Gaurav Ghop gg2961

NYU TANDON SCHOOL OF ENGINEERING

COMPUTER SYSTEMS ARCHITECTURE

HW-6 Solutions

NYU Tandon School of Engineering

Fall 2022, ECE 6913

HW 6 Solutions

---

1. Consider a version of the pipeline from *Section 4.5 in RISC-V text* that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting `NOP` instructions where necessary). Suppose that (after optimization) a typical `n`- instruction program requires an additional **0.**`4*n NOP` instructions to correctly handle data hazards.

1.1 Suppose that the cycle time of this pipeline without forwarding is 250 ps. Suppose also that adding forwarding hardware will reduce the number of `NOPs` from `.4*n to .05*n`, but increase the cycle time to 300 ps. What is the speedup of this new pipeline compared to the one without forwarding?

Making the implicit assumption that the average CPI for the n-instruction Program is 1 and assuming no data hazards are encountered, in a Pipeline executing at cycle time of 250 ps takes

T0 = n x 250 ps

Assuming 0.4 NOPS per instruction,

T1 = 1.4n x 250ps

Adding forwarding hardware reduces the number of NOPS to 0.05n but also increases the cycle time to 300ps yielding total execution time of:

T2 = 1.05 x n x 300 ps

Therefore, speedup = T1/T2 = [1.4 x 250] /[1.05 x 300] = 1.11

1.2 Different programs will require different amounts of `NOPs`. How many `NOPs` (as a percentage of code instructions) can remain in the typical program before that program runs slower on the pipeline with forwarding?

Without Forwarding, typical program execution time,

$$T0 = 1.4n \times 250ps$$

With Forwarding, assuming γ = number of stalls per instruction due to forwarding, execution time,

$$TF = (1 + \gamma) \times n \times 300ps$$

maximum number of stalls per instruction given by

T0 ≥ TF

1.4n × 250ps ≥ (1 + γ) × n × 300ps

Solving, γ ≤ 0.167

1.3 Repeat 1.2; however, this time let x represent the number of NOP instructions relative to n. (In

1.2, x was equal to 0.4) Your answer will be with respect to x.

Replacing 0.4 for x in Problem 1.2, we get

(1+x)n × 250ps ≥ (1 + γ) × n × 300ps solving

for g,

[A] γ < (250x – 50)/300

1.4 Can a program with only .075*n NOPs possibly run faster on the pipeline with forwarding?

Explain why or why not.

No. In the best case, where forwarding results in no NOPS at all, the execution time will be 300n which is more than 250 x 1.075n. Forwarding is more likely to help improve performance of programs with few data hazards if the overheads in cycle time using forwarding hardware are minimized.

1.5 At minimum, how many NOPs (as a percentage of code instructions) must a program have before it can possibly run faster on the pipeline with forwarding?

The larger the number of NOPS per instruction, **x** without forwarding, the easier it is for forwarding to work and lower the number of NOPS per instruction. As the number of NOPS per instruction without forwarding, **x** decreases, and the benefits from forwarding are limited to overcoming the overheads in cycle time from using forwarding hardware. When overheads in cycle time balance the benefits of forwarding, □ = 0. From this point onward, smaller x does not permit improvements in performance. Thus **x** is at a minimum corresponding to the case of □=0 in [A] above.

