Problemas Tema 5:

Diseño monociclo del procesador

Fernando Castro Rodríguez
José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

2) En el procesador **monociclo** estudiado en clase, justifica los valores que toman las señales de control para la ejecución de las instrucciones “lw”.

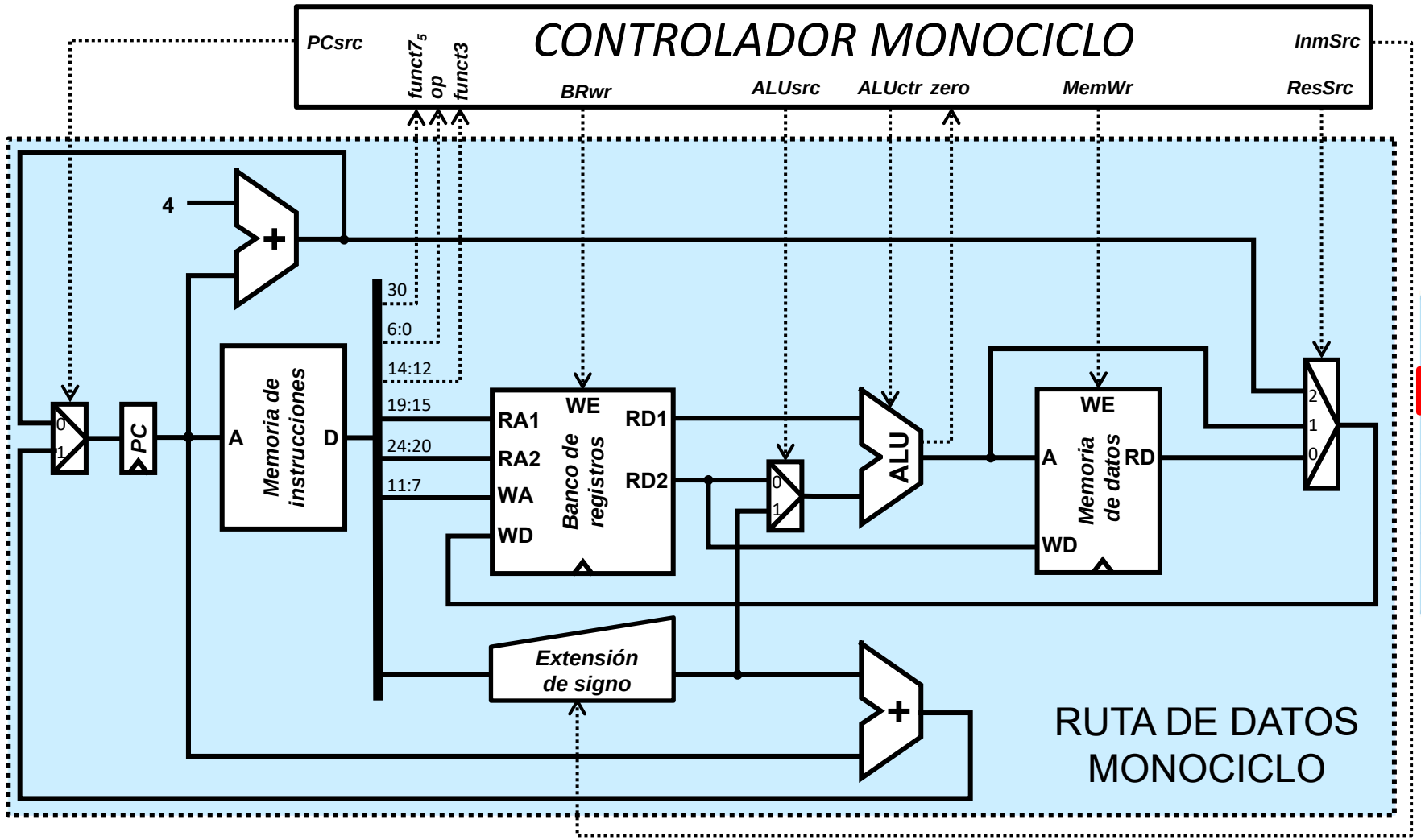


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

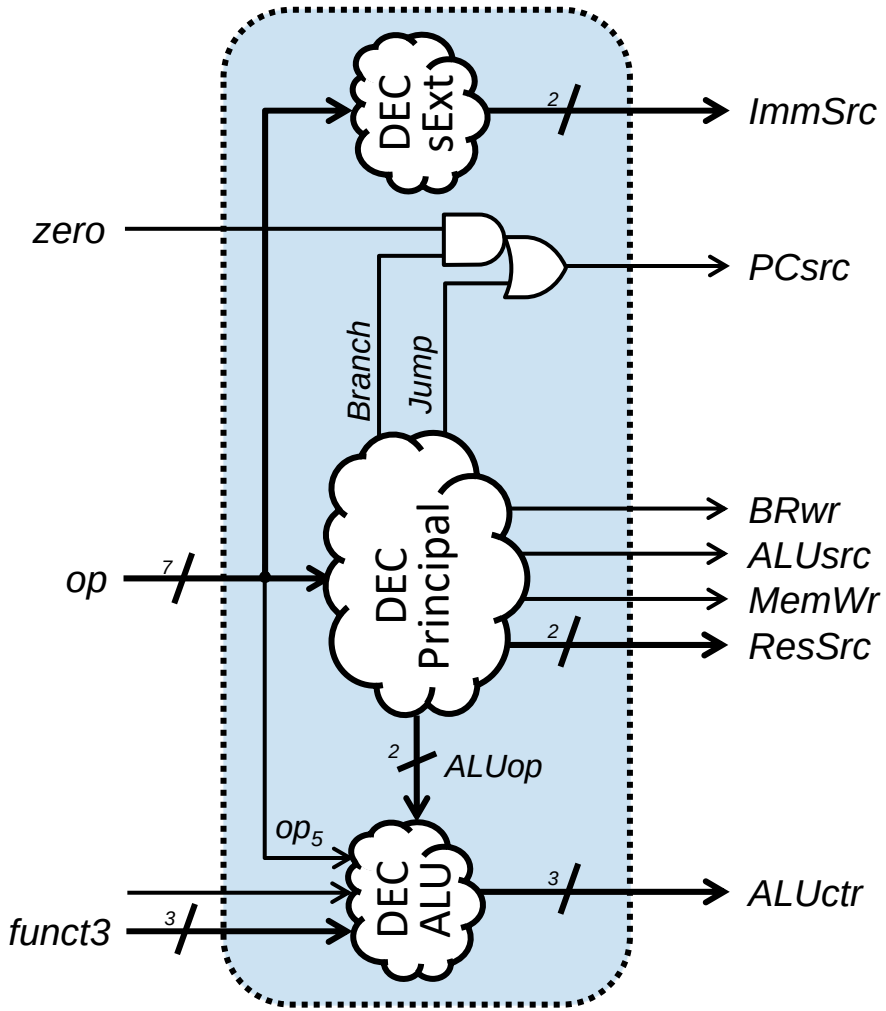
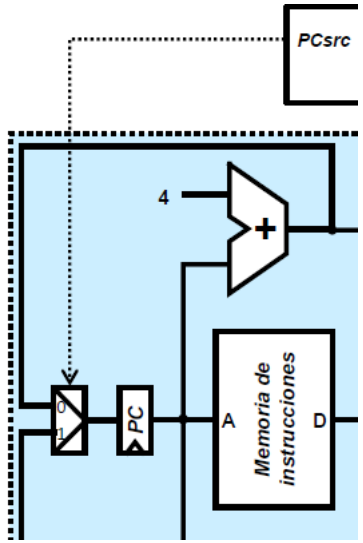


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- Las señales Branch y Jump están a 0 porque no se trata de una instrucción de salto condicional (beq) ni de salto incondicional (jal), por lo que la señal PCsrc será también 0, lo que indica que el PC se actualizará con el valor PC+4 (siguiente instrucción al “lw”)



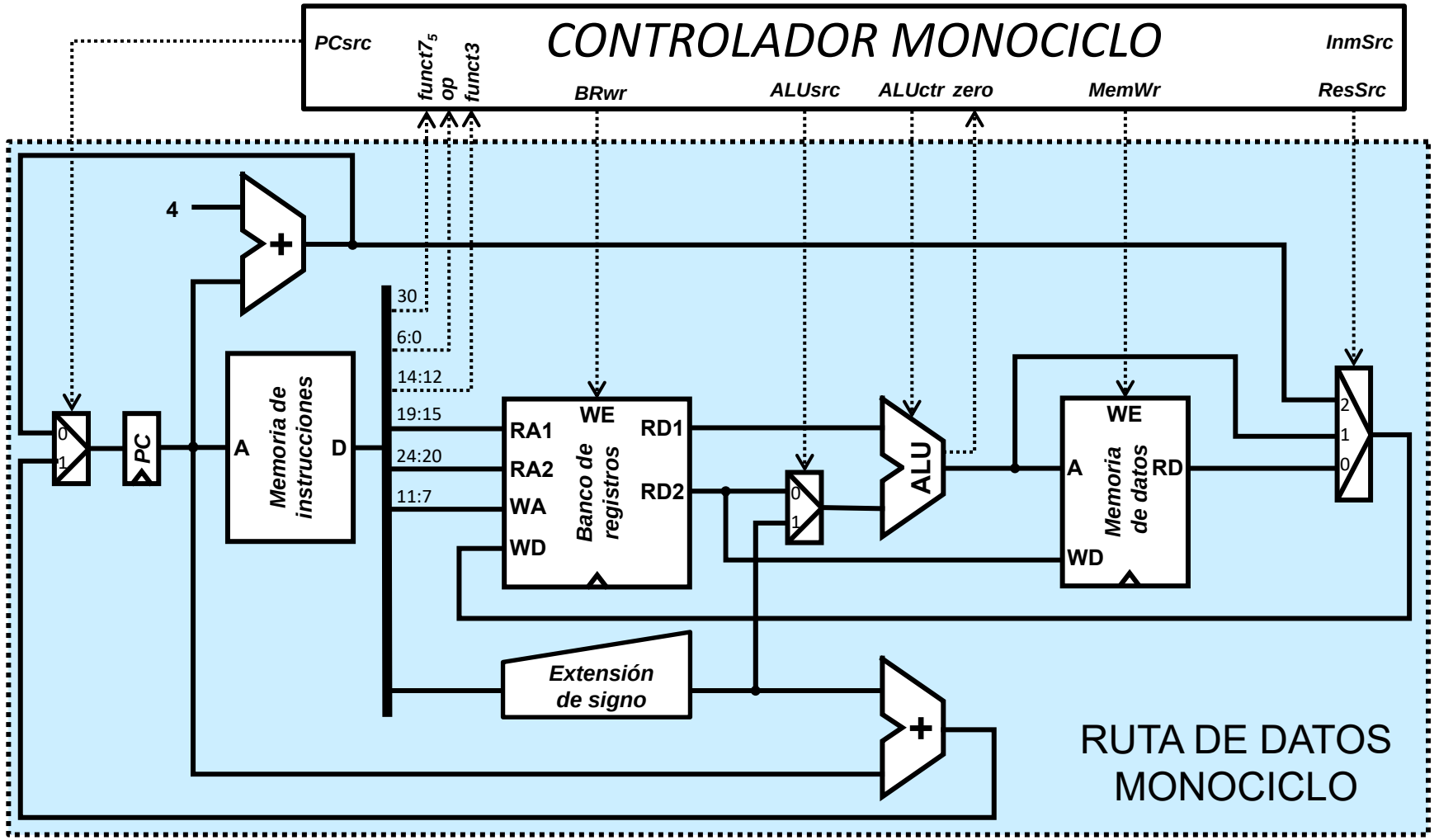


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- Las señales BRwr y MemWr están a 1 y a 0 respectivamente porque las instrucciones “lw” sí van a escribir en el banco de registros (el dato que leen de la memoria lo guardan en el registro destino), pero no escriben en memoria (de la que únicamente leen sin modificarla)

