1	/20
2	/17
3	/16

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.191 Computation Structures
Updated Fall 2022

Quiz #2

Name Solutions	*Athena login name*	*Score*

Recitation section

☐ WF 10, 34-301 (Grace)	☐ WF 2, 13-5101 (Frances)	☐ WF 12, 36-155 (Shiqi)
☐ WF 11, 34-301 (Grace)	☐ WF 3, 13-5101 (Frances)	☐ WF 1, 36-155 (Shiqi)
☐ WF 12, 35-310 (Alexandra)	☐ WF 10, 13-4101 (Georgia)	☐ WF 1, 34-303 (Amelia)
☐ WF 1, 35-308 (Alexandra)	☐ WF 11, 13-4101 (Georgia)	☐ WF 2, 34-303 (Amelia)
		☐ 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 backs of the pages for scratch work.

Problem 1. Sequential Circuits in Minispec (20 points)

You are frustrated with the 77 Mass. Ave crosswalk and decide to design a better traffic signal in Minispec. To start, you want to make sure the traffic light will function well in the daytime when there's lots of traffic. After carefully analyzing traffic patterns, you define the following specification:

- The traffic light should be red for 4 cycles, then green for 10 cycles, then yellow for 1 cycle, and repeat this pattern indefinitely.
- The light starts red (and should stay red for 4 cycles before turning green).
- **Pedestrians can only cross when the light is red.**

(A) (6 points) Fill in the Minispec module on the next page to track the traffic light state as a sequential circuit.
- The `pedestriansCanCross` method should return True if and only if the light is in a state where pedestrians are allowed to cross.
- The `currentLight` method should return the current state of the traffic light.
- We have provided a **`counter` register – use this to count down** to the next state transition.

```
typedef enum { Green, Yellow, Red } LightState;

module TrafficLight;

	Reg#(LightState) light(___Red___);

	Reg#(Bit#(____4_____)) counter(_____3______);

	method Bool pedestriansCanCross = _light == Red__;

	method LightState currentLight = __light_________;

	rule tick;

		if (light == Green) begin

			if (counter == 0) begin

				light <= __Yellow_____;

			end else begin

				counter <= __counter - 1__;

			end

		end else if (light == Yellow) begin

			light <= ___Red________;

			counter <= ______3______;

		end else if (light == Red) begin

			if (counter == 0) begin

				light <= ___Green____;

				counter <= ____9_______;

			end else begin

				counter <= __counter - 1__;

			end

		end

	endrule

endmodule
```

(B) (6 points) To ensure that your module behaves as expected, fill in the timing chart below with the register values and outputs for the first 6 cycles.

Cycle	**0**	**1**	**2**	**3**	**4**	**5**
counter	3	2	1	0	9	8
light	Red	Red	Red	Red	Green	Green
currentLight	Red	Red	Red	Red	Green	Green
pedestriansCanCross	True	True	True	True	False	False

(C) (8 points) Now you want to add a new feature to your traffic light. During the daytime, you want it to work as in part (A). But during the nighttime, the traffic light should work differently:

- By default, the light should be green.
- When a pedestrian requests to cross the street and the light is green, it should remain green for **3 more cycles**, turn yellow for 1 cycle, then red for 4 cycles. Then it should go back to being green indefinitely.
- If a pedestrian requests to cross the street when the light is yellow or red, this request should be ignored and have no effect.
- If a pedestrian requests to cross the street while the light is green, and a pedestrian requests to cross the street in a following cycle when the light is still green, this request should also have no effect.

You also want to add a feature for emergency pedestrian requests. In an **emergency**, if a pedestrian requests to cross**, the light should immediately turn yellow on the next cycle**. The pedestrian request is provided as a `Maybe#(PedestrianRequest)` type – on each cycle it will either be:

- Invalid (no pedestrian request)
- Standard (a standard pedestrian request was made)
- Emergency (an emergency pedestrian request was made)

Note that your implementation should still work when the input transitions from daytime to nighttime, even though in daytime the Green light is 10 cycles and in nighttime it is only 3 cycles following a pedestrian request. Thus, if it is nighttime and our counter variable is too large (because we were counting down from a larger value during the daytime), we should "clamp" it to be no larger than it can be in nighttime. We have provided a currentCounter variable to use for this purpose – i.e. it will be clamped to the maximum value the counter can be during nighttime.

Fill in the Minispec module below to add this functionality. We have provided two inputs – one for whether it is currently nighttime or daytime, and one for whether a pedestrian has requested to cross the street in this cycle.

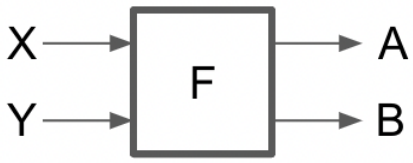
```
typedef enum { Green, Yellow, Red } LightState;

typedef enum { Daytime, Nighttime } TimeOfDay;

typedef enum { Standard, Emergency } PedestrianRequest;

module TrafficLight;

       Reg#(LightState) light(_<answer from Part A>_);

       Reg#(Bit#(_<answer from Part A>_)) counter(_<answer from part A>_);

       input TimeOfDay timeOfDay default = Nighttime;

       input Maybe#(PedestrianRequest) pedestrianRequest default = Invalid;

       method Bool pedestriansCanCross = <answer from part A>;

       method LightState currentLight = <answer from part A>;
```

