1	/16
2	/14
3	/18
4	/18
5	/16
6	/18

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.191 Computation Structures
Spring 2026

Quiz #2

Name	*Athena login name*	*Score*

Recitation section

☐ WF 10, 34-302 (Jan)	☐ WF 2, 34-302 (Abdullah)	☐ WF 12, 35-308 (Nathan)
☐ WF 11, 34-302 (Jan)	☐ WF 3, 34-302 (Abdullah)	☐ WF 1, 35-308 (Nathan)
☐ WF 12, 34-302 (Qihang)	☐ WF 10, 35-308 (Raul)	☐ WF 2, 8-205 (Aarush)
☐ WF 1, 34-302 (Qihang)	☐ WF 11, 35-308 (Raul)	☐ WF 3, 8-205 (Aarush)
		☐ opt-out

Please enter your name, Athena login name, and recitation section above. Enter your answers in the spaces provided below. Show your work for potential partial credit. You can use the extra white space and the back of each page for scratch work.

Problem 1: Sequential Circuits (16 points)

Sam Sequential and Carla Concurrent have been working together to build the world's greatest GCD multiplier. Like any great team, Sam and Carla split up the work: Sam would build the input-output interface for the system and Carla would implement the computation. Unfortunately, they forgot to decide upon a clear interface:

- Carla implemented a module that accepts four inputs at the same time, performs their computations, and returns four outputs.
- Sam implemented an interface that can accept one input and produce one output.

The rest of their team have already implemented their design around Sam's interface so he must find a way to connect his interface with Carla's module.

In this problem, we analyze a **Deserializer** and implement a **Serializer**. A serializer converts a stream that produces N parallel inputs into a stream that produces one input at a time. A deserializer does the opposite: it collects N inputs together and then sends them out as a bundle.

(A) (8 points) Read the implementation of the Deserializer module and fill in the values that describe its cycle-level behavior in the table that follows the code.

```
module Deserializer#(Integer packSize, Integer width);
  Vector#(packSize, Reg#(Bit#(width))) buffer(0);
  Reg#(Bit#(log2(packSize)+1)) idx(0);

  input Maybe#(Bit#(width)) data_i default = Invalid;
  input Bool clear default = False;

  Bool is_done = idx == packSize;

  rule tick;
    if (clear) begin
      idx <= 0;
    end else if (idx < packSize && isValid(data_i)) begin
      buffer[idx] <= fromMaybe(?, data_i);
      idx <= idx + 1;
    end
  endrule

  method Maybe#(Vector#(packSize, Bit#(width))) out;
    if (is_done) begin
      Vector#(packSize, Bit#(width)) data_o;
      for (Integer i = 0; i < packSize; i = i + 1) begin
        data_o[i] = buffer[i];
      end
      return Valid(data_o);
    end else begin
      return Invalid;
    end
  endmethod
endmodule
```

Describe the behavior of a `Deserializer#(3, 32)` module. We've provided the inputs and filled in the first cycle. You should abbreviate values in a vector as a list of numbers with the leftmost number as being at index zero (as shown in cycle 1).

	1	**2**	**3**	**4**	**5**	**6**	**7**
data_i	Valid(10)	Invalid	Valid(20)	Valid(30)	Valid(40)	Valid(50)	Valid(60)
clear	False	False	False	False	False	True	True
buffer	0,0,0						
idx	0						
is_done	False						
out	Invalid						

(B) (8 points) Now that we've seen Sam's implementation of the `Deserializer`, our next job is to implement a `Serializer` module with the following interface:

- The serializer can be in one of two states:
  - The module is `Idle` and can accept new data to be serialized.
  - The module has some data, `HasData`, that can be read off.
- The inputs, `data_i` and `next`, should be treated as follows:
  - If the module has no data, giving it valid data should update its internal state to contain the data.
  - If the module has data:
    - New data should be ignored.
    - It should track the currently active piece of data and provide it using the `get_data` method.
    - If the `next` input is set, it should either move to the next piece of the data, or it should reset the state of the module to be `Idle` if it no longer has valid data.

A template of the code is provided for you on the next page. Fill in the missing minispec code for the blank line. Also, fill in the code for the `rule` and the `get_data` method. The rule and method code may require multiple lines of code.

