ECE 6913, Computing Systems Architecture

Spring 2020 NYU ECE

Please fill in your name: _____

Quiz 2, May 4th 2020

Maximum time: 80 minutes : **9:30 AM** - **11:00 AM** [80 minutes + 10 minutes to assemble PDF and upload]

Open Book, Open Notes,

Calculators allowed.

Must show your work in steps – to get any credit

This is NOT a group project You may NOT discuss, share your Quiz solutions with anyone else.

You must stay logged in to Zoom throughout the Quiz

Instructor available online if you have questions on the Quiz, during the Quiz – enter question in Zoom chat box at any time during Quiz

This Test has 4 problems. Please attempt all of them. Please show **all work**. Please write **legibly**

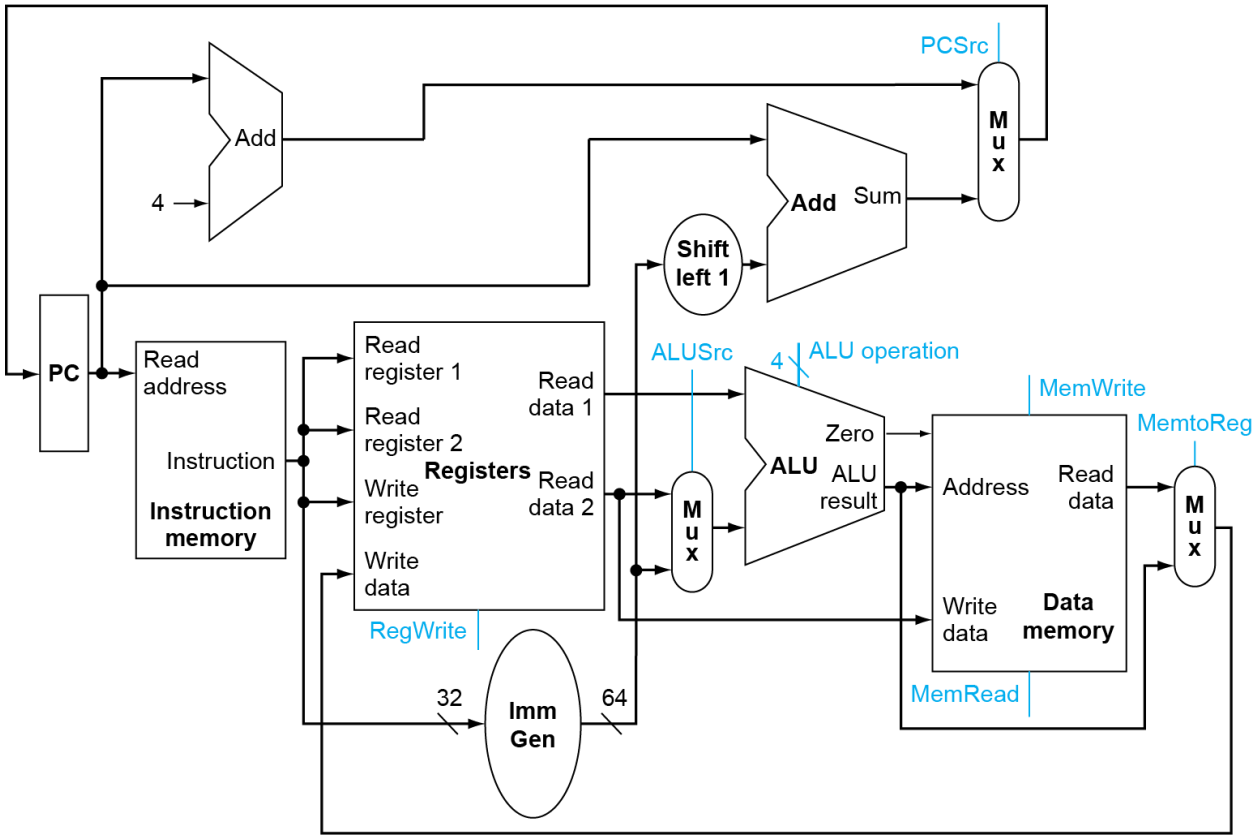
1. Please be sure to have 5-10 sheets of white or ruled paper & a Pencil, Eraser
2. Write down your solutions on 8.5 x 11 sheets of white paper, single sided with your name printed in top right corner of each sheet and with **Page Number and Problem number identified clearly on each sheet**
3. **Stop working on your Quiz at 10:50 AM** – you have 10 minutes to scan/take pictures of each sheet and upload them as completed PDF assignment to NYU Classes – you may use any of several smartphone apps to integrate your scans/pictures of sheets into a PDF file
4. Take pictures of each sheet and **upload** the PDF of all sheets after checking you have all sheets in the right order **by 11:00 AM latest.**
5. You may use iPAD to write down your solutions directly rather than on paper
6. Portal **will close at 11:00 AM** not allowing upload of your quiz after 11:00 AM