This minimum value of x can be solved in [A] for □ =0 as x =

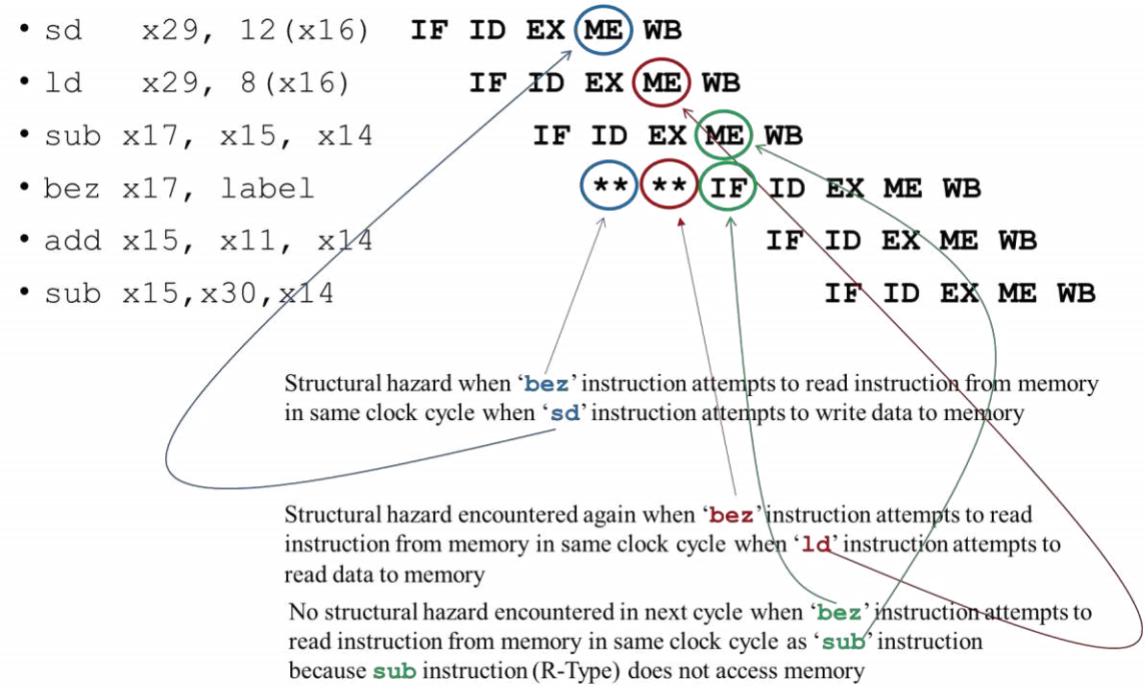
0.2

2. Consider the fragment of RISC-V assembly below:

```
sd x29, 12(x16)
ld x29, 8(x16)
sub x17, x15, x14
beqz x17, label
add x15, x11, x14
sub x15, x30, x14
```

Suppose we modify the pipeline so that it has only one memory (*that handles both instructions and data*). In this case, there will be a structural hazard every time a program needs to fetch an instruction during the same cycle in which another instruction accesses data.

2.1 Draw a pipeline diagram to show where the code above will stall.



2.2 In general, is it possible to reduce the number of stalls/NOPs resulting from this structural hazard by reordering code?

The conflict for access to Memory will always be present when a load or store instruction is present - between the first RISC pipeline stage of the instruction 3 cycles later and the Fourth pipeline stage of the load or store instruction

It is not possible to lower NOPs due to structural hazards from load or store instructions by reordering the code 3 cycles downstream from the load or store instruction since every such instruction 3 cycles after a load/store instruction must access memory for IF

2.3 Must this structural hazard be handled in hardware? We have seen that data hazards can be eliminated by adding `NOPs` to the code. Can you do the same with this structural hazard? If so, explain how. If not, explain why not.

Yes, this structural hazard can be eliminated by inserting a NOP 3 instructions down stream for each load/store instruction. The penalty in performance can be very high since load/store instructions typically account for 30%-40% of the instructions in a Program. Adding a NOP for each load/store instruction can increase the execution time by 30%-40%.

The only way to resolve this structural hazard effectively in hardware without the heavy overheads from introducing NOPS is by adding a separate memory array for Data and for Instructions.

2.4 Approximately how many stalls would you expect this structural hazard to generate in a typical program? (*Use the instruction mix shown below*)

R-type/I-type (non-ld)	ld	sd	beq
52%	25%	11%	12%

36% of the instructions will stall corresponding to all of the load/store instructions in a typical program represented in table above

3. If we change load/store instructions to use a register (without an offset) as the address, these instructions no longer need to use the ALU. (See Problem 4 in HW 4) As a result, the MEM and EX stages can be overlapped and the pipeline has only four stages.