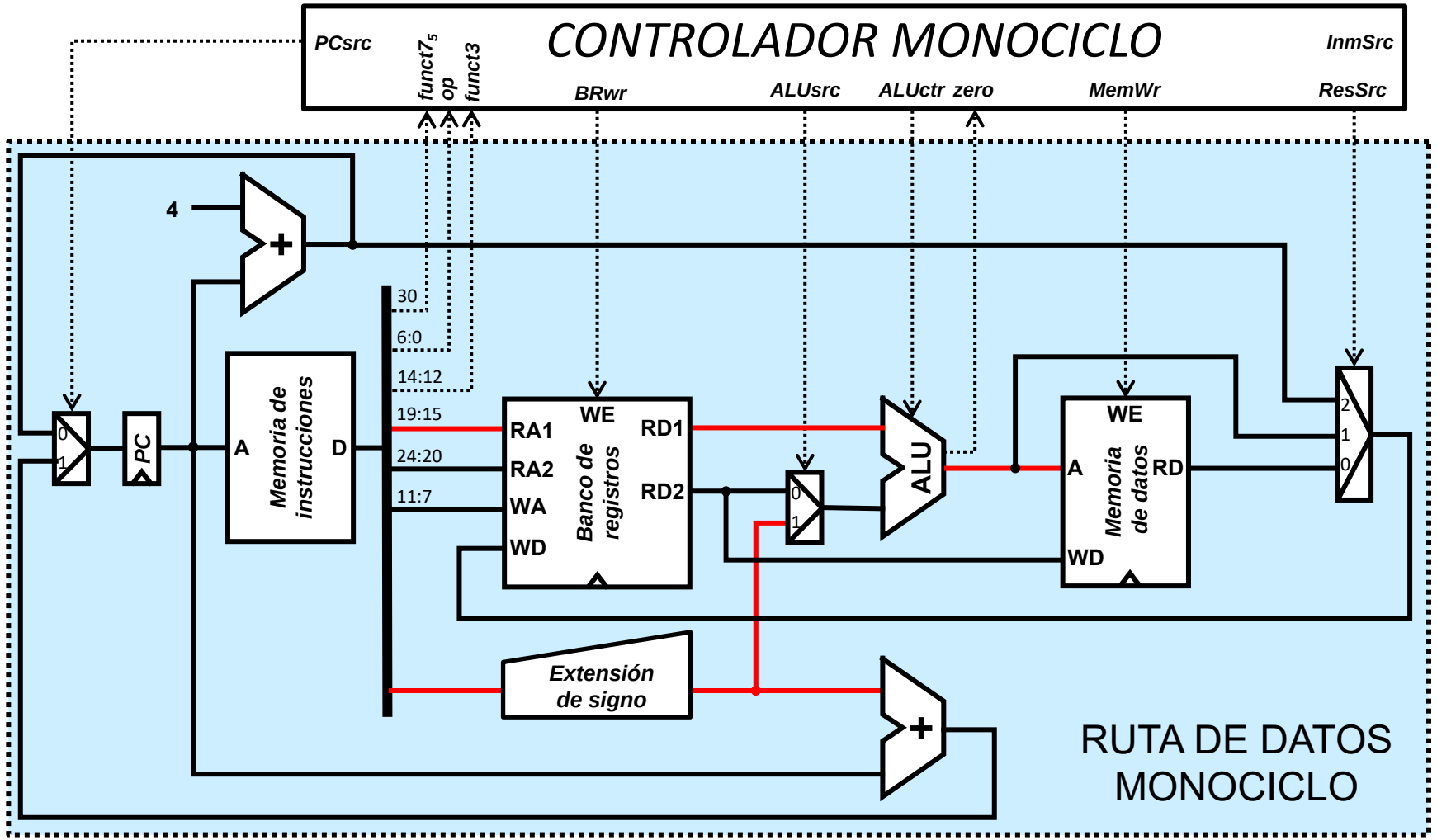


Tabla de verdad

op	Branch	Jump	RegWr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

Tabla de verdad

ALUop	op_5	$funct7_5$	funct3	ALUctr
00 (sumar)	X	X	XXX	000 (A + B)
01 (restar)	X	X	XXX	001 (A – B)
10 (operar)	0	X	000 (addi)	000 (A + B)
10 (operar)	1	0	000 (add)	000 (A + B)
10 (operar)	1	1	000 (sub)	001 (A – B)
10 (operar)	X	X	010 (slt/slti)	101 (A < B)
10 (operar)	X	X	110 (or/ori)	011 (A | B)
10 (operar)	X	X	111 (and/andi)	010 (A & B)

- La señal ALUsrc (que controla la procedencia del valor que se suministra a la entrada inferior de la ALU) debe valer 1, ya que al contenido del registro fuente rs1 se le suma un inmediato tras haberle realizado la extensión del signo. La ALU realiza entonces la operación de suma necesaria para calcular la dirección de acceso a memoria, de la cual leerá el dato. Para llevar a cabo esta operación de suma en la ALU, ALUop toma el valor 00 (al igual que en las operaciones de store en donde el cálculo de la dirección de acceso a memoria se realiza de la misma forma), tomando ALUctr el valor 000.

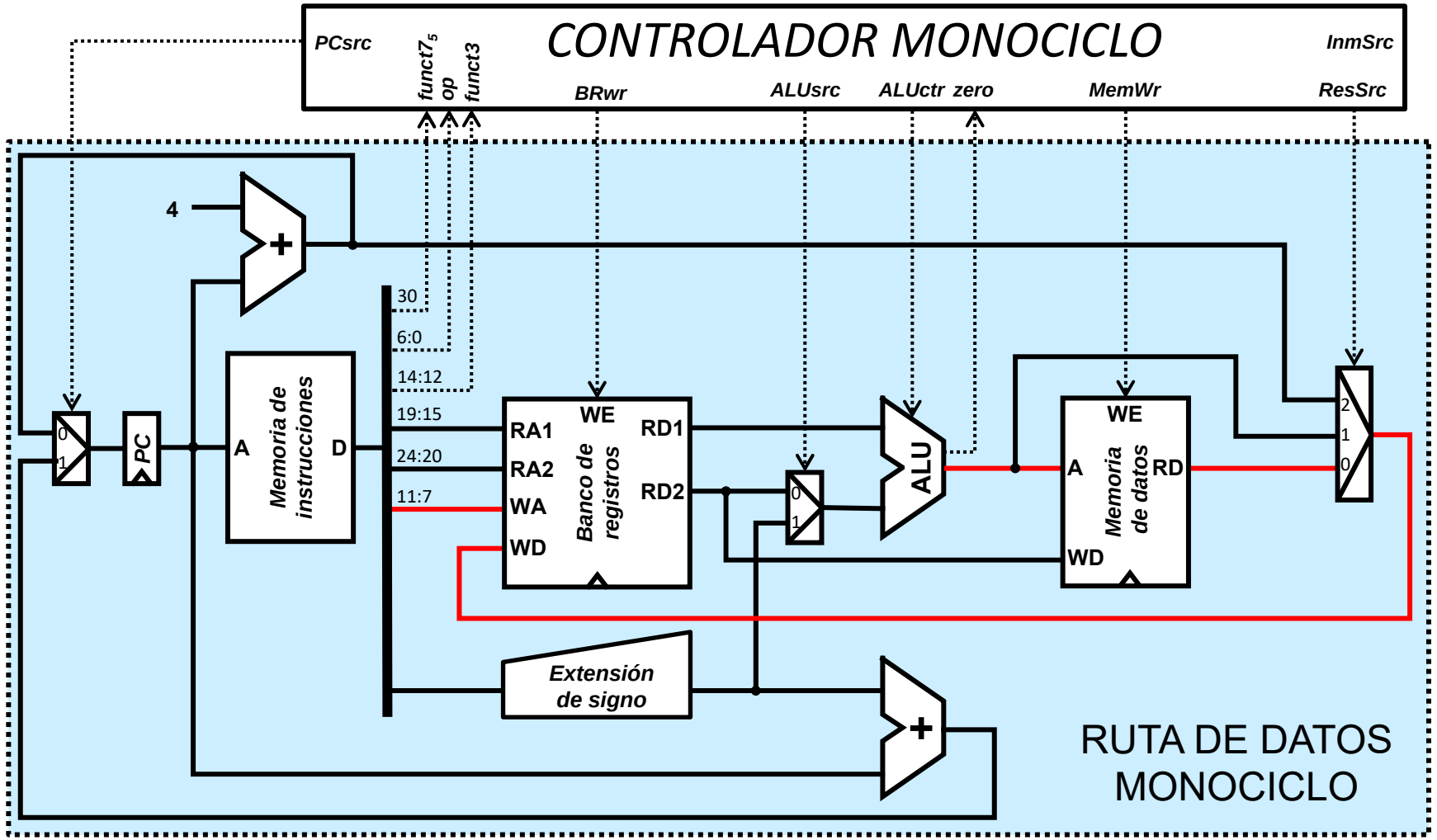


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- Por último, la señal ResSrc, que controla el origen del dato que se va a escribir en el banco de registros, toma el valor 00 para indicar que lo que se va a escribir sobre el registro destino (determinado por los bits 11:7 de la instrucción), es el dato leído de la memoria de datos.

3) En el procesador **monociclo** estudiado en clase, justifica los valores que toman las señales de control para la ejecución de las instrucciones “andi”.

