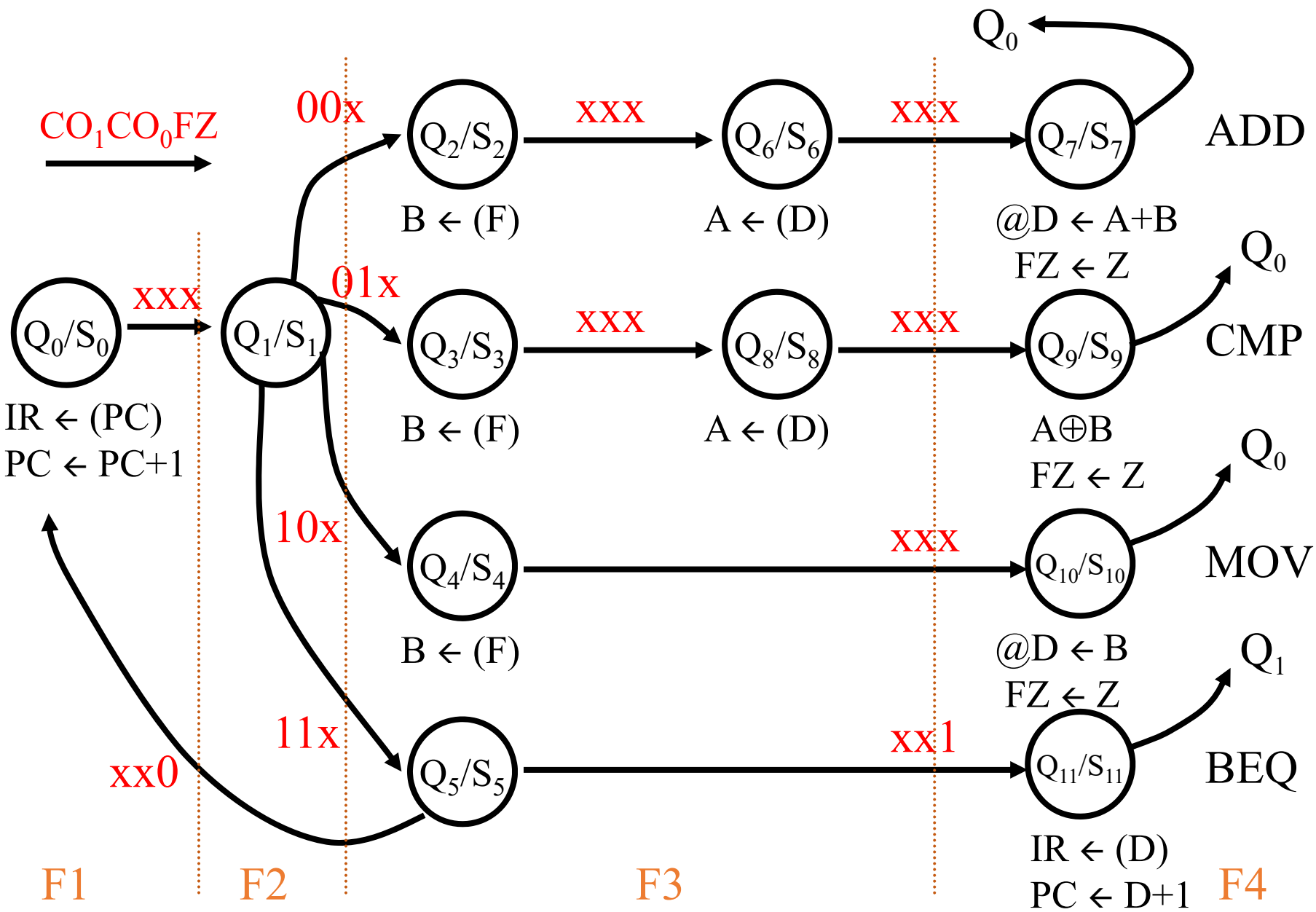
Autómata Moore de la U.C.



Autómata Moore de la U.C.



Autómata Moore de la U.C.



	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
mx_1	0	X	1	1	1	X	1	1	1	X	1	1
mx_0	0	X	0	0	0	X	1	1	1	X	1	1
alu_1	X	X	X	X	X	X	X	0	X	0	1	X
alu_0	X	X	X	X	X	X	X	0	X	1	0	X
$\overline{L}/E$	0	0	0	0	0	0	0	1	0	0	1	0
pc	1	0	0	0	0	0	0	0	0	0	0	1
ir	1	0	0	0	0	0	0	0	0	0	0	1
a	0	0	0	0	0	0	1	0	1	0	0	0
b	0	0	1	1	1	0	0	0	0	0	0	0
fz	0	0	0	0	0	0	0	1	0	1	1	0

S_0
IR ← (PC)
PC ← PC+1

	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
mx_1	0	X	1	1	1	X	1	1	1	X	1	1
mx_0	0	X	0	0	0	X	1	1	1	X	1	1
alu_1	X	X	X	X	X	X	X	0	X	0	1	X
alu_0	X	X	X	X	X	X	X	0	X	1	0	X
$\overline{L}/E$	0	0	0	0	0	0	0	1	0	0	1	0
pc	1	0	0	0	0	0	0	0	0	0	0	1
ir	1	0	0	0	0	0	0	0	0	0	0	1
a	0	0	0	0	0	0	1	0	1	0	0	0
b	0	0	1	1	1	0	0	0	0	0	0	0
fz	0	0	0	0	0	0	0	1	0	1	1	0

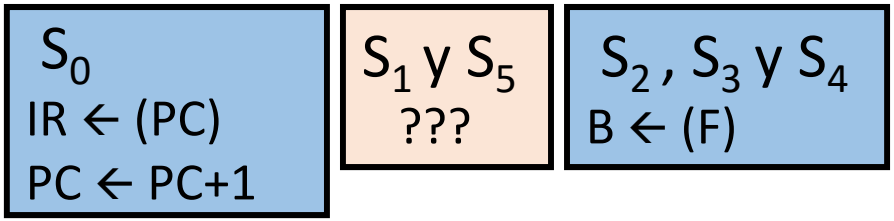
S_0
IR ← (PC)
PC ← PC+1

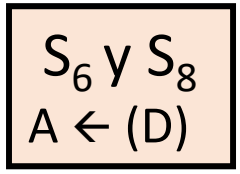
S_1 y S_5
???

	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
mx_1	0	X	1	1	1	X	1	1	1	X	1	1
mx_0	0	X	0	0	0	X	1	1	1	X	1	1
alu_1	X	X	X	X	X	X	X	0	X	0	1	X
alu_0	X	X	X	X	X	X	X	0	X	1	0	X
$\overline{L}/E$	0	0	0	0	0	0	0	1	0	0	1	0
pc	1	0	0	0	0	0	0	0	0	0	0	1
ir	1	0	0	0	0	0	0	0	0	0	0	1
a	0	0	0	0	0	0	1	0	1	0	0	0
b	0	0	1	1	1	0	0	0	0	0	0	0
fz	0	0	0	0	0	0	0	1	0	1	1	0

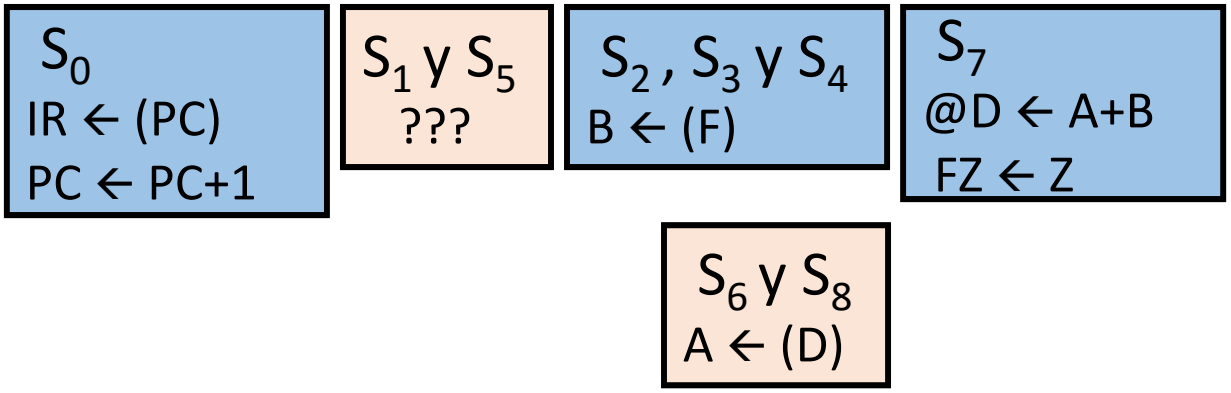
S_0	S_1 y S_5	S_2 , S_3 y S_4
IR ← (PC) PC ← PC+1	???	B ← (F)

	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
mx_1	0	X	1	1	1	X	1	1	1	X	1	1
mx_0	0	X	0	0	0	X	1	1	1	X	1	1
alu_1	X	X	X	X	X	X	X	0	X	0	1	X
alu_0	X	X	X	X	X	X	X	0	X	1	0	X
$\overline{L}/E$	0	0	0	0	0	0	0	1	0	0	1	0
pc	1	0	0	0	0	0	0	0	0	0	0	1
ir	1	0	0	0	0	0	0	0	0	0	0	1
a	0	0	0	0	0	0	1	0	1	0	0	0
b	0	0	1	1	1	0	0	0	0	0	0	0
fz	0	0	0	0	0	0	0	1	0	1	1	0

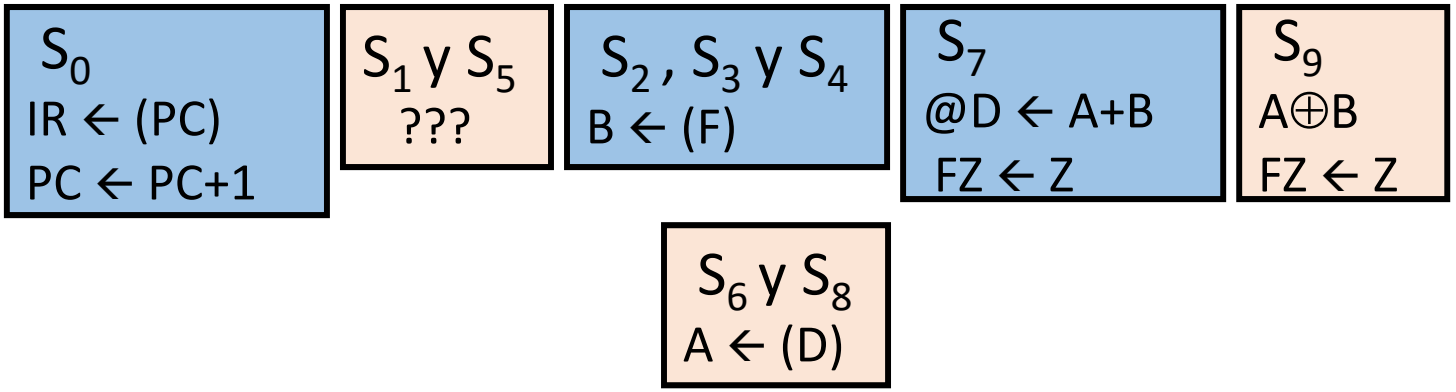




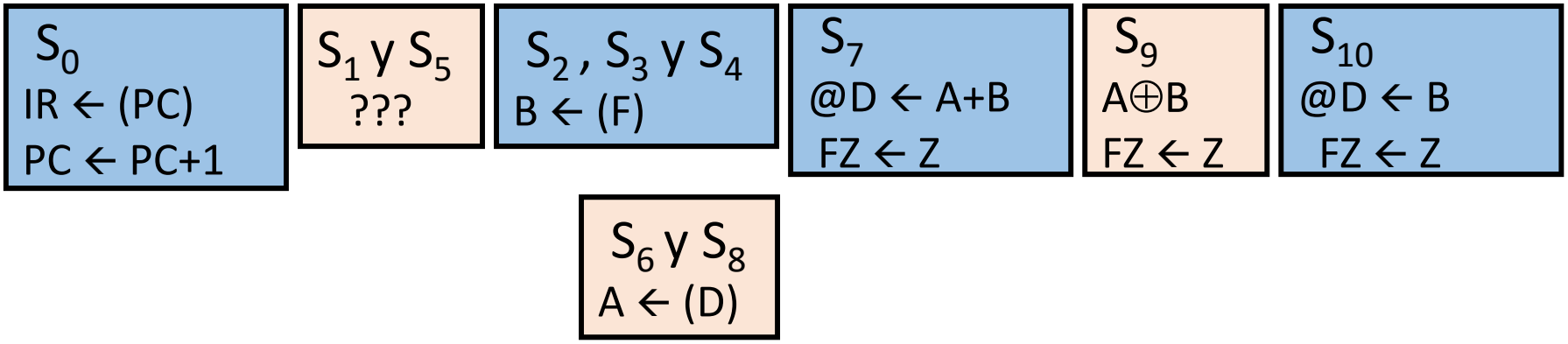
	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
mx_1	0	X	1	1	1	X	1	1	1	X	1	1
mx_0	0	X	0	0	0	X	1	1	1	X	1	1
alu_1	X	X	X	X	X	X	X	0	X	0	1	X
alu_0	X	X	X	X	X	X	X	0	X	1	0	X
$\overline{L/E}$	0	0	0	0	0	0	0	1	0	0	1	0
pc	1	0	0	0	0	0	0	0	0	0	0	1
ir	1	0	0	0	0	0	0	0	0	0	0	1
a	0	0	0	0	0	0	1	0	1	0	0	0
b	0	0	1	1	1	0	0	0	0	0	0	0
fz	0	0	0	0	0	0	0	1	0	1	1	0



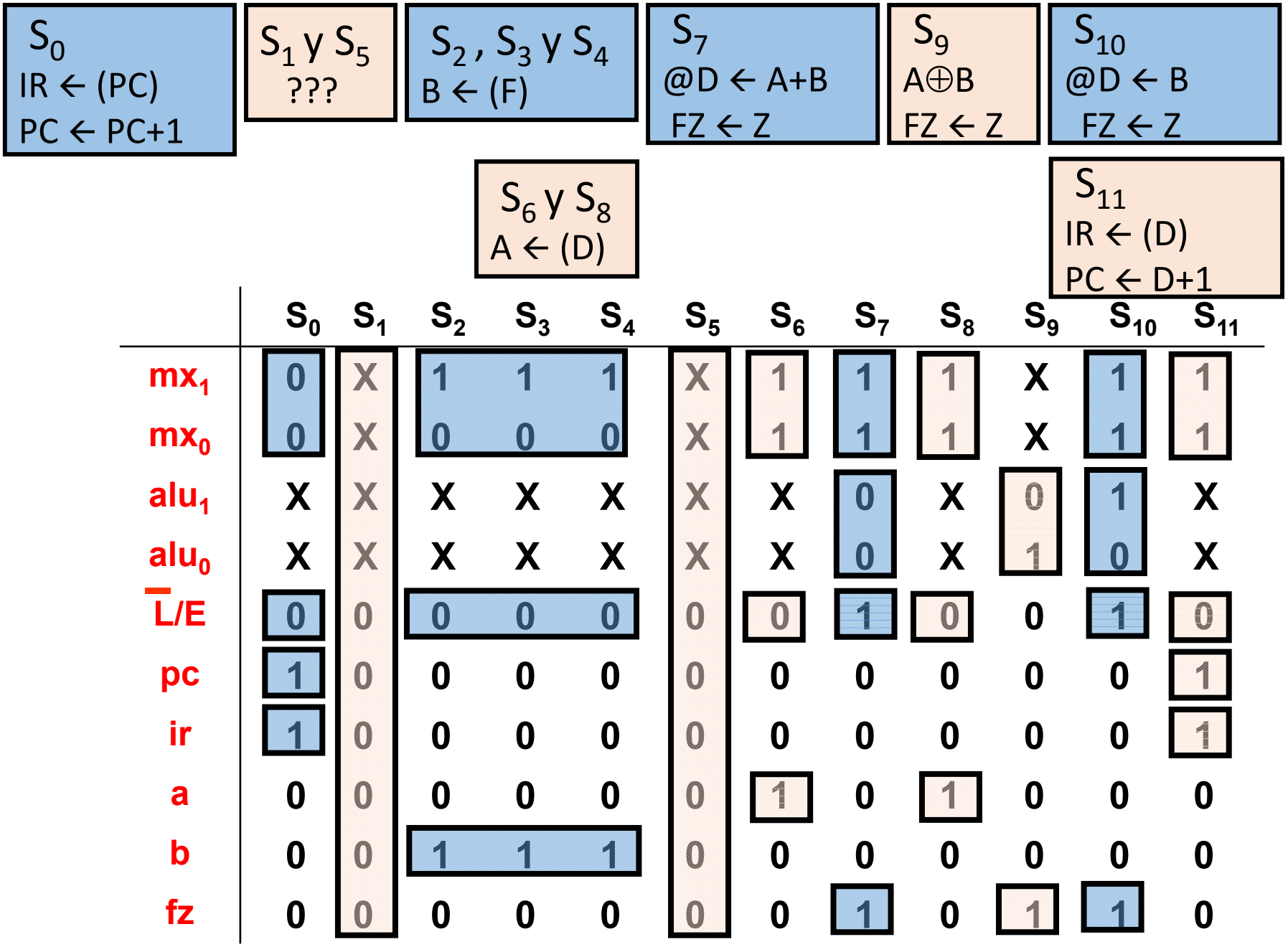
	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
mx_1	0	X	1	1	1	X	1	1	1	X	1	1
mx_0	0	X	0	0	0	X	1	1	1	X	1	1
alu_1	X	X	X	X	X	X	X	0	X	0	1	X
alu_0	X	X	X	X	X	X	X	0	X	1	0	X
$\overline{L/E}$	0	0	0	0	0	0	0	1	0	0	1	0
pc	1	0	0	0	0	0	0	0	0	0	0	1
ir	1	0	0	0	0	0	0	0	0	0	0	1
a	0	0	0	0	0	0	1	0	1	0	0	0
b	0	0	1	1	1	0	0	0	0	0	0	0
fz	0	0	0	0	0	0	0	1	0	1	1	0



