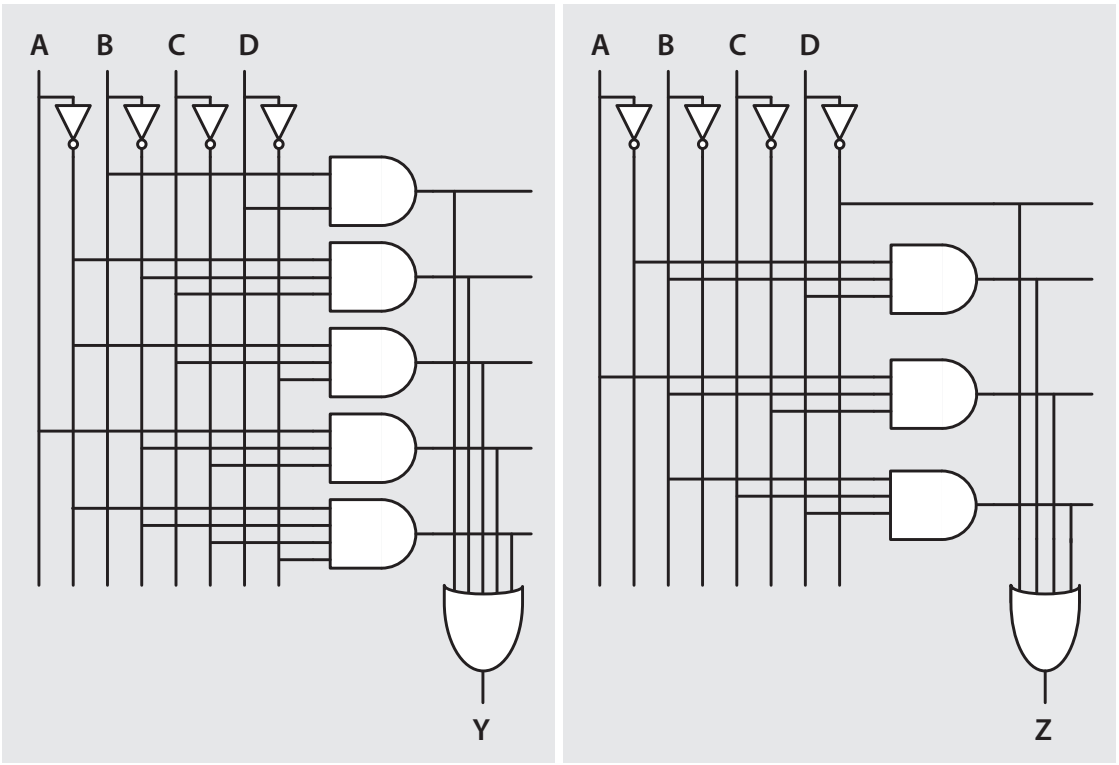
Computer Architecture
Homework 5 – final practice

Paul Mellies

Exercise 1.

§1a. Write the two Boolean equations for **Y** and **Z** in the circuit below. You do not need to minimize the equations at this stage.

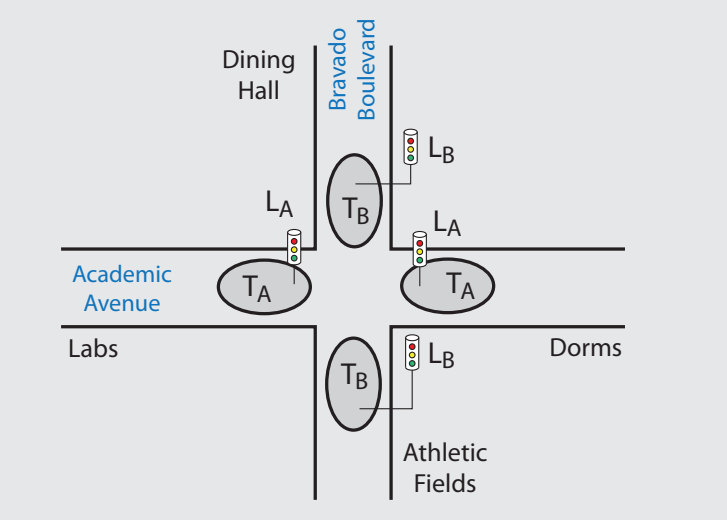


§1b. From this, use Karnaugh maps to minimize the two Boolean expressions **Y** and **Z** and draw an improved circuit equivalent to the original one for both **Y** and **Z**.

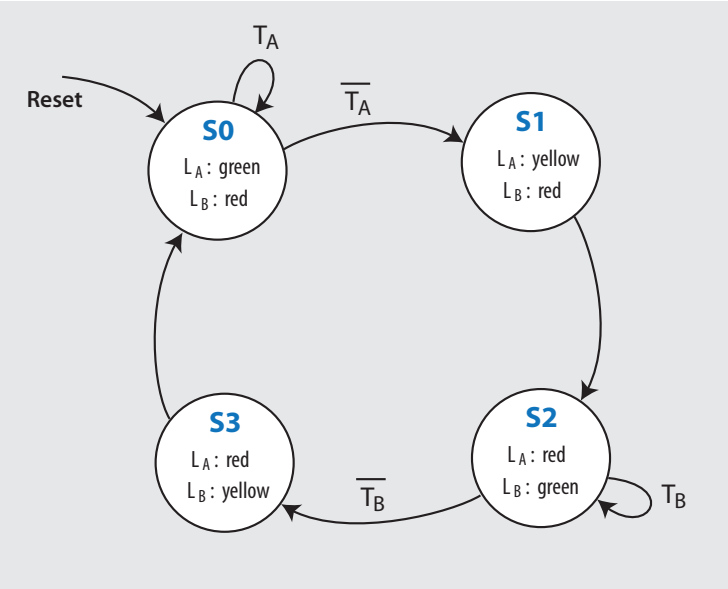
Besides your explanations, you are welcome to sketch the extended circuit on the diagram above.

Exercise 2.

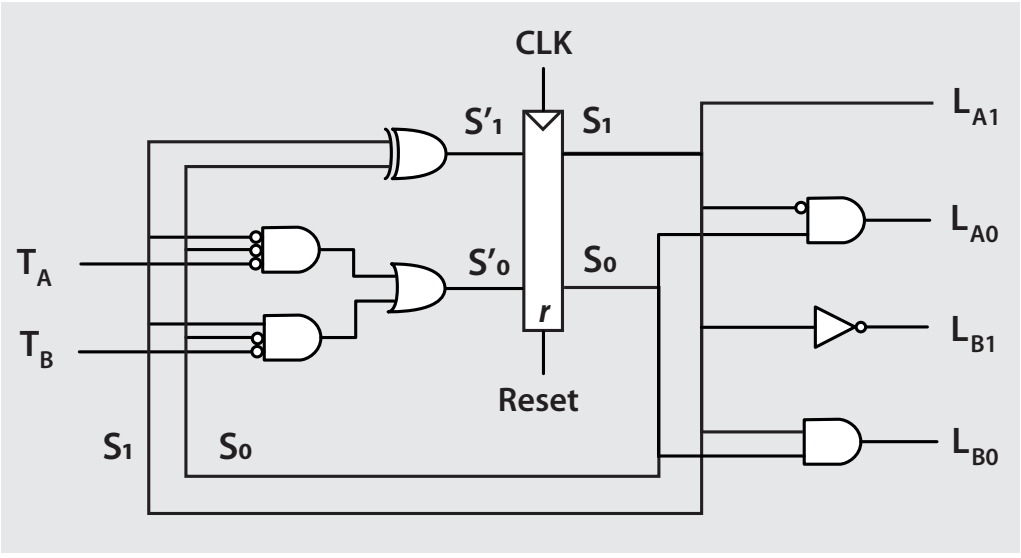
We recall the Moore machine studied in Lecture 13, whose purpose is to implement a traffic light controller between the Academy Avenue and the Bravado Boulevard, using sensors $\mathbf{T_A}$ and $\mathbf{T_B}$ on each street.



