Karan Vora (kv2154)
Computer Systems Architecture Assignment 6

Problem 1):

The number of NOPs required to optimize the n-instruction without forwarding to handle data hazards is = 0.4 * n

Solution 1.1):

The cycle time of pipeline without forwarding is 250 ps

The number of NOPs required to optimize the n-instruction with forwarding to handle data hazards is 0.05 * n

The cycle time of pipeline with forwarding is 300 ps

Speed of the pipeline for n instructions without forwarding
= 250 * (n + 0.4 * n)
= 250 * 1.4 * n
= 350n ps

Speed of the pipeline for n instructions with forwarding
= 300 * (n + 0.05 * n)
= 300 * 1.05 * n
= 315n ps

The speedup of the new pipeline compared to the without forwarding is

$$= \frac{350-315}{350} * 100$$

$$= \frac{35}{350} * 100$$

= 10%

Therefore, the speedup of the new pipeline compared to the without forwarding by 10%

Solution 1.2):

Suppose that n-instruction program with xn NOPs and (1-x)n non-NOP instructions can run faster on the pipeline with forwarding (x(0.05 / 0.4))n NOPs and without forwarding (1 - x)n non – NOPs

Number of NOPs can remain in the typical program before that programs runs slower on the pipeline with forwarding is
= 250 ps * n < 300 ps * [(1 - x) + (0.125x)]n
= 5 < 6(1 – 0.875x)
= 5.25x < 1

= x < 0.1905
≈ 19.05% of Instructions

Therefore about 19.05% of code instructions

Solution 1.3):

The number of NOPs required to optimize the n-instruction with forwarding to handle data hazards is = 0.075 * n

Speed of the pipeline for n instructions with forwarding
= 300 * (n + 0.75 * n)
= 300 * 1.75 * n
= 525n ps

The speedup of the new pipeline compared to the without forwarding is

$$= \frac{525-315}{525} * 100$$

$$= \frac{210}{525} * 100$$

= 40%

Therefore, the speedup of the pipeline compared to the without forwarding by 40%

Solution 1.4):

Speedup of the new pipeline compared to the without forwarding by 40%

The cycle time of pipeline with forwarding is 300 ps

Speed of the pipeline for n-instructions with forwarding = 300 * 40 / 100 = 120 NOPs

Solution 1.5):

The larger the number of NOPs per instruction, x without forwarding, the easier it is for forwarding hardware to work and lower the number of NOPs per instruction. oAs the number of NOPS per instruction without forwarding, x decreases, it becomes physically impossible to go faster with forwarding if there are very few data hazards. This minimum occurrence of data hazards is when x is at a minimum corresponding to the case of γ = 0.

==========

Problem 2):

Solution 2.1):

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13
sd x29, 12(x16)	IF	ID	EX	MEM	WB								
ld x29, 8(x29)		IF	ID	EX	MEM	WB							
sub x17, x15, x14			IF	ID	EX	MEM	WB						
beqz x17, label				Stall	Stall	Stall	IF	ID	EX	MEM	WB		
add x15, x11, x14								IF	ID	EX	MEM	WB	
Sub x15, x30, x14									IF	ID	EX	MEM	WB

Solution 2.2):

Structural hazards are due to resource sharing. This cannot be eliminated by reordering instructions as every instructions will go through all stages of pipelining. This can be solved either by increasing hardware resources or by stalling the pipeline.

Solution 2.3):

It has to be handled by hardware. NOPs can handle data hazards but not structural hazards because NOPs is also an instruction which needs to be fetched like other instructions => to eliminate structural hazard a hardware hazard detection unit has to be used.

Solution 2.4):

3 stall cycles.

==========

Problem 3):

Solution 3.1): The clock period won't change because we aren't making any changes to the slowest stage.

Solution 3.2): Moving the MEM stage in parallel with the EX stage will eliminate the need for a cycle between loads and operations that use the result of the loads. This can potentially reduce the number of stalls in a program.

