EGC442 **HW#5** **Dr. Izadi**

First Name: _______________ Last Name: ___________________________

Question 1 (25 Points)

For the code below,

```
lw      $t0, 0($s1)
add     $t1, $s1, $a2
sub     $t0, $t0, $s2
sw      $t1, 0($s1)
addi    $s1, $s1,–4
```

a. On the diagram, mark and identify all the data dependencies in the code given below and identify which dependencies will cause data hazards without forwarding hardware.
b. Assuming there is no special hardware that is added for forwarding, add "nop" instructions to the code to avoid the data hazards.
c. Assume that the hardware supports forwarding and stalling. Show from which pipe the data is taken from and where it is forwarded. How many cycles will it take to execute this code (no need for nops)? Indicate what each stage will do during the 6th clock cycle.
d. How many cycles will it take to execute this code in parts b. and c.?

Question 2 (25 Points)

For the code below,

```
lw      $1, 0($1)
lw      $1, 0($1)
add     $1, $1, $2
lw      $1, 0($1)
sw      $2, 12($1)
```

a. On the diagram, mark and identify all the data dependencies in the code given below and identify which dependencies will cause data hazards without forwarding hardware.
b. Assuming there is no special hardware that is added for forwarding, add "nop" instructions to the code to avoid the data hazards.
c. Assume that the hardware supports forwarding and stalling. Show from which pipe the data is taken from and where it is forwarded. How many cycles will it take to execute this code (no need for nops)? Indicate what each stage will do during the 6th clock cycle.
d. How many cycles will it take to execute this code in parts b. and c.?

Question 3 (25 Points)
In this exercise, we examine how data dependences affect execution in the basic 5-stage pipeline described in COD Section 4.5 (An overview of pipelining). Problems in this exercise refer to the following sequence of instructions:

or r1, r2, r3

or r2, r1, r4

or r1, r1, r2

Also, assume the following cycle times for each of the options related to forwarding:

Without Forwarding	With Full Forwarding	With ALU-ALU Forwarding Only
250ps	300ps	290ps

a. Indicate dependences and their type.

b. Assume there is no forwarding in this pipelined processor. Indicate hazards and add nop instructions to eliminate them.

c. Assume there is full forwarding. Indicate hazards and add NOP instructions to eliminate them.

d. What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

e. Add nop instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).

f. What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

Note: With full forwarding, an ALU instruction can forward a value to EX stage of the next instruction without a hazard, With ALU-ALU-only forwarding, an ALU instruction can forward to the next instruction

Question 4 (25 Points)

4.13 This exercise is intended to help you understand the relationship between forwarding, hazard detection, and ISA design. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a 5-stage pipelined datapath:

add r5, r2, r1

lw r3, 4(r5)

lw r2, 0(r2)

or r3, r5, r3

sw r3, 0(r5)

a. If there is no forwarding or hazard detection, insert nops to ensure correct execution.

b. Repeat 4.13.1 but now use nops only when a hazard cannot be avoided by changing or rearranging these instructions. You can assume register R7 can be used to hold temporary values in your modified code.

c. If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when this code executes?

d. If there is forwarding, for the first five cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in COD Figure 4.60 (Pipelined control overview, showing the two multiplexor for forwarding …).

e. If there is no forwarding, what new inputs and output signals do we need for the hazard detection unit in COD Figure 4.60 (Pipelined control overview, showing the two multiplexor for forwarding …)? Using this instruction sequence as an example, explain why each signal is needed.

f. For the new hazard detection unit from 4.13.5, specify which output signals it asserts in each of the first five cycles during the execution of this code.