The code continues on the next page.

```
        rule tick;
                if (timeOfDay == Daytime) begin
                        <Your answer from Part A>

                end else begin
                        if (light == Green) begin

                                Bit#(<answer from part A>) currentCounter;

                                // Clamp currentCounter to the maximum value counter
                                // can be for a Green light at night

                                currentCounter = counter > ___3_ ? ___3_ : counter;

                                if (currentCounter == 0) begin

                                        light <= ___Yellow___;

                                // Check if received pedestrian request this cycle

                                end else if (_____isValid(pedestrianRequest)_____)
                                        begin

                                        // Handle emergency request

                                        if (fromMaybe(?, pedestrianRequest) ==
                                Emergency) begin


                                                light <= ____Yellow__________________;

                                        end else begin

                                                counter <= ___currentCounter - 1____;
                                        end

                                end else if (currentCounter < ______3________) begin

                                        counter <= ___currentCounter - 1______;

                                end else begin

                                        counter <= ___currentCounter_(or 3)____;
                                end

                        end else if (light == Yellow) begin
                                <Your answer from Part A>

                        end else if (light == Red) begin
                                <Your answer from Part A>
                        end
                end
        endrule
endmodule
```

Problem 2. Arithmetic Pipelines (17 points)

You are given a module, named "F." This module has two inputs, **X** and **Y**, and two outputs, **A** and **B**. You are told that the circuit functions, but its throughput is too low. You decide to take a look and try to pipeline the circuit.

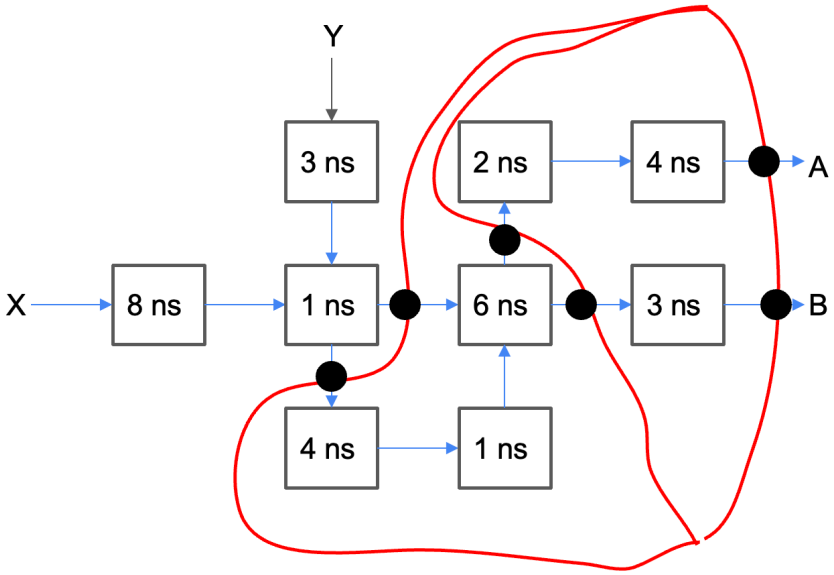


For each of the questions below, please create a valid K-stage pipeline of the given circuit. Each component in the circuit is annotated with its propagation delay in nanoseconds. **Show your pipelining contours** and place large black circles (●) on the signal arrows to **indicate the placement of pipeline registers.** Give the latency and throughput of each design, assuming ideal registers (t_{PD}=0, t_{SETUP}=0). Remember that our convention is to place a pipeline register on each output.

(A) (1 point) What is the propagation delay of the whole circuit shown below as-is without pipelining?

t_{PD} (ns): ____26____

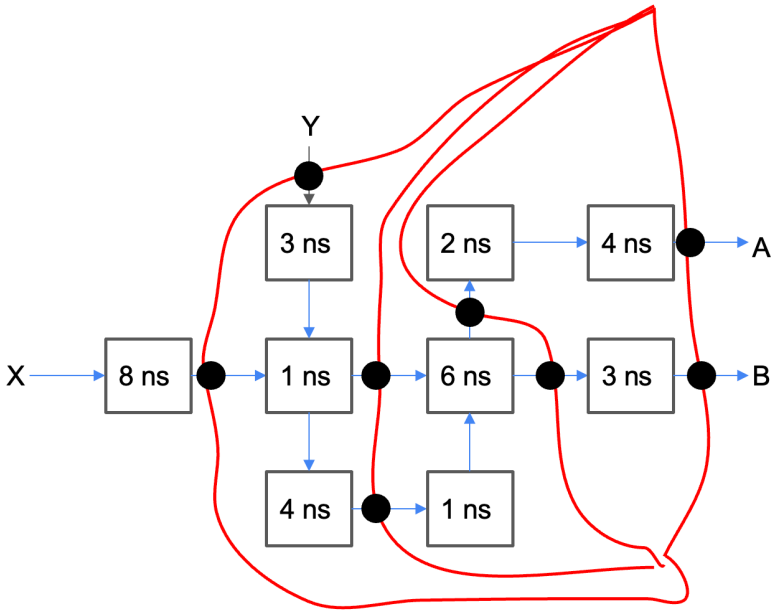
(B) (4 points) Show the **maximum-throughput 3-stage pipeline** using a minimal number of registers. What are the latency and throughput of the resulting circuit? Pay close attention to the direction of each arrow. In case you need them, extra copies of the circuit are available on the last page of the exam.