3.1 How will the reduction in pipeline depth affect the cycle time?

Cycle time is dependent on the latency of the slowest of the 5 RISC pipeline stages not on the number of stages. Cycle time will not change by reduction of number of pipeline stages or the pipeline depth of the processor

3.2 How might this change improve the performance of the pipeline?

By overlapping the MEM with the EX pipeline stages, fewer pipeline stages enables the latency of an instruction to improve. But this improvement is only by 1 cycle total. However, load-use-data hazards that require a NOP cycle inserted between a load instruction and an instruction that uses this data read from memory, can eliminate the need of this NOP cycle because the MEM stage executes in Clock Cycle 3 of the load instruction after which the pipeline registers can forward data read from in MEM stage to the ALU input multiplexor - potentially lowering the Execution Time penalty in NOPS per instruction for load-usedata hazards

3.3 How might this change degrade the performance of the pipeline?

Removing the offset from load/store instructions can require addi to be used/paired with load/store instructions thus increasing the number of instructions in the program.

4. Which of the two pipeline diagrams below better describes the operation of the pipeline's hazard detection unit? Why?

Choice 1:

```
 ld x11, 0(x12): IF ID EX ME WB
add x13, x11, x14: IF ID EX..ME WB
or x15, x16, x17: IF ID..EX ME WB
```

Choice 2:

```
 ld x11, 0(x12): IF ID EX ME WB
add x13, x11, x14: IF ID..EX ME WB
or x15, x16, x17: IF..ID EX ME WB
```

```
                                CC1CC2CC3CC4CC5CC6CC7CC8
ld x11, 0(x12):          IF ID EX ME WB
add x13, x11, x14:          IF ID EX .. ME WB
or x15, x16, x17:              IF ID .. EX ME WB

ld x11, 0(x12):          IF ID EX ME WB
add x13, x11, x14:          IF ID .. EX ME WB
or x15, x16, x17:              IF .. ID EX ME WB
```

In the first sequence, **content of register x11 becomes available only at the end of CC4 but is being used at the start of CC4 in the execute stage of the add instruction – this is a 'load-use-data-hazard'. Insertion of a stall in CC5 is too late**

In the second sequence, **content of register x11 which becomes available at the end of CC4 is used at the start of CC5 (and not CC4 as above) since the EX stage has been delayed by insertion of a stall in CC4**

5. Consider the following loop.

```
LOOP: ld   x10, 0(x13)
ld   x11, 8(x13)
add  x12, x10, x11
subi x13, x13, 16
 bnez x12, LOOP
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, that the pipeline has full forwarding support, and that branches are resolved in the `EX` (as opposed to the `ID`) stage.

5.1 Show a pipeline execution diagram for the first two iterations of this loop.

`Please see below in response to 5.2`

5.2 Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the `subi` is in the `IF` stage. End with the cycle during which the `bnez` is in the `IF` stage.)

[1] Since perfect branch prediction is used, we do not lose any cycles due to branch hazards – that is, the branch is always predicted and taken correctly

[2] Availability of full forwarding support is assumed, so we can assume data hazards that can be resolved with forwarding do not stall the pipeline

[3] Load-use-hazards are identified below in red boxes - resolved by stalling the pipeline

[4a] pipeline stages unused by any instruction are identified in **blue**

[4b] any clock cycle that does not have all of the pipeline stages utilized is identified with **'N'**