Recall that the finite state machine has the following state transition diagram:



which can be implemented by the following FSM schematics:

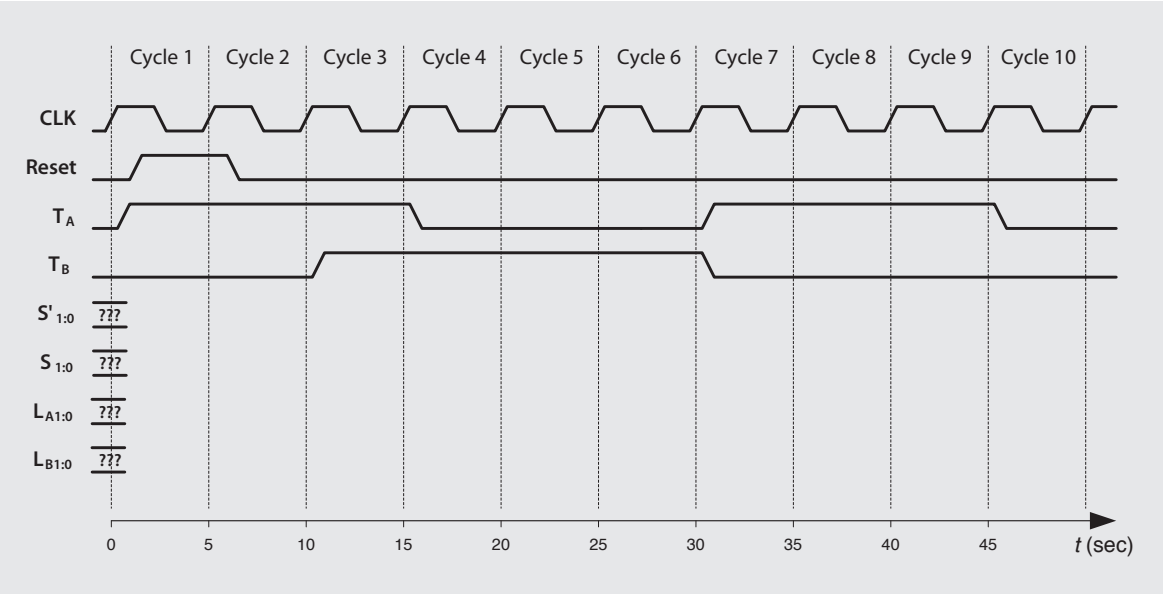


We recall the state and traffic light encodings used in the construction of the Moore machine:

State	Encoding $S_{1:0}$
S0	00
S1	01
S2	10
S3	11

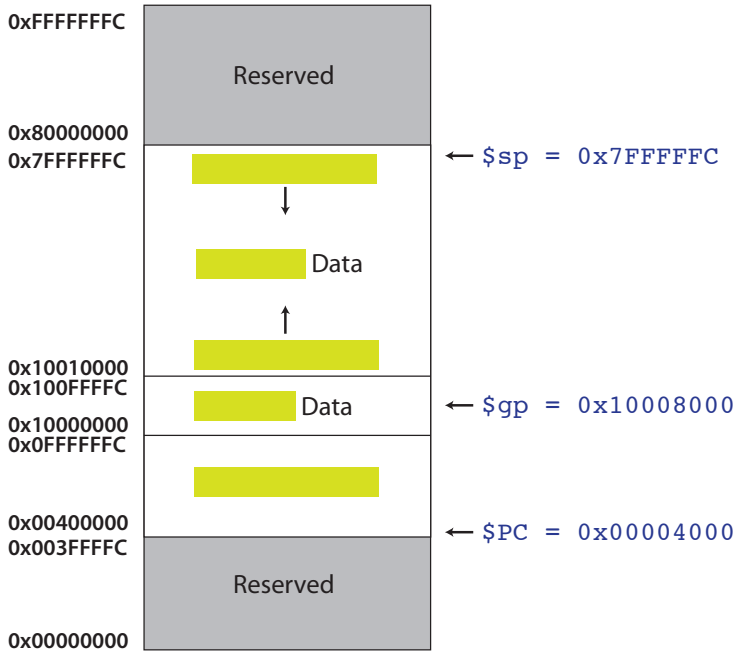
State	Encoding $L_{1:0}$
green	00
yellow	01
red	10

Complete the timing diagram below, in the same way as done during the course:



Exercise 3.

Five words have been hidden in the description of the memory map. Each of them is hidden by a green ribbon. Can you retrieve them and write them on the ribbon ?



Exercise 4.

Consider the MIPS assembly code below, where `func1`, `func2`, `func3` are non-leaf functions and `func4` is a leaf function.

```
0x00401000  func1 : ...          # func1 uses $s0-$s1
0x00401020          jal func2
   ...
0x00401100  func2 : ...          # func2 uses $s2-$s7
0x0040117C          jal func3
   ...
0x00401400  func3 : ...          # func3 uses $s1-$s3
0x00401704          jal func4
   ...
0x00403008  func4 : ...          # func4 uses no preserved
0x00403118          jr $ra$
```

§4a. How many words are the stack frames of each function?

§4b. Sketch the stack after `func4` is called. Clearly indicate which registers are stored on the stack and mark each of the stack frames. Give values where possible. Hint: we suppose that the value of the register `$sp` is equal to the hexadecimal number `0xFFFFFF00` when the function `func1` is called.

Exercise 5.