Due: 4/14/2023

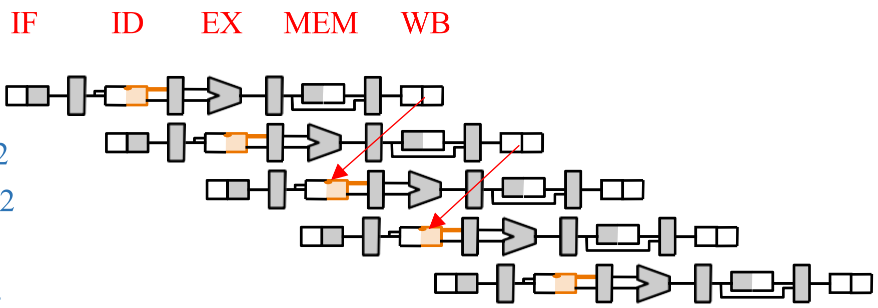
EGC442 **HW#5** Dr. Izadi

Key

Question 1 (25 Points)
a On the diagram, mark and identify all the data dependencies in the code given below and identify which dependencies will cause data hazards without forwarding hardware.

```
lw      $t0, 0($s1)
add     $t1, $t2, $a2
sub     $t0, $t0, $s2
sw      $t1, 0($s1)
addi    $s1, $s1,-4
```

```
lw      $t0, 0($s1)
add     $t1, $t2, $a2
sub     $t0, $t0, $s2
sw      $t1, 0($s1)
addi    $s1, $s1,–4
```



All dependencies are marked. Every dependency will cause data hazard.

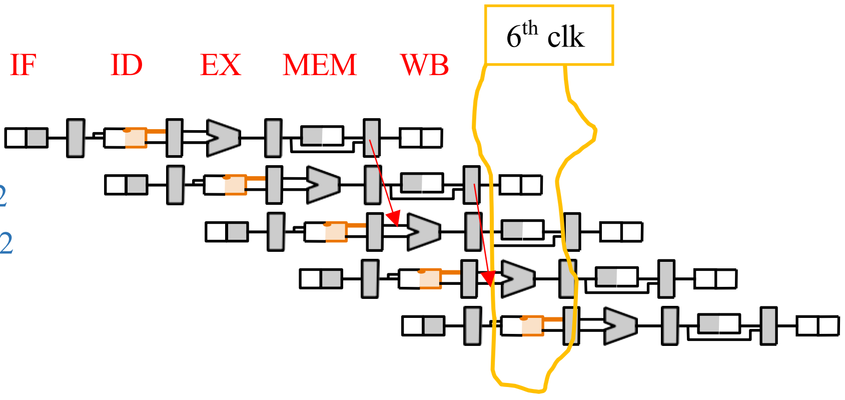
a. Assuming there is no special hardware that is added for forwarding, add “nop” instructions to the code to avoid the data hazards.

Every dependency can be resolved by delaying two clock cycles, assuming we have the hardware to write on the first half of the clock and read on the second half of it. By placing the nop after the add, both hazards are taken care of.

```
lw      $t0, 0($s1)
add     $t1, $t2, $a2
nop
sub     $t0, $t0, $s2
sw      $t1, 0($s1)
addi    $s1, $s1,-4
```

b. Assume that the hardware supports forwarding and stalling. Show from which pipe the data is taken from and where it is forwarded. How many cycles will it take to execute this code (no need for nops)? Indicate what each stage will do during the 6th clock cycle.

```
lw      $t0, 0($s1)
add     $t1, $t2, $a2
sub     $t0, $t0, $s2
sw      $t1, 0($s1)
addi    $s1, $s1,–4
```



It takes 9 cycles to execute this code.

During the 6th cycle, contents of register $t1 is written with the sum of $t2 and $a2, Mem is not doing anything useful. ALU is adding $s1 to 0, register $s1 is read. Side note: instruction following addi is fetched.

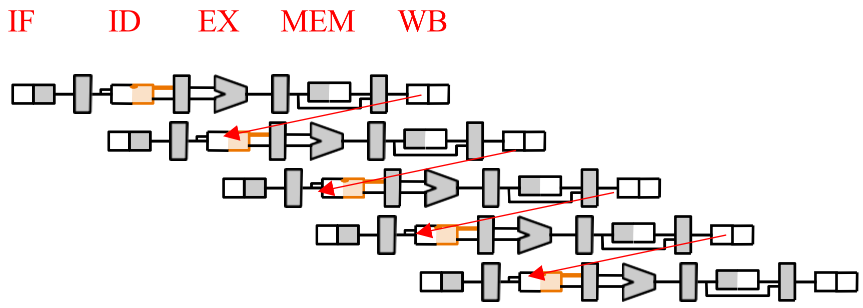
c. How many cycles will it take to execute this code in parts b. and c?