```
1. loop ld x10, 0(x13)  IF ID EX ME | WB
2. ld x11, 8(x13)          IF ID EX | ME WB
3. add x12, x10, x11          IF ID | .. EX ME  WB
4. addi x13, x13, -16            IF | .. ID EX  ME  WB
5. bnez x12, loop                   | .. IF ID  EX  ME  WB
6. ld x10, 0(x13)                         IF  ID  EX  ME  WB
7. ld x11, 8(x13)                             IF  ID  EX  ME WB
8. add x12, x10, x11                              IF  ID  .. EX | ME  WB
9. addi x13, x13, -16                                 IF  .. ID | EX  ME  WB
10.bnez x12, loop                                         IF | ID  EX  ME  WB
All pipeline stages utilized?      |  N  N  N   N   N   N   N  N  |
```

6. This exercise is intended to help you understand the cost/complexity/performance trade-offs of forwarding in a pipelined processor. Problems in this exercise refer to pipelined datapaths from *Figure 4.53 in RISC-V text (reproduced below)*. These problems assume that, of all the instructions executed in a processor, the following fraction of these instructions has a particular type of RAW data dependence.

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

The type of RAW data dependence is identified by the stage that produces the result (EX or MEM) and the next instruction that consumes the result (1st instruction that follows the one that produces the result, 2nd instruction that follows, or both). We assume that the register write is done in the

first half of the clock cycle and that register reads are done in the second half of the cycle, so "EX to 3rd" and "MEM to 3rd" dependences are not counted because they cannot result in data hazards.

We also assume that branches are resolved in the EX stage (as opposed to the ID stage), and that the CPI of the processor is 1 if there are no data hazards.

EX to 1st Only	MEM to 1st Only	EX to 2nd Only	MEM to 2nd Only	EX to 1st and EX to 2nd
5%	20%	5%	10%	10%

Assume the following latencies for individual pipeline stages. For the EX stage, latencies are given separately for a processor without forwarding and for a processor with different kinds of forwarding.

IF	ID	EX (no FW)	EX (full FW)	EX (FW from EX/MEM only)	EX (FW from MEM/WB only)	MEM	WB
120 ps	100 ps	110 ps	130 ps	120 ps	120 ps	120 ps	100 ps

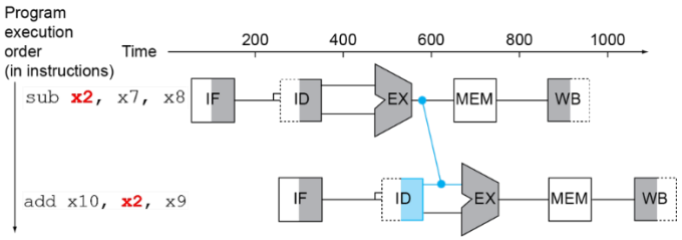
