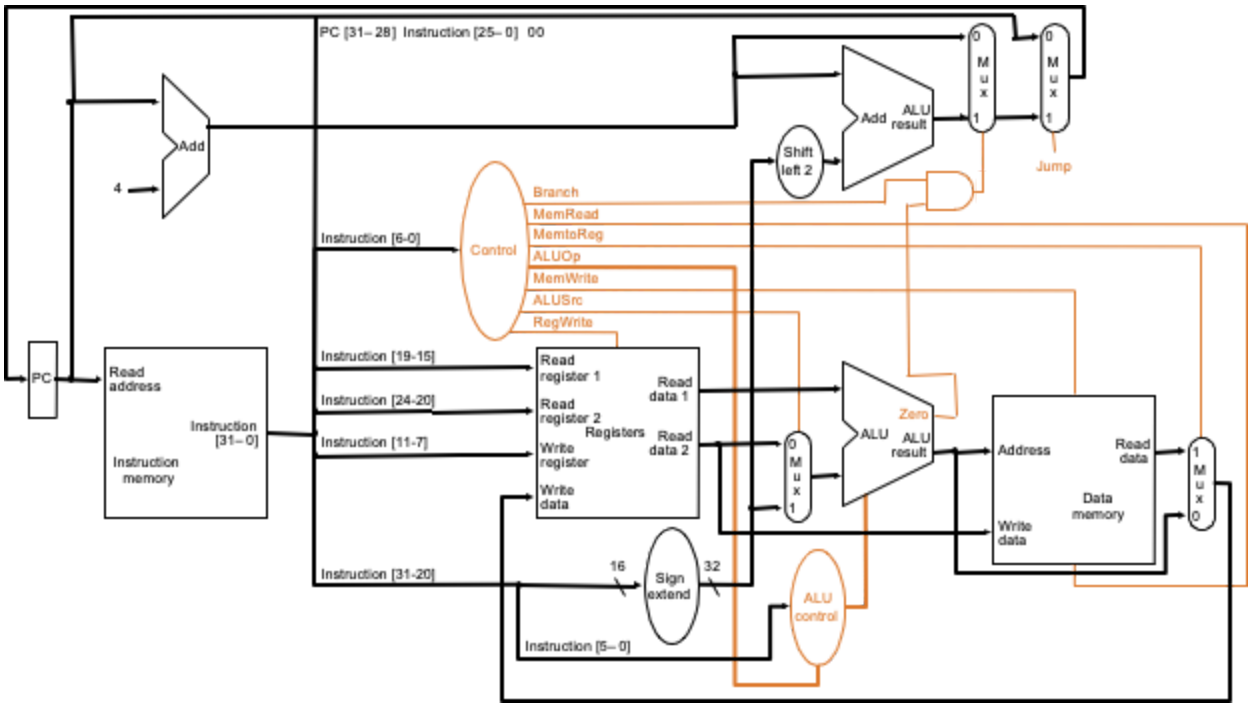
# CMPE110 HA3 Solutions

# Pipeline Detective




1) Consider the single cycle processor shown above. Determine the control signals as they are generated by the Control logic block by filling in the table below. For each instruction, set the four control signals to either 1 (enable), 0 (disable) or X (don't care). Don't care means: for a specific instruction the control signal can be either 0 or 1. **(4 Points)**

| Instruction | Opcode | RegWrite | Branch | MemToReg/ RegDst | ALUSrc |
|---|---|---|---|---|---|
| beq | 1100011 | **0** | **1** | **X** | **0** |
| addi | 0010011 | **1** | **0** | **0** | **1** |
| sub | 0110011 | **1** | **0** | **0** | **0** |
| lb | 0000011 | **1** | **0** | **1** | **1** |
| sb | 0100011 | **0** | **0** | **X** | **1** |

2) Determine the boolean combinational logic expression of opcode bits to generate the control signals. Fill in the table below. Assume only the 5 instructions above (ignore all other instructions RISC-V may define). Use Op[x] to select a particular bit from the opcode. Enumerate the opcode bits from right to left, e.g. given the opcode 0100011, op[0]=1, op[1]=1, op[2]=0, op[3]=0, op[4]=0, op[5]=1, op[6]=0. An example expression using the correct notation could be: (!op[0] & op[2]) || (op[3] & !op[4]). Try to minimize your expression (use the smallest required number of opcode bits to generate a signal). **(4 Points)**

| Control Signal | Expression |
|---|---|
| RegWrite | **op[4] || (!op[4] & !op[5])** |
| Branch | **op[6]** |
| MemToReg/RegDst | **!op[4]** |
| ALUSrc | **(!op[4] || !op[5]) & !op[6]** |

3) Using two sentences, describe the difference between a data dependency and a data hazard. **(2 Points)**

**In a data dependency situation, an instruction is dependent on a result from a sequentially previous instruction in order to complete its execution. Data hazards occur when the instructions are close enough that the overlap caused by pipelining could change the order of accesses to an operand.**

