ECE 6913, Computing Systems Architecture, Fall 2021 Quiz 2 Solutions

1. In the conventional fully bypassed 5-stage pipeline discussed in class, we were able to make the assumption that the ALU is able to complete in one cycle because we assumed integer operations. For this problem, we will assume that the ALU takes two cycles to complete. In other words, *the ALU is pipelined itself, making the entire pipeline 6 cycles*. The ALU only *generates a result after 2 cycles* (i.e., there is no way to extract any meaningful result after the ALU's first cycle). Memory (instruction and data) still returns data after only one cycle. You may ignore branch and jump instructions in this problem.

As a first step, we will examine how this changes data hazards. For this question, we will examine only the ALU-ALU read after write (RAW) hazard where the two instructions are consecutive:

```
ADD x1, x2, x3          #x1 <= x2 + x3

ADD x5, x4, x1          #x5 <= x4 + x1
```

(i) Describe *how would you resolve this hazard with the minimum number of bubbles (if any) using a combination of data forwarding and stalls (if necessary)*.

(ii) Then fill the following timing diagram to illustrate how the pipeline will behave, and show where data forwarding happens, if at all, by drawing an arrow between the two stages that participate in it.

	CC0	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
ADD x1, x2, x3	IF	ID	ALU 1	ALU 2	MEM	WB			
NOP									
ADD x5, x4, x1			IF	ID	ALU 1	ALU 2	MEM	WB	

Data forwarding alone does not suffice here because by the time the first ALU instruction completes the second ALU (execute) step, the second ALU instruction is already in the first ALU step and thus it's too late.

Therefore, a NOP between the 2 instructions, as well as data forwarding between the second ALU step and the decode step.

I-Mem	1 Register Read	Register Write	D-Mem Read	D-Mem Write
140pJ	70pJ	60pJ	140pJ	120pJ

Assume that components in the datapath have the following latencies. You can assume that the other components of the datapath have negligible latencies.

I-Mem	Control	Register Read or Write	ALU	D-Mem Read or Write
200 ps	150 ps	90 ps	90 ps	250 ps

1.1 How much energy is spent to execute an **addi** instruction in a single-cycle design and in the five-stage pipelined design

I-Mem is read, two registers are read, and a register is written
We have: 140pJ + 2*70pJ + 60j = 340pJ

1.2 How much energy is spent to execute a **lw** instruction in a single-cycle design

140pJ + 2*70pJ + 60pJ + 140pJ = 480pJ

1.3 How much energy is spent to execute a **beq** instruction in a single-cycle design

I-Mem + 2 registers = 140pJ + 2 * 70pJ = 280pJ

ECE 6913, Computing Systems Architecture, Fall 2021 Quiz 2 Solutions

1. In the conventional fully bypassed 5-stage pipeline discussed in class, we were able to make the assumption that the ALU is able to complete in one cycle because we assumed integer operations. For this problem, we will assume that the ALU takes two cycles to complete. In other words, *the ALU is pipelined itself, making the entire pipeline 6 cycles*. The ALU only *generates a result after 2 cycles* (i.e., there is no way to extract any meaningful result after the ALU's first cycle). Memory (instruction and data) still returns data after only one cycle. You may ignore branch and jump instructions in this problem.

As a first step, we will examine how this changes data hazards. For this question, we will examine only the ALU-ALU read after write (RAW) hazard where the two instructions are consecutive:

```
ADD x1, x2, x3          #x1 <= x2 + x3

ADD x5, x4, x1          #x5 <= x4 + x1
```

(i) Describe *how would you resolve this hazard with the minimum number of bubbles (if any) using a combination of data forwarding and stalls (if necessary)*.

(ii) Then fill the following timing diagram to illustrate how the pipeline will behave, and show where data forwarding happens, if at all, by drawing an arrow between the two stages that participate in it.

	CC0	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
ADD x1, x2, x3	IF	ID	ALU 1	ALU 2	MEM	WB			
NOP									
ADD x5, x4, x1			IF	ID	ALU 1	ALU 2	MEM	WB	

Data forwarding alone does not suffice here because by the time the first ALU instruction completes the second ALU (execute) step, the second ALU instruction is already in the first ALU step and thus it's too late.