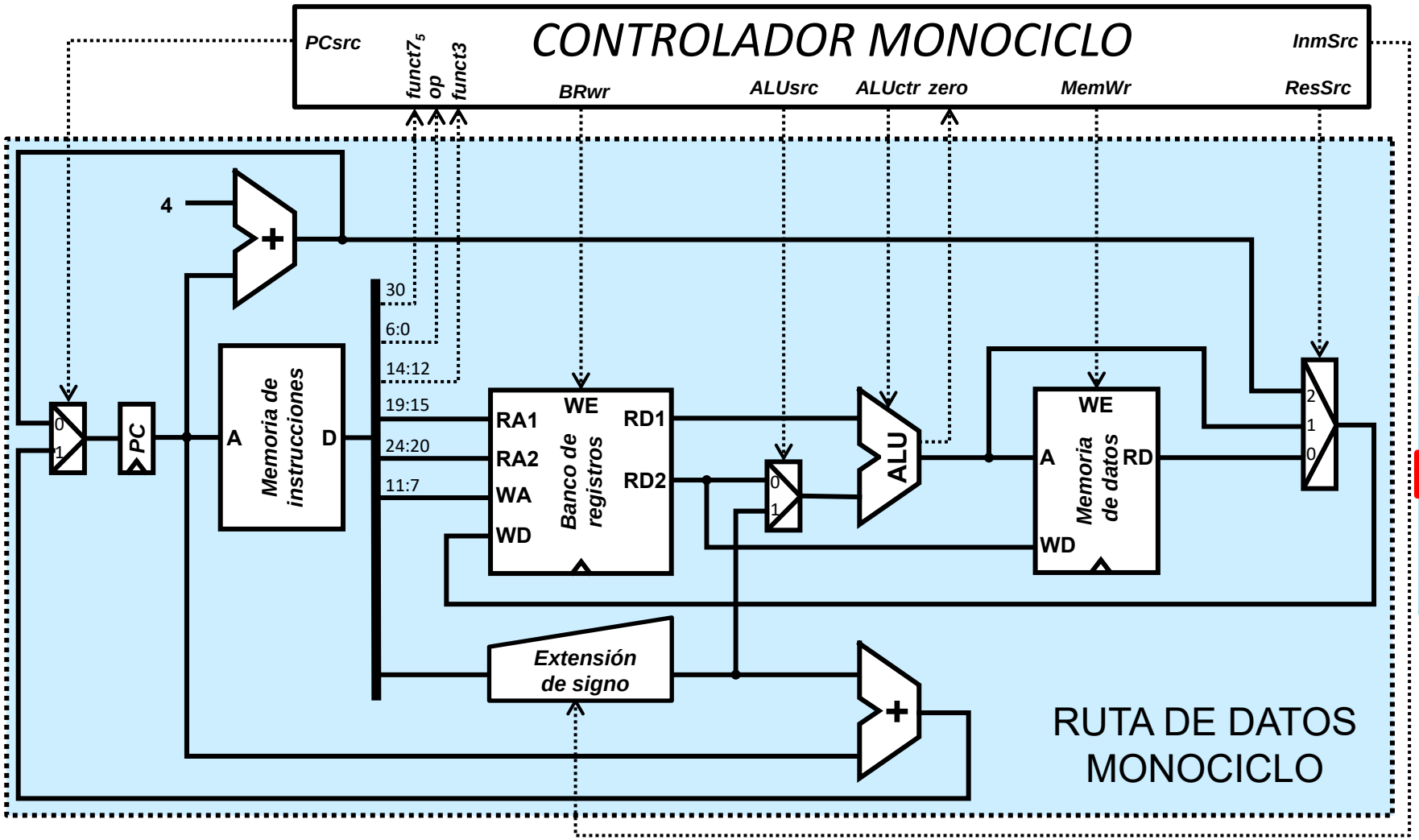


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

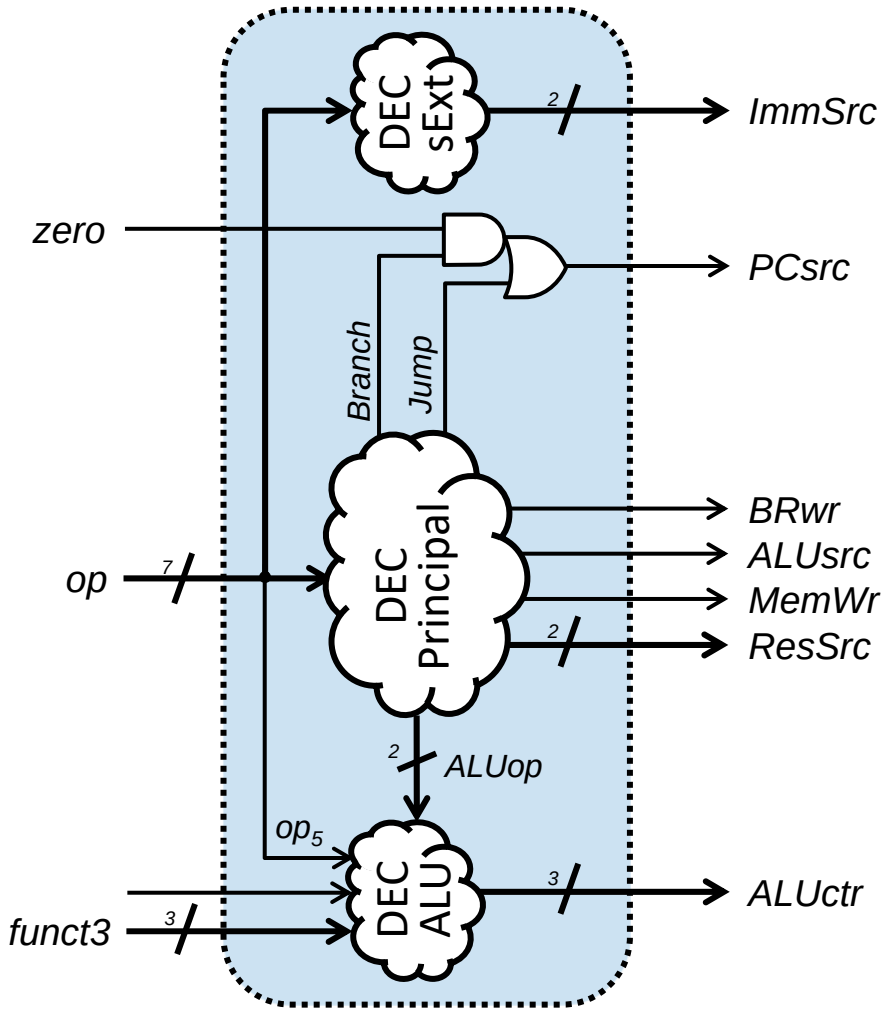
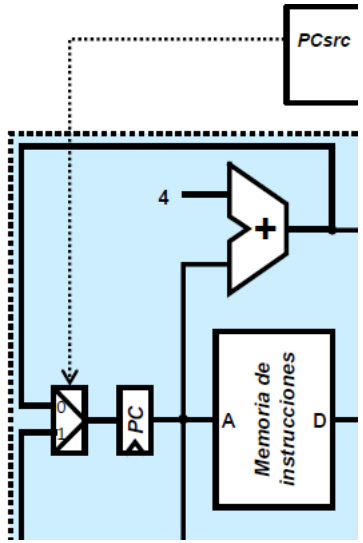


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- Las señales Branch y Jump están a 0 porque no se trata de una instrucción de salto condicional (beq) ni de salto incondicional (jal), por lo que la señal PCsrc será también 0, lo que indica que el PC se actualizará con el valor PC+4 (siguiente instrucción a “andi”)



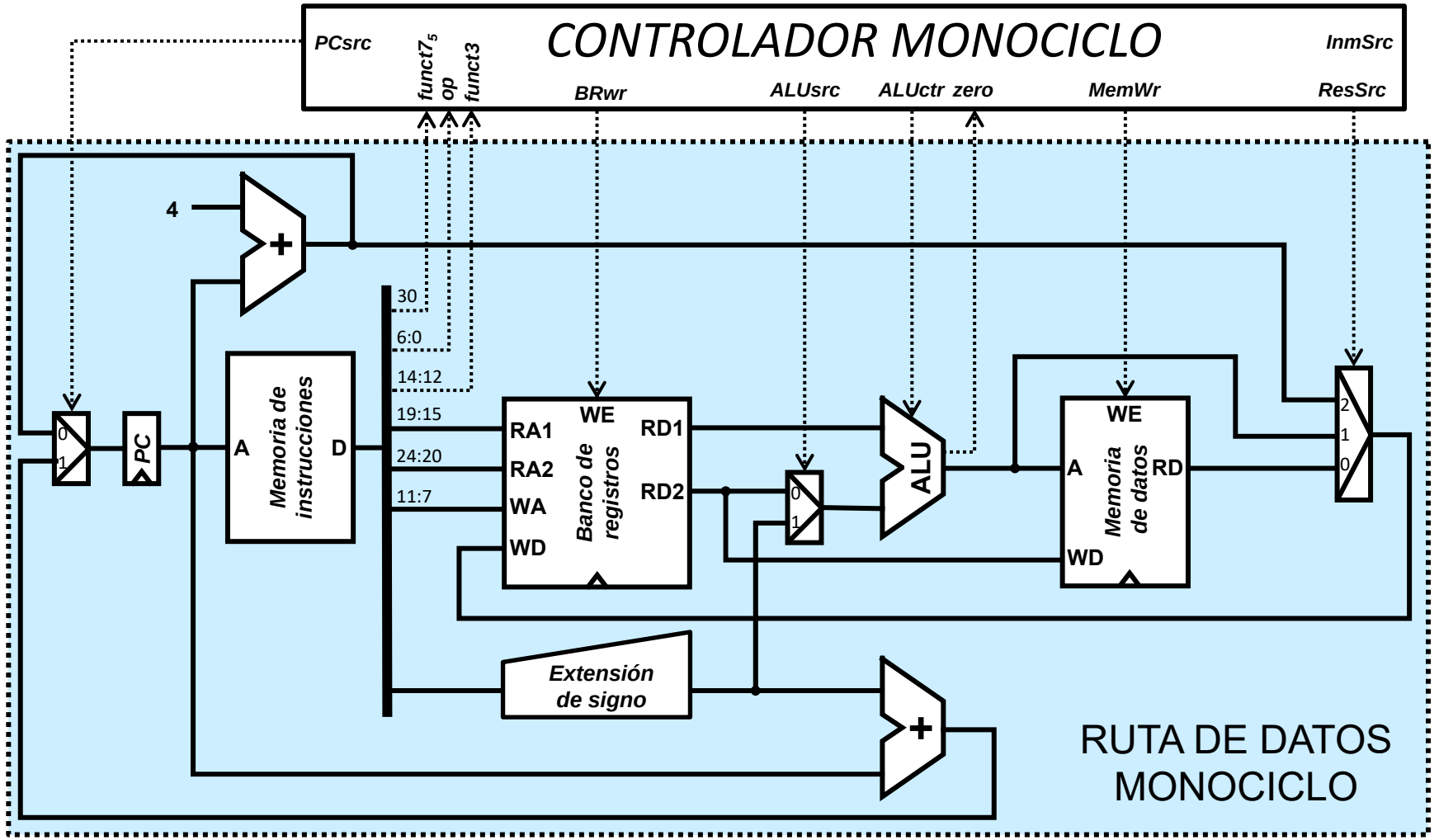


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- Las señales BRwr y MemWr están a 1 y a 0 respectivamente porque las instrucciones “andi” sí van a escribir en el banco de registros (el dato que generan tras realizar la operación and del contenido de un registro con un inmediato), pero no escriben (ni leen) en memoria.

