Canvas Group Name: Student 1: Student 2:

CMPE110 HA3: Hazard Detective

Solution

1) 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)	
Addi 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

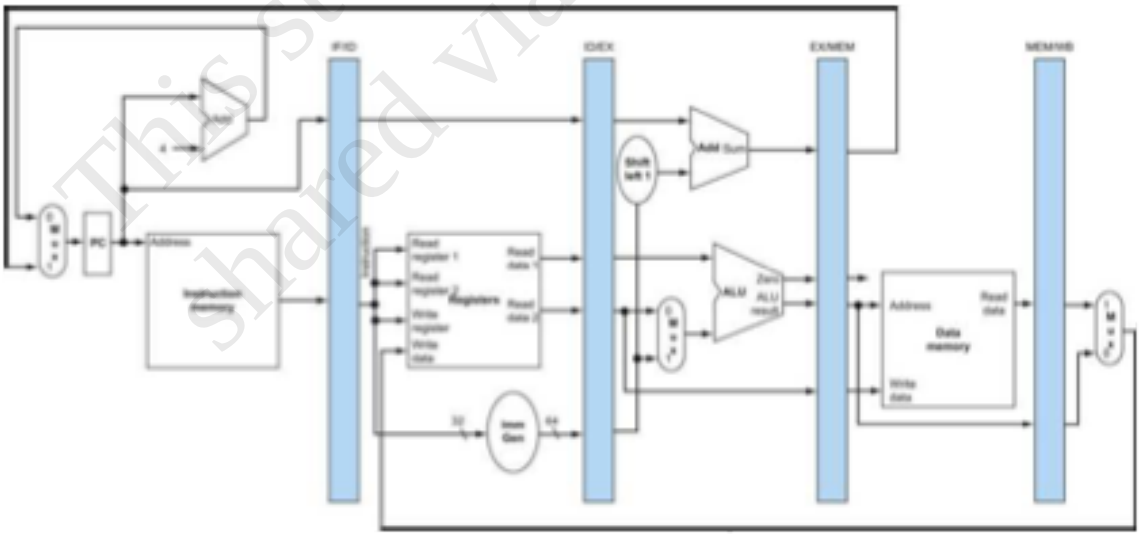


Figure 1: 5-stage pipeline without forwarding and hazard detection

2) Consider the 5-stage pipeline design we discussed in the lecture, shown in Figure 1, **without forwarding paths** and without hazard detection logic. In this case the compiler is responsible for avoiding RAW hazards. 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	NOP
5	Addi 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 cycles / 9 instructions_____

3) Consider the 5-stage pipeline design we discussed in the lecture, shown in Figure 1, without forwarding paths and without hazard detection logic. In this case the compiler is

responsible for avoiding RAW hazards. Optimize execution time by re-ordering instructions and then insert the minimum number of required NOPs into the instruction stream. Be careful to not change the output of the code when applying transformations. Calculate the average CPI of this code, assuming a CPI of one for all instrs. **(4 Points)**

#	Instruction
1	Load x1, 0(x31)
2	Load x2, 8(x31)
3	Load x6, 16(x31)
4	Sub x7, x8, x9
5	Addi 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 cycles / 9 instructions_____