Latency (ns): ____33____

Throughput (ns^{-1}): ___1/11____

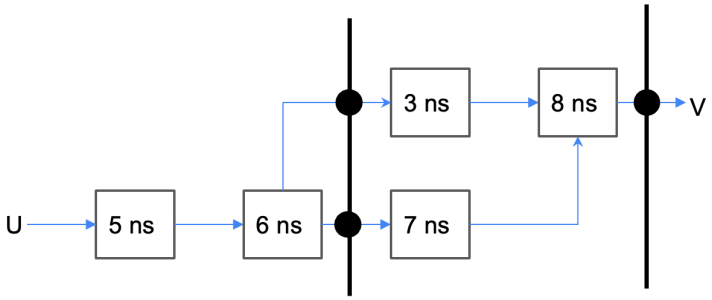
(C) (4 points) Show the **maximum-throughput pipeline** using a minimal number of registers. What are the latency and throughput of the resulting circuit? In case you need them, extra copies of the circuit are available on the last page of the exam.



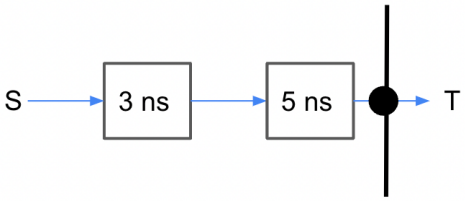
Latency (ns): ____32____

Throughput (ns^{-1}): ___1/8____

(D) Now, you are given two new modules: module "G" takes in input **U** and produces output **V**, and module "H" takes in input **S** and produces output **T**. You are given module "G" implemented with a 2-stage pipeline, with registers denoted by the large black circles (●), and module "H" implemented with a single stage pipeline as shown below.



Implementation of Module G



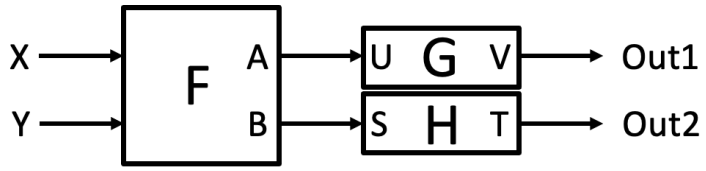
Implementation of Module H

(i) (2 points) Given the implementation of the modules above, what are the throughputs of the modules?

Throughput of G(ns^{-1}): ____1/15___

Throughput of H(ns^{-1}): ____1/8____

You want to connect these two modules with the module "F", from the previous parts of this problem, so the output **A** of Module "F" is connected to the input **U** and output **B** is connected to the input **S** as shown below.



(ii) (2 points) Do any changes need to be made to modules G or H to ensure that the combined circuit above behaves as a proper pipeline? If so, draw any updated modules below. If no changes are required, say "No changes required".

Add a pipeline stage to H so that both G and H have 2 pipeline stages. (Preferred)
OR
Remove the first pipeline stage on G (between 6 and 3/7)

(iii) (4 points) When connecting module F to module G and module H, you have two options for module F: your 3-stage pipeline, or your maximum-throughput pipeline. If you want to maximize throughput, while minimizing latency and the number of registers used, which implementation of module F would you use? What would the latency and throughput of the combined device be? Assume that any required changes to modules G or H have been made.

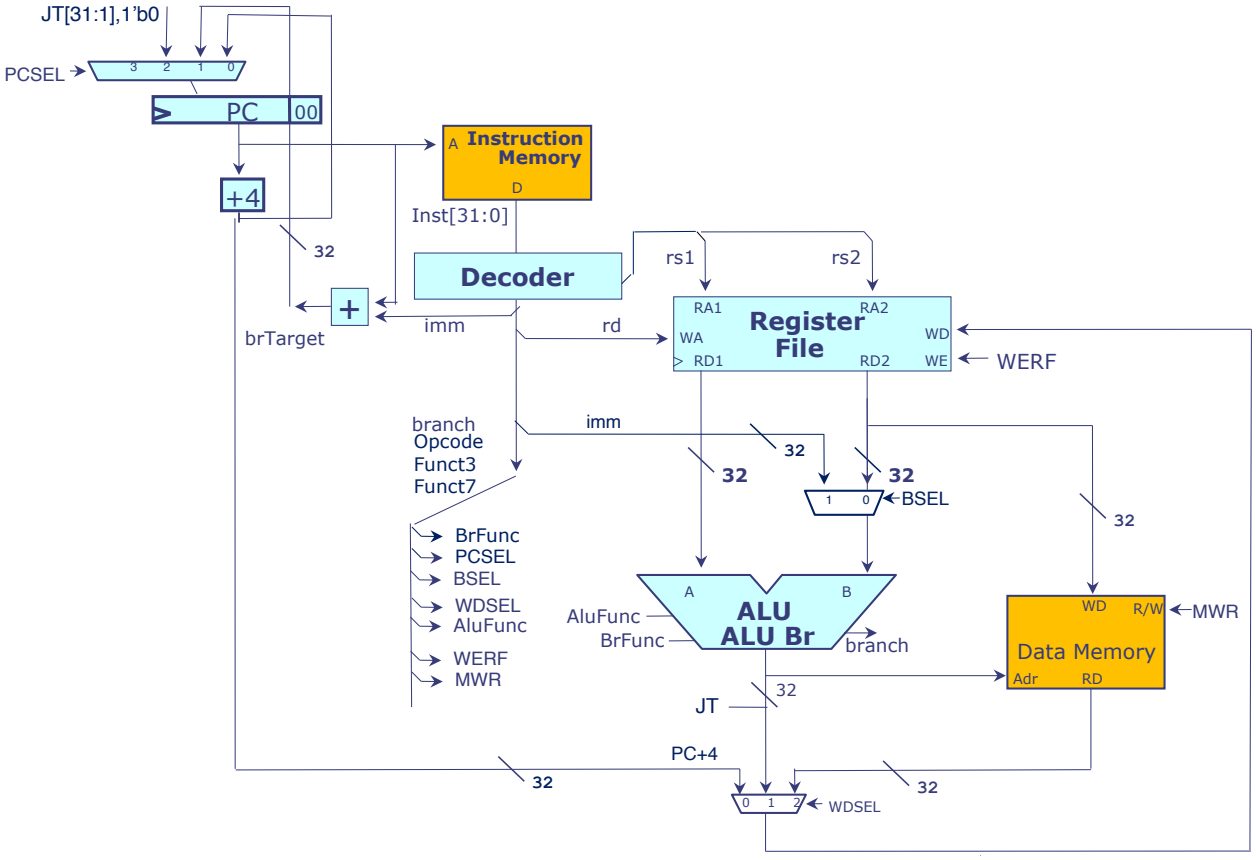
Module F implementation (circle one): 3-stage pipeline Maximum-throughput pipeline