For part b, we would end up with 10 cycles. As indicated above, we use 9 cycles in part c.

Question 2 (25 Points)
For the code below,

```
lw      $1, 0($1)
lw      $1, 0($1)
add     $1, $1, $2
lw      $1, 0($1)
sw      $2, 12($1)
```

```
lw      $1, 0($1)
lw      $1, 0($1)
add     $1, $1, $2
lw      $1, 0($1)
sw      $2, 12($1)
```



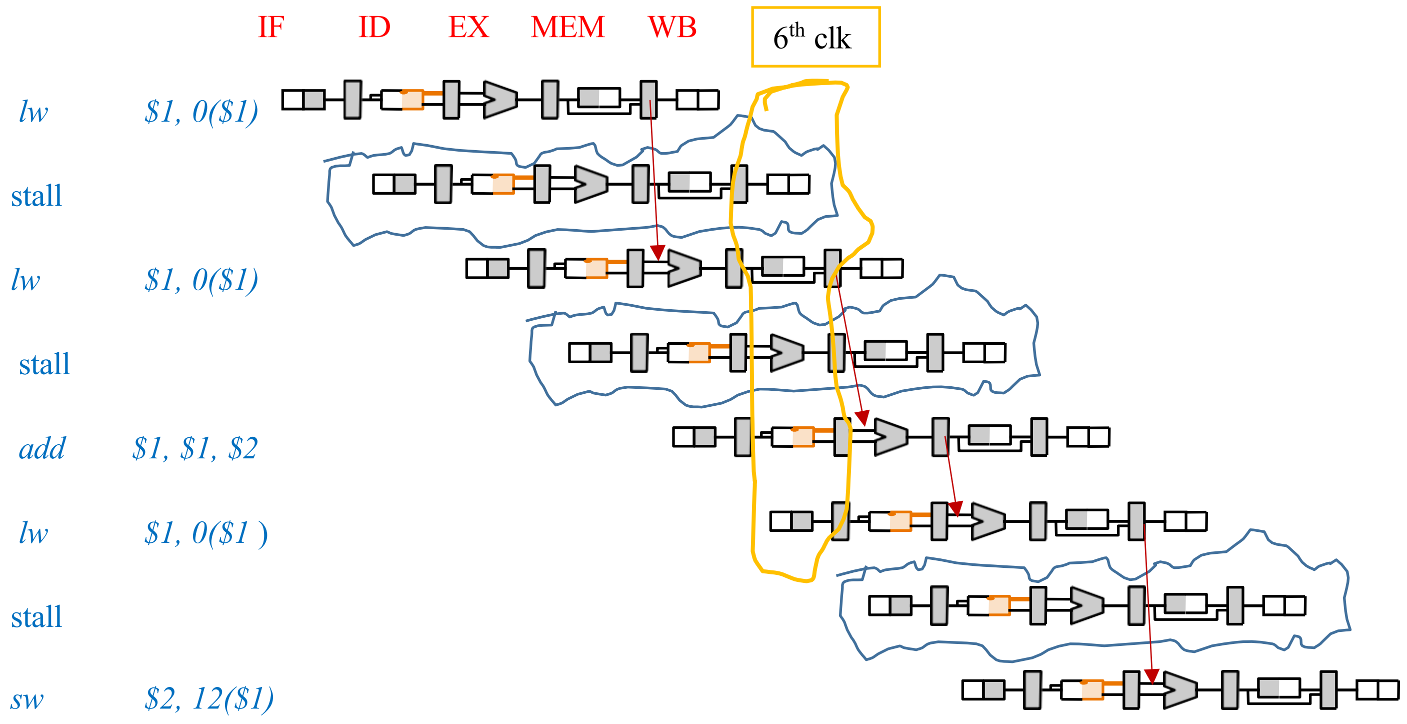
All dependencies are marked. Every dependency will cause data hazard.

b. Assuming there is no special hardware that is added for forwarding, add "nop" instructions to the code to avoid the data hazards.