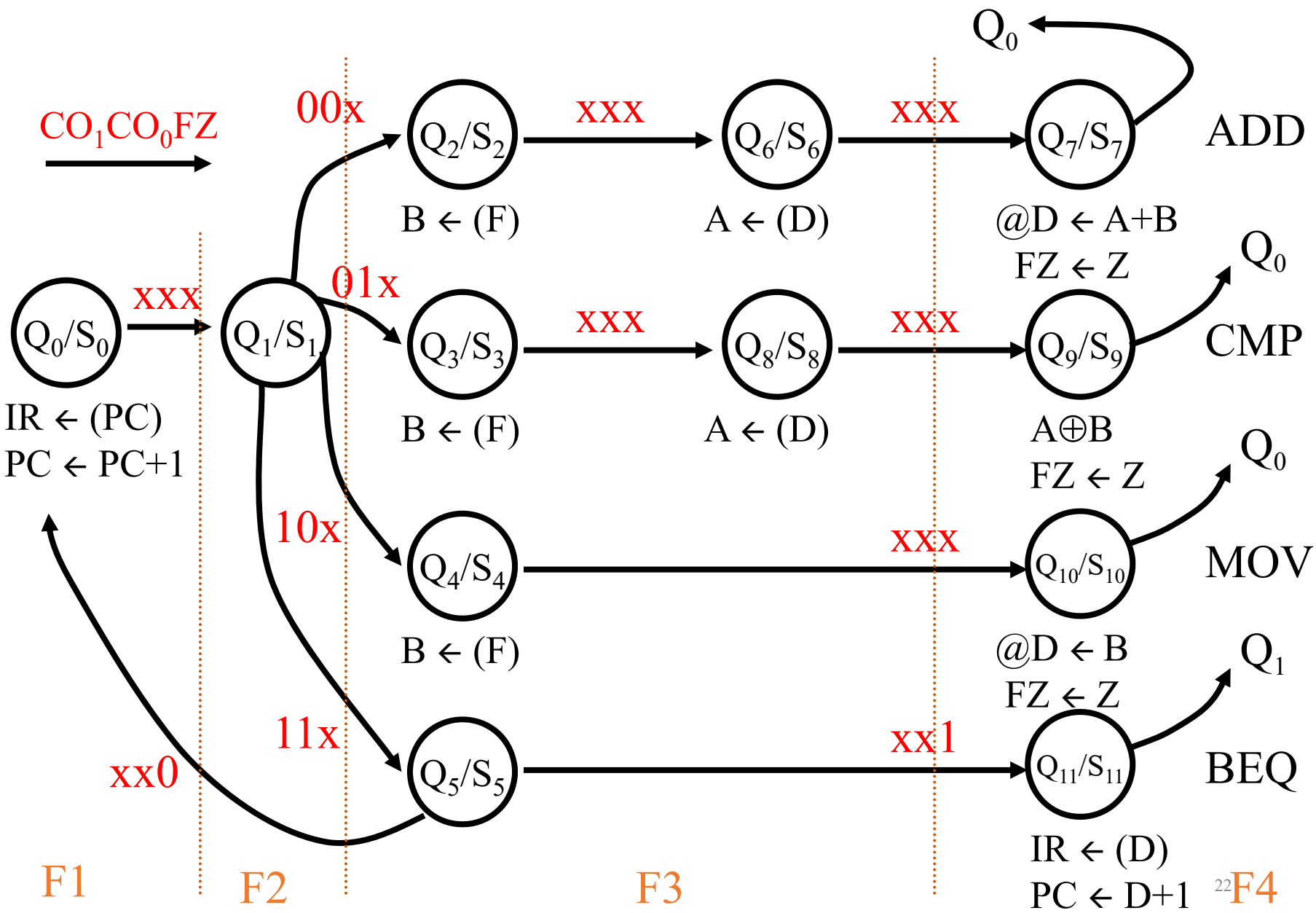
	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
mx_1	0	X	1	1	1	X	1	1	1	X	1	1
mx_0	0	X	0	0	0	X	1	1	1	X	1	1
alu_1	X	X	X	X	X	X	X	0	X	0	1	X
alu_0	X	X	X	X	X	X	X	0	X	1	0	X
$\overline{L}/E$	0	0	0	0	0	0	0	1	0	0	1	0
pc	1	0	0	0	0	0	0	0	0	0	0	1
ir	1	0	0	0	0	0	0	0	0	0	0	1
a	0	0	0	0	0	0	1	0	1	0	0	0
b	0	0	1	1	1	0	0	0	0	0	0	0
fz	0	0	0	0	0	0	0	1	0	1	1	0



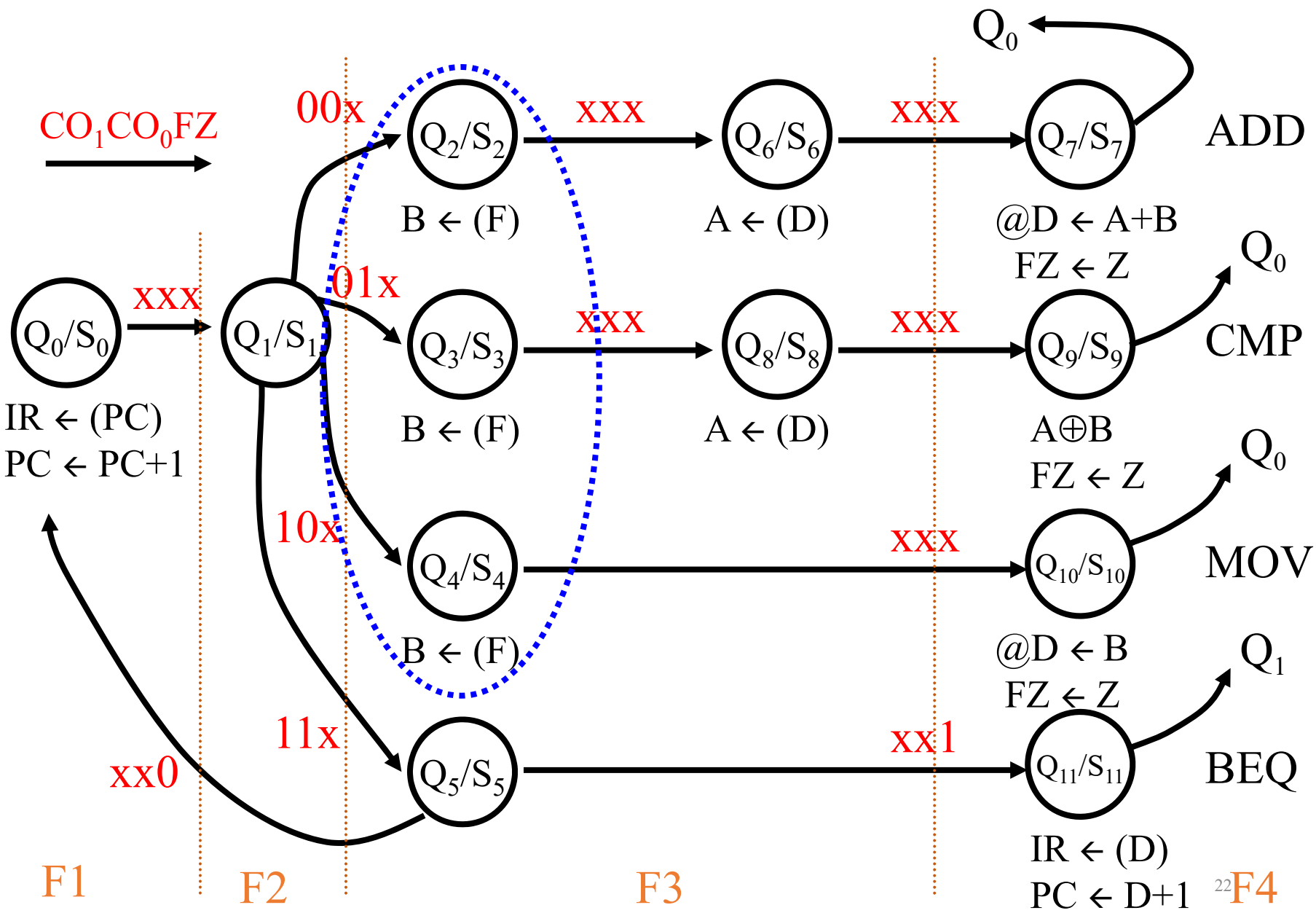
	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}
mx_1	0	X	1	1	1	X	1	1	1	X	1	1
mx_0	0	X	0	0	0	X	1	1	1	X	1	1
alu_1	X	X	X	X	X	X	X	0	X	0	1	X
alu_0	X	X	X	X	X	X	X	0	X	1	0	X
$\overline{L}/E$	0	0	0	0	0	0	0	1	0	0	1	0
pc	1	0	0	0	0	0	0	0	0	0	0	1
ir	1	0	0	0	0	0	0	0	0	0	0	1
a	0	0	0	0	0	0	1	0	1	0	0	0
b	0	0	1	1	1	0	0	0	0	0	0	0
fz	0	0	0	0	0	0	0	1	0	1	1	0


S0
IR ← (PC)
PC ← PC+1
S1 y S5
???
S2 , S3 y S4
B ← (F)
S7
@D ← A+B
FZ ← Z
S9
A⊕B
FZ ← Z
S10
@D ← B
FZ ← Z
S6 y S8
A ← (D)
S11
IR ← (D)
PC ← D+1
S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11
mx1 0 X 1 1 1 X 1 1 1 X 1 1
mx0 0 X 0 0 0 X 1 1 1 X 1 1
alu1 X X X X X X X 0 X 0 1 X
alu0 X X X X X X X 0 X 1 0 X
L/E 0 0 0 0 0 0 0 1 0 0 1 0
pc 1 0 0 0 0 0 0 0 0 0 0 1
ir 1 0 0 0 0 0 0 0 0 0 0 1
a 0 0 0 0 0 0 1 0 1 0 0 0
b 0 0 1 1 1 0 0 0 0 0 0 0
fz 0 0 0 0 0 0 0 1 0 1 1 0