We have defined the `SerStates` enum to help with modeling the states of this module.

```
typedef enum { Idle, HasData } SerStates;
module Serializer#(Integer packSize, Integer width);
  Vector#(packSize, Reg#(Bit#(width))) buffer(0);
  Reg#(Bit#(log2(packSize)+1)) idx(0);
  Reg#(SerStates) state(_____________________________);

  // Provide new data to the module
  input Maybe#(Vector#(packSize, Bit#(width))) data_i;

  // Should we move to the next element in the current buffer.
  input Bool next default = False;

  rule tick;
  // Fill in missing code









  endrule

  method Maybe#(Bit#(width)) get_data;
  // Fill in missing code









  endmethod
endmodule
```

Problem 2: Arithmetic Pipelines (14 points)

Ben Bitdiddle has recently started playing a new videogame, "Find the Bit", where he must successfully go through a maze to find a secret code, upon which he completes that level. However, Ben has grown tired of playing the game and wants to complete the levels as fast as possible. He has created a circuit that, given valid inputs from each level (Maze Map, Current Level, and Health Status), will output the secret code necessary to finish the level.

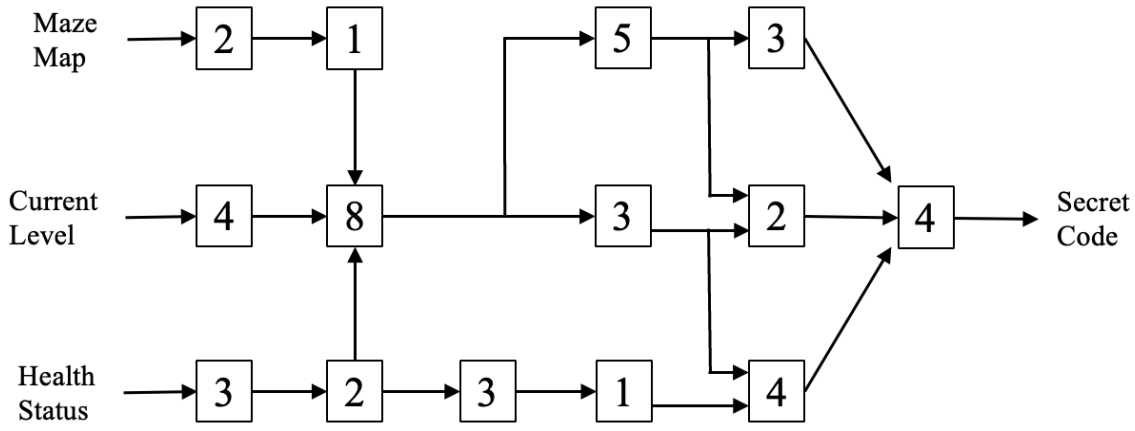
Each component in the below circuits is labeled with its propagation delay in nanoseconds. For all pipeline diagrams, use a minimum number of registers and remember that our convention is to place a pipeline register on each output. When drawing pipeline contours, place large black circles on the signal arrows to indicate where registers are to be inserted. Unless otherwise stated, assume all registers are ideal ($t_{setup} = 0$, $t_{pd} = 0$).

(A) (2 points) For the circuit shown in part (B), what is the latency and throughput for the 1-stage pipeline for Ben's circuit?

Latency: _____

Throughput: _____

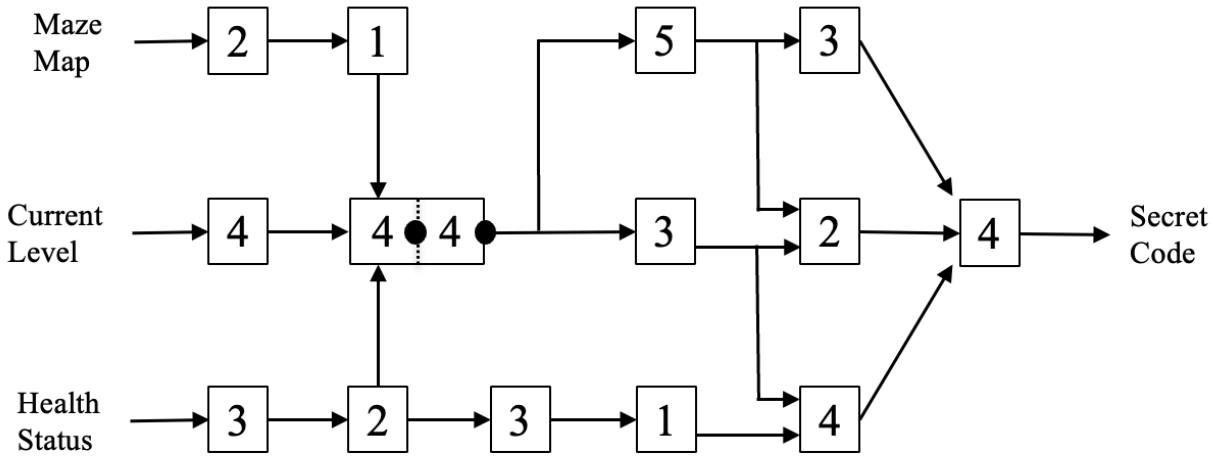
(B) (4 points) Ben has developed a circuit to quickly solve the levels of the game using the Current Level, Health Status, and Maze Map. Help him pipeline the circuit to achieve the maximum throughput with the minimum number of registers and report the latency and throughput of the resulting circuit.



