1	/14
2	/18
3	/16
4	/20
5	/16
6	/16

a

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

6.191 Computation Structures
Fall 2023

Quiz #2

Name Solutions	*Athena login name*	*Score*

Recitation section

☐ WF 10, 34-301 (Wendy)	☐ WF 2, 34-303 (Jessica)	☐ WF 12, 36-155 (Joey)
☐ WF 11, 34-301 (Wendy)	☐ WF 3, 34-303 (Jessica)	☐ WF 1, 36-155 (Joey)
☐ WF 12, 35-310 (Rona)	☐ WF 10, 34-302 (Helen)	☐ WF 1, 34-303 (Catherine)
☐ WF 1, 35-308 (Rona)	☐ WF 11, 36-112 (Helen)	☐ WF 2, 34-304 (Catherine)
		☐ 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 in Minispec (14 points)

Dr. Hoofenshmirtz is making a **WeatherInator** to cause mischief in the Tri-State Area. Upon receiving a WeatherForecast from the Tri-State Area Weather Service, the WeatherInator should generate weather conditions on the subsequent 4 cycles. As shown in the structs below, each **WeatherForecast** received by the WeatherInator contains an 8-bit temperature, and each **WeatherCondition** generated by the WeatherInator contains a Climate field and a Precipitation field.

```
typedef enum { Hot, Cold} Climate;

typedef enum { Sun, Rain, Snow} Precipitation;

typedef struct {
   Climate c;        // the Climate generated by the WeatherInator
   Precipitation p;  // the Precipitation generated by the WeatherInator

} WeatherCondition;

typedef struct {
    Bit#(8) temperature;  // the temperature of the WeatherForecast
} WeatherForecast;
```

(A) (6 points) Complete the implementation of a **WeatherInator** module in Minispec, which outputs a WeatherCondition generated by the WeatherInator in the get_weather method. The WeatherInator should operate as follows:

- If the forecasted temperature is 8'b00111100 or above, the WeatherInator should generate cold and snowy weather on the next 2 cycles followed by cold and rainy weather for 2 cycles.
- If the forecasted temperature is below 8'b00111100, the WeatherInator should generate hot and sunny weather on the next 2 cycles followed by hot and rainy weather for 2 cycles.
- If it has been more than 4 cycles since receiving a forecast, the WeatherInator should do nothing and the get_weather method should return Invalid.

You may assume the WeatherInator does not receive more than one forecast every 4 cycles. We have provided a **counter register which is set to 4 when a new valid forecast is received – use this to count down** to the next state transition. **Make sure you do not allow the counter to drop below zero.**

Fill in the Minispec code for the WeatherInator module below so that it conforms to the specifications described above. You may use all Minispec operators, including +, -, *, /, and %.

```
module WeatherInator;

    // WeatherForecast Received by the WeatherInator
    input Maybe#(WeatherForecast) in_forecast default = Invalid;
    Reg#(Climate) climate(Hot);
    Reg#(Precipitation) precipitation(Sun);
    Reg#(Bit#(3)) counter(0);

    rule tick;
        if(isValid(in_forecast)) begin
            let forecasted_temp = __fromMaybe(?, in_forecast).temperature;

            if (_____forecasted_temp _____ >= 8'b00111100) begin
                climate <= _________ Cold _____________;
                precipitation <= _________ Snow ____________;
            end else begin
                climate <= __________ Hot ___________;
                precipitation <= __________ Sun ____________;
            end
            counter <= 4;
        end else begin
            if (counter == ________3__________) begin
                precipitation <= _________ Rain __________;
            end
            counter <= _____(counter == 0)? 0: counter-1____;
        end
    endrule

    method Maybe#(WeatherCondition) get_weather;
        return (counter > _________0_________)?
               Valid(WeatherCondition{c: climate, p: precipitation})
               : Invalid;
    endmethod
endmodule
```

(B) (8 points) To ensure that your module behaves as expected, fill in the timing chart below with the register values and outputs for the first 9 cycles of the WeatherInator module. If the value is not known, write a "?" in the slot.

