Name: _____

Student #: _____

King Fahd University of Petroleum and Minerals
College of Computing and Mathematics
Department of Computer Engineering

COE 301 – Computer Organization (T212)
ICS 233 – Computer Architecture & Assembly Language (T212)

Final Exam – SOLUTION

Date & Time: Wednesday May 25, 2022 (12:30 PM – 03:00 PM)

- **This is a CLOSED books, CLOSED notes exam.**
- **Answer ALL problems.**
- **Show all your work. NO partial credit will be given if work is not shown.**
- **Use of mobile phones, smart phones/watches, tablets is prohibited.**

Problem	Mark	Score
1	5.0	
2	5.0	
3	11.0	
4	14.0	
5	5.0	
6	10.0	
7	6.0	
8	13.0	
9	7.0	
Total	76.0	

Select your section number:

- ☐ **COE 301 – Section 1** (UTR 08:00 – Dr. Ayaz Khan)
- ☐ **COE 301 – Section 2** (UTR 11:00 – Dr. Marwan Abu-Amara)
- ☐ **ICS 233 – Section 1** (UTR 11:00 – Dr. Ayaz Khan)
- ☐ **ICS 233 – Section 2** (UTR 10:00 – Dr. Ayaz Khan)

Problem 1 (5 points): **Floating-Point Addition**

Given that A and B are single-precision floats, compute **A - B**. Use rounding to negative infinity. Perform the operation using guard, round and sticky bits.

A = +1.100 1001 1100 0001 0100 1111 × 2^{-4}
B = +1.011 0110 0100 1100 0001 0001 × 2^{-1}

+ -	1.100 1001 1100 0001 0100 1111 1.011 0110 0100 1100 0001 0001	× 2^{-4} × 2^{-1}
+ -	0.001 1001 0011 1000 0010 1001 111 1.011 0110 0100 1100 0001 0001	× 2^{-1} × 2^{-1}
	00.001 1001 0011 1000 0010 1001 111 10.100 1001 1011 0011 1110 1111	× 2^{-1} × 2^{-1}
	10.110 0010 1110 1100 0001 1000 111	× 2^{-1}
-	01.001 1101 0001 0011 1110 0111 001	× 2^{-1}
-	1.001 1101 0001 0011 1110 1000	× 2^{-1}

Problem 2 (5 points): **Floating-Point Multiplication**

Given that A and B are single-precision floats, compute **A** × **B**. Use rounding to nearest even. Perform the operation using guard, round and sticky bits.

A = +1.100 1001 1100 0001 0100 1111 × 2^{-4}
B = +1.010 0000 0000 0000 0001 0000 × 2^{-1}

+ 1.100 1001 1100 0001 0100 1111 × 2^{-4}
+ 1.010 0000 0000 0000 0001 0000 × 2^{-1}

