Family Name: SOLUTIONS First Name: Student ID:

Final Exam

Digital Design and Computer Architecture (252-0028-00L)

ETH Zürich, Spring 2022

Prof. Onur Mutlu

Problem 1 (40 Points):	Boolean Logic Circuits	
Problem 2 (50 Points):	Finite State Machines	
Problem 3 (30 Points):	ISA vs. Microarchitecture	
Problem 4 (60 Points):	Verilog	
Problem 5 (30 Points):	Memory Potpourri	
Problem 6 (70 Points):	Performance Evaluation	
Problem 7 (70 Points):	Pipelining	
Problem 8 (60 Points):	Tomasulo's Algorithm	
Problem 9 (45 Points):	GPUs and SIMD	
Problem 10 (70 Points):	Branch Prediction	
Problem 11 (BONUS: 50 Points):	Prefetching	
Problem 12 (BONUS: 70 Points):	Caches	
Total (645 (525+120 bonus) Points):		

Examination Rules:

1. Written exam, 180 minutes in total.
2. **No books, no calculators, no computers or communication devices**. 3 double-sided (or 6 one-sided) A4 sheets of handwritten notes are allowed.
3. Write all your answers on this document; space is reserved for your answers after each question.
4. You are provided with scratchpad sheets. Do not answer questions on them. **We will not collect them**.
5. Clearly indicate your final answer for each problem. Answers will only be evaluated if they are readable.
6. Put your Student ID card visible on the desk during the exam.
7. If you feel disturbed, immediately call an assistant.
8. Write with a black or blue pen (no pencil, no green, red or any other color).
9. Show all your work. For some questions, you may get partial credit even if the end result is wrong due to a calculation mistake. If you make assumptions, state your assumptions clearly and precisely.
10. Please write your initials at the top of every page.

Tips:

- **Be cognizant of time.** Do not spend too much time on one question.
- **Be concise.** You may be penalized for verbosity.
- **Show work when needed.** You will receive partial credit at the instructors' discretion.
- **Write legibly.** Show your final answer.

This page intentionally left blank

1 Boolean Logic Circuits [40 points]

During your job interview, you are asked to design a combinational circuit with a four-bit input, $\{A, B, C, D\}$ (A is the most significant bit and D is the least significant bit), and two 1-bit outputs, *Factorial* and *Div*4. The value of each output is determined as follows:

- The output *Factorial* is 1 only when the input 4-bit number is a product of ALL positive integers that are less than or equal to the input number.
- The output *Div*4 is 1 only when the input 4-bit number is divisible by 4.
- Otherwise, the corresponding outputs are zero.

Please answer the following four questions.

(a) [10 points] Fill in the missing entries in the truth table below for the combinational circuit you are designing.

Inputs				**Outputs**	
A	B	C	D	*Factorial*	*Div*4
0	0	0	0	1	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

(b) [10 points] Express the output $Div4$ as the simplest *sum of products* representation. Show your work step-by-step.

$Div4 = (\overline{C}.\overline{D})$

Explanation:

$Div4 = (\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}) + (\overline{A} \cdot B \cdot \overline{C} \cdot \overline{D}) + (A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}) + (A \cdot B \cdot \overline{C} \cdot \overline{D})$
$Div4 = (\overline{A} \cdot \overline{C} \cdot \overline{D})(\overline{B} + B) + (A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}) + (A \cdot B \cdot \overline{C} \cdot \overline{D})$
$Div4 = (\overline{A} \cdot \overline{C} \cdot \overline{D}) + (A \cdot \overline{B} \cdot \overline{C} \cdot \overline{D}) + (A \cdot B \cdot \overline{C} \cdot \overline{D})$
$Div4 = (\overline{C} \cdot \overline{D})(A \cdot \overline{B} + \overline{A}) + (A \cdot B \cdot \overline{C} \cdot \overline{D})$
$Div4 = (\overline{C} \cdot \overline{D} \cdot \overline{B}) + (\overline{C} \cdot \overline{D} \cdot \overline{A}) + (A \cdot B \cdot \overline{C} \cdot \overline{D})$
$Div4 = (\overline{C} \cdot \overline{D} \cdot \overline{A}) + (\overline{C} \cdot \overline{D})(A \cdot B + \overline{B})$
$Div4 = (\overline{C} \cdot \overline{D})(\overline{A} + A) + (\overline{C} \cdot \overline{D} \cdot \overline{B})$
$Div4 = (\overline{C} \cdot \overline{D}) + (\overline{C} \cdot \overline{D} \cdot \overline{B})$
$Div4 = (\overline{C} \cdot \overline{D})$

(c) [20 points] Find the simplest representation of the *Factorial* output by using *only* NOR gates. Show your work step-by-step.

$Factorial = \overline{B + A + \overline{\overline{C + C} + \overline{D + D}}}$
Explanation:

$\overline{Factorial} = (A + B + C + D) \cdot (A + B + C + \overline{D}) \cdot (A + B + \overline{C} + D)$

$\overline{Factorial} = (A \cdot (A + B + C + \overline{D}) + B \cdot (A + B + C + \overline{D}) + C \cdot (A + B + C + \overline{D}) + D \cdot (A + B + C + \overline{D})) \cdot (A + B + \overline{C} + D)$

$\overline{Factorial} = (A + AB + AC + A\overline{D} + AB + B + BC + B\overline{D} + AC + CB + C + C\overline{D} + AD + BD + CD + D\overline{D}) \cdot (A + B + \overline{C} + D)$

$\overline{Factorial} = (A + B + C) \cdot (A + B + \overline{C} + D)$

$\overline{Factorial} = (A + AB + A\overline{C} + AD) + (AB + B + B\overline{C} + BD) + (AC + BC + CD)$

$Factorial = \overline{B + A + CD}$

$Factorial = \overline{B + A + \overline{\overline{CD}}}$

$Factorial = \overline{B + A + \overline{\overline{C} + \overline{D}}}$

$Factorial = \overline{B + A + \overline{\overline{C + C} + \overline{D + D}}}$

2 Finite State Machines [50 points]

The Polybahn from Central to Polyterasse has broken down! To fix it you need to design a finite state machine that controls the Polybahn's two doors A and B.

