German University in Cairo - GUC
Faculty of Media Engineering and Technology- MET
Department of Computer Sciences
Assoc. Prof. Milad Ghantous

CSEN702-Microprocessors
Winter Semester 2022

Midterm Exam - Solution

Bar Code

Instructions: **Read Carefully Before Proceeding.**

1- Non-programmable calculators are allowed
2- Write your solutions in the space provided
3- The exam consists of **(5) questions**
4- This exam booklet contains **(XX) pages** including this page
5- Total time allowed for this exam is **(120) minutes**
6- When you are told that time is up, stop working on the test

Good Luck!

Question	1	2	3	4	5	∑
Possible Marks	10	16	14	17	20	77
Final Marks						

Question 1 (10 pts)

Mark each of the following statement with T or F. Ambiguous letters will not be considered.

#	Statement	T/F
1	"Early Restart" and "Critical word first" help in reducing miss rate.	F
2	Having multi-level caches will help in decreasing the miss rate of L1 cache.	F
3	"Way prediction" optimization in set associative caches targets the hit time.	T
4	A program with 25 loads and 25 stores, has a 1.5 memory reference per instruction, on average.	F
5	A CPU with higher power consumption means it has a higher energy consumption as well.	F
6	Turning off the clock for some unused components may help in reducing leakage (static power).	F
7	Per second, memory requests and memory accesses are equal.	F
8	When A[0] is requested, A being a single-precision float array, A[1] up to A[7] were brought to the cache as well. the cache block size is 64 bytes.	F
9	A write-thru cache is preferred in multi-core systems because data will be always coherent between caches and main memory.	T
10	When a CPU issues a read request for a word from a block, but that block's "dirty bit" was found to be 1 in the cache, it's a miss and the clean block needs to be requested from the main memory.	F

Question 2 (16 pts)
Short Answers

Part 3.1) Assume you had the choice to choose a clock rate for each piece of code you want to execute on your processor. For a specific low-priority code, you decided to use a slower rate to reduce energy and heat. (4 pts)
Will this work? Explain.
Solution:
Using the dynamic energy and power formulas, clearly the rate affects the power but not the energy as a slower rate will take the same code more time to execute, even with less power, but overall energy might remain the same or more.
However, the heat will be reduced due to the decrease in power.

Part 3.2) Consider this. Argue if it can be parallelized or not. Explain your answer. (4)

```
for(i=1; i<N; i+=2){
a[i] = a[i] + a[i-1]
```

A[1] = a[1]+a[0]
A[3]= a[3]+a[2]
A[5] = a[5]+a[4]
...
Iterations don't depend on each other.
So it can be parallelized.

Part 3.3) Consider a pipelined CPU that incurs 1 stall every 12 instructions due to data hazards. It employs an always taken prediction scheme for its branches. A Structural stall occurs 3 times every 100 instructions.
Given that the programs contain on average 20% branches that are executed in the decode stage and these branches were found to be taken 90% of the time, Calculate the CPI. (4)

Solution
Cpi = 1 + (1/12) + (3/100) + 0.2*0.1*1 =1.13

Part 3.4) Consider a processor that executes codes having 20% floating point instructions, 50% ALU and the remaining are memory instructions.
A Floating point instruction takes 8 cycles on average to complete, while any ALU/branch instruction takes 6 cycles on average, and a memory instruction takes 7 cycles.
We have the following measurements:

- The integer multiply instructions constitute 10% of all ALU instructions and each one requires, alone, 9 cycles.
- The floating multiply instructions constitute 50% of all floating point instructions, and each one, requires, alone, 14 cycles.

What speedup would we gain, against the current CPI, if we were able to decrease the CPI of the integer multiply to 5 cycles and, at the same time, the floating point multiply down to 11 cycles?

Original CPI = 0.2*8 + 0.5*6 + 0.3*7 = 6.7 cycles. (1 pt)

New CPI floating: old cpi-0.5*(14-11) = 8 – 0.5*3 = 6.5 cycles
New CPI integer: 6 – 0.1*(9-5) = 6-0.4= 5.6 cycles (1pt)

New overall CPI = 20%*(6.5)+ 50%*(5.6) + 30%*7 = 6.2 cycles (1 pt)

The speedup is 6.7/6.2 = 1.08 times. (1)

Question 3 (14 pts)

A) Consider the following code. A is a double-precision floating point array.

- Assume the cache is very large, starts empty, and a block size of 16 bytes.
- The array first element is stored at address 0 in memory.

```
for(i=0;i<=10;i++)
   { A[i] = A[10-i]+1; }
```

1) Consider a cache that uses the "no-write-allocate" strategy. How many read and write *misses* will this code incur. Show your work.

Each block contains 2 array cells (since it's 16 bytes so it can fit 2 array cells)
Block 0: A[0] and A[1], Block 1: A[2] and A[3] ... Block 4: A[8] and A[9]
Block 5: A[10] and the 8 bytes after it.