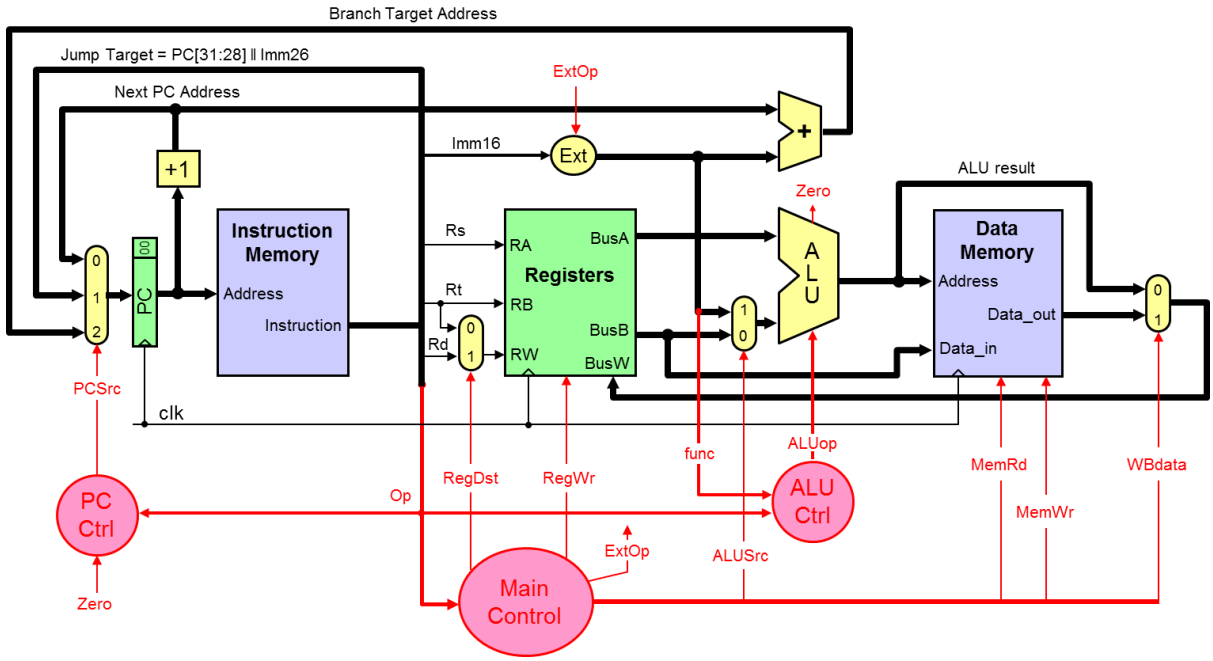
Result exponent = (-4)+(-1) = -5

```
X                        1.10010011100000101001111
                         1.01000000000000000010000
-------------------------------------------------------
                       1100100111000001010011111
    1100100111000001010011111
1.10010011100000101001111
-------------------------------------------------------
1.11111000011000110111011111110000010100111110000
```

+ 1.11111000011000110111011 11.. × 2^{-5}
+ 1.11111000011000110111100 × 2^{-5}

Problem 3 (11 points): Single-Cycle Processor

The single-cycle datapath and control of a MIPS-like processor is shown below. However, it is limited to the implementation of few instructions.



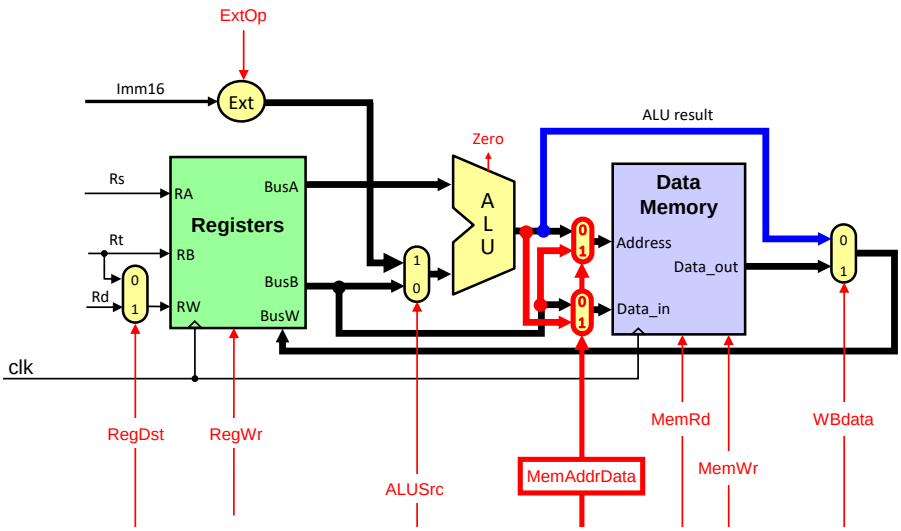
1) **(3 points)** Consider the execution of **`BNE Rs, Rt, label`** with the **`Zero`** flag equal to **`0`**. Fill out the control signal values for the **`BNE`** instruction in the table shown below.

Instruction	RegDst	RegWr	ExtOp	ALUSrc	MemRd	MemWr	WBdata	PCSrc
`BNE Rs, Rt, label (Zero = 0)`	`X`	`0`	`1`	`0`	`0`	`0`	`X`	`2`

2) Consider adding a new instruction, **`AISW`**, to the above datapath. The **`AISW`** instruction stands for "Add Immediate and Store Word" and is an I-type that has a unique opcode. The addition is **signed** addition.