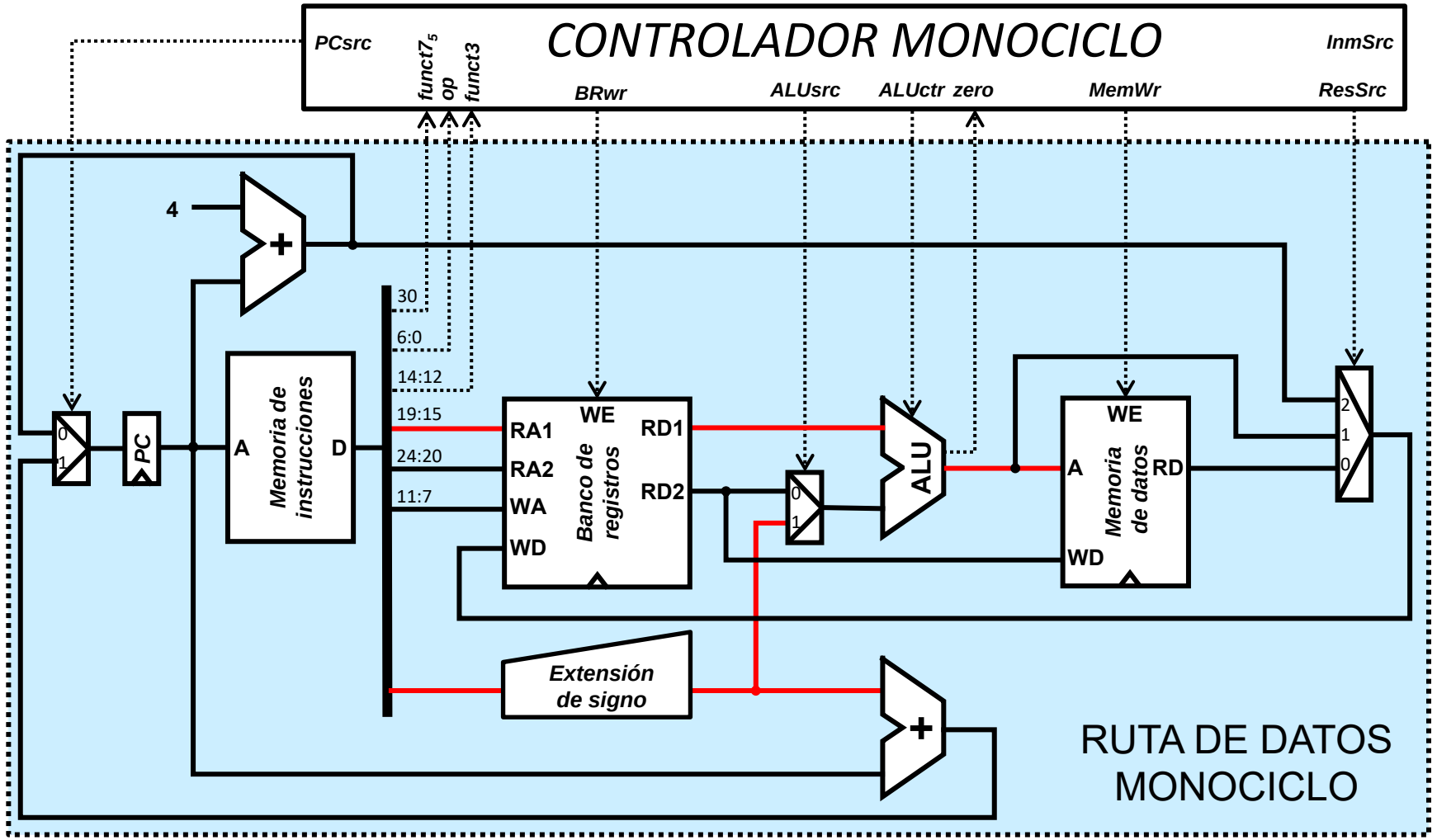


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

Tabla de verdad

ALUop	op_5	$funct7_5$	funct3	ALUctr
00 (sumar)	X	X	XXX	000 (A + B)
01 (restar)	X	X	XXX	001 (A – B)
10 (operar)	0	X	000 (addi)	000 (A + B)
10 (operar)	1	0	000 (add)	000 (A + B)
10 (operar)	1	1	000 (sub)	001 (A – B)
10 (operar)	X	X	010 (slt/slti)	101 (A < B)
10 (operar)	X	X	110 (or/ori)	011 (A | B)
10 (operar)	X	X	111 (and/andi)	010 (A & B)

- La señal ALUsrc (que controla la procedencia del valor que se suministra a la entrada inferior de la ALU) debe valer 1, ya que al contenido del registro fuente rs1 se le realiza la operación and con un inmediato tras haberle realizado la extensión del signo. Para llevar a cabo esta operación de and lógica en la ALU, ALUop toma el valor 10 y ALUctr toma el valor 010

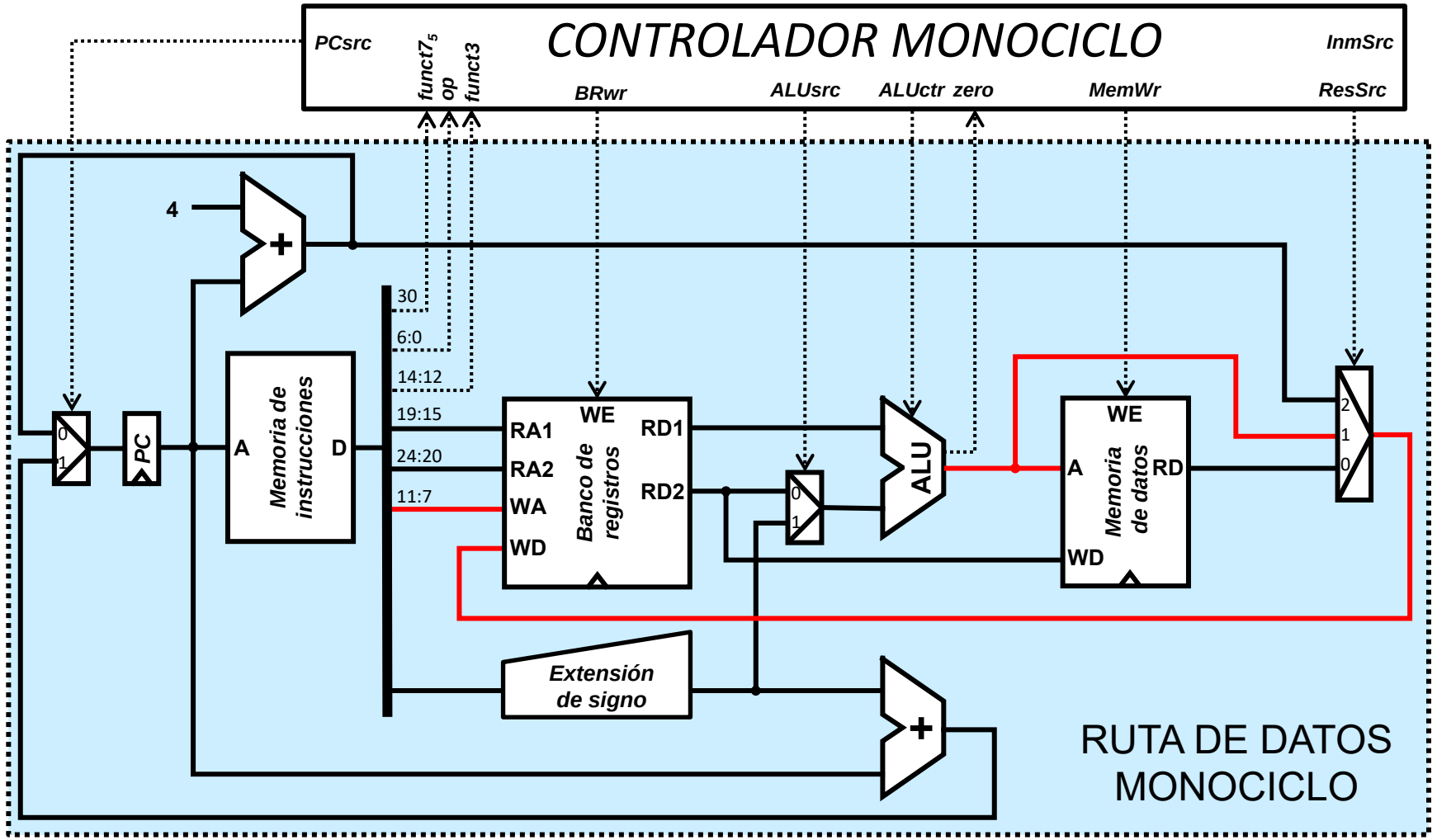


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- Por último, la señal ResSrc, que controla el origen del dato que se va a escribir en el banco de registros, toma el valor 01 para indicar que lo que se va a escribir sobre el registro destino (determinado por los bits 11:7 de la instrucción), es el dato que genera la ALU tras realizar la operación lógica “and”.

4) En el procesador **monociclo** estudiado en clase, justifica los valores que toman las señales de control para la ejecución de las instrucciones “beq”.

