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.
```

2. 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
```

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

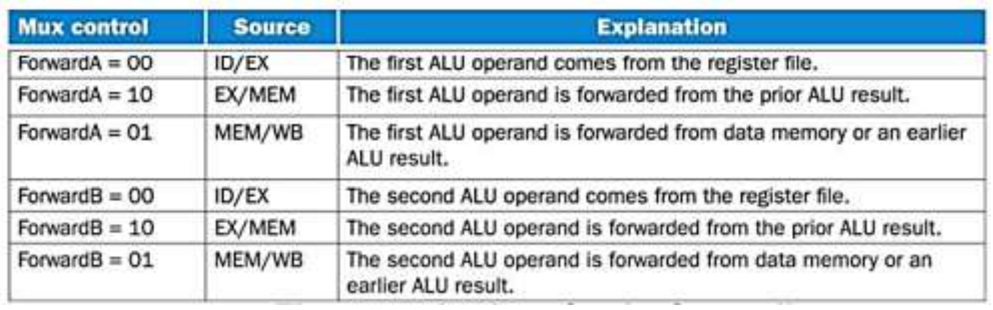
<u>Hazards identified:</u>

```
or   x13, x12, x11
ld   x10, 0(x13)           EX to 1st RAW Hazard
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
```

<u>NOPS introduced to resolve Hazards:</u>

```
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
```

2.2 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)*

3. You purchased an old IBM System whose model number is unknown so you do not have access to any manuals or spec sheets of the computer – except those listed below by a previous user:

1. 95% of all memory accesses are found in the cache.
2. Each cache block is two words, and the whole block is read on any miss.
3. The processor sends references to its cache at the rate of 10^9 words per second.
4. 25% of those references are writes.
5. Assume that the memory system can support 10^9 words per second, reads or writes.
6. The bus reads or writes a single word at a time (the memory system cannot read or write two words at once).
7. Assume at any one time, 30% of the blocks in the cache have been modified.
8. The cache uses write allocate on a write miss.

You are considering adding an IBM compatible peripheral to the system, and you want to know *how much of the memory system bandwidth is already used*. Calculate the percentage of memory system bandwidth used on the average in the two cases below. Be sure to state your assumptions.

3.1 The cache is <u>Write-Through</u>.

3.2 The cache is <u>Write-Back</u>.

We know:
* Miss rate = 0.05

* Block size = 2 words (8 bytes)

* Frequency of memory operations from processor = 10^9

* Frequency of writes from processor = $0.25 * 10^9$

* Bus can only transfer one word at a time to/from processor/memory

* On average 30% of blocks in the cache have been modified (must be written back in the case of the write back cache)

* Cache is write allocate

So:

Fraction of read hits = $0.75 * 0.95 = 0.7125$

Fraction of read misses = $0.75 * 0.05 = 0.0375$

Fraction of write hits = $0.25 * 0.95 = 0.2375$

Fraction of write misses = $0.25 * 0.05 = 0.0125$

6.1 Write through cache

- On a read hit there is no memory access
- On a read miss memory must send two words to the cache
- On a write hit the cache must send a word to memory

- On a write miss memory must send two words to the cache, and then the cache must send a word to memory

Thus:

Average words transferred = $0.7125 * 0 + 0.0375 * 2 + 0.2375 * 1 + 0.0125 * 3 = 0.35$

Average bandwidth used = $0.35 * 10^9$

Fraction of bandwidth used =

$[0.35 \text{ x } 10^9] / 10^9$

= 0.35 (1)

6.2 Write back cache

On a read hit there is no memory access

On a read miss:

1. If replaced line is modified then cache must send two words to memory, and then

memory must send two words to the cache

2. If replaced line is clean then memory must send two words to the cache

On a write hit there is no memory access

On a write miss:

1. If replaced line is modified then cache must send two words to memory, and then memory must send two words to the cache

2. If replaced line is clean then memory must send two words to the cache

Thus:

Average words transferred = $0.7125 * 0 + 0.0375 * (0.7 * 2 + 0.3 * 4) + 0.2375 * 0 + 0.0125 *$

$(0.7 * 2 + 0.3 * 4) = 0.13$

Average bandwidth used = $0.13 * 10^9$

Fraction of bandwidth used =

$0.13 \text{ x } 10^9/10^9$

= 0.13 (2)

Comparing 1 and 2 we notice that the write through cache uses more than twice the cache-memory bandwidth of the write back cache.

4A. Given a 4-way set associative cache of total size 2MB that has a 26-bit cache address and blocks of size 8KB each, answer the following questions.

4.1 How many bits in the "index" field of the cache address? Explain your answer

If the cache is 4-way set associative, then the number of sets is not 4. Rather, the number of sets is the number of rows in the cache divided by 4. Since this cache has 256 rows (2MB / 8KB), there are 64 sets. The answer is then: log_2(64) = 6 index bits

4.2 How many bits in the "offset" field of the cache address? Explain your answer

Observe: 8KB/block → 2^{13} bytes per block → $\log_2(2^{13})$ = 13 bits if byte-aligned. Assuming that the blocks are word-aligned, then we have Offset_bits = 13bits-2 bits = 11bits

4.3 How many bits in the "Tag" field of the cache address?

26 – 6 – 11 = 9 bits

4B. Mark whether the following modifications to cache parameters will cause each of the categories to increase, decrease, or whether the modification will have no effect. You can assume the baseline cache is set associative. Explain your reasoning. Assume that in each case the other cache parameters (number of sets, number of ways, number of bytes/line) and the rest of the machine design remain the same.

	compulsory misses	conflict misses	capacity misses
increasing number of sets	no effect block size is constant	decreases more sets are available for data, so there is less of a chance for two ops to collide and evict one another	decreases capacity increases
increasing number of ways	no effect block size is constant	decreases there are more ways available for data to be placed into	decreases capacity increases
increasing number of bytes per line	decreases more data is brought in on a given cache miss	no effect associativity and number of sets is constant	decreases capacity increases

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

6. This problem explores energy efficiency and its relationship with performance. The parts of this problem assume the following energy consumption for activity in Instruction memory, Registers, and Data memory. You can assume that the other components of the datapath consume a negligible amount of energy. ("Register Read" and "Register Write" refer to the register file only.)

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

6.1 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

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

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

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

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