Solution 3.3): Removing the off set from ld and sd may increase the total number of instructions because some ld and sd instructions will need to be replaced with a addi/ld or addi/sd pair.

==========

Problem 4):

Choice 2 describes pipeline hazard detection in a better way. Because the different cycles of the instruction executed are mentioned as next instruction was started before the first on get executed.

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

==========

Problem 5):

Solution 5.1):

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. Availability of full forwarding support is assumed, so we can assume data hazards that can be resolved with forwarding do not stall the pipeline. Load-use-hazards cannot be resolved and are identified in next slide in red boxes - resolved by stalling the pipeline. pipeline stages unused by any instruction are identified in blue. Any clock cycle that does not have all of the pipeline stages utilized is identified with ‘N’.

Solution 5.2):

==========

Problem 7):

Solution 7.5):

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.

Solution 7.6):

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

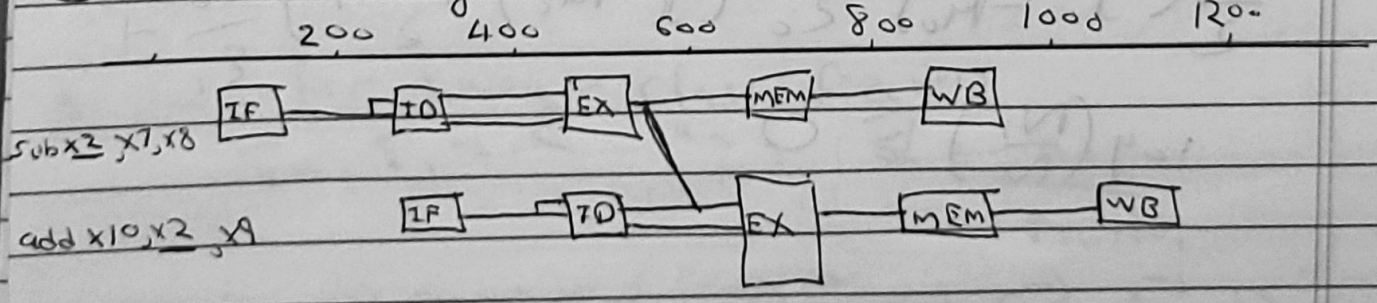
Solution 5.2):

```
loop ld x10, 0(x13)   IF ID EX ME | WB
ld x11, 8(x13)          IF ID EX | ME WB
add x12, x10, x11         IF ID | .. EX ME WB
addi x13, x13, -16          IF | . . ID EX ME WB
bnez x12, loop                | .. IF ID EX ME WB
ld x10, 0(x13)                  IF ID EX ME WB
ld x11, 8(x13)                    IF ID EX ME WB
add x12, x10, x11                 IF ID .. EX | ME WB
addi x13, x13, -16                  IF .. ID | EX ME WB
bnez x12, loop                        IF | ID EX ME WB
```

Problem (6)

Solution (6.1) For each RAW dependency listed in table previous slide, give a sequence of atleast three assembly statements that exhibits that dependency.

(1) EX to 1st Only



(1)

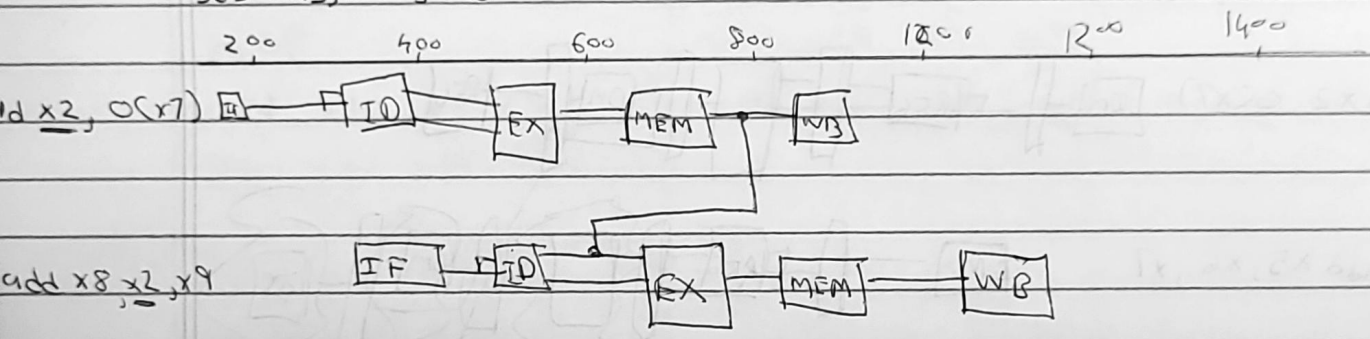
(2)