Latency: _____

Throughput: _____

(C) (4 points) Ben has found that one of his modules can be replaced by a pipelined component with 2 pipelined stages as shown below. Show the maximum throughput pipeline using a minimal number of registers. What is the latency and throughput of the resulting circuit? *Make sure to account for the two registers (black dots) already present in the pipelined component.*



Latency: _____

Throughput: _____

(D) (2 points) The game developers have realized their mistake! If someone has access to all the game data, they can quickly solve all the levels. To fix this, the developers have slowed down the release of game data: instead of releasing the information in one go, players must wait for a day before accessing a new level's Maze Map and Current Level information.

Ben's circuits always produce the correct secret code and are idle until Ben provides the new level's inputs. Which circuit should he use if he wants to finish a level as fast as possible after it is accessible?

Circle one: 1-stage Pipeline Pipeline from part (B) Pipeline from part (C)

(E) (2 points) Ben accidentally spilled coffee on his circuit and is now trying to rebuild it. **He can no longer use the new component from part C and now must use registers each with $t_{PD,REG}$ = 2ns and $t_{setup,REG}$ = 0ns. All components now also have t_{CD} of 1ns.** If Ben pipelines this new circuit for maximum throughput with minimal registers, what is the throughput and latency of the circuit?

Latency: _____

Throughput: _____

Problem 3. Processor Implementation (18 points)

Victor Vector has written an assembly program that clears an array by setting all its elements to 0. The array's address is stored in `x1`, its size is stored in `x2`, and `x3` is used as a temporary variable.

```
    mv      x3, x1
loop:
    sw      x0, 0(x3)
    addi    x3, x3, 4
    addi    x2, x2, -1
    bnez    x2, loop
```

Victor notices that this loop uses 4 instructions to clear one array entry, he thinks he can accomplish the same in one instruction:

```
repclear rs1, rs2
```

The `repclear` instruction takes the size of the array in `rs1`, and a memory address in register `rs2`, both as unsigned integers. It sets an entry in the array to zero, decrements the size, and moves on to the next entry. The `repclear` instruction **keeps repeating**, not moving to `pc + 4`, until `reg[rs1]` is 0. The behavior of this instruction is summarized below:

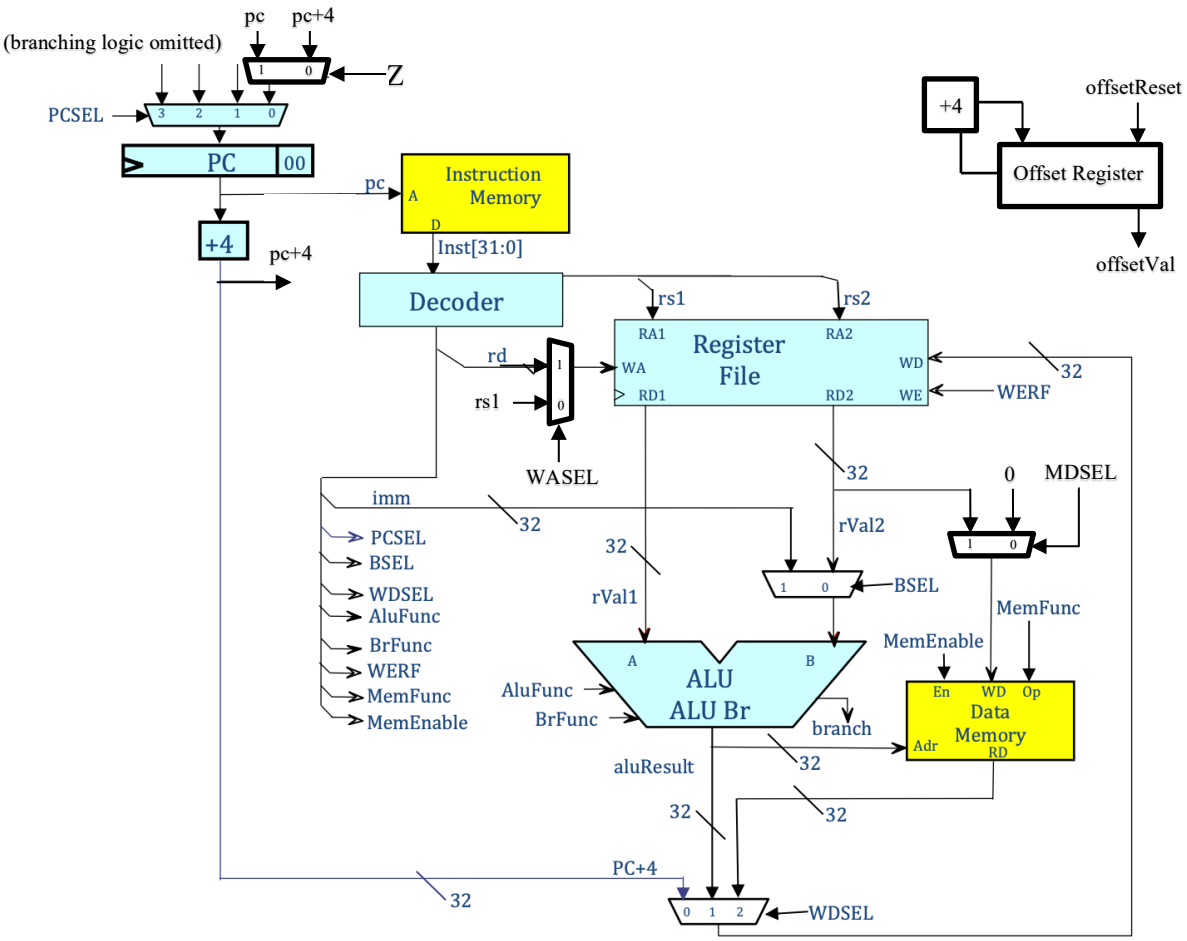
```
if (reg[rs1] >= 1)
      pc <= pc
      Mem[reg[rs2] + offset] <= 0
      reg[rs1] <= reg[rs1] – 1
      offset <= offset + 4
else
      pc <= pc + 4
endif
```

Notice that the `repclear` instruction can take multiple cycles to execute but on each cycle it can update one array element.

To implement this instruction Victor added an offset register that keeps incrementing by 4. It can be reset to zero by setting the *offsetReset* signal to 1. The value can be read from the *offsetVal* signal. He also added 3 muxes (Z, WASEL, and MDSEL). You can find this additional hardware highlighted in bold in the diagram. Throughout this problem, assume that all newly added control signals are produced by the Decoder.

Assume that this processor is capable of performing both an ALU and an ALU Br operation in the same cycle. However, the inputs for the ALU and the ALU Br must be the same.

(A) (3 points) Victor tried to connect the new wires, but he can't find a way to make `repclear` work with this new hardware.