Just as studied in Lecture 13, the following recursive MIPS assembly code `factorial` implements the factorial function.

```
0x90 factorial: addi $sp, $sp, -8  # make room on stack
0x94            sw   $a0, 4($sp)   # store $a0
0x98            sw   $ra, 0($sp)   # store $ra
0x9C            addi $t0, $0, 2    # $t0 = 2
0xA0            slt  $t0, $a0, $t0 # n <= 1 ?
0xA4            beq  $t0, $0, else # no: goto else
0xA8            addi $v0, $0, 1    # yes: return 1
0xAC            addi $sp, $sp, 8   # restore $sp
0xB0            jr   $ra           # return
0xB4      else: addi $a0, $a0, -1  # n = n - 1
0xB8            jal  factorial     # recursive call
0xBC            lw   $ra, 0($sp)   # restore $ra
0xC0            lw   $a0, 4($sp)   # restore $a0
0xC4            addi $sp, $sp, 8   # restore $sp
0xC8            mul  $v0, $a0, $v0 # n * factorial(n-1)
0xCC            jr   $ra           # return
```

Suppose that one deletes the instructions at addresses `0x98` and `0xBC` which save and restore `$ra`, and obtains the following MIPS code for the `factorial2` function:

```
0x90 factorial2: addi $sp, $sp, -4  # make room on stack
0x94             sw   $a0, 0($sp)   # store $a0
0x98             addi $t0, $0, 2    # $t0 = 2
0x9C             slt  $t0, $a0, $t0 # n <= 1 ?
0xA0             beq  $t0, $0, else # no: goto else
0xA4             addi $v0, $0, 1    # yes: return 1
0xA8             addi $sp, $sp, 4   # restore $sp
0xAC             jr   $ra           # return
0xB0       else: addi $a0, $a0, -1  # n = n - 1
0xB4             jal  factorial2    # recursive call
0xB8             lw   $a0, 0($sp)   # restore $a0
0xBC             addi $sp, $sp, 4   # restore $sp
0xC0             mul  $v0, $a0, $v0 # n * factorial(n-1)
0xC4             jr   $ra           # return
```

§5a. Explain whether the `factorial2` code called with the register `$a0` as argument

1. enters an infinite loop but does not crash;
2. crashes (typically causes the stack to grow beyond the dynamic data segment or the PC to jump to a location outside the program);
3. produces an incorrect value in the register `$v0` when the program returns (if so, what value?),
4. or produces the correct result despite the deleted lines?

Justify your answer by describing sufficiently precisely the execution of the program.

§5b. Repeat the exercise §6a. with the `factorial3` code obtained by removing from `factorial` the instruction at address `0xAC` which restores the register `$sp`:

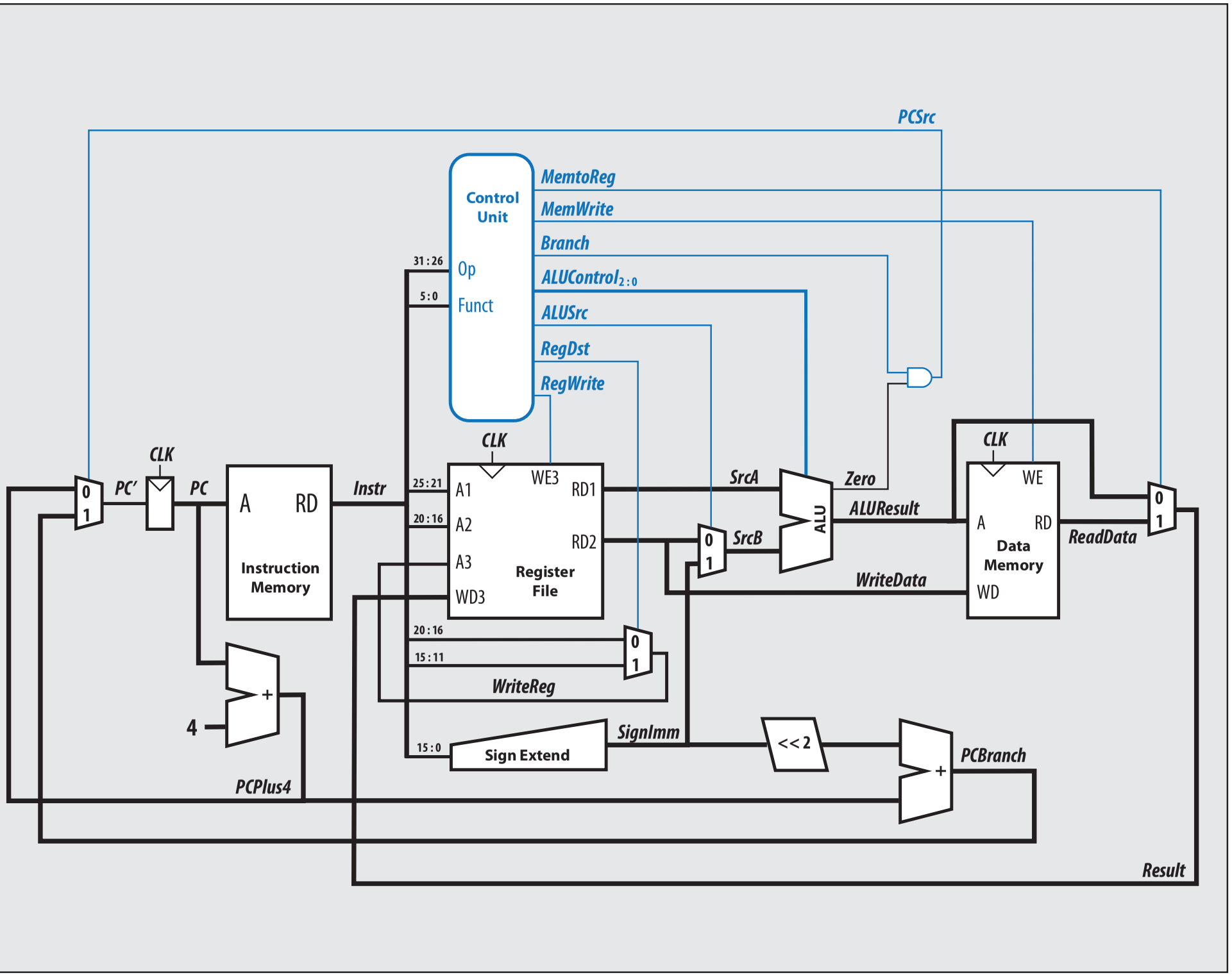
```
0x90 factorial3: addi $sp, $sp, -8  # make room on stack
0x94             sw   $a0, 4($sp)   # store $a0
0x98             sw   $ra, 0($sp)   # store $ra
0x9C             addi $t0, $0, 2    # $t0 = 2
0xA0             slt  $t0, $a0, $t0 # n <= 1 ?
0xA4             beq  $t0, $0, else # no: goto else
0xA8             addi $v0, $0, 1    # yes: return 1
0xAC             jr   $ra           # return
0xB0       else: addi $a0, $a0, -1  # n = n - 1
0xB4             jal  factorial3    # recursive call
0xB8             lw   $ra, 0($sp)   # restore $ra
0xBC             lw   $a0, 4($sp)   # restore $a0
0xC0             addi $sp, $sp, 8   # restore $sp
0xC4             mul  $v0, $a0, $v0 # n * factorial(n-1)
0xC8             jr   $ra           # return
```