Cycle	0	1	2	3	4	5	6	7	8
isValid(in_forecast)	True	False	False	False	True	False	False	False	False
fromMaybe(?,in_forecast).temperature	8'b00111100	Invalid	Invalid	Invalid	8'b0	Invalid	Invalid	Invalid	Invalid
climate	Hot	Cold	Cold	Cold	Cold	Hot	Hot	Hot	Hot
precipitation	Sun	Snow	Snow	Rain	Rain	Sun	Sun	Rain	Rain
isValid(get_weather)	False	True	True	True	True	True	True	True	True
counter	0	4	3	2	1	4	3	2	1

Problem 2. Arithmetic Pipelines (18 points)

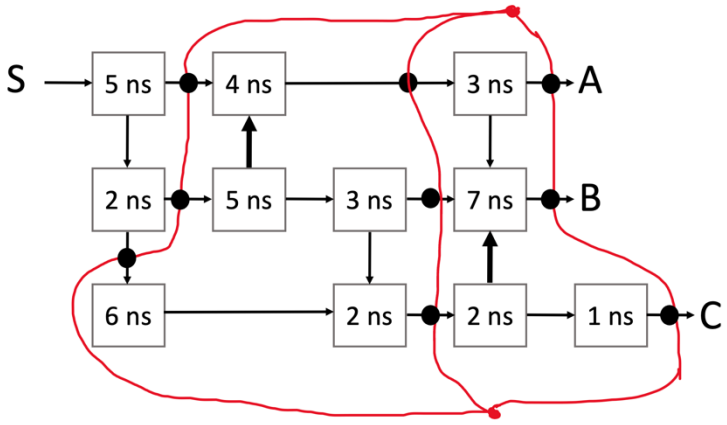
Your friendly neighborhood squid comes to you with a module named "FISH." This module has one input, **S**, and three outputs, **A**, **B**, and **C**. The squid tells you that this circuit functions, but its throughput is too low for it to feed his family. You decide to help, and try to pipeline the 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. **Note that invalid pipeline diagrams will receive 0 points.**

(A) (1 point) Based on the circuit shown in part (B), what is the propagation delay of the whole FISH circuit as-is, without pipelining?

t_{PD} (ns): ____26____

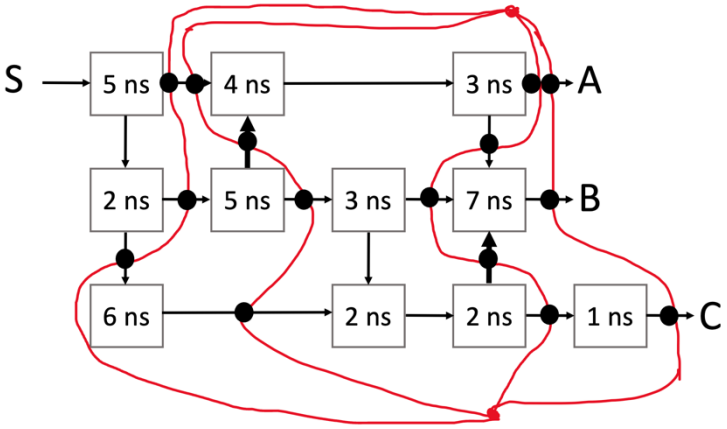
(B) (6 points) Show the **maximum-throughput 3-stage pipeline** using a minimal number of registers. What are the latency and throughput of the resulting circuit? In case you need them, extra copies of the circuit are available on the last page of the exam.



Latency (ns): ____30____

Throughput (ns^{-1}): ___1/10____

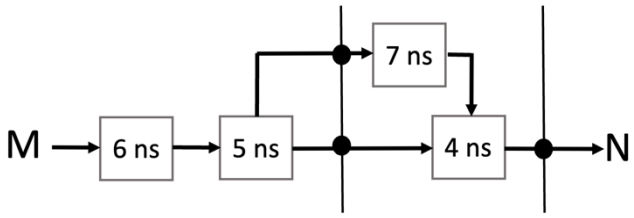
(C) (6 points) Show the **maximum-throughput pipeline** using a minimal number of registers. What are the latency and throughput of the resulting circuit? In case you need them, extra copies of the circuit are available on the last page of the exam.



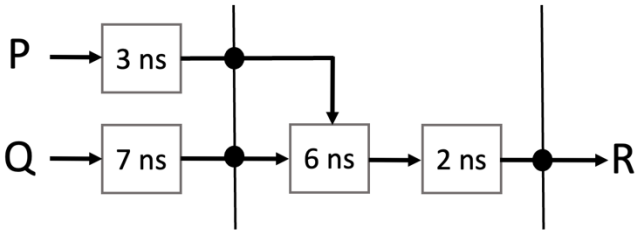
Latency (ns): ____28____

Throughput (ns^{-1}): ____1/7___

(D) (5 points) The squid is overjoyed at the improvements to the FISH module. Now, he presents to you two new modules: module "FRY" takes in input **M** and produces output **N**, and module "BOIL" takes in inputs **P** and **Q** and produces output **R**. Both modules are implemented with 2-stage pipelines, with registers denoted by the large black circles (●).