The Polybahn should operate as follows:

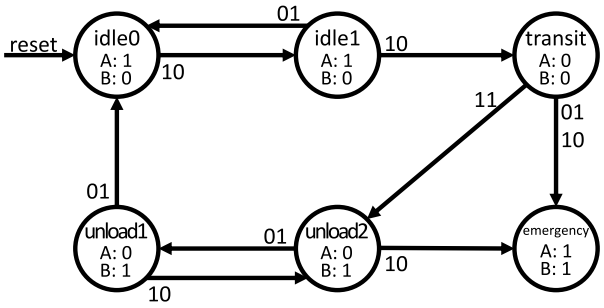
- Initially the Polybahn is empty and **idle**, and passengers can enter through door A.
- The Polybahn is full when it carries 2 passengers.
- When it is full and idle, the Polybahn goes into **transit** to the other station with both doors closed.
- After reaching the station, the Polybahn will **unload** all passengers through door B, while door A is still closed.
- After the last passenger has exited the Polybahn, door B closes and the Polybahn becomes **idle**.
- Should (1) a passenger fall out of the Polybahn during **transit**, or (2) the Polybahn become overfull (≥ 3 passengers) at any point, it stops in **emergency** mode, where it opens all doors and remains (unless reset to the initial **idle** state).

The FSM receives two input bits, with the following meaning:

Input	Meaning
00	no change
01	exactly one passenger left
10	exactly one passenger entered
11	the Polybahn arrived in a station

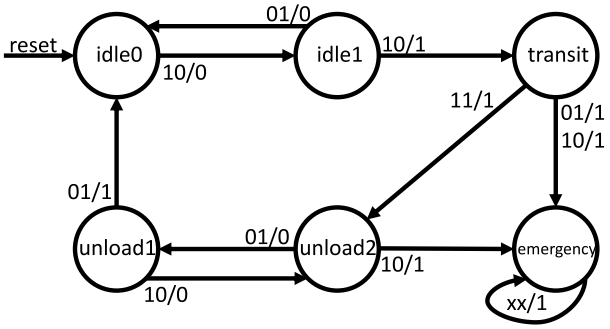
The FSM produces two output bits: The first bit, A, holds door A open when it is 1. The second bit, B, holds door B open when it is 1.

(a) [25 points] Complete the Moore-type FSM below by (1) drawing the transition edges between the states (including reset), (2) specifying the edges' respective input bits, and (3) specifying the output bits of each state. Any input for which no outgoing edge is specified is assumed as a self loop. The 6 given states are sufficient, do not draw additional states.
Note: Passengers sometimes slip in or out through incorrect doors, or even through closed doors. Your FSM must correctly handle such cases.



(b) [25 points] You need to design a second FSM that controls only the bell. The bell should ring (i.e., the output bit is 1) whenever the Moore-type FSM you designed for part (a) opens or closes a door, and is constantly ringing in the case of an **emergency**.

Complete the Mealy-type FSM below by (1) drawing the transition edges between the states (including reset), and (2) specifying the edges' respective input and output bits. Label edges in the following format: [input0][input1]/[bell], e.g., 10/0 = passenger entered, bell off. Any input for which no outgoing edge is specified is assumed as a self loop with output 0.



3 ISA vs. Microarchitecture [30 points]

Circle whether each of the following is an aspect of the ISA or the microarchitecture.

Note: we will subtract 1 point for each ***incorrect*** *answer and award 0 points for unanswered questions.*

1. [2 points] Two-level global branch prediction.

   1. ISA [2. Microarchitecture]

2. [2 points] Location of the bits that identify the destination register in an ADD instruction.

   [1. ISA] 2. Microarchitecture

3. [2 points] Number of instructions fetched per cycle.

   1. ISA [2. Microarchitecture]

4. [2 points] Ratio of the number of floating-point to integer general-purpose registers.

   [1. ISA] 2. Microarchitecture

5. [2 points] Number of integer arithmetic and logic units (ALUs).

   1. ISA [2. Microarchitecture]

6. [2 points] Instruction issue width of the processor core's pipeline.

   1. ISA [2. Microarchitecture]

7. [2 points] SIMD support.

   [1. ISA] 2. Microarchitecture

8. [2 points] L3 cache replacement policy.

   1. ISA [2. Microarchitecture]

9. [2 points] Width of the data bus to memory.

   1. ISA [2. Microarchitecture]

10. [2 points] The size of the addressable memory by programs.

    [1. ISA] 2. Microarchitecture

11. [2 points] Number of cycles it takes to execute an ADD instruction.

    1. ISA [2. Microarchitecture]

12. [2 points] Ability to choose a specific cache replacement policy using operating system code.

    [1. ISA] 2. Microarchitecture

13. [2 points] Number of read/write ports in the physical register file.

    1. ISA [2. Microarchitecture]

14. [2 points] Function of each bit in a programmable prefetcher's configuration register.

    [1. ISA] 2. Microarchitecture

15. [2 points] Number of L3 cache banks.

    1. ISA [2. Microarchitecture]

4 Verilog [60 points]

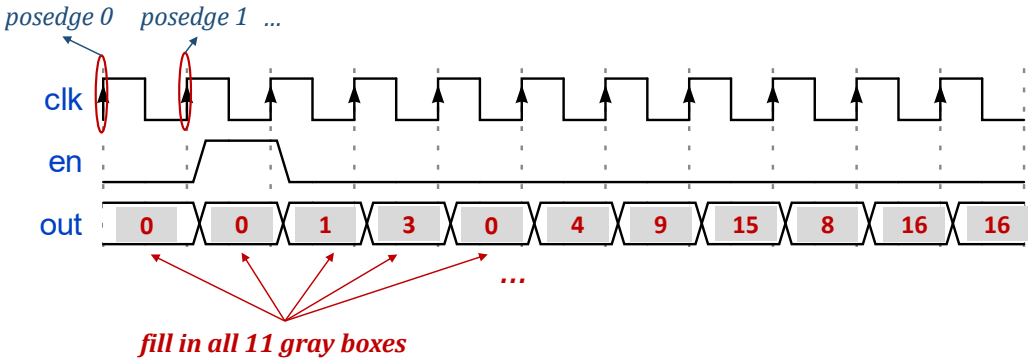
4.1 What Does This Code Do? [30 points]

Analyze the following Verilog module and answer the question.

```
1  module mystery_module (clk, en, in1, in2, out);
2
3    input clk, en;
4    input[63:0] in1;
5    input[7:0] in2;
6    output reg[10:0] out = 0;
7
8    reg[2:0] var1 = 0;
9
10   always @(posedge clk) begin
11     out <= out;
12     if (en & (var1 == 0)) begin
13       var1 <= var1 + 1'b1;
14
15       if (in2[var1])
16         out <= 11'd0 + in1[var1*8 +: 8];
17       else
18         out <= 11'd0 - in1[var1*8 +: 8];
19     end
20
21     if (var1 != 0) begin
22       var1 <= var1 + 1'b1;
23
24       if (in2[var1])
25         out <= out + in1[var1*8 +: 8];
26       else
27         out <= out - in1[var1*8 +: 8];
28     end
29   end
30
31 endmodule
```

Assume that the inputs `in1` and `in2` *always* have the following values:
`in1 = 64'h0807060504030201`
`in2 = 8'b10111011`

What **unsigned decimal** values does the `out` signal get in the following waveform diagram? Fill in the gray boxes with an `out` value for each `clk` cycle. Briefly explain your answer.