Instruction	Format	Meaning
`AISW Rt, Rs, Imm16`	`Op, Rs, Rt, Imm16`	`Mem[Reg[Rt]] = Reg[Rs] + Imm16`

(i) **(5 points)** Redraw the **necessary/minimal** changes to the above datapath to implement **`AISW`**, and show any needed new control signal(s). The modified datapath should still support the execution of all previously implemented instructions.
Draw only the modified parts and explain why they are needed.



(ii) **(3 points)** Identify any new control signal needed to implement **`AISW`**. Fill out both the existing and the new control signals values for the **`AISW`** instruction in the table shown below. Note: Use as many of the "**New Control Signal(s)**" columns as needed.

Instruction	Existing Control Signals								New Control Signal(s)		
	RegDst	RegWr	ExtOp	ALUSrc	MemRd	MemWr	WBdata	PCSrc	MemAddrData		
`AISW Rt, Rs, Imm16`	X	0	1	1	0	1	X	0	1		

Problem 4 (14 points): Performance of a MIPS program

(1) Consider a 5-stage MIPS processor with the following delay of each stage:

Instruction memory access time = 400 ps
Instruction Decode and Register read = 200 ps
Data memory access time = 400 ps
Register write = 200 ps
ALU delay = 100 ps

Ignore the delays of other components like multiplexers, wires, etc.

Assume the following instruction mix:

Instruction Class	ALU	Load	Store	Taken Branches	Not Taken Branches	Jump
Frequency	40%	15%	10%	25%	5%	5%

a) **(6 points)** Compute the delay for each instruction class for the single-cycle processor.

Instruction Class	**Instruction Memory**	**Decode and Register Read**	**ALU**	**Data Memory**	**Write Back**	**Total Delay**
ALU	**400**	**200**	**100**		**200**	**900**
Load	**400**	**200**	**100**	**400**	**200**	**1300**
Store	**400**	**200**	**100**	**400**		**1100**
Taken Branch	**400**	**200**	**100**			**700**
Not Taken Branch	**400**	**200**	**100**			**700**
Jump	**400**	**200**				**600**

b) **(1 point)** Compute the clock cycle for the single-cycle processor

Clock Cycle = max (900,1300,1100,700,700,600) = 1300 ps

c) **(1 point)** Compute the clock cycle for the multi-cycle processor, given that each stage should be completed in one clock cycle.

Clock Cycle = max (400,200,400,200,100) = 400 ps

d) **(1 point)** Compute the average CPI for the multi-cycle processor.

Average CPI = 0.4 × 4 + 0.15 × 5 + 0.1 × 4 + 0.25 × 3 + 0.05 × 3 + 0.05 × 2 = 3.75

e) **(1 point)** Determine **quantitatively** if there is a speedup when using the multi-cycle processor.

Speedup = (1 × 1300) / (3.75 × 400) = 0.867 (multicycle is slower)

f) **(1 point)** Calculate the average CPI of a pipelined processor that has a base CPI = 1. The pipeline stalls 1 cycle after each jump, and 2 cycles after each taken branch.

Pipelined CPI = 1 + 0.05 × 1 + 0.25 × 2 = 1.55

g) **(1 point)** Determine **quantitatively** if there is a speedup when using the pipelined processor over the single-cycle processor.