Therefore, a NOP between the 2 instructions, as well as data forwarding between the second ALU step and the decode step.

Problem 3.

Consider the following RISC V Instruction sequence executing in a 5-stage pipeline:

```
or   x13, x12, x11
ld   x10, 0(x13)
ld   x11, 8(x13)
add  x12, x10, x11
subi x13, x12, 16
```

3.1 Identify all of the data hazards and their resolution with NOPs assuming no forwarding or hazard detection hardware is being used

Hazards identified:

```
or   x13, x12, x11
ld   x10, 0(x13)          EX to 1st RAW IHazard
ld   x11, 8(x13)          EX to 2nd RAW Hazard
add  x12, x10, x11        MEM to 1st RAW [load-use-data] & MEM to 2nd Hazards
subi x13, x12, 16         Ex to 1st RAW Hazard
```

NOPS introduced to resolve Hazards:

```
or   x13, x12, x11
NOPS
NOPS
ld   x10, 0(x13)          EX to 1st RAW Hazard resolution with 2 NOPs
ld   x11, 8(x13)          EX to 2nd RAW Hazard resolved as well from above 2 NOPs
NOPS
NOPS
add  x12, x10, x11        MEM to 1st RAW [load-use-data] & MEM to 2nd Hazards
                          resolved with 2 NOPs
NOPS
NOPS
subi x13, x12, 16         Ex to 1st only RAW Hazard resolved with 2 NOPs
```

3.2 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 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	10
1	or	IF	ID	**EX**	MEM	WB					
2	ld		IF	ID	**EX**	MEM	WB				
3	ld			IF	ID	EX	MEM	WB			
4	NOP	*mandatory NOP for which no forwarding solution possible: load-data-use*									
5	add					IF	ID	EX	MEM	WB	
6	subi						IF	ID	EX	MEM	WB

(1) A=x B=x *(no instruction in* EX *stage yet)*

(2) A=x B=x *(no instruction in* EX *stage yet)*

(3) A=0 B=0 *(both operands of the* or *instruction:* x11, x12 *come from Reg File)*

(4) A=2 B=0 *(base (*RS1*) in first* ld *(*x13*) taken from EX/MEM of previous instruction)*

(5) A=1 B=0 *(base (*RS1*) in 2nd* ld *(*x13*) taken from MEM/WB of a previous instruction)*

(6) A=x B=x *(no instruction in* EX *stage yet because NOP introduced to resolve MEM to 1st*

(7) A=0 B=1 *(*RS2 *in the* add *instruction is* x11 *which is forwarded from MEM/WB of 2nd* ld*, the result of the 1st* ld *(*x10*) has already been written into Reg File in CC 6 - so, no forwarding necessary for first operand)*

(8) A=1 B=0 *(RS1 of* subi *instruction forwarded from EX/MEM of* add *instruction)*

5. Indicate if the following modifications (A,B,C) will cause each of the three metrics (three rightmost columns) to *increase*, *decrease*, or have *no effect*. Explain your reasoning

Assume the initial machine is pipelined. Also assume that any modification is done in a way that preserves correctness and maintains efficiency, but that the rest of the machine remains unchanged.

		Instructions/Program	CPI (Cycles/Instruction)	Circuit complexity
A	Replace the 2 operand ALU with a 3 operand one and add 3 operand register-register instructions to the ISA (for example, `ADD rs1,rs2,rs3,rd`)	**Decrease** The new instructions will replace any two-instruction sequence that accomplished the same, such as `ADD rs1, rs2, rd` `ADD rs3, rd, rd`	**The same** The ALU will perform the three-way addition in one cycle. Also acceptable increase because more RAW	**Increase** Three-operand ALU is more complex than a two-operand one
B	Use the same ALU for instructions and for incrementing the PC by 4	**The same** No difference to instructions	**Increase** All instructions now use the same ALU ALU operations now have to stall	**Decrease** Now there is one less adder/ALU cycle
C	Increase the number of user registers from 32 to 64	**The same or decrease** If the more registers enable the Compiler to avoid loads and stores, Decrease. Otherwise the same.	**The same** Does not affect the actions of each instructions	**Increase** More registers complicate the register file

5) No. of stages of pipeline is 5 and stages are fetch, decode, ALU, memory & writeback stage. For given code we see Hazard b/w xor & sw instruction and also between addi & or instruction. As it is having Hazard unit, forwarding takes care of it.

