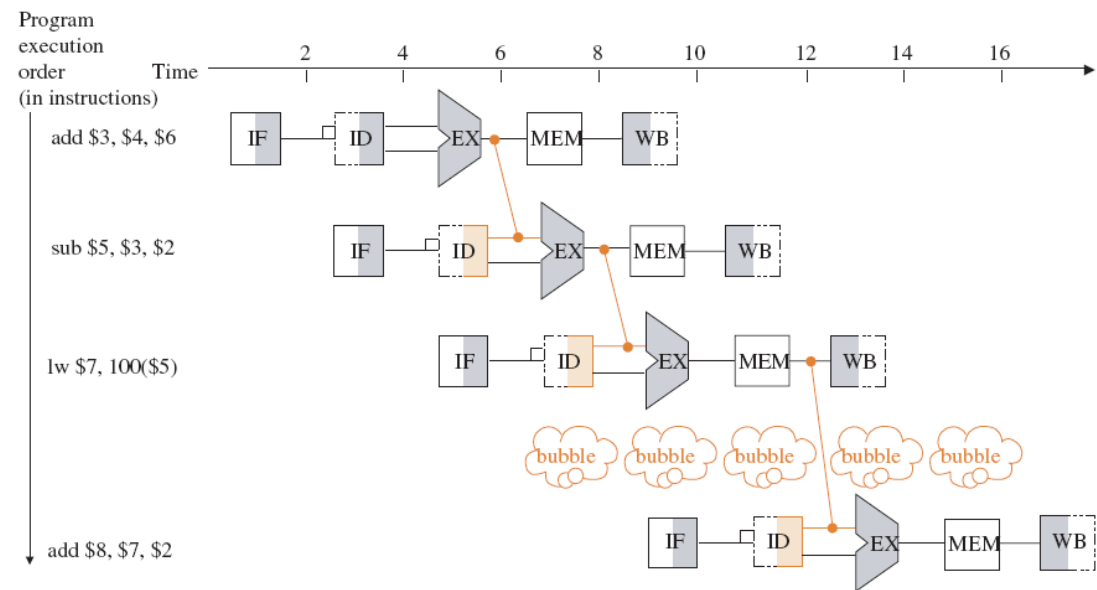


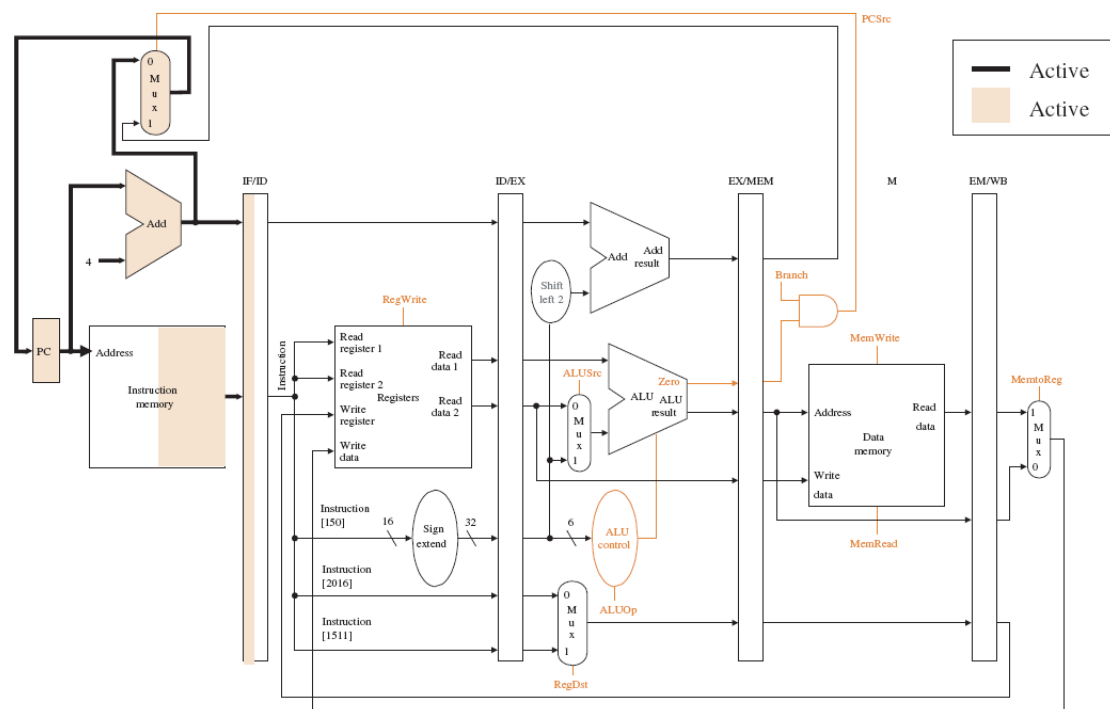
# Answers to Homework 3

6.3(10)