Speedup = (1 × 1300) / (1.55 × 400) = 2.097 (pipeline is faster)

---

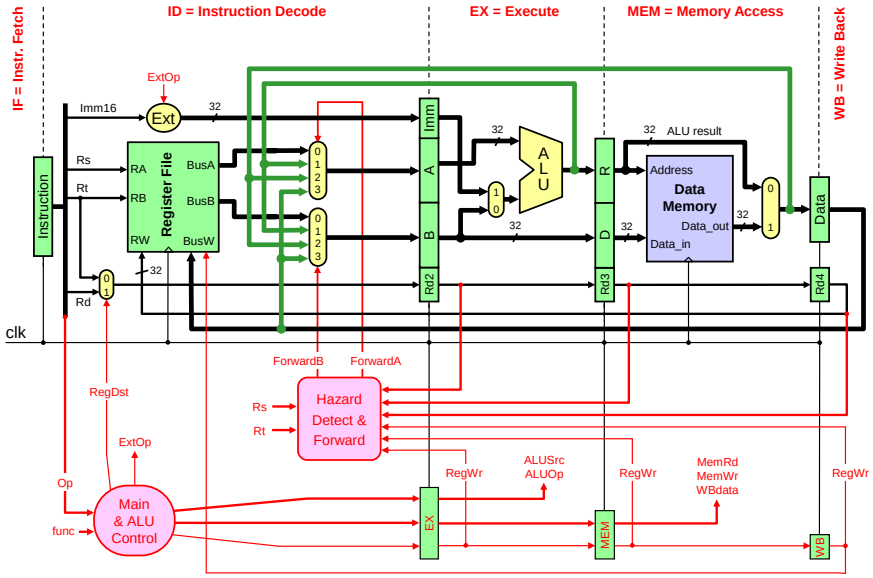
(2) **(2 points)** We want to improve the overall speedup in a program by a factor of 1.25. Assume that the multiplication instruction accounts for 40% of the instructions. Compute the amount of speedup needed in executing the multiplication instruction to achieve an overall speedup of 1.25 for this program.

$$Overall\ Speedup = 1.25 = \frac{1}{\frac{f}{s}+(1-f)} = \frac{1}{\frac{0.4}{s}+0.6} \quad \Rightarrow \quad \boldsymbol{s = 2}$$

⇒ Multiplication instruction execution must be sped up by a factor of 2 to achieve the desired overall speedup of 1.25. That is, the new multiplication instructions execution time should be 50% shorter than the old multiplication instructions execution time.

Problem 5 (5 points): Pipelined Processor Design

Consider the 5-stage pipelined processor datapath and control shown below. However, it is limited to the implementation of few instructions.



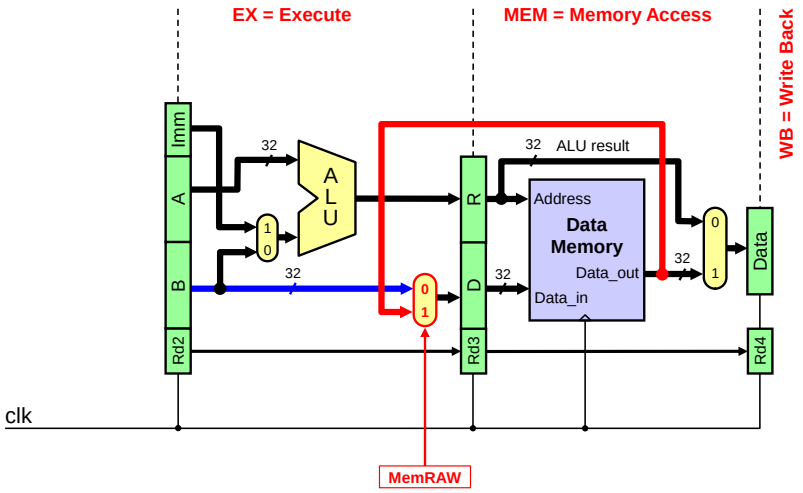
Consider the following partial code:

```
...
LW    Rt1,imm(Rs1)
SW    Rt2,imm(Rs2)
...
```

Consider the scenario when **`Rt1`** and **`Rs2`** are two **different** registers while **`Rt1`** is the same register as **`Rt2`**, then the execution of the two instructions (**`LW`** followed by **`SW`**) causes a **`RAW`** data hazard that requires stall cycles.

Modify the 5-stage pipelined processor by using proper **forwarding** to eliminate this specific data hazard and the need for any stall cycles. Redraw the **necessary/minimal** changes to the above datapath, and show in which pipeline stage(s) the changes are needed along with showing all newly needed control signal(s). The modified datapath should still support the execution of all previously implemented instructions.