6.1 For each RAW dependency listed above, give a sequence of at least three assembly statements that exhibits that dependency.

(1) EX to 1st only:

sub **x2**, x7, x8

add x10, **x2**, x9

add x27, x28, x29

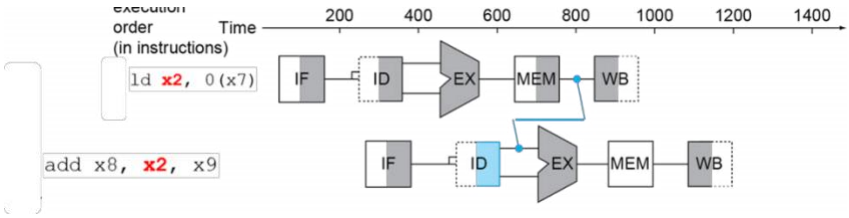


(2) MEM to 1st only:

ld **x2**, 0(x7)

add x8, **x2**, x9

sub x15, x22, x23

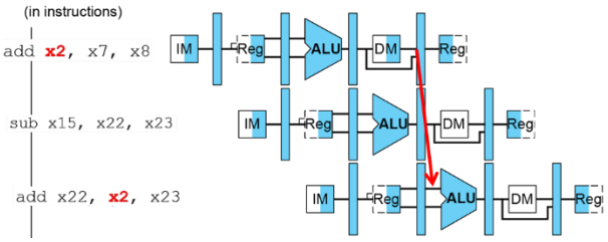


(3) EX to 2nd only:

add **x2**, x7, x8

sub x15, x22, x23

add x22, **x2**, x23

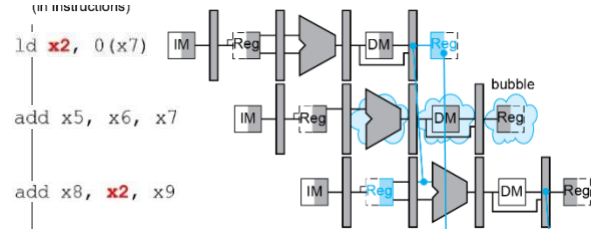


(4) MEM to 2nd only:

ld **x2**, 0(x7)

add x5, x6, x7

add x8, **x2**, x9

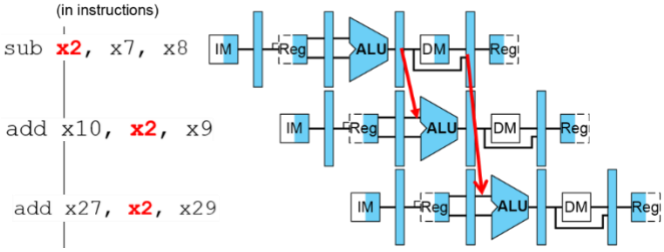


(5) EX to 1st and EX to 2nd :

sub **x2**, x7, x8

add x10, **x2**, x9

add x27, **x2**, x29



6.2 For each RAW dependency above, how many NOPs would need to be inserted to allow your code from ***6.1*** to run correctly on a pipeline with no forwarding or hazard detection? Show where the NOPs could be inserted.

EX to 1st only: sub **x2**, x7, x8

NOP

NOP

add x10, **x2**, x9

add x27, x28, x29

MEM to 1st only:

ld **x2**, 0(x7)

NOP

NOP

add x8, **x2**, x9

EX to 2nd only:

add **x2**, x7, x8

sub x15, x22, x23

NOP

add x22, **x2**, x23

MEM to 2nd only:

ld **x2**, 0(x7) add

x5, x6, x7

NOP

add x8, **x2**, x9

EX to 1st and EX to 2nd :

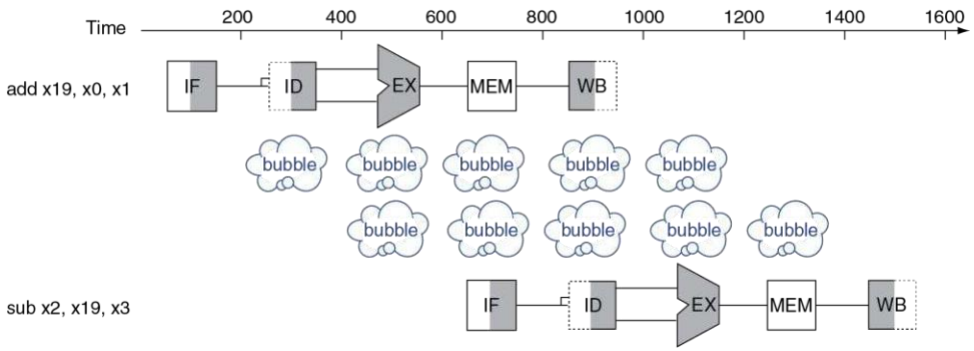
sub **x2**, x7, x8

NOP

NOP

add x10, **x2**, x9 add

x27, **x2**, x29



6.3 Analyzing each instruction independently will over-count the number of NOPs needed to run a program on a pipeline with no forwarding or hazard detection. Write a sequence of three assembly instructions so that, when you consider each instruction in the sequence independently, the sum of the stalls is larger than the number of stalls the sequence actually needs to avoid data hazards.

Please see response for 6.2

6.4 Assuming no other hazards, what is the CPI for the program described by the table above when run on a pipeline with no forwarding? What percent of cycles are stalls? (For simplicity, assume that all necessary cases are listed above and can be treated independently.)

EX to 1st Only	MEM to 1st Only	EX to 2nd Only	MEM to 2nd Only	EX to 1st and EX to 2nd
5%	20%	5%	10%	10%

Taking a weighted average of the number of NOPs for each from 6.2 gives