(2) MEM to 1st only:-

ld x2, 0(x7)

add x8, x2, x9

sub x15, x22, x23

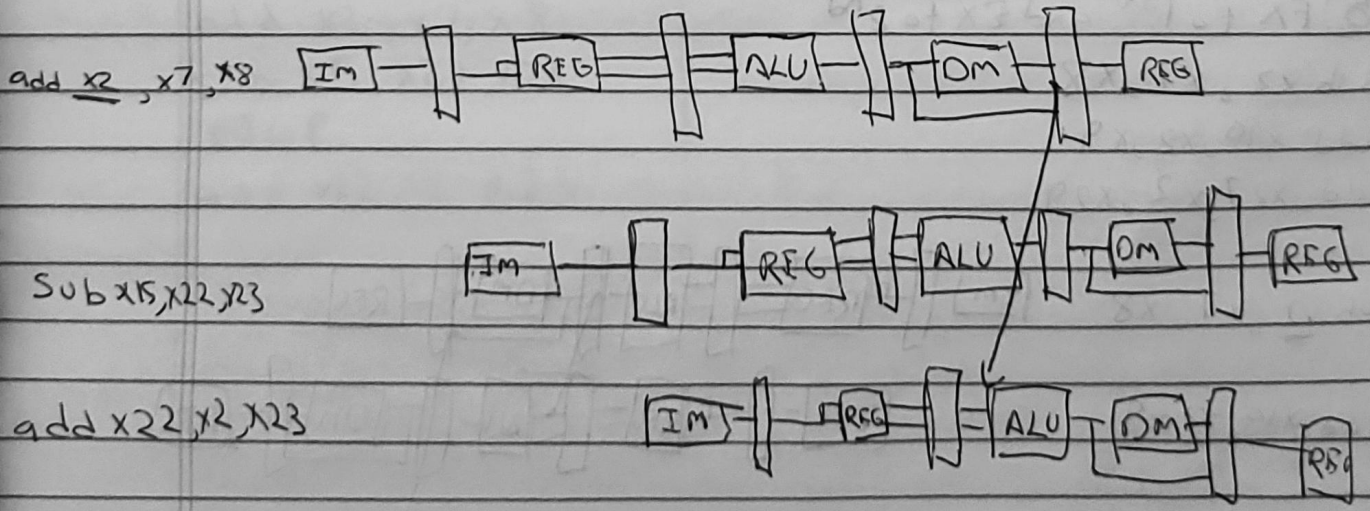


(3) Ex to 2nd only:-