Every dependency can be resolved by delaying two clock cycles, assuming we have the hardware to write on the first half of the clock and read on the second half of it.

```
lw      $1, 0($1)
nop                         (2 or 3 nop's would be OK)
nop
lw      $1, 0($1)
nop                         (2 or 3 nop's would be OK)
nop
add     $1, $1, $2
nop                         (2 or 3 nop's would be OK)
nop
lw      $1, 0($1)
nop                         (2 or 3 nop's would be OK)
nop
sw      $2, 12($1)
```

c. Assume that the hardware supports forwarding and stalling. Show from which pipe the data is taken from and where it is forwarded. How many cycles will it take to execute this code (no need for nops)? Indicate what each stage will do during the 6th clock cycle.



It takes 12 cycles to execute this code.

During the 6th cycle, contents of Mem[$1 +0] is re-written back to $1, Data memory Mem[$1 +0] read , $1 and constant 0 is added again, registers $1 and $2 are read, and lw instruction is fetched.

d. How many cycles will it take to execute this code in parts b. and c?

For part b, we would end up with 17 cycles (21 cycles if we use 3 nop's). As indicated above, we use 12 cycles in part c.

Question 3 (25 Points)

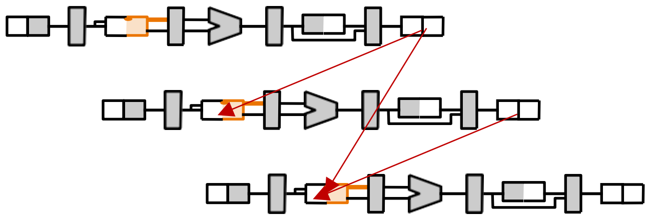
Note: With full forwarding, an ALU instruction can forward a value to EX stage of the next instruction without a hazard, With ALU-ALU-only forwarding, an ALU instruction can forward to the next instruction

a.

I1: or r1, r2, r3

I2: or r2, r1, r4

I3: or r1, r1, r2



I2 is dependent on r1 of I1, I3 is dependent on r1 of I1 and r2 of I2. All are data hazard.

b.

```
or      r1, r2, r3
nop
nop
or      r2, r1, r4
nop
nop
or      r1, r1, r2
```

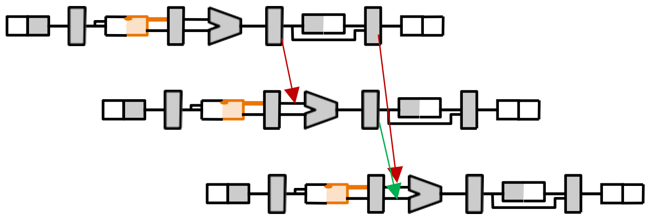
c. (With full forwarding, an ALU instruction can forward a value to EX stage of the next instruction without a hazard)

With full forwarding, an ALU instruction can forward a value to EX stage of the next instruction without a hazard. However, a load cannot forward to the EX stage of the next instruction (by can to the instruction after that). The code that eliminates these hazards by inserting NOP instructions is:

I1: or r1, r2, r3

I2: or r2, r1, r4

I3: or r1, r1, r2



d.

The total execution time is the clock cycle time times the number of cycles. Without any stalls, a three-instruction sequence executes in 7 cycles (5 to complete the first instruction, then one per instruction). The execution without forwarding must add a stall for every NOP we had in b. So, we get (7+4) * 250 ps = 2750 ps.
The execution with forwarding only uses 7 cycles. However, each cycle take 300 ps. Therefore we get 7 * 300 = 2100.
Hence, the speedup due to forwarding is 2750/2100 = 1.31

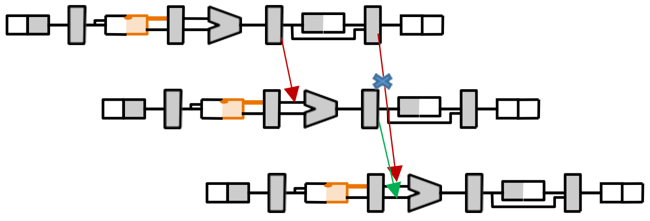
e. (With ALU-ALU-only forwarding, an ALU instruction can forward to the next instruction),

With ALU-ALU-only forwarding, an ALU instruction can forward to the next instruction, but not to the second-next instruction (because that would be forwarding from MEM to EX).