Draw only the modified parts and explain why they are needed.



Problem 6 (10 points): Pipelined CPU

A hypothetical processor has a pipeline consisting of 6 stages as shown in the table below. The first row in the table below shows the pipeline stage number, second row gives the name of each stage, and third row gives the delay of each stage in nano-seconds. The name of each stage describes the task performed by it. Each stage takes 1 clock cycle.

1	2	3	4	5	6
Instruction Fetch (IF)	Instruction Decode (ID)	Instruction Execute 1 (EX1)	Instruction Execute 2 (EX2)	Memory Access (MEM)	Register Write back (WB)
1ns	1.5ns	1.2ns	1.2 ns	2.5ns	1.1ns

The hypothetical 6-stage pipelined processor also implements a set of instructions to perform composite operations (two operations on the same operands such that **`EX1 result = A op B`** and **`EX2 result = EX1 result op B`**). The respective result (depending on the instruction) will then be forwarded to the next stage.

a) (1 point) How much time (in ns) is required to fetch and complete one instruction if the six stages were **not pipelined**?

Answer: 1+1.5+1.2+1.2+2.5+1.1 = 8.5 ns

b) (1 point) What is the pipeline clock cycle (in ns) if **all the six stages are pipelined**?

Answer: Clock Cycle = 2.5 ns (longest delay is for the MEM stage)

c) (1 point) How much time (in ns) is required to fetch and complete one instruction on the **six-stage pipeline**?

Answer: 2.5 × 6 = 15 ns

d) (1 point) How many clock cycles are required to fetch and complete **17 instructions** on this six-stage pipeline? Assume that the pipeline is **initially empty**, and no stall cycles occur during the execution of all 17 instructions.

Answer: 6 + 16 = 22 cycles

The following are **two dependent instructions** that are executed on the 6-stage pipeline:

```
LW    R5 = 8(R4)          ; R5 = mem[R4+8]
ADD   R2 = R5, R7         ; R2 = R5+R7
```

e) (3 points) Assuming **no forwarding** circuitry is implemented in the 6-stage pipeline, fill the timing diagram for the **LW** and **ADD** instructions, showing **S** for stall cycles, and a **–** for an unused pipeline stage. How many **stall cycles** will occur in the pipeline during the execution of above two instructions?

	1	2	3	4	5	6	7	8	9	10	11	12
LW	IF	ID	EX1	EX2/-	MEM	WB						
ADD		IF	S	S	S	S	ID	EX1	EX2/-	-	WB	

4 stall cycles.

f) (3 points) Assuming **full-forwarding** circuitry is implemented in the 6-stage pipeline, fill the timing diagram for the **LW** and **ADD** instructions. Show **S** for stall cycles, a **–** for an unused pipeline stage, and **draw an arrow** for forwarding data between stages. How many **stall cycles** will occur in the pipeline during the execution of the above two instructions?

	1	2	3	4	5	6	7	8	9	10	11
LW	IF	ID	EX1	EX2/-	MEM	WB					
ADD		IF	ID	S	S	EX1	EX2/-	-	WB		

2 stall cycles.

Problem 7 (6 points): Branch Predictions

We have a program core consisting of three conditional branches. The program core will be executed millions of times. Below are the outcomes of each branch for one execution of the program core (**T** for taken and **N** for not taken).

Branch 1: **T-T-T-T-T**
Branch 2: **T-T-T-N-N-N**
Branch 3: **T-T-T-N-T-T-T-N-T**

Assume that the behavior of each branch remains the same for each program core execution. For dynamic branch prediction schemes, assume that each branch has its own prediction buffer and each buffer is initialized to the same state before each execution. List the correct/wrong predictions and the accuracies for each of the following branch prediction schemes in the tables below: *(Hint: Accuracy = Total Correct Predictions / Total Branches)*

a) Predict branch "Not Taken"
b) 1-bit predictor, initialized to **predict taken**
c) 2-bit predictor, initialized to **weakly predict taken**

a) (2 points) Predict branch "Not Taken"