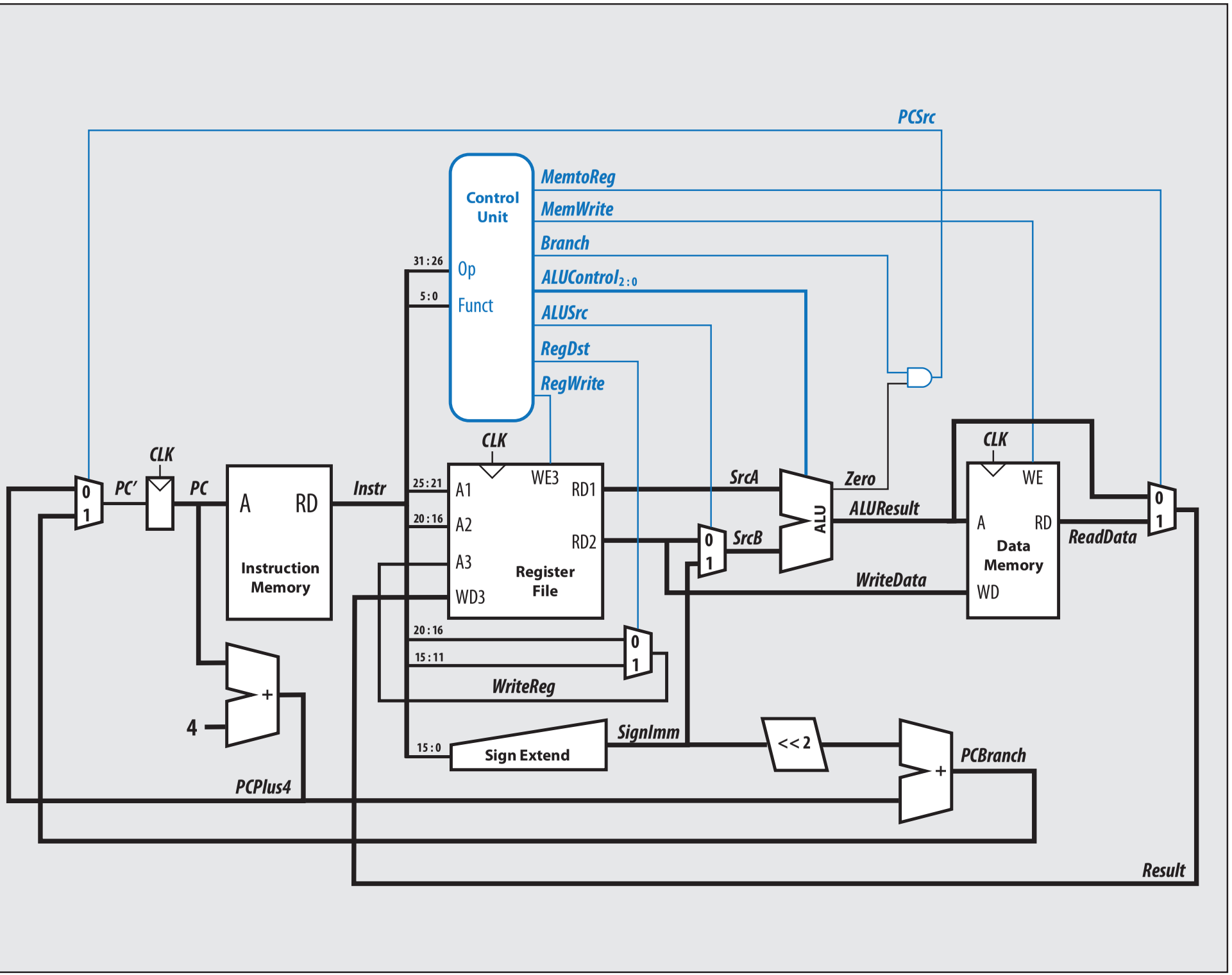
Exercise 6.

Draw the datapath of the `lw` instruction on the picture of the MIPS single-cycle microarchitecture which you will find next page. Indicate in particular the values of the control flags. Note: the encoding of the instructions on the 3-bit wire $\mathbf{ALUControl}_{2:0}$ coincides with the encoding of the ALU instructions in Exercise 2.

Exercise 7.

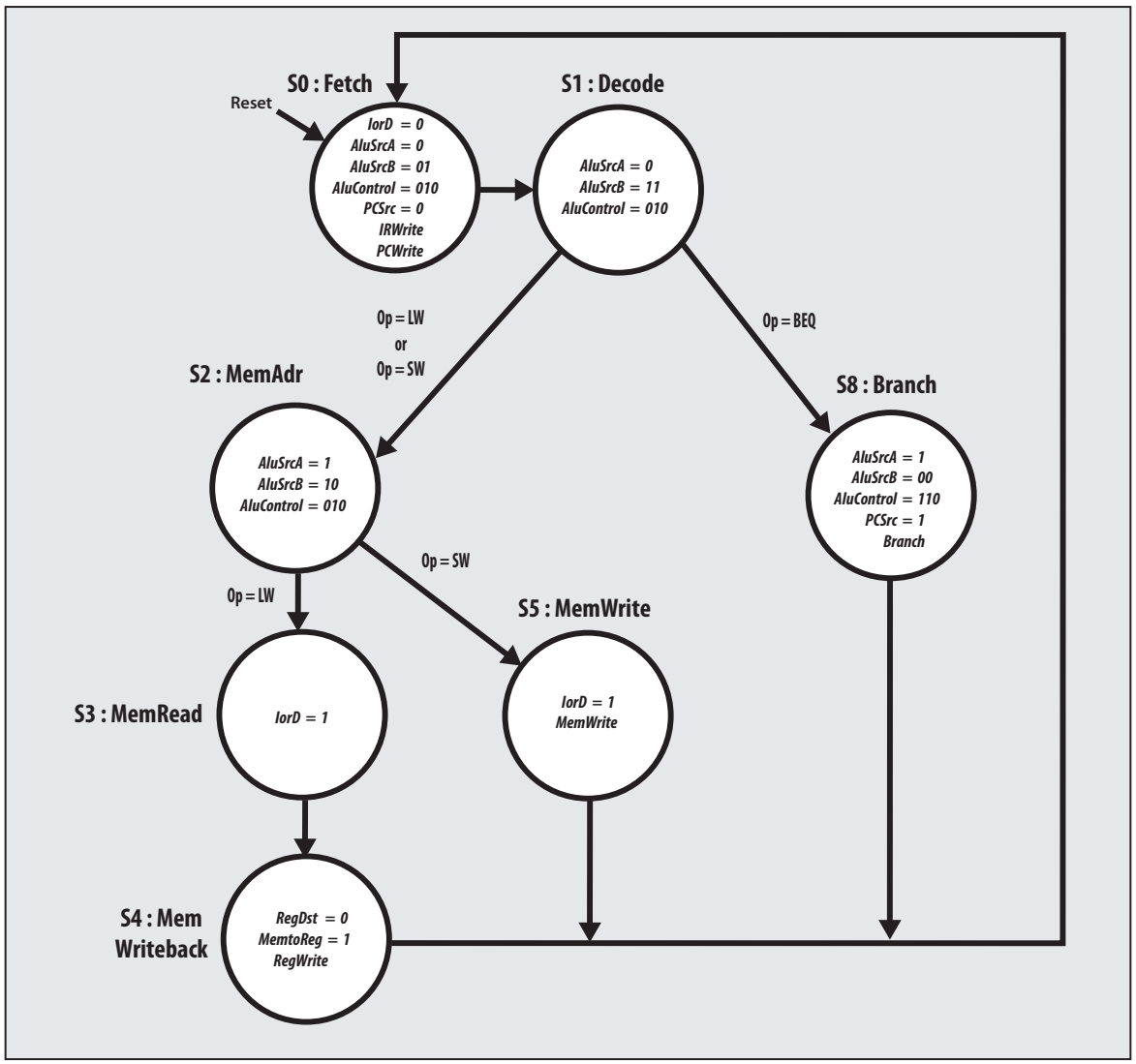
Draw the datapath of the `beq` instruction on the picture of the MIPS single-cycle microarchitecture which you will find next page. Indicate in particular the values of the control flags. Note: the encoding of the instructions on the 3-bit wire $\mathbf{ALUControl}_{2:0}$ coincides with the encoding of the ALU instructions in Exercise 2.