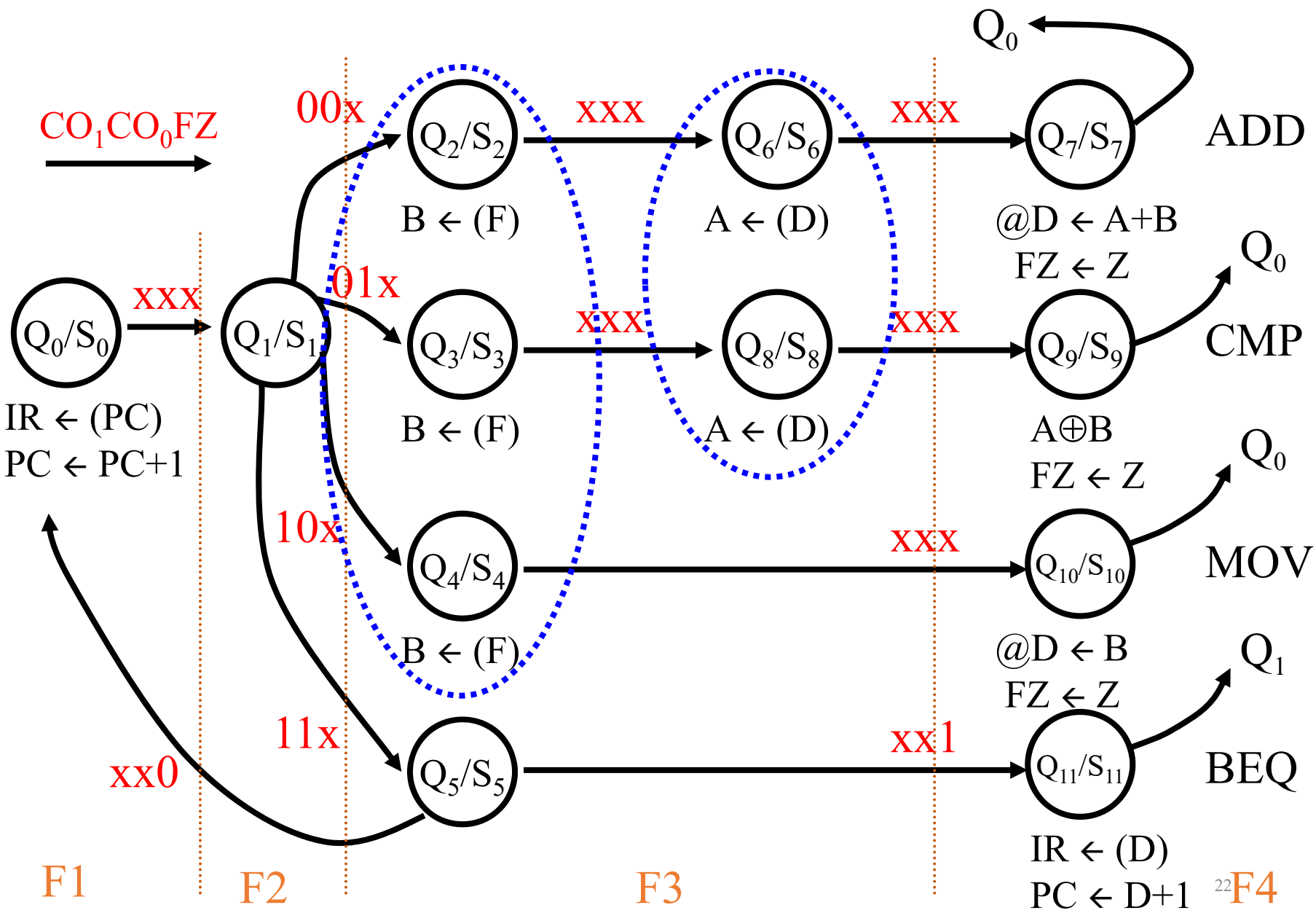
Reducción de estados



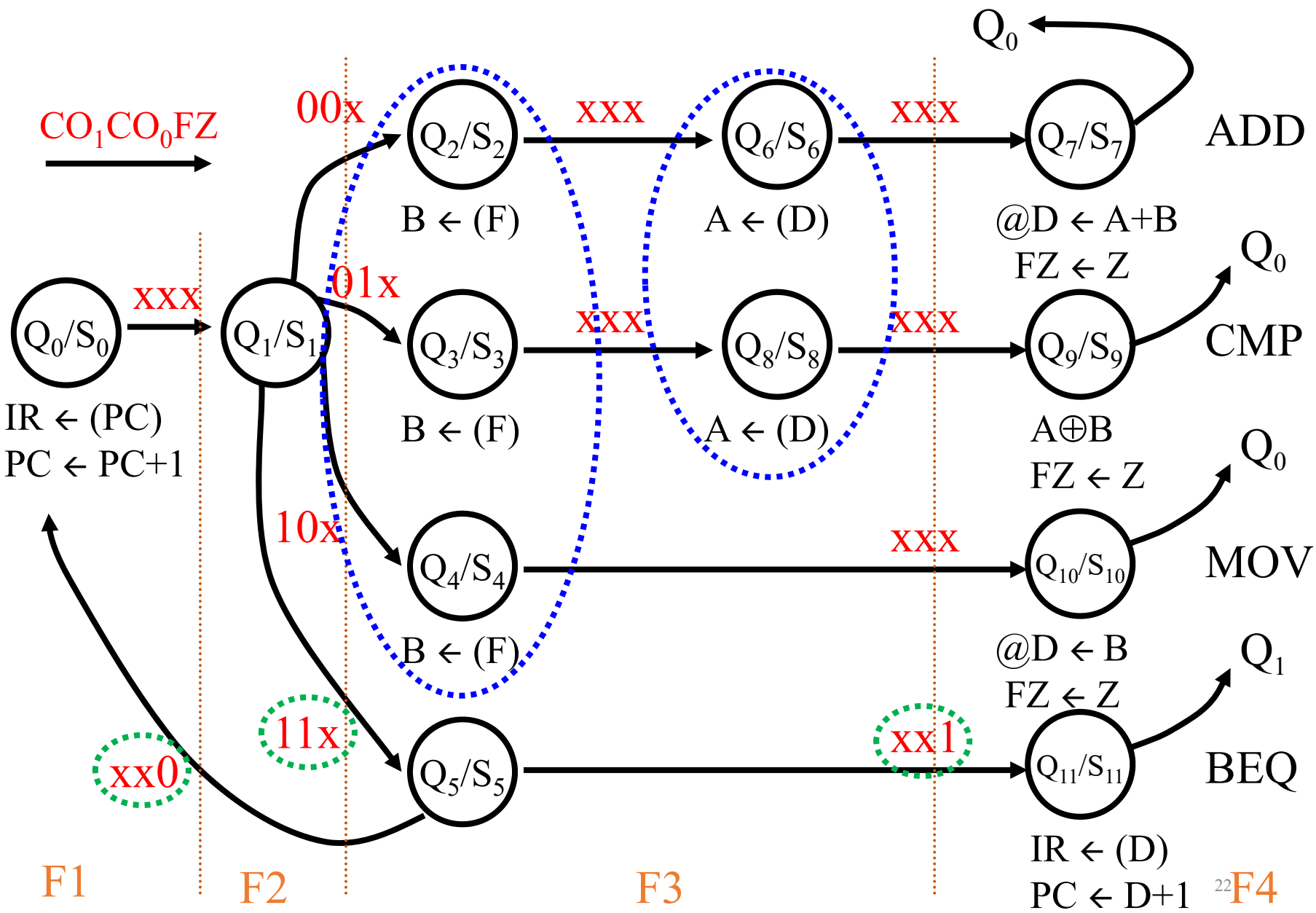
Reducción de estados



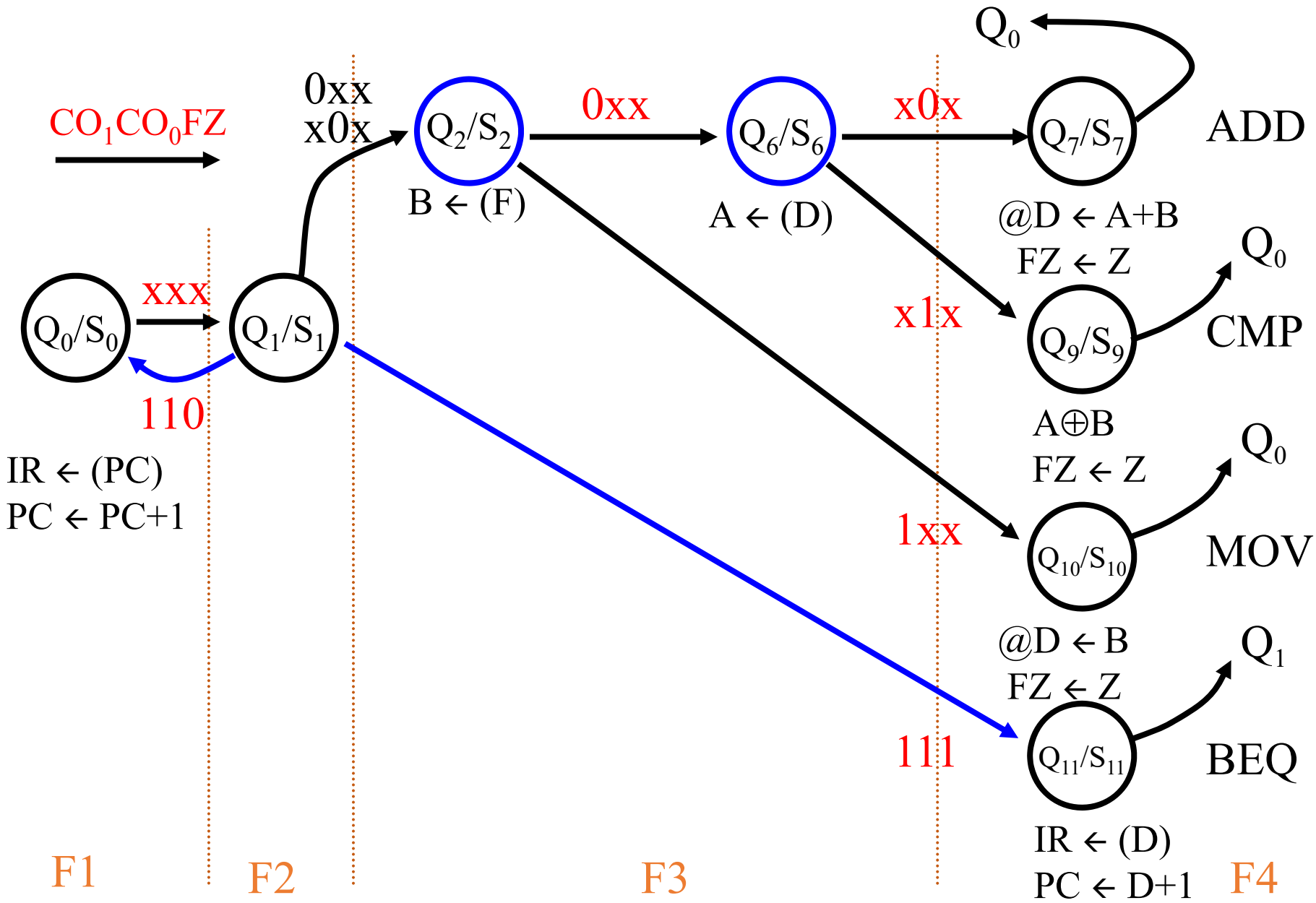
Reducción de estados



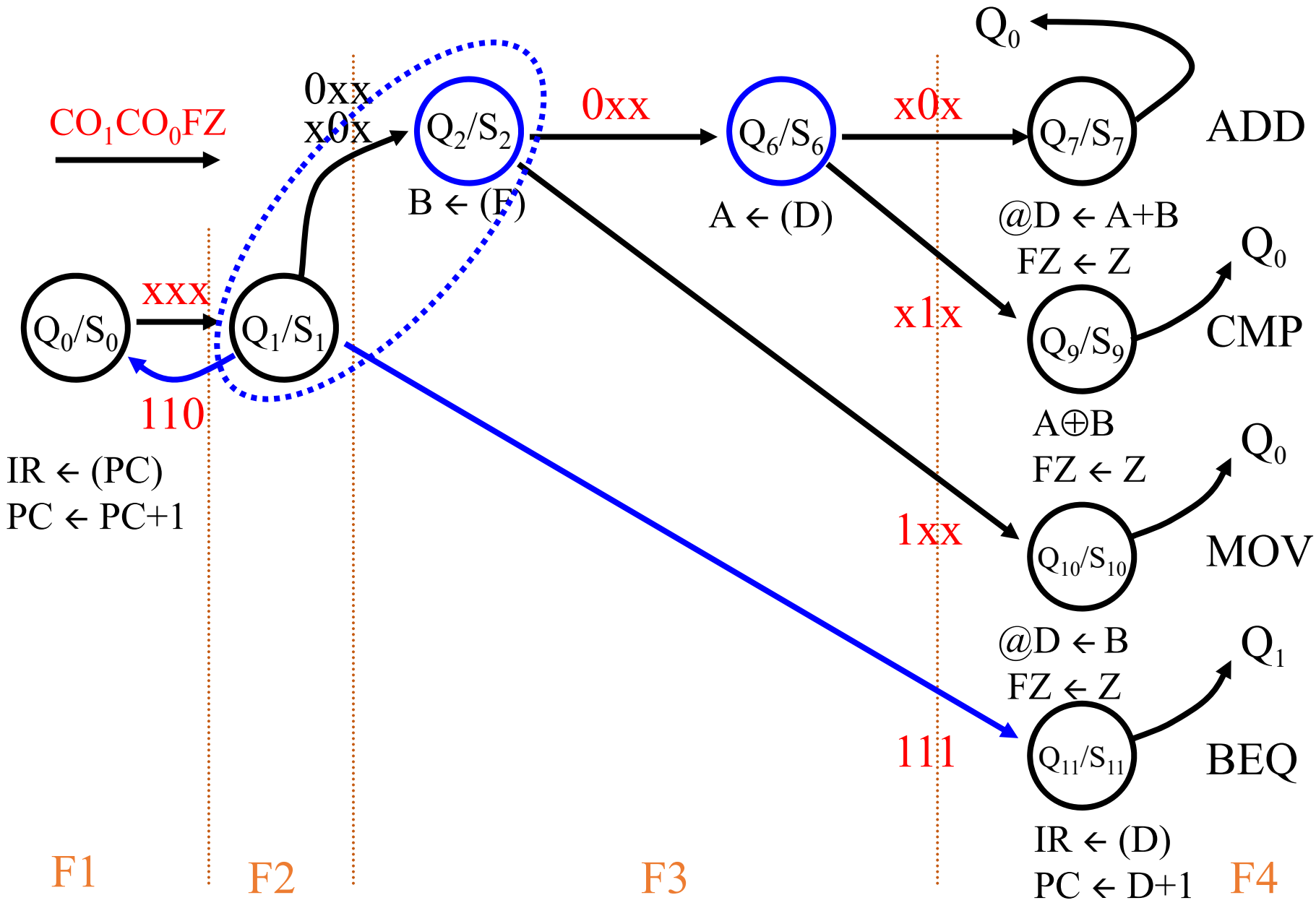
Reducción de estados



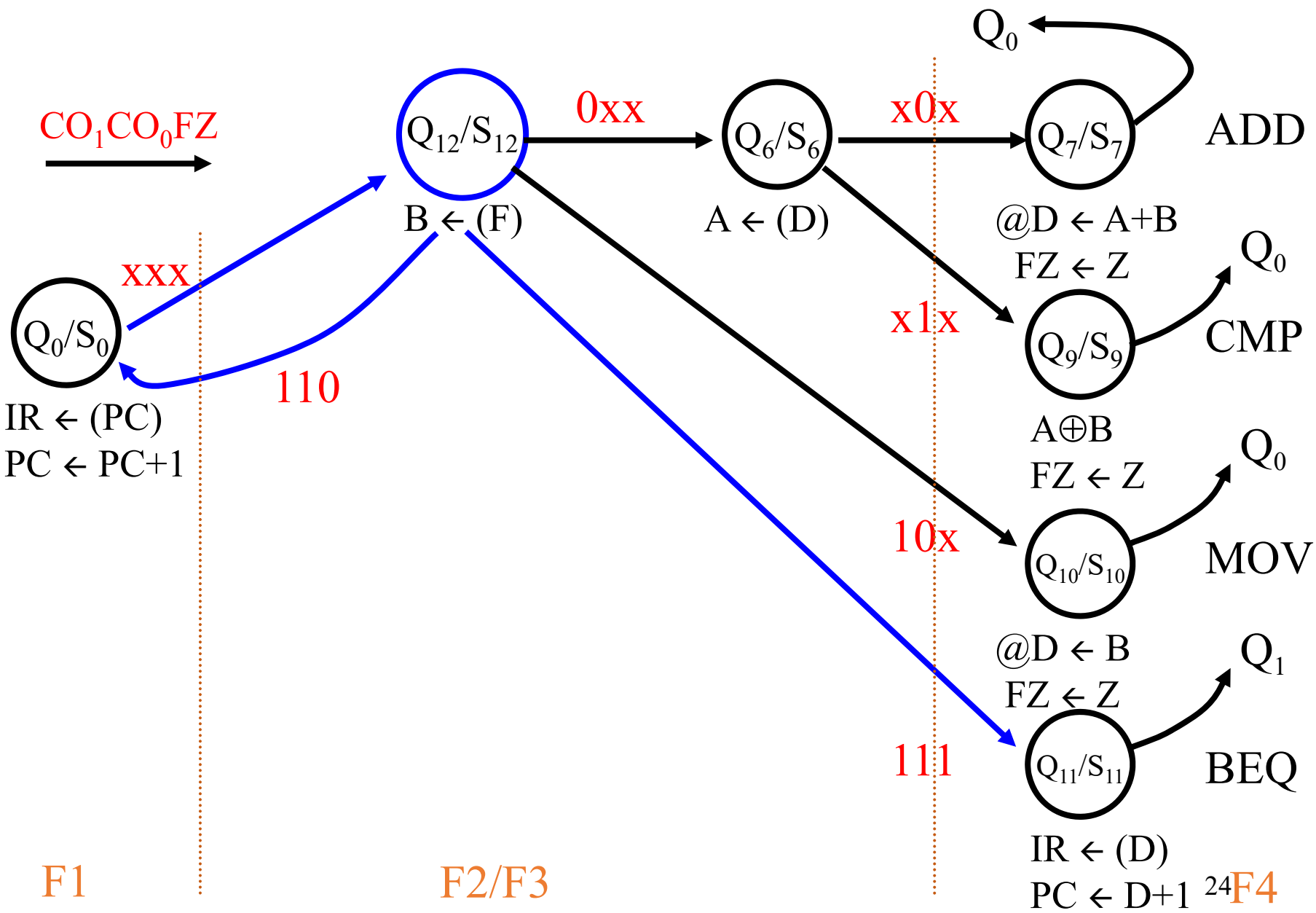
Reducción de estados



Reducción de estados



Reducción de estados



Implementación de la U.C.: autómata de Moore

- Tabla de transición

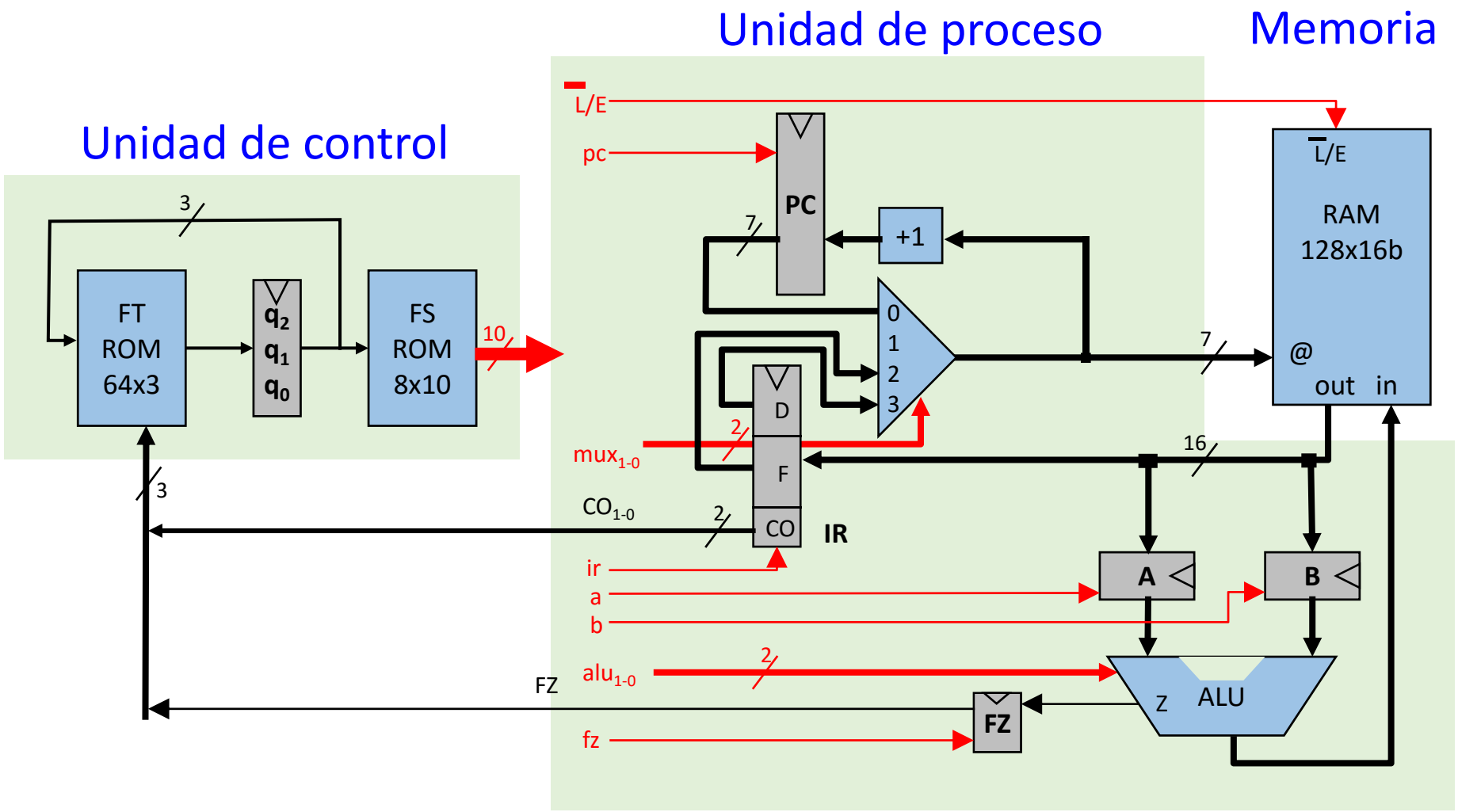
Q	ENT	Q+
$q_2\ q_1\ q_0$	$CO_1\ CO_0\ FZ$	$q_2^+\ q_1^+\ q_0^+$
...	...	...

- Tabla de salida

Q	SAL
$q_2\ q_1\ q_0$	$mx_1\ \ mx_0\ \ alu_1\ alu_0\ \ \overline{L}/E\ \ pc\ \ ir\ \ a\ \ b\ \ fz$
...	...