Implementation of Module FRY



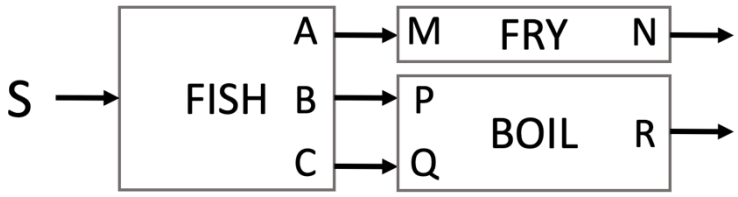
Implementation of Module BOIL

i. Given the implementations shown above, what are the throughputs of the modules?

Throughput of FRY (ns^{-1}): ____1/11____

Throughput of BOIL (ns^{-1}): ____1/8_____

The squid explains to you that he wants the "FRY" and "BOIL" modules to process the outputs of the module "FISH." That is, connect the output **A** of Module "FISH" to the input **M**, and connect the outputs **B** and **C** to the inputs **P** and **Q**, as shown below.



ii. When connecting module FISH to module FRY and module BOIL, you have two options for module FISH: your 3-stage pipeline or your maximum-throughput pipeline. The squid asks you to maximize throughput, and then minimize latency and number of registers used. Which implementation of module FISH would you use? What would the latency and throughput of the **combined** device be?

Module FISH implementation (circle one):

3-stage pipeline **Maximum-throughput pipeline**

Latency (ns): _____55_____

Throughput (ns^{-1}): ____1/11_____

Problem 3. Processor Implementation (16 points)

Queen Frog's domain has quintupled in size since the last census, and she has asked the Head of the Census, Matthew, to begin automating their work. As an alumnus of 6.191 and RISC-V enthusiast, Matthew has written a program in RISC-V assembly. He analyzes his code and realizes that his program repeats a sequence of multiple instructions that perform a *conditional load word*.

More specifically, his code contains instructions that repeatedly perform the following logic, which loads the contents located at memory address `reg[rs2]` into register `rd` if the value in register `rs1` isn't 0.

```
if (reg[rs1] != 0) begin
   reg[rd] <= Mem[reg[rs2]]
end
```

Matthew wants his code to run efficiently, so he decides to change the processor implementation in order to execute the above code in one cycle. He wants to add the following *conditional load word* instruction to the RISC-V ISA:

```
clw rd, rs1, rs2
```

Matthew has chosen to encode the `clw` instruction in the following way:

31...25	24...20	19...15	14...12	11...7	6...0
0000000	rs2	rs1	110	rd	0000011

(A) (1 point) Encode the following instruction as a 32-bit binary word (provide your answer in hexadecimal notation):

```
clw x4, x3, x1
```

0000|000 0|0001| 0001|1 110| 0010|0 000|0011
0x0011E203

Encoding in hexadecimal 0x: _____0x0011E203_____

Now, help Matthew modify the following processor implementation (shown on the next page) to support the new *conditional load word* instruction.