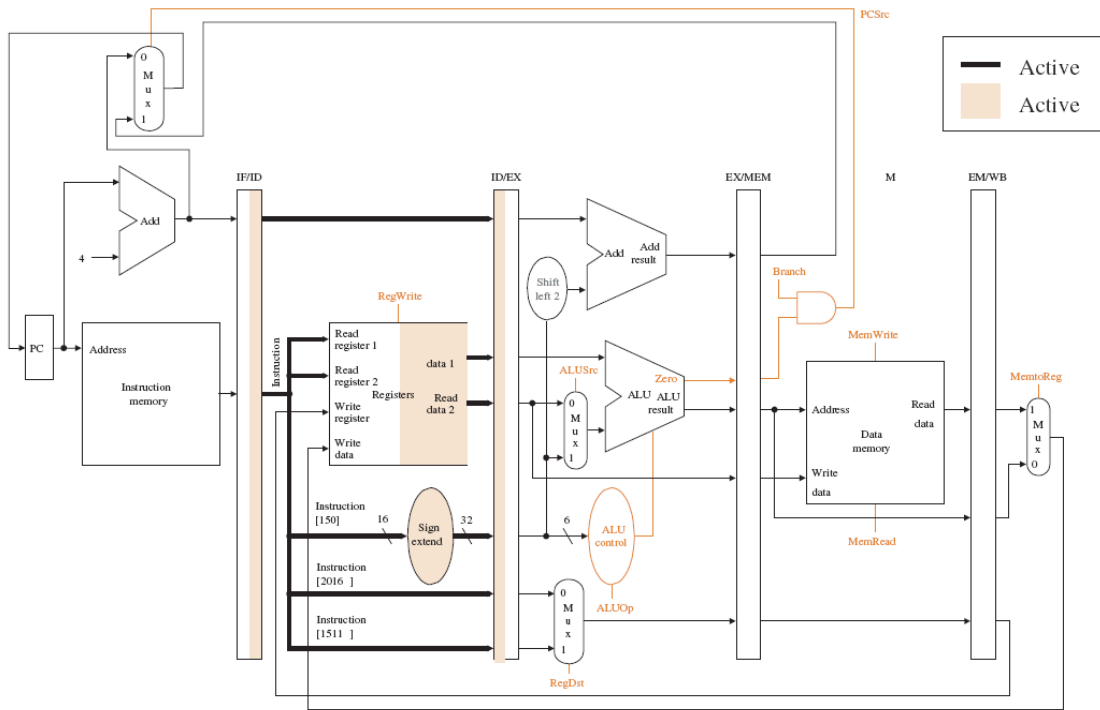


6.6(20)

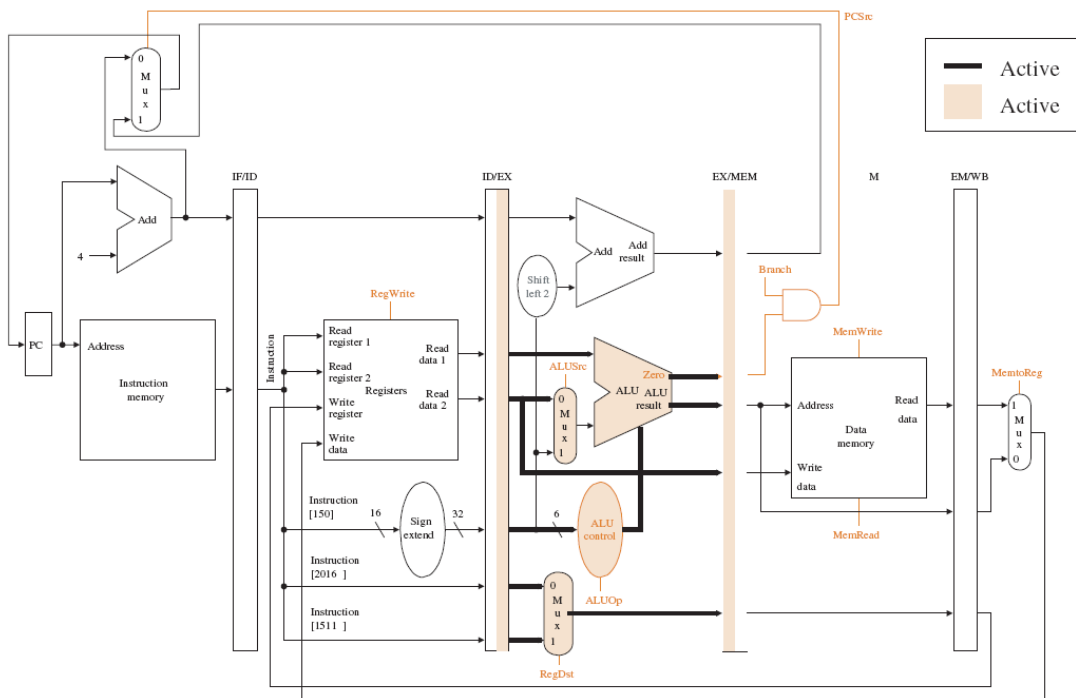
Stage One:



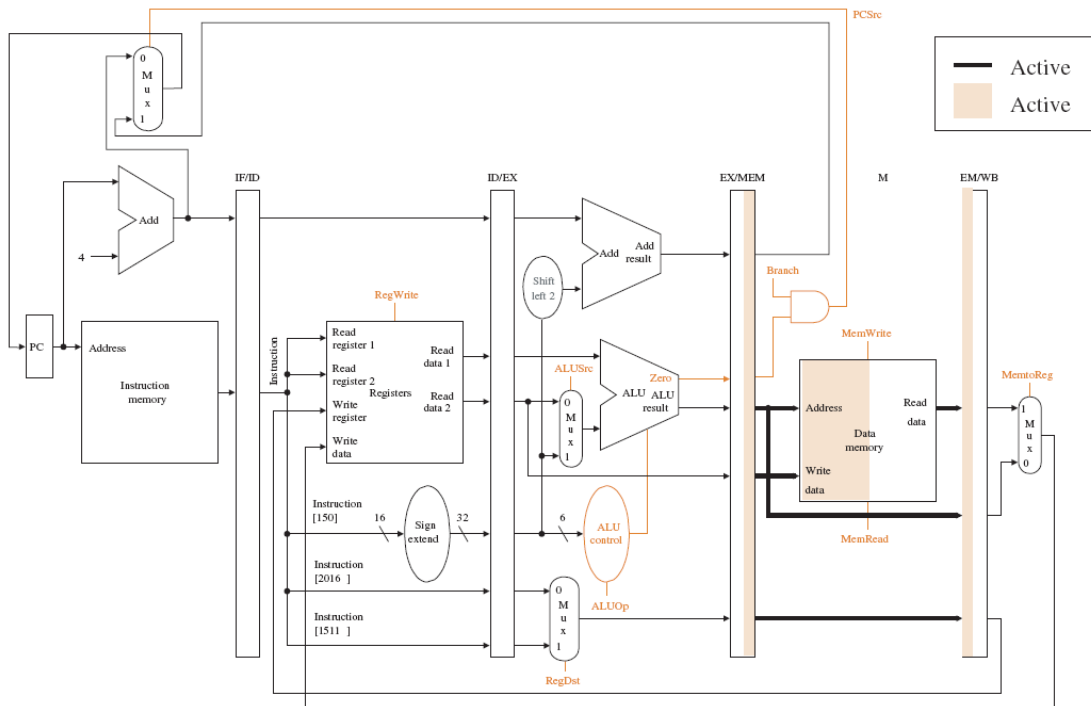
### Stage Two:



### Stage Three:

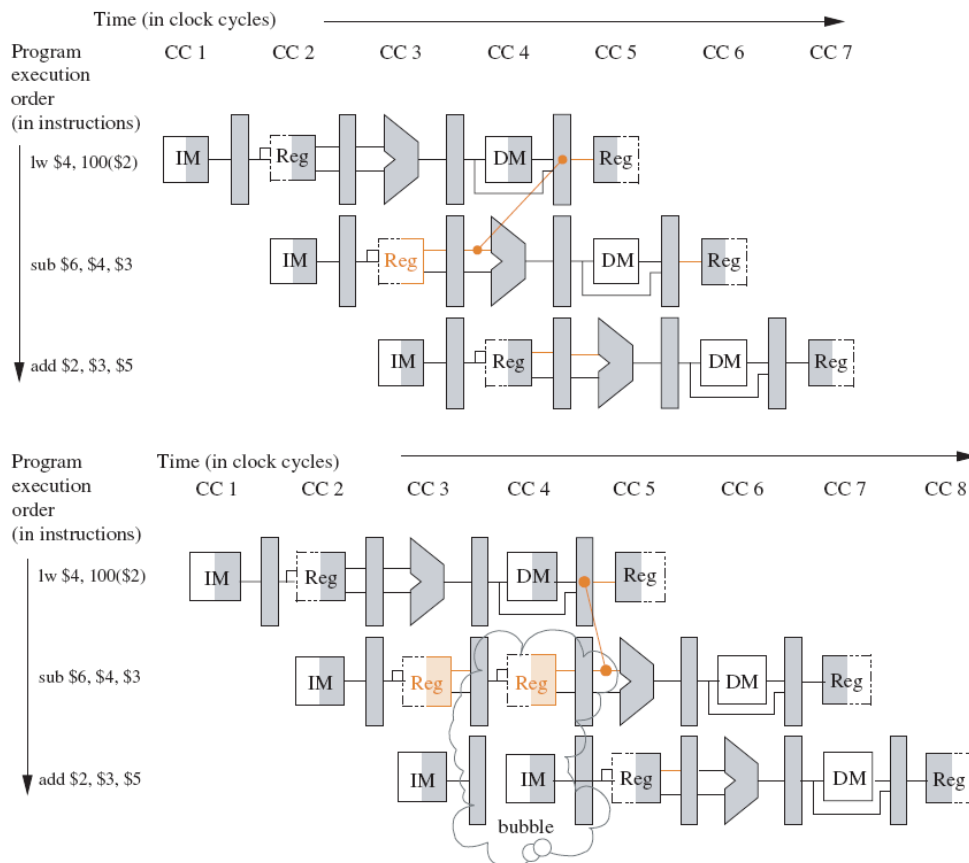


### Stage Four:



Since this is an sw instruction, there is no work done in the WB stage.

### 6.22(10)



It will take 8 cycles to execute this code, including a bubble of 1 cycle due to the dependency between the `lw` and `sub` instructions.

6.34(10)

Branches take 1 cycle when predicted correctly, 3 cycles when not (including one more memory access cycle). So the average clock cycle per branch is  $0.75 * 1 + 0.25 * 3 = 1.5$ .

For loads, if the instruction immediately following it is dependent on the load, the load takes 3 cycles. If the next instruction is not dependent on the load but the second following instruction is dependent on the load, the load takes two cycles. If neither two following instructions are dependent on the load, the load takes one cycle.

The probability that the next instruction is dependent on the load is 0.5. The probability that the next instruction is not dependent on the load, but the second following instruction is dependent, is  $0.5 * 0.25 = 0.125$ . The probability that neither of the two following instructions is dependent on the load is 0.375.

Thus the effective CPI for loads is  $0.5 * 3 + 0.125 * 2 + 0.375 * 1 = 2.125$ .

Using the data from the example on page 425, the average CPI is  $0.25 * 2.125 + 0.10 * 1 + 0.52 * 1 + 0.11 * 1.5 + 0.02 * 3 = 1.47$ .

Average instruction time is  $1.47 * 100\text{ps} = 147\text{ ps}$ . The relative performance of the restructured pipeline to the single-cycle design is  $600/147 = 4.08$ .

6.41( extra credit 10)

The store instruction can read the value from the register if it is produced at least 3 cycles earlier. Therefore, we only need to consider forwarding the results produced by the two instructions right before the store. When the store is in EX stage, the instruction 2 cycles ahead is in WB stage. The instruction can be either a `lw` or an ALU instruction.

```
assign EXMEMrt = EXMEMIR[20:16];
assign bypassVfromWB = (IDEXop == SW) & (IDEXrt != 0) &
    ( ((MEMWBop == LW) & (IDEXrt == MEMWBrt)) |
      ((MEMWBop == ALUop) & (IDEXrt == MEMWBrd)) );
```

This signal controls the store value that goes into EX/MEM register. The value produced by the instruction 1 cycle ahead of the store can be bypassed from the MEM/WB register. Though the value from an ALU instruction is available 1 cycle earlier, we need to wait for the load instruction anyway.

```
assign bypassVfromWB2 = (EXMEMop == SW) & (EXMEMrt != 0) & (!bypassVfromWB) &
    ( ((MEMWBop == LW) & (EXMEMrt == MEMWBrt)) |
      ((MEMWBop == ALUop) & (EXMEMrt == MEMWBrd)) );
```

This signal controls the store value that goes into the data memory and MEM/WB register.