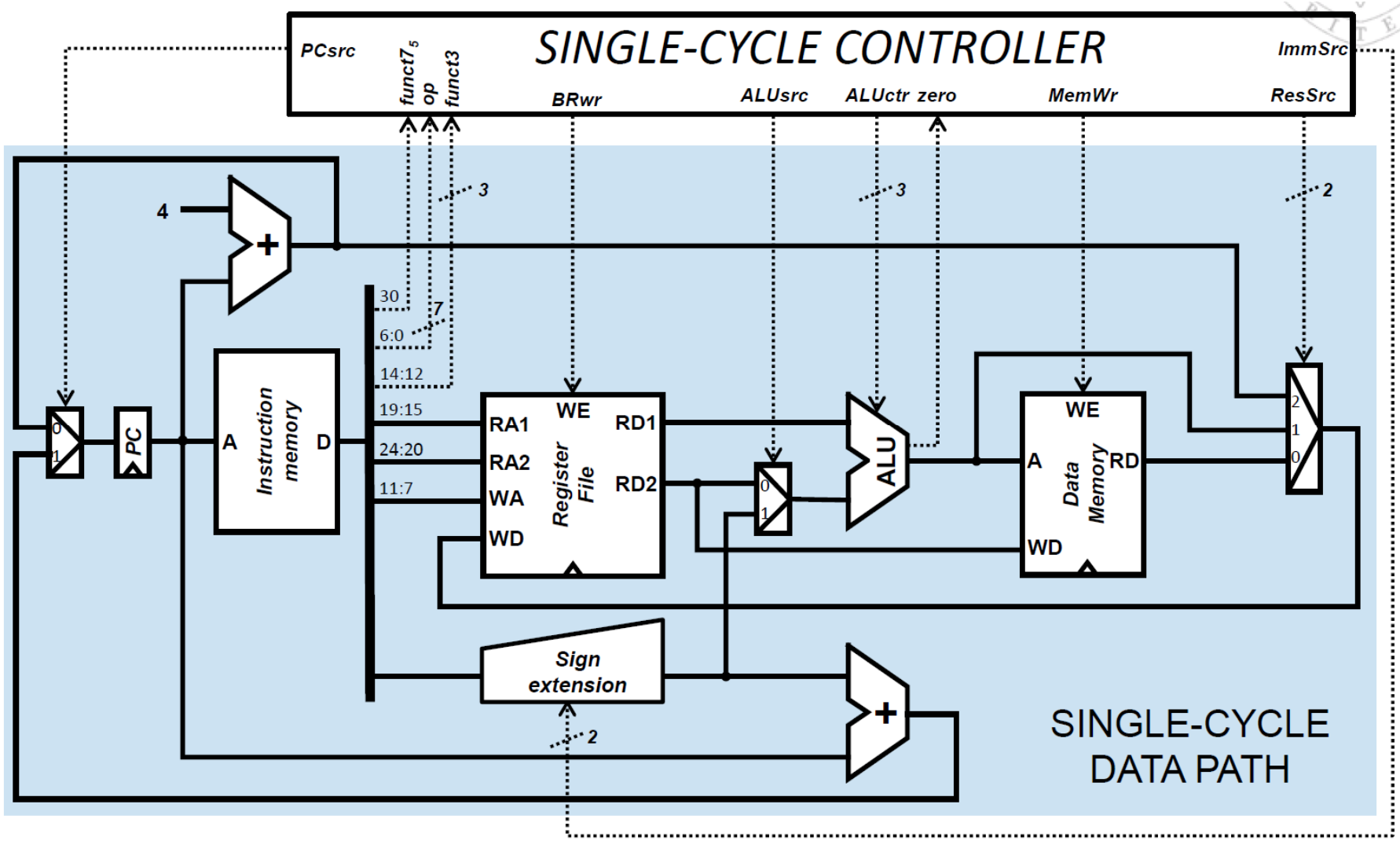
Module 5 - Problems:

Single-cycle processor design

Introduction to Computers II

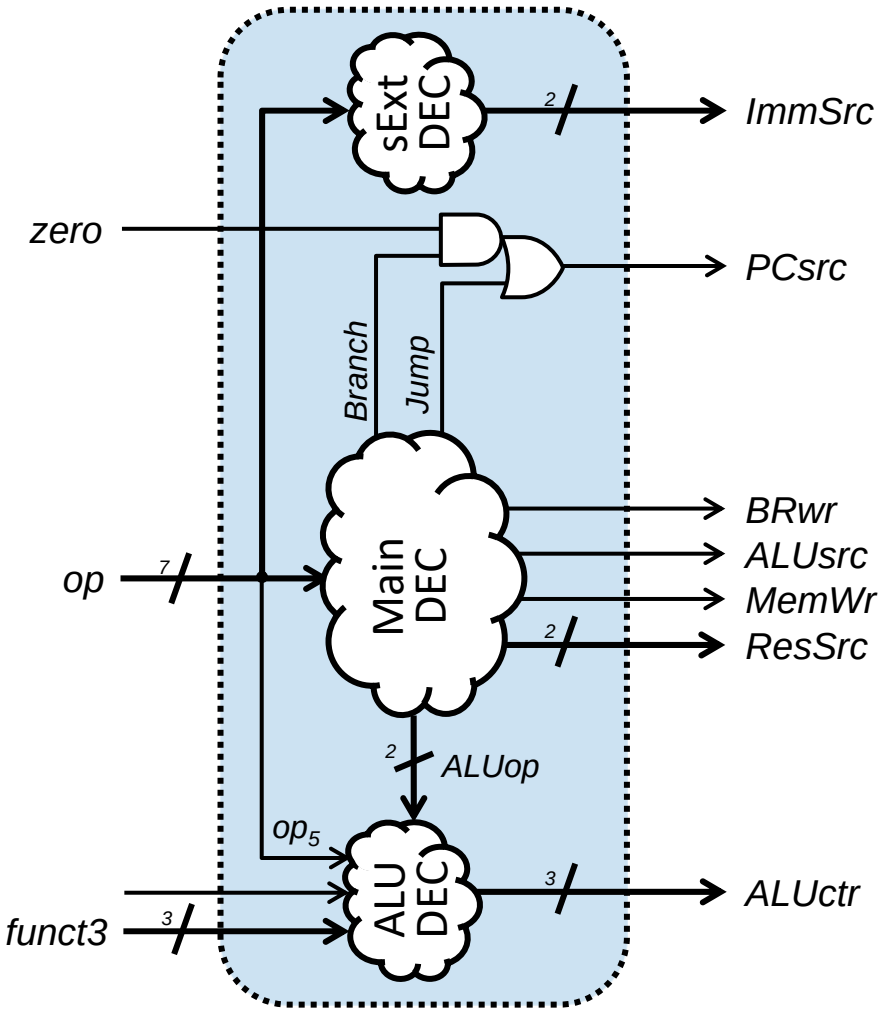
Fernando Castro Rodríguez
José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

2) Provide the value of the control signals produced in a single-cycle RISC-V when executing a `lw` instruction.



Truth table

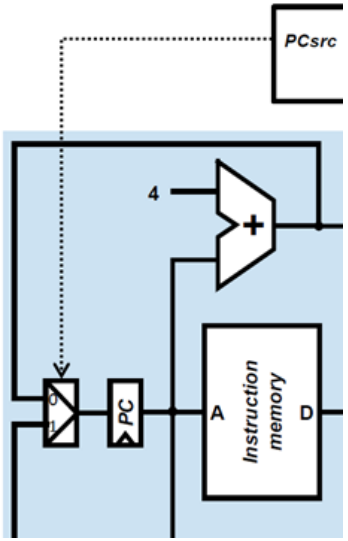
op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (`lw`)	0	0	1	1	00 (add)	0	00
0100011 (`sw`)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (`beq`)	1	0	0	0	01 (subtract)	0	–
1101111 (`jal`)	0	1	1	–	–	0	10

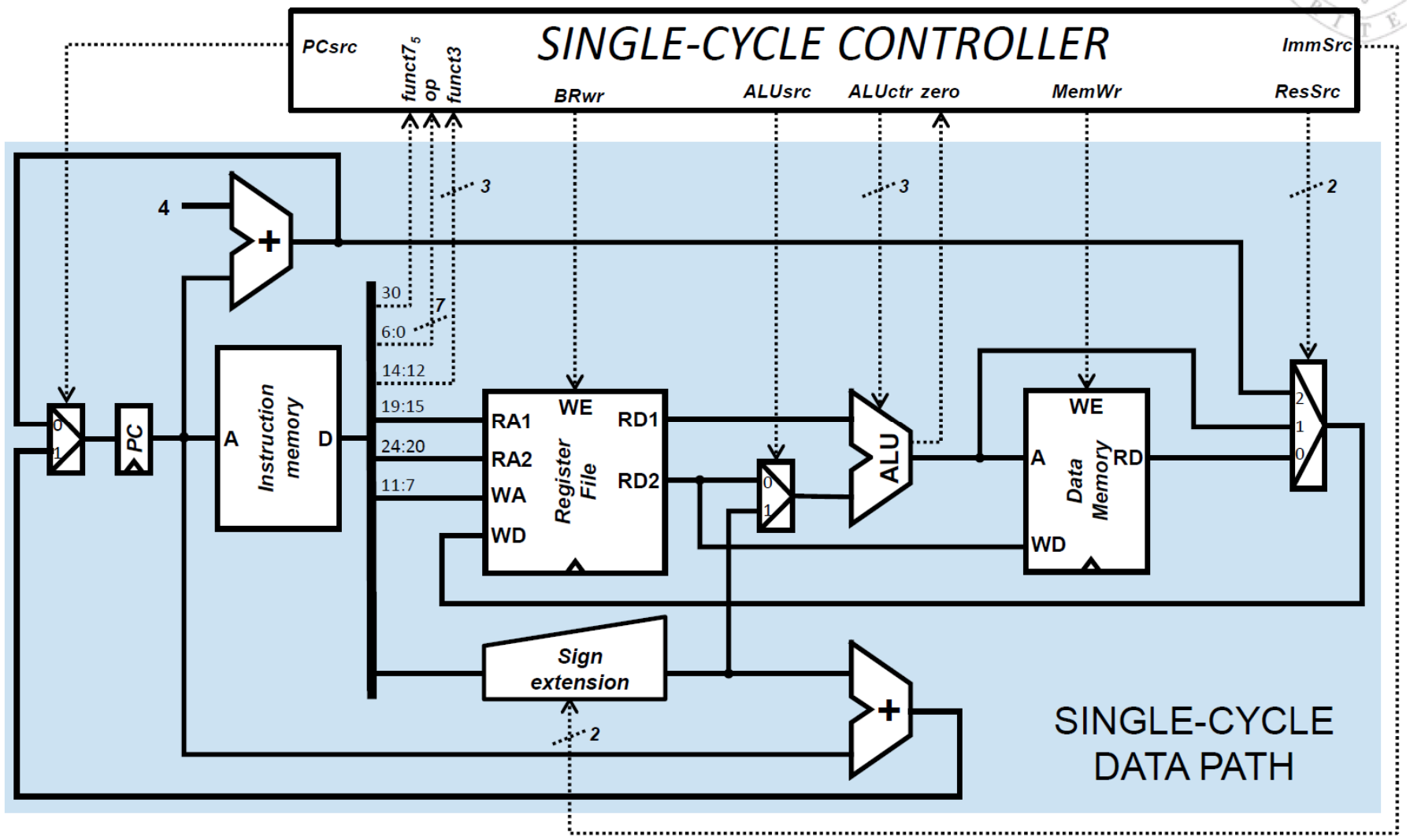


Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- The Branch and Jump signals are 0 because this is not a conditional branch (beq) or an unconditional branch (jal), and therefore the PCsrc signal will also be 0. The PC will be updated with PC+4 (instruction after “lw”)