PCSrc
Control Unit
Op
Funct
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
31:26
5:0
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
WD
Data Memory
ReadData
0
1
20:16
15:11
WriteReg
4
+
PCPlus4
15:0
Sign Extend
SignImm
<<2
PCBranch
Result


PCSrc
Control Unit
Op
Funct
MemtoReg
MemWrite
Branch
ALUControl2:0
ALUSrc
RegDst
RegWrite
31:26
5:0
CLK
PC'
PC
A
RD
Instruction Memory
Instr
25:21
20:16
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
WD
Data Memory
ReadData
0
1
20:16
15:11
WriteReg
4
+
PCPlus4
15:0
Sign Extend
SignImm
<<2
PCBranch
Result

Exercise 8.

We recall below the transition diagram of the multicycle control Finite State Machine of the MIPS processor.



§8a. Use the circuits below to draw the datapaths corresponding to the six states

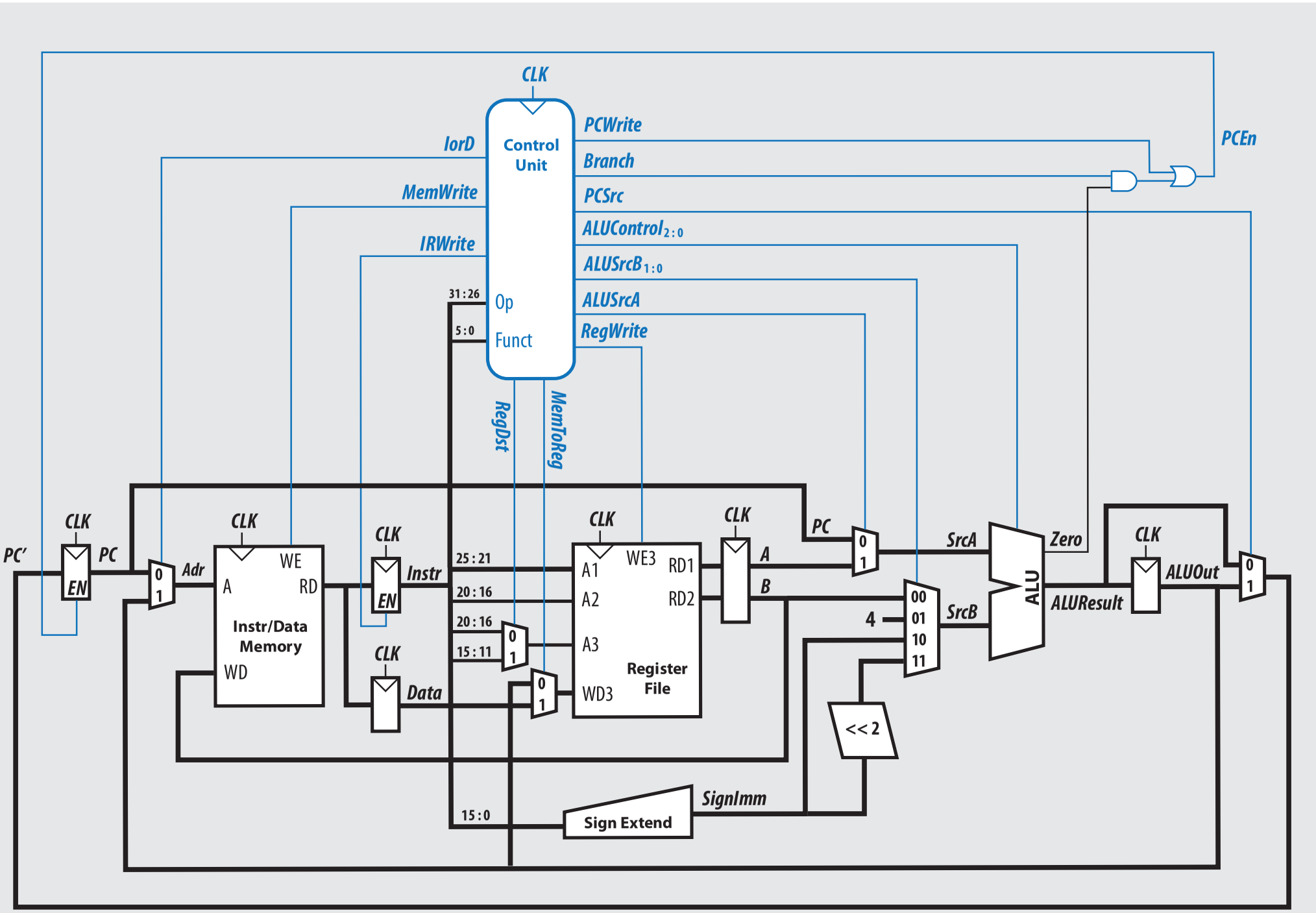
$$S0 \quad S1 \quad S2 \quad S3 \quad S4 \quad S5$$