4) Consider the following instruction sequence. Determine all RAW dependencies in the code below and list them in the table **(2 Points)**

| Instruction | RAW Dependencies |
|---|---|
| Load x1, 0(x31) | |
| Load x2, 8(x31) | |
| Add x3, x1, x2 | **x1 x2** |
| Mul x5, x1, x3 | **x1 x3** |
| Sub x5, x5, x3 | **x3 x5** |
| Load x6, 16(x31) | |

| | |
|---|---|
| Add x6, x5, x6 | **x5 x6** |
| Sub x7, x8, x9 | |
| Div x7, x7, x8 | **x7** |

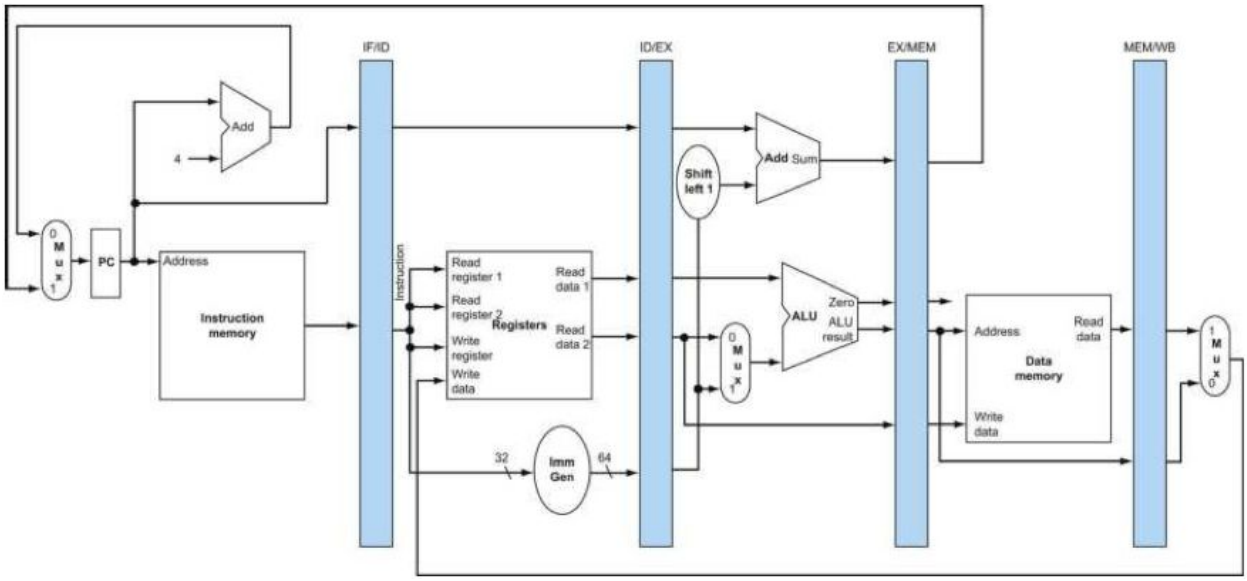



Figure 1: 5-stage Pipeline without forwarding

5) Consider the 5-stage pipeline design we discussed in the lecture, shown in Figure 1, **without forwarding** paths. Insert the minimum number of required NOPs into the instruction stream **without re-ordering** instructions to avoid all hazards. Calculate the average CPI of this code, assuming a CPI of one for all instructions. Assume that registers from the register file can be written and read in the same cycle. **(4 Points)**

| # | Instruction |
|---|---|
| **1** | **Load x1, 0(x31)** |
| **2** | **Load x2, 8(x31)** |
| **3** | **NOP** |
| **4** | **NOP** |
| **5** | **Add x3, x1, x2** |

| | |
|---|---|
| **6** | **NOP** |
| **7** | **NOP** |
| **8** | **Mul x5, x1, x3** |
| **9** | **NOP** |
| **10** | **NOP** |
| **11** | **Sub x5, x5, x3** |
| **12** | **Load x6, 16(x31)** |
| **13** | **NOP** |
| **14** | **NOP** |
| **15** | **Add x6, x5, x6** |
| **16** | **Sub x7, x8, x9** |
| **17** | **NOP** |
| **18** | **NOP** |
| **19** | **Div x7, x7, x8** |

CPI of this code: **23/9**

6) Consider the 5-stage pipeline design we discussed in the lecture, shown in Figure 1, **without forwarding paths**. Optimize execution time by **re-ordering** instructions and then insert the minimum number of required NOPs into the instruction stream. The re-ordering must be correct, i.e. not change the output result of the code when applying transformations. Calculate the average CPI of this code, assuming a CPI of one for all instructions. Assume that registers from the register file can be written and read in the same cycle. **(4 Points)**

| # | Instruction |
|---|---|
| **1** | **Load x1, 0(x31)** |
| **2** | **Load x2, 8(x31)** |
| **3** | **Load x6, 16(x31)** |
| **4** | **Sub x7, x8, x9** |

| | |
|---|---|
| **5** | **Add x3, x1, x2** |
| **6** | **NOP** |
| **7** | **NOP** |
| **8** | **Mul x5, x1, x3** |
| **9** | **NOP** |
| **10** | **NOP** |
| **11** | **Sub x5, x5, x3** |
| **12** | **Div x7, x7, x8** |
| **13** | **NOP** |
| **14** | **Add x6, x5, x6** |