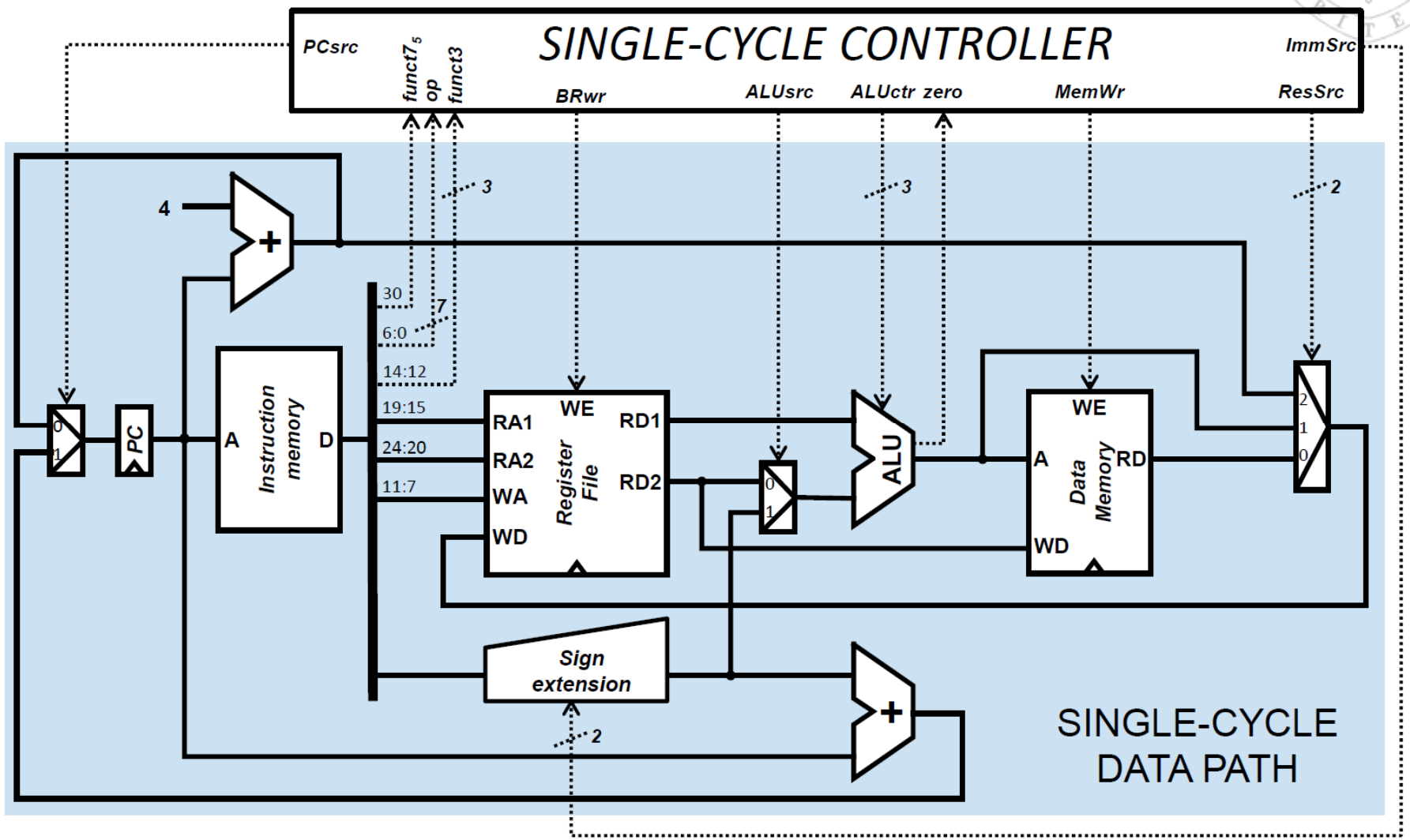




Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- The BRwr and MemWr signals will be 1 and 0 respectively, because “lw” instructions write in the register file (the data read from memory will be loaded into the destination register), but they do not write in the memory (only “store” instructions can write in the memory)



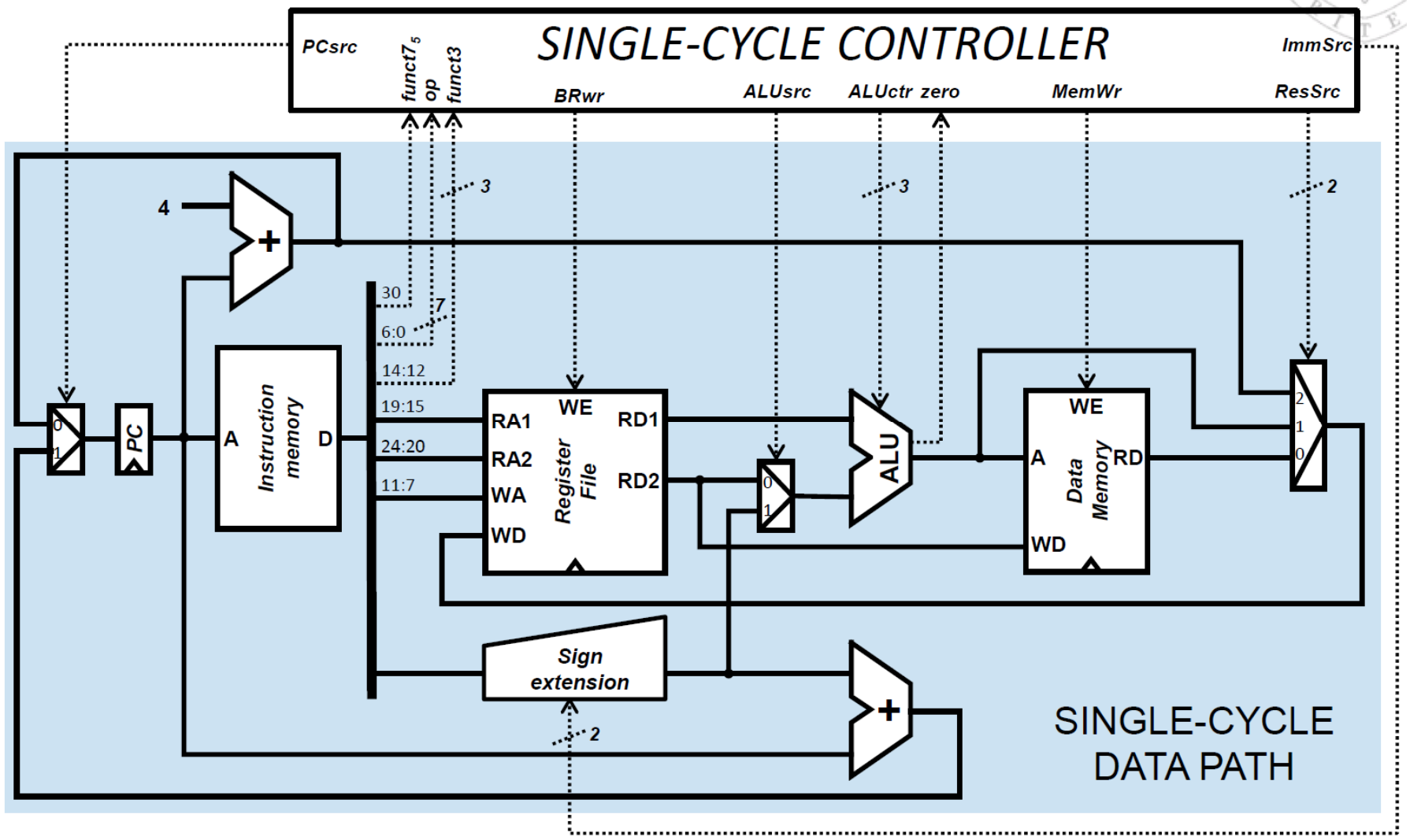
Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

Truth table

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

- The ALUsrc signal (which controls the source of the lower operand of the ALU) has a value of 1, because the content of source register rs1 is added with a sign-extended immediate. The ALU will perform the addition needed to calculate the effective memory address, from which the data will be read. To do this, ALUop will be 00 (same as with “store” instructions, since both perform the address calculation in the same way). Therefore, ALUctr will be 000.