Brief explanation (to help us award you partial credit):

Explanation. Once `en` becomes `1`, the `mystery_module` begins processing the inputs `in1` and `in2`.
The output signal `out` is initially `0` (line 6).
`var1` is used to index both `in1` and `in2`. `in1` is indexed in 8-bit data chunks and every bit in `in2` is indexed separalety. `var1` is initially `0` and indexes both inputs starting from their least-significant bits.
When `var1` is `0`, the `mystery_module` adds (if `in2[0] = 1`) or subtracts (if `in2[0] = 0`) the least significant 8 bits of `in1` (i.e., `in1[7:0]`) to/from the 11-bit `out` register. In consecutive cycles, `var1` gets incremented by 1 and the module adds or subtracts the other 8-bit data chunks in `in1` to/from `out`.

For the given values of `in1` and `in2`, `out` gets the following values:
cycle 0: `out = 0` (`en = 0` so `out` remains `0`)
cycle 1: `out = 0` (`en = 1` but `out` will be updated for the next cycle (after `posedge 2`))
cycle 2: `out = 0 + 1 = 1`
cycle 3: `out = 1 + 2 = 3`
cycle 4: `out = 3 - 3 = 0`
cycle 5: `out = 0 + 4 = 4`
cycle 6: `out = 4 + 5 = 9`
cycle 7: `out = 9 + 6 = 15`
cycle 8: `out = 15 - 7 = 8`
cycle 9: `out = 8 + 8 = 16`
cycle 10: `out = 16` (all inputs have been processed. `var1` becomes 0 and `out` remains as is for future cycles unless `en` becomes `1`)

4.2 Complete the Verilog code [30 points]

For each numbered blank (1)-(5) in the following Verilog code, **mark the choice below** (i.e., one of options A, B, C, D) that makes the Verilog module operate as described in the comments. The resulting code must have correct syntax.

```
1  module my_module (input clk, input rst,
2    input[1:0] data, (1) result);
3
4    (2) state = 2'b00; // defining a 2-bit signal with an initial value of 0
5
6    always @(posedge clk) begin
7      case (state)
8        2'b00:
9          state <= state + (3); // set the next 'state' to 2'b11
10       2'b01:
11         state <= 2'b00;
12       2'b10: begin
13         state <= 2'b11;
14
15         if ((4)data)         // set the next 'state' to 2'b01 if
16           state <= 2'b01; // all bits of 'data' are 1
17       end
18       2'b11:
19         state <= 2'b10;
20     endcase
21
22   end
23
24   assign result = (5)state; // assign 1'b1 to 'result' if 'state' has any bit set to 1
25                             // otherwise assign 1'b0
26 endmodule
```

Provide your choice for each blank (1)-(5) below. Circle only one of A, B, C, D for each blank.

(1):	A. output	B. output reg	C. output reg[0:0]	D. input reg
(2):	A. reg[1:0]	B. reg	C. wire	D. wire[1:0]
(3):	A. 1'b3	B. 3'b2	C. 2'd11	D. 3
(4):	A. ||	B. &	C. !	D. 1
(5):	A. |	B. &	C. &&	D. ^

Explanation.
(1): `result` must be specified as a single bit 'output' signal because it gets assigned either 1'b0 or 1'b1 via an 'assign' operator. It cannot be specified as a 'output reg' because the 'assign' operator can be used only with 'wire' signals. Note: 'output' is the same as 'output wire'.
(2): `state` is a two-bit signal (as we can tell from lines 8, 10, 12) and must be a 'reg' because it gets assigned a value in an 'always' block.
(3): In order to transition to `state = 3` from `state = 0`, we need to add 3. Note that A. 1'b3 is not a valid syntax as 3 is not a binary number. Not in the choices, but 1'd3 would also be incorrect since 3 cannot be encoded with a single bit. C. 2'd11 has a similar problem as decimal 11 cannot be encoded with 2 bits.
(4): In the given choices, only the AND-reduction (&) operator provides the expected functionality of resulting in 1 when all bits of `data` are 1.
(5): In the given choices, only the OR-reduction (|) operator provides the expected functionality of resulting in 1 when `state` contains at least one 1.

5 Memory Potpourri [30 points]

Read the following statements about memory organization & technology. Circle "True" if the statement is true and "False" otherwise. *Note: we will subtract 1 point for each* ***incorrect*** *answer and award 0 points for unanswered questions.*

1. [2 points] A main memory access typically consumes less energy than a register file access.

   1. True 2. False

2. [2 points] Building a larger memory array by increasing the length of the array's wordlines and bitlines increases the cost ($) but does not increase the access time of the array.

   1. True 2. False

3. [2 points] Activating a DRAM cell temporarily destroys the value stored in the DRAM cell.

   1. True 2. False

4. [2 points] DRAM cost ($) per bit is much higher than that of SRAM.

   1. True 2. False

5. [2 points] The memory hierarchy of a typical computer system comprises different memory technologies.

   1. True 2. False

6. [2 points] Recently accessed data should be kept at the bottom-level in the memory hierarchy (e.g., main memory or disk) and not at the top-level (e.g., caches) in the hierarchy.

   1. True 2. False

7. [2 points] A program with no branches has high temporal locality in its instruction memory references.

   1. True 2. False

8. [2 points] A cache that has a block size equal to word size of memory access instructions cannot exploit spatial locality.

   1. True 2. False

9. [2 points] Memory banking enables concurrent access to the memory structure.

   1. True 2. False

10. [2 points] In DRAM, accesses to different rows in one bank can be serviced faster compared to accesses to the same row in one bank.

    1. True 2. False

11. [2 points] PCM is non-volatile, which means PCM retains stored data even when it is powered off.

    1. True 2. False

12. [2 points] If a hypothetical system is not constrained by chip area, memory cost ($), and energy consumption, DRAM would be the best memory technology to use in that system.

    1. True 2. False

13. [2 points] The entire page table is typically stored in physical memory.

    1. True 2. False

14. [2 points] Virtual-to-physical address translation is on the critical path of a memory access.