5 Stages at 15ns/stage (if gave 1st answer to dii) or 4 Stages at 26ns/stage (if gave 2nd answer to dii) Latency (ns): __75 or 104___

Throughput (ns^{-1}): __1/15 or 1/26___

Problem 3. A RISCier processor (16 points)

Consider the single-cycle processor implementation we saw in lecture:



The timing characteristics of all components are listed below:

Component	Propagation delay (t_{PD})
Register	1ns
Decoder	2ns
RegFile read	3ns
MUX	1ns
ALU	4ns
Adder	3ns
Memory read (instruction or data)	4ns

Setup/hold times for clocked inputs (registers and writes to RegFile and data memory)

Setup time (t_{SETUP})	2 ns
Hold time (t_{HOLD})	0 ns

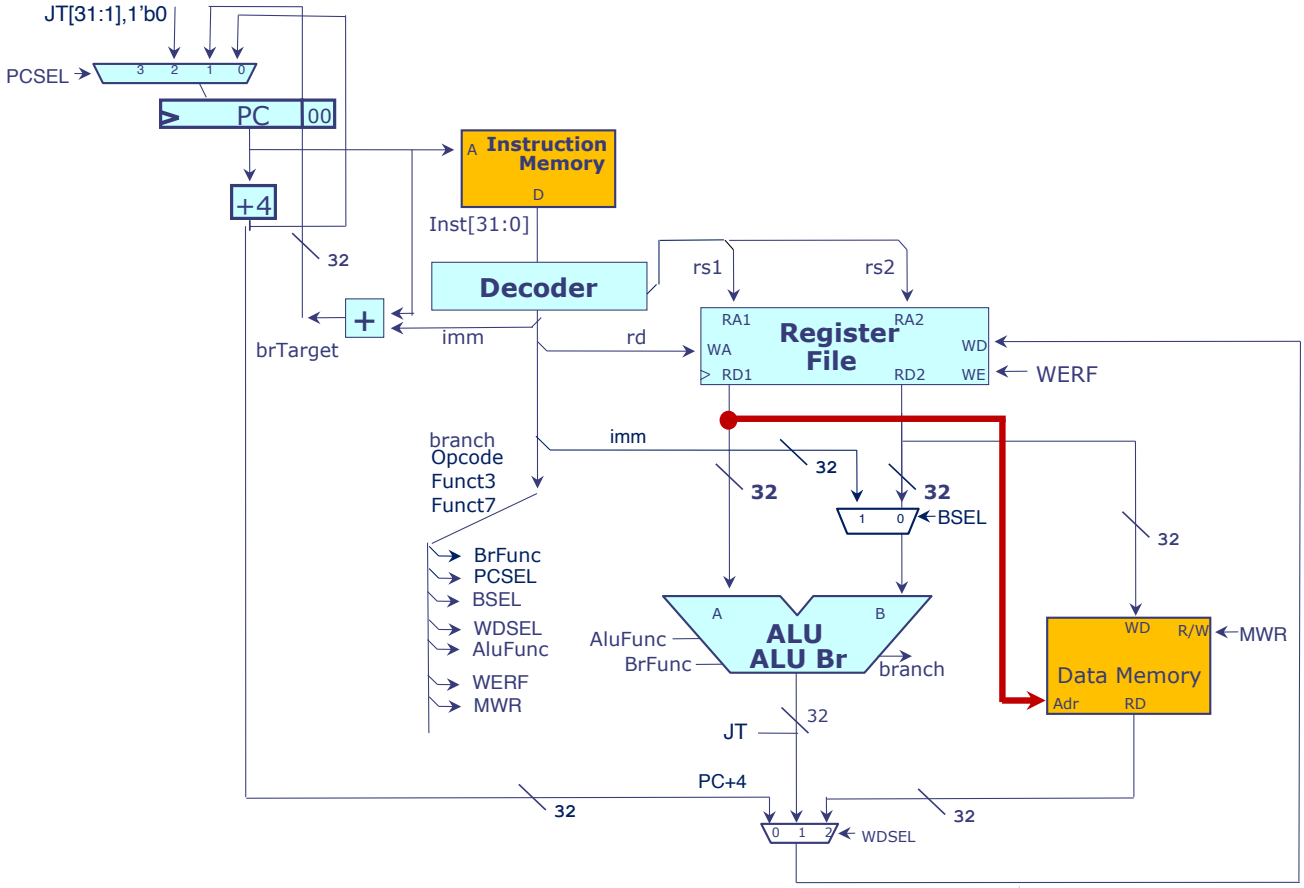
Assume that any components not listed have a delay of 0 ns.