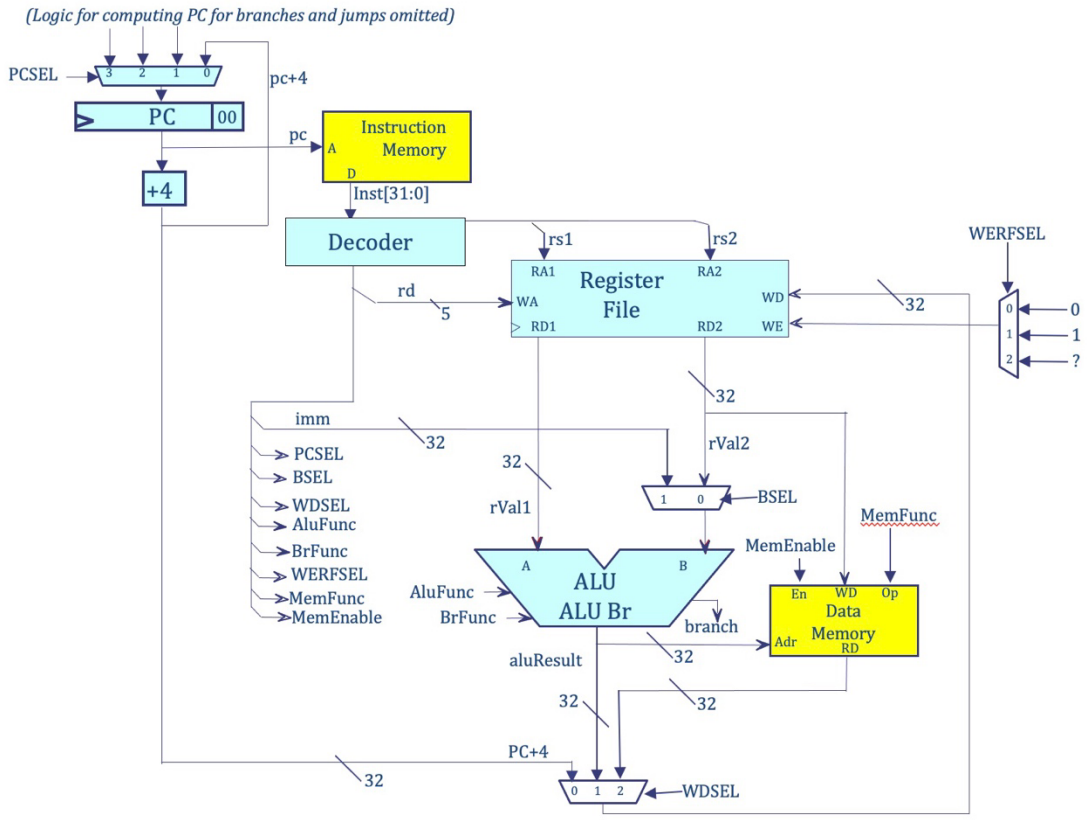
(B) (2 points) For each of the following signals, determine whether the mux being controlled by that signal needs an extra input to accommodate the new instruction. If so, indicate the name or value of **the signal that needs to be added as an input to the mux**. If not, indicate which existing value of **the mux control signal** (i.e., 0, 1, 2) is required to make the instruction work properly.

BSEL: **Needs new input?** **(YES)** **NO**

New input/Existing control signal: _____0_____

WDSEL: **Needs new input?** **YES** **(NO)**

New input/Existing control signal: ___2 (data memory output)____

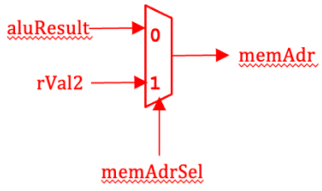


The WERFSEL mux was added to the datapath to support the addition of the `clw` instruction. The decoder will set WERFSEL = 2 for the `clw` instruction.

(C) (2 points) What should input 2 of the WERFSEL mux be connected to (i.e., what signal should '?' be replaced with?) to support the `clw` instruction?

WERFSEL input 2 signal:_____ branch _____

(D) (3 points) You want to ensure that when adding support for the new `clw` instruction, that the processor implementation continues to properly support all existing instructions. To do this, in addition the WERFSEL mux, is there a need for a second additional mux to be added to the datapath to support the `clw` instruction? If so, draw the mux below and clearly label the inputs and outputs of the mux. In addition, state where in the datapath this mux should be added. If you believe that no additional mux is needed then provide an explanation for your conclusion. If you decide that you do need an additional mux, then you may use the new mux output as a control signal for the following questions.



The memAdr connects to the Adr port of the data memory.

(E) (4 points) Now let's take a closer look at the Data Memory module. For each of the following input signals of the Data Memory module, decide what the value of the signal should be when executing the `clw` instruction. If the value of the signal doesn't matter, then put **N/A**. The possible values of the **MemFunc** signal are provided below.

MemFunc: Lw, Lh, Lhu, Lb, Lbu, Sw, Sh, Sb

MemEnable: _____1 (True)_____

WD (write data): _____ N/A _____

Adr (address): _____memAdr_____

MemFunc: _____Lw_____

(F) (2 points) Finally, decide for each of the following control signals what their values should be when executing the `clw` instruction. If the value of the signal doesn't matter, then put **N/A**. The possible values of each signal are provided below.