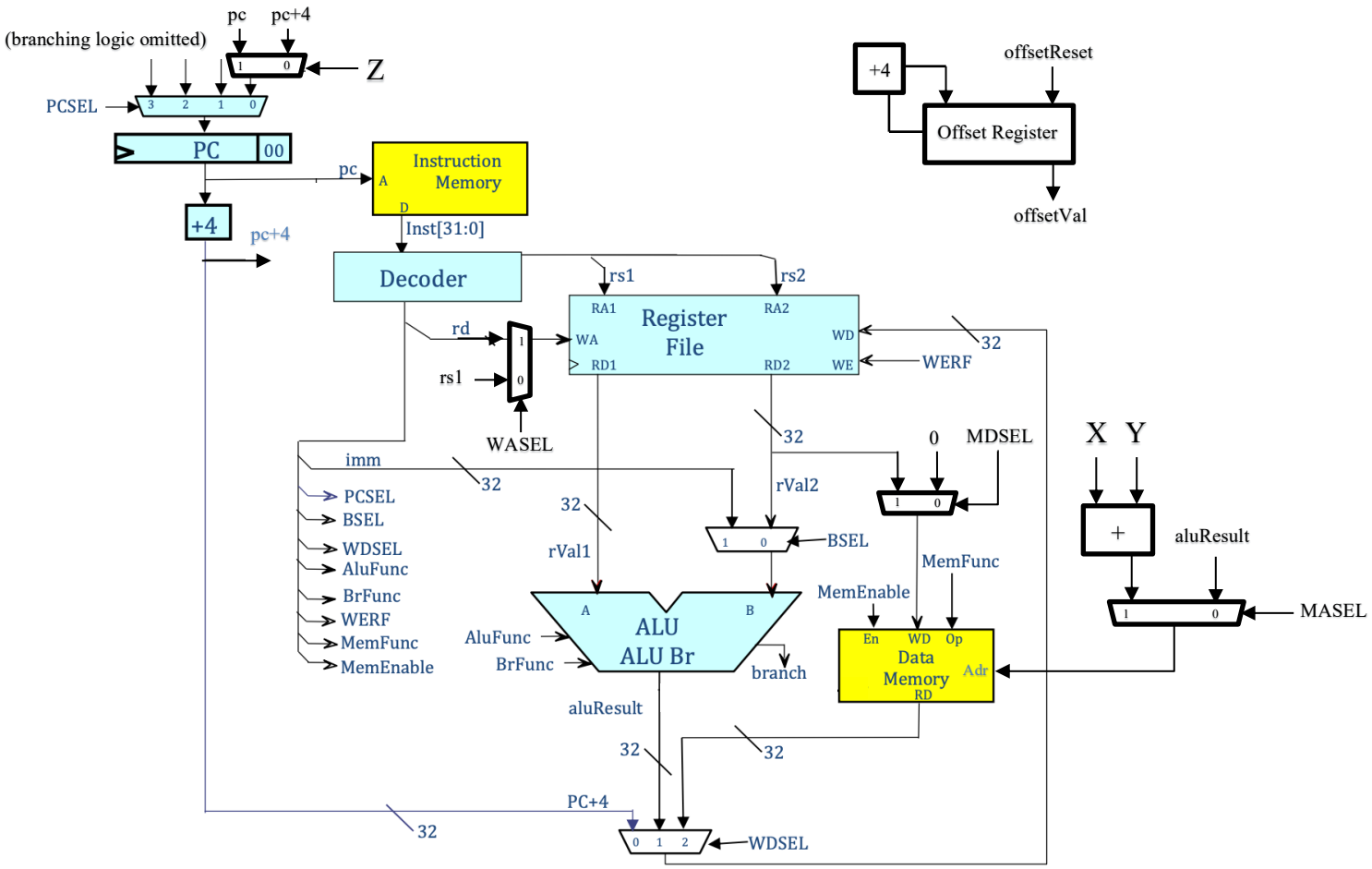


One of the lines below cannot be implemented with the current hardware. Identify which one by placing an X on the corresponding blank line and explain why it cannot be implemented with the existing hardware.

```
if (reg[rs1] >= 1)                                  _____
      pc <= pc                                      _____
      Mem[reg[rs2] + offset] <= 0                   _____
      reg[rs1] <= reg[rs1] - 1                      _____
      offset <= offset + 4                          _____
else
      pc <= pc + 4                                  _____
endif
```

Explanation: _____

Victor adds some additional hardware. This additional hardware is also shown in bold in the diagram below.



(B) (3 points) What internal signal or constant should the wires *X*, *Y*, and *Z* be connected to so that the processor can implement the `repclear` instruction.

X: _____

Y: _____

Z: _____

(C) (1 point) What should `offsetReset` be for instructions that are not `repclear`?

Value of `offsetReset` for non-`repclear` instructions: _____

(D) (3 points) For `repclear`, what should be the selector values for *MDSEL, MASEL and WASEL*?

MDSEL (for `repclear`): _____

MASEL (for `repclear`): _____

WASEL (for `repclear`): _____

(E) (8 points) For `repclear`, what should the rest of the decode signals be set to? You can set each decode signal to a constant or to an internal wire/signal.

`aluFunc` can be set to one of: `Add, Sub, And, Or, Xor, Slt, Sltu, Sll, Srl, Sra`
`brFunc` can be set to one of: `Eq, Neq, Lt, Ltu, Ge, Geu`
`MemFunc` can be set to one of: `Lw, Lh, Lhu, Lb, Lbu, Sw, Sh, Sb`

	Field	Value
repclear	imm	
	aluFunc	
	brFunc	
	BSEL	
	MemFunc	
	MemEnable	
	WDSEL	
	WERF	
	PCSEL	

Problem 4: Cache Reverse Engineering (18 points)

After getting an C in 6.191 Ben decided to start a company designing RISC-V CPUs. However, he relied on an LLM too much when learning about caches and is now uncertain what cache geometry would best suit his product. Rather than finding the best design himself, Ben decided to reverse-engineer the designs of his competitors. Since the other companies keep their cache specifications secret, he must find the cache properties by analyzing the cache behavior on various workloads.