(A) (3 points) What is the minimum clock cycle time of this processor? (For partial credit, draw the critical path in the diagram above.)

Critical path is PC -> IMem -> Decoder -> RF(read) -> BSEL mux -> ALU -> DMem (read) -> WDSEL mux -> RF(write) => t_{CLK} >= 1 + 4 + 2 + 3 + 1 + 4 + 4 + 1 + 2 (RF setup)

Minimum t_{CLK}: ____22____ ns

Ben Bitdiddle is unhappy with the performance of this processor. After a Ouija session with legendary CPU designer Seymour Cray, Ben implements this alternative datapath (the control logic stays the same), where the data memory's Adr input comes from a different place:



(B) (3 points) What is the minimum clock cycle time of Ben's new processor? (For partial credit, draw the critical path in the diagram above.)
Critical path is PC -> IMem -> Decoder -> RF(read) -> BSEL mux -> ALU -> WDSEL mux -> RF(write) => t_{CLK} >= 1 + 4 + 2 + 3 + 1 + 4 + 1 + 2 (RF setup)

Minimum t_{CLK}: ____18____ ns

(C) (2 points) Ben's processor executes some instructions incorrectly according to the RISC-V ISA. Give an example of one such instruction, and write the equivalent RISC-V instruction that is actually executed by the processor.

Example of incorrect instruction: ____lw a0, 8(a1)_____
(any lw or sw instruction with a non-zero offset)

RISC-V instruction that produces the same behavior as executing the above incorrect instruction: _____lw a0, 0(a1)_____
(the same lw or sw instruction with the offset set to 0)

(D) (5 points) The program below takes 90 instructions to execute in the original processor. However, it produces incorrect results in Ben's new processor. Modify the program so that it runs correctly on the new processor. **For full credit, the number of executed instructions should not increase compared to the original code**. How many instructions does your assembly code execute?

C code

```
int x[16];
for (int i = 0; i < 15; i++)
  x[i+1] = x[i] + x[i+1];
```

Assembly code

```
# Initial values:
# a0: address of x[0]
# a7: address of x[15]
loop:  lw a1, 0(a0)
       lw a2, 4(a0)
       add a3, a2, a1
       sw a3, 4(a0)
       addi a0, a0, 4
       blt a0, a7, loop
```

Modified assembly that executes correctly on new processor:

```
# Initial values:
# a0: address of x[0]
# a7: address of x[15]
       lw a1, 0(a0)
       addi a0, a0, 4
loop:  lw a2, 0(a0)
       add a1, a1, a2
       sw a1, 0(a0)
       addi a0, a0, 4
       ble a0, a7, loop
```

Number of executed instructions of new program: ___15*5 + 2 = 77____

(other solutions are possible; any solution with fewer instructions than the original one earns full credit, and any correct solution earns at least 2 points of partial credit)

(E) (2 points) What is the execution time of the above program in the original and new processors? (Use the appropriate variant of the program for each processor.)

Execution time on original processor: ___77*22 =1694____ ns

Execution time on Ben's new processor: ___77*18=1386____ ns

(or consistent with D)

Problem 4. Caches (20 points)

Cache Ketchum wants to design a cache to help keep track of his Pokedex entries. He's enlisted your help as a talented 6.191 student! 😀

(A) (3 points) Ketchum wants to build a direct-mapped cache with a **block size of eight words.** He also wants the cache to hold a total of **2^9 = 512 data words**. Which address bits should be used for the block offset, cache index, and tag? Assume that data words and addresses are 32 bits wide.

Address bits used for block offset: A[__4__ : __2__]

Address bits used for cache index: A[__10__ : __5__]

Address bits used for tag: A[__31__ : __11__]

(B) (2 points) Ketchum ponders over the design and decides that he wants to double the number of cache lines in his direct-mapped cache. However, he wants to keep the total number of words in the cache the same. How will the **number of bits** used to represent the block offset change as a result?

Change in # bits to represent block offset (select one of the choices below):

UNCHANGED … +1 … -1 … 2x … 0.5x … CAN'T TELL

Ketchum decides he doesn't want a direct-mapped cache at all! He wants a two-way set-associative cache.

The remainder of the problem will be considering this **2-way set-associative cache with a capacity of 32 words**. Below is a snapshot of this cache during the execution of some unknown code. V is the valid bit and D is the dirty bit of each set. Assume an LRU replacement policy and that Way 0 is currently holds the LRU cache line for all sets.