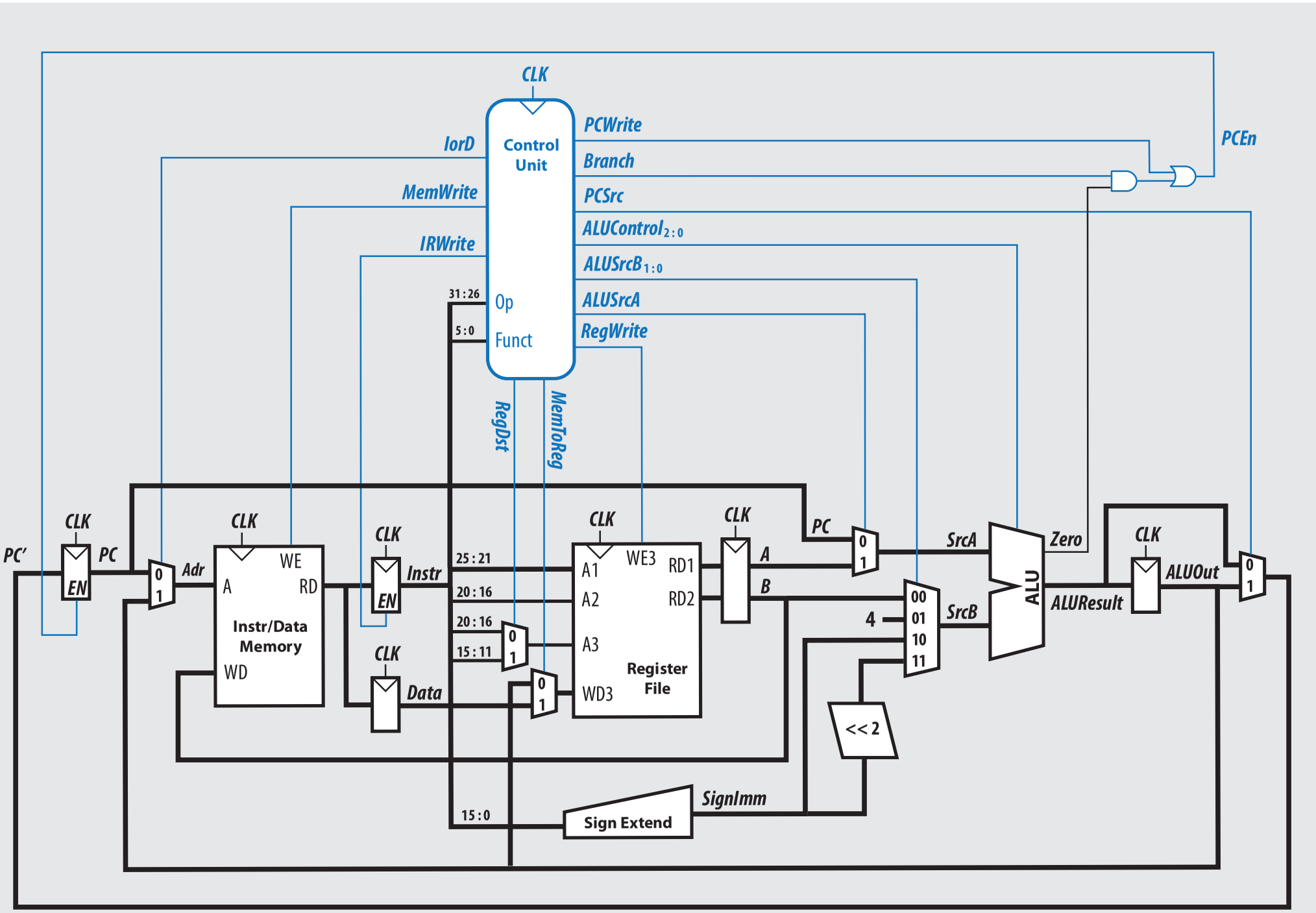
implementing the `lw` and the `sw` instructions. Explain in each case what the datapath does.

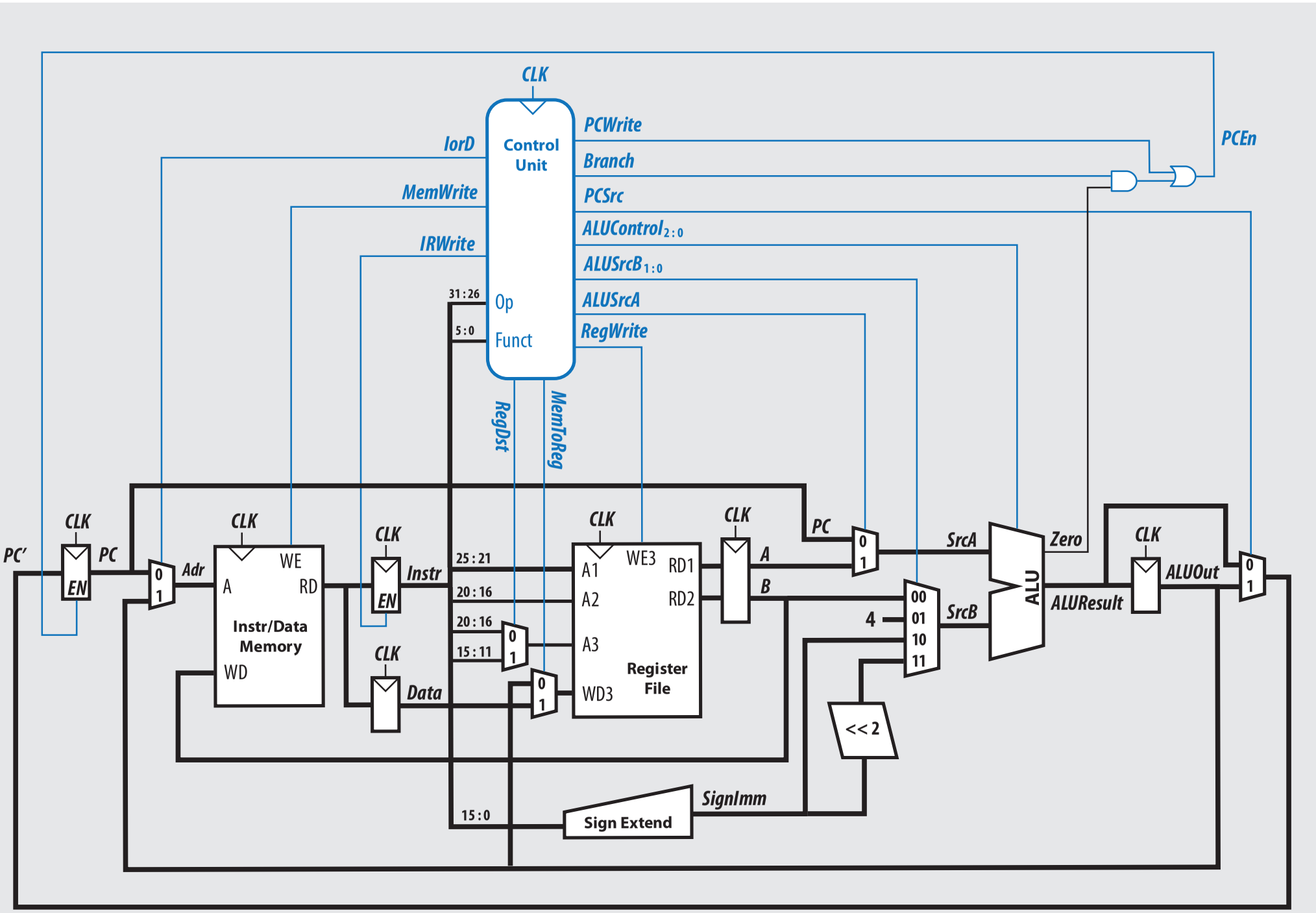
§8b. Draw the datapath of the state $S8$ and explain why the decode state $S1$ is required to produce the control signals