- Resumen:
  - Q: 3 FFs D
  - FT: ROM 64x3
  - FS: ROM 8x10

Máquina sencilla completa de 7 estados



Tiempo de ejecución de un programa en la máquina sencilla

$$Tex(programa) = nº\ ciclos \cdot Tc$$

Ejercicio: calcula el tiempo de ejecución del programa de multiplicación con (a=3, b=20612) para una MS con un reloj de 1 GHz

Tiempo de ejecución de un programa en la máquina sencilla

$$Tex(programa) = nº\ ciclos \cdot Tc$$

$$= (I_{add} \cdot C_{add} + I_{cmp} \cdot C_{cmp} + I_{mov} \cdot C_{mov} + I_{beq} \cdot C_{beq}) \cdot Tc$$

Ejercicio: calcula el tiempo de ejecución del programa de multiplicación con (a=3, b=20612) para una MS con un reloj de 1 GHz

Tiempo de ejecución de un programa en la máquina sencilla

$$Tex(programa) = nº\ ciclos \cdot Tc$$

$$= (I_{add} \cdot C_{add} + I_{cmp} \cdot C_{cmp} + I_{mov} \cdot C_{mov} + I_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \cdot (f_{add} \cdot C_{add} + f_{cmp} \cdot C_{cmp} + f_{mov} \cdot C_{mov} + f_{beq} \cdot C_{beq}) \cdot Tc$$

Ejercicio: calcula el tiempo de ejecución del programa de multiplicación con (a=3, b=20612) para una MS con un reloj de 1 GHz

Tiempo de ejecución de un programa en la máquina sencilla

$$Tex(programa) = nº\ ciclos \cdot Tc$$

$$= (I_{add} \cdot C_{add} + I_{cmp} \cdot C_{cmp} + I_{mov} \cdot C_{mov} + I_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \cdot (f_{add} \cdot C_{add} + f_{cmp} \cdot C_{cmp} + f_{mov} \cdot C_{mov} + f_{beq} \cdot C_{beq}) \cdot Tc$$

f_{mov} ↓ fracción de MOV

Ejercicio: calcula el tiempo de ejecución del programa de multiplicación con (a=3, b=20612) para una MS con un reloj de 1 GHz

Tiempo de ejecución de un programa en la máquina sencilla

$$Tex(programa) = nº\ ciclos \cdot Tc$$

$$= (I_{add} \cdot C_{add} + I_{cmp} \cdot C_{cmp} + I_{mov} \cdot C_{mov} + I_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \cdot (f_{add} \cdot C_{add} + f_{cmp} \cdot C_{cmp} + f_{mov} \cdot C_{mov} + f_{beq} \cdot C_{beq}) \cdot Tc$$

f_{mov}: fracción de MOV

C_{mov}: ciclos de MOV

Ejercicio: calcula el tiempo de ejecución del programa de multiplicación con (a=3, b=20612) para una MS con un reloj de 1 GHz

Tiempo de ejecución de un programa en la máquina sencilla

$$Tex(programa) = nº\ ciclos \cdot Tc$$

$$= (I_{add} \cdot C_{add} + I_{cmp} \cdot C_{cmp} + I_{mov} \cdot C_{mov} + I_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \cdot (f_{add} \cdot C_{add} + f_{cmp} \cdot C_{cmp} + f_{mov} \cdot C_{mov} + f_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \times CPI \times Tc$$

f_{mov}: fracción de MOV

C_{mov}: ciclos de MOV

Ejercicio: calcula el tiempo de ejecución del programa de multiplicación con (a=3, b=20612) para una MS con un reloj de 1 GHz

Tiempo de ejecución de un programa en la máquina sencilla

$$Tex(programa) = nº\ ciclos \cdot Tc$$

$$= (I_{add} \cdot C_{add} + I_{cmp} \cdot C_{cmp} + I_{mov} \cdot C_{mov} + I_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \cdot (f_{add} \cdot C_{add} + f_{cmp} \cdot C_{cmp} + f_{mov} \cdot C_{mov} + f_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \times CPI \times Tc$$

fracción de MOV

ciclos de MOV

nº de instrucciones ejecutadas

- *Algoritmo*
- *Compilador*

Ejercicio: calcula el tiempo de ejecución del programa de multiplicación con (a=3, b=20612) para una MS con un reloj de 1 GHz

Tiempo de ejecución de un programa en la máquina sencilla

$$Tex(programa) = nº\ ciclos \cdot Tc$$

$$= (I_{add} \cdot C_{add} + I_{cmp} \cdot C_{cmp} + I_{mov} \cdot C_{mov} + I_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \cdot (f_{add} \cdot C_{add} + f_{cmp} \cdot C_{cmp} + f_{mov} \cdot C_{mov} + f_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \times CPI \times Tc$$

fracción de MOV

ciclos de MOV

nº de instrucciones ejecutadas

nº medio de ciclos por instrucción

- *Unidad de control*
- *Algoritmo*
- *Compilador*

Ejercicio: calcula el tiempo de ejecución del programa de multiplicación con (a=3, b=20612) para una MS con un reloj de 1 GHz

Tiempo de ejecución de un programa en la máquina sencilla

$$Tex(programa) = nº\ ciclos \cdot Tc$$

$$= (I_{add} \cdot C_{add} + I_{cmp} \cdot C_{cmp} + I_{mov} \cdot C_{mov} + I_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \cdot (f_{add} \cdot C_{add} + f_{cmp} \cdot C_{cmp} + f_{mov} \cdot C_{mov} + f_{beq} \cdot C_{beq}) \cdot Tc$$

$$= I \times CPI \times Tc$$

fracción de MOV

ciclos de MOV

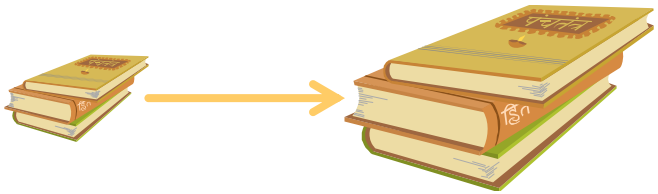
tiempo de ciclo ---------> • *Ruta de datos, memoria*

nº medio de ciclos por instrucción ----------> • *Unidad de control*

nº de instrucciones ejecutadas ----------> • *Algoritmo* • *Compilador*

Ejercicio: calcula el tiempo de ejecución del programa de multiplicación con (a=3, b=20612) para una MS con un reloj de 1 GHz

Mejoras: añadir instrucciones al repertorio



Programa en Lenguaje Ensamblador

Programa traductor de Lenguaje Ensamblador a Lenguaje Máquina

Compilador

	Usuario
uso	Lenguajes de Alto Nivel (C, C++, …)
uso	Llamadas al Sistema Operativo
uso	**Arquitectura Lenguaje Máquina**
diseño	Organización (µArquitectura) → UC + UP
diseño	Circuitos digitales
diseño	Electrónica

µProcesador

¿ Como añadir una instrucción al repertorio?

- Tres formas:
  - *Software*: no se modifica nada (barata y lenta)
    → se añade únicamente al lenguaje ensamblador !
    el compilador emite la secuencia necesaria de instrucciones máquina
  - *Hardware*: se modifica UP y UC
    (caro y rápido)
  - *Firmware*: se modifica sólo la UC
- Ejemplo: añadir instrucción
  - CUAD F, D @D ← 4 * (F)

Ejemplo: añadimos CUAD al repertorio

- **SW**:
  CUAD a ,b ≡

  MOV a, b

  ADD b, b

  ADD b, b

- **HW**: añado inst CUAD modificando ALU

  B ← (F)

  @D ← 4*B

- **FW**:
  añado inst CUAD sin modificar ALU

  A ← (F) ; B ← (F)

  @D ← A + B

  A ← (D) ; B ← (D)

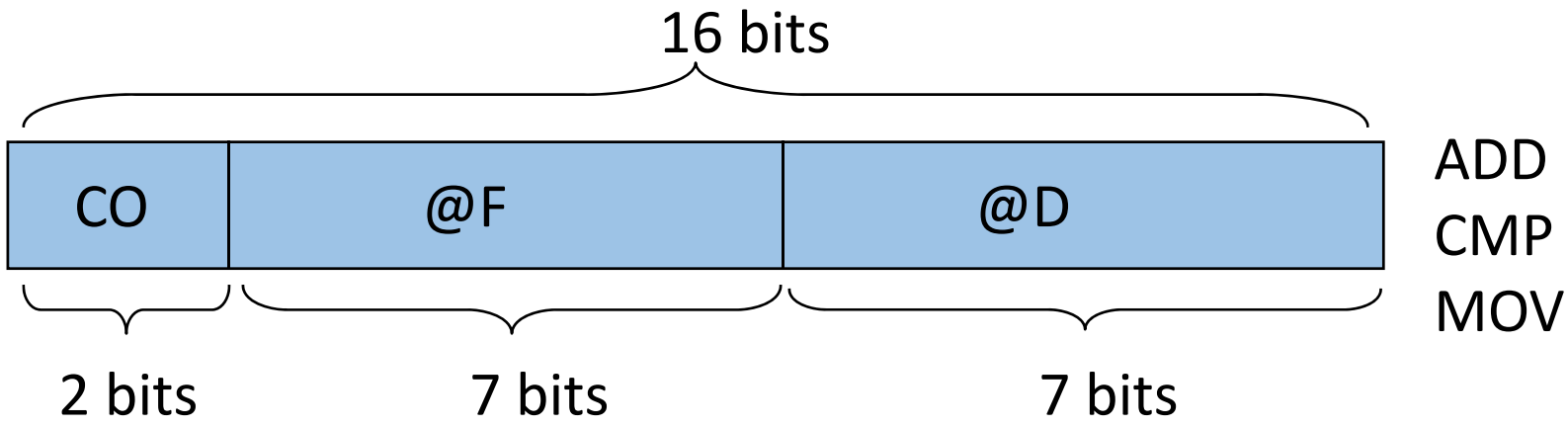
  @D ← A + B

Ejemplo: añadimos tres instrucciones. Modificación HW

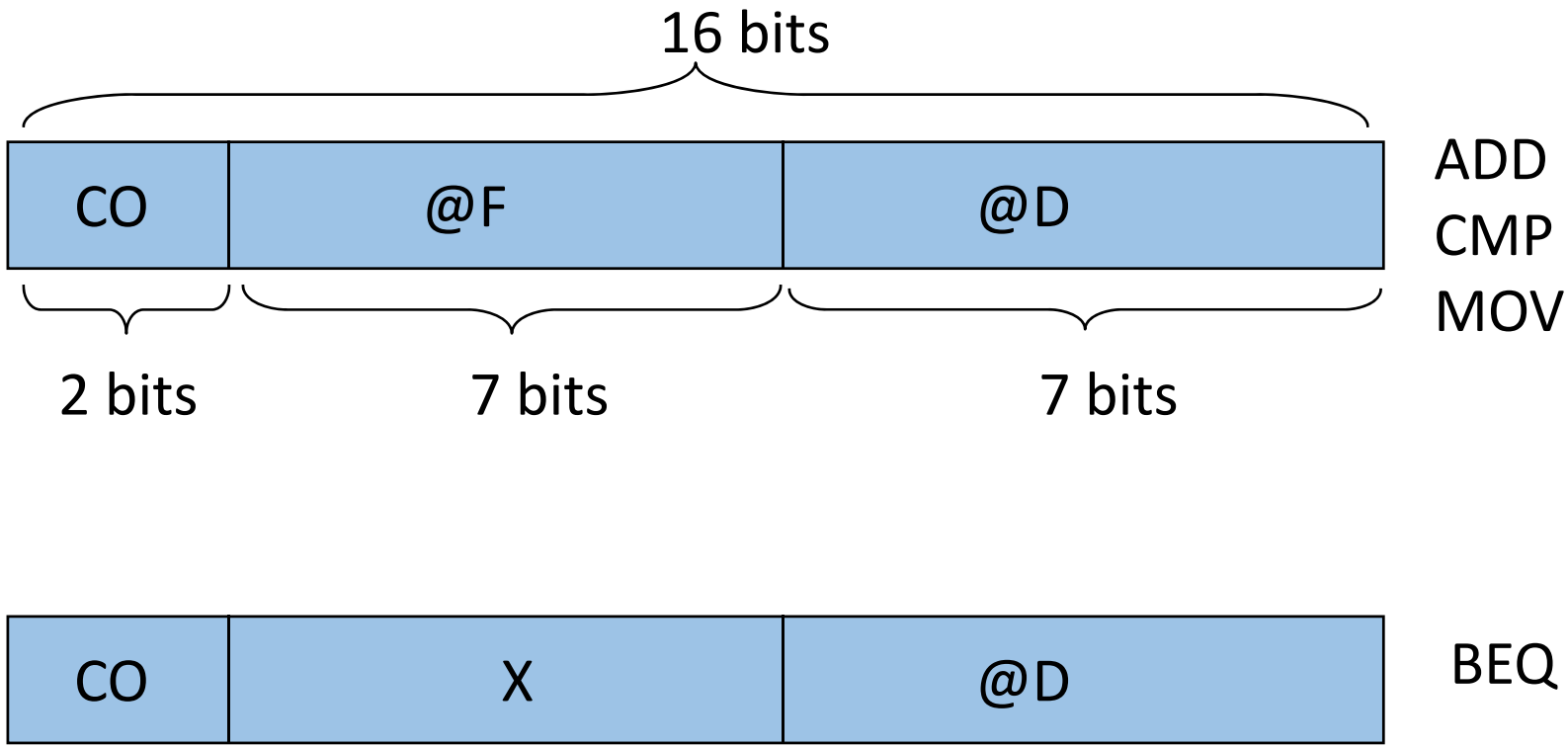
- Añadir las siguientes instrucciones:
  - CLEAR D @D ← 0
  - MOVD K, D @D ← K ; K constante de 16 bits
  - ACUM K, D @D ← (D) + K ; K constante de 16 bits