I1: or r1, r2, r3

I2: or r2, r1, r4

I3: or r1, r1, r2

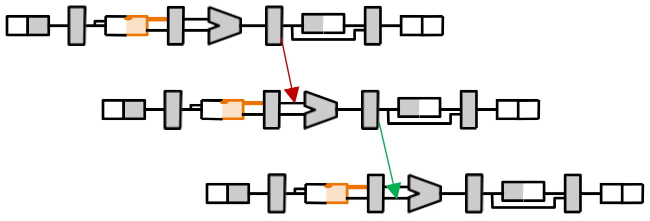


Therefore, r1 from the first instruction can't be forwarded to the 3rd one.

I1: or r1, r2, r3

I2: or r2, r1, r4

I3: or r1, r1, r2



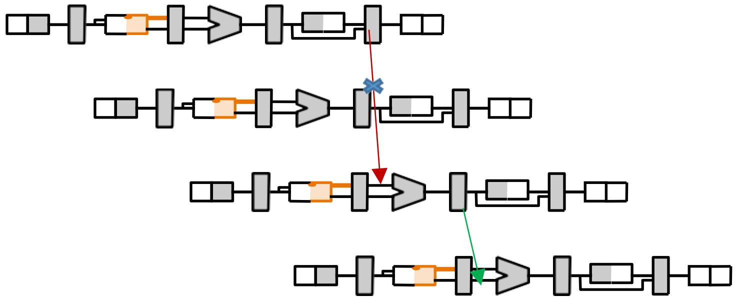
The problem is that adding any nop would require the forwarding to be from MEM to EX i.e

I1: or r1, r2, r3

nop

I2: or r2, r1, r4

I3: or r1, r1, r2



Therefore, we would be forced into regular nop's

```
or      r1, r2, r3
nop
nop
or      r2, r1, r4
nop
nop
or      r1, r1, r2
```

f.

The execution without forwarding adds a stall for every NOP. So, we get (7+4) * 250 ps = 2750 ps.

The execution with ALU - ALU forwarding uses 11 cycles at 290 ps. So, we get (7+4) * 290 ps = 3190 ps.

So speedup = 2750/3190 = .86

Question 4 (25 Points)

a.

```
ADD R5,R2,R1
NOP
NOP
LW R3,4(R5)
LW R2,0(R2)
NOP
OR R3,R5,R3
NOP
NOP
SW R3,0(R5)
```

b.

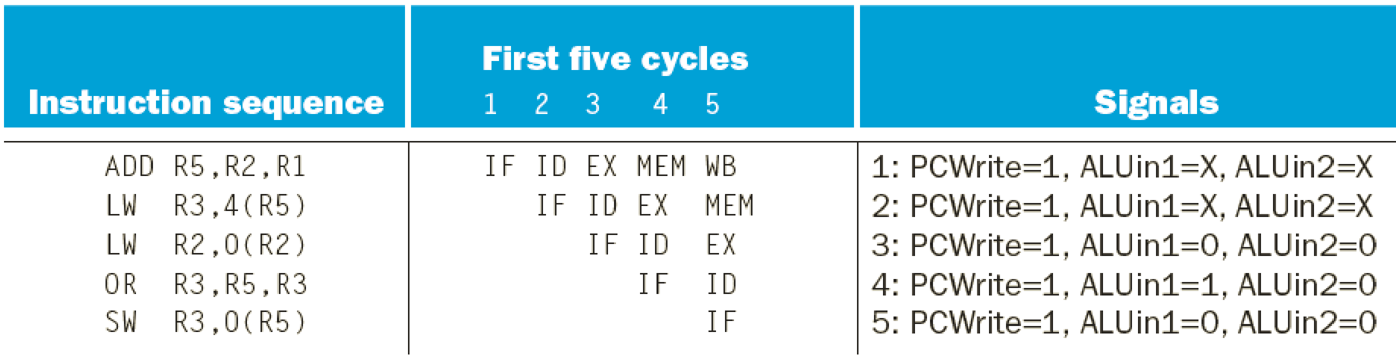
We can move up an instruction by swapping its place with another instruction that has no dependences with it, so we can try to fill some NOP slots with such instructions. We can also use R7 to eliminate WAW or WAR dependences so we can have more instructions to move up.