    1. True 2. False

15. [2 points] Virtual memory makes programmer's and microarchitect's tasks easier.

1. `True` 2. `False`

6 Performance Evaluation [70 points]

Some fellow students are working on a project called *AwesomeMEM*, where their goal is to optimize the memory hierarchy (caches and DRAM) to enhance the performance of a multi-core system.

They evaluate two system configurations. First, the *Baseline* configuration constitutes a system with two processors, a last-level cache (LLC), and DRAM as main memory. Second, the *AwesomeMEM* configuration builds on top of the *Baseline* configuration by employing optimizations to the memory hierarchy.

The students evaluate the performance benefits of *AwesomeMEM* in simulation as follows:

1. First, they collect performance metrics of four single-threaded applications (App_1, App_2, App_3, App_4) running in isolation in the *Baseline* configuration.
2. Second, they create two-application mixes to perform a multi-program simulation, where two applications run concurrently in the *Baseline* configuration, each in a dedicated processor. They evaluate two application mixes: Mix_1 (consisting of App_1 and App_2); and Mix_2 (consisting of App_3 and App_4).
3. Third, they use the same two-application mixes as in the second step to perform a multi-program simulation, where two applications run *concurrently* in the *AwesomeMEM* configuration, each in a dedicated processor.

Table 1 summarizes the performance metrics the students collected for each step.

Table 1: Performance metrics the students collected.

Execution Mode	Application Mix	Configuration	Application	Executed Instructions	Executed Cycles	LLC Miss Rate (%)	Branch Misprediction Rate (%)	DRAM Bank Conflict Rate (%)
Single-threaded	N/A	Baseline	App_1	100,000	40,000	26%	1%	42%
			App_2	100,000	800,000	99%	1%	94%
			App_3	100,000	500,000	52%	1%	89%
			App_4	100,000	20,000	10%	1%	14%
Multi-programmed	Mix_1	Baseline	App_1	100,000	200,000	99%	1%	97%
			App_2	100,000	900,000			
		AwesomeMEM	App_1	80,000	100,000	65%	1%	55%
			App_2	80,000	400,000			
	Mix_2	Baseline	App_3	100,000	600,000	60%	1%	90%
			App_4	100,000	20,000			
		AwesomeMEM	App_3	80,000	400,000	50%	1%	45%
			App_4	100,000	20,000			

Answer the following questions based on the performance metrics the students collected.

(a) [20 points] What is the Instructions Per Cycle (IPC) of each of the four applications when the application is executed *in isolation* in the *Baseline* configuration? Show your work.

App_1:

$$IPC = \frac{\#instructions}{\#cycles} = \frac{100{,}000}{40{,}000} = \mathbf{2.5}$$

App_2:

$$IPC = \frac{\#instructions}{\#cycles} = \frac{100{,}000}{800{,}000} = \mathbf{0.125}$$

App_3:

$$IPC = \frac{\#instructions}{\#cycles} = \frac{100{,}000}{500{,}000} = \mathbf{0.2}$$

App_4:

$$IPC = \frac{\#instructions}{\#cycles} = \frac{100{,}000}{20{,}000} = \mathbf{5}$$

To measure the system throughput of a multi-core system, the students use the *weighted speedup* metric, which sums the Instructions Per Cycle (IPC) slowdown experienced by each application compared to when it is run alone (IPC_i^{alone}) for the same number of instructions as it executed in the multi-programmed application mix (IPC_i^{shared}):

$$\text{System Throughput} = \text{Weighted Speedup} = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}}$$

(b) [20 points] What is the IPC_i^{shared}, $i \in \{App_1, App_2, App_3, App_4\}$, of each of the four applications when they are executed *concurrently* in accordance with their multi-programmed application mix in the *Baseline* and *AwesomeMEM* configurations? Show your work.

App_1:

Baseline: For the *Baseline*: $IPC_{App_1}^{shared} = \frac{\#instructions}{\#cycles} = \frac{100{,}000}{200{,}000} = \mathbf{0.5}$

AwesomeMEM: For *AwesomeMEM*: $IPC_{App_1}^{shared} = \frac{\#instructions}{\#cycles} = \frac{80{,}000}{100{,}000} = \mathbf{0.8}$

App_2:

Baseline: For the *Baseline*: $IPC_{App_2}^{shared} = \frac{\#instructions}{\#cycles} = \frac{100{,}000}{900{,}000} = \mathbf{0.11}$

AwesomeMEM: For *AwesomeMEM*: $IPC_{App_2}^{shared} = \frac{\#instructions}{\#cycles} = \frac{80{,}000}{400{,}000} = \mathbf{0.2}$

App_3:

Baseline: For the *Baseline*: $IPC_{App_3}^{shared} = \frac{\#instructions}{\#cycles} = \frac{100{,}000}{600{,}000} = \mathbf{0.16}$

AwesomeMEM: For *AwesomeMEM*: $IPC_{App_3}^{shared} = \frac{\#instructions}{\#cycles} = \frac{80{,}000}{400{,}000} = \mathbf{0.2}$

App_4:

Baseline: For the *Baseline*: $IPC_{App_4}^{shared} = \frac{\#instructions}{\#cycles} = \frac{100{,}000}{20{,}000} = \mathbf{5}$

AwesomeMEM: For *AwesomeMEM*: $IPC_{App_4}^{shared} = \frac{\#instructions}{\#cycles} = \frac{100{,}000}{20{,}000} = \mathbf{5}$

(c) [10 points] What is the *weighted speedup* of each of the two application mixes when it is executed in the *Baseline* configuration? Show your work.

Mix_1:

$$\text{Weighted Speedup} = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}} = \frac{IPC_{App_1}^{shared}}{IPC_{App_1}^{alone}} + \frac{IPC_{App_2}^{shared}}{IPC_{App_2}^{alone}}$$

$$\text{Weighted Speedup} = \frac{0.5}{2.5} + \frac{0.11}{0.125} = \mathbf{1.08}$$

Mix_2:

$$\text{Weighted Speedup} = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}} = \frac{IPC_{App_3}^{shared}}{IPC_{App_3}^{alone}} + \frac{IPC_{App_4}^{shared}}{IPC_{App_4}^{alone}}$$

$$\text{Weighted Speedup} = \frac{0.16}{0.2} + \frac{5}{5} = \mathbf{1.8}$$

(d) [10 points] What is the *weighted speedup* of each of the two application mixes when it is executed in the *AwesomeMEM* configuration? Show your work.