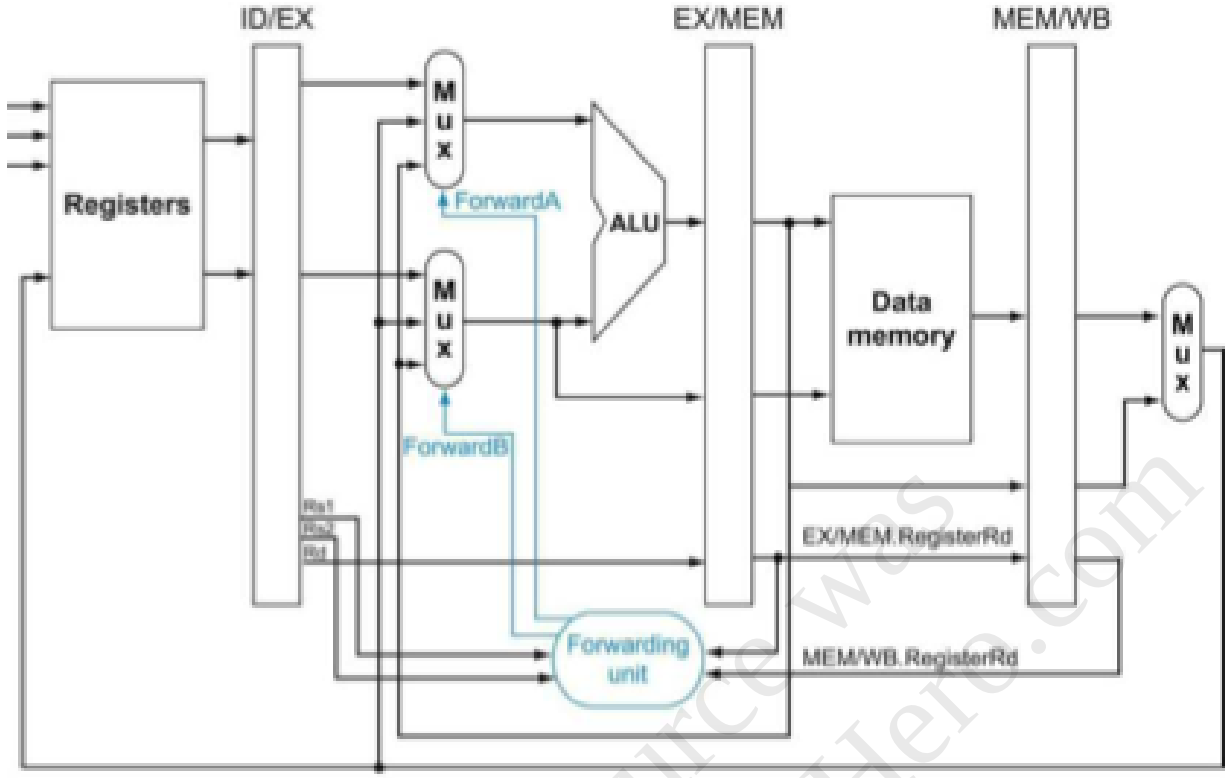


Figure 2: 5-stage pipeline with forwarding and without hazard detection

4) Consider the 5-stage pipeline design with forwarding, but without hazard detection logic shown in Figure 2. In this case the compiler is responsible for avoiding RAW hazards. 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. **(3 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 cycles / 9 instructions_____

5) Consider the 5-stage pipeline design with forwarding, but without hazard detection logic shown in Figure 2. In this case the compiler is responsible for avoiding RAW hazards. Optimize execution time by re-ordering instructions and then insert the minimum number of required NOPs into the instruction stream. Be careful to not change the output of the code when applying transformations. Calculate the average CPI of this code, assuming a CPI of one for all instructions. **(3 Points)**

#	Instruction
1	Load x1, 0(x31)
2	Load x2, 8(x31)
3	Load x6, 16(x31)
4	Addi x3, x1, x2

5	Mul x5, x1, x3
6	Sub x5, x5, x3
7	Add x6, x5, x6
8	Sub x7, x8, x9
9	Div x7, x7, x8

CPI of this code: ___13 cycles / 9 instructions_____

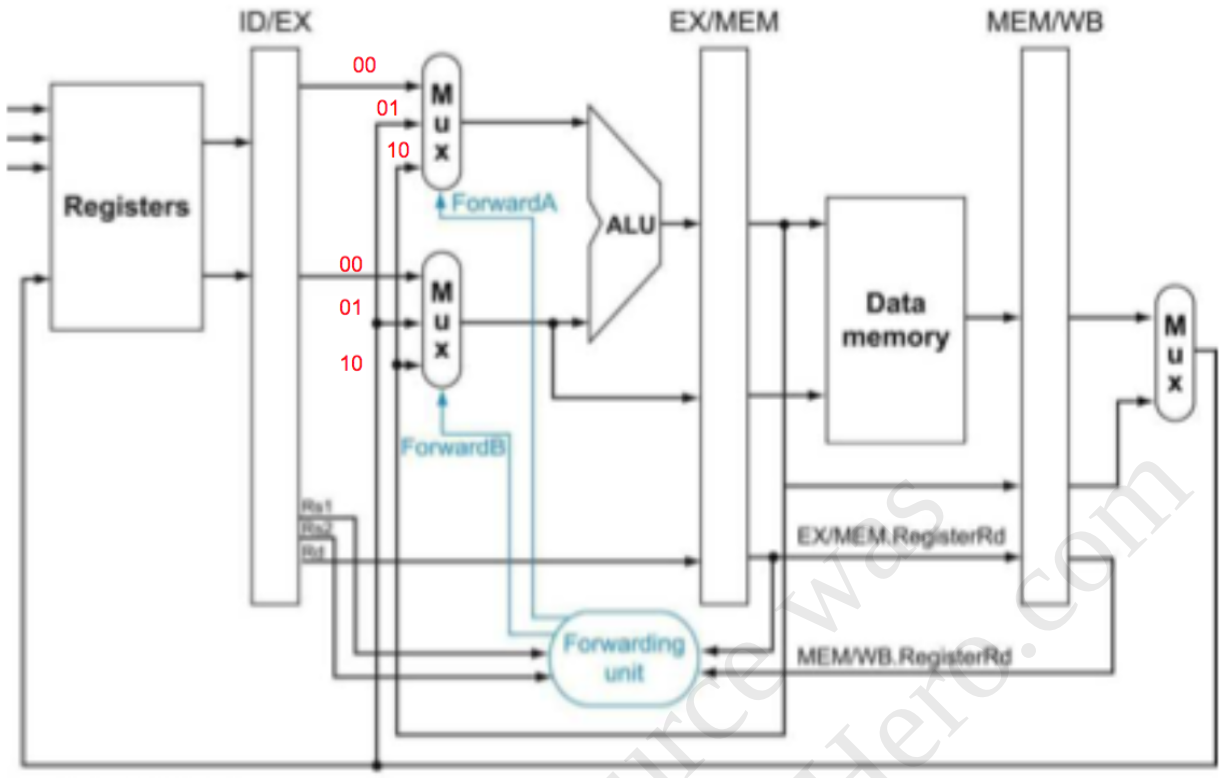
6) Consider the 5-stage pipeline design with forwarding, but without hazard detection logic shown in Figure 2. Determine the combinatorial logic required to control the forwarding multiplexers. Use two different expressions, one for forwarding path A and one for forwarding path B. **(2 Point)**

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)};



Mux control	source
ForwardA = 00	ID/EX
ForwardA = 01	EX/MEM
ForwardA = 10	MEM/WB
ForwardB = 00	ID/EX
ForwardB = 01	EX/MEM
ForwardB = 10	MEM/WB

7) You want to minimize the hardware required to implement the forwarding control logic. Which part of the logic terms to produce ForwardA and ForwardB can you share? **(1 Point)**

(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)

(MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)