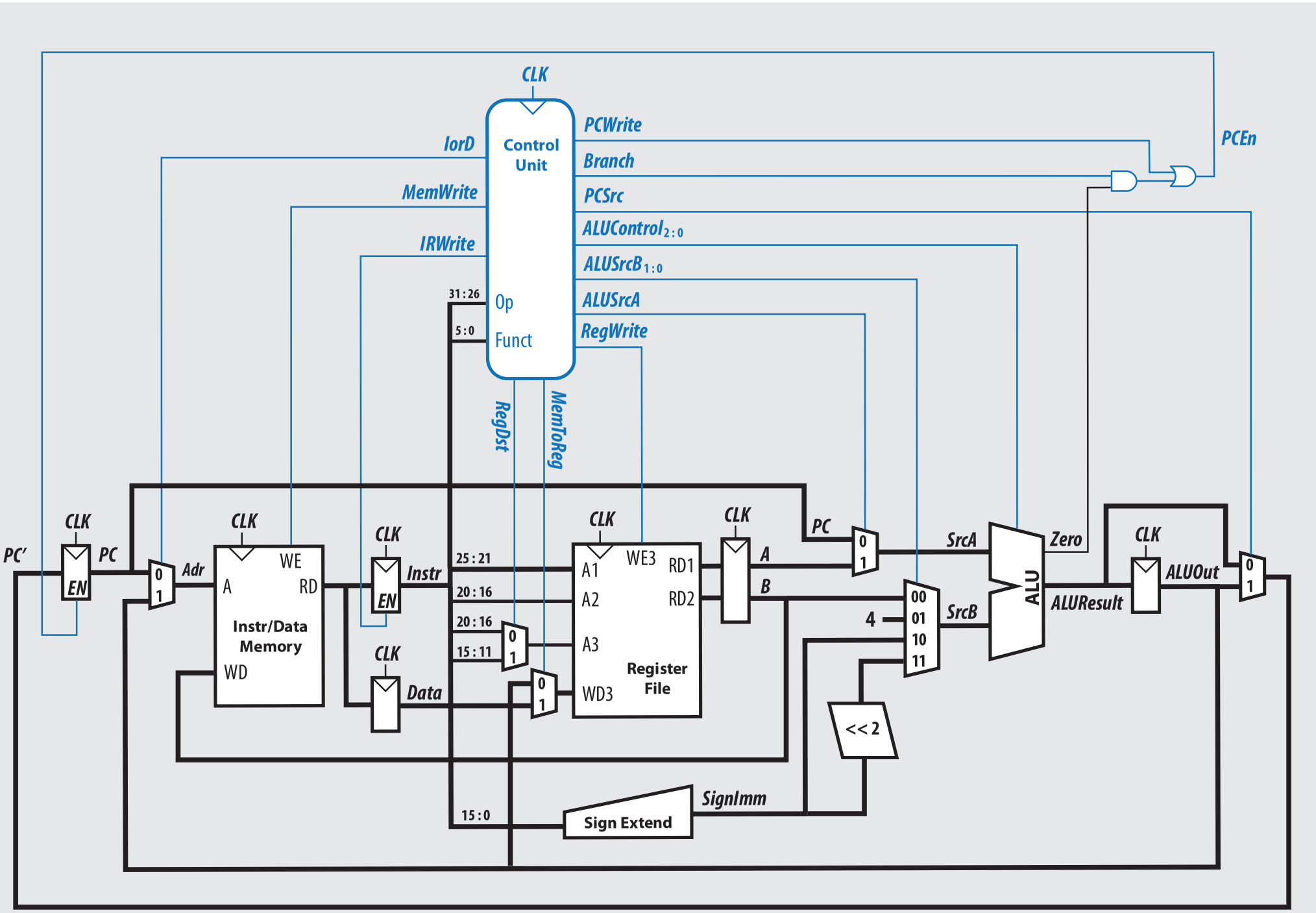
$$AluSrcA = 0 \qquad AluSrcB = 11 \qquad AluControl = 010$$

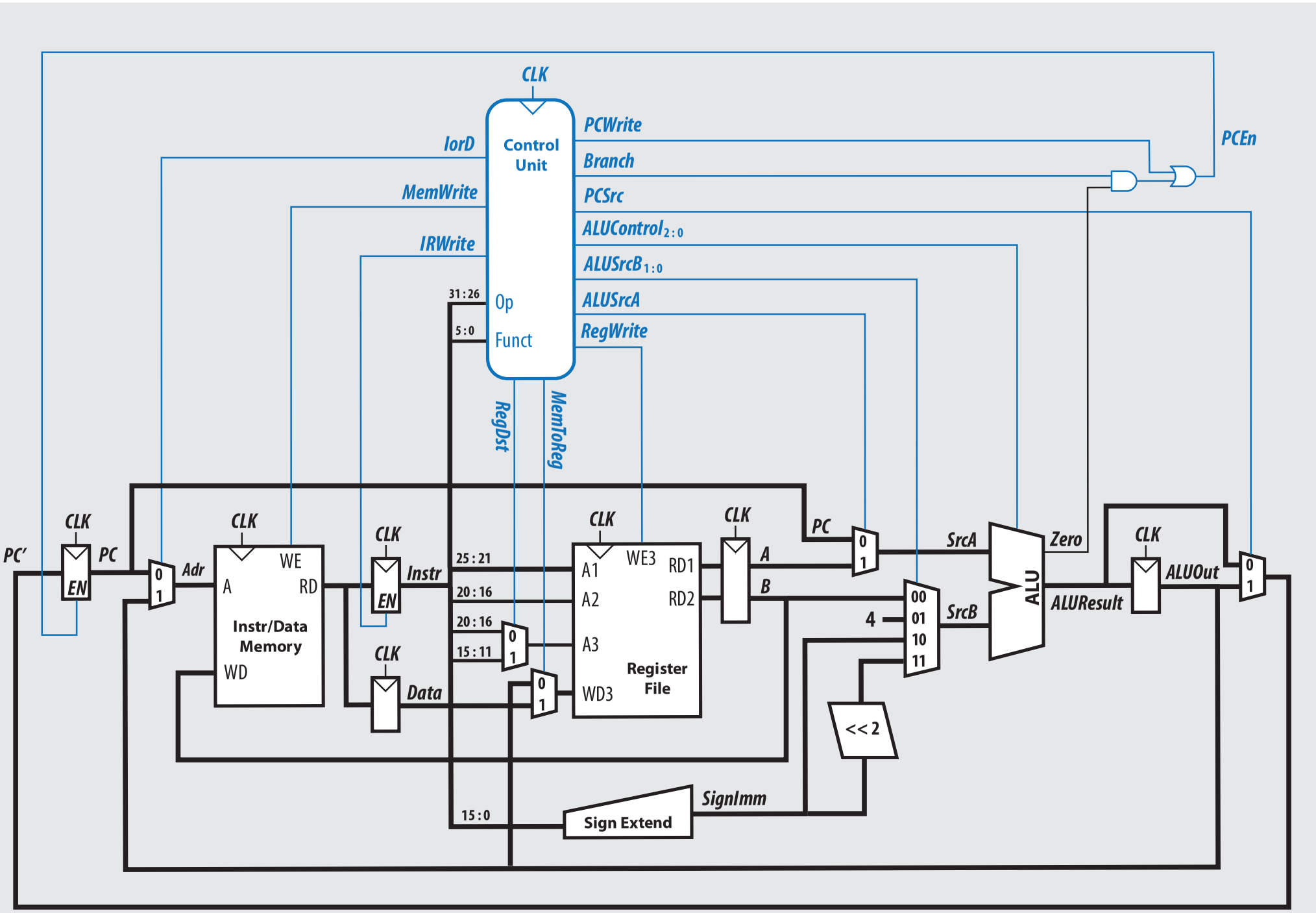
in order to implement the beq instruction.