CPI of this code: **18/9**

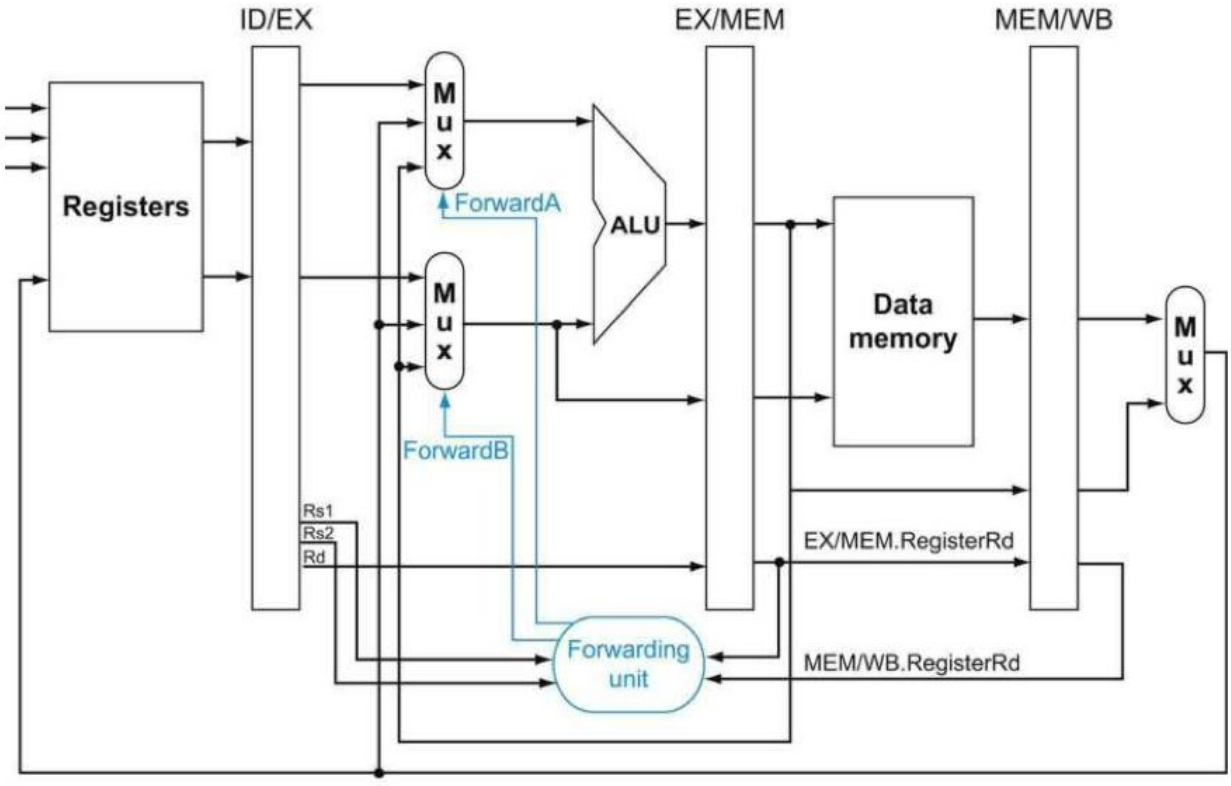



Figure 2: 5-stage pipeline with forwarding

7) Consider the 5-stage pipeline design **with forwarding** shown in Figure 2. Insert the minimum number of required NOPs into the instruction stream **without re-ordering** instructions to avoid all hazards. Calculate the average CPI of this code, assuming a CPI of one for all instructions. **(4 Points)**

| # | Instruction |
|---|---|
| 1 | Load x1, 0(x31) |
| 2 | Load x2, 8(x31) |
| 3 | NOP |
| 4 | Addi x3, x1, x2 |
| 5 | Mul x5, x1, x3 |
| 6 | Sub x5, x5, x3 |
| 7 | Load x6, 16(x31) |
| 8 | NOP |
| 9 | Add x6, x5, x6 |
| 10 | Sub x7, x8, x9 |
| 11 | Div x7, x7, x8 |

CPI of this code: **15/9**

8) Consider the 5-stage pipeline design with forwarding. Determine the combinatorial logic required to control the forwarding multiplexers. Use two different expressions, one for forwarding path A and one for forwarding path B. **(2 Point)**

**ForwardA[1] = EX/MEM.RegWrite & (EX/MEM.RegisterRd != 0) &**
**(EX/MEM.RegisterRd == ID/EX.RegisterRs)**

**ForwardA[0] = MEM/WB.RegWrite & (MEM/WB.RegisterRd != 0) &**
**(MEM/WB.RegisterRd == ID/EX.RegisterRs)**

**ForwardB[1] = EX/MEM.RegWrite & (EX/MEM.RegisterRd != 0) &**
**(EX/MEM.RegisterRd == ID/EX.RegisterRt)**

**ForwardB[0] = MEM/WB.RegWrite & (MEM/WB.RegisterRd != 0) &**
**(MEM/WB.RegisterRd == ID/EX.RegisterRt)**