Way 0

V	D	Tag	Word 0	Word 1	Word 2	Word 3
1	0	0x28	0xA65	0x521	0xA2C	0x947
1	1	0x1D	0xB54	0xE95	0x9AA	0xC7A
1	0	0x4D	0xE71	0x2FE	0xC58	0x4C4
1	0	0x085	0xB6B	0xD55	0x27D	0xE1E

Way 1

V	D	Tag	Word 0	Word 1	Word 2	Word 3
1	1	0x093	0x2EA	0x4CE	0x42D	0x462
1	1	0x093	0x3C2	0x152	0xB9C	0xC23
1	0	0xAF	0xC05	0xE81	0xCEA	0x60B
1	0	0xA5	0x57B	0xC5F	0xA1F	0xAF5

(C) (5 points) Identify whether each of the following memory accesses is a hit or a miss. Consider each memory access independently. If it is a hit, specify what value is returned; if it is a miss, write N/A. In addition, if it is a miss, determine if any values need to be written back to main memory, and if so, to which location(s) in main memory? List all updated main memory word addresses. If no writes to main memory are needed, write N/A.

Load from address 0x2974
0x2974 = 0010_1001_0111_0100
tag = 0xA5, index = 3, block offset = 1

Circle One: (Hit) / Miss

Returned value if hit or N/A if miss: __C5F_____

All updated main memory word addresses or N/A: ___N/A_____

Load from address 0x11D8
0x11D8 = 0001_0001_1101_1000
tag = 0x47, index = 1, block offset = 2
miss → replaces way 0 line 1 which is dirty, must first write cache line back to memory
if tag = 0x1D and index = 1, then memory addresses are 111_0101_XX00 = 0x750, 0x754, 0x758, 0x75C.

Circle One: Hit / (Miss)

Returned value if hit or N/A if miss: ___N/A_____

All updated main memory word addresses or N/A: ___0x750, 0x754, 0x758, 0x75C
_____

After testing, Ketchum decides to use the cache with the following RISC-V assembly program that **increments every element in an array and stores the changed elements in another array**.

```
// Assume the following registers are initialized:
// x1 = 0xC0 (base address of input array)
// x2 = 0x80 (base address of output array)
// x3 = 4 (number of elements in input and output arrays)

    . = 0x100            // The following code starts at address 0x100
    slli x6, x3, 2
    add  x6, x1, x6      // address of end of input array

loop:
    lw   x4, 0(x1)       // get array element
    addi x4, x4, 1       // increment element
    sw   x4, 0(x2)       // store element into output array
    addi x1, x1, 4       // compute next address for input array
    addi x2, x2, 4       // compute next address for output array
    blt  x1, x6, loop    // continue looping
```

Answer the following questions about the behavior of the cache during execution of the above code. Note the cache has 2 ways and uses an LRU replacement policy**. Assume that the cache is initially empty.**

(D) (1 point) How many instruction fetches and data accesses occur per iteration of the loop?

Number of instruction fetches per loop iteration: _____6_____

Number of data accesses per loop iteration: _____2_____

(E) (1 point) How many instruction fetches and data accesses occur during execution of the **entire program**, including the instructions outside of the loop and the **four** iterations of the loop?

Total number of instruction fetches: _____26_____

Total number of data accesses: _____8_____

To help you think through the behavior of the cache on this program, we provide you with a diagram of the empty cache. You may use if you find it helpful, **but you do not need to fill it out**. Extra copies are available at the end of the exam.

Way 0

V	D	Tag	Word 0	Word 1	Word 2	Word 3
1	0	0x4	slli	add	lw	addi
1	0	0x4	sw	addi	addi	blt

Way 1

V	D	Tag	Word 0	Word 1	Word 2	Word 3
1	1	0x3, 0x2	M[0xC0], M[0x80]	M[0xC4], M[0x84]	M[0xC8], M[0x88]	M[0xCC], M[0x8C]

Instructions miss: ¼ for fetching first word in block, so total of 2
Data miss: 0xC0 and 0x80 both map to index 0 so they keep swapping each other out so you miss every time, so total of 8
So 10 misses out of 34 or 24 hits out of 34.

(F) (4 points) How many instruction fetch misses and data access misses occur during execution of the **entire program**?

Number of instruction fetch misses: _____2_____

Number of data access misses: _____8_____

(G) (1 point) What is the hit ratio for the execution of this program? You may leave your answer as a fraction.

Hit ratio: ____24/34 = 12/17_____

(H) (3 points) Ketchum wants to get the best performance out of his cache. He is considering modifying his current cache to **double the number of cache lines** while leaving all other parameters of the cache the same (2-way set associative and a block size of 4), thus doubling the total capacity of the cache. However, this new cache is a lot more expensive! Ketchum wants to choose the cheapest cache that maximizes the hit ratio. Which one should he choose? Explain your answer.

Circle One: Current Cache **New Cache**

Why?

Currently, the memory accesses overwrite each other every time since the indices overlap. If we have three bits to represent the index instead of two, the indices will no longer conflict, reducing the misses to 2/8.

Problem 5. Pipelined Processors (18 points)

Consider the loop below, which sums the values in a linked list. We run this code on a standard 5-stage RISC-V processor with full bypassing. Assume that branches are always predicted not taken and that branch decisions are made in the EXE stage. Assume that the loop repeats many times and it's currently in the middle of its execution.