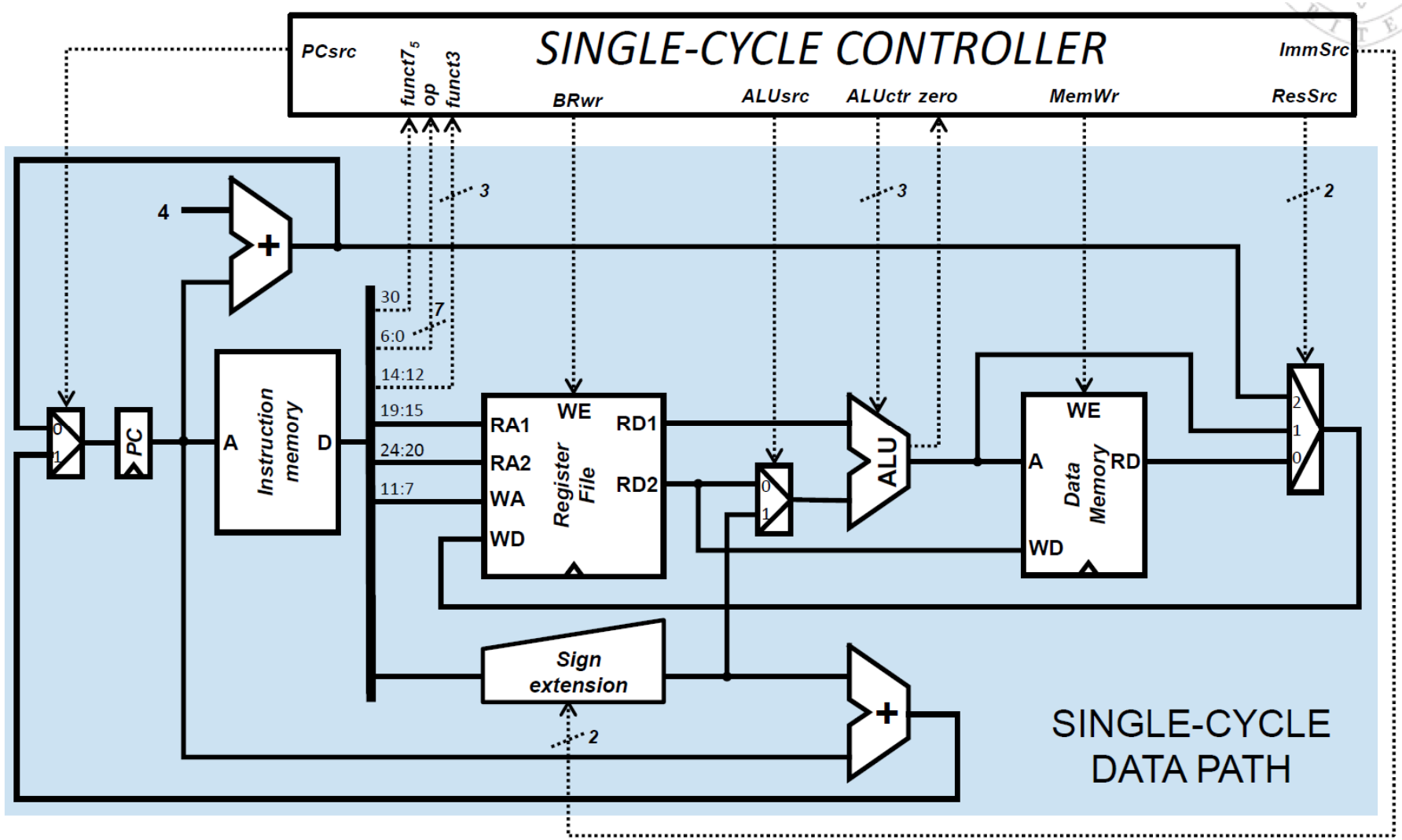


Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

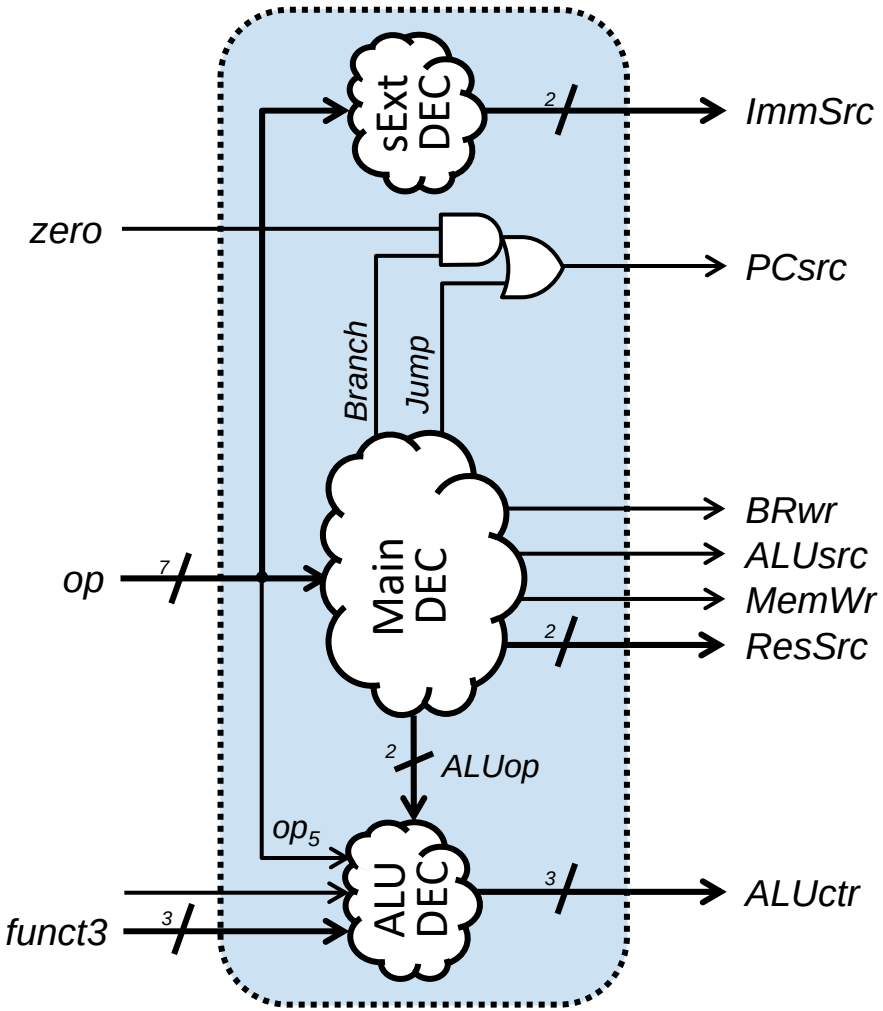
- Finally, the ResSrc signal, which controls the source of the data that will be written in the register file, has a value of 00. This is to indicate that the value to be written in the destination register (determined by bits 11:7 of the instruction), is the data read from the data memory.

3) Provide the value of the control signals produced in a single-cycle RISC-V when executing an **`andi`** instruction.



Truth table

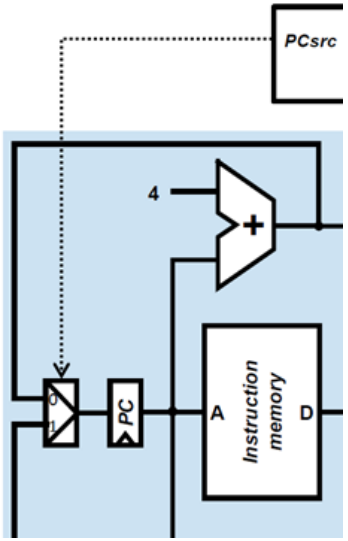
op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (`lw`)	0	0	1	1	00 (add)	0	00
0100011 (`sw`)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (`beq`)	1	0	0	0	01 (subtract)	0	–
1101111 (`jal`)	0	1	1	–	–	0	10

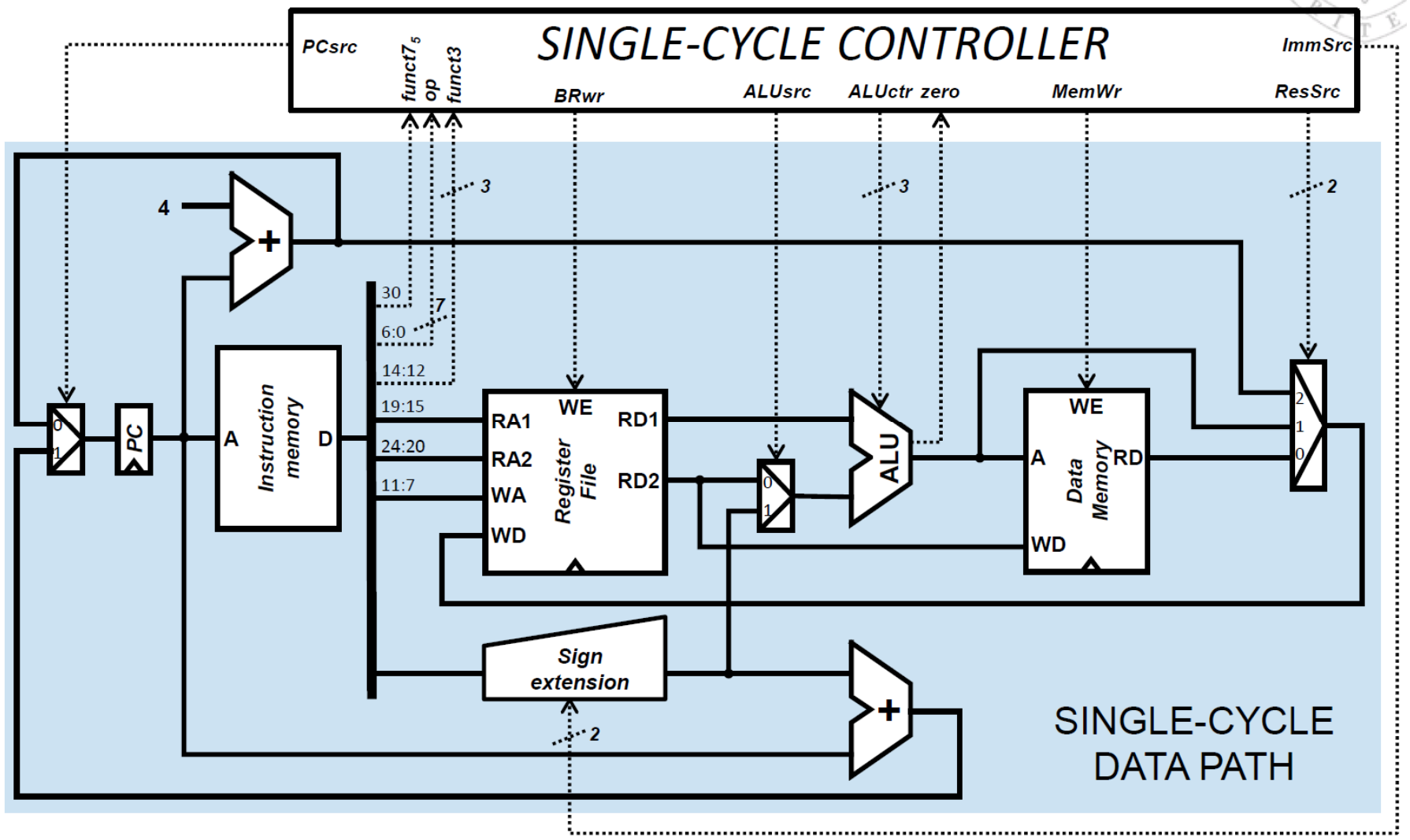


Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- The Branch and Jump signals are 0 because this is not a conditional branch (beq) or an unconditional branch (jal), and therefore the PCsrc signal will also be 0. The PC will be updated with PC+4 (instruction after “andi”)

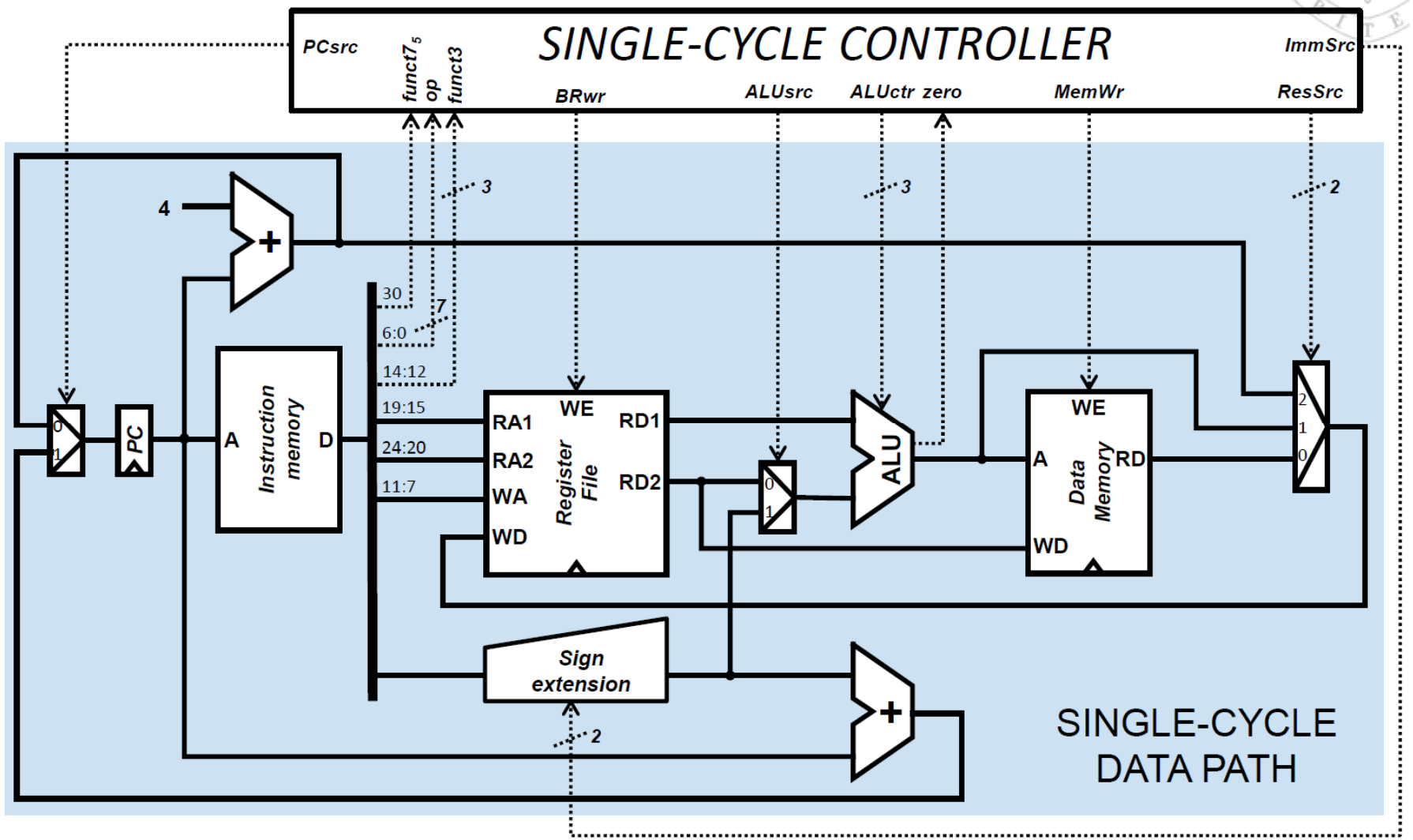




Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- The BRwr and MemWr signals will be 1 and 0 respectively, because “andi” instructions write in the register file (the result of the “and” operation), but they do not write in the memory (only “store” instructions can write in the memory)