- Codificación expandida
  - usar bits libres en alguna instrucción

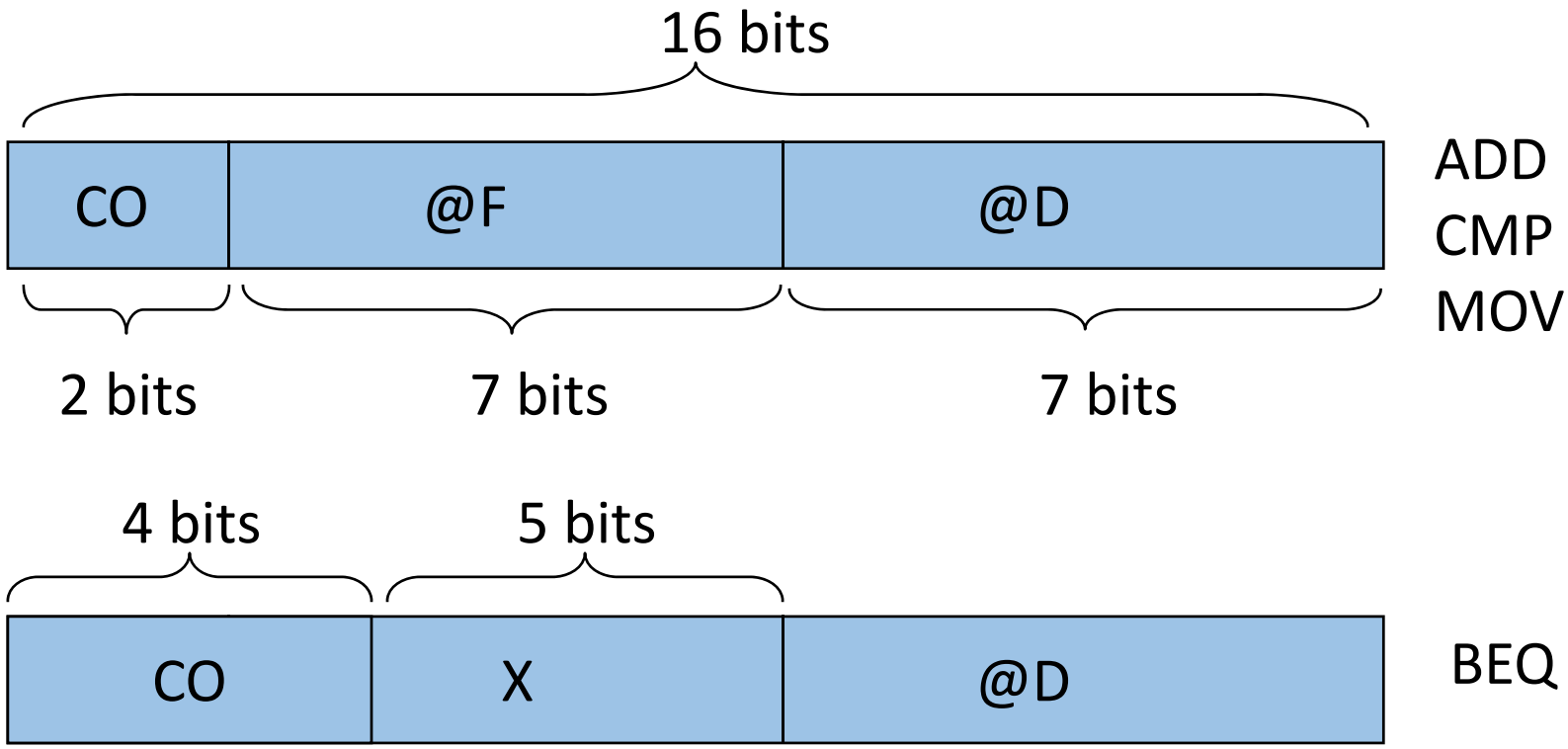
Codificación expandida



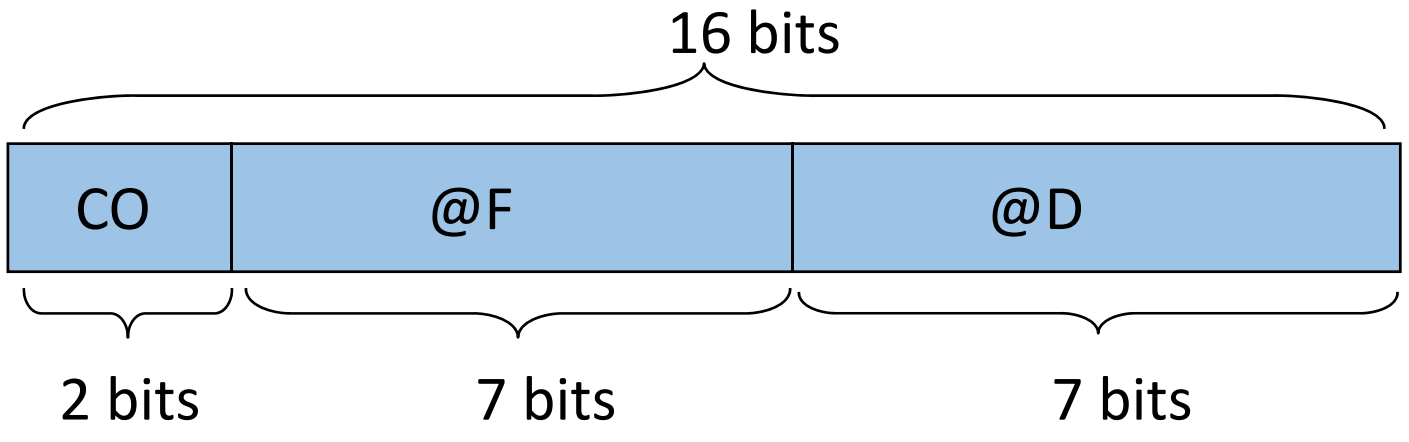
Codificación expandida



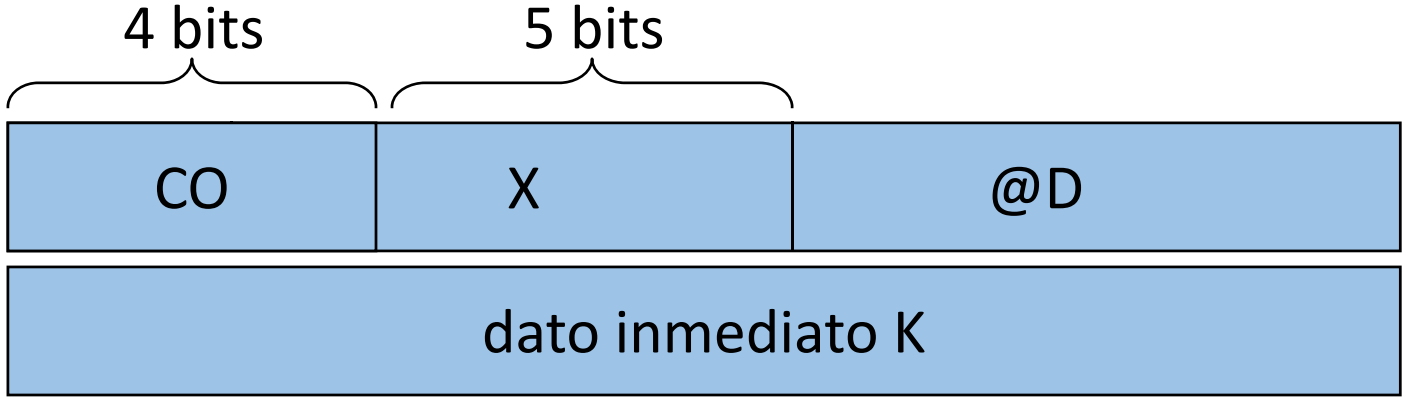
Codificación expandida



Codificación expandida

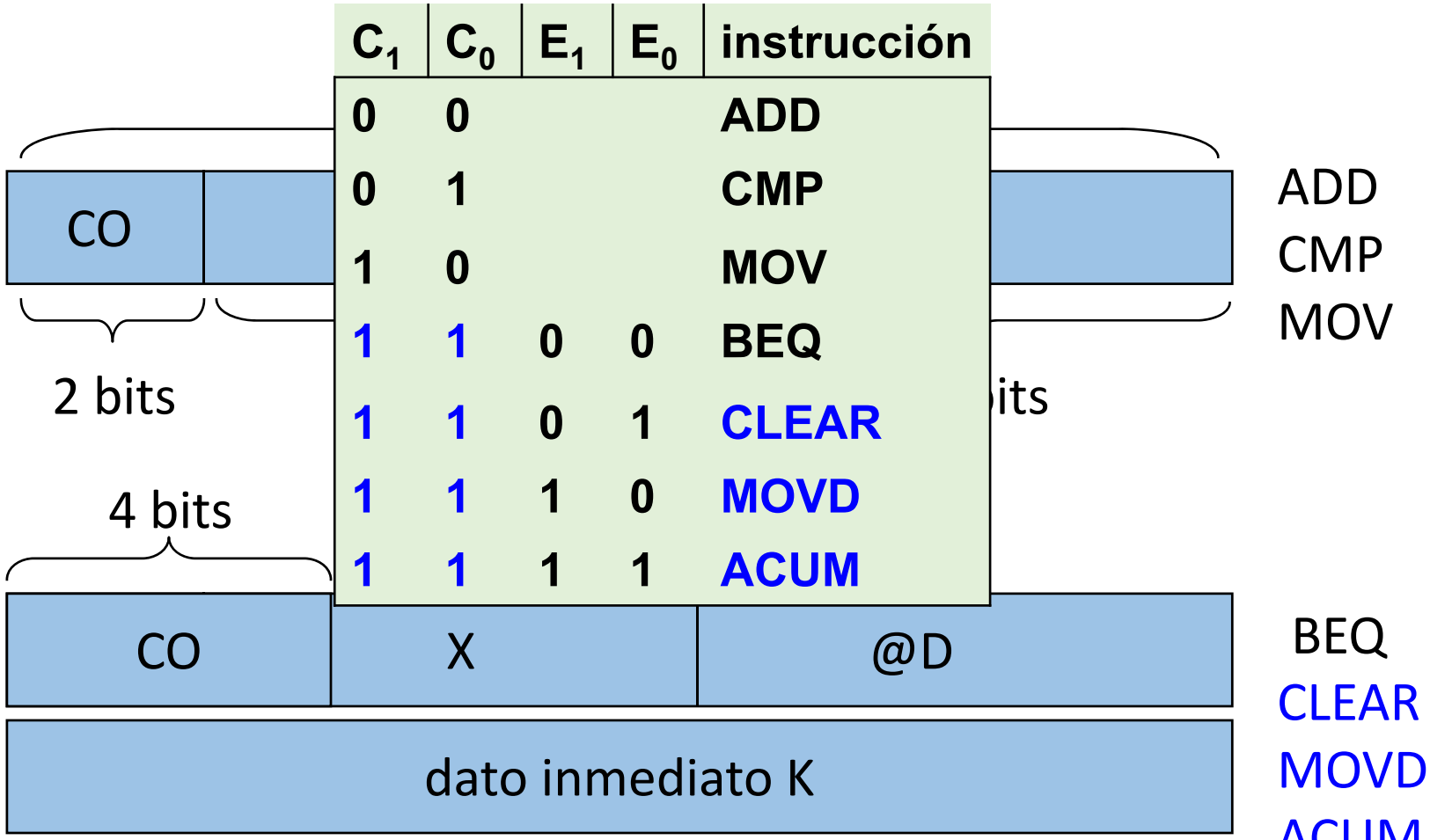


ADD
CMP
MOV

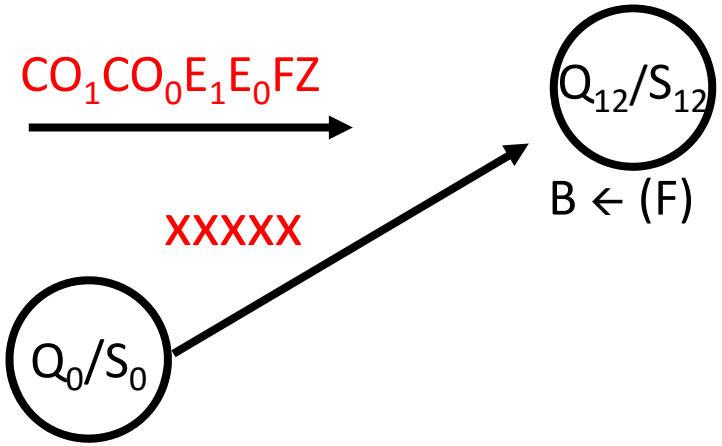


BEQ
CLEAR
MOVD
ACUM

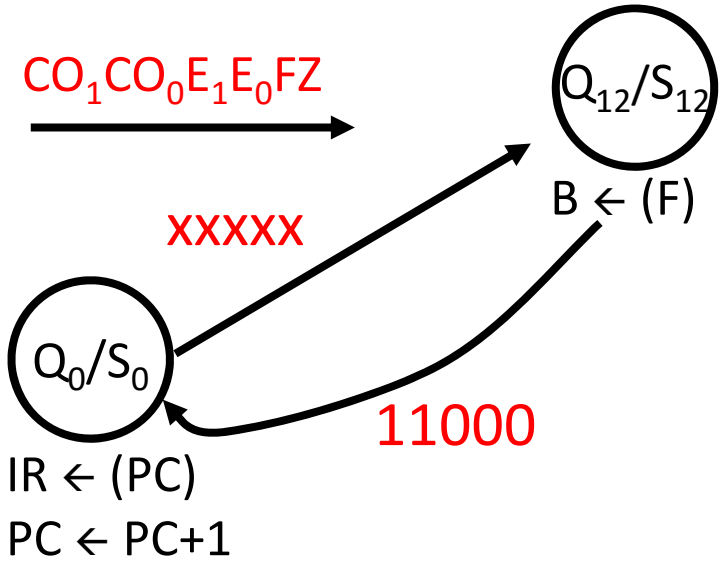
Codificación expandida



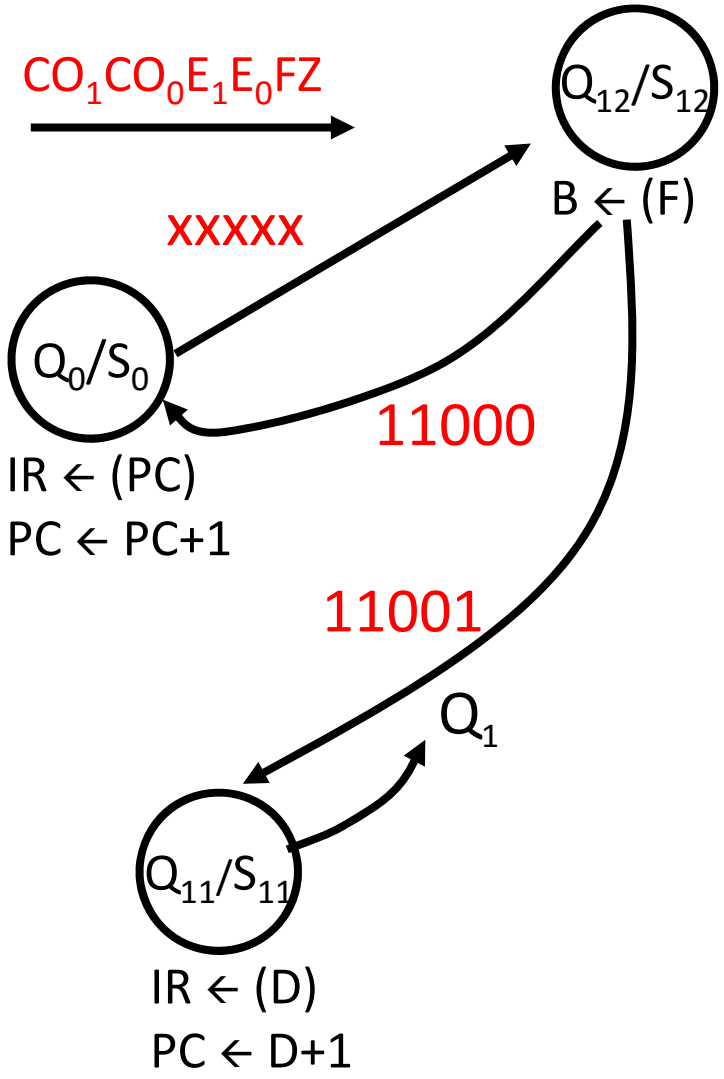
Autómata de la U.C.



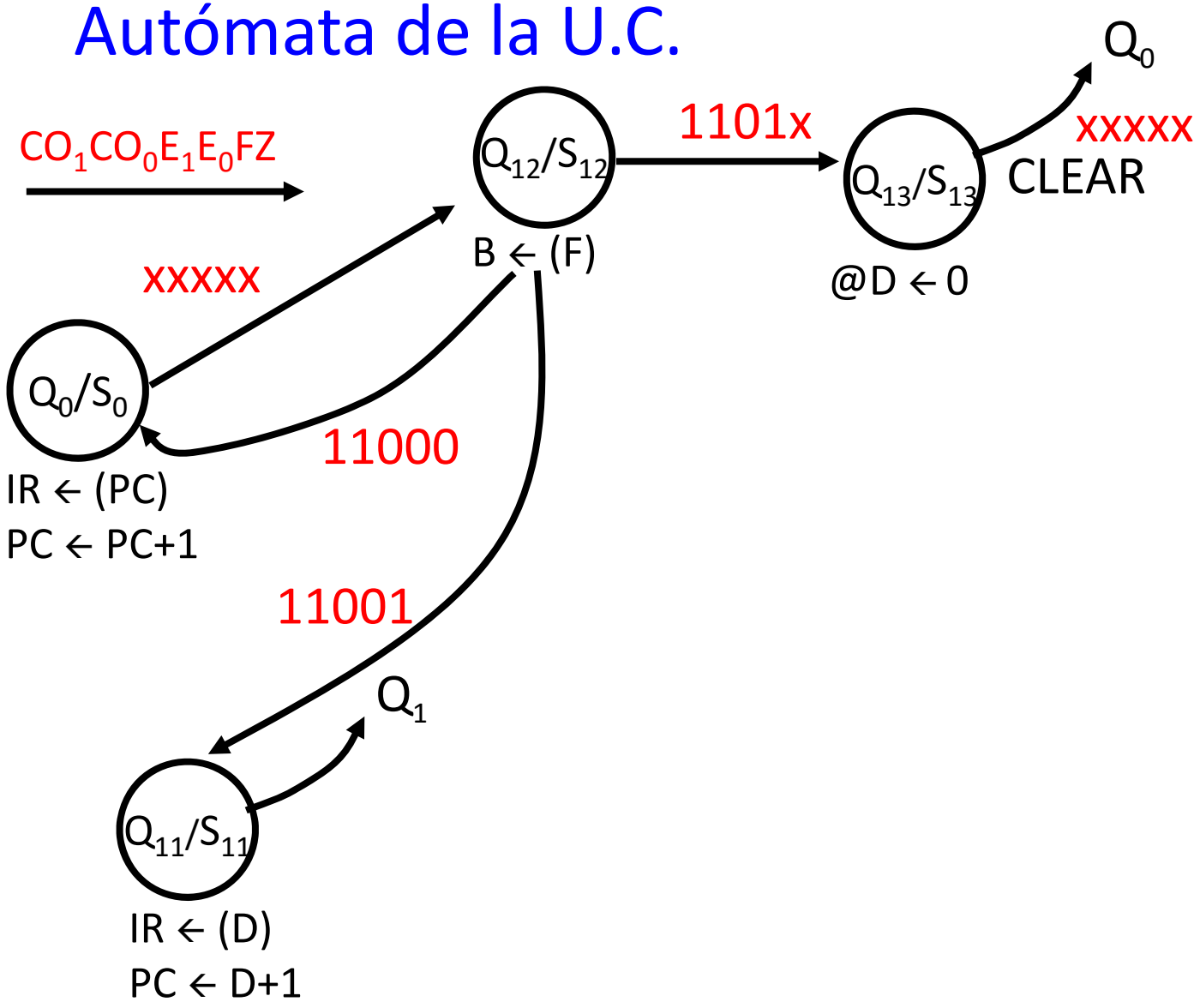
Autómata de la U.C.



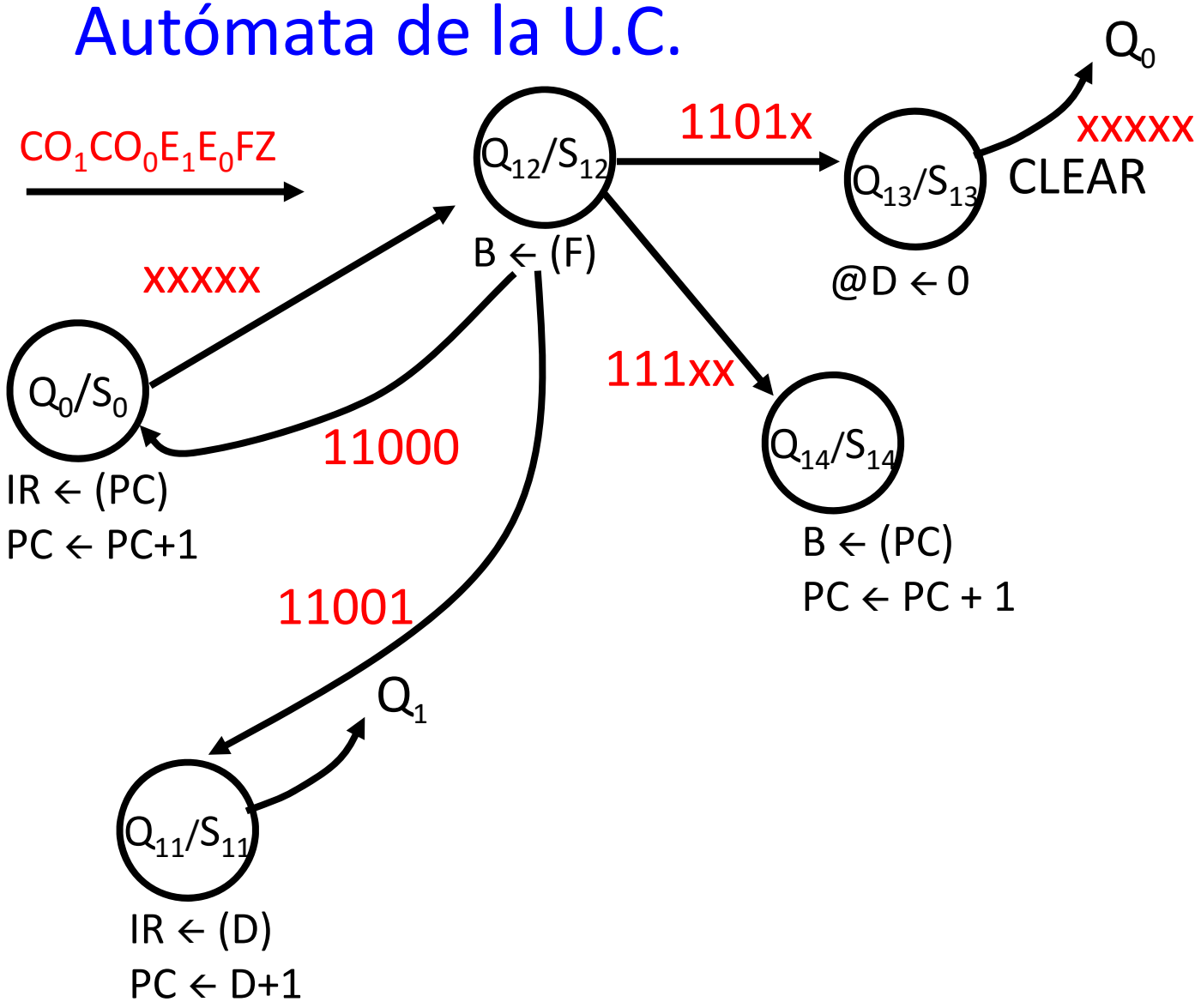
Autómata de la U.C.