0.05*2 + 0.2*2 + 0.05*1 + 0.1*1 + 0.1*2 = 0.85 NOPs per instruction

A CPI of 1.85.

So, 0.85/1.85 cycles = 46%, are NOPs.

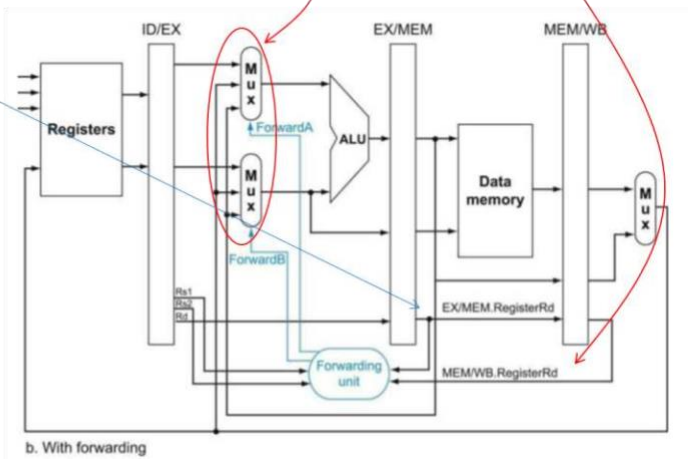
6.5 What is the CPI if we use full forwarding (forward all results that can be forwarded)? What percent of cycles are stalls?

The only RAW dependency that cannot be handled with forwarding is load-use-data hazards: from the MEM stage to the next instruction

20% of instructions will generate one NOP in these increasing CPI from 1 to 1.2.

So, 0.2 out of 1.2 cycles are NOPs.. that is, 17% are NOPs

6.6 Let us assume that we cannot afford to have **three-input** multiplexors that are needed for full forwarding. We have to decide if it is better to forward only from the **EX/MEM pipeline register** (next-cycle forwarding) or only from the MEM/WB pipeline register (two-cycle forwarding). What is the CPI for each option?



If we forward using the **EX/MEM register only**, we have the following number of stalls/NOPs

EX to 1st: **0** (eq 1) in slide 22 This hazard needs the **EX/MEM** register with which the forwarding unit eliminates the NOP, so NOPs = 0
MEM to 1st: **2** (eq 2) [must insert 2 NOPs] This hazard needs the MEM/WB register (load-use-data hazard) and cannot be resolved, so 2 NOPs must be inserted
EX to 2nd: **1** (eq 3) This hazard needs the MEM/WB register and cannot be resolved so a NOP must be inserted
MEM to 2nd: **1** (eq 4) This hazard needs the **MEM/WB** register and cannot be resolved, so a NOP is inserted
EX to 1st and 2nd: **1** (eq 5) EX to 1st can be resolved with the EX/MEM register but the EX to 2nd needs the MEM/WB register and cannot be resolved so a NOP is inserted to resolve the EX to 2nd hazard

With MEM/WB register unavailable,

Average NOPs of 0.05*0 + 0.2*2 + 0.05*1 + 0.10*1 + 0.10*1

= 0.65 stalls/instruction

So, CPI = 1.65

If we forward **using the MEM/WB register only**, we have the following number of stalls/NOPs

EX to 1st: **1**(eq 1) in slide 22 This hazard needs the **EX/MEM** register and can be resolved with the MEM/WB register, if the MEM/WB register forwards to the ALU after 1 NOP, so NOP =1

MEM to 1st: 1 (eq 2) This hazard needs the MEM/WB register (load-use-data hazard) but 1 NOP must be inserted
EX to 2nd: 0 (eq 3) This hazard needs the MEM/WB register and and is resolved with 0 NOPs