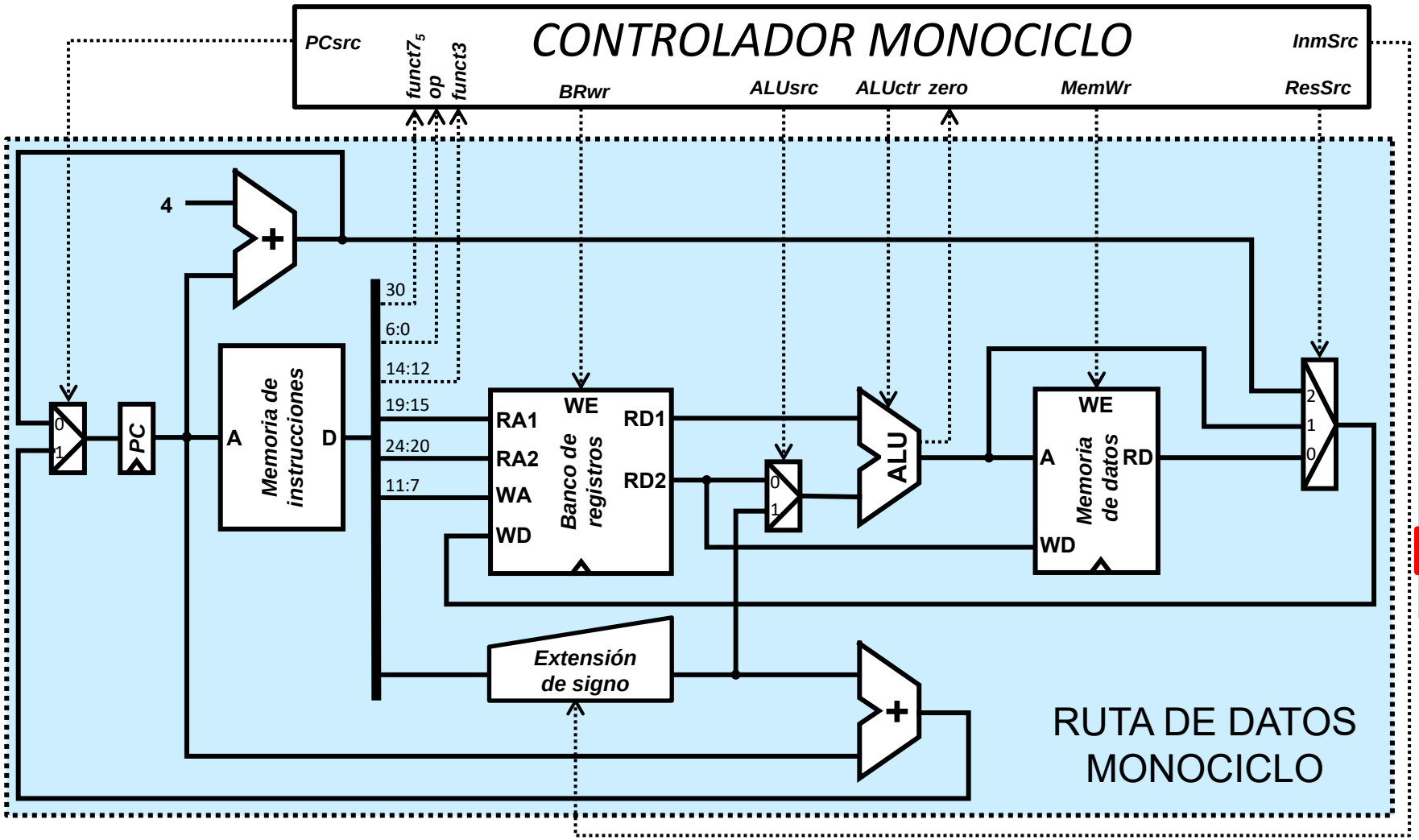


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

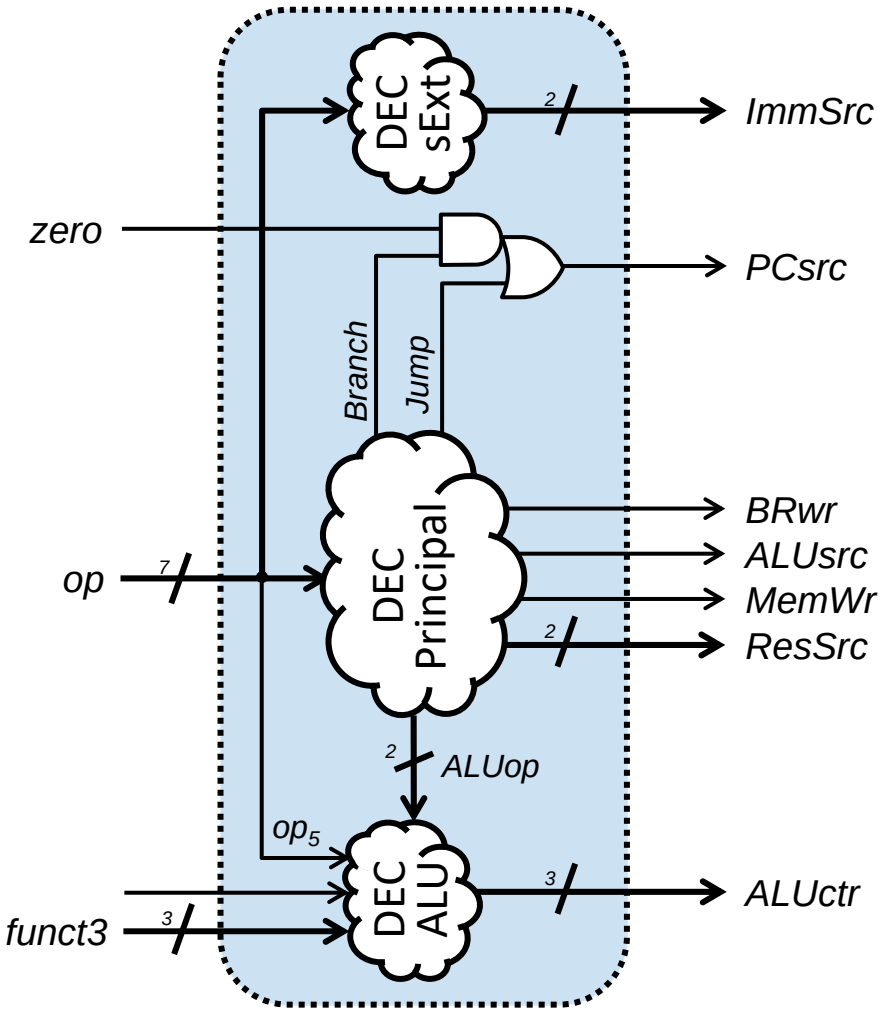
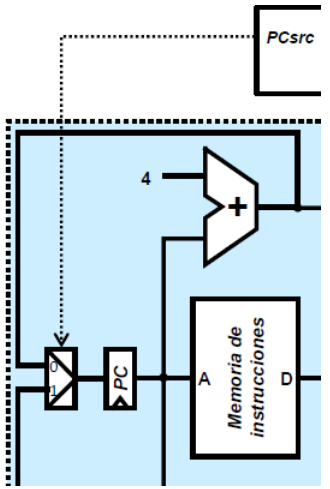


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- La señal Branch está a 1 porque se trata de una instrucción de salto condicional (beq), mientras que la de Jump está a 0 porque no se trata de un salto incondicional (jal). De este modo, el valor de la señal PCsrc dependerá lo que valga la señal de estado "zero", que a su vez dependerá de que se cumpla o no la condición de igualdad evaluada:
- Si "zero" = 1 → la condición de igualdad se cumple, por lo que PCsrc=1 y el PC se actualiza con la dirección destino del salto (PC + SExt(inmediato) generada por el sumador situado en la parte inferior de la ruta de datos
- Si "zero" = 0 → la condición de igualdad no se cumple, por lo que PCsrc=0 y el PC se actualizará con el valor PC+4 (siguiente instrucción a "beq")



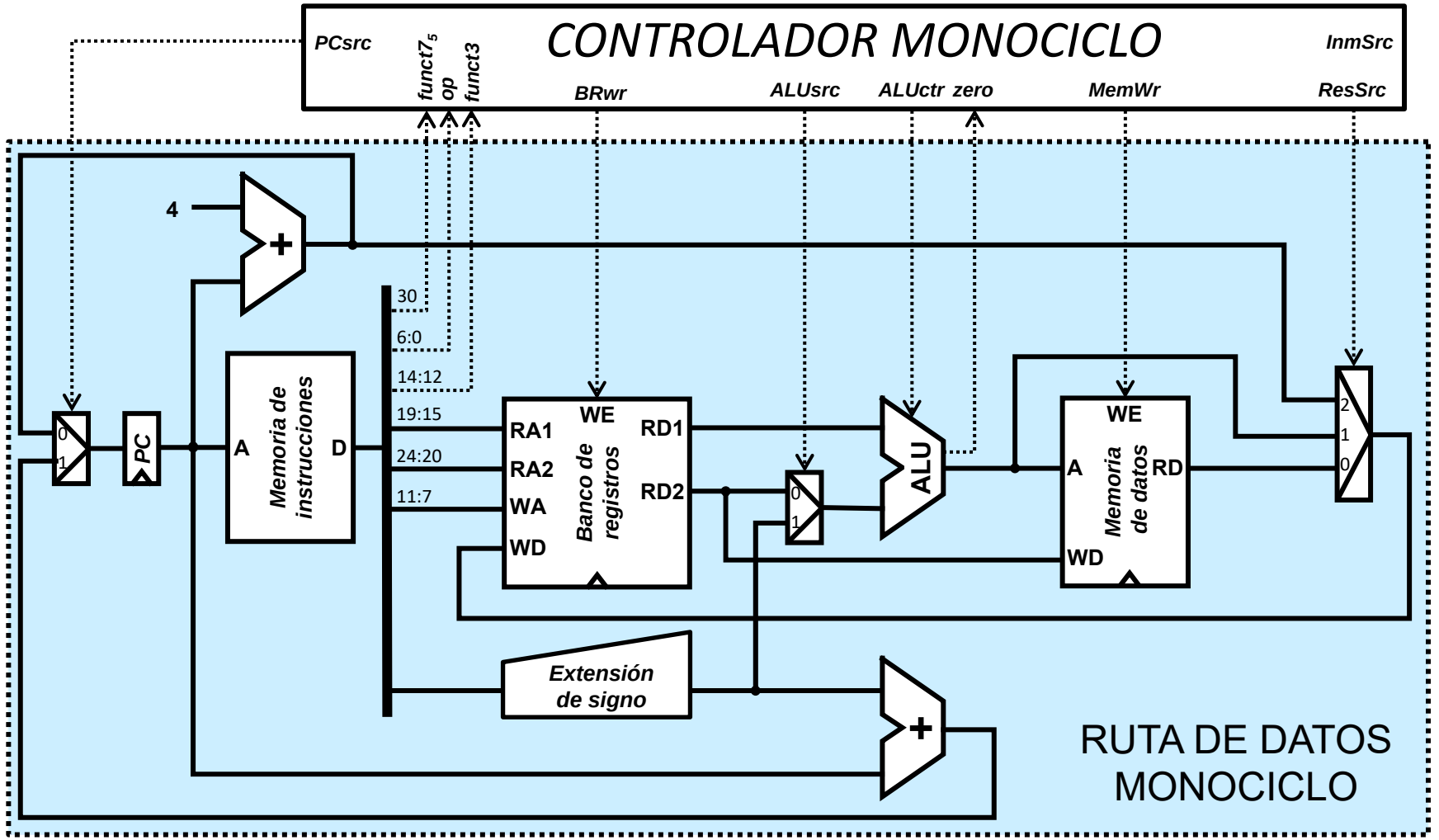


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- Las señales BRwr y MemWr están a 0 en ambos casos porque las instrucciones “beq” no van a escribir sobre ningún registro ni sobre la memoria

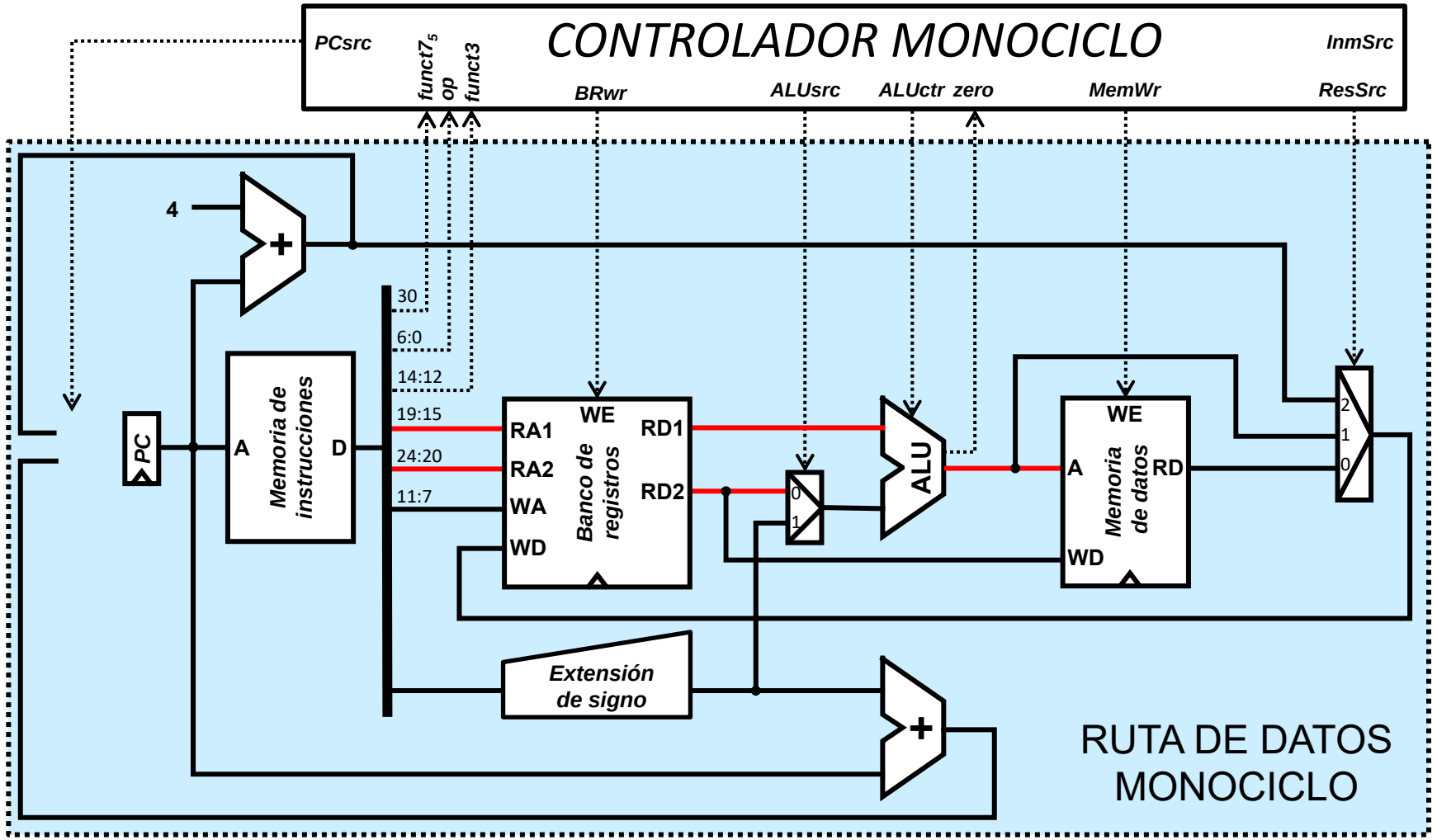


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

Tabla de verdad

ALUop	op_5	$funct7_5$	funct3	ALUctr
00 (sumar)	X	X	XXX	000 (A + B)
01 (restar)	X	X	XXX	001 (A – B)
10 (operar)	0	X	000 (addi)	000 (A + B)
10 (operar)	1	0	000 (add)	000 (A + B)
10 (operar)	1	1	000 (sub)	001 (A – B)
10 (operar)	X	X	010 (slt/slti)	101 (A < B)
10 (operar)	X	X	110 (or/ori)	011 (A | B)
10 (operar)	X	X	111 (and/andi)	010 (A & B)

- La señal ALUsrc (que controla la procedencia del valor que se suministra a la entrada inferior de la ALU) debe valer 0, ya que al contenido del registro fuente rs1 se le debe restar el contenido del registro fuente rs2 para poder evaluar la condición de igualdad entre ambos registros. Para llevar a cabo esta resta en la ALU, ALUop toma el valor 01 y ALUctr toma el valor 001

