Computer Architecture and Programming Languages
Jacobs University Bremen
Jovan Shandro

Course: CO20-320241
November 18, 2019

# Homework 9

## Problem 9.1

**Solution:**

**a**) In a single cycle datapath the PC does not need an explicit write signal because it is always updated at the end of each instruction

**b**) In a multi cycle datapath every instruction can take more than one cycle, therefore it is required for the PC to get updated at the end of each instruction, which means that a write signal is needed for each update of the PC.

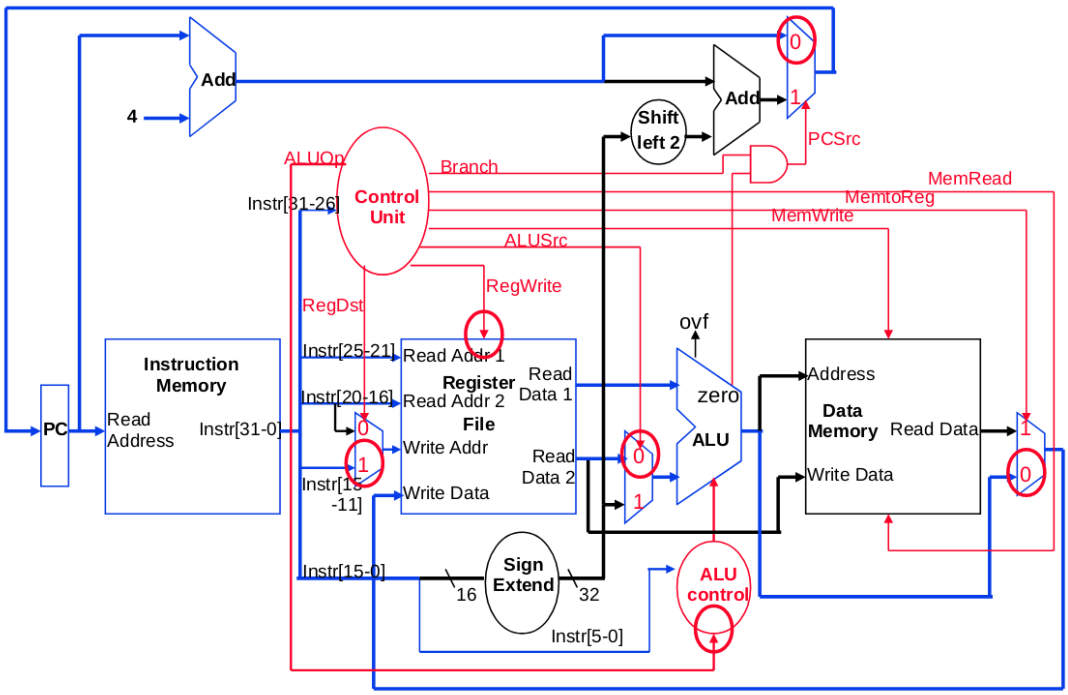
## Problem 9.2

**Solution:**

**a.i**) For the instruction

**add** $s0, $s1, $s2

the single cycle datapath goes as follows:



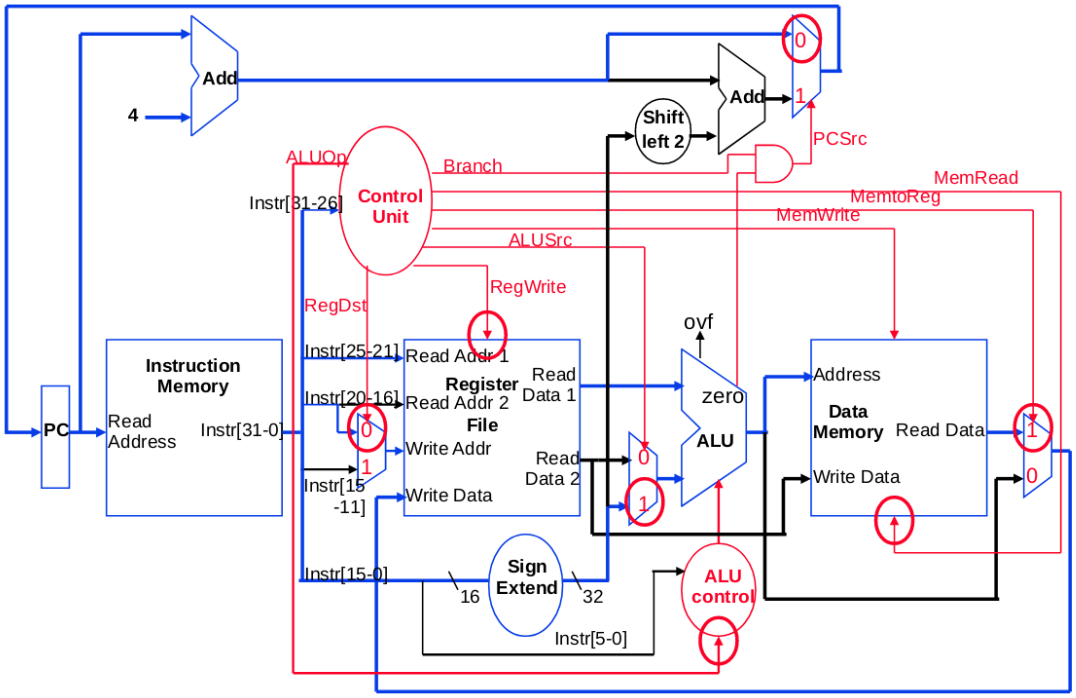

Now, considering the values of control lines, as we have an R-type instruction and there is no branching and no need for using memory, Branch, MemRead, and MemWrite are all 0. Since the instruction has a destination register ($s0), RegDst is set to 1, and since the result of the addition is going to be written in this destination register, RegWrite is also set to 1. Since the ALU will directly write the sum in the destination register, MemtoReg is also 0. At last, as **add** is an R type inst. ALUOp is set to 10. So the complete table of values is:

| ALUOp | Branch | MemRead | MemtoReg | MemWrite | ALUSrc | RegWrite | RegDst |
|---|---|---|---|---|---|---|---|
| 10 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

**a.ii**) For the instruction

**lw** $s3, 16($s2)

the single cycle datapath goes as follows:




Now, as above, considering the values of control lines, as we have an I-type instruction and the destination register is in rt (so rd is not used), RegDst is 0, and since there is no branching, Branch is also 0. There is no need for saving to memory, but the instruction reads from it, so MemRead is 1 and MemWrite is 0, and as the values are loaded from memory, MemtoReg is also 1. The result will be written in the 'destination' register (in this case rt), so RegWrite is also 1. At last, as **lw** is an I type inst. ALUOp is set to 00. So the complete table of values is:

| ALUOp | Branch | MemRead | MemtoReg | MemWrite | ALUSrc | RegWrite | RegDst |
|---|---|---|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |

**b**) There may be different scenarios in which the ALU will need to add its inputs, some examples (explained thoroughly below) would include the **add** instruction in which it would need to add the values stored in the registers, and also in the **lw** to compute the address (in our example above $s2 + 4 · 16). A detailed explanation would be:

**i**) For the add instruction (which is of type R), we know the opcode is 0 and the funct code is 32. ALUOp is 10 and from the table in slide 59 in the presentation from lectures 16 17, we deduce that the ALU operation is 0010 which represents addition.

**ii**) For the lw instruction (which is of type I), ALUOp is 00 (again as seen in point a) and from the table in slide 59 in the presentation from lectures 16 17, we deduce that the ALU operation is 0010 which represents addition.

**P.S: Slide 58 in the presentation above also states that for both instructions, lw, and add, ALU action is addition, above I simply rephrased it.**