1. Consider a datapath similar to the one in figure below, but for a processor that only has one type of instruction: *unconditional PC-relative branch*. What would the cycle time be for this datapath? How about the case for a *conditional PC-relative branch - What would its cycle time be?*

I-Mem, D-Mem	Register File	Mux	ALU	Adder	Shift left 2	Register Read	Register Setup	Sign Extend	Control
250ps	150ps	25ps	200ps	150ps	20ps	30ps	20ps	50ps	50ps

"Register read" is the time needed after the rising clock edge for the new register value to appear on the output. This value applies to the PC only.

"Register setup time" is the amount of time a register's data input must be stable before the rising edge of the clock. This value applies to ***both the PC and Register File****.*



(i) A *conditional* PC relative branch needs the ALU to perform a subtraction to determine if the source registers identified in the instruction are equal

PC → Instr Memory → Register File → Mux → ALU → Single Gate → Mux → PC Register Setup Time → PC
30ps + 250ps +150ps + 25ps + 200ps + 5ps + 25ps + 20ps = **705ps**

Note that the path to calculate the branch address is faster than the concurrent path above between Register File and PC Mux

(ii) An *unconditional* PC relative branch instruction does not need the ALU to determine if a branch is taken.
The path of execution is:

PC → Instr Memory → Sign-extend → Shift left 2 → Add → Mux → Setup delay for PC Registers →PC

From the above Table, this equals
30ps + 250ps + 50ps + 20ps + 150ps + 25ps + 20ps = **545ps**

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

You are considering adding a peripheral to the system, and *you want to know how much of the memory system bandwidth is already used*.

(a) Calculate the percentage of memory system bandwidth used assuming the cache is Write Back.
(b) Calculate the percentage of memory system bandwidth used assuming the cache is Write Through.
Be sure to state your assumptions.

We know:

* Miss rate = 0.05

* Block size = 2 words (8 bytes)

* Frequency of memory operations from processor = 109

* Frequency of writes from processor = $0.25 * 109$

* 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 = $0.25 * 0.05 = 0.0125$

2 (b) 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 =

$[0.35 \times 10^9] / 10^9$

$= 0.35$ (1)

2(a) 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 =

$0.13 \times 10^9/10^9$

$= 0.13$ (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.

3. Assume we have a computer where the CPI is 1.0 when all memory accesses (including data and instruction accesses) hit in the cache. The cache is a unified (data + instruction) cache of size 256 KB, 4-way set associative, with a block size of 64 bytes. The data accesses (loads and stores) constitute 50% of the instructions. The unified cache has a miss penalty of 25 clock cycles and a miss rate of 2%. Assume 32-bit instruction and data addresses.

1.1 What are the Number of bits used for block offset?
Block size = 64 bytes, so 6 bits decode one of 64 bytes in a Block. Number of bits used for Block offset = 6

1.2 What are the Number of sets in the cache?
256KB /(64B x 4way) = 1K sets

1.3 What are the Number of bits for the cache index?
bits for the cache index pick one of all available sets ➔10 bits

1.4 What are the Number of bits for the tag?
Address has a total of 32 bits of which 6 bits used for Block offset and 10 bits used for the cache index. So, 32 – 16 = 16 bits for the Tag

1.5 Calculate the number of Stall Cycles per instruction
CPI for a computer that always hits = 1
Stall_Cycles_per_instruction = [Memory accesses per instr] x MR x MP
Memory accesses per instr. = 1 (for Instruction) + 0.5 (for Data) = 1.5

SCPI= 1.5 mem access/instruction x 0.02 misses/access x 25 cycles = 0.75 cycles/instruction

1.6 How much faster would the computer be if all memory accesses were cache hits?

1.75/1 = 1.75 faster

4. 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
```

4.1 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 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
```

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
```

4.2 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)*