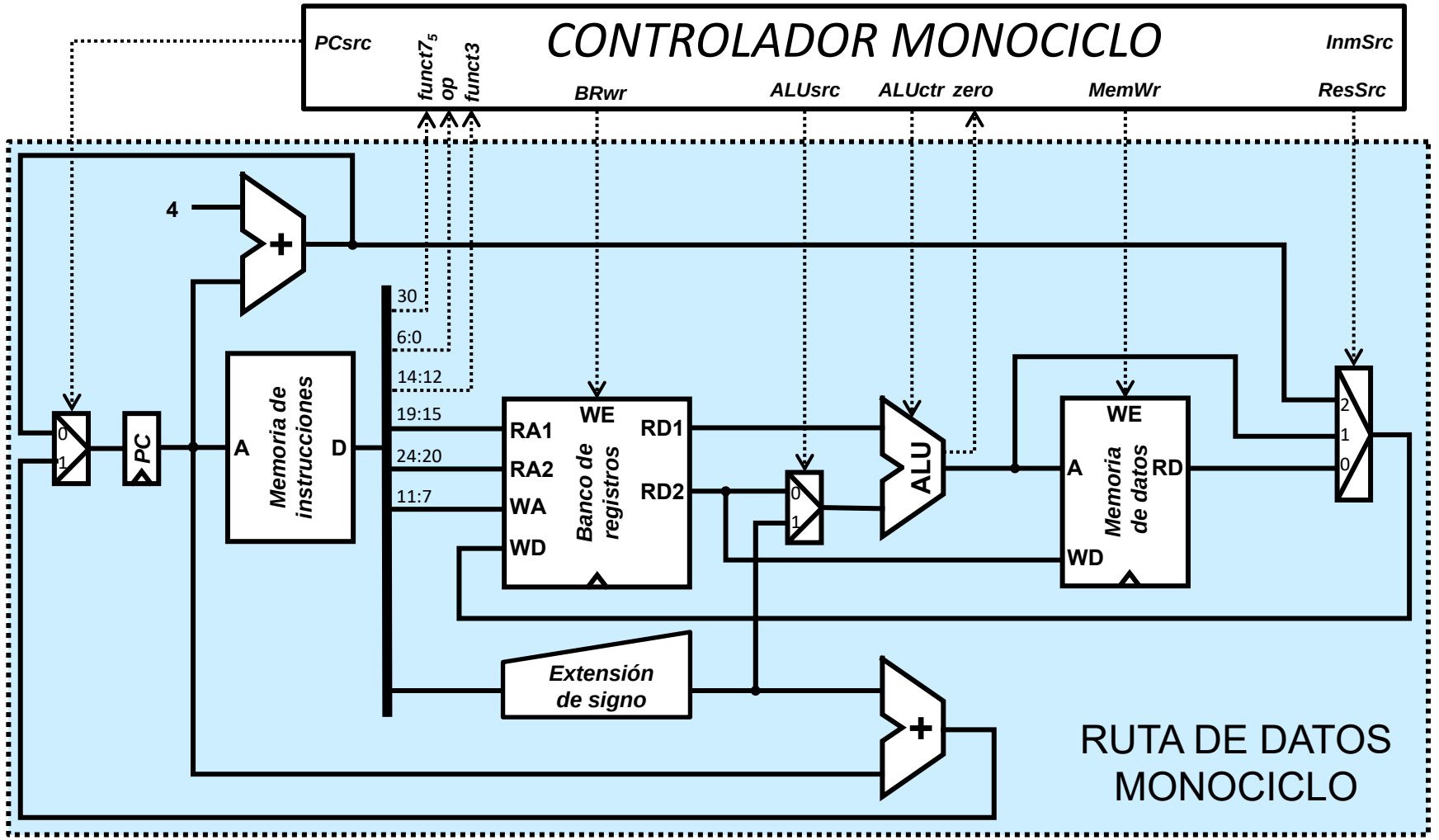


Tabla de verdad

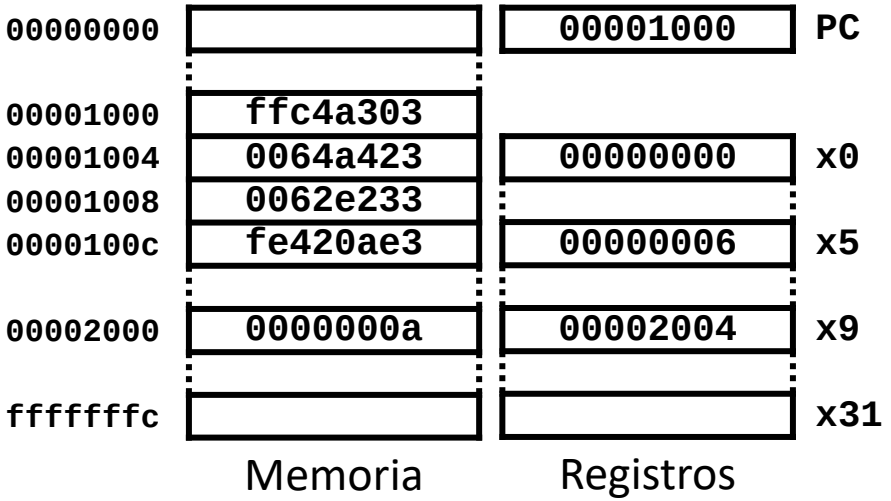
op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

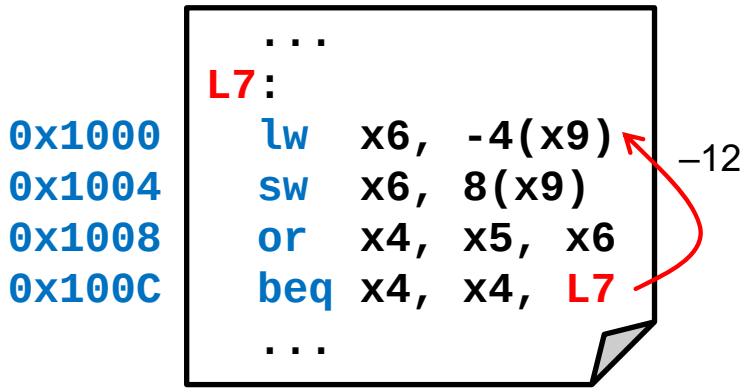
- Por último, es indiferente el valor que tome la señal ResSrc (que controla el origen del dato que se va a escribir en el banco de registros), ya que la instrucción “beq” no escribe en banco de registros (BRwr=0)

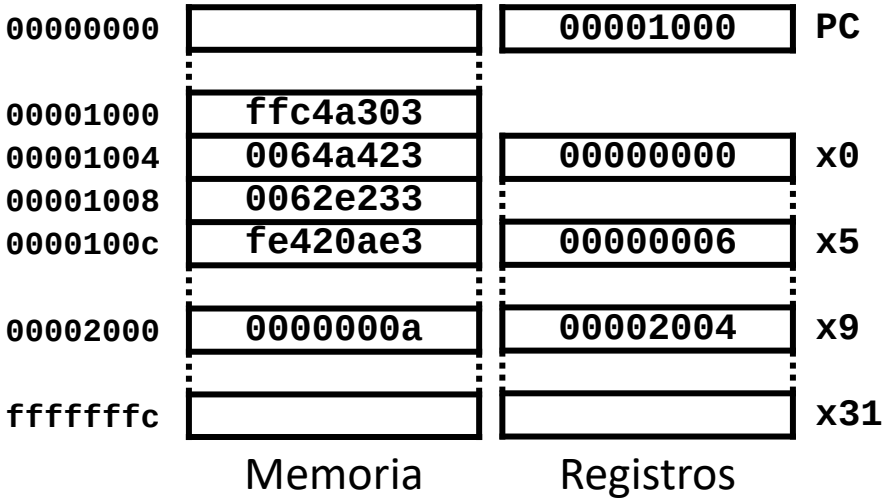
8) Supongamos que sobre el procesador **monociclo** estudiado en clase se quiere ejecutar el programa que se muestra a continuación, siendo el contenido inicial de los registros y la memoria el mostrado en la figura. Representa los diagramas de ejecución para los registros y la memoria, así como para las señales de estado y de control, de modo que se visualicen sus valores en cada ciclo de reloj correspondiente a la ejecución del programa.

```
          ...
L7:
0x1000    lw   x6, -4(x9)
0x1004    sw   x6, 8(x9)
0x1008    or   x4, x5, x6
0x100C    beq  x4, x4, L7
          ...
```

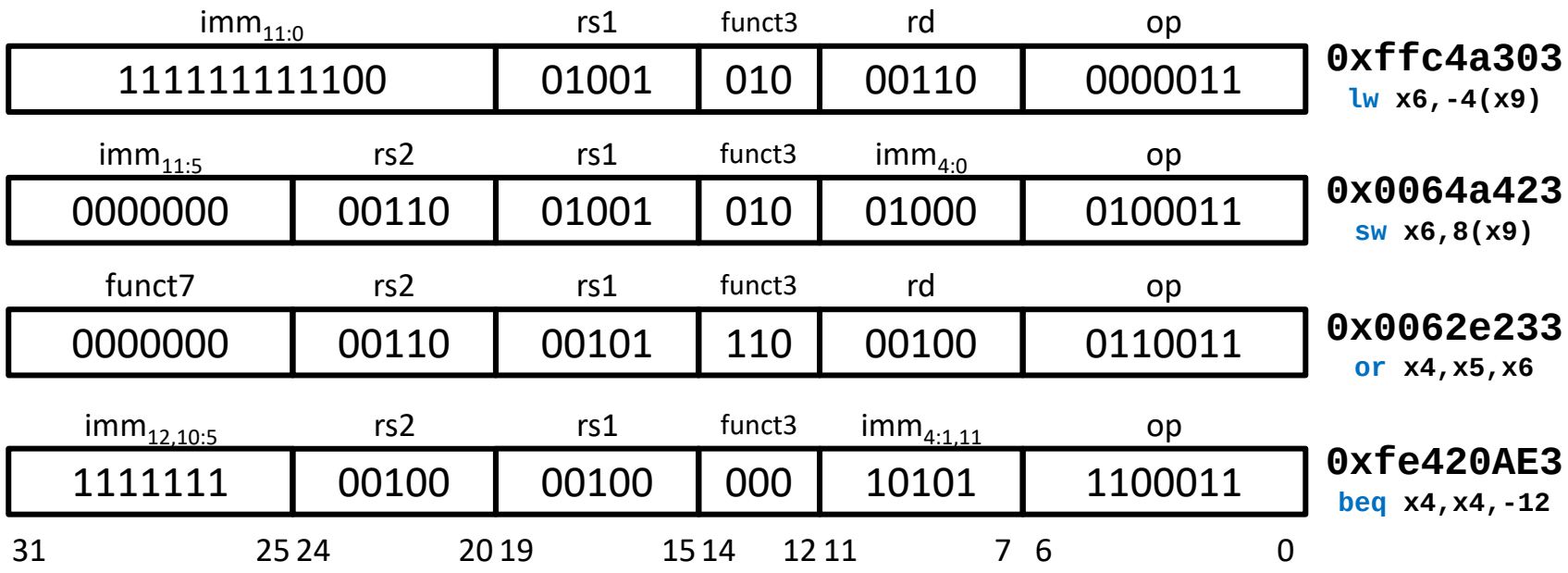
–12

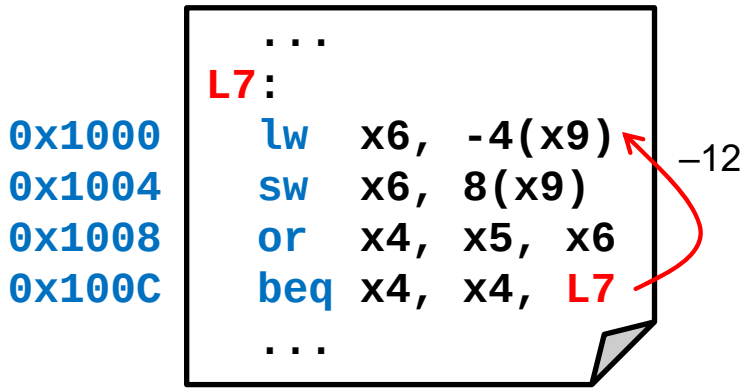






La representación en código máquina de las instrucciones que conforman el programa es la siguiente:





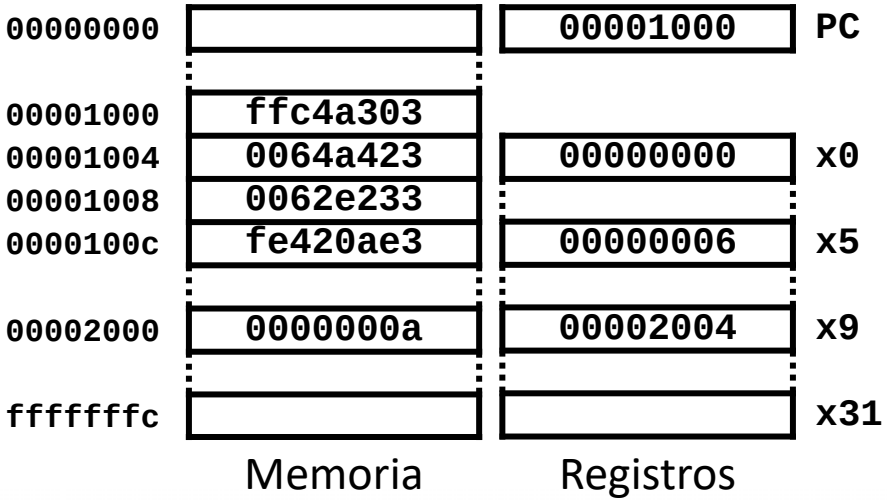
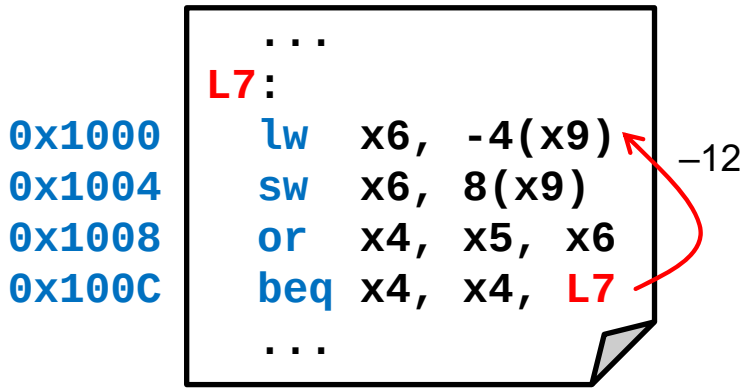


Diagrama de ejecución: registros y memoria

	lw x6,-4(x9)	sw x6,8(x9)	or x4,x5,x6	beq x4,x4,-12	lw x6,-4(x9)
PC	00001000				
x4	????????				
x5	00000006				
x6	????????				
x9	00002004				
MEM[0x2000]	0000000a				
MEM[0x200c]	????????				
clk					
# ciclo	1	2	3	4	5



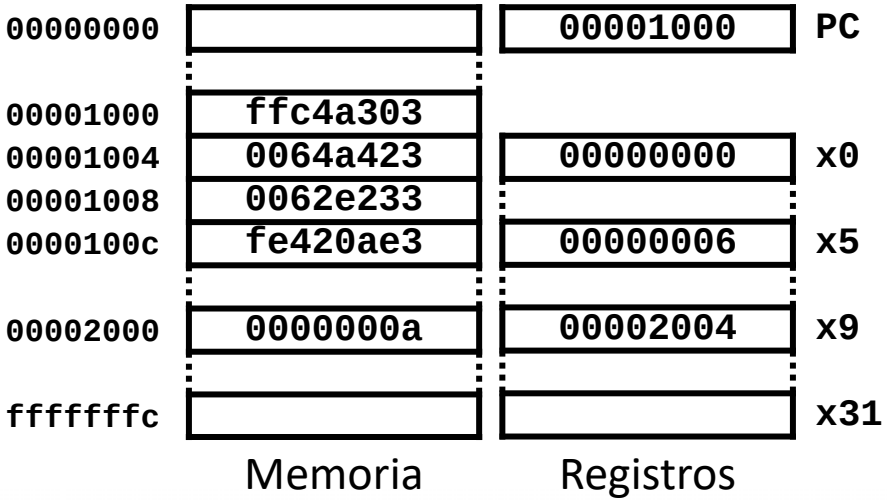
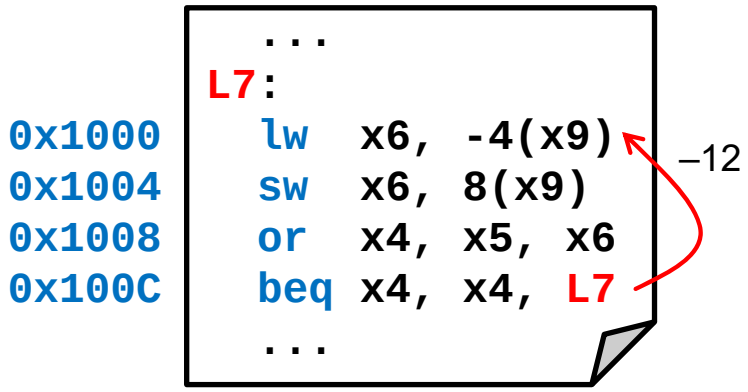


Diagrama de ejecución: registros y memoria

	lw x6,-4(x9)	sw x6,8(x9)	or x4,x5,x6	beq x4,x4,-12	lw x6,-4(x9)
PC	00001000	00001004			
x4	????????	=			
x5	00000006	=			
x6	????????	0000000a			
x9	00002004	=			
MEM[0x2000]	0000000a	=			
MEM[0x200c]	????????	=			
clk					
# ciclo	1	2	3	4	5



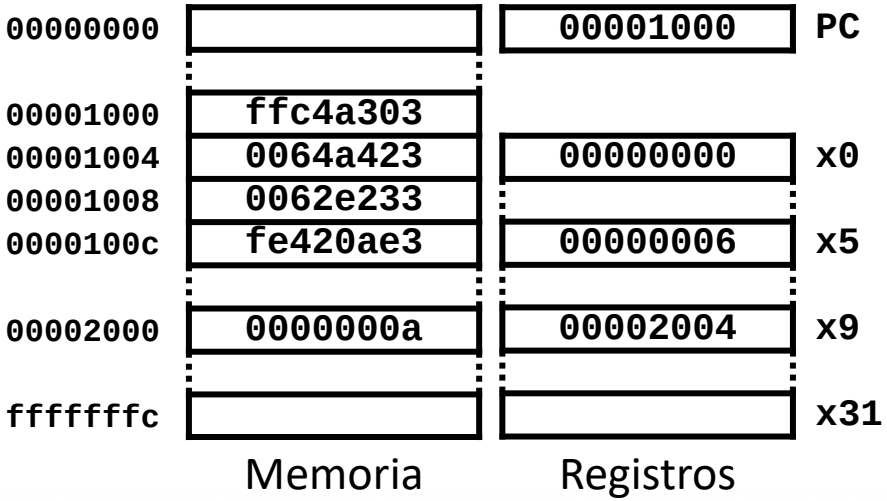
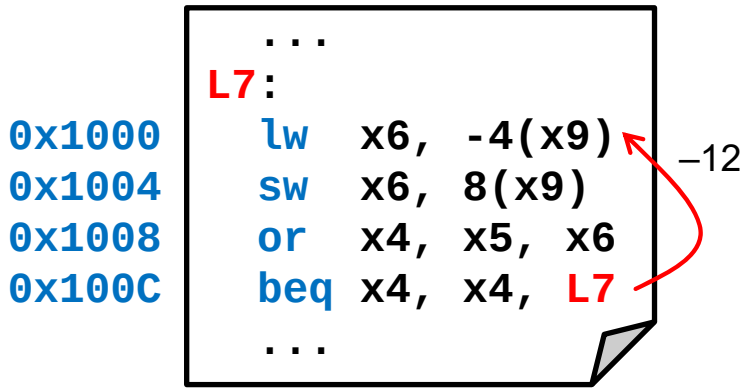


Diagrama de ejecución: registros y memoria

	lw x6,-4(x9)	sw x6,8(x9)	or x4,x5,x6	beq x4,x4,-12	lw x6,-4(x9)
PC	00001000	00001004	00001008		
x4	????????	????????	????????		
x5	00000006	00000006	=		
x6	????????	0000000a	=		
x9	00002004	00002004	=		
MEM[0x2000]	0000000a	0000000a	=		
MEM[0x200c]	????????	????????	0000000a		
clk					
# ciclo	1	2	3	4	5



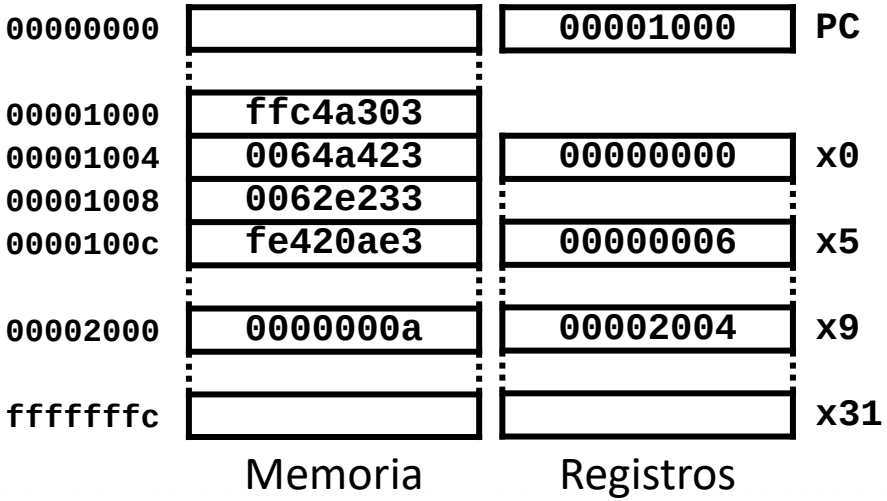
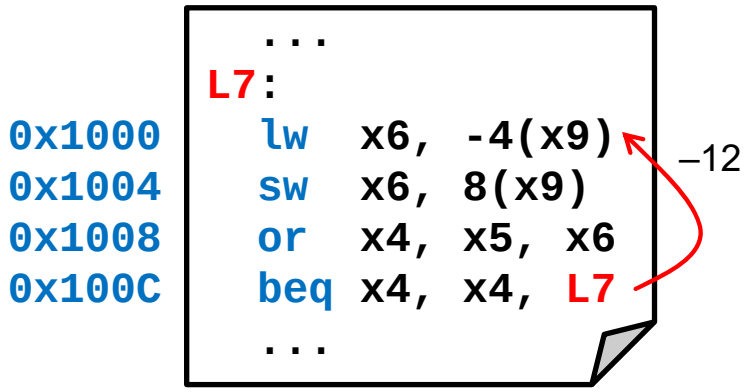


Diagrama de ejecución: registros y memoria

	lw x6,-4(x9)	sw x6,8(x9)	or x4,x5,x6	beq x4,x4,-12	lw x6,-4(x9)
PC	00001000	00001004	00001008	0000100c	
x4	????????	????????	????????	0000000e	
x5	00000006	00000006	00000006	=	
x6	????????	0000000a	0000000a	=	
x9	00002004	00002004	00002004	=	
MEM[0x2000]	0000000a	0000000a	0000000a	=	
MEM[0x200c]	????????	????????	0000000a	=	
clk					
# ciclo	1	2	3	4	5