In case you need them, extra pipeline diagrams are available at the end of the quiz.

```
loop: lw a1, 0(a0)
      lw a0, 4(a0)
      add a2, a2, a1
      bnez a0, loop
      mv a0, a2
      addi sp, sp, 8
      ret
```

(A) (7 points) Fill in the pipeline diagram for cycles 100-109, assuming that at cycle 100 the `lw a1, 0(a0)` instruction is fetched. **Draw arrows indicating each use of bypassing.** Ignore any cells shaded in gray.

	100	101	102	103	104	105	106	107	108	109
IF	lw	lw	add	bnez	bnez	mv	addi	lw	lw	add
DEC		lw	lw	add	add	bnez	mv	NOP	lw	lw
EXE			lw	lw	NOP	add	bnez	NOP	NOP	lw
MEM				lw	lw	NOP	add	bnez	NOP	NOP
WB					lw	lw	NOP	add	bnez	NOP

How many cycles does each iteration of the loop take? For each loop iteration, how many cycles are wasted due to stalls? How many are wasted due to annulments?

Number of cycles per loop iteration: 7

Number of cycles per loop iteration wasted due to stalls: 1

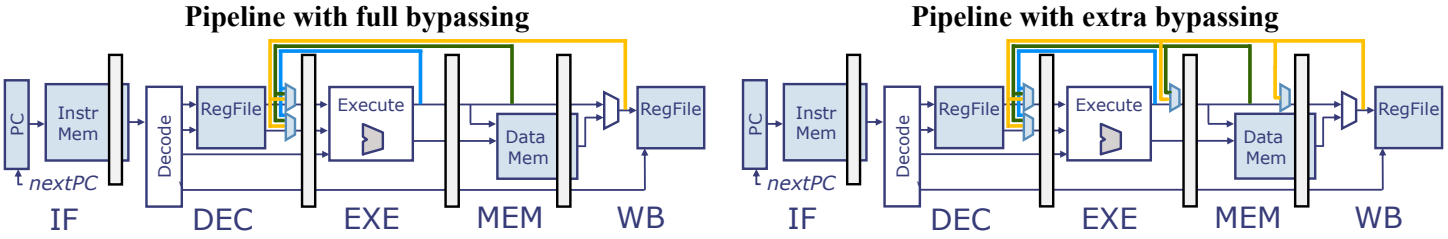
Number of cycles per loop iteration wasted due to annulments: 2

Ben Bitdiddle is looking to improve the performance of his processor. He observes that it is possible to save some cycles with extra bypass paths. Despite its name, *full* bypassing is not using as many bypasses as possible: it is bypassing to DEC from all later stages, but it is only bypassing to DEC. Sometimes, instructions do not need their inputs in the EXE stage, but at a later stage, and stalling on DEC wastes cycles.

To solve this, Ben proposes a new bypassing scheme called **extra bypassing**. In extra bypassing:

- There is a bypass path from each stage to DEC and all other stages between DEC and the source of the bypass. For example, the 5-stage pipeline has paths EXE->DEC, MEM->DEC, WB->DEC, MEM->EXE, WB->EXE, and WB->MEM. (Note full bypassing has only the first 3 of these paths.)
- As usual, each bypass path bypasses to the end of the stage (e.g., MEM->DEC bypasses to the end of DEC, right before the DEC-EXE pipeline register and after register reads, and MEM->EXE bypasses to the end of EXE, right before the EXE-MEM pipeline register and after the ALU).
- If an instruction depends on a value produced by an earlier instruction, and the value is still not available (on either the register file or a bypass path), the instruction *does not stall until the stage before where the value is needed*. For example, if a value is needed in EXE and has not been produced yet, the instruction will stall in DEC (as with full bypassing). But if the value is needed in MEM, the instruction will proceed from DEC to EXE, and stall in EXE until the value becomes available through a bypass path.
- Instructions capture missing inputs from bypass paths on the first opportunity they get, even if they are stalled for other reasons. (This ensures that, if an instruction advances past DEC, it always gets to bypass the correct input value before it needs it.)

The diagram below shows the bypass paths in full vs. extra bypassing.



(B) (4 points) Give an example 2-instruction sequence that incurs two cycles worth of stalls with full bypassing, but only one cycle with extra bypassing. Specify which bypass paths are exercised in each case.

Example 2-instruction sequence:

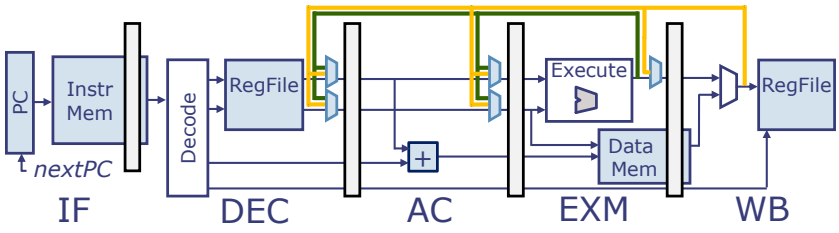
```
lw a1, 0(a0)
sw a1, 0(a2)
```

Bypass paths exercised with full bypassing: _WB->DEC_____

Bypass paths exercised with extra bypassing: __WB->EXE_____

To better leverage extra bypassing, Ben implements a new 5-stage pipeline, shown below, where **the ALU is placed in the fourth stage**:

- In this pipeline, the first two stages, IF and DEC, work as in the standard 5-stage pipeline; the third stage, AC, performs address calculation for loads and stores; the fourth stage, EXM, performs ALU operations and data accesses; and the fifth stage, WB, works as in the standard pipeline.
- Branches are predicted not-taken and resolved in the EXM stage.
- The pipeline has extra bypassing.