add x2, x7, x8

sub x15, x22, x23

add x22, x2, x23

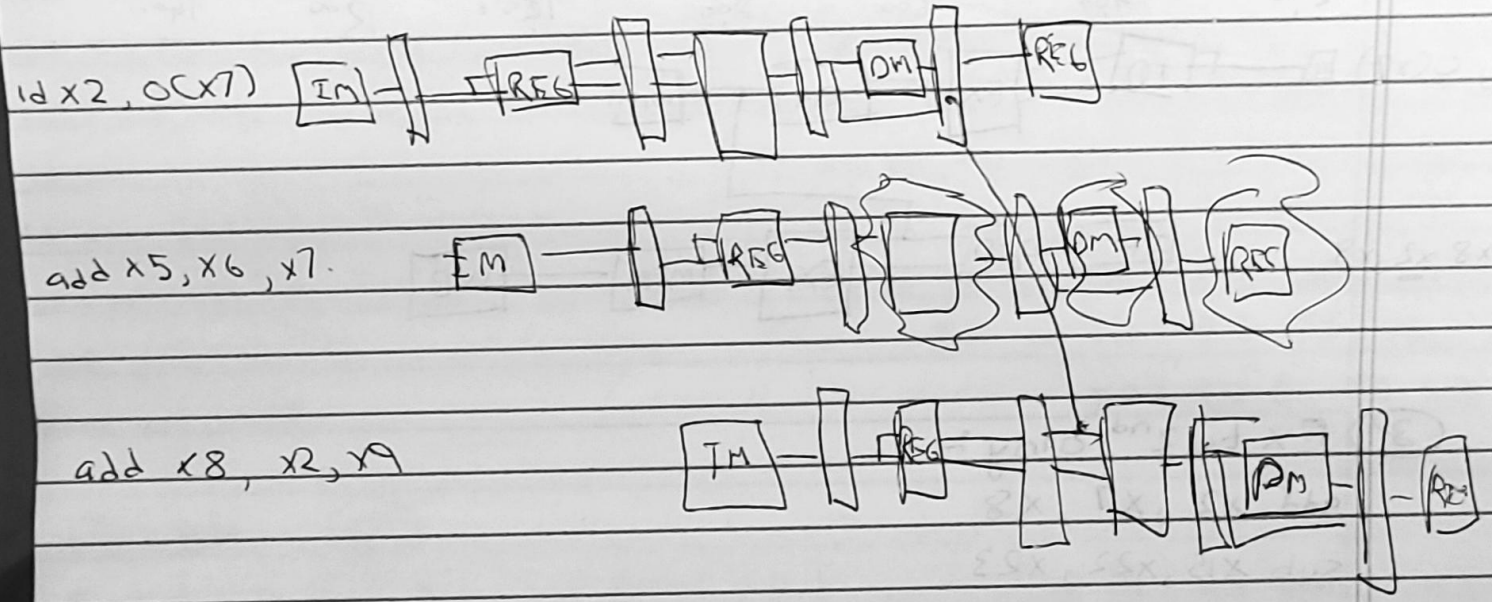


④ MEM to 2nd Only:

ld x2, 0(x7)