Ben obtained a CPU from Company#1 and learned that it has separate instruction and data caches. The data cache has a total capacity of 64 words with a block size of 4 words. Ben does not know the associativity of the data cache, but he knows it is either direct mapped, or 2-way set associative with LRU replacement policy.

(A) (2 points) Determine which address bits are used for the data cache index in each of the two possible configurations.

Cache index in direct mapped cache: A [____: ____]

Cache index in 2-way set associative cache: A [____: ____]

To determine whether the data cache is direct mapped or 2-way set associative, Ben ran the following program on Company#1's CPU with the cache initially empty:

```
   mv x1, x0         // byte index into array
   li x2, 128        // set bound for end of array
                     // (arrays contain 32 words = 128 bytes each)
loop:
   lw x3, 0x100(x1)  // load next value from array A
   lw x4, 0x180(x1)  // load next value from array B
   add x5, x3, x4    // add the values
   sw x5, 0x200(x1)  // store sum in array C
   addi x1, x1, 4          // increment array byte index
   blt x1, x2, loop  // process until end of array

   unimp

. = 0x100
A: .word 1           // 32-element integer array A
   ...
   .word 32
. = 0x180
B: .word 1           // 32-element integer array B
   ...
   .word 32
. = 0x200
C: .word 1           // 32-element integer array C
   ...
   .word 32
```

(B) (3 points) Ben observed that data cache hit ratio for this program is 0%. Based on this result, determine which cache type Company#1 is using. Circle the correct answer.

Company#1's data cache type: **Direct Mapped** **2-Way Set Associative**

(C) (4 points) What would be the data cache hit ratio if Company#1 used the **other cache** type? (the one you did not circle in the previous part)

Data cache hit ratio using the alternative cache type: _____

Ben next acquired a CPU from Company#2. He knows it has separate instruction and data caches. The data cache is a direct mapped data cache of 64-word capacity, but the block size is unknown. He ran the following program on Company#2's CPU with the cache initially empty:

```
   mv x1, x0         // byte index into array
   li x2, 128        // set bound for end of array
                     // (array contains 32 words = 128 bytes)
loop:
   lw x3, 0x100(x1)  // load next value from array A
   addi x3, x3, 1    // increment the value
   sw x3, 0x100(x1)  // store the value back in array A
   addi x1, x1, 4          // increment array byte index
   blt x1, x2, loop  // process until end of array

   unimp

. = 0x100
A: .word 1           // 32-element integer array A
   ...
   .word 32
```

(D) (3 points) If Company#2's data cache uses a block size of 8 words, what data cache hit ratio would Ben observe when running this program?

Data cache hit ratio with block size 8 words: _____

(E) (2 points) If Ben observes a 50% data cache hit ratio, what block size must the data cache use? Answer in number of words in the block.

Block size required for 50% data cache hit ratio: _____

(F) (2 points) If Ben observes a 7/8 data cache hit ratio, what block size must the data cache use? Answer in number of words in the block.

Block size required for 7/8 data cache hit ratio: _____

(G) (2 points) In addition to cache geometry, Ben wants to find the access time of Company#2's data cache to compare with his own design. He measured the AMAT for data accesses to be 10 cycles with 7/8 data cache hit ratio. On a miss, it takes an additional 64 cycles to retrieve the data from main memory and update the cache. What is the access time of the data cache?

Data cache access time: _____

Problem 5. Pipelined Processors (16 points)

Assume a 5-stage in-order pipeline with stages IF, DEC, EXE, MEM, WB. Each stage takes one cycle. As usual this pipelined processor,

- register file is read in DEC and written in WB
- branch outcome is resolved in EXE
- `jal` changes control flow and its target is known in EXE
- processor has support for branch annulment and always predicts that branches are not taken
- when a hazard cannot be resolved, the processor inserts NOPs / stalls

Unlike most pipelined processors, though, this processor returns data from memory in the MEM stage.