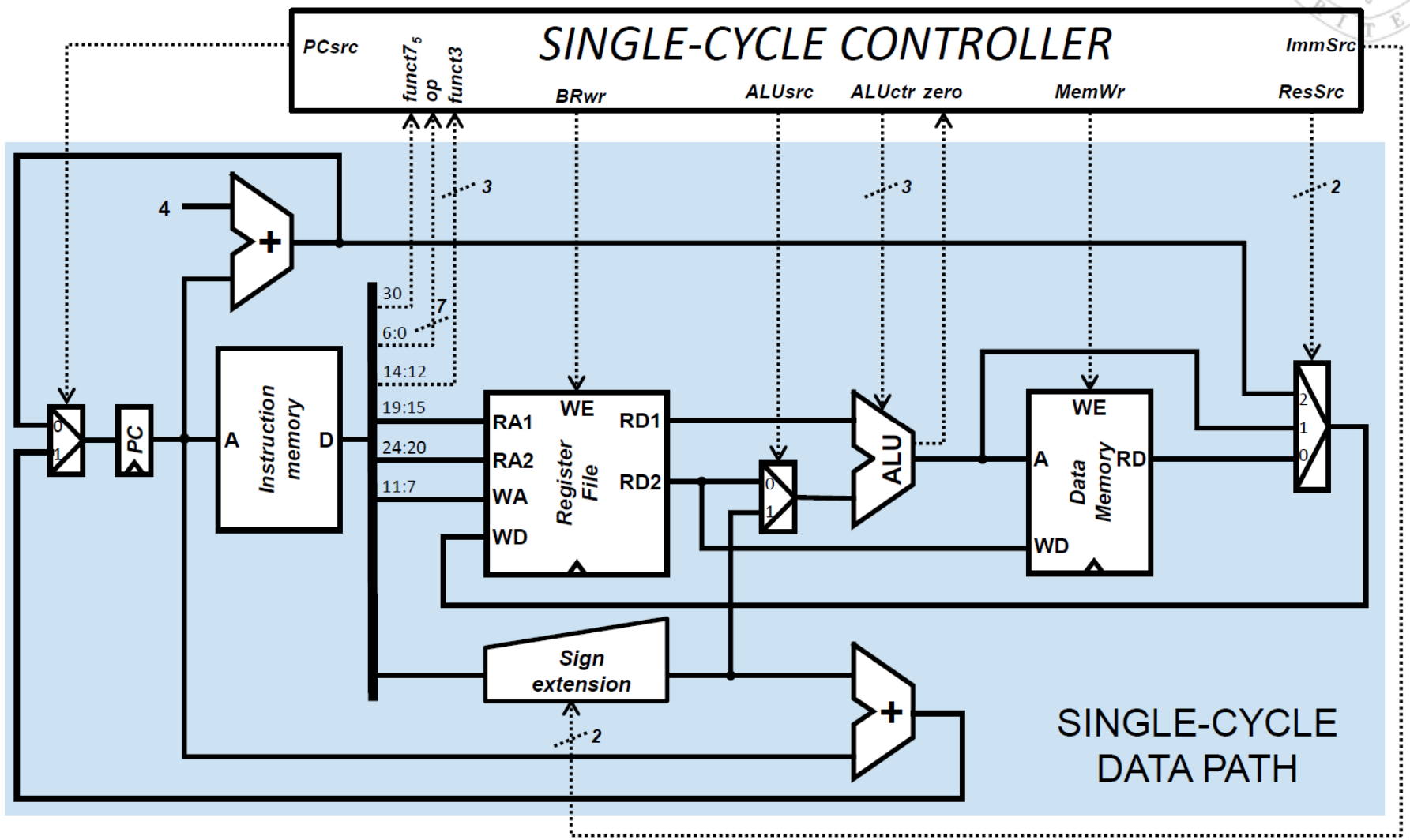
Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

Truth table

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

- The ALUsrc signal (which controls the source of the lower operand of the ALU) has a value of 1, because an “and” operation is performed on the content of source register rs1 with a sign-extended immediate. The ALU will perform the “and” needed to carry out this operation. To do this, ALUop will be 10 and ALUctr will be 010.

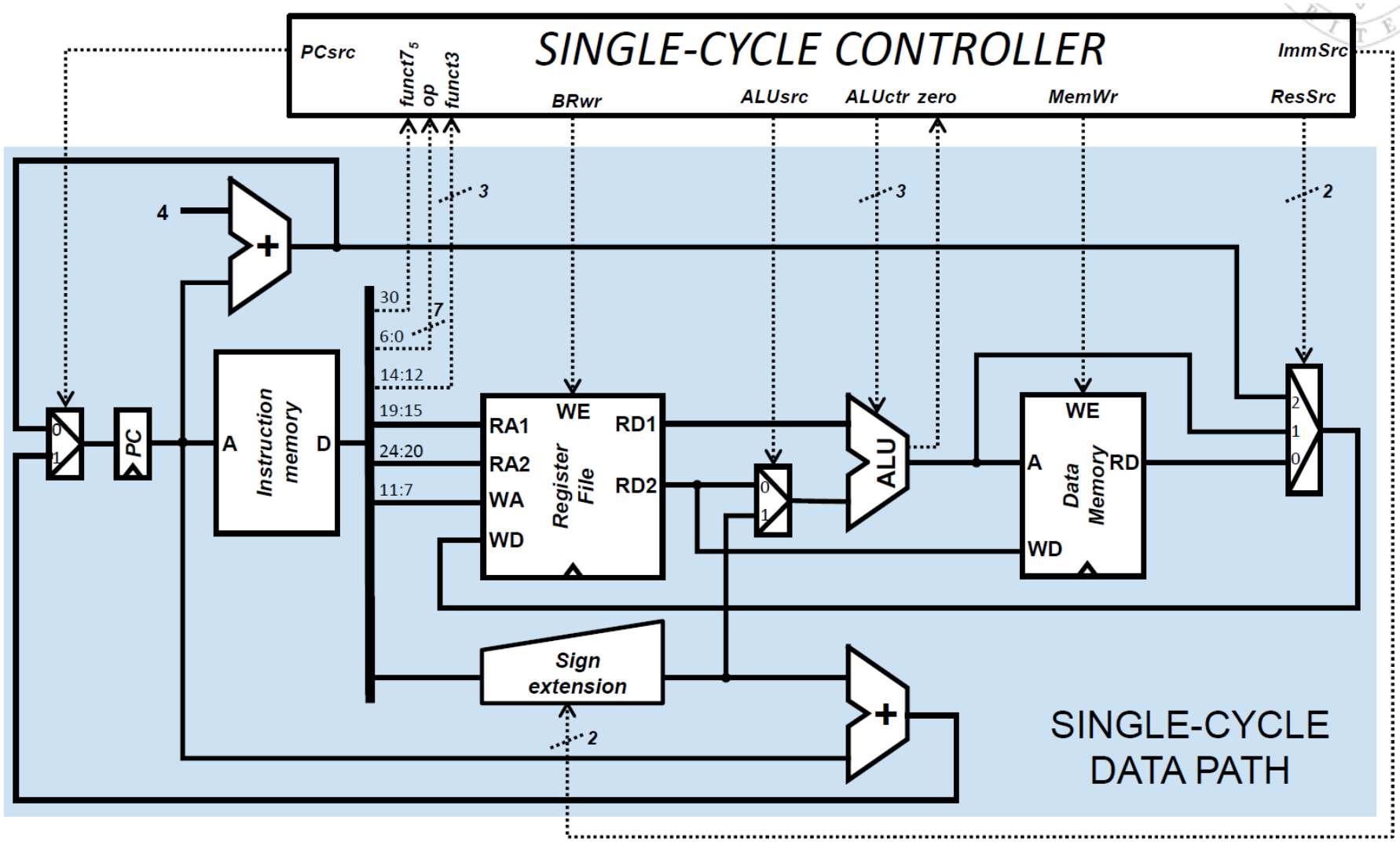


Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

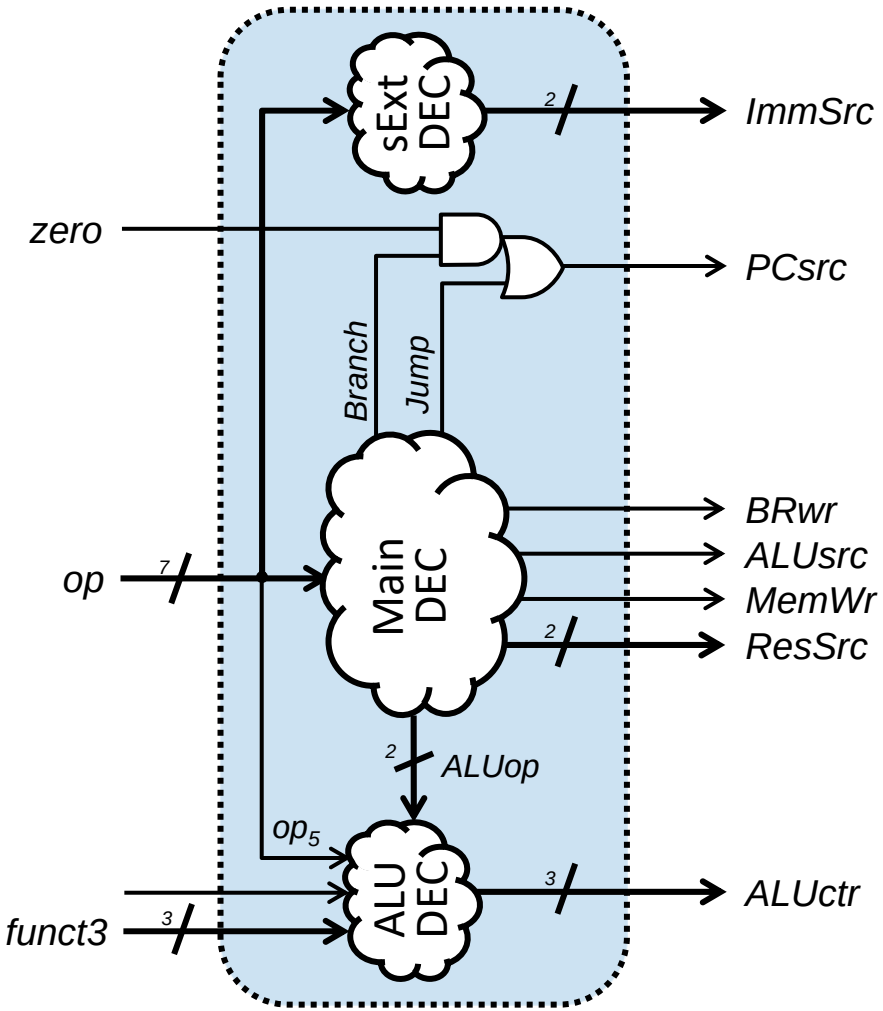
- Finally, the ResSrc signal, which controls the source of the data that will be written in the register file, has a value of 01. This is to indicate that the value to be written in the destination register (determined by bits 11:7 of the instruction), is the result produced by the ALU after the “and” operation.