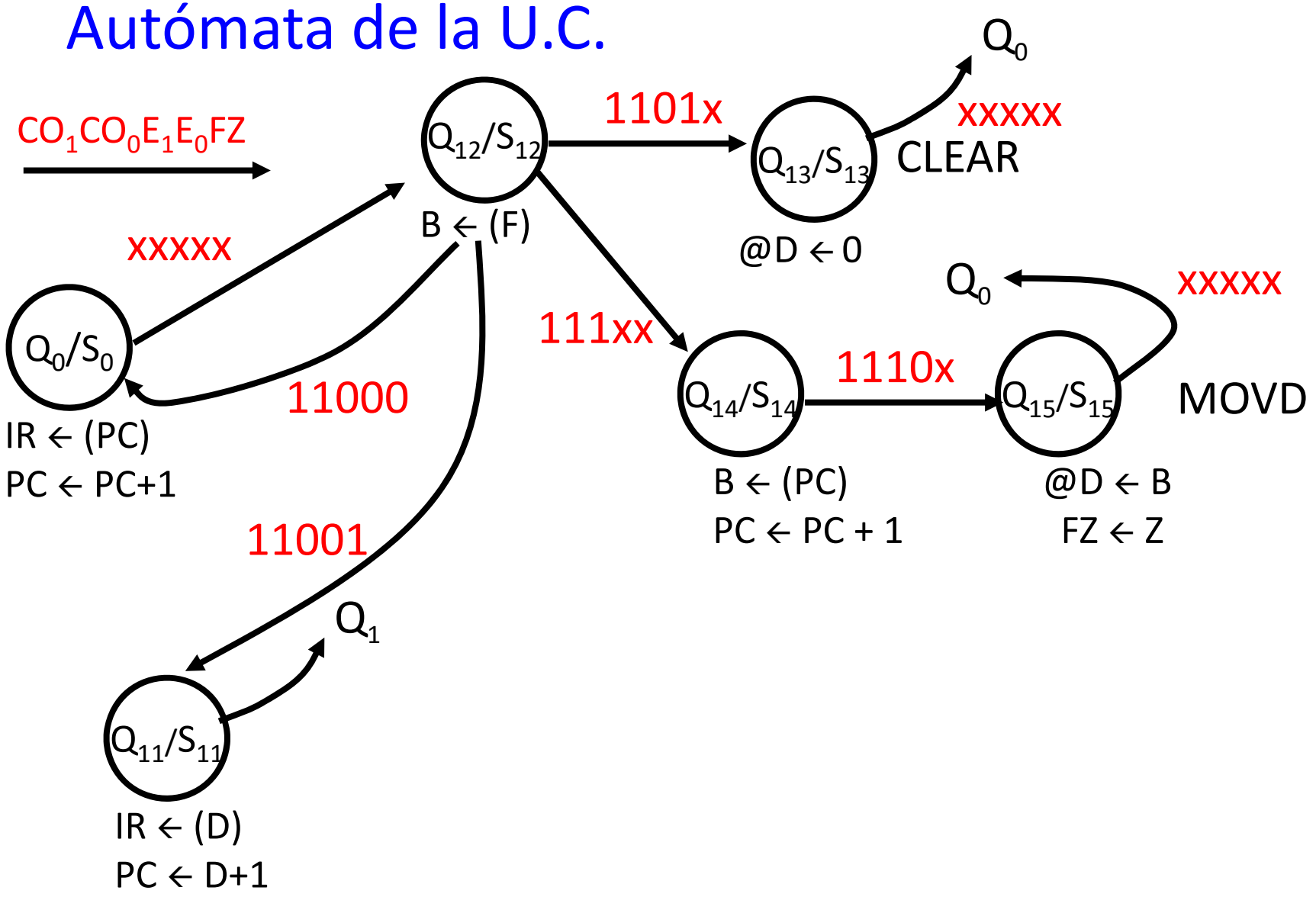
Autómata de la U.C.



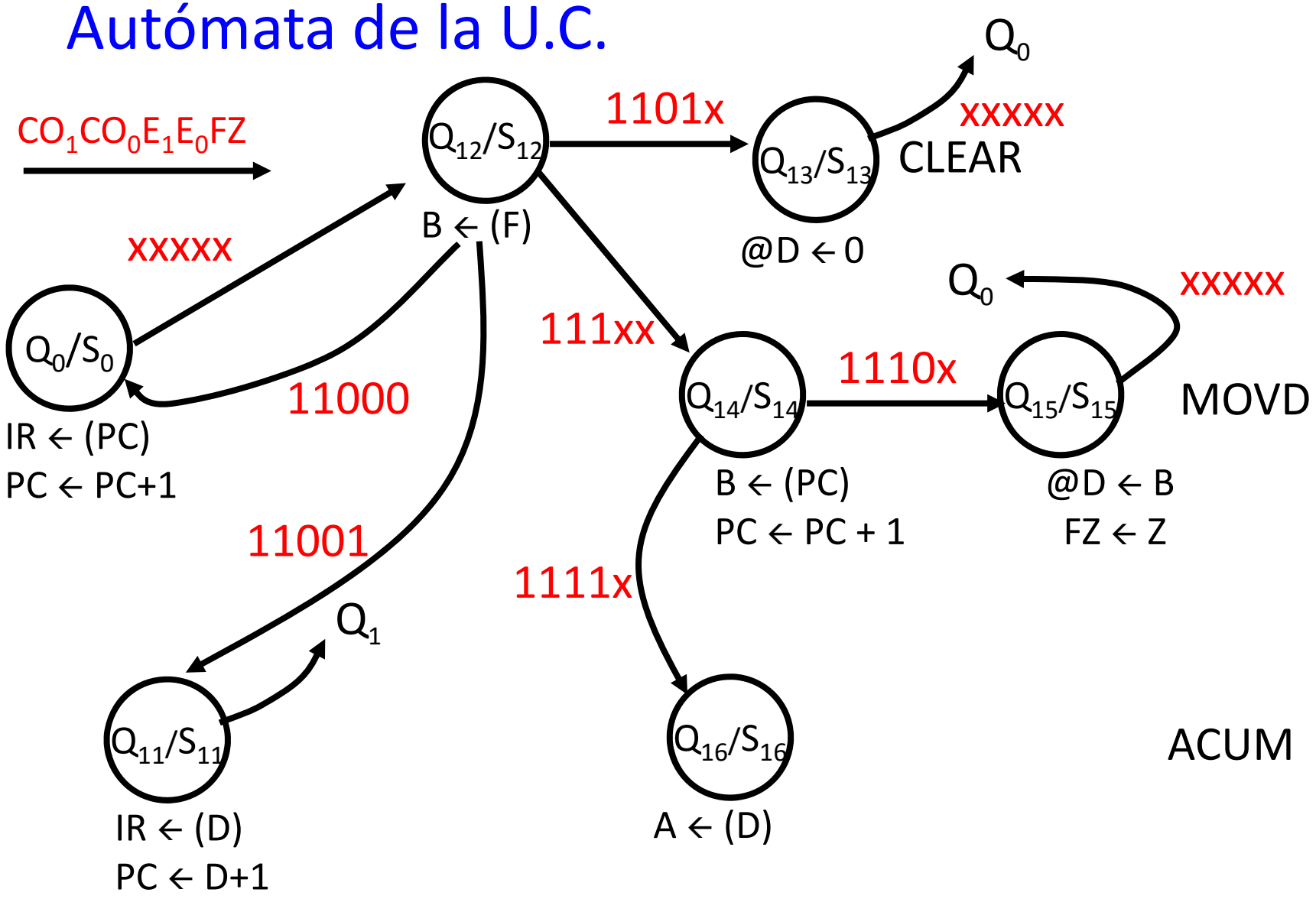
Autómata de la U.C.



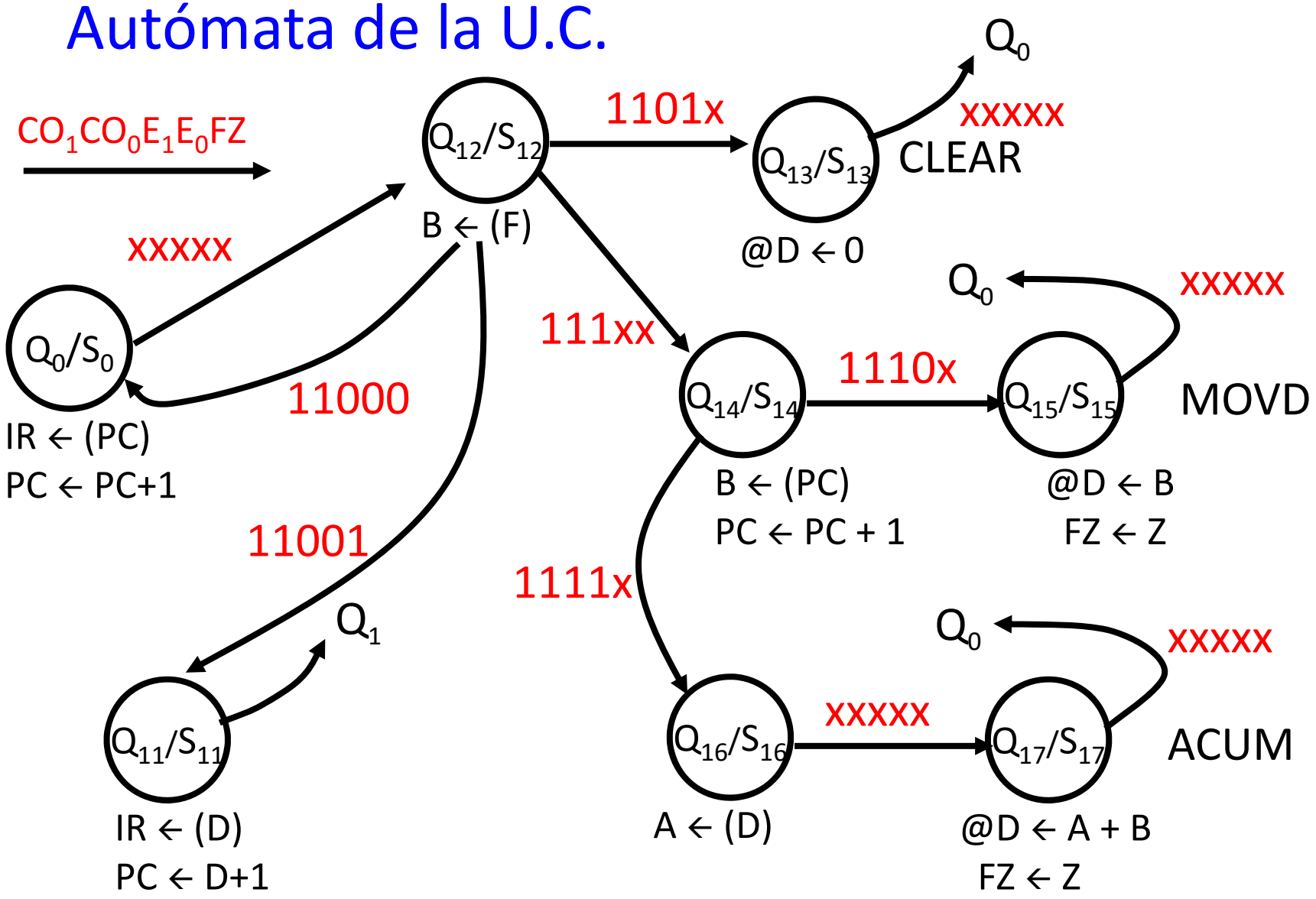
Autómata de la U.C.



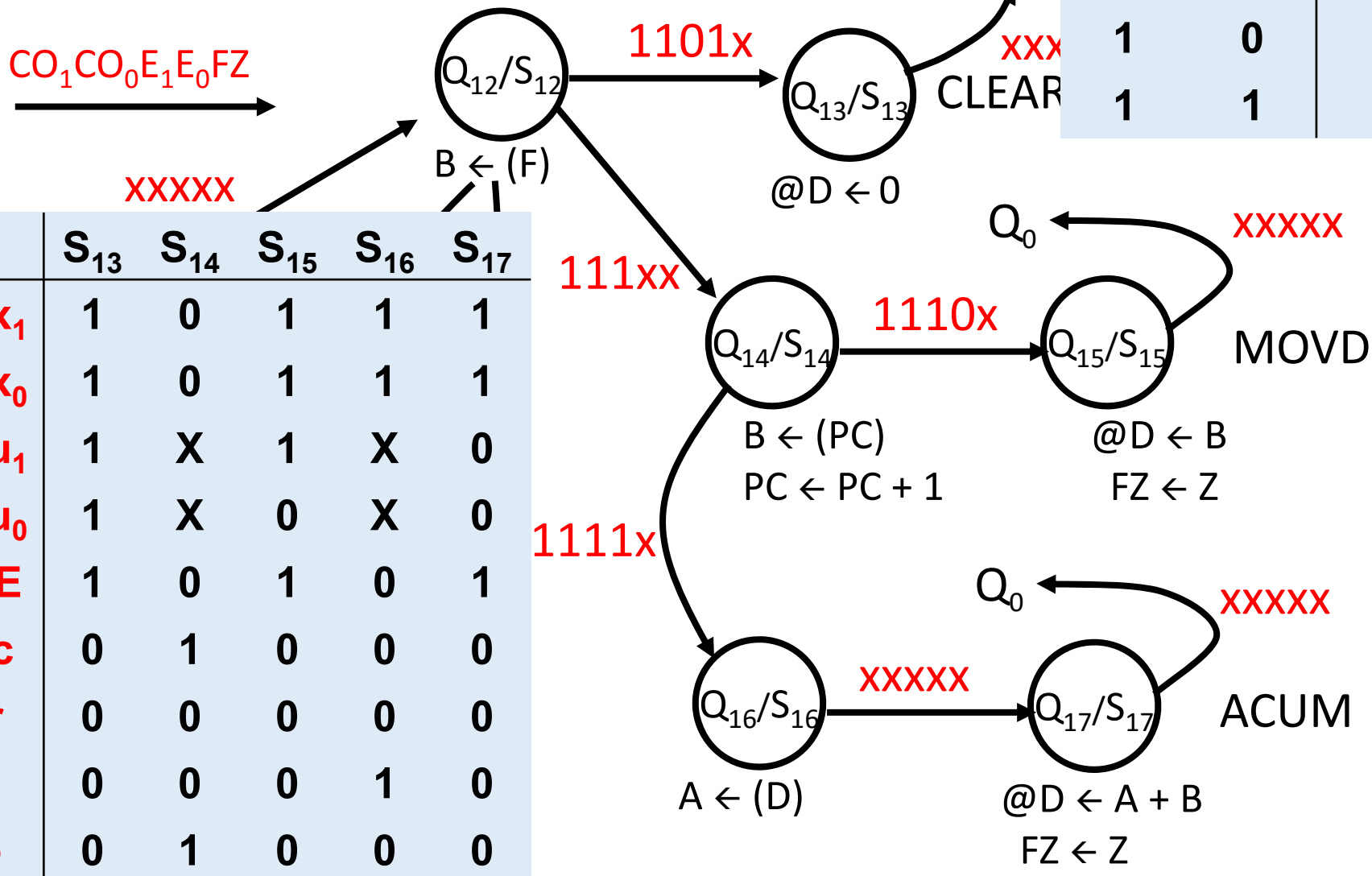
Autómata de la U.C.



Autómata de la U.C.



Autómata de la U.C.