AluFunc: Add, Sub, And, Or, Xor, Slt, Sltu, Sll, Srl, Sra

BrFunc: Eq, Neq, Lt, Ltu, Ge, Geu

AluFunc: _____ N/A _____

BrFunc: _____Neq_____

(G) (2 points) If instead, Matthew's program had to repeatedly perform a *conditional store word*, could we have modified our processor **control signals** to support this instruction? **Explain your answer**. The code implementation of the *conditional store word* instruction is provided below:

```
if (reg[rs1] != 0) begin
   Mem[reg[rd]] <= reg[rs2]
end
```

Could we implement a conditional store word? YES (NO)

Explanation: The conditional store word instruction requires the processor to read 3 values from the register: register `rs1` for the branch comparison, register `rs2` for the write data of the memory request, and register `rd` for the write address of the memory request. Our register file only has 2 read ports, so we could not have implemented this new instruction.

Problem 4. Caches (20 points)

Assume that addresses and data words are 32 bits. Consider a **direct mapped cache** with **32 data words** with **1 word/cache line**.

(A) (2 points) Which address bits should the cache use for the cache index, tag field, and block offset. Write [X : X] if no bits are used.

Address bits for byte offset: A[1 : 0]

Address bits for cache index: A[6 : 2]

Address bits for tag field: A[31 : 7]

Address bits for block offset: A[X : X]

Now consider a **2-way set associative** cache with **8 sets** and a **block size of 2**. You will use this architecture for parts (B) – (E).

(B) (2 points) How will the following cache parameters change in the new 2-way cache relative to the direct mapped cache in part (A)? Please circle the best answer.

# of cache index bits: UNCHANGED … +1 … -1 … +2 … -2 … CAN'T TELL

# of tag field bits: UNCHANGED … +1 … -1 … +2 … -2 … CAN'T TELL

# of block offset bits: UNCHANGED … +1 … -1 … +2 … -2 … CAN'T TELL

(C) (10 points) Now analyze the performance of this cache (2-way set associative, 8 sets, block size of 2 words) using the following assembly program, which iterates through array `A` (base address `0x130`) and array `B` (base address `0x200`) and stores the result of `A[i]-B[i]` back into `A[i]`.

```
// Assume the following registers are initialized:
// x1 = 0 (loop index i)
// x2 = 4 (number of elements in arrays A and B)
. = 0x0 // The following code starts at address 0x0
loop:
    slli x3, x1, 2      // convert to byte offset
    lw x4, 0x130(x3)    // load value from A[i]
    lw x5, 0x200(x3)    // load value from B[i]
    sub x4, x4, x5      // x4 = A[i]-B[i]
    sw x4, 0x130(x3)    // store A[i]-B[i] into A[i]
    addi x1, x1, 1      // increment index
    blt x1, x2, loop    // loop 4 times

    unimp               // halt, all done
```

Assume the cache is empty before execution of this code (i.e., all valid bits are 0). Assume that the cache uses a least-recently used (LRU) replacement policy, and that **all cache lines in Way 0 are currently the least-recently used**. Fill in, or update, all the known values of the LRU bit, the dirty bit (D), the valid bit (V), the tag, and the data words **after one loop iteration** (after executing the `blt` instruction for the first time). For word fields, fill them in with the opcode if they are instructions (e.g., `blt`) or fill them in with the array element if they are data (e.g., `B[0]`). *You may assume that if V = 0 then D = 0.*

```
loop:  // The following code starts at address 0x0
    slli x3, x1, 2        // convert to byte offset
    lw x4, 0x130(x3)      // load value from A[i]
    lw x5, 0x200(x3)      // load value from B[i]
    sub x4, x4, x5        // x4 = A[i]-B[i]
    sw x4, 0x130(x3)      // store A[i]-B[i] into A[i]
    addi x1, x1, 1        // increment index
    blt x1, x2, loop      // loop 4 times

    unimp                 // halt, all done
```

Index	LRU after one loop iteration
0	0
1	1
2	1
3	1
4	0
5	0
6	1
7	0

Way 0 (After one loop iteration)

Index	V	D	Tag	Word 1	Word 0
0	1	0	0x0	lw	slli
1	1	0	0x0	sub	lw
2	1	0	0x0	addi	sw
3	1	0	0x0	unimp	blt
4	0	0			
5	0	0			
6	1	1	0x4	A[1]	A[0]
7	0	0			

Way 1 (After one loop iteration)

Index	V	D	Tag	Word 1	Word 0
0	1	0	0x8	B[1]	B[0]
1	0	0			
2	0	0			
3	0	0			
4	0	0			
5	0	0			
6	0	0			
7	0	0			

(D) (5 points) Fill out the table below with the number of instruction hits, instruction misses, data hits, and data misses for each of the four iterations of the loop.