Consider again the code from part (A):

```
loop: lw a1, 0(a0)
      lw a0, 4(a0)
      add a2, a2, a1
      bnez a0, loop
      mv a0, a2
      addi sp, sp, 8
      ret
```

(C) (7 points) Fill in the pipeline diagram for cycles 100-109, assuming that at cycle 100 the `lw a1, 0(a0)` instruction is fetched. **Draw arrows indicating each use of bypassing.** Ignore any cells shaded in gray. **Recall that the Execute module is in the EXM pipeline stage.**

	100	101	102	103	104	105	106	107	108	109
IF	lw	lw	add	bnez	mv	addi	ret	lw	lw	add
DEC		lw	lw	add	bnez	mv	addi	NOP	lw	lw
EXE			lw	lw	add	bnez	mv	NOP	NOP	lw
MEM				lw	lw	add	bnez	NOP	NOP	NOP
WB					lw	lw	add	bnez	NOP	NOP

How many cycles does each iteration of the loop take? For each loop iteration, how many cycles are wasted due to stalls? How many are wasted due to annulments?

Number of cycles per loop iteration: __7_____

Number of cycles per loop iteration wasted due to stalls: __0_____

Number of cycles per loop iteration wasted due to annulments: __3_____

Problem 6. Processor Pipeline Performance (16 points)

You are designing a 5- stage pipelined (IF, DEC, EXE, MEM, WB) RISC-V processor with the same functionality in each stage that we have seen in lecture:

- IF: Initiate instruction fetch
- DEC: Decode instruction and gather source operands (stall if not available)
- EXE: Perform ALU operations and resolve branches
- MEM: Initiate data memory accesses
- WB: Write results back to register file

The processor has the following features:

- The instruction memory responds to every request in one cycle.
- The data memory responds to **cache hits** in one cycle. **The cache miss penalty of the data memory is 1 additional cycle.**
- **You have a cache with 4-word cache lines for the data memory.**
- The processor predicts that all branches are **TAKEN** and can start fetching the instruction at the target address on the cycle **immediately following** the branch.

This processor will spend most of its time executing the following loop:

```
loop:
0x100 lw x1, 0(x2)
0x104 lw x3, 0(x1) # address depends on value loaded above
0x108 add x4, x3, x1
0x10c sw x4, 0(x2)
0x110 addi x2, x2, 4
0x114 addi x6, x6, -1 # assume x6 initially is 128
0x118 bne x6, x0, loop
```

Assume that:

- `x2` has an initial value of `0x1000`, so on the first iteration, the `lw x1, 0(x2)` instruction accesses a memory address with cache block offset 0.
- The memory location accessed by the `lw x3, 0(x1)` instruction does not overlap with the locations accessed by `lw x1, 0(x2)` and `sw x4, 0(x2)`, and each access maps to a different cache line.

The code is repeated here for your reference:

```
loop:
0x100 lw x1, 0(x2)
0x104 lw x3, 0(x1) # address depends on value loaded above
0x108 add x4, x3, x1
0x10c sw x4, 0(x2)
0x110 addi x2, x2, 4
0x114 addi x6, x6, -1 # assume x6 initially is 128
0x118 bne x6, x0, loop
```

(A) (8 points) We are concerned with average performance and the loop runs for many iterations. Fill out the pipeline diagram below for **the 5th iteration of the loop.** Note that the processor predicts the branch to be taken. You may leave boxes blank to indicate NOP operations. **Draw arrows to indicate where a bypass path was used.** Then specify the number of cycles for the 5th iteration of the loop.

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
IF	lw	lw	add	add	add	add	sw	sw	sw	sw	addi	addi	bne	lw	lw	add
DEC		lw	lw	lw	lw	lw	add	add	add	add	sw	addi	addi	bne	lw	lw
EXE			lw				lw				add	sw	addi	addi	bne	lw
MEM				lw				lw				add	sw	addi	addi	bne
WB					lw	lw			lw	lw			add	sw	addi	addi

Number of cycles for 5th iteration of the loop: _____13_____

(B) (3 points) How many cycles does **the 6th iteration of the loop** take? The processor again predicts the branch to be taken.

Number of cycles for 6th iteration of the loop: _____12_____

(C) (3 points) On average, how many cycles does a loop iteration take?

(3*12 + 13)/4 = 49/4 = 12.25

Cycles per iteration: _____12.25_____

(D) (2 points) We now slightly modify the loop to use a multiply instruction. All other instructions are the same as before. **The execute stage of the multiplication takes 4 cycles.**

```
loop:
0x100 lw x1, 0(x2)
0x104 lw x3, 0(x1) # address depends on value loaded above
0x108 mul x4, x3, x1
0x10c sw x4, 0(x2)
0x110 addi x2, x2, 4
0x114 addi x6, x6, -1 # assume x6 initially is 128
0x118 bne x6, x0, loop
```

On average, how many additional cycles does an iteration of this new loop take versus your answer in part C?

Additional cycles per iteration: _____3_____

END OF QUIZ 2!