- **memory accesses for `lw` are returned in MEM state (in the same cycle)**

```
start:      li   t0, 0
            mv   t1, a0
loop:       beq  t1, x0, done
            lw   t3, 8(t1)
            lw   t2, 0(t1)
            add  t0, t0, t2
            mv   t1, t3
            jal  x0, loop
done:       mv   a0, t0
            ret
```

Consider the program above which sums the elements of a linked list. A linked list is a data structure consisting of nodes where each node contains a value and a pointer to the next element of the linked list. `a0` initially contains the pointer to the first element in the list. Each node stores its value at offset `0` and the pointer to the next node at offset `8`. The program iterates through the list, adds each node's value into `t0`, updates the current pointer to the next node, and stops when the pointer becomes null. The final sum is returned in `a0`. *Note: you do not need to understand the code to do this problem.*

(A) (7 points) Consider the above program running on a **processor with no bypass paths**. Assume that the loop has run through many times and will continue being repeated. Fill out the pipeline diagram below assuming that at cycle 101 a new iteration of the loop is about to begin with the `beq` instruction. What is the average CPI assuming the loop runs many times?

Cycle	101	102	103	104	105	106	107	108	109	110	111	112	113	114
IF	`beq`													
DEC														
EXE														
MEM														
WB														

Average CPI: _____

(B) (7 points) Next, consider the above program running on a **processor with full bypassing and jal target known in DEC stage**. All other parameters remain unchanged. Fill in the pipeline diagram assuming that at cycle 101 a new iteration of the loop is about to begin with the `beq` instruction. **Draw arrows indicating each use of bypassing.** What is average CPI assuming the loop runs many times?

Cycle	101	102	103	104	105	106	107	108	109	110	111	112	113	114
IF	`beq`													
DEC														
EXE														
MEM														
WB														

Average CPI: _____

```
start:      li   t0, 0            (1)
            mv   t1, a0           (2)
loop:       beq  t1, x0, done     (3)
            lw   t3, 8(t1)        (4)
            lw   t2, 0(t1)        (5)
            add  t0, t0, t2       (6)
            mv   t1, t3           (7)
            jal  x0, loop         (8)
done:       mv   a0, t0           (9)
            ret                   (10)
```

(C) (2 points) Our code is repeated above with each line labeled with its line number. You are asked to **swap** two instructions in the code so that all stalls are eliminated. Assuming the same processor as in part (B), identify the line numbers of two instructions that should be swapped to eliminate pipeline stalls in the steady state execution of the loop without changing the program functionality. Briefly explain your answer. *There may be more than one pair of instructions that can achieve this.*

Line numbers of swapped instructions: _____

Explanation:

Problem 6. Pipelined Processor Performance (18 points)

Ben Bitdiddle is trying to book the cheapest flight for his spring break vacation. He has gathered a list of 1000 flight prices and stored them in an array of integers in memory. Unknown to Ben, **the prices are already sorted in ascending order**. He writes the following RISC-V program to find the cheapest price:

Assume `t2` contains the base address of the array and all other registers are initialized to 0.

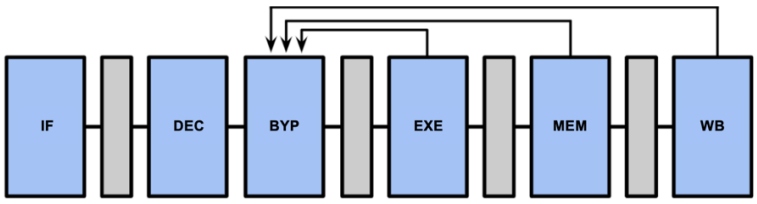
```
            addi t0, x0, 1000
            lw a0, 0(t2)
loop:       lw a1, 0(t2)
            bge a1, a0, skip
            mv a0, a1
skip:       addi t2, t2, 4
            addi t0, t0, -1
            bnez t0, loop
            sw a0, 0x1000(x0)
            ret
```

Ben runs this code on a standard 5-stage RISC-V processor with full bypassing and branch annulment. Assume that branches are always predicted not taken and that branch decisions are made in the EXE stage. **Assume that the loop will repeat many times and notice that since the array is already sorted, the `bge` branch will always be taken.**

(A) (8 points) Fill in Ben's 5-stage pipeline diagram below for cycles 3-18, assuming that at cycle 3 the `lw, a1, 0(t2)` instruction is fetched for the first time. **Draw arrows indicating each use of bypassing.** Ignore cells shaded in gray.