Mix_1:

$$\text{Weighted Speedup} = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}} = \frac{IPC_{App_1}^{shared}}{IPC_{App_1}^{alone}} + \frac{IPC_{App_2}^{shared}}{IPC_{App_2}^{alone}}$$

$$\text{Weighted Speedup} = \frac{0.8}{2.5} + \frac{0.2}{0.125} = \mathbf{1.92}$$

Mix_2:

$$\text{Weighted Speedup} = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}} = \frac{IPC_{App_3}^{shared}}{IPC_{App_3}^{alone}} + \frac{IPC_{App_4}^{shared}}{IPC_{App_4}^{alone}}$$

$$\text{Weighted Speedup} = \frac{0.2}{0.2} + \frac{5}{5} = \mathbf{2}$$

The students do *not* want to reveal the primary technique behind *AwesomeMEM*. When asked, they provided the following list of architectural techniques and told you that some of them could be the reason behind *AwesomeMEM*'s system throughput improvement:

(i) *AwesomeMEM* increases the LLC capacity by 2× that of the *Baseline*.

(ii) *AwesomeMEM* randomizes main memory requests to reduce DRAM bank conflicts.

(iii) *AwesomeMEM* employs a perfect branch predictor that always predicts a branch's direction correctly.

(iv) *AwesomeMEM* employs an efficient hardware prefetcher.

(e) [10 points] Which of the above explanations **cannot** possible be a reason for *AwesomeMEM*'s higher performance over the *Baseline*? Explain your reasoning based on the data in Table 1.

Option (iii) cannot possible be a reason for *AwesomeMEM*'s performance improvement compared to the *Baseline*.

Explanation:
(iii) is *not* possible since the branch misprediction rate n the *AwesomeMEM* configuration is the same for Mix_1 and Mix_2 compared to the *Baseline* configuration.

(i) is possible since LLC miss rate in the *AwesomeMEM* configuration drops for Mix_1 and Mix_2 compared to the *Baseline* configuration.

(ii) is possible since bank conflicts in the *AwesomeMEM* configuration drops for Mix_1 and Mix_2 compared to the *Baseline* configuration.

(iv) is possible since LLC miss rate in the *AwesomeMEM* configuration drops for Mix_1 and Mix_2 compared to the *Baseline* configuration.

7 Pipelining [70 points]

The following piece of code runs on an in-order pipelined processor as shown in the table (F: Fetch, D: Decode, E: Execute, M: Memory, W: Write back). Instructions are in the form "Instruction Destination,Source1,Source2/Immediate." For example, "ADD A, B, C" means A ← B + C.

	Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	MUL R5, R6, R7	F	D1	D2	E1	E2	E3	M	W								
2	ADDI R4, R6, 5		F	-	D1	E1	-	-	M	W							
3	MUL R4, R7, R8				F	D1	D2	E1	E2	E3	M	W					
4	ADD R5, R5, R6					F	-	D1	E1	-	-	M	W				
5	ADD R6, R7, R5							F	D1	-	-	-	E1	M	W		
6	ADD R7, R1, R4								F	-	-	-	D1	D2	E1	M	W

Use this information to reverse engineer the microarchitecture of this processor to answer the following questions. Answer the questions as precisely as possible with the provided information. If the provided information is not sufficient to answer a question, answer "Unknown" and explain your reasoning clearly.

(a) [10 points] What is the ALU's latency for an addition and for a multiplication, respectively?

Addition:

1 cycle for an addition (E1).

Multiplication:

3 cycles for a multiplication (E1, E2, E3).

(b) [10 points] Does this processor implement data forwarding? If so, between which pipeline stages? Explain your reasoning.

The processor implements data forwarding from *W* to *E1*.

(c) [10 points] The number of cycles in the decode stage dynamically varies between instructions. Explain why this might be the case. **Hint:** Register values are read from the register file in the decode stage.

All listed instructions require two operands for the ALU, which require up to two cycles to read from the register file. If one of the inputs is an immediate (e.g., instruction 2) or is forwarded from an earlier instruction, the register file has to be queried for only one input. Then, a shorter decode stage is sufficient.

(d) [10 points] What is the minimum number of register file read ports and write ports that this processor implements? Explain.

The register file has one read port and one write port.

Acording to the timeline in cycles 2 and 3, the decode stage operates in two cycles for an instruction that has two register operands. Also, acording to cycle 4, the decode stage takes one cycle for an instruction with one register operand. We conclude that the decode stage needs one cycle to decode and read each register operand which means the register file has one read port.

The register file has at least one dedicated write port since acording to cycle 12, the decode stage and the write back stage are both using the register file, and we are sure that both have been serviced by the register file within that cycle since the next cycle is not a stall for any of them.

(e) [15 points] Can we reduce the execution time of this code by enabling more read or write ports in the register file? Explain. If yes, what is the speedup compared to the baseline microprocessor assuming the changes do not impact clock frequency? Show your work.

Yes. Adding a new read port to the register file will enable the register file to service the decode unit in one cycle for any instruction with one or two register operands. The speedup is 16/13. Here is the new timeline:

	Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
1	MUL R5, R6, R7	F	D	E1	E2	E3	M	W						
2	ADDI R4, R6, 5		F	D	E1	-	-	M	W					
3	MUL R4, R7, R8			F	D	E1	E2	E3	M	W				
4	ADD R5, R5, R6				F	D	-	E1	-	M	W			
5	ADD R6, R7, R5					F	-	D	-	-	E1	M	W	
6	ADD R7, R1, R4							F	-	-	D	E1	M	W

(f) [15 points] Is it possible to run this code faster by adding more data forwarding paths to the original pipeline? If it is, explain how and calculate the speedup with respect to the original pipeline assuming the changes do not impact clock frequency. Otherwise, explain why it is not possible.