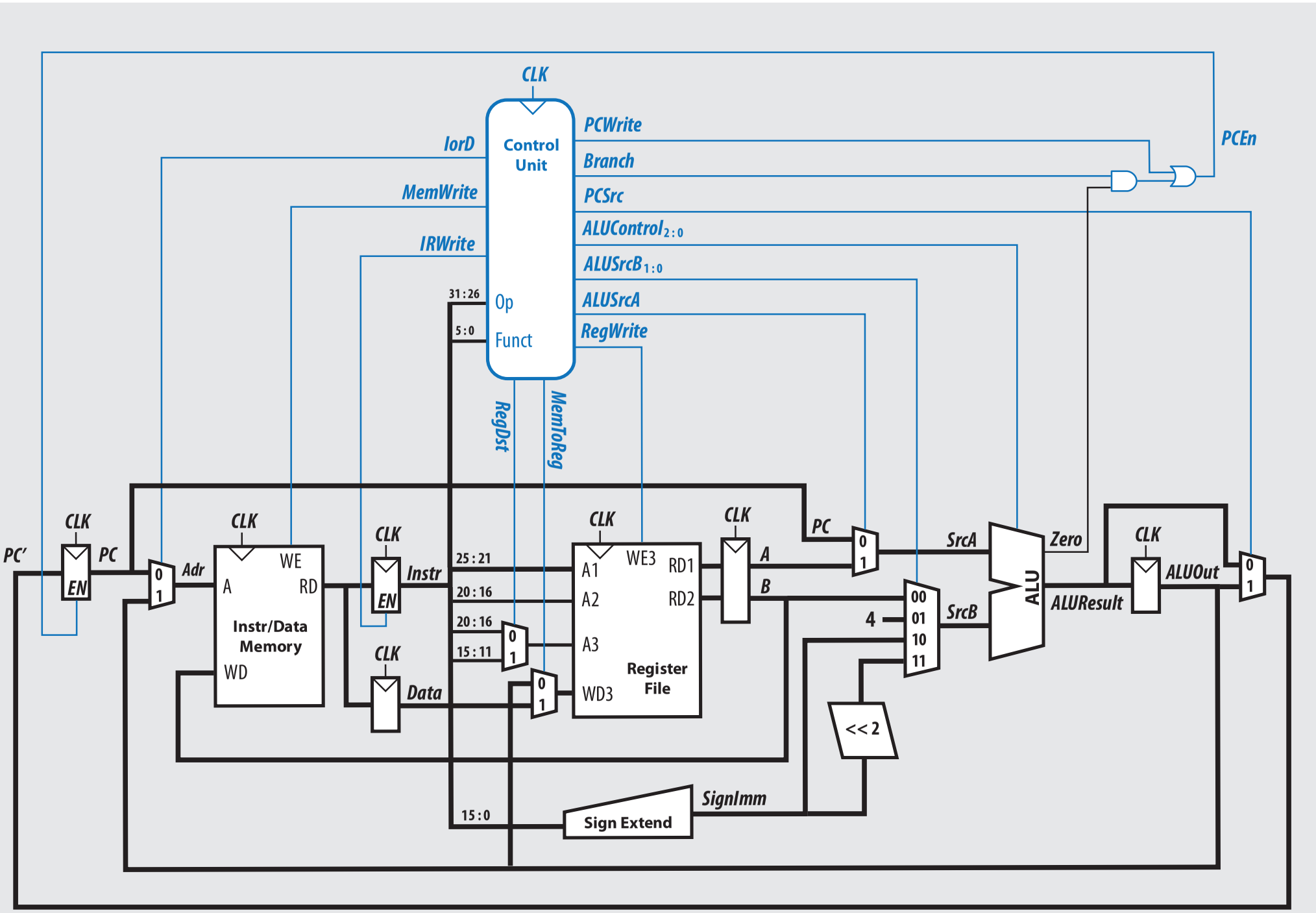

CLK
Control Unit
IorD
MemWrite
IRWrite
31:26
5:0
Op
Funct
PCWrite
Branch
PCSrc
ALUControl2:0
ALUSrcB1:0
ALUSrcA
RegWrite
RegDst
MemToReg
PCEn
PC'
PC
EN
Adr
A
WE
RD
WD
Instr/Data Memory
Instr
Data
25:21
20:16
20:16
15:11
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
A
B
PC
SrcA
SrcB
00
01
10
11
4
ALU
Zero
ALUResult
ALUOut
<< 2
15:0
Sign Extend
SignImm
0
1


CLK
Control Unit
IorD
MemWrite
IRWrite
31:26
5:0
Op
Funct
PCWrite
Branch
PCSrc
ALUControl2:0
ALUSrcB1:0
ALUSrcA
RegWrite
RegDst
MemToReg
PCEn
PC'
PC
EN
Adr
A
WE
RD
WD
Instr/Data Memory
Instr
Data
25:21
20:16
20:16
15:11
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
A
B
PC
SrcA
SrcB
00
01
10
11
4
ALU
Zero
ALUResult
ALUOut
<< 2
15:0
Sign Extend
SignImm
0
1


CLK
Control Unit
IorD
MemWrite
IRWrite
31:26
5:0
Op
Funct
PCWrite
Branch
PCSrc
ALUControl2:0
ALUSrcB1:0
ALUSrcA
RegWrite
RegDst
MemToReg
PCEn
PC'
PC
EN
Adr
A
WE
RD
WD
Instr/Data Memory
Instr
Data
25:21
20:16
20:16
15:11
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
A
B
PC
SrcA
SrcB
00
01
10
11
4
ALU
Zero
ALUResult
ALUOut
<< 2
15:0
Sign Extend
SignImm
0
1


CLK
Control Unit
IorD
MemWrite
IRWrite
31:26
5:0
Op
Funct
PCWrite
Branch
PCSrc
ALUControl2:0
ALUSrcB1:0
ALUSrcA
RegWrite
RegDst
MemToReg
PCEn
PC'
PC
EN
Adr
A
WE
RD
WD
Instr/Data Memory
Instr
Data
25:21
20:16
20:16
15:11
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
A
B
PC
SrcA
SrcB
00
01
10
11
4
ALU
Zero
ALUResult
ALUOut
<< 2
15:0
Sign Extend
SignImm
0
1


CLK
Control Unit
IorD
MemWrite
IRWrite
31:26
5:0
Op
Funct
PCWrite
Branch
PCSrc
ALUControl2:0
ALUSrcB1:0
ALUSrcA
RegWrite
RegDst
MemToReg
PCEn
PC'
PC
EN
Adr
A
WE
RD
WD
Instr/Data Memory
Instr
Data
25:21
20:16
20:16
15:11
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
A
B
PC
SrcA
SrcB
00
01
10
11
4
ALU
Zero
ALUResult
ALUOut
<< 2
15:0
Sign Extend
SignImm
0
1


CLK
Control Unit
IorD
MemWrite
IRWrite
31:26
5:0
Op
Funct
PCWrite
Branch
PCSrc
ALUControl2:0
ALUSrcB1:0
ALUSrcA
RegWrite
RegDst
MemToReg
PCEn
PC'
PC
EN
Adr
A
WE
RD
WD
Instr/Data Memory
Instr
Data
25:21
20:16
20:16
15:11
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
A
B
PC
SrcA
SrcB
00
01
10
11
4
ALU
Zero
ALUResult
ALUOut
<< 2
15:0
Sign Extend
SignImm
0
1