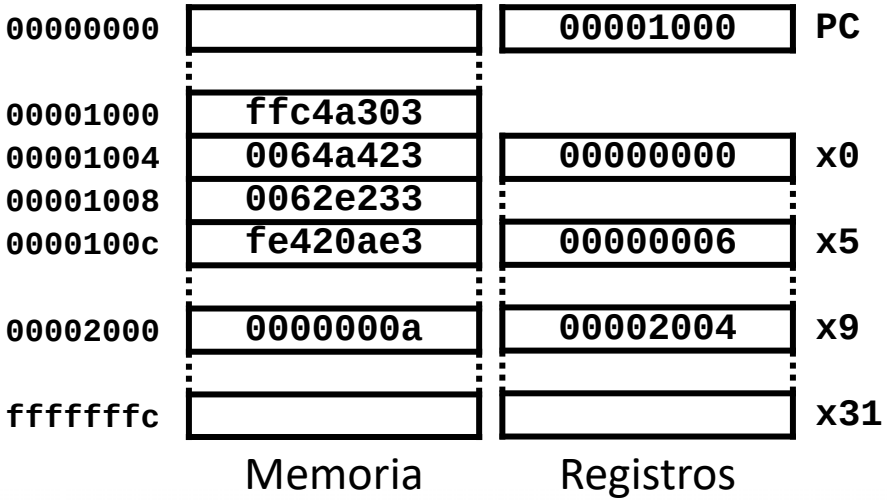
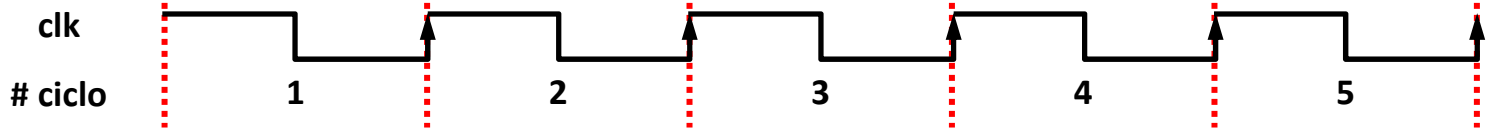
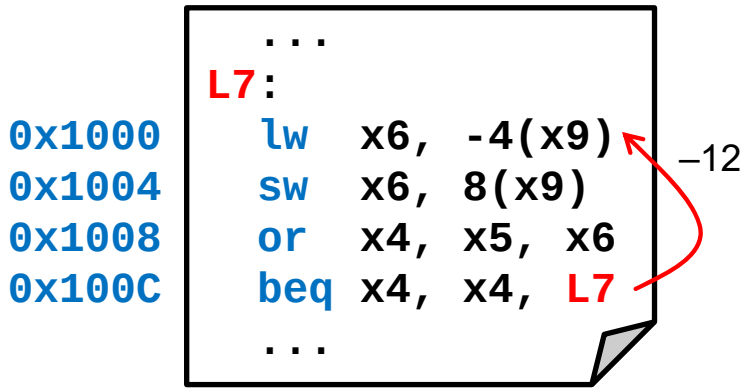


Diagrama de ejecución: registros y memoria

	lw x6,-4(x9)	sw x6,8(x9)	or x4,x5,x6	beq x4,x4,-12	lw x6,-4(x9)
PC	00001000	00001004	00001008	0000100c	00001000
x4	????????	????????	????????	0000000e	=
x5	00000006	00000006	00000006	00000006	=
x6	????????	0000000a	0000000a	0000000a	=
x9	00002004	00002004	00002004	00002004	=
MEM[0x2000]	0000000a	0000000a	0000000a	0000000a	=
MEM[0x200c]	????????	????????	0000000a	0000000a	=





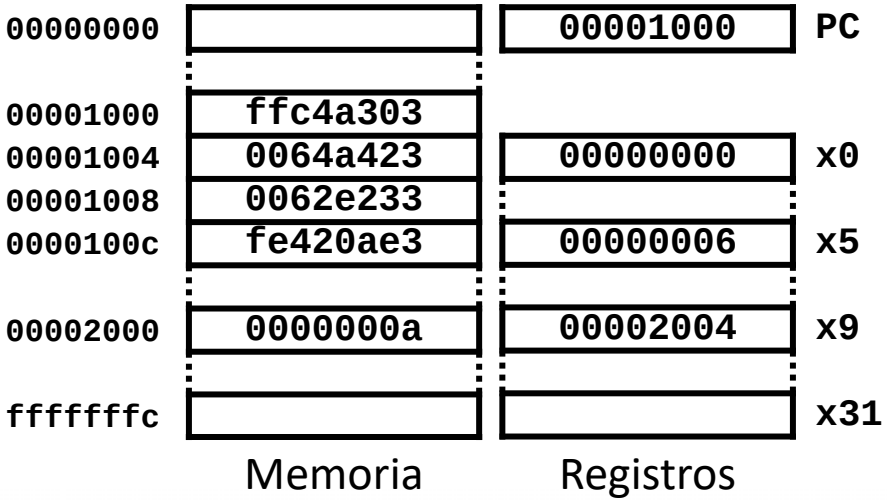


Diagrama de ejecución: registros y memoria

	lw x6,-4(x9)	sw x6,8(x9)	or x4,x5,x6	beq x4,x4,-12	lw x6,-4(x9)
PC	00001000	00001004	00001008	0000100c	00001000
x4	????????	????????	????????	0000000e	0000000e
x5	00000006	00000006	00000006	00000006	00000006
x6	????????	0000000a	0000000a	0000000a	0000000a
x9	00002004	00002004	00002004	00002004	00002004
MEM[0x2000]	0000000a	0000000a	0000000a	0000000a	0000000a
MEM[0x200c]	????????	????????	0000000a	0000000a	0000000a

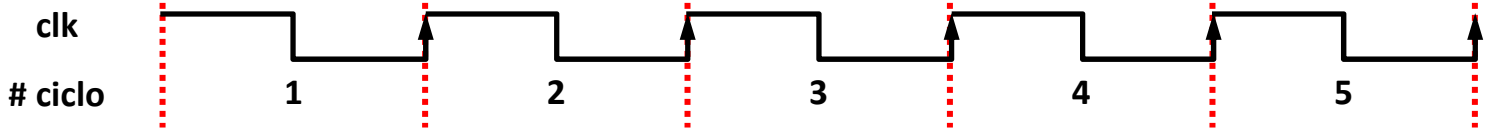


Diagrama de ejecución: señales de estado y control

	`lw x6,-4(x9)`	`sw x6,8(x9)`	`or x4,x5,x6`	`beq x4,x4,-12`	`lw x6,-4(x9)`
op	0000011	0100011	0110011	1100011	0000011
$funct7_5$	–	–	0	–	–
funct3	010	010	110	000	010
zero	–	–	–	1	–
Branch					
Jump					
BRwr					
ALUsrc					
ALUop					
MemWr					
ResSrc					
InmSrc					
ALUctr					
PCsrc					
clk					
# ciclo	1	2	3	4	5

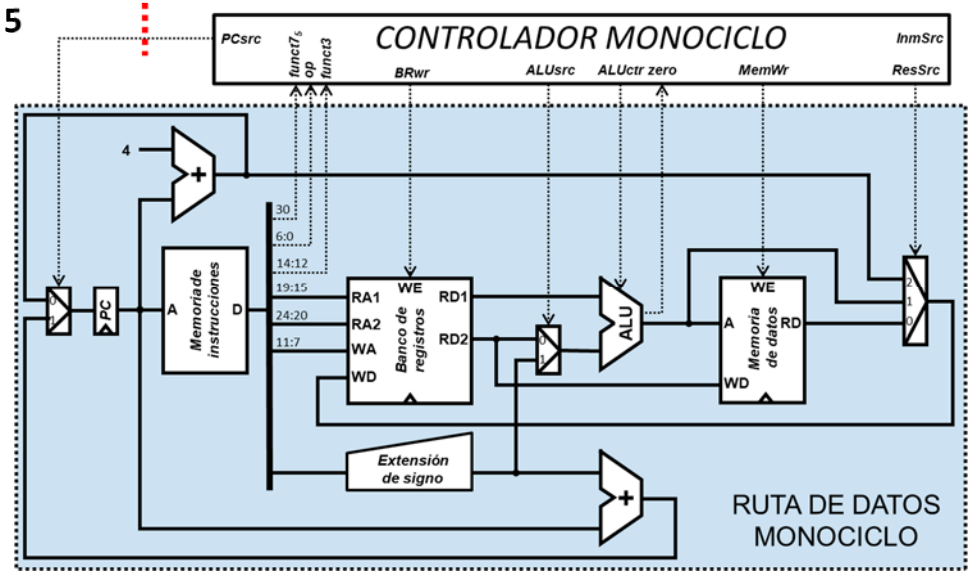
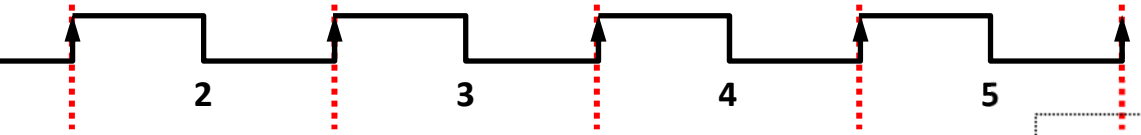
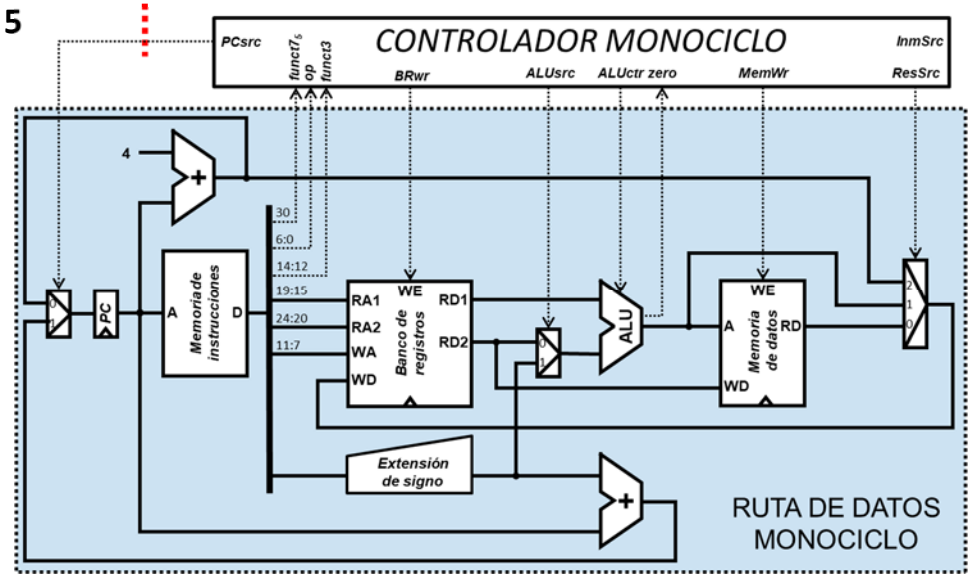


Diagrama de ejecución: señales de estado y control

	`lw x6,-4(x9)`	`sw x6,8(x9)`	`or x4,x5,x6`	`beq x4,x4,-12`	`lw x6,-4(x9)`
op	0000011	0100011	0110011	1100011	0000011
funct7$_5$	–	–	0	–	–
funct3	010	010	110	000	010
zero	–	–	–	1	–
Branch	0	0	0	1	0
Jump	0	0	0	0	0
BRwr	1	0	1	0	1
ALUsrc	1	1	0	0	1
ALUop	00	00	10	01	00
MemWr	0	1	0	0	0
ResSrc	00	–	01	–	00
InmSrc	00	01	–	10	00
ALUctr	000	000	011	001	000
PCsrc	0	0	0	1	0





Acerca de *Creative Commons*

- Licencia CC (Creative Commons)
  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

Reconocimiento (*Attribution*):
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No comercial (*Non commercial*):
La explotación de la obra queda limitada a usos no comerciales.

Compartir igual (*Share alike*):
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/