4) Provide the value of the control signals produced in a single-cycle RISC-V when executing a **beq** instruction.



Truth table

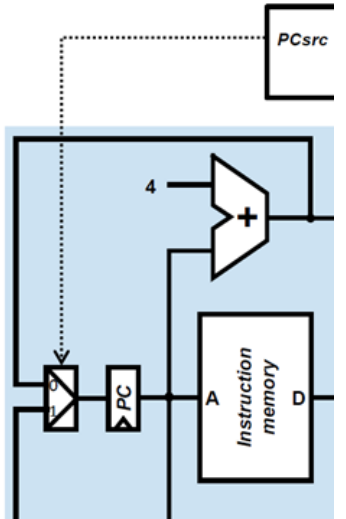
op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

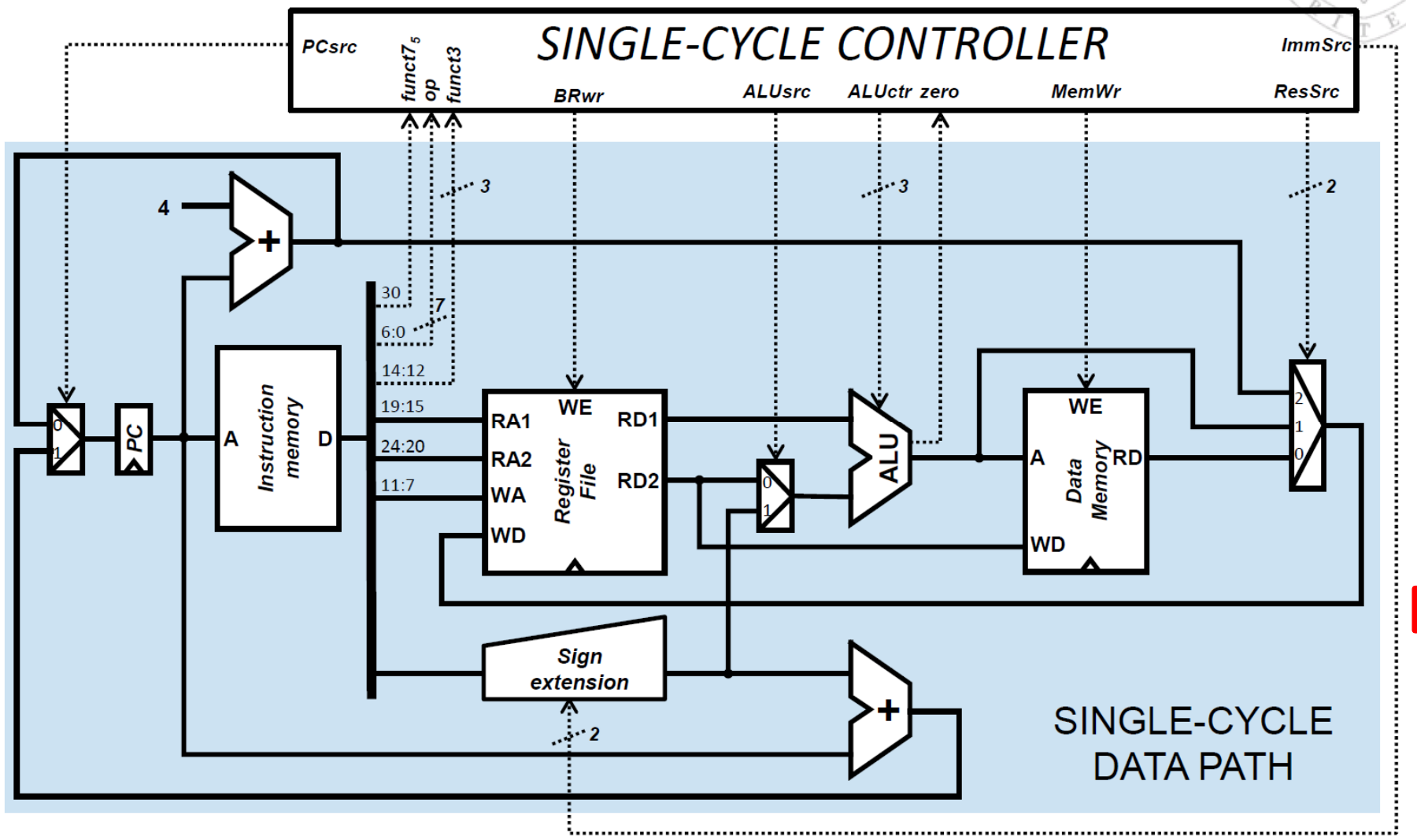


Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- The Branch signal is 1, because this is a conditional branch (beq), and Jump is 0 because this is not an unconditional branch (jal). Therefore, the value of PCsrc will depend on the value of the status signal “zero”, produced by the evaluation of the condition:
- If “zero” = 1 → the condition is met, so PCsrc=1 and the PC will be updated with the branch destination address (PC + SExt(immediate) generated by the adder in the lower side of the data path.
- If “zero” = 0 → the condition is not met, so PCsrc=0 and the PC will be updated with PC+4 (instruction after “beq”).

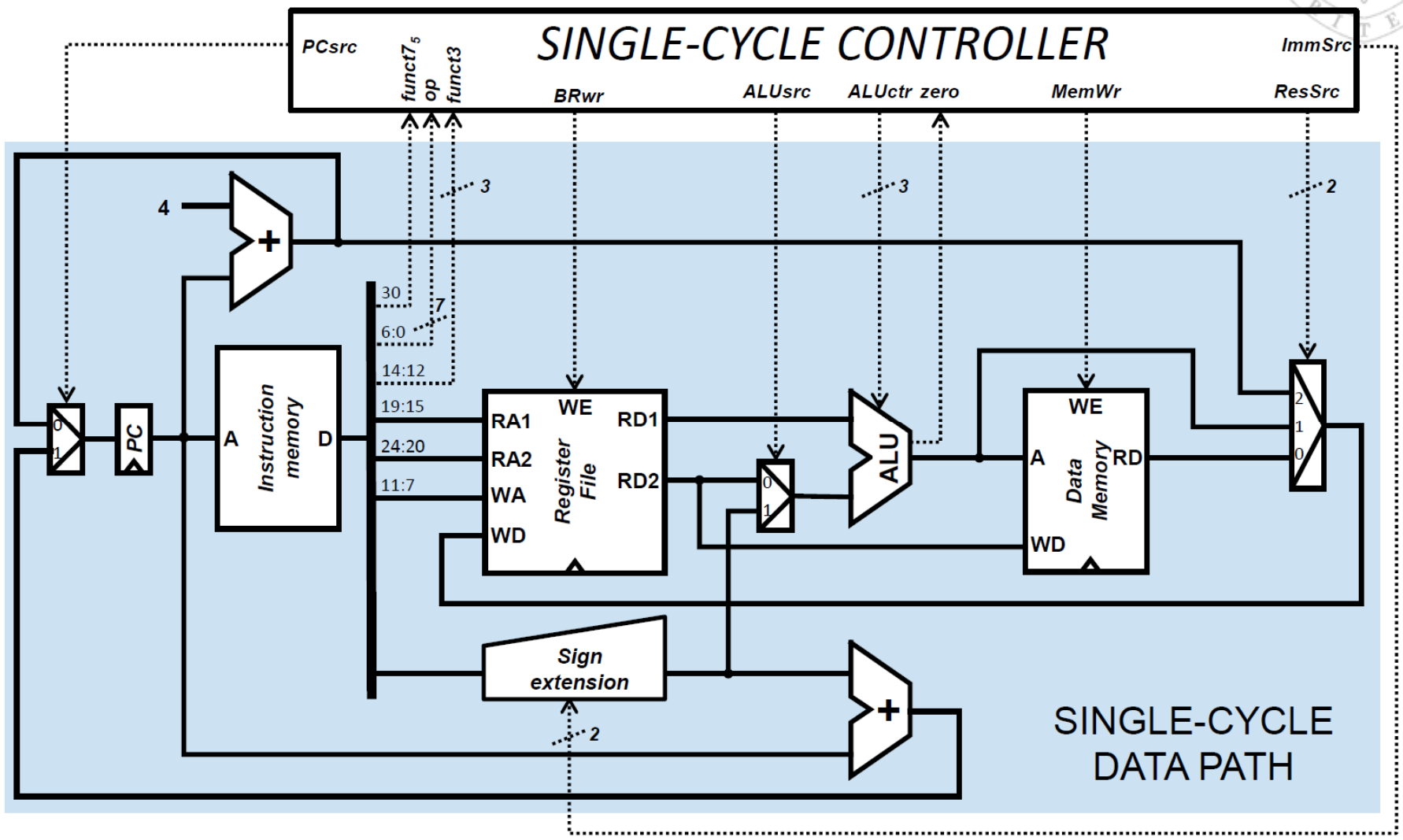




Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

- The BRwr and MemWr signals are both 0, because “beq” instructions do not write in the register file or in the memory.