Cycle	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
IF	`lw`															
DEC																
EXE																
MEM																
WB																

Ben's 5-stage full-bypassing processor has the following propagation delays for each piece of combinational logic:



IF	5 ns
DEC	10 ns
EXE	15 ns
MEM	20 ns
WB	10 ns
BYP	2 ns

What is the minimum clock period for this processor assuming ideal registers with ($t_{PD} = 0$, $t_{SETUP} = 0$)? Once the program has reached steady state, how many cycles does it take to complete one iteration of the loop? What is the time it takes to execute one steady state iteration of the loop?

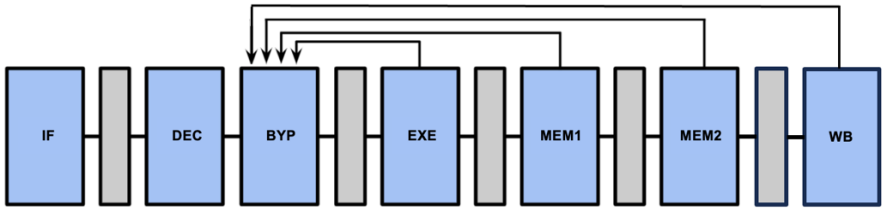
Processor minimum clock period (ns): _____

Number of cycles per loop iteration: _____

Time to execute one iteration of the loop (ns): _____

Louis Reasoner wants to help Ben speed up his pipeline. He suggests pipelining the data memory into two sub-stages (MEM1 and MEM2), each taking one cycle.

(B) (3 points) Louis' processor design has the following propagation delays for each piece of combinational logic:



IF	5 ns
DEC	10 ns
EXE	15 ns
MEM1	12 ns
MEM2	12 ns
WB	10 ns
BYP	2 ns

What is the minimum clock period for this processor assuming ideal registers with ($t_{PD} = 0$, $t_{SETUP} = 0$)?

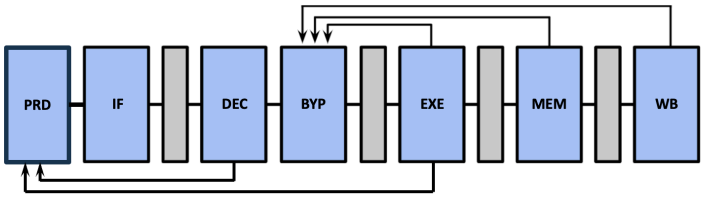
Processor minimum clock period (ns): _____

How do the number of cycles per loop iteration change when using Louis' processor as compared to Ben's processor of part (A). Circle the correct answer.

i. Number of cycles per loop iteration is the same.
ii. Number of cycles per loop increases by one when using Louis' processor
iii. Number of cycles per loop increases by two when using Louis' processor
iv. Number of cycles per loop decreases by one when using Louis' processor
v. Number of cycles per loop decreases by two when using Louis' processor

Alyssa P. Hacker realizes that flight prices are always sorted in either ascending or descending order and suggests a different improvement. Starting from Ben's original processor, she proposes adding a branch predictor. On the first execution of a branch, the processor predicts that the branch is not taken. On all subsequent executions, it predicts based on the outcome of the most recent execution of that branch. For each branch instruction, it keeps track of whether the branch was previously taken and the target address of the branch.

When a branch resolves in EXE, the processor records the outcome. **When a branch, that has previously been executed, is detected in DEC, the predictor uses this recorded outcome to determine which instruction to fetch next in that same cycle**. The schematic below shows the position of the prediction logic (PRD) and the propagation delays for each piece of combinational logic:



PRD	2 ns
IF	5 ns
DEC	10 ns
EXE	15 ns
MEM	20 ns
WB	10 ns
BYP	2 ns

(C) (7 points) Assume that the loop repeats many times and it's currently in the middle of its execution. Fill in Alyssa's 5-stage pipeline diagram below for cycles 100-115, assuming that at cycle 100 the `lw, a1, 0(t2)` instruction is fetched. **Draw arrows indicating each use of bypassing.** Ignore cells shaded in gray.

Cycle	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	**108**	**109**	**110**	**111**	**112**	**113**	**114**	**115**
IF	lw															
DEC																
EXE																
MEM																
WB																

What is the minimum clock period for this processor assuming ideal registers with ($t_{PD} = 0$, $t_{SETUP} = 0$)? Once the program has reached steady state, how many cycles does it take to complete one iteration of the loop? What is the time it takes to execute one steady state iteration of the loop?

Processor minimum clock period (ns): _____

Number of cycles per loop iteration: _____

Time to execute one iteration of the loop (ns): _____

END OF QUIZ 2!