Yes, it is possible. Adding a forwarding path from *M* to *E1* can improve the performance. The speedup is 16/15. Here is a new timeline:

	Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	MUL R5, R6, R7	F	D1	D2	E1	E2	E3	M	W							
2	ADDI R4, R6, 5		F	-	D1	E1	-	-	M	W						
3	MUL R4, R7, R8				F	D1	D2	E1	E2	E3	M	W				
4	ADD R5, R5, R6					F	-	D1	E1	-	-	M	W			
5	ADD R6, R7, R5							F	D1	-	-	E1	M	W		
6	ADD R7, R1, R4								F	-	-	D1	D2	E1	M	W

8 Tomasulo's Algorithm [60 points]

Consider an in-order fetch, out-of-order dispatch, and in-order retirement execution engine that employs Tomasulo's algorithm. This engine has the following characteristics:

- The engine has four main pipeline stages: Fetch (F), Decode (D), Execute (E), and Write-back (W).
- The engine can fetch one instruction per cycle, decode one instruction per cycle, and write back the result of one instruction per cycle.
- The engine has two execution units: 1) an adder to execute ADD instructions and 2) a multiplier to execute MUL instructions.
- The execution units are fully pipelined. The adder has two stages (E1-E2), and the multiplier has four stages (E1-E2-E3-E4). Execution of each stage takes one cycle.
- The adder has a two-entry reservation station, and the multiplier has a three-entry reservation station.
- An instruction always allocates the first available entry of the reservation station (in top-to-bottom order) of the corresponding execution unit.
- Full data forwarding is available, i.e., during the last cycle of the E stage, the tags and data are broadcast to the reservation station and the Register Alias Table (RAT). For example, an ADD instruction updates the reservation station entries of the dependent instructions in the E2 stage. So, the updated value can be read from the reservation station entry in the next cycle. Therefore, a dependent instruction can potentially begin its execution in the next cycle (after E2).
- The multiplier and adder have separate output data buses, which allow both the adder and the multiplier to update the reservation station and the RAT in the same cycle.
- An instruction continues to occupy a reservation station slot until it finishes the Write-back (W) stage. The reservation station entry is deallocated after the Write-back (W) stage.

8.1 Problem Definition

The processor is to fetch and execute *five* instructions. Assume the *reservation stations (RS)* are all initially empty, and the initial state of the *register alias table (RAT)* is given below in Figure (a). Instructions are fetched, decoded, and executed as discussed in class. At some point during the execution of the five instructions, a snapshot of the state of the RS and the RAT is taken. Figures (b) and (c) show the state of the RS and the RAT at the snapshot time. A dash (–) indicates that a value has been cleared. A question mark (?) indicates that a value is unknown to you.

Reg	Valid	Tag	Value
R0	1	–	256
R1	1	–	28
R2	1	–	1
R3	1	–	3
R4	1	–	30
R5	1	–	5
R6	1	–	23
R7	1	–	20
R8	1	–	61
R9	1	–	4

(a) Initial state of the RAT

Reg	Valid	Tag	Value
R0	1	?	256
R1	1	?	28
R2	1	?	1
R3	1	?	50
R4	1	?	30
R5	0	X	?
R6	1	?	23
R7	0	Y	?
R8	0	Z	?
R9	0	T	?

(b) State of the RAT at the snapshot time

ID	V	Tag	Value	V	Tag	Value
-	-	-	-	-	-	-
Z	1	?	28	1	?	1

+

ID	V	Tag	Value	V	Tag	Value
X	1	?	50	1	?	1
Y	0	X	?	1	?	20
T	1	?	23	1	?	50

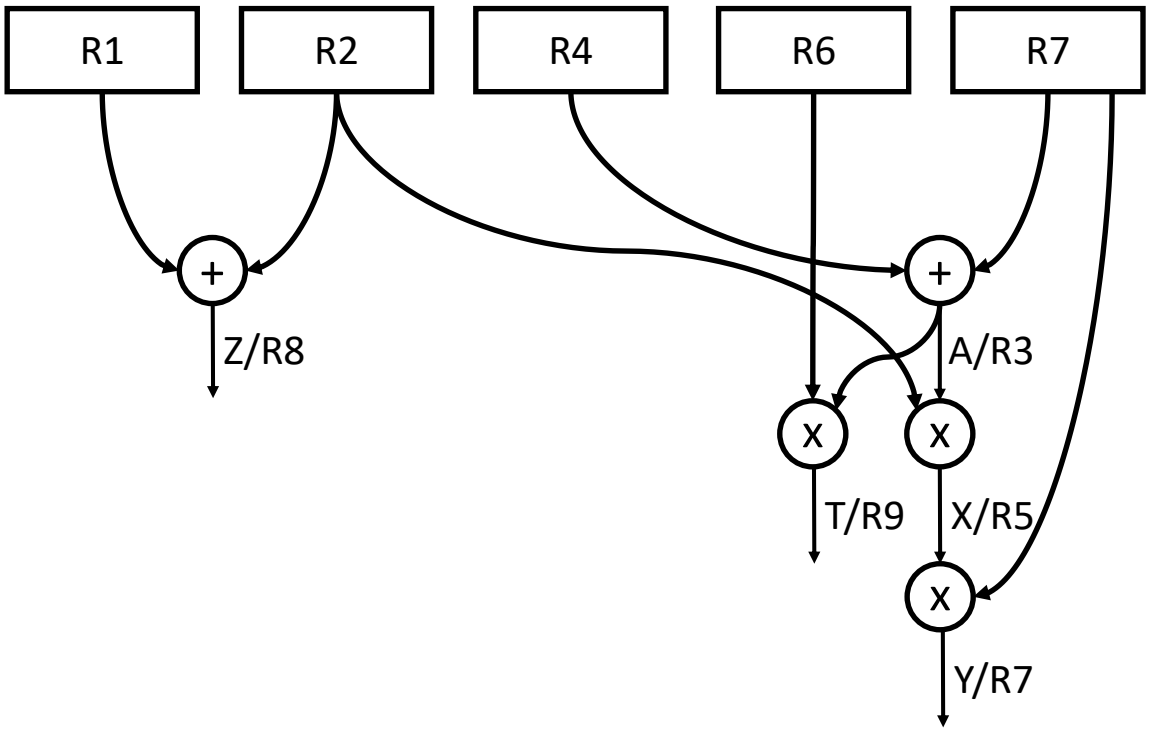
×

(c) State of the RS at the snapshot time

8.2 Questions

8.2.1 Dataflow Graph [40 points]

Based on the information provided above, identify the instructions and provide the dataflow graph below for the instructions that have been fetched. Please appropriately connect the nodes using edges and specify the direction of each edge. Label each edge with the destination architectural register and the corresponding Tag.



8.2.2 Program Instructions [20 points]

Fill in the blanks below with the five-instruction sequence in program order. There can be more than one correct ordering. Please provide *only one* correct ordering. When referring to registers, please use their architectural names (R0 through R9). Place the register with the smaller architectural name on the left source register box.
For example, `ADD R8 ⇐ R1, R5`.

ADD	R3	⇐	R7	,	R4
MUL	R5	⇐	R3	,	R2
MUL	R7	⇐	R5	,	R7
ADD	R8	⇐	R1	,	R2
MUL	R9	⇐	R6	,	R3

9 GPUs and SIMD [45 points]

We define the *SIMD utilization* of a program that runs on a GPU as the fraction of SIMD lanes that are kept busy with *active threads* during the run of a program. As we saw in lecture and practice exercises, the SIMD utilization of a program is computed across the *complete run* of the program.