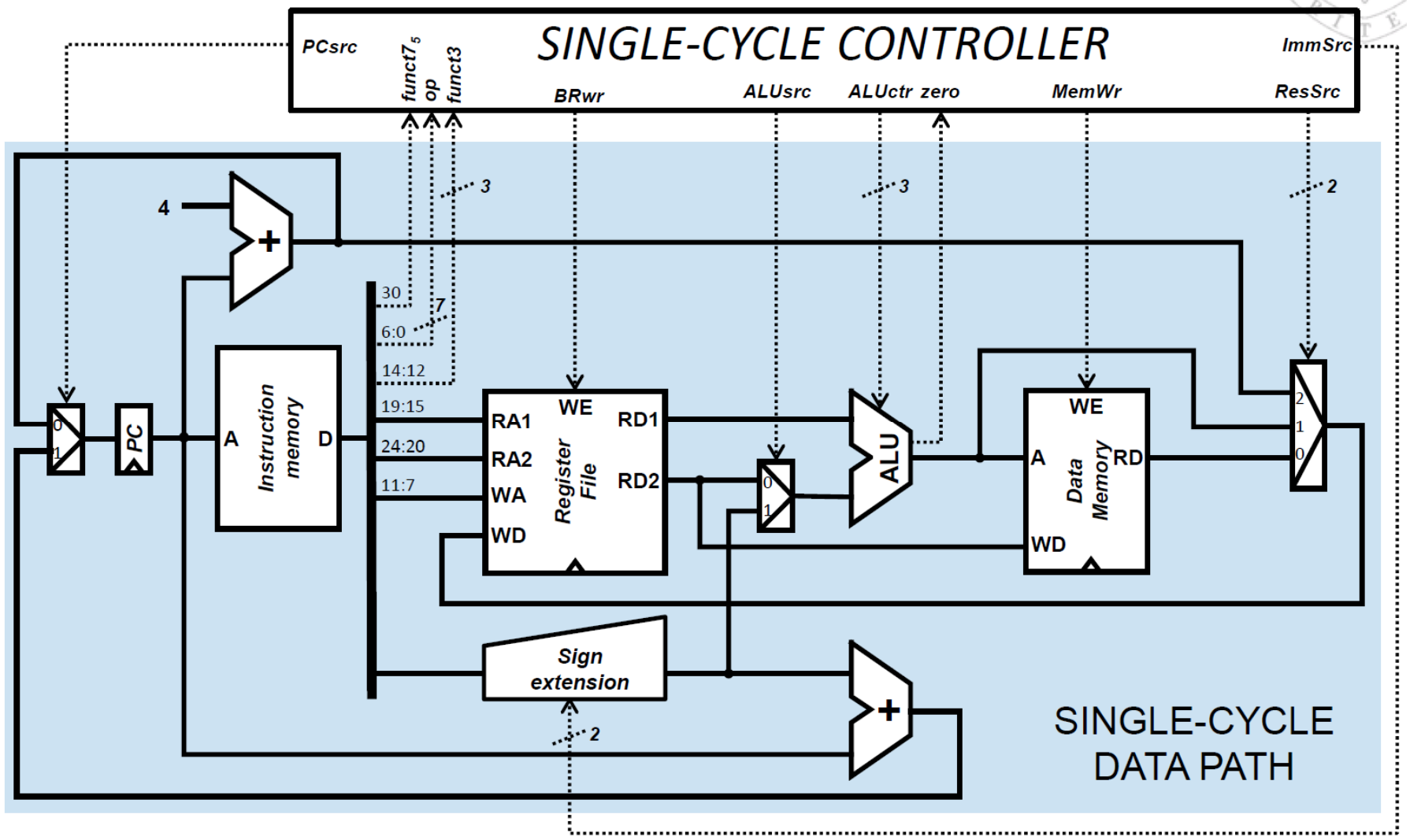
Truth table

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

Truth table

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

- The ALUsrc signal (which controls the source of the lower operand of the ALU) has a value of 0, because source registers rs1 and rs2 have to be subtracted to determine the equality condition between them. The ALU will perform this subtraction. To do this, ALUop will be 01 and ALUctr will be 001.

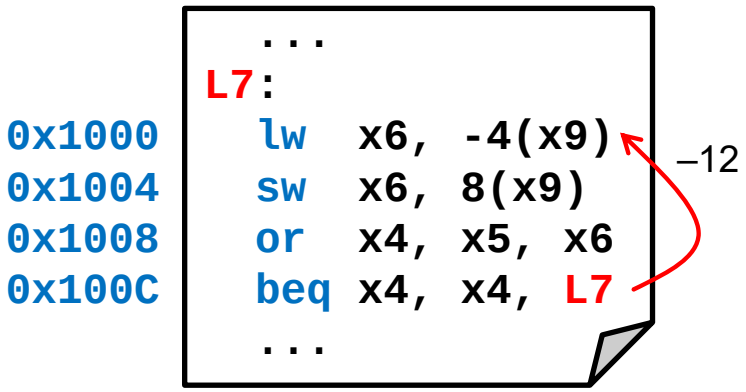


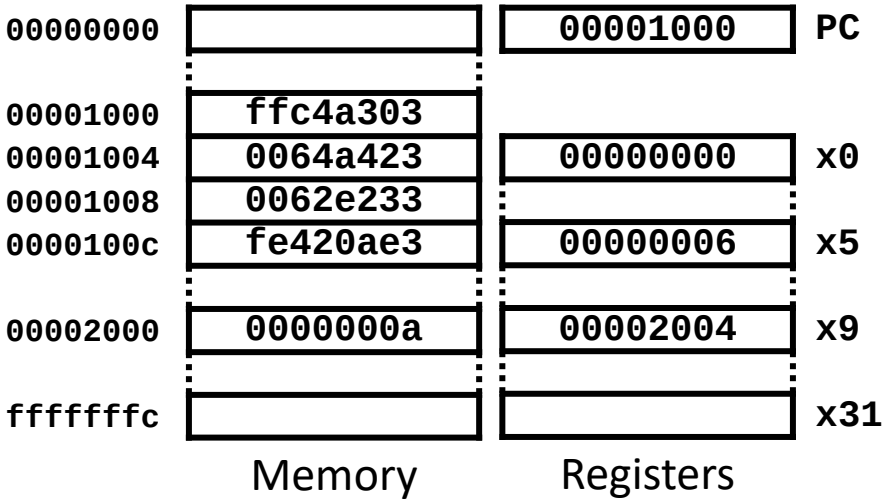
Truth table

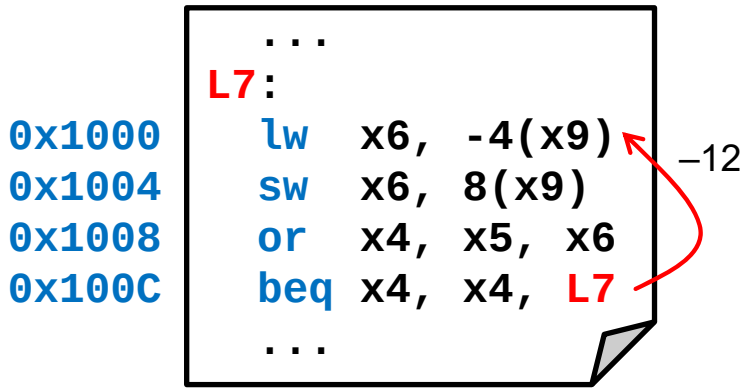
op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (add)	0	00
0100011 (sw)	0	0	0	1	00 (add)	1	–
0010011 (I-type)	0	0	1	1	10 (operate)	0	01
0110011 (R-type)	0	0	1	0	10 (operate)	0	01
1100011 (beq)	1	0	0	0	01 (subtract)	0	–
1101111 (jal)	0	1	1	–	–	0	10

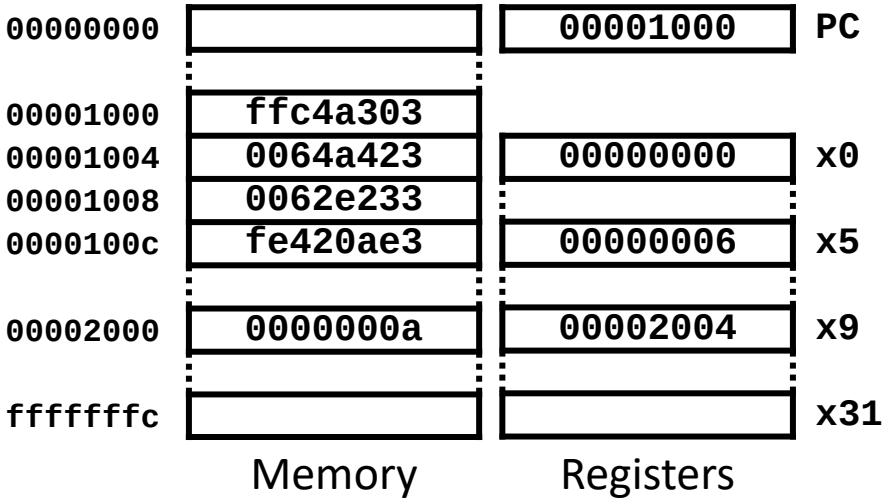
- Finally, the ResSrc signal, which controls the source of the data that will be written in the register file, is a “don’t care”, because “beq” instructions do not write in the register file (BRwr=0).

8) Assume that the following program is running on a single-cycle RISC-V, with the initial value of the memory and registers shown in the picture. Represent the execution diagrams for the registers and the memory, as well as for the control and status signals, so that their values are shown for each clock cycle.