	S_{13}	S_{14}	S_{15}	S_{16}	S_{17}
mx_1	1	0	1	1	1
mx_0	1	0	1	1	1
alu_1	1	X	1	X	0
alu_0	1	X	0	X	0
$\overline{L/E}$	1	0	1	0	1
pc	0	1	0	0	0
ir	0	0	0	0	0
a	0	0	0	1	0
b	0	1	0	0	0
fz	0	0	1	0	1

ALU_1	ALU_0	op
0	0	A + B
0	1	A ⊕ B
1	0	B
1	1	0

U.C. microprogramada

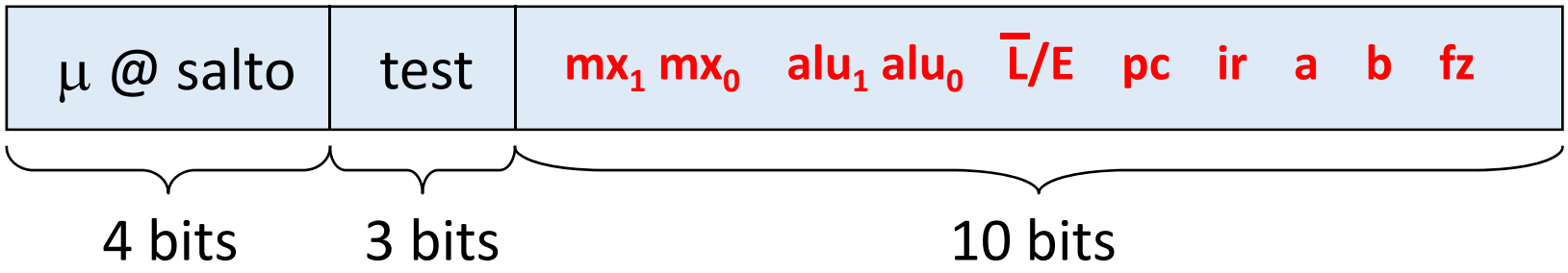
- U.C. cableada:
  - autómata → estado → RTL
  - decodificación

- U.C. μprogramada:
  - μprograma → μinstrucciones → RTL
  - μPC
    - ✓ secuenciamiento implícito
    - ✓ μ saltos

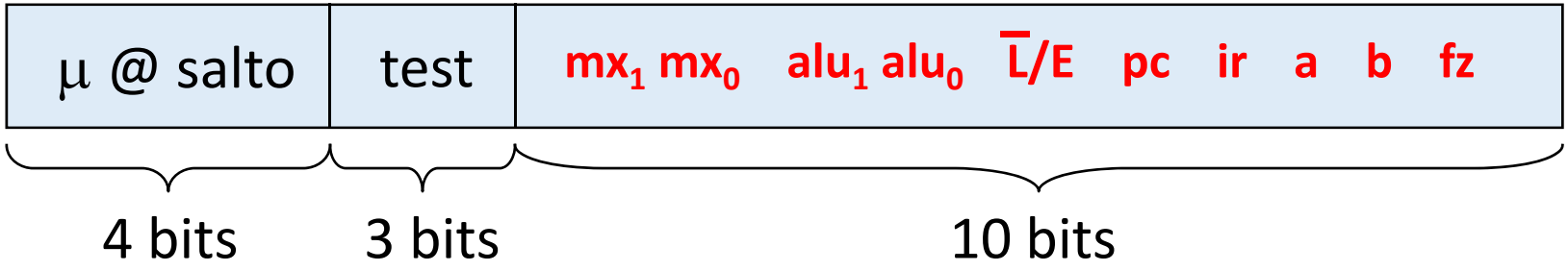
¿Cómo sería el μprograma?

μ@	RTL μinstrucción	
0	IR ← (PC) PC ← PC + 1	
1	Decod	
2	B ← (F)	ADD
3	A ← (D)	
4	@D ← A + B FZ ← Z	
5	B ← (F)	CMP
6	A ← (D)	
7	A ⊕ B FZ ← Z	
8	B ← (F)	MOV
9	@D ← B FZ ← Z	
10	IR ← (D) PC ← D + 1	BEQ

Formato μinstrucción

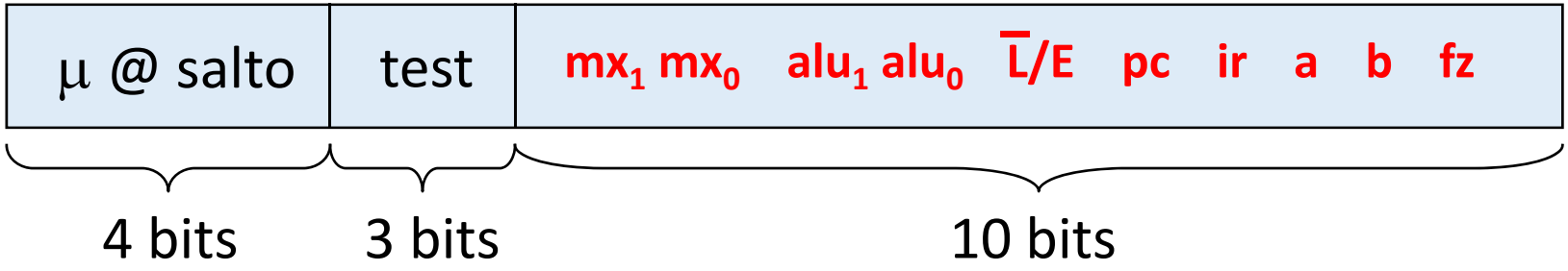


Formato μinstrucción



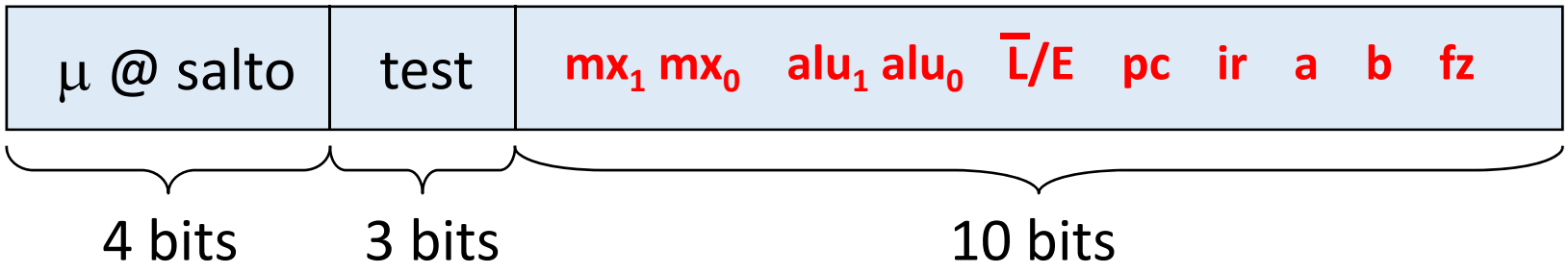
$$\mu PC^{+} = \begin{cases} \\ \\ \end{cases}$$

Formato μinstrucción



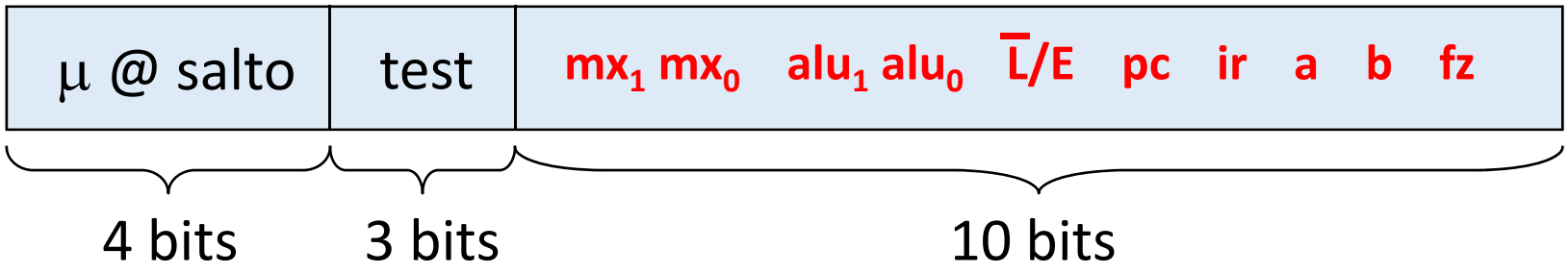
$$\mu PC^{+} = \begin{cases} \mu\text{@ salto} \quad \underline{\text{si}} \quad \text{bit}(\text{test}_{1\text{-}0}) = \text{test}_2 \\ \\ \end{cases}$$

Formato μinstrucción



$$\mu PC^{+} = \begin{cases} \mu@\ \text{salto} \quad \underline{si} \quad bit(test_{1\text{-}0}) = test_2 \\ \mu PC + 1 \quad \underline{cc} \quad (\neq) \end{cases}$$

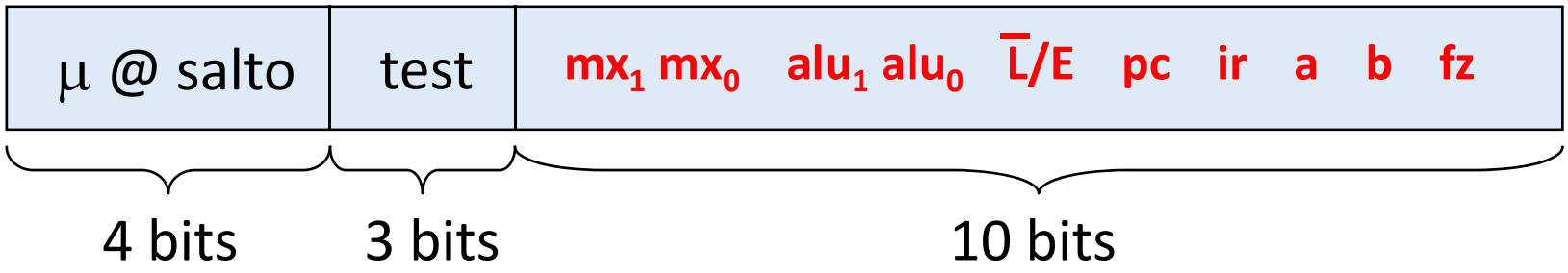
Formato μinstrucción



$test_{1\text{-}0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

$$\mu PC^{+} = \begin{cases} \mu@\ \text{salto} \quad \underline{si} \quad bit(test_{1\text{-}0}) = test_2 \\ \mu PC + 1 \quad \underline{cc} \quad (\neq) \end{cases}$$

Formato μinstrucción

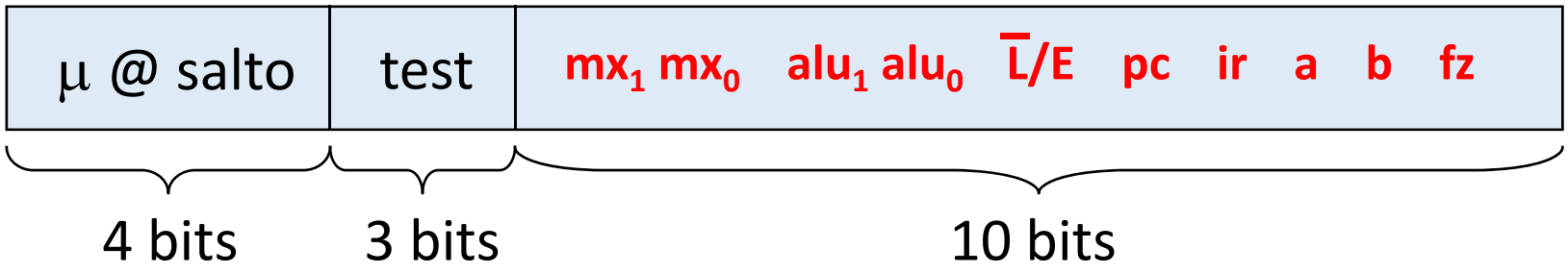


$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

$$\mu PC^{+} = \begin{cases} \mu@ \text{ salto} \quad \underline{si} \quad bit(test_{1-0}) = test_2 \\ \mu PC + 1 \quad \underline{cc} \quad (\neq) \end{cases}$$

- En "$test_{1-0}$" se codifica 1 bit

Formato μinstrucción

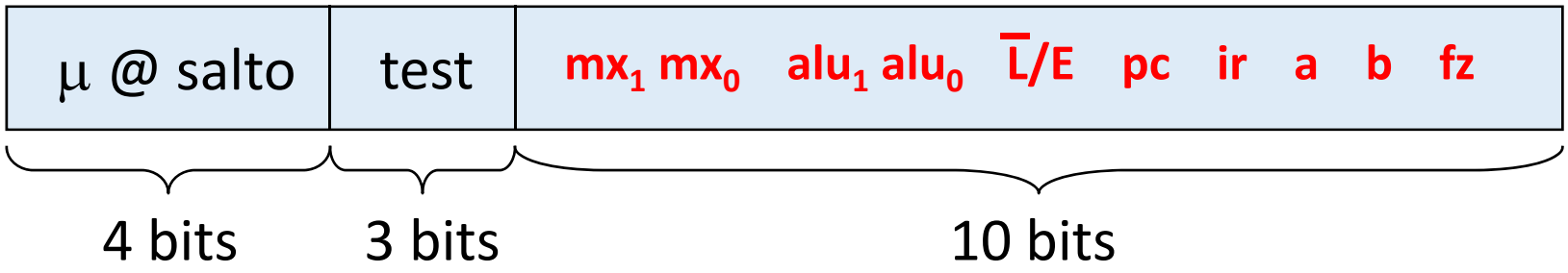


$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

$$\mu PC^{+} = \begin{cases} \mu@ \text{ salto} \quad \underline{si} \quad bit(test_{1-0}) = test_2 \\ \mu PC + 1 \quad \underline{cc} \quad (\neq) \end{cases}$$