The following code segment is run on a GPU. A warp in the GPU consists of 32 threads, and there are 32 SIMD lanes in the GPU. Each thread executes **a single iteration** of the shown loop. Assume that the data values of the arrays `A`, `B` and `C` are already in vector registers so there are no loads and stores in this program. Both `B` and `C` are arrays of integers and each integer in these arrays has an absolute value of less than 10 (i.e., $|B[i]| < 10$ and $|C[i]| < 10$, for all i).

```
for (i = 0; i < 1024; i++) {
    A[i] = B[i] * C[i];     // instruction 1
    if (/* Condition */) { // instruction 2
        // instruction 3
        // instruction 4
        .
        .
        .
        // instruction k + 2
    }
    C[i] = C[i] - 1;        // instruction k + 3
}
```

Please answer the following four questions.

(a) [5 points] How many warps does it take to execute this program? Show your work.

32 Warps.

Explanation:
Warps = (Number of threads) / (Number of threads per warp) Number of threads = 2^{10} (i.e., one thread per loop iteration) Number of threads per warp = 32 = 2^5 (given) Warps = $2^{10}/2^5 = 2^5$

(b) [20 points] Assume that the condition for the `if` statement is `(i % 16 == 0)`. What is the number of instructions (`k`) in the body of the conditional block given a SIMD utilization of $\frac{11}{32}$? Assume that there are **no** control flow instructions in the body of the `if` statement. Show your work.

7 Instructions.

Explanation:
Two of the 32 threads go inside of the conditional block. This pattern is homogeneous through all warps.

$\frac{2\times(3+k)+30\times3}{32\times(3+k)} = \frac{11}{32} \rightarrow k = 7$ instructions.

(c) [20 points] Assume that the condition for the `if` statement is `(i % 16 == 0 && i < 512)`. What is the number of instructions (`k`) in the body of the conditional block given a SIMD utilization of $\frac{5}{8}$? Assume that there are **no** control flow instructions in the body of the `if` statement. Show your work.

4 Instructions.

Explanation:
Two of the 32 threads **only within the first 16 warps** go inside of the conditional block. In the rest of the warps no thread goes inside of the conditional block.

$\frac{16(32\times(3))+16(2\times(k+3)+30\times3)}{16(32\times(3+k))+16(32\times3)} = \frac{5}{8} \rightarrow k = 4$ instructions.

10 Branch Prediction [70 points]

A processor implements an *in-order* pipeline with multiple stages. Each stage completes in a single cycle. The pipeline stalls after fetching a *conditional branch instruction* and resumes execution once the condition of the branch is evaluated. There is no other case in which the pipeline stalls.

10.1 Part I: Microbenchmarking [30 points]

You create a microbenchmark as follows to reverse-engineer the pipeline characteristics:

```
LOOP1:
    SUB R1, R1, #1  // R1 = R1 - 1
    BGT R1, LOOP1   // Branch to LOOP1 if R1 > 0

LOOP2:
    B   LOOP2       // Branch to LOOP2
                    // Repeats until program is killed
```

The microbenchmark takes one input value `R1` and runs until it is killed (e.g., via an external interrupt).

You carefully run the microbenchmark using different input values (`R1`) as summarized in Table 2. You terminate the microbenchmark using an external interrupt such that each run is guaranteed to execute exactly 50 *dynamic instructions* (i.e., the instructions actually executed by the processor, in contrast to *static instructions*, which is the number of instructions the microbenchmark has).

Initial R1 Value	Number of Cycles Taken
2	71
4	83
8	107
16	155

Table 2: Microbenchmark results.

Using this information, you need to determine the following two system characteristics. *Clearly show all work to receive full points!*

1. How many stages are in the pipeline?
2. For how many cycles does a conditional branch instruction cause a stall?

1. 10 pipeline stages
2. 6 cycles

Explanation: We have a system of equations in the variables:

- C is the total number of cycles taken
- P is the total number of pipeline stages
- I is the total number of dynamic instructions executed
- B is the number of conditional branch instructions executed
- D is the number of cycles stalled for each conditional branch

The total number of cycles can be expressed as $C = P + I - 1 + B * D$.
We know that $I = 50$, and Table 2 gives us B and C, which we can use to solve the following system of equations:

- $71 = P + 50 - 1 + 2 * D$
- $83 = P + 50 - 1 + 4 * D$

Solving this system, we obtain $P = 10, D = 6$

10.2 Part II: Performance Enhancement [40 points]

To improve performance, the designers add a *mystery* branch prediction mechanism to the processor. All we know about this *mystery* branch predictior is that it does not stall the piepline at all if the prediction is correct. They keep the rest of the design exactly the same as before. You re-run the same microbenchmark with $R1 = 4$ for the same number of total dynamic instructions with the new design, and you find that the microbenchmark executes in 77 cycles.

Based on this given information, determine which of the following branch prediction mechanisms could be the *mystery* branch predictor implemented in the new version of the processor. For each branch prediction mechanism below, you should circle the configuration parameters that makes it match the performance of the mystery branch predictor.

(a) [10 points] **Static Branch Predictor**

Could this be the mystery branch predictor: `YES` `NO`

If `YES`, for which configuration below is the answer `YES`?

(I) Static Prediction Direction

`Always taken` `Always not taken`

Explain:

> `YES`, if the static prediction direction is `always not taken`.
> **Explanation:** The execution time corresponds to 3 mispredictions and 1 correct prediction. The correct prediction occurs when the branch condition evaluates to FALSE and execution falls through to the following instruction (i.e., NOT TAKEN).

(b) [10 points] **Last Time Branch Predictor**

Could this be the mystery branch predictor?

`YES` `NO`

If YES, for which configuration is the answer *YES*? Pick an option for each configuration parameter.

(I) Initial Prediction Direction

`Taken` `Not taken`

(II) Local for each branch instruction (PC-based) or global (shared among all branches) history?

`Local` `Global`

Explain:

> *NO.*
> **Explanation:** The last-time predictor will make a correct prediction at least three times, which means that it cannot be the mystery predictor.

(c) [10 points] **Backward taken, Forward not taken (BTFN)**

Could this be the mystery branch predictor?

YES NO

Explain:

NO.

Explanation: The BTFN predictor makes exactly one *mis*prediction, which is the opposite of what the mystery predictor achieves.

(d) [10 points] **Forward taken, Backward not taken (FTBN)**

Could this be the mystery branch predictor?

YES NO

Explain:

YES.

Explanation: The FTBN predictor makes exactly one correct prediction, which is what we observe from the microbenchmark.