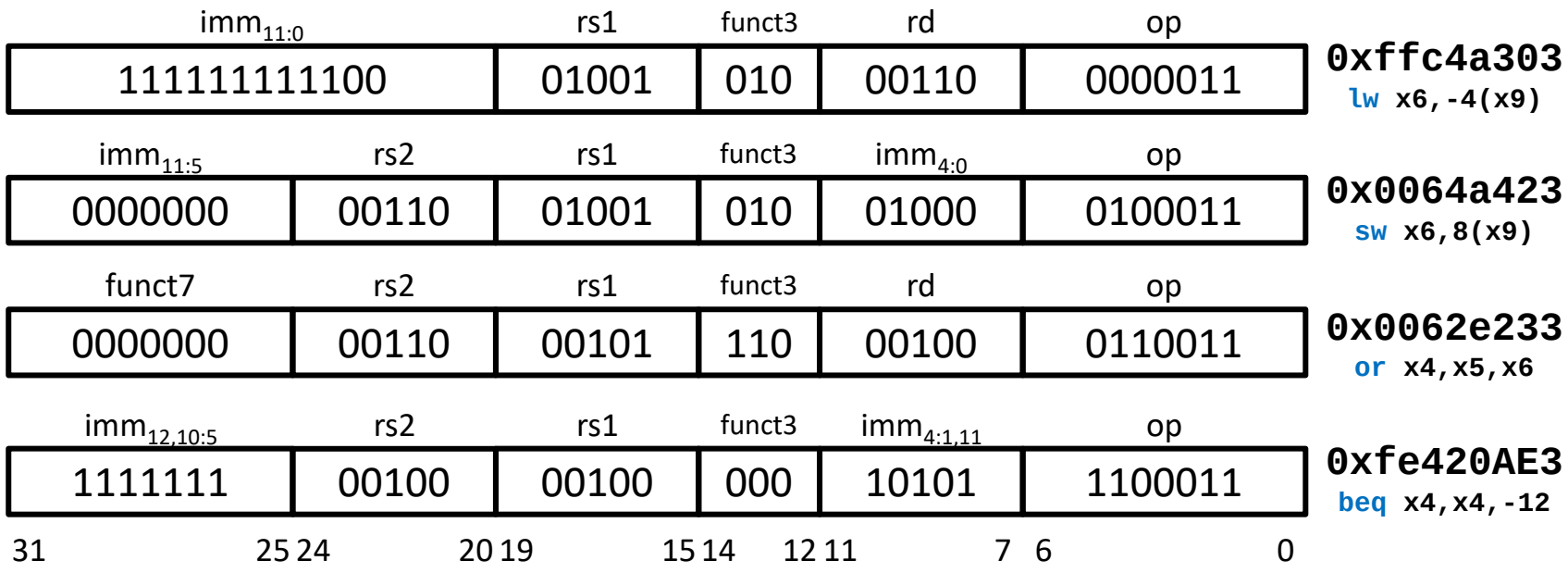


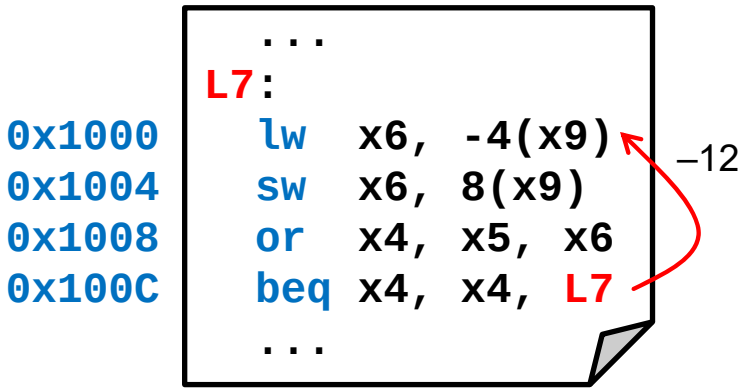


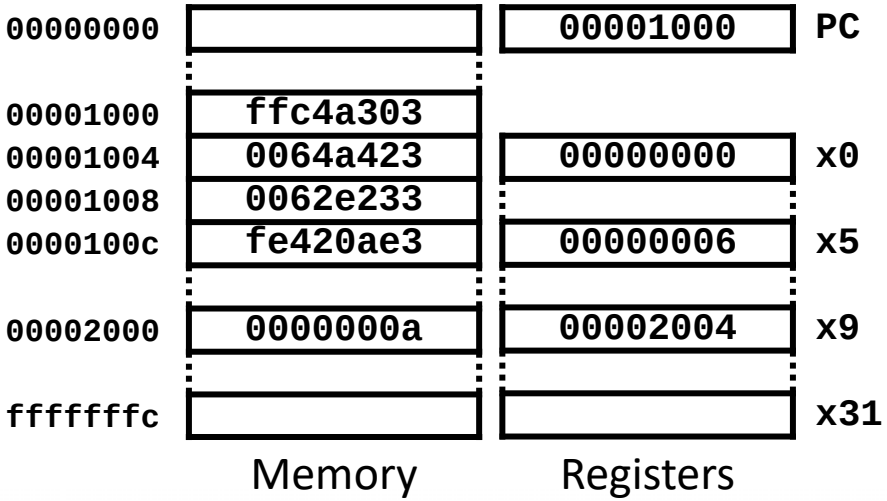




The machine code representation of the program instructions is the following:

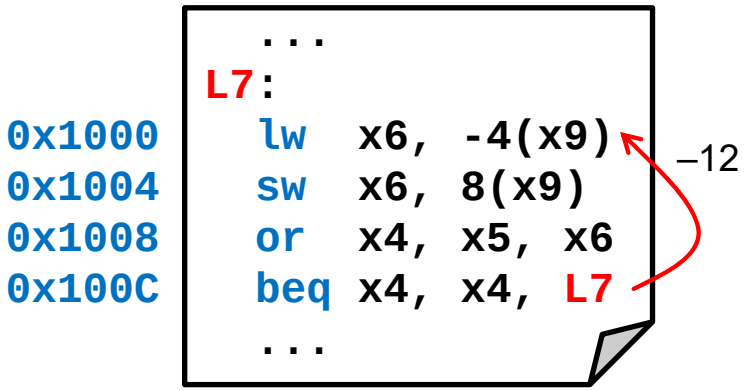


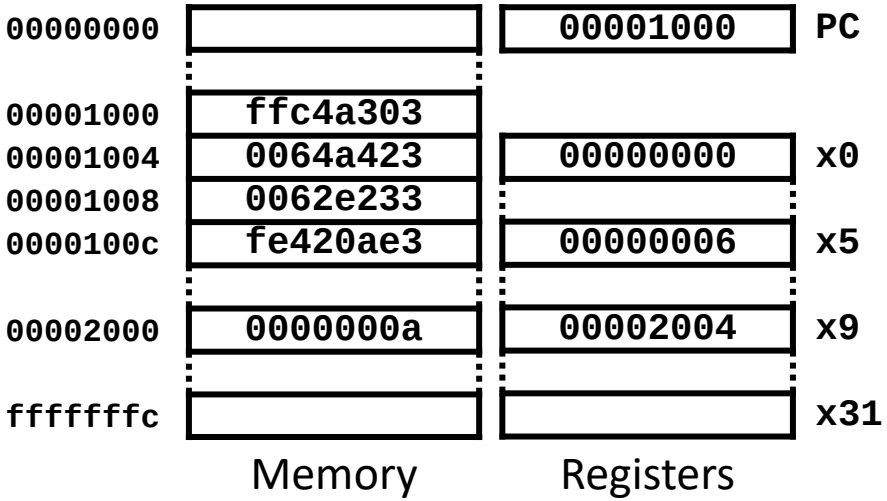




Execution diagram: registers and memory

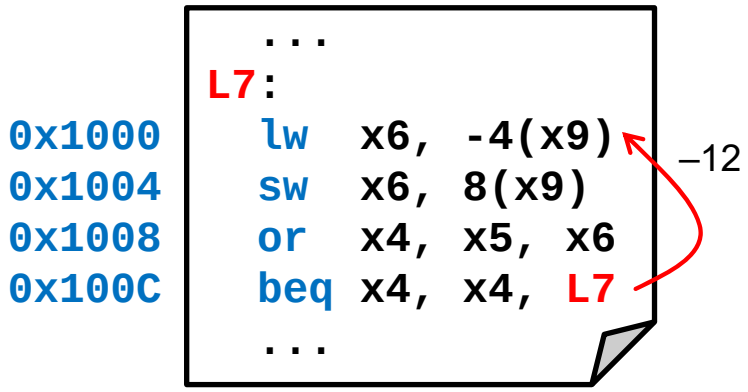
	lw x6,-4(x9)	sw x6,8(x9)	or x4,x5,x6	beq x4,x4,-12	lw x6,-4(x9)
PC	00001000				
x4	????????				
x5	00000006				
x6	????????				
x9	00002004				
MEM[0x2000]	0000000a				
MEM[0x200c]	????????				
clk					
# cycle	1	2	3	4	5

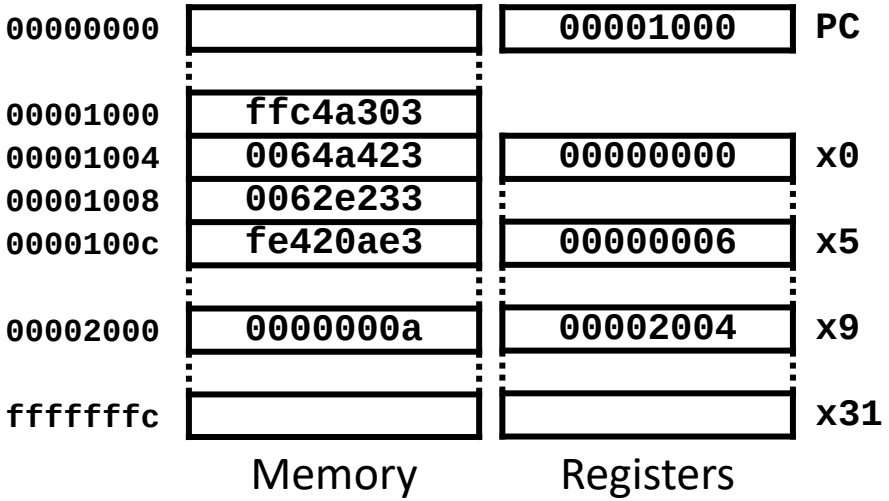




Execution diagram: registers and memory

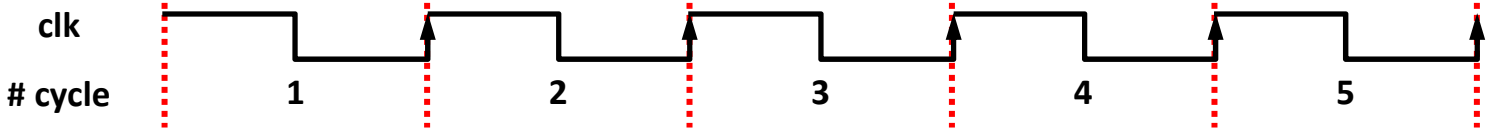
	lw x6,-4(x9)	sw x6,8(x9)	or x4,x5,x6	beq x4,x4,-12	lw x6,-4(x9)
PC	00001000	00001004			
x4	????????	=			
x5	00000006	=			
x6	????????	0000000a			
x9	00002004	=			
MEM[0x2000]	0000000a	=			
MEM[0x200c]	????????	=			
clk					
# cycle	1	2	3	4	5