MEM to 2nd: 0 (eq 4) This hazard needs the MEM/WB register and and is resolved with 0 NOPs
EX to 1st and 2nd: 1 (eq 5) EX to 1st cannot be resolved and needs 1 NOP, but the EX to 2nd can be resolved with the MEM/WB register

With EX/MEM register unavailable,

Average NOPs of 0.05*1 + 0.2*1 + 0.1*1

= 0.35 stalls/instruction

So, CPI = 1.35

6.7 For the given hazard probabilities and pipeline stage latencies, what is the speedup achieved by each type of forwarding (EX/MEM, MEM/WB, for full) as compared to a pipeline that has no forwarding?

- We calculate CPI for NF, EX/MEM only, MEM/WB only and with Full Forwarding
- Period for any of the above is the longest latency stage [FF needs 130 ps w FF unit]

	No Forwarding	EX/MEM	MEM/WB	Full Forwarding
CPI	1.85	1.65	1.35	1.2
Period	120ps	120ps	120ps	130ps
Time	222ps	198ps	162ps	156ps
Speedup	ref	1.12	1.37	1.42

6.8 What would be the additional speedup (relative to the fastest processor from 6.7) be if we added "timetravel" forwarding that eliminates all data hazards? Assume that the yet-to-beinvented time-travel circuitry adds 100 ps to the latency of the full-forwarding EX stage.

CPI with full forwarding is 1.2

CPI for "time travel" forwarding is 1.0 [Max with 0 Hazards]

Clock period for full forwarding is 130

Clock period for zero hazards forwarding is 230

Speedup = Told/Tnew = (1.2*130)/ (1*230) = 0.68

Zero hazard forwarding actually slows the CPU because the overhead in cycle time from adding forwarding hardware (100 ps or 77% of the cycle time without forwarding hardware) is too large for forwarding to improve performance

6.9 The table of hazard types has separate entries for "EX to 1st " and "EX to 1st and EX to 2nd ". Why is there no entry for "MEM to 1st and MEM to 2nd "?

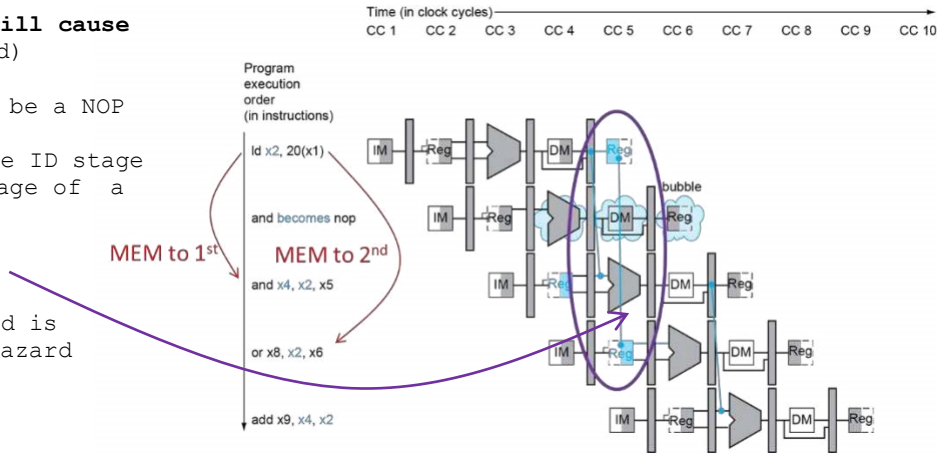
All MEM to 1st instructions **will cause a stall** (load-use-data hazard)

So instruction after ld must be a NOP

So, 2nd instruction would have ID stage in same Clock Cycle as WB stage of a ld instruction

This is not a hazard anymore

So, if a MEM to 1st RAW hazard is present, the MEM to 2nd RAW hazard cannot be present!



7. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
add x15, x12, x11
ld x13, 4(x15)
ld x12, 0(x2)
or x13, x15, x13
sd x13, 0(x15)
```