	Instructions		**Data**	
	Hits	**Misses**	**Hits**	**Misses**
First loop iteration	**3**	**4**	**1**	**2**
Second loop iteration	**7**	**0**	**3**	**0**
Third loop iteration	**7**	**0**	**1**	**2**
Fourth loop iteration	**7**	**0**	**3**	**0**

(E) (1 point) What is the hit ratio for the execution of the **four loop iterations** (Note: do not include execution of the `unimp` instruction)? You may leave your answer as a fraction.

Hit Ratio: _____32/40_____

Problem 5. Pipelined Processors (16 points)

Ben Bitdiddle was adding two vectors in RISC-V assembly. He wrote the following code.

```
        mv x3, x4
loop:   lw x6, 0(x4)            // get vector 1 element
        lw x7, 0x300(x4)        // get vector 2 element
        add x7, x7, x6          // add the two element values
        sw x7, 0(x4)            // store result back into vector 1
        addi x4, x4, 4          // increment vector offset
        blt x4, x8, loop        // repeat loop for each element
        mv x10, x3
        addi sp, sp, 8          // sp is same as x2
        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** (i.e., we speculate that the branch is 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.

(A) (7 points) Fill in the pipeline diagram for cycles 100-110 assuming that at cycle 100 the `lw x6, 0(x4)` 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	110
IF	lw	lw	add	sw	sw	sw	addi	blt	mv	addi	lw
DEC		lw	lw	add	add	add	sw	addi	blt	mv	NOP
EXE			lw	lw	NOP	NOP	add	sw	addi	blt	NOP
MEM				lw	lw	NOP	NOP	add	sw	addi	blt
WB					lw	lw	NOP	NOP	add	sw	addi

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

Number of cycles per loop iteration: _____10_____

Number of cycles per loop iteration wasted due to stalls: _____2_____

Number of cycles per loop iteration wasted due to annulments: _____2_____

Ben Bitdiddle is looking to improve the performance of his processor. Ben notices that his vectors are very long and hence the branch will be taken almost always. So, he adds a branch predictor to his processor which detects the location at which the loop starts the first few times the loop runs. After that, in steady state, it always **predicts the branch as taken** to the start of the loop. In the EXE stage, it is determined whether the decision was correct or not and if it is incorrect, all instructions fetched from the start of the loop are flushed from the pipeline and the processor starts fetching from the instruction after the branch instruction.

(B) (4 points) Considering this optimization, fill in the pipeline diagram for cycles 200-204, assuming that at cycle 200 the `blt x4, x8, loop` instruction is fetched. **Draw arrows indicating each use of bypassing.** Ignore any cells shaded in gray.

	200	201	202	203	204
IF	`blt`	`lw`	`lw`	`add`	`sw`
DEC		`blt`	`lw`	`lw`	`add`
EXE			`blt`	`lw`	`lw`
MEM				`blt`	`lw`
WB					`blt`

How many cycles are saved per loop iteration due to this better branch prediction?

Number of cycles saved per loop iteration in steady state: _____2_____

Ben recalls that there are multiple ways to improve the overall speed of a computation. He explored optimizing the number of cycles in part B, but now he decides to make each cycle faster. He read in a 6.191 lecture that too much bypassing can be counterproductive as it can increase the propagation delay of the circuit. So, Ben takes his processor **from Part A** (without branch prediction), and removes the bypass from WB to DEC.

(C) (2 points) What is the number of cycles per loop iteration after the bypass from WB to DEC is removed?

Number of cycles per loop iteration:_____11_____

(E) (3 points) Ben synthesizes his circuit and notices that removing bypassing reduced his critical path from 100ns to 90ns. Is it beneficial for the performance of this program to remove bypassing?

YES NO **Justification:_**

Latency in Part A = Time taken per cycle x number of cycles = 100 ns (10 cycles/ns) = 1000 ns
Latency in Part C = Time taken per cycle x number of cycles = 90 ns (11 cycles/ns) = 990 ns, so performance improves if the WB to DEC bypass is removed.

Problem 6. Pipelined Processor Performance (16 points)

Ben Bitdiddle has recently gotten into an unhealthy obsession with the Fibonacci sequence. He writes the following loop in RISC-V assembly that calculates the first 1000 Fibonacci numbers (he doesn't have the patience for more than 1000 numