ECE 6913, Computing Systems Architecture, Fall 2019

Quiz 2, November 20th 2019

Maximum time: 2.5 hours : **9:50 AM – 12:20 PM**

Closed Book, Closed Notes, Calculators allowed. RISC V Card provided

Devices with internet connectivity not allowed

This Test has 5 problems each with several parts. Please attempt all of them. Please show all work. Please write legibly

1. Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 64-bit memory address references, given as word addresses.

`0xa2, 0xf4, 0xf5, 0xef, 0xaf`

1.1 For each of these references, *identify* (i) the binary word address, (ii) the tag, and (iii) the index given a **direct-mapped cache** with **32 one-word blocks**. Also, list (iv) whether each reference is a hit or a miss, assuming the cache is initially empty.

Hex Memory Reference	Binary Reference	Tag	Index	Hit / miss
0xa2	1010 0010	101	0 0010	m
0xf4	1111 0100	111	1 0100	m
0xf5	1111 0101	111	1 0101	m
0xef	1110 1111	111	0 1111	m
0xaf	1010 1111	101	0 1111	m

1.2 For each of these references, identify (i) the binary word address, (ii) the tag, (iii) the index, and (iv) the offset given a **direct-mapped cache** with **two-word blocks** and a **total size of 16 blocks**. Also, list (iv) if each reference is a hit or a miss, assuming the cache is initially empty.

Hex Memory Reference	Binary Reference	Tag	Cache Index	Block Offset	Hit / miss
0xa2	1010 0010	101	0001	0	m
0xf4	1111 0100	111	1010	0	m
0xf5	1111 0101	111	1010	1	h
0xef	1110 1111	111	0111	1	m
0xaf	1010 1111	101	0111	1	m

1.3 You are asked to **optimize a cache design** for the given references. There are **three direct-mapped cache designs possible**, all with a *total of 128 words of data*: C1 has 4-word blocks, C2 has 8-word blocks, and C3 has 16-word blocks.

Word Address	Binary Address	Tag bits in hex	Cache 1 block size = 4 word			Cache 2 block size = 8 word			Cache 3 block size = 16 word		
			index	offset	Hit/ Miss	index	offset	Hit/ Miss	index	offset	Hit/ Miss
0xa2	1010 0010	1	0 1000	10	m	0100	010	m	010	0010	m
0xf4	1111 0100	1	1 1101	00	m	1110	100	m	111	0100	m
0xf5	1111 0101	1	1 1101	01	h	1110	101	h	111	0101	h
0xef	1110 1111	1	1 1011	11	m	1101	111	m	110	1111	m
0xaf	1010 1111	1	0 1011	11	m	0101	111	m	010	1111	h

Cache 1: 32 blocks; **Cache 2:** 16 blocks; **Cache 3:** 8 blocks

1.4 How does the Miss rate depend on the size of the Block? Why? Can you list 2 opportunities to improve latency that accompanies increase in block size?

Miss rates lower as block size gets larger in number of words – due to spatial locality of data.

we may be able to hide some of the transfer time so that the miss penalty is effectively smaller. The easiest method for doing this, called **early restart**, is simply to resume execution as soon as the requested word of the block is returned, rather than wait for the entire block. Many processors use this technique for instruction access, where it works best. Instruction accesses are largely sequential, so if the memory system can deliver a word every clock cycle, the processor may be able to restart operation when the requested word is returned, with the memory system delivering new instruction words just in time. This technique is usually less effective for data caches because it is likely that the words will be requested from the block in a less predictable way, and the probability that the processor will need another word from a different cache block before the transfer completes is high. If the processor cannot access the data cache because a transfer is ongoing, then it must stall.

An even more sophisticated scheme is to organize the memory so that the requested word is transferred from the memory to the cache first. The remainder of the block is then transferred, starting with the address after the requested word and wrapping around to the beginning of the block. This technique, called **requested word first** or **critical word first**, can be slightly faster than early restart, but it is limited by the same properties that restrain early restart

2. Consider the following program and cache behaviors.

Data Reads per 1K instructions	Data Writes per 1K instructions	Instruction Cache Miss Rate	Data Cache Miss Rate	Block Size (Bytes)
300	150	0.5%	5%	128

2.1 Suppose a CPU with a write-through, write allocate cache achieves a CPI of 2. What are the read and write bandwidths (measured by bytes per cycle) between RAM and the cache? (Assume each miss generates a request for one block.). *For a write-allocate policy, a write miss also makes a read request to RAM – please be sure to consider its impact on Read Bandwidth*

Instruction Bandwidth:
When the CPI is 2, there are, on average, 0.5 instruction accesses per cycle.
0.5 instructions read from Instruction memory per cycle
0.5% of these *instruction* accesses cause a cache ***Read*** miss (and subsequent memory request).
[0.5 instr/cycle] x [0.005 misses/instruction] = missed instructions/cycle
Assuming each miss requests one block and each block is 128 bytes [16 words with 8 bytes (64 bits) per word] , instruction accesses generate an average of
[0.5 instr/cycle] x [0.005 misses/instruction] x[128 bytes/miss] =
= 0.32 bytes/cycle of *read traffic*

Read Data bandwidth:
30% of instructions generate a ***read*** request from data memory.
[0.5 instr/cycle] x [0.3 Read Data Accesses/instruction] = [0.15 Read Data Accesses / cycle]
5% of these generate a cache miss;
[0.15 Read Data Accesses / cycle] x [0.05 misses / Read Data Access] = 0.0075 Read Misses/cycle
Assuming each miss requests one block and each block is 128 bytes [16 words with 8 bytes (64 bits) per word] ,
[0.0075 Read Misses/cycle] x [128 Bytes/block] x [1 block/miss] = 0.0075 x 128 Bytes/cycle
= 0.96 Bytes/cycle
Write Data bandwidth:
15% of instructions generate a ***write*** request into data memory.
[0.5 instr/cycle] x [0.15 Write Data Accesses/instruction] = [0.075 Write Data Accesses / cycle]
All of the words written to the cache must be written into Memory:
[0.075 Write Data Accesses / cycle] x [8 bytes/word] x [1 word/write-through] = 0.6 Bytes/cycle

For a *Write-allocate policy*, a Write miss also makes *a **read** request* to RAM
[0.5 inst/cycle] x [0.15 Write Data Accesses/instruction] x [0.05 misses/Write Data Access] x [128 Bytes/miss]
= 0.48 Bytes/cycle
Assuming each miss requests one Word (8 bytes) since this is a write-through cache *with only 1 word written per miss into memory*,
[0.00375 Write Misses/cycle] x [8 Bytes/word] x [1 word/miss] = 0.03 Bytes/cycle

Total Read Bandwidth
0.32 (Instruction memory) + **0.96** (data memory) + **0.48** (Write-miss in Write-through cache with Write Allocate) **Bytes/cyle = 1.76 Bytes/cycle**
Total Write Bandwidth:
0.6 Bytes/cycle + 0.03 Bytes/cycle = 0.63 Bytes/cycle

2.2 For a write-back, write-allocate cache, assuming 30% of replaced data cache blocks are dirty, what are the read and write bandwidths needed for a CPI of 2?

The instruction and data ***read*** bandwidth requirement is the same: **1.76 Bytes/cycle**

the data ***write*** bandwidth requirement becomes

[0.5 inst/cycle] x [0.15 Write Data Accesses/instruction + 0.3 Read Data Accesses/instruction]
= 0.225 Accesses/cycle
[0.225 Accesses/cycle] x [0.05 misses /Access]
= 0.01125 misses/cycle
[0.01125 misses/cycle] x [0.3 blocks/miss]
= 0.003375 blocks/cycle
[0.003375 blocks/cycle] x [128 bytes/block] =
=0.432 bytes/cycle

3. Assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows for the processor shown in the Figure below:

add	addi	not	beq	lw	sw
17%	18%	5%	25%	25%	10%

3.1 In what fraction of all cycles is the shift-left 2 unit used?

The shift left-2 unit is used only for branch instructions: 25%

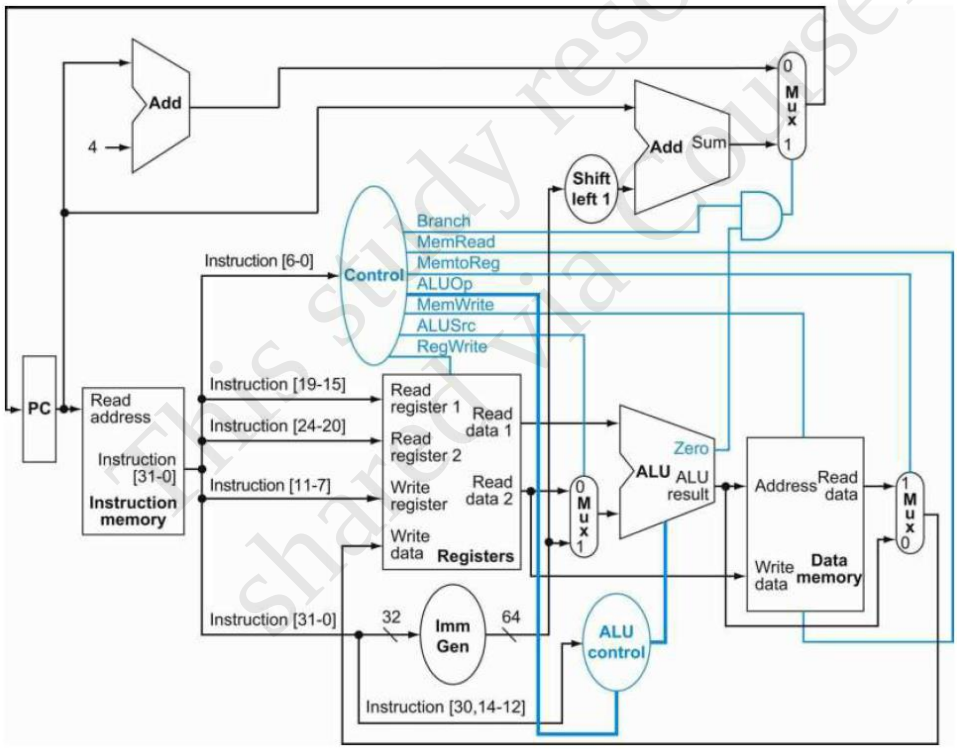
3.2 In what fraction of all cycles is the input of the sign-extend circuit needed?

addi, beq, lw, sw: 18 + 25 + 25 + 10 = 78%

3.3 What is the utilization of the *second* input data port of the ALU?

The second input port of the ALU is used in all instructions except the unconditional branch instruction (jal for example) where the ALU does not need to determine if a branch is to be taken.

add, addi, not, beq, lw, sw = 100%



4. 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

4.1 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

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

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

4.3 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 2020 NYU ECE

Please fill in your name: _____

Quiz 2, November 13th 2020

Maximum time: 140 minutes : 2 **PM** - **4:20 PM** [+ 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, with Camera on

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 4:20 PM** – 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 4:30 PM latest.**
5. You may use iPAD to write down your solutions directly rather than on paper
6. Portal **will close at 4:30 PM** not allowing upload of your quiz after this time

Problem 1.

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.

Problem 2.

One difference between a write-through cache and a write-back cache can be in the time it takes to write. During the first cycle, we detect whether a hit will occur, and during the second (assuming a hit) we actually write the data.

Let's assume that 50% of the blocks are dirty for a write-back cache. For this question, assume that the write buffer for the write through will never stall the CPU (no penalty). Assume a cache read hit takes 1 clock cycle, the cache miss penalty is 50 clock cycles, and a block write from the cache to main memory takes 50 clock cycles. Finally, assume the instruction cache miss rate is 0.5% and the data cache miss rate is 1%. Assuming that on average 26% and 9% of instructions in the workload are loads and stores, respectively, **estimate the performance of a write-through cache with a two-cycle write versus a write-back cache with a two-cycle write.**

CPU performance equation: CPUTime = IC * CPI * ClockTime

CPI = CPIexecution + StallCyclesPerInstruction

We know:

Instruction miss penalty is 50 cycles

Data read hit takes 1 cycle

Data write hit takes 2 cycles

Data miss penalty is 50 cycles for write through cache

Data miss penalty is 50 cycles or 100 cycles for write back cache

Miss rate is 1% for data cache (MRD) and 0.5% for instruction cache (MRI)

50% of cache blocks are dirty in the write back cache

26% of all instructions are loads

9% of all instructions are stores

Then: CPIexecution = 0.26 * 1 + 0.09 * 2 + 0.65 * 1 = 1.09

Write through

StallCyclesPerInstruction = MRI * 50 + MRD * (0.26 * 50 + 0.09 * 50) = 0.425

so: CPI = 1.09 + 0.425 = 1.515 (1)

Write back

StallCyclesPerInstruction = MRI * 50 + MRD * (0.26 * (0.5 * 50 + 0.5 * 100) + 0.09 * (0.5 * 50 + 0.5 * 100)) = 0.5125

so: CP I = 1.09 + 0.5125 = 1.6025 (2)

Comparing 1 and 2 we notice that the system with the write back cache is 6% slower.

Problem 3.

. Consider the following program and cache behaviors.

Data Reads per 1K instructions	Data Writes per 1K instructions	Instruction Cache Miss Rate	Data Cache Miss Rate	Block Size (Bytes)
300	150	0.5%	5%	128

Suppose a CPU with a write-through, write allocate cache achieves a CPI of 2. What are the read and write bandwidths (measured by bytes per cycle) between RAM and the cache? (Assume each miss generates a request for one block.). *For a write-allocate policy, a write miss also makes a read request to RAM – please be sure to consider its impact on Read Bandwidth*

Instruction Bandwidth:
When the CPI is 2, there are, on average, 0.5 instruction accesses per cycle.
0.5 instructions read from Instruction memory per cycle
0.5% of these *instruction* accesses cause a cache ***Read*** miss (and subsequent memory request).
[0.5 instr/cycle] x [0.005 misses/instruction] = missed instructions/cycle
Assuming each miss requests one block and each block is 128 bytes [16 words with 8 bytes (64 bits) per word] , instruction accesses generate an average of
[0.5 instr/cycle] x [0.005 misses/instruction] x[128 bytes/miss] =
= 0.32 bytes/cycle of *read traffic*

Read Data bandwidth:
30% of instructions generate a ***read*** request from data memory.
[0.5 instr/cycle] x [0.3 Read Data Accesses/instruction] = [0.15 Read Data Accesses / cycle]
5% of these generate a cache miss;
[0.15 Read Data Accesses / cycle] x [0.05 misses / Read Data Access] = 0.0075 Read Misses/cycle
Assuming each miss requests one block and each block is 128 bytes [16 words with 8 bytes (64 bits) per word] ,
[0.0075 Read Misses/cycle] x [128 Bytes/block] x [1 block/miss] = 0.0075 x 128 Bytes/cycle
= 0.96 Bytes/cycle
Write Data bandwidth:
15% of instructions generate a ***write*** request into data memory.
[0.5 instr/cycle] x [0.15 Write Data Accesses/instruction] = [0.075 Write Data Accesses / cycle]
All of the words written to the cache must be written into Memory:
[0.075 Write Data Accesses / cycle] x [8 bytes/word] x [1 word/write-through] = 0.6 Bytes/cycle

For a *Write-allocate policy*, a Write miss also makes *a **read** request* to RAM
[0.5 inst/cycle] x [0.15 Write Data Accesses/instruction] x [0.05 misses/Write Data Access] x [128 Bytes/miss]
= 0.48 Bytes/cycle
Assuming each miss requests one Word (8 bytes) since this is a write-through cache *with only 1 word written per miss into memory*,
[0.00375 Write Misses/cycle] x [8 Bytes/word] x [1 word/miss] = 0.03 Bytes/cycle

Total Read Bandwidth
0.32 (Instruction memory) + **0.96** (data memory) + **0.48** (Write-miss in Write-through cache with Write Allocate) **Bytes/cyle = 1.76 Bytes/cycle**
Total Write Bandwidth:
0.6 Bytes/cycle + 0.03 Bytes/cycle = 0.63 Bytes/cycle

Problem 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)*

ECE 6913, Computing Systems Architecture

Spring 2021 NYU ECE

Please fill in your name: _____

Quiz 2, April 16th 2021

Maximum time: 80 minutes : **11:00 AM** - **1:00 PM** [includes time to assemble PDF and upload]

Open Book, Open Notes,

Calculators allowed.

Must show your work in steps – to get any credit

You may NOT discuss, share your Quiz solutions with anyone else.

You must stay logged in to Zoom throughout the Quiz, with your camera on – you may leave after you upload your 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 5 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 12:50 PM** – 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 1:00 PM latest.**
5. You may use iPAD to write down your solutions directly rather than on paper

Portal **will close at 1:00 PM** not allowing upload of your quiz after this time

1. Students M, O & P are building custom hardware and compilers for their project

The following observations are known. CPI = 1

• Student O has built his design and it has a known execution time for a new program of 600 seconds.

• Student M proposes a single cycle data path, and plans to use a compiler that will generate 300 Billion dynamic instructions per execution.

• Student P proposes a pipelined data path, with a clock period of 1.2 ns. His compiler will generate 400 Billion instructions. Of these 400 Billion instructions, 25% are loads, and 20% are branches. 40% of all loads cause a 2-cycle load-use stall. The penalty for a mis-predicted branch is 5 cycles. The processor has full bypassing, and does not suffer from any other stalls.

How fast does M need to make his clock cycle in order to make the program run faster than O's?

$Tcycle * 300B \text{ instructions} * CPI = Tcycle * 300B * 1 < 600 \text{ secs}$

$Tcycle < 600/300B = 2ns$

2. Assume that you have a computer with 1 clock cycle per instruction (CPI=1) when all accesses to memory are in cache. The only accesses to data come from load and store instructions. Those accesses account for 25 % of the total number of instructions. Miss penalty is 50 clock cycles and miss rate is 5 %. Determine the speedup obtained when there is no cache miss compared to the case when there are cache misses

Given: **cp**: Processor cycles. **cw**: Wait cycles. **T**: Clock period. **IC**: Number of instructions. **CPI**: Cycles per instruction. $\mathbf{a_i}$: Amount of accesses produced by instructions. $\mathbf{a_d}$: Amount of accesses produced by data. **mr**: Miss rate. **mp**: Miss penalty.

In case of no misses:

$Tcpu = (cp + cw) \cdot T = (IC \cdot CPI + 0) \cdot T = IC \cdot 1 \cdot T = IC \cdot T$

Including misses:

$cw = IC \cdot (a_i + a_d) \cdot mr \cdot mp$

$= IC(1 + 1 \cdot 0.25) \cdot 0.05 \cdot 50 = IC \cdot 1.25 \cdot 0.05 \cdot 50 = 3.125$

So,

$Tcpu = (IC \cdot 1 + IC \cdot 3.125) \cdot T = 4.125 \cdot IC \cdot T$

$Speedup = S = 4.125 \cdot IC \cdot T / [IC \cdot T] = 4.125$

3. When Program X is run, the user CPU time is 3 seconds, the total wall clock time is 4 seconds, and the system performance is 10 MFLOP/sec. Assume that there are no other processes taking any significant amount of time, and the computer is either doing calculations in the CPU, or doing I/O, but it can't do both at the same time. We now replace the processor with one that runs six times faster, but doesn't affect the I/O speed. What will the user CPU time, the wall clock time, and the MFLOP/sec performance be now?

$CPU\ performance_B / CPU\ performance_A = CPU\ time_A / CPU\ time_B$

$6 = 3/CPU\ time_B$

User CPU Time = .5 seconds

Since the I/O time is unaffected by the performance increase, it still takes 1 second to do I/O. Therefore, it takes 1 + .5 = 1.5 seconds to run Program A on the faster CPU

Wall clock Time = 1.5 seconds

System Performance in MFLOPS =

Number of Floating-Point Operations * 10^6 / Wall clock Time

Old System Performance (10 MFLOP/sec) = #FLOP * $10^6/4$

#FLOP = 40 * 10^6

New System Performance = 40 * 10^6 /1.5

= 26.67 MFLOP/sec

4. For the 5-stage pipeline, with forwarding hardware discussed in class, our assumption was that the ALU is able to complete in one cycle because we assumed integer operations. Here, we 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 one cycle. You may ignore branch and jump instructions in this problem.

Let's look at 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.

5. 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

(a) 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)

(b) 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.

NYU TANDON SCHOOL OF ENGINEERING

Review Problems Quiz 2

Azeez Bhavnagarwala

ECE 6913, Fall 2021

Computer Systems Architecture

NYU Tandon School of Engineering

Problem 1: AMAT improvement with L2

- The average memory access time for a microprocessor with 1 level of cache is 2.4 clock cycles
  - If data is present and valid in the cache, it can be found in 1 clock cycle
  - If data is not found in the cache, 80 clock cycles are needed to get it from off-chip memory
- Designers are trying to obtain a 65% improvement in average memory access time, and are considering adding a 2nd level of cache on-chip.
  - This second level of cache could be accessed in 6 clock cycles
  - The addition of this cache does not affect the first level cache's access patterns or hit times
  - Off-chip accesses would still require 80 additional CCs.

To obtain the desired speedup, how often must data be found in the 2nd level cache?

Solution

- We must first determine the miss rate of the L1 cache to use in the revised AMAT formula:
  - AMAT = Hit Time + Miss Rate x Miss Penalty
  - 2.4 = 1 + Miss Rate x 80
  - Miss Rate = 1.75%
- Next, we can calculate the target AMAT … i.e. AMAT with L2:
  - Speedup = Time (old) / Time (new)
  - 1.65 = 2.4 / Time (new)
  - Time (new) = 1.4545 clock cycles

Solution

- We can then again use the AMAT formula to solve for highest acceptable miss rate in L2:
  - 1.4545 = 1 + 0.0175 x (6 + (Miss Rate_L2)(80))
- Solving for Miss Rate_L2 suggests that the highest possible miss rate is ~24.96%
- **Thus, as the hit rate is (1 – Miss Rate), the L2 hit rate must be ~75%.**

Problem 2: CPI of MC Processor w Shared Memory

- Assume that the base CPI for a pipelined datapath on a single core system is 1.
  - Note that this does NOT include the overhead associated with cache misses!!!
- Profiles of a benchmark suite that was run on this single core chip with an L1 cache suggest that for every 10,000,000 accesses to the cache, there are 308,752 L1 cache misses.
  - If data is found in the cache, it can be accessed in 1 clock cycle, and there are no pipe stalls
  - If data is not found in the cache, it can be accessed in 10 clock cycles
- Now, consider a multi-core chip system where each core has an equivalent L1 cache:
  - All cores reference a common, centralized, shared memory
  - Potential conflicts to shared data are resolved by snooping and an MSI coherency protocol
- Benchmark profiling obtained by running the same benchmark suite on the multi-core system suggests that, on average, there are now 452,977 misses per 10,000,000 accesses.
  - If data is found in a cache, it can still be accessed in 1 clock cycle
  - On average, 14 cycles are now required to satisfy an L1 cache miss
- What must the CPI of the multi-core system be for it to be worthwhile to abandon the single core approach?

Solution

- We first need to calculate the CPI of the single core system and incorporate the overhead of cache misses:
  - CPI = 1 + (308,753 / 10,000,000)(10)
  - = 1.309
- We need to better this number with the multi-core approach. Thus:
  - 1.309 > X + (452,977 / 10,000,000)(14)
  - 1.309 > X + 0.634
  - 0.675 > X
- Thus, the CPI must be less than 0.675

Problem 3: Pipelining Vs Multi-cycle

Assuming that N instructions are executed with cycle time 305ps, and all N instructions are `add` instructions, what is the speedup of a pipelined implementation when compared to a multi-cycle implementation? Your answer should be an expression that is a function of N

- For the multi-cycle approach:

  Each add instruction would take 4 clock cycles and each clock cycle would take 305 ps.

  Thus, the total time would be: 1220(N)

- For the pipelined approach:

  For N instructions, Total Time : ST + (N-1)T = NT + (S-1)T

  Thus, the total time would be:

  = 305(N) + (5-1)(305)

  = 305N + 1220 ps

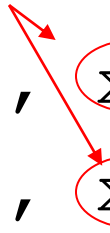
- Thus, the overall speedup is:

  1220(N) / [305(N) + 1220]

Problem 4: Pipeline Depth Trade-offs

- If the Data Memory access stage in a Pipeline has the largest latency, how can pipeline throughput be improved?
- Assume you break up the memory stage into 2 stages instead of 1 to improve throughput in a pipelined datapath (now operating with higher pipeline CLK frequency and the same total latency)
- The pipeline stages are now: F, D, EX, M1, M2, WB
- Show how the instructions below would progress though this 6 stage pipeline. Assume full forwarding hardware is available

```
lw x5, 0(x4)
add x7, x5, x5      load-use data hazard
sub x8, x5, x9
```



Solution

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`lw x5, 0(x4)`	F	D	EX	M1	M2	WB									
`add x7,x5,x5`		F	D	*	*	EX	M1	M2	WB						
`sub x8,x5,x9`			F	*	*	D	EX	M1	M2	WB					

Additional Cycle Penalty for Load-Use data hazard with Data Memory Access cycle split

Problem 5: Speedup with Multiple Cores

- Assume that you have 10 cores that you can use to solve a problem in parallel. 98% of your code is parallelizable
- Can you get a speedup of 7? If so, how many cores are needed?

Use Amdahl's Law:

$$7 = \frac{1}{(1 - 0.98) + \frac{0.98}{X}}$$

Solving, X = 7.97 or 8 Cores

Problem 6:

Briefly explain why many of the transistors on a modern microprocessor chip are devoted to Level 1, Level 2, and sometimes Level 3 cache.

1. Processing logic is faster than off-chip, 1-transistor DRAM – and this performance gap has been continually growing - Accessing DRAM has a severe Miss penalty in number of core clock cycles
2. Lower Miss Penalty by doing 2 things:
   - Add levels to the Memory hierarchy with smaller penalty in core clock cycles using faster on-chip memory (SRAM) that can be accessed within a few cycles of core clock rate
   - Increase the size of added levels of the Hierarchy to lower their miss rates and ensuing penalties to go off-chip to DRAM
3. Size of Processor chip goes up (and its cost) substantially. So does leakage power from the large number of SRAM bitcells now on-chip in the L2, L3

Problem 7a

- Some versions of the Pentium 4 microprocessor have two 8 Kbyte, Level 1 caches – one for data and one for instructions. However, a design team also considered another option – a single, 16 Kbyte cache that holds both instructions and data where
  - Each block will hold 32 bytes of data (not including tag, valid bit, etc.)
  - The cache would be 2-way set associative
  - Physical addresses are 32 bits
  - Data is addressed to the word and words are 32 bits

Solution

- How many blocks would be in this cache?
  - 2^{14} bytes in the cache (16KBytes) / 32 bytes/block = **512 blocks**
- How many bits of tag are stored with each block entry?
- 32 address Bits are dedicated to the offset, index and tag
- Index:
  - # of sets: 512 /2 = 256 = 2^8
  - Therefore **8** bits of index are needed
- Offset:
  - # of words per block = 32 / 4 = 8
  - 2^3 = 8
  - Therefore **3** bits for Block offset
- Tag
  - 32 – 3 – 8 = **21** bits of tag

Problem 7b

- Each instruction fetch means a reference to the instruction cache and 35% of all instructions reference data memory. With the first implementation:
  - The average miss rate in the L1 instruction cache was 2%
  - The average miss rate in the L1 data cache was 10%
  - In both cases, the miss penalty is 9 CCs
- For the new unified design, the average miss rate is 3% for the cache as a whole, and the miss penalty is again 9 CCs.
- Which design is better and by how much?

Solution

- Miss penalty Separate Instruction (8KB) and Data (8KB) Cache

= (1)(.02)(9) + (0.35)(.1)(9) = .18 + .063 = 0.495

- Miss penalty unified 16KB Instruction & Data Cache

= (.03)(9) = 0.270

Unified Cache is the right design choice

Problem 8: DM Vs Associative Caches

- Briefly explain the advantages and disadvantages (in 4-5 sentences or a bulleted list) of using a direct mapped cache instead of an 8-way set associative cache
  - A direct mapped cache should have a faster hit time; there is only one block that data for a physical address can be mapped to
  - If there are successive reads to 2 separate addresses that map to the same cache block, then there will be a cache miss – degrading performance in DM.
  - In the 8-way set associative caches, a block can map to one of 8 blocks within a set. Thus, both above references would be hits and there would be no conflict misses.
  - A set associative cache will take a bit longer to search – could decrease clock rate.

Problem 9a:

- Assume you have a 2-way set associative cache.
  - Words are 4 bytes
  - Addresses are to the byte
  - Each block holds 512 bytes
  - There are 1024 blocks in the cache
- If you reference a 32-bit physical address – and the cache is initially empty how many data words are brought into the cache with this reference?
  - The entire block will be filled
  - If words are 4 bytes long and each block holds 512 bytes, there are $2^9/2^2$ words in the block i.e. there are 2^7 or 128 words in each block

Problem 9b

- Which set does the data that is brought in go to if the physical address F A B 1 2 3 8 9 (in hex) is supplied to the cache?
- We need to determine what the index bits are. From 9a, we know the offset is 9 bits (byte addressable) – so we will need to break up the hex address into binary:
- 1111 1010 1011 0001 0010 0011 1000 1001 [block offset in blue]
- 2-way set associative, so # Blocks/2 = # Index = 1024/2 = 512
- $512 = 2^9$ so, 9 bits [in red] used to identify cache index
- The value in red indicates the address maps to the 145^{th} set

Problem 10

- Consider an Intel P4 microprocessor with a 16 Kbyte unified L1 cache. The miss rate for this cache is 3% and the hit time is 2 CCs. The processor also has an 8 Mbyte, on-chip L2 cache. 95% of the time, data requests to the L2 cache are found. If data is not found in the L2 cache, a request is made to a 4 Gbyte main memory. The time to service a memory request is 100,000 CCs. On average, it takes 3.5 CCs to process a memory request. How often is data found in main memory?

Solution

- Average memory access time = Hit Time + (Miss Rate x Miss Penalty)
- Average memory access time = $\text{Hit Time}_{L1} + (\text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1})$
- $\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + (\text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$
- $\text{Miss Penalty}_{L2} = \text{Hit Time}_{Main} + (\text{Miss Rate}_{Main} \times \text{Miss Penalty}_{Main})$

$3.5 = 2 + 0.03\ (15 + 0.05\ (200 + X\ (100{,}000)))$

$3.5 = 2 + 0.03\ (15 + 10 + 5000X)$

$3.5 = 2 + 0.03\ (25 + 5000X)$

$3.5 = 2 + 0.75 + 150X$

$3.5 = 2.75 + 150X$

$0.75 = 150X$

$X = .005$

Problem 11

- A computer M2 has the following CPIs for instruction types A thru D, and a program P3 has the following mix of instructions:
  - $CPI_A = 1.7$ $CPI_B = 2.1$ $CPI_C = 2.7$ $CPI_D = 2.4$
  - A = 22% B = 29% C = 17% D = 32%
- Calculate average CPI, Execution time of P3 if IC = 22,311 and clock rate is 3.3 GHz:
  - $CPI_{avg} = \Sigma i = A \rightarrow D\ (CPI_i \times fraction_i)$
  - $= 1.7(0.22) + 2.1(0.29) + 2.7(0.17) + 2.4(0.32)$
  - $= 0.374 + 0.609 + 0.459 + 0.768 =$ **2.21**
  - = 22,311 instr x 2.21 cycles/instr / (3.3 GHz) = **14.9 us**

Problem 12a

- Given a computer M4 with clock rate = 3.1 GHz and a hardware accelerator that can make Type A instructions go 7 times faster
- If a program P4 has 29% Type A instructions and the remainder are Type B instructions, and its IC = 35,450 – will the accelerator make P4 run at least 2 times faster on M4 than it would without an accelerator?

Solution

- Given: (1) factor of improvement fI = 7 (2) fraction of system improved or enhanced fE = 0.29
- What is the limiting (maximum) speedup in this case?
- If runtime $\Delta t = 1$ before accelerator was used, then runtime after accelerator is used is given by
  - $\Delta t_A = (1 - fE) + (fE / fI) = 0.71 + (0.29 / 7) \sim 0.751$
  - So max. speedup is calculated as
  - = 1 / 0.751 ~ 1.33 times < 2 times

Problem 12b

- Use Amdahl's Law to determine how much faster the accelerator needs to be to make P4 run 5.5 times faster on M4 (with accelerator) than it did without an accelerator
- Suppose the accelerator was infinitely fast (as fast as it can go!!). Then, 0.71 of the program would not be speeded up, so the limiting speedup in this case would be
- = 1 / 0.71 ~ 1.408 times < 5.5 times

Misc Problems Prep Quiz 2

Computer Architecture (New York University)

ECE 6913, Spring 2021

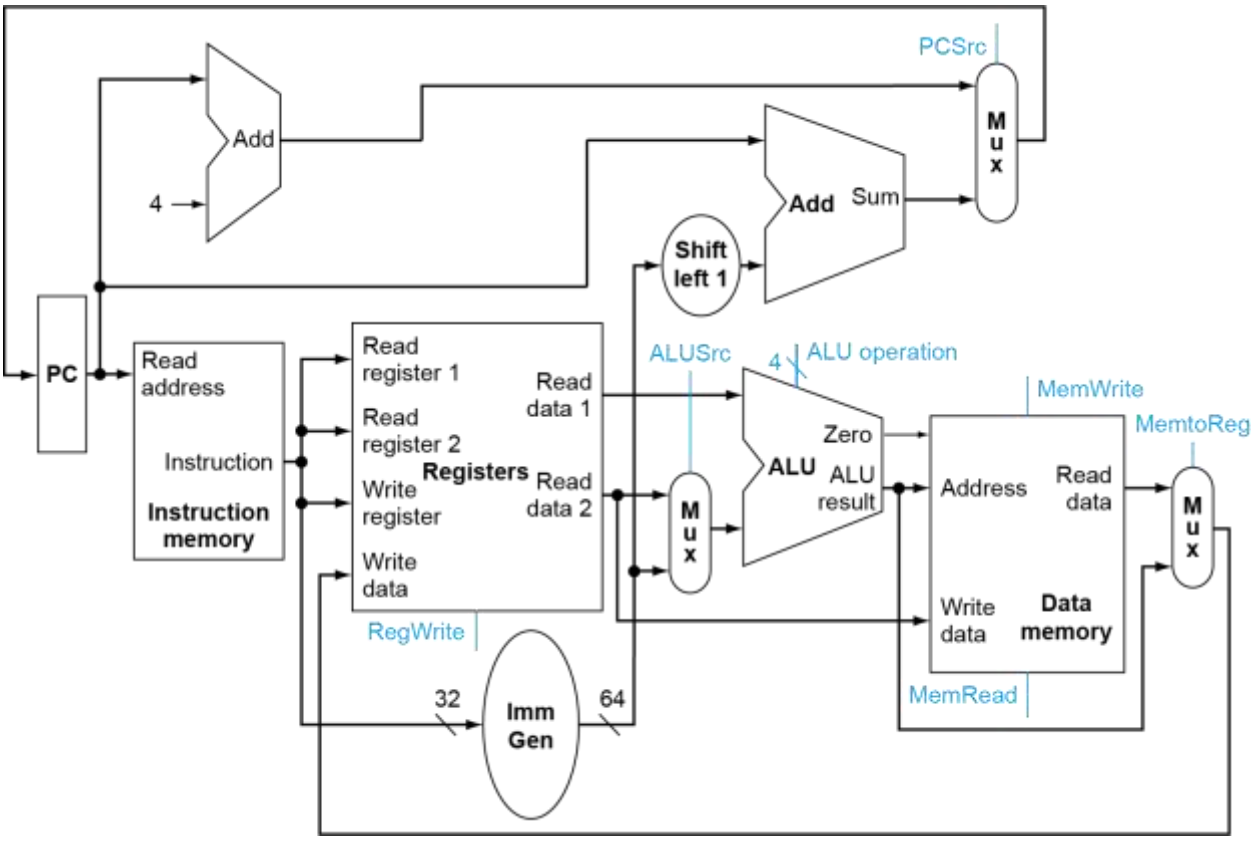
Misc Problems on topics covered on Quiz 2