I1: ADD R5,R2,R1	
I3: LW R2,0(R2)	Moved up to fill NOP slot
NOP	
I2: LW R3,4(R5)	
NOP	Had to add another NOP here,
NOP	so there is no performance gain
I4: OR R3,R5,R3	
NOP	
NOP	
I5: SW R3,0(R5)	

c.With forwarding, the hazard detection unit is still needed because it must insert a one-cycle stall whenever the load supplies a value to the instruction that immediately follows that load. Without the hazard detection unit, the instruction that depends on the immediately preceding load gets the stale value the register had before the load instruction.

Code executes correctly (for both loads, there is no RAW dependence between the load and the next instruction).

d. The outputs of the hazard detection unit are PCWrite, IF/IDWrite, and ID/EXZero (which controls the Mux aft er the output of the Control unit). Note that IF/IDWrite is always equal to PCWrite, and ED/ExZero is always the opposite of PCWrite. As a result, we will only show the value of PCWrite for each cycle. Th e outputs of the forwarding unit is ALUin1 and ALUin2, which control Muxes that select the fi rst and second input of the ALU. Th e three possible values for ALUin1 or ALUin2 are 0 (no forwarding), 1 (forward ALU output from previous instruction), or 2 (forward data value for second-previous instruction). We have:



Instruction sequence	First five cycles					Signals
	1	2	3	4	5	
ADD R5,R2,R1	IF	ID	EX	MEM	WB	1: PCWrite=1, ALUin1=X, ALUin2=X
LW R3,4(R5)		IF	ID	EX	MEM	2: PCWrite=1, ALUin1=X, ALUin2=X
LW R2,0(R2)			IF	ID	EX	3: PCWrite=1, ALUin1=0, ALUin2=0
OR R3,R5,R3				IF	ID	4: PCWrite=1, ALUin1=1, ALUin2=0
SW R3,0(R5)					IF	5: PCWrite=1, ALUin1=0, ALUin2=0

i. The instruction that is currently in the ID stage needs to be stalled if it depends on a value produced by the instruction in the EX or the instruction in the MEM stage. So we need to check the destination register of these two instructions. For the instruction in the EX stage, we need to check Rd for R-type instructions and Rd for loads. For the instruction in the MEM stage, the destination register is already selected (by the Mux in the EX stage) so we need to check that register number (this is the bottommost output of the EX/MEM pipeline register). The additional inputs to the hazard detection unit are register Rd from the ID/EX pipeline register and the output number of the output register from the EX/MEM pipeline register. The Rt field from the ID/EX register is already an input of the hazard detection unit in Figure 4.60. No additional outputs are needed. We can stall the pipeline using the three output signals that we already have.

ii. As explained for part e, we only need to specify the value of the PCWrite signal, because IF/IDWrite is equal to PCWrite and the ID/EXzero signal is its opposite. We have:

Instruction sequence	First five cycles					Signals
	1	2	3	4	5	
`ADD R5,R2,R1`	IF	ID	EX	MEM	WB	1: PCWrite=1
`LW R3,4(R5)`		IF	ID	***	***	2: PCWrite=1
`LW R2,0(R2)`			IF	***	***	3: PCWrite=1
`OR R3,R5,R3`					***	4: PCWrite=0
`SW R3,0(R5)`						5: PCWrite=0