11 BONUS: Data Prefetching [50 points]

You and your colleague are tasked with designing the prefetcher of a machine your company is designing. The machine has a single processor attached to a main memory (DRAM) system.

You need to examine different prefetcher designs and analyze the trade-offs involved. For all parts of this question, you need to compute the *coverage* or *overhead* of the prefetcher in its **steady state**.

You run an application that has the following memory access pattern (note that these are cache block addresses). **Assume this memory access pattern repeats for a long time.**

$$A,\ A+1,\ A+9,\ A+10,\ A+18,\ A+19,\ A+27,\ A+28,\ A+36,\ A+37, ...$$

(a) [10 points] You first design a stride prefetcher $Pref_X$ that observes the last three cache block requests. If there is a constant stride S between the last three requests, $Pref_X$ issues a prefetch to the next cache block using the stride S. In absence of a constant stride, $Pref_X$ refrains from prefetching. What is the coverage of $Pref_X$ for the application? Show your work. Please recall, prefetcher coverage is defined as:

$$\frac{Total\ number\ of\ prefetch\ requests\ used\ by\ the\ program}{Total\ number\ of\ main\ memory\ requests\ without\ the\ prefetcher}$$

0%

Explanation: Since the stride in the address pattern is changing between +1 and +8, the stride prefetcher $Pref_X$ cannot learn any constant stride to issue prefetch requests.

(b) [10 points] You then design a next-N-block prefetcher $Pref_Y$. For every memory access to cacheline address A, the $Pref_Y$ prefetches addresses $A+1, A+2, ..., A+N$. What is the coverage of $Pref_Y$ if you set $N = 2$?

50%

Explanation: $Pref_Y$ will prefetch $A+1$ by seeing A, $A+9$ by seeing $A+8$, and so on. Hence every alternate memory requests will be sucessfully prefetched.

(c) [10 points] A prefetcher also incurs bandwidth overhead to the system. We define a prefetcher's bandwidth overhead to the the system as:

$$\frac{Total\ number\ of\ main\ memory\ requests\ with\ the\ prefetcher}{Total\ number\ of\ main\ memory\ requests\ without\ the\ prefetcher}$$

Please note that, if multiple prefetch requests are generated for one memory address, *only one* request goes to the DRAM.

What is the bandwidth overhead of $Pref_Y$ when $N = 2$? Show your work.

3/2

Explanation:
For $Pref_Y$:
- A will prefetch addresses $A + 1, A + 2$
- $A + 1$ will prefetch addresses $A + 2, A + 3$

So, for every 2 unique cache block requests without the prefetcher, there are 3 unique cache block requests with the prefetcher $Pref_Y$. Hence the bandwidth overhead is $\frac{3}{2}$.

(d) [10 points] What is the minimum value of N required to achieve a 100% prefetch coverage for $Pref_Y$? Show your work. Remember that you should consider the prefetcher's coverage in its steady state.

8

Explanation: At $N = 8$, $A + 1$ can prefetch for $A + 9$, thus acheiving 100% coverage.

(e) [10 points] What is the bandwidth overhead of $Pref_Y$ at the value of N you find for part (d)? Show your work.

9/2

Explanation:
For $Pref_Y$ at $N = 8$:
- A will prefetch addresses $A + 1, A + 2, A + 3, A + 4, A + 5, A + 6, A + 7, A + 8$
- $A + 1$ will prefetch addresses $A + 2, A + 3, A + 4, A + 5, A + 6, A + 7, A + 8, A + 9$

So, for every 2 unique cache block requests without the prefetcher, there are 9 unique cache block requests with the prefetcher $Pref_Y$. Hence the bandwidth overhead is $\frac{9}{2}$.

12 BONUS: Cache Reverse Engineering [70 points]

You are trying to reverse-engineer the cache associativity in a newly-released system. You already know that the cache has a FIFO replacement policy and 8 blocks with a block size of 4 B. Starting with an empty cache, an application accesses five byte addresses in the following order

$$2 \rightarrow 9 \rightarrow 16 \rightarrow 25 \rightarrow 33$$

Assume you can access three addresses after the above sequence and observe the cache hit rate across these three accesses.

1. [30 points] Which three addresses should you access in order to identify the set-associativity of the cache (1-, 2-, 4- or 8-way)? There may be multiple solutions; ***please give the lowest possible addresses that can enable the identification of the set-associativity***. Please explain every step in detail to get full points.

0 → 8 → 16
Explanation. There are four possible set/way configurations, shown below. Each configuration shows the cache state after the five initial accesses. Rows and columns represent sets and ways, respectively, and the byte address accessed is shown for each occupied set:

(a) **(8 sets, 1 way)**

33
-
9
-
16
-
25
-

(b) **(4 sets, 2 ways)**

33	16
-	-
9	25
-	-

(c) **(2 sets, 4 ways)**

33	9	16	25
–	–	–	–

(d) **(1 set, 8 ways)**

2	9	16	25	33	–	–	–

At this point, all four cache associativities have 100% miss rate since they started cold. In order to differentiate the four cases with *just three* more accesses, we need to induce *different* hit/miss counts in each of the four types of cache associativities. The only way this is possible is if one cache type experiences three hits, another experiences three misses, the third one has one hit and two misses, and the last one has two hits and one miss.
Only two solutions exist to produce this case. In the two solutions, any address in each of the address ranges below can be accessed to reverse-engineer the cache associativity.

- **(0-3)→(16-19)→(32-35)**
- **(0-3)→(8-11)→(16-19)**

Choosing the lowest possible addresses, the correct solution is **0 → 8 → 16**

2. [20 points] What is the associativity of the cache if the cache hit rate observed over the 3 extra addresses accessed in Part (1) were:

Hit rate	Associativity
0	
1/3	
2/3	
1	

Based on the solution to Part (1), these are the cache associativities corresponding to different hit rates.

Hit rate	Associativity
0	4-way, 2 sets
1/3	2-way, 4 sets
2/3	1-way, 8 sets
1	8-way, 1 set

3. [20 points] When you accessed the three addresses you determined in Part (1), you observed a 100% hit rate across these three accesses. Now, your friend asks you to access four more addresses in the following order:

$$32 \rightarrow 0 \rightarrow 8 \rightarrow 28$$

Which of the above four addresses would result in a cache miss?

28.
The cache associativity which provides 100% cache hit for the three extra accesses in Part (1) is **8-way and 1 set**. The three addresses 32, 0 and 8 are already available in the cache before the four new accesses requested by your friend. 28 will result in a miss and will be added to the cache.