add x5, x6, x7

add x8, x2, x9

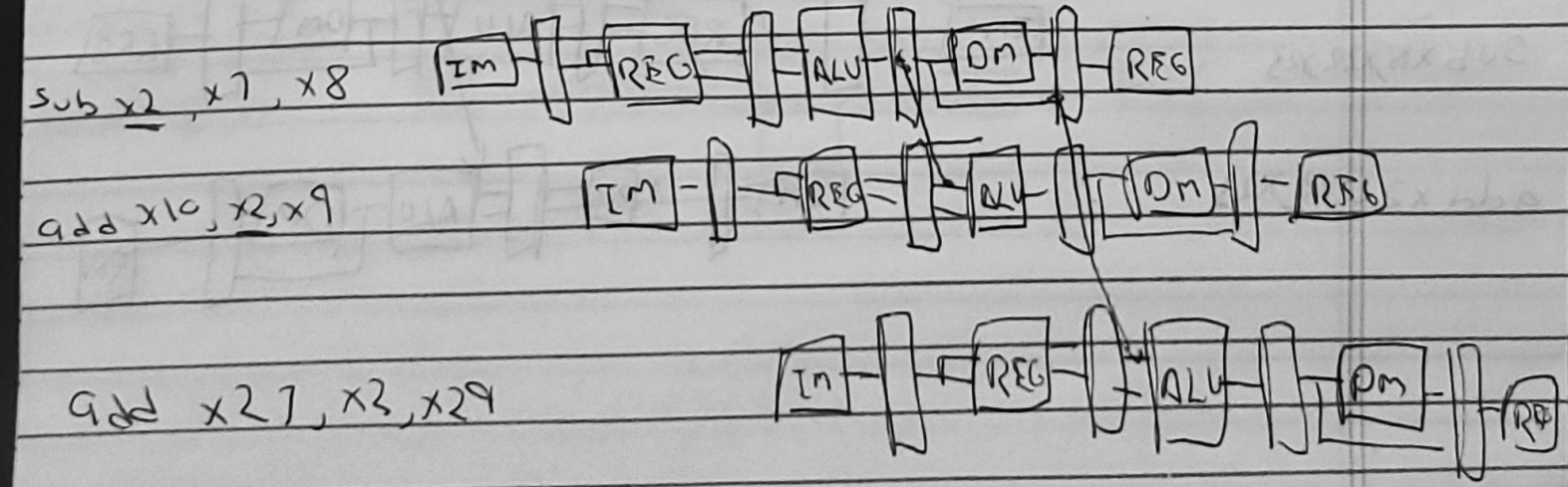


⑤ EX to 1st and EX to 2nd

Sub x3, x7, x8

add x10, x2, x9

add x27, x2, x29



Solution 6.2, 6.3

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 x5, 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
```

Solution 6-4

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 $0.85 / 1.85 = 46\%$ are NOPs

Solution 6.5

The only RAW dependency that cannot be handled by load-use-data hazard with forwarding is from the MEM Stage to the next instruction. 20% of Instructions will generate one NOP from CPI of 1.2, $0.2 / 1.2 = 17\%$ NOPs

Solution 6-6

If we Forward from to EX/MEM register only we have the following Stalls/NOPs

RX to 1st : 0

This hazard needs EX/MEM register with which the forwarding unit eliminates the NOP, so NOPs = 0

MEM to 1st : 2

Needs the MEM/WB register cannot be solved so 2 NOPs must be inserted

EX to 2nd : (1)
This hazard needs MEM/WB register and cannot be resolved so NOP must be inserted

MEM to 2nd (1)
This hazard needs MEM/WB register and cannot be resolved, so a NOP is inserted

EX to 1st and 2nd (1)
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.

=) If we forward from the MEM/WB register, we have the following stalls /NOPs
EX to 1st : 1
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)
This hazard needs the MEM/WB register but 1 NOP must be inserted.

EX to 2Nd: 0
This hazard needs the MEM/WB register and is resolved with 0 NOPs

MEM to 2Nd: 0
THIS hazard needs the MEM/WB register and is resolved with 0 NOPs

EX to 1st and 2nd
EX to 1st cannot be resolved and needs 1 NOP, but the EX to 2nd can be resolved with the MEM/WB register

⇒ CPI with Partial forwarding with MEM/WB register unavailable
Avg NOP = 0.05*0 + 0.2*2 + 0.2*1 + 0.1*1 + 0.1*1
= 0.85 stalls/Instruction
CPI = 1.65

With EX/MEM register unavailable.
Avg NOP, of 0.05*1 + 0.2*1 + 0.1
= 0.35 stalls/Instruction
CPI = 1.35

Solution (6.7)

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

Solution (6.8)

CPI with full forwarding is 1.2
CPI for time travel forwarding is 1.0
clock period for full forwarding is 130
clock period for zero hazards forwarding is 130
Speedup - Told / Tnew = (1.2*130) / (1*130) = 0.92
Zero hazard forwarding actually slows the CPU

Problem (7)

Solution (7.1)

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

Solution (7.2)

Not possible to reduce the Number of NOP

Solution (7.3)

The code executes correctly, we need hazard detection only to insert a stall when the instruction following a load uses the result of the load. That does not happen in this case.

Solution (7.4)

Because there are no stalls in the code, PCWrite and IF/IDWrite are

```
add  IF ID EX ME WB
lw      IF ID EX ME WB
ld         IF ID EX ME WB
or            IF ID EX ME WB
sd               IF ID EX ME WB
```

Solution (5)

The instruction that is currently in the ID Stage needs to stall it