- A[0]= A[10]+1→
  1 read miss, we bring A[10] &the 8 bytes after it.
- 1 W_miss for A[0] but we don't allocate it

A[1]= A[9]+1 →
- 1 miss, we bring A[8] and A[9]
- 1 W-miss for A[1]

A[2]= A[8] +1 →
- HIT for A[8]
- 1 W_miss for A[2]

A[3]= A[7] +1 →
- 1 miss, we bring A[6] and A[7]
- 1 W_miss for A[3]

A[4]= A[6] +1 →
- HIT
- 1 W-miss for A[4]

A[5]= A[5] +1 →
- 1 miss, we bring A[4] and A[5] and then write to it.

A[6]= A[4] +1 →
- HIT

A[7]= A[3] +1 →
- 1 miss, we bring A[2] and A[3]

A[8]= A[2] +1 →
- HIT

A[9]= A[1] +1 →
- 1 miss, we bring A[0] and A[1]

A[10] = A[0] +1 →
- HIT

TOTAL = 6 read misses. And 5 write misses

2) Compute the number of read *misses and number of write misses* if the system uses “write-allocate” strategy. Also start from cache empty initially. Show your work.

```
A[0]= A[10] +1 →   1 miss, bring A[10] and the 8 bytes after it.
And another 1 miss for A[0], we bring A[0] and A[1]
A[1]= A[9] +1  →   1 miss, we bring A[8] and A[9]
A[2]= A[8] +1 →   1 miss for A[2], we bring A[2] and A[3]
A[3]= A[7] +1 →   1 miss, we bring A[6] and A[7]
A[4]= A[6] +1 →   1 miss for A[4], we bring A[4] and A[5]
A[5]= A[5] +1 →   HIT
A[6]= A[4] +1 →   HIT
A[7]= A[3] +1 →   HIT
A[8]= A[2] +1 →   HIT
A[9]= A[1] +1 →   HIT
A[10] = A[0] +1 → HIT
```

TOTAL = 3 read misses and 3 write misses.

3) Consider that the “no-write-allocate” strategy in part 1, uses a write-buffer of size 16 bytes, and gets transferred to main memory only when full. The buffer is not checked on read misses. Will this pose any data inconsistency issues in the above code or not? Explain.

```
A[0]= A[10] +1 →   send A[0] to buffer
A[1]= A[9] +1 →   send A[1] to buffer  → full, transfer to main

A[2]= A[8] +1 →   send A[2] to buffer
A[3]= A[7] +1 →   send A[3] to buffer → full, transfer to main
A[4]= A[6] +1 →   send A[4] to buffer
A[5]= A[5] +1 →   when we requested A[5], it's a miss, so we
request the block containing A[4] and A[5] from main memory but
A[4] is still in buffer! So problem yes!
```

Question 4 (17 pts)

Consider the following measurements for several cache designs:

I. Unified 64 KB cache: 40 misses per 1000 instructions.
II. 48 KB data cache: 30 misses per 1000 instructions.
III. 32 KB data cache: 25 misses per 1000 instructions.
IV. 16 KB instruction cache: 5 misses per 1000 instructions.
V. 32 KB instruction cache: 8 misses per 1000 instructions.

You have the option to choose one of the three following configurations for a processor:

- Configuration 1: The 64 KB unified cache.
- Configuration 2: The 48 KB data cache with 16 KB instruction cache.
- Configuration 3: The 32 KB data cache with 32 KB instruction cache.

- The hit time in all caches is 1 clock cycle.
- The miss penalty for all caches with sizes up to and including 16KB is 40 clock cycles, while the penalty for all larger sizes is 120 clock cycles.
- The unified cache is multi-ported and can serve an instruction and data hits in the same cycle.
- 30% of the instructions are loads and stores.

A) Compute the individual miss rates of all caches I through V. (5 pts)

Solution:

I. miss rate = 40/1000 / 1.3 = 0.03 or 3% (1.3 because 1 for instructions and then 0.3 for data total is 1.3)
II. miss rate= 30/1000 / 0.3 = 0.1 or 10%
III. miss rate = 25/1000/0.3 = 0.08 or 8%
IV. miss rate = 5/1000 /1 = 0.005 or 0.5%
V. miss rate = 8/1000 / 1 = 0.008 or 0.8%

B) Compute the overall miss rate of configuration 2 and the overall miss rate of configuration 3. (4 pts)

Before we do, we must compute the percentage of instruction access and data access.
For a total of 100 instructions for example, 100 are instruction access and 30 for data, total is 130 which means: (2 pts)
100/130 = 77% instruction access
30/130 = 23% data access

- Configuration 2: (1 pts)

0.77 x 0.005 + 0.23 x 0.1 = 0.026 or 2.6%

- Configuration 3: (1 pts)

0.77 x 0.008 + 0.23 x 0.08 = 0.0245 or 2.4 %

C) Compare the results of all configuration miss rates and discuss your choice. (2 pts)
It appears that configuration 3 has the lowest miss rate of all so it's the best so far. However this is not enough to take a decision, we must include the time factor.
D) Compute the average memory access time of all configurations and discuss the result. (6 pts)

- For configuration 1: avg mem access time = hit time + miss rate x miss penalty = 1+0.03x120 = 4.6 cycles.