	Correct Predictions	**Wrong Predictions**
Branch 1	0	5
Branch 2	3	3
Branch 3	2	7

Accuracy = **5/20 = 25** **%**

b) (2 points) 1-bit predictor, initialized to **predict taken**

	Correct Predictions	**Wrong Predictions**
Branch 1	5	0
Branch 2	5	1
Branch 3	5	4

Accuracy = **14/20 = 70** **%**

c) (2 points) 2-bit predictor, initialized to **weakly predict taken**

	Correct Predictions	**Wrong Predictions**
Branch 1	5	0
Branch 2	4	2
Branch 3	7	2

Accuracy = **14/20 = 70** **%**

Problem 8 (13 points): **Cache Memory**

(1) Assume that a computer system uses a 16-bit address and a cache with 4K byte data size (not including tag and valid bits) and a 32-byte block size.

a) (3 points) Assume the cache is organized as **direct-mapped**. Compute the number of bits in the offset, index and tag fields.

Block offset bits	**= $\log_2(32)$**	**= 5 bits**
Index bits	**= $\log_2(4K/32)$**	**= 7 bits**
Tag bits	**= 16 – 5 – 7**	**= 4 bits**

b) (2 points) Assume the cache is organized as **8-way set associative**. Compute the number of bits in the offset, index and tag fields.

Block offset bits	**= $\log_2(32)$**	**= 5 bits**
Index bits	**= $\log_2(4K/8 \times 32)$**	**= 4 bits**
Tag bits	**= 16 – 5 – 4**	**= 7 bits**

---

(2) **(8 points)** Assume a **2-way set-associative** cache where each set has two blocks only (way 0 and way 1). Each cache block contains 16 bytes of data, and the cache is initially empty. The cache uses 16-bit memory addresses divided into 4 bits of offset, 4 bits of index, and 8 bits of tag. The following sequence of 16-bit addresses are sent to the cache: **`0x321C, 0x329D, 0x521B, 0x3215, 0x5298`**. For each addresses, identify the **set index**, **tag (hexadecimal)**, **way** (0 or 1), and indicate whether there is a **cache hit or miss** by filling the table shown below. Use the FIFO replacement policy.

Memory Address	Set Index	Tag	Way (0 or 1)	Hit or Miss
0x321C	1	0x32	0	miss
0x329D	9	0x32	0	miss
0x521B	1	0x52	1	miss
0x3215	1	0x32	0	hit
0x3915	1	0x39	0	miss

Problem 9 (7 points): Cache Memory Performance

A processor runs at 3.0 GHz and has a CPI = 1.25 for a perfect cache (i.e., without including the stall cycles due to cache misses). Assume that load and store instructions are 20% of the instructions. The processor has an I-cache with a 10% miss rate and a D-cache with 5% miss rate. The hit time is 1 clock cycle for both caches. Assume that the miss penalty is 30 ns for both caches. For each question, show your work and calculations.

(a) **(2 points)** Compute the combined misses per instruction in the I-cache and D-cache.

Combined misses per instruction = 0.1 + 0.2 × 0.05 = 0.11

(b) **(2 points)** Compute the number of stall cycles per instruction and the overall CPI.

Miss penalty = 30 ns = 90 cycles
Memory stall cycles per instruction = 0.11 × 90 cycles = 9.9 cycles
Overall CPI = 1.25 + 9.9 = 11.15

(c) **(2 points)** Compute the memory access per instruction and the combined miss rate for both the I-cache and D-cache.

Memory Access per Instruction = 1 (instruction fetch) + 0.20 (load and store) = 1.20
Combined Miss Rate = Combined Misses per Instruction / Memory Access per Instruction
Combined Miss Rate = 0.11 / 1.20 = 0.0917 = 9.17%

(d) **(1 point)** Compute the average memory access time (AMAT) in clock cycles.

AMAT = Hit Time + Miss Rate × Miss Penalty = 1 cycle + 0.0917 × 90 cycles = 9.253 cycles