1.1 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**

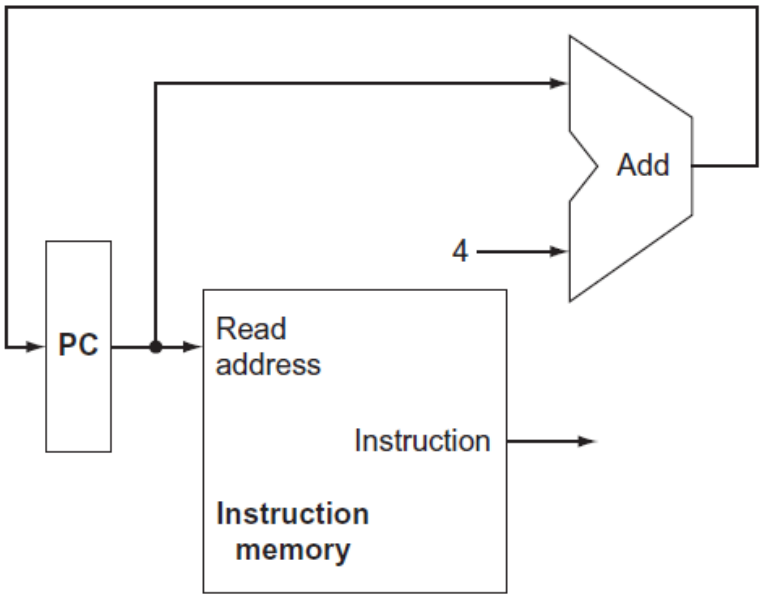
Note that the path to calculate the branch address is faster than the concurrent path above between Register File and PC Mux

(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**

1.2 If the only thing we need to do in a processor is fetch consecutive instructions using the figure below, what would the cycle time be?



Fetch of instructions would be dominated by the path from Program Counter to Instruction availability at the output of Instruction Memory. The Read delay of the Program Counter (30ps) following a clock edge added to the access delay of the Instruction memory (250ps) exceeds the concurrent execution of incrementing the current address by 4 and updating the Program Counter with it
30ps + 250ps = **280ps cycle time**

The other path to the adder followed by write into the Program Counter equals
30ps + 150ps + 20ps = 200ps
and is faster, so **the cycle time is limited by the slower 280 ps**

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.

3.1 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

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

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

3.4 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

3.5 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

3.6 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)*

5. 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.

6. 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

7. In general, cache access time is proportional to capacity. Assume that main memory accesses take 100 ns and that 40% of all instructions access data memory.

The following table shows data for L1 caches attached to each of two processors, P1 and P2

	L1 Size	L1 Miss Rate	L1 Hit Time
P1	4KiB	10%	0.75ns
P2	8KiB	7%	1.0ns

7.1 Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

P1: 1.33GHz

P2: 1GHz

7.2 What is the Average Memory Access Time for P1 and P2 (in cycles)?

P1: AMAT = 0.75ns + 100ns x 10% = 10.75ns

P2: AMAT = 1ns + 100ns x 7% = 8ns

7.3 Assuming a base CPI of 1.5 without any memory stalls, *what is the total CPI for P1 and P2? Which processor is faster?* (When we say a "base CPI of 1.5", we mean that 2 instructions complete in three cycles, unless either the instruction access or the data access causes a cache miss.)

Total CPI for P1= 1.5 + [(100x0.1/0.75ns)] x 0.4 = 6.83

Total CPI or P2 = 1.5 + [(100 x 0.07)/1.0] x 0.4 = 4.3

7.4 Consider the addition of an L2 cache to P1 (to presumably make up for its limited L1 cache capacity). Use the L1 cache capacities and hit times from the previous table when solving these problems. The L2 miss rate indicated is its local miss rate.

L2 Size	L2 Miss Rate	L2 Hit Time
4MiB	90%	10ns

What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

P1 AMAT: 0.75ns + 0.1 x (10ns + 0.9 x 100ns) = 10.75 ns

AMAT is the same

7.5 Assuming a base CPI of 1.5 without any memory stalls, *what is the total CPI for P1 with the addition of an L2 cache?*

CPI = 1.5 + 0.4 x [0.1(10 + 0.9 x 100)/0.75] = 6.83

8. 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

9. Assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows for the processor shown in the Figure below:

add	addi	not	beq	lw	sw
17%	18%	5%	25%	25%	10%

9.1 In what fraction of all cycles is the shift-left 2 unit used?

The shift left-2 unit is used only for branch instructions: 25%

9.2 In what fraction of all cycles is the input of the sign-extend circuit needed?

addi, beq, lw, sw: 18 + 25 + 25 + 10 = 78%

9.3 What is the utilization of the *second* input data port of the ALU?

The second input port of the ALU is used in all instructions except the unconditional branch instruction (jal for example) where the ALU does not need to determine if a branch is to be taken.

add, addi, not, beq, lw, sw = 100%