- For configuration 2: 0.77 (1+0.005 x 40) + 0.23 (1+0.1x120) = 3.914 cycles

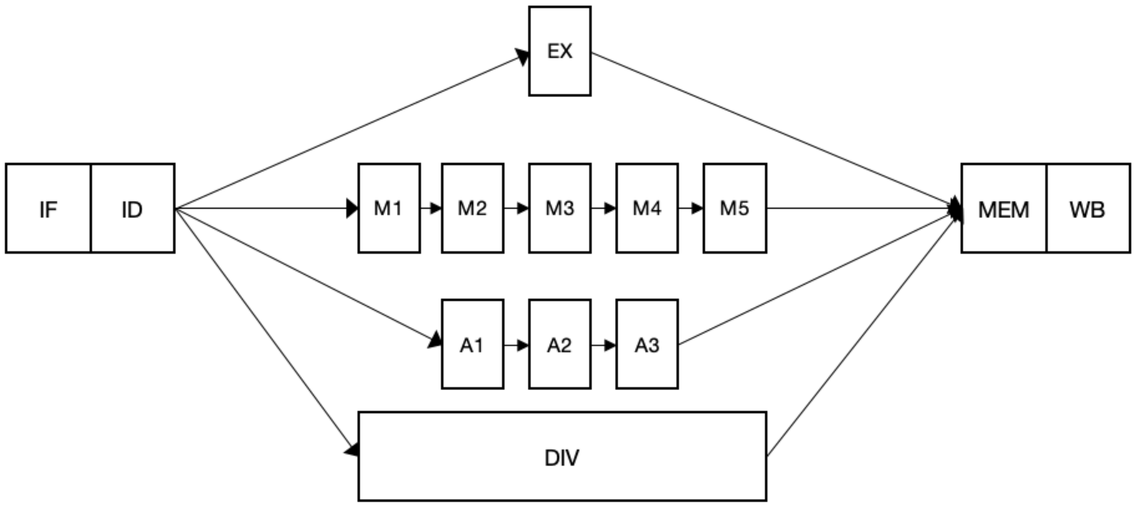
- For configuration 3:

1 + 0.0245 x 120 = 3.94 **<u>OR</u>** (0.77(1+0.008x120) + 0.23(1+0.08x120) = 3.94)
→ this makes configuration 2 the best since it uses a small cache for instructions with much lower penalty time which offers a great time saving.

Question 5 (20 pts)

Consider the following MIPS architecture where the floating point/integer multiplier is pipelined into 5 stages, the floating point adder is pipelined into 3 stages, and the divider is unpipelined and requires 10 cycles. The integer ALU (EX stage) takes 1 cycle and can handle integer arithmetic such as additions and subtractions, as well as branch outcome executions.

- Assume forwarding is active everywhere.
- Assume we have a hardware that detects if the data cache is used in the MEM stage or not, by an instruction, and if not, another instruction can use the cache in the same cycle. Also assume the same thing for register file during the WB stage.



A) Compute the number of stalls required between the following. Mark your answers in the below table. (6 pts)

	Result Producer	**Result Consumer**	**# of Stalls**
1	Load	Any arithmetic instruction	1
2	Multiply	Any arithmetic instruction	4
3	FP addition	Store	1
4	Integer arithmetic	Branch	0

Show your work here: (1.5 pts each)

1)

f	d	x	mem	W	
	f	d	STALL	x	mem

→ 1 stall

2)

f	d	M1	M2	M3	M4	M5	mem	wb	
	f	d	stall	stall	stall	stall	execute		

→ 4 stalls

3)

f	d	A1	A2	A3	mem	Wb			
	f	d	ex	STALL	mem	WB			

→ 1 stall

4)

f	d	ex	mem	wb					
	f	d	Ex	mem	wb				

→ 0 stalls

B) Using the latencies you found in part A), compute the number of cycles needed per 1 iteration of the following code. Show your work. (4 pts)

```
LOP,    L.S    F1,  0(R1)
        MUL.S  F2,  F1,  F30
        ADD.S  F3,  F2 , F2
        S.S    F3,  0(R1)
        DADDUI R1,  R1, -4
        BNE    R1,  R2, LOP
```

```
Solution:
LOP,    L.S    F1,  0(R1)
     stall
        MUL.S  F2,  F1,  F30
     Stall stall stall stall
        ADD.S  F3,  F2 , F2
     stall
        S.S    F3,  0(R1)
        DADDUI R1,  R1, -4
        BNE    R1,  R2, LOP

12 cycles.
```

C) Unroll 2 times and schedule the loop to the best of your ability, and compute the speedup you gained. (10 pts)

```
LOP     L.S    F1,  0(R1)
        L.S    F4,  -4(R1)
        MUL.S  F2,  F1,  F30
        MUL.S  F5,  F4,  F30
        DADDUI R1,  R1, -8
        Stall stall
        ADD.S  F3,  F2 , F2
        ADD.S  F6,  F5 , F5
        S.S    F3,  8(R1)
        S.S    F6,  4(R1)
        BNE    R1,  R2, LOP
```

Total = 12 cycles so per iteration = 6! (50% reduction)