---

add x15, x12, x11

```
ld x13, 4(x15)          EX to 1st RAW Hazard
ld x12, 0(x2)
or x13, x15, x13        MEM to 2nd RAW Hazard
sd x13, 0(x15)          EX to 1st RAW Hazard
```

7.1 If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.

```
add x15, x12, x11
nop
nop                     EX to 1st RAW Hazard resolution with 2 NOPs
ld x13, 4(x15)
ld x12, 0(x2)
nop                     MEM to 2nd RAW Hazard resolution with 1 NOP
or x13, x15, x13
nop
nop                     EX to 1st RAW Hazard resolution with 2 NOPs
sd x13, 0(x15)
```

7.2 Now, change and/or rearrange the code to minimize the number of NOPs needed. You can assume register x17 can be used to hold temporary values in your modified code.

not possible to reduce the number of NOPs.

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

The code executes correctly.

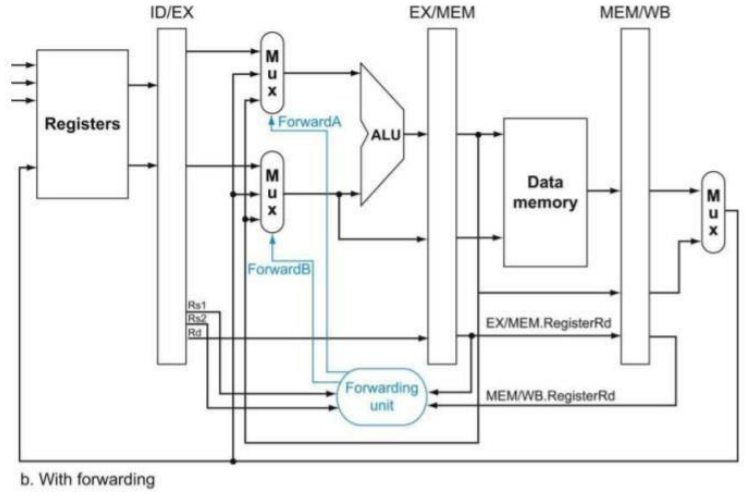
Hazard detection relevant only to insert a stall for load-use-data hazards

The given instruction sequence does not have **ld** followed by use of register written into

7.4 If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.53 of the RISC V text (reproduced below).

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

Clock Cycle	1	2	3	4	5	6	7	8	9
add	IF	ID	EX	MEM	WB				
ld		IF	ID	EX	MEM	WB			
ld			IF	ID	EX	MEM	WB		
or				IF	ID	EX	MEM	WB	
sd					IF	ID	EX	MEM	WB



b. With forwarding

(1) ForwardA = X; ForwardB = X (no instruction in EX stage yet)

(2) ForwardA = X; ForwardB = X (no instruction in EX stage yet)

(3) ForwardA = 0; ForwardB = 0 (no forwarding; values taken from registers)

(4) ForwardA = 2; ForwardB = 0 (base register taken from result of previous instruction)

(5) ForwardA = 0; ForwardB = 0 (base register taken from result of two instructions previous)

(6) ForwardA = 0; ForwardB = 1 (rs1 = x15 taken from register; rs2 = x13 taken from result of 1st ld—two instructions ago)

(7) ForwardA = 0; ForwardB = 2 (base register taken from register file. Data to be written taken from previous instruction)

7.5 If there is no forwarding, what new input and output signals do we need for the hazard detection unit in the Figure above? Using this instruction sequence as an example, explain why each signal is needed.

The hazard detection unit additionally needs the values of rd that comes out of the MEM/WB register. The instruction that is currently in the ID stage needs to be stalled if it depends on a value produced by (or forwarded from) the instruction in the EX or the instruction in the MEM stage. So, we need to check the destination register of these two instructions. The Hazard unit already has the value of **rd** from the EX/MEM register as inputs, so we need only add the value from the MEM/WB register.

No additional outputs are needed. We can stall the pipeline using the three output signals that we already have.

The value of **rd** from EX/MEM is needed to detect the data hazard between the **add** and the following **ld**. The value of **rd** form MEM/WB is needed to detect the data hazard between the first **ld** instruction and the **or** instruction.

7.6 For the new hazard detection unit from Problem 6.5 of this HW assignment, specify which output signals it asserts in each of the first five cycles during the execution of this code.

Clock Cycle	1	2	3	4	5	6	7	8	9
add	IF	ID	EX	MEM	WB				
ld		IF	ID	-	-	EX	MEM	WB	
ld			IF	-	-	ID	EX	MEM	WB

(1) PCWrite = 1; IF/IDWrite = 1; control mux = 0

(2) PCWrite = 1; IF/IDWrite = 1; control mux = 0

(3) PCWrite = 1; IF/IDWrite = 1; control mux = 0

(4) PCWrite = 0; IF/IDWrite = 0; control mux = 1

(5) PCWrite = 0; IF/IDWrite = 0; control mux = 1