- En "$test_{1-0}$" se codifica 1 bit
- Sólo hay 1 destino de salto

Formato μinstrucción

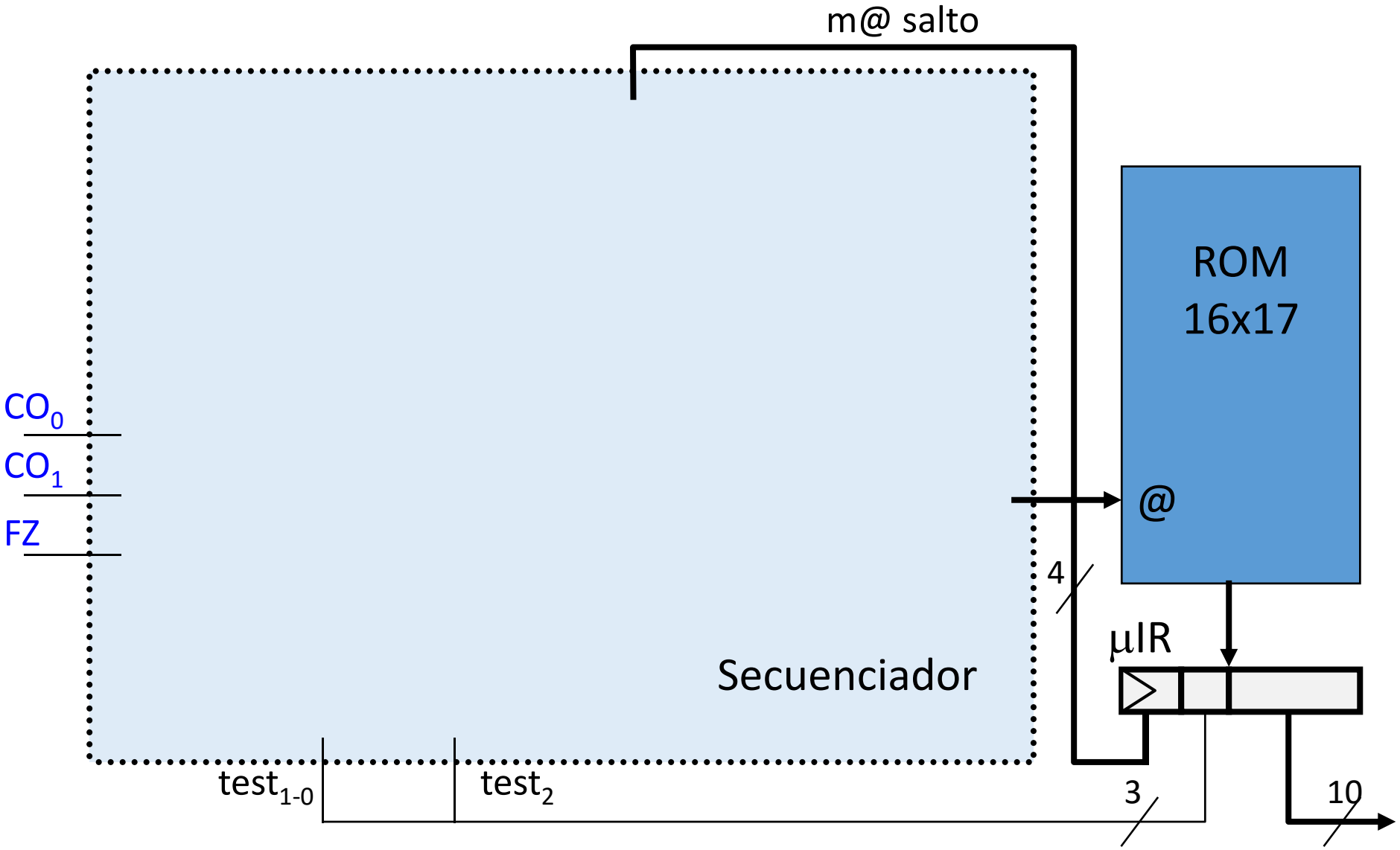


$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

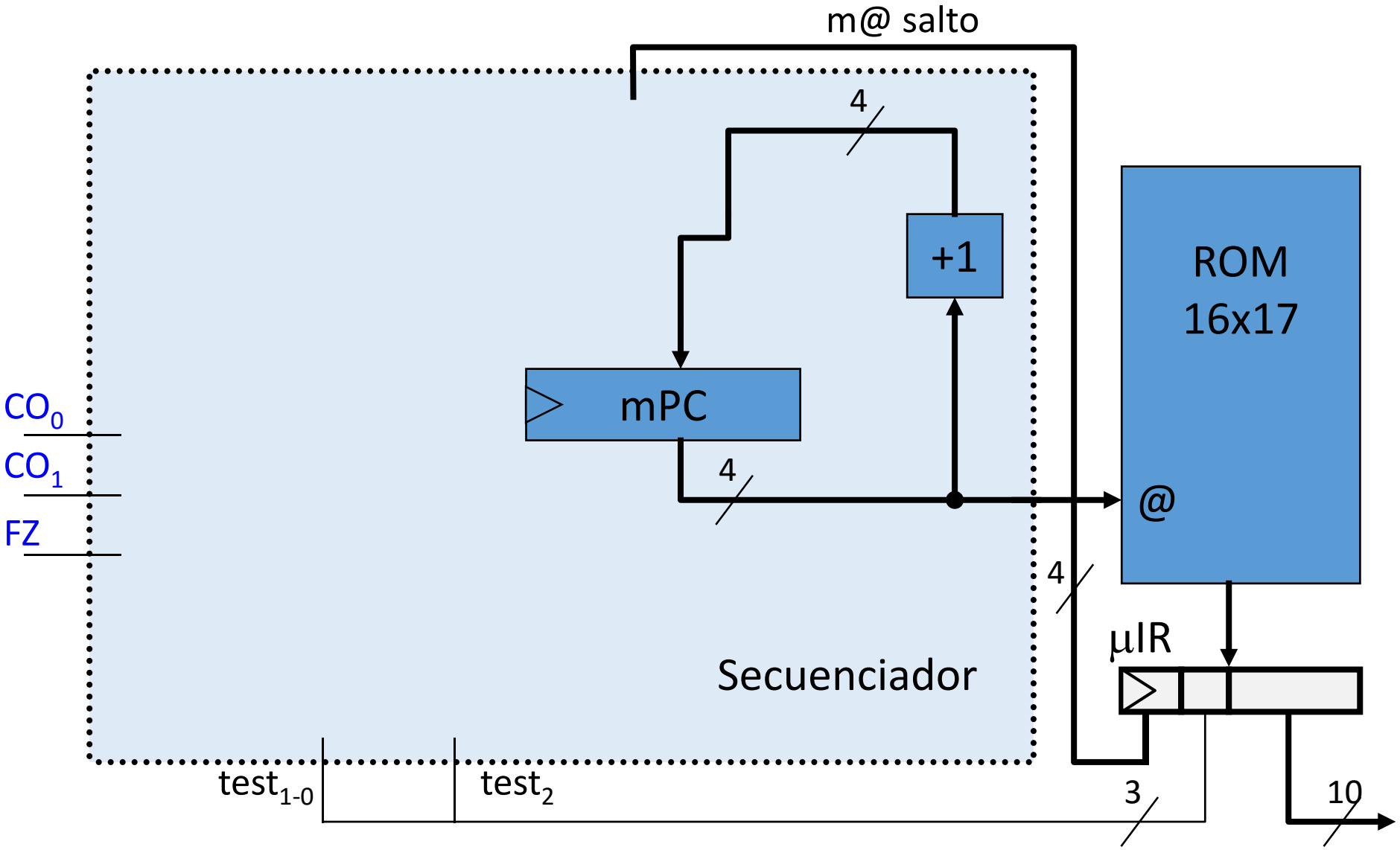
$$\mu PC^{+} = \begin{cases} \mu@ \text{ salto} \quad \underline{si} \quad bit(test_{1-0}) = test_2 \\ \mu PC + 1 \quad \underline{cc} \qquad\qquad (\neq) \end{cases}$$

- En "$test_{1-0}$" se codifica 1 bit
- Sólo hay 1 destino de salto
- ¿Porqué comparar $test_2$ con una constante?

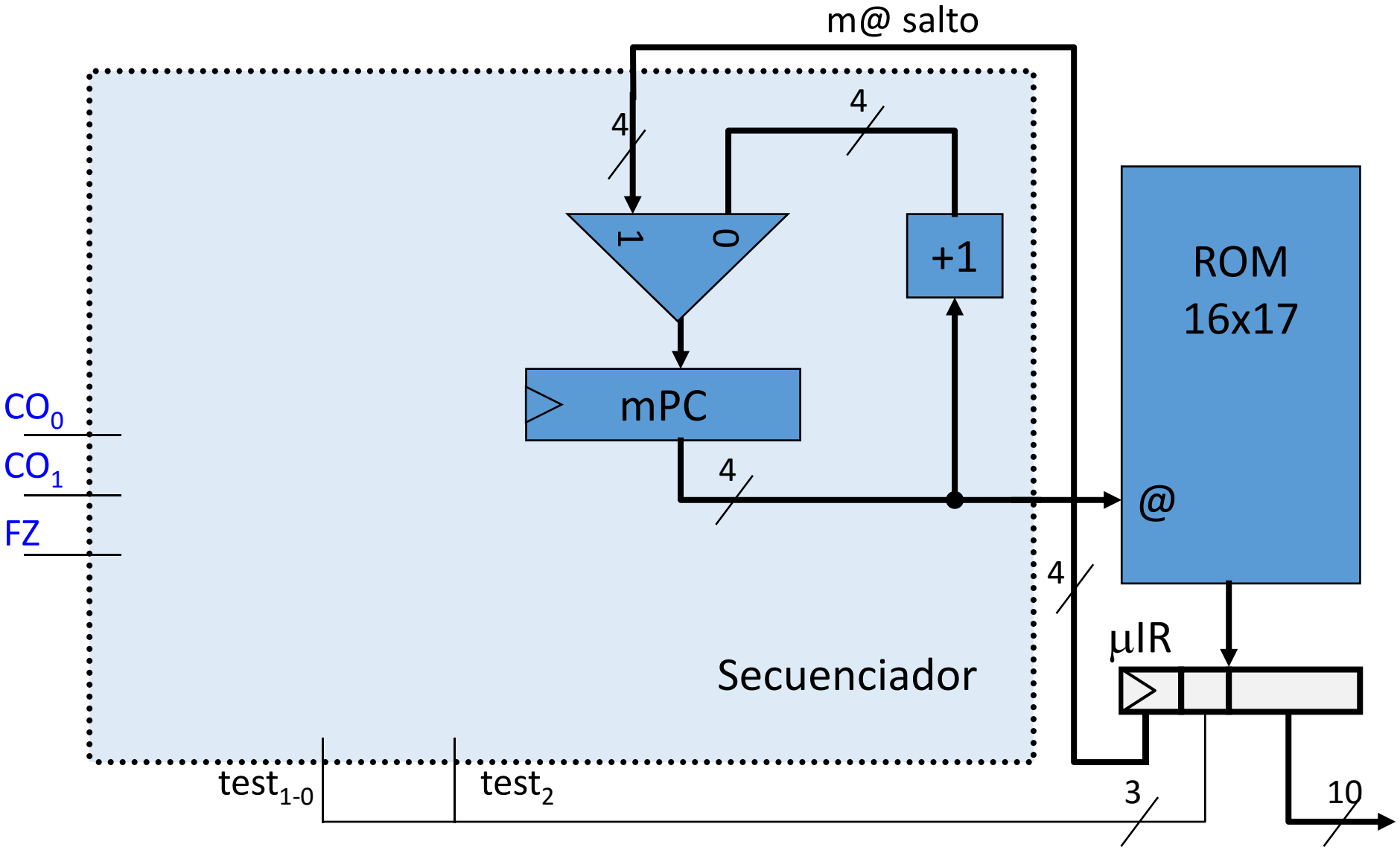
U.C. µprogramada



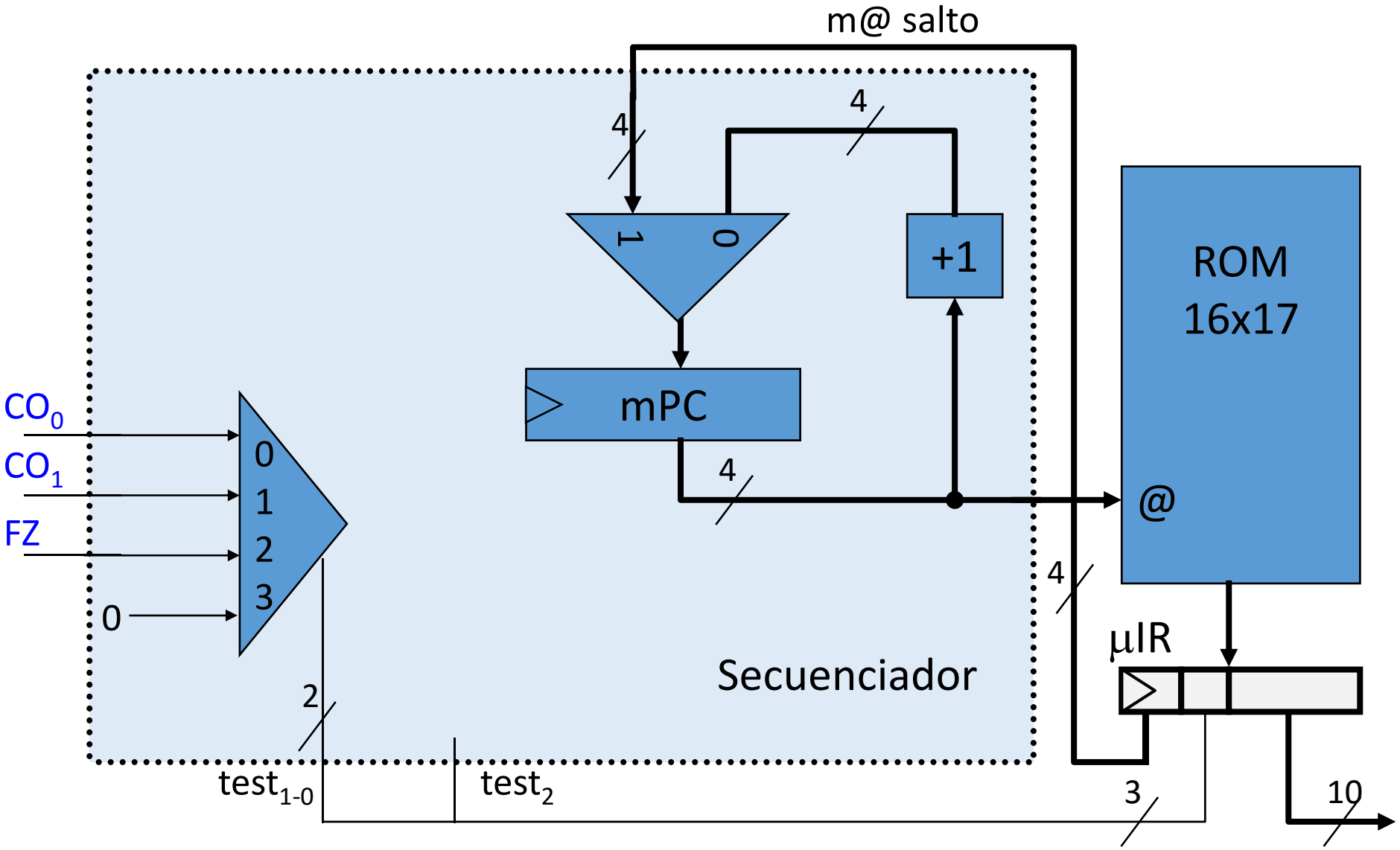
U.C. µprogramada



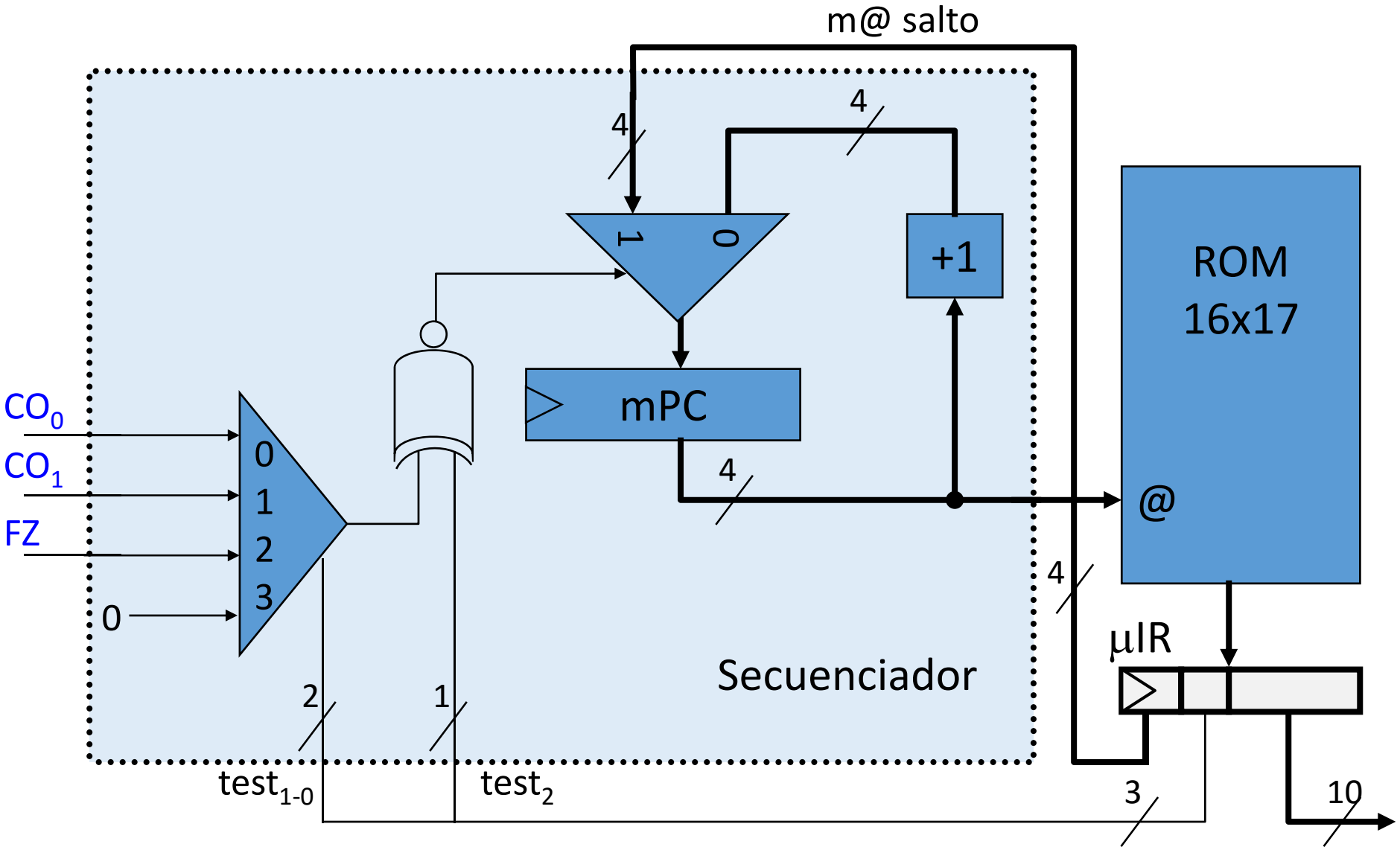
U.C. µprogramada



U.C. µprogramada



U.C. µprogramada



mprograma (ROM)

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X	X	0	0	0	0	0	0
3	XXXX	111	1	0	X	X	0	0	0	0	1	0
4	XXXX	111	1	1	X	X	0	0	0	1	0	0
5	0000	011	1	1	0	0	1	0	0	0	0	1
6	XXXX	111	1	0	X	X	0	0	0	0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	‘0’

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

0=1

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	‘0’

0=1

CO_0=1

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

0=1

CO_0=1

CO_1=1

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

0=1

CO_0=1

CO_1=1

0=0

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

0=1

CO_0=1

CO_1=1

0=0

CO_1=1

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

0=1

CO_0=1

CO_1=1

0=0

CO_1=1

FZ=0

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

0=1

CO_0=1

CO_1=1

ADD (00)

0=0

CO_1=1

FZ=0

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

0=1

CO_0=1

CO_1=1

ADD (00)

0=0

MOV (10)

CO_1=1

FZ=0

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

0=1

CO_0=1

CO_1=1

ADD (00)

0=0

MOV (10)

CO_1=1

CMP (01)

FZ=0

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

$0=1$

$CO_0=1$

$CO_1=1$

ADD (00)

$0=0$

MOV (10)

$CO_1=1$

CMP (01)

$FZ=0$

BEQ (11)

μ@	μ@ salto	test	mx_1	mx_0	alu_1	alu_0	$\overline{L}/E$	pc	ir	a	b	fz
0	XXXX	111	0	0	X	X	0	1	1	0	0	0
1	1000	100	X	X	X	X	0	0	0	0	0	0
2	0110	101	X	X	X					0	0	0
3	XXXX	111	1	0	X					0	1	0
4	XXXX	111	1	1	X					1	0	0
5	0000	011	1	1	0					0	0	1
6	XXXX	111	1	0	X					0	1	0
7	0000	011	1	1	1	0	1	0	0	0	0	1
8	1100	101	X	X	X	X	0	0	0	0	0	0
9	XXXX	111	1	0	X	X	0	0	0	0	1	0
10	XXXX	111	1	1	X	X	0	0	0	1	0	0
11	0000	011	X	X	0	1	0	0	0	0	0	1
12	0000	010	X	X	X	X	0	0	0	0	0	0
13	0001	011	1	1	X	X	0	1	1	0	0	0

$test_{1-0}$	bit
00	CO_0
01	CO_1
10	FZ
11	'0'

0=1

CO_0=1

CO_1=1

ADD (00)

0=0

MOV (10)

CO_1=1

CMP (01)

FZ=0

BEQ (11)

diagrama, RTL y nº ciclos