At 5th cycle; instruction 1 xor is in write back stage as forwarding is there it is written to s_1 in cycle 4. So no operation in cycle 5.

At cycle 5, addi is in memory stage as it is arithmetic operation no operation done in this stage

At cycle 5, lw is in ALU stage and addition is performed, no reading or writing is performed.

At cycle 5, sw is in Decode stage and it reads from register s_1.

At cycle 5, or is in fetch stage, hence no reading/writing.

Hence only s_1 register is read in fifth cycle.

Assume 5 stages (IF(Instruction fetch), Instruction decode(ID), Execution(Ex), memory(MEM), WriteBack(WB))

IF, ID, MEM, WB takes 1 clock cycles, Ex takes 2 clock cycle.

I_1 Instruction ADD X_1, X_2, X_3 $X_1 \leftarrow X_2 + X_3$

I_2 Instruction ADD X_5, X_4, X_1 $X_5 \leftarrow X_4 + X_1$

There exists Read after write hazards (RAW) between I_1 & I_2. First Instruction should be executed then only second statement should be executed. If second instruction is executed first then x_1 take old value, not updated value. by I_1 instruction.

Clock cycle →

	CC0	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1	IF	ID	Ex	Ex	Mem	WB				
I_2		IF	ID	—	—	—	Ex	Ex	Mem	WB

3 Stall cycle if we NOT use Datafurwarding.

First Instruction executed normally. we can fetch, we can decode second instruction but we can not execute I_2 as because I_1 instruction is not executed and write back updated. We can execute I_2 only after CC5 (clock cycle 5) when I_1 is executed completely.

(a) Resolving hazards by data forwarding

Output of Execution stage is forwarded to input of Execution stage of I_2 instruction.

data forwarding b/w Execution stage of I_1 & Execution stage of I_2. By doing data forwarding we have minimize no. of stall cycle (Specially 2 stall cycle).

Without using data forwarding = 03 stall cycles

With data forwarding = 01 stall cycle.

(b)

clock cycles →

	CC_0	CC_1	CC_2	CC_3	CC_4	CC_5	CC_6	CC_7	CC_8	CC_9
I_1	IF	ID	Ex	Ex	MEM	WB				
I_2		IF	ID	–	Ex	Ex	MEM	WB		

1 stall cycle

To minimize the stall in the Pipeline because of true data dependency existed b/w two instruction. we use here operand forwarding. also called Bipassing or short circuitary.

operand forwarding means " one stage ALU o/p is connected as another stage ALU i/p. "

CS Scanned with

I-Mem	1 Register Read	Register Write	D-Mem Read	D-Mem Write
140pJ	70pJ	60pJ	140pJ	120pJ

Assume that components in the datapath have the following latencies. You can assume that the other components of the datapath have negligible latencies.

I-Mem	Control	Register Read or Write	ALU	D-Mem Read or Write
200 ps	150 ps	90 ps	90 ps	250 ps

1.1 How much energy is spent to execute an **addi** instruction in a single-cycle design and in the five-stage pipelined design

```
I-Mem is read, two registers are read, and a register is
written
We have: 140pJ + 2*70pJ + 60j = 340pJ
```

1.2 How much energy is spent to execute a **lw** instruction in a single-cycle design

```
140pJ + 2*70pJ + 60pJ + 140pJ = 480pJ
```

1.3 How much energy is spent to execute a **beq** instruction in a single-cycle design

```
I-Mem + 2 registers = 140pJ + 2 * 70pJ = 280pJ
```

i)

ii) bge x6, x5, COND2
add x5, x5, 1
COND2: add x6, x6, -1

iii) blt x6, x5, COND2
addi x5, x0, 0. # q=0
COND2: addi x6, x0, 0. # r=0.