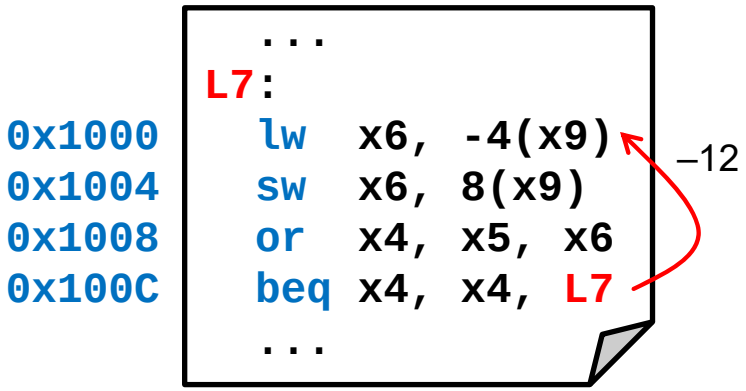


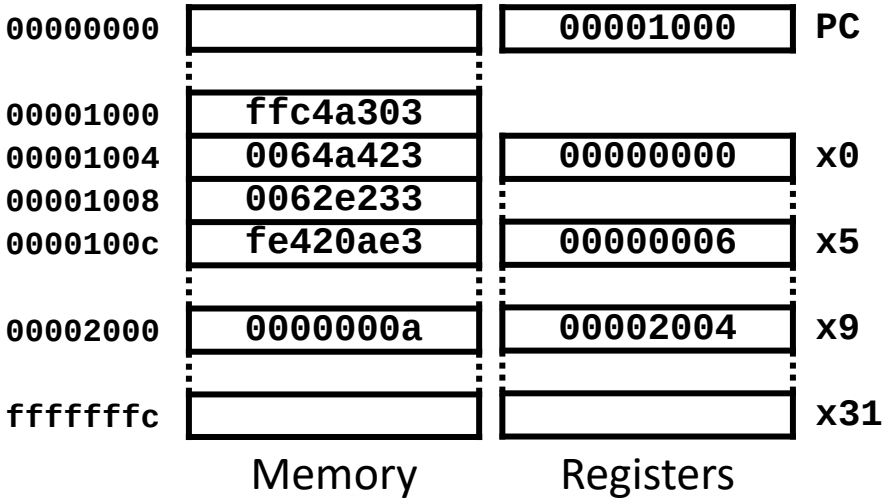


Execution diagram: registers and memory

	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		

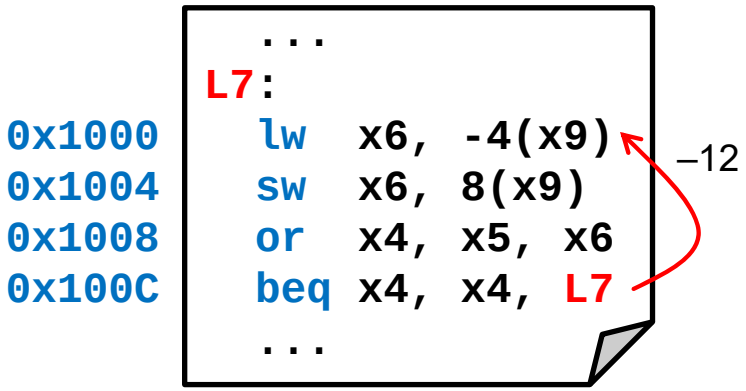


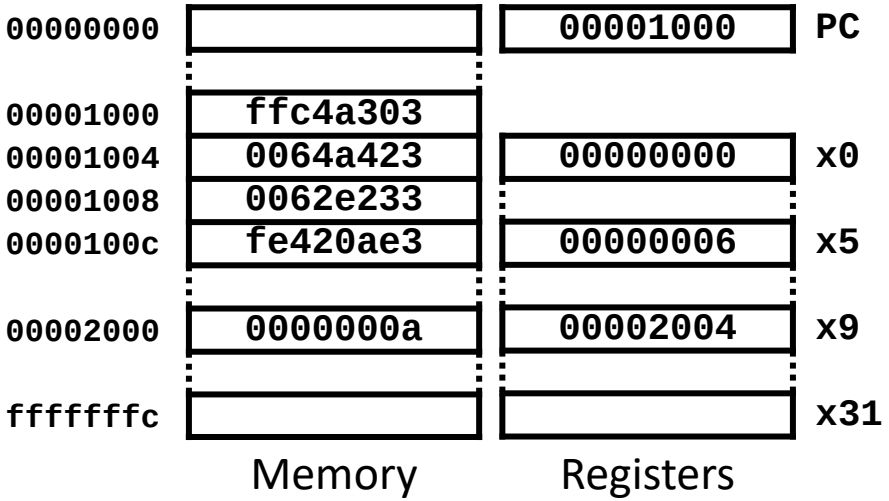




Execution diagram: registers and memory

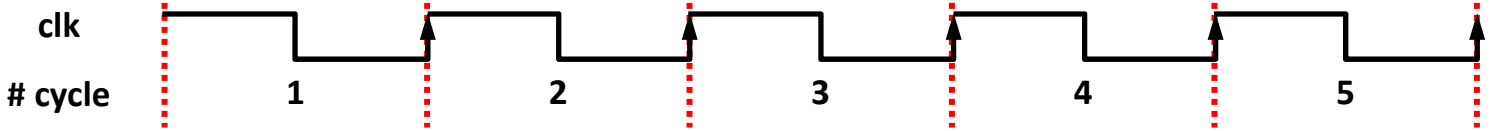
	lw x6,-4(x9)	sw x6,8(x9)	or x4,x5,x6	beq x4,x4,-12	lw x6,-4(x9)
PC	00001000	00001004	00001008	0000100c	
x4	????????	????????	????????	0000000e	
x5	00000006	00000006	00000006	=	
x6	????????	0000000a	0000000a	=	
X9	00002004	00002004	00002004	=	
MEM[0x2000]	0000000a	0000000a	0000000a	=	
MEM[0x200c]	????????	????????	0000000a	=	
clk					
# cycle	1	2	3	4	5

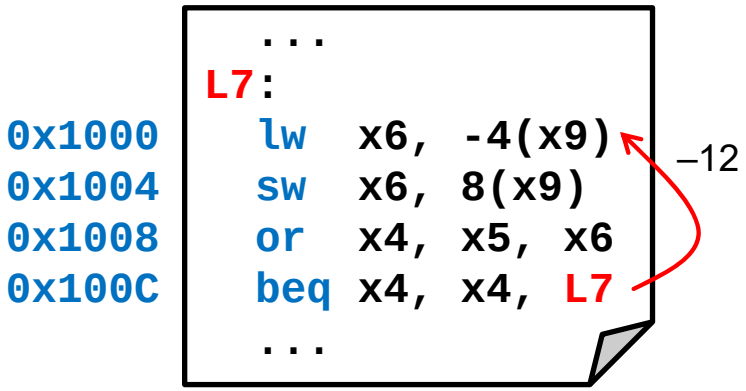


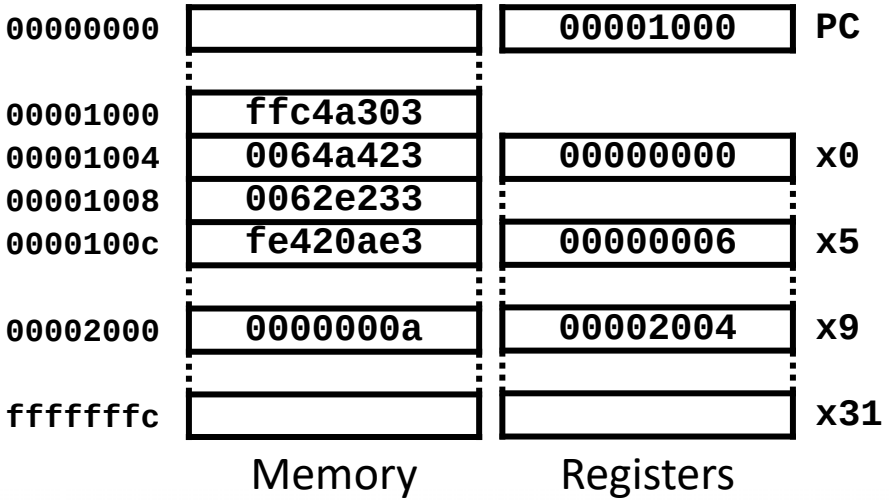


Execution diagram: registers and memory

	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	=

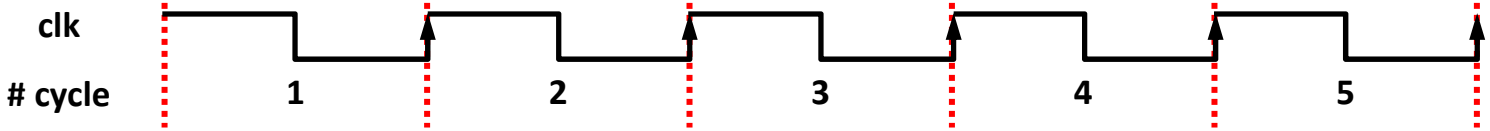






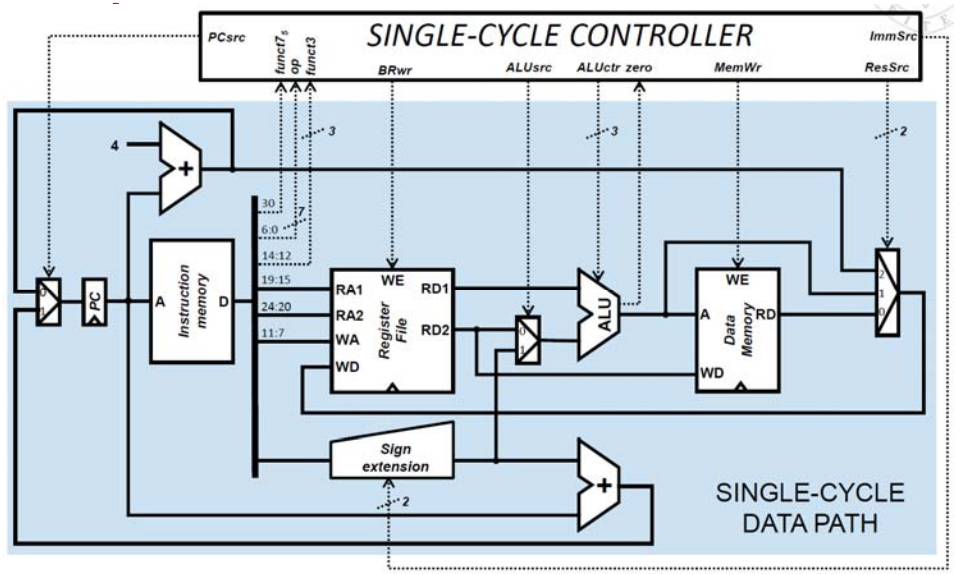
Execution diagram: registers and memory

	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



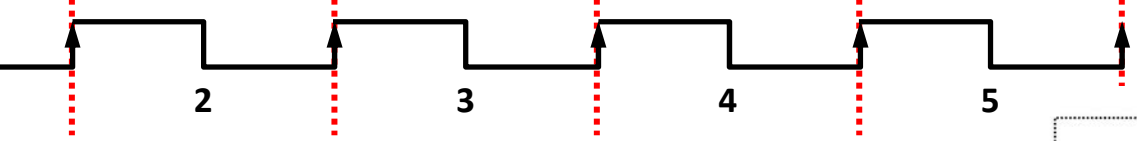
Execution diagram: status and control signals

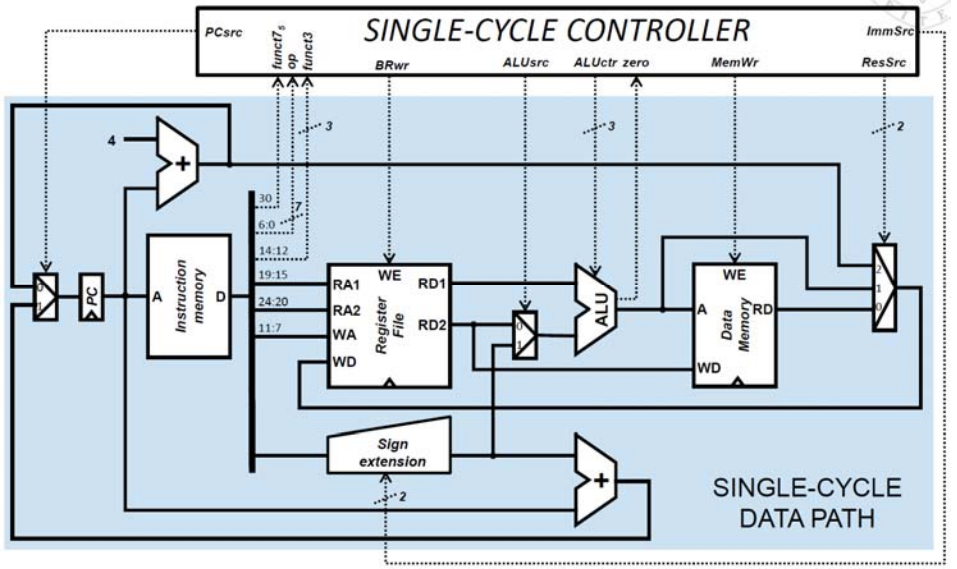
	`lw x6,-4(x9)`	`sw x6,8(x9)`	`or x4,x5,x6`	`beq x4,x4,-12`	`lw x6,-4(x9)`
op	0000011	0100011	0110011	1100011	0000011
$funct7_5$	–	–	0	–	–
funct3	010	010	110	000	010
zero	–	–	–	1	–
Branch					
Jump					
BRwr					
ALUsrc					
ALUop					
MemWr					
ResSrc					
InmSrc					
ALUctr					
PCsrc					
clk				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



Execution diagram: status and control signals

	`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





About *Creative Commons*

- CC license (Creative Commons)

  - This license enables reusers to distribute, remix, adapt, and build upon the material in any medium or format for noncommercial purposes only, and only so long as attribution is given to the creator. If you remix, adapt, or build upon the material, you must license the modified material under identical terms:

Attribution:
Credit must be given to the creator.

Non commercial:
Only noncommercial uses of the work are permitted.

Share alike:
Adaptations must be shared under the same terms.

More information: https://creativecommons.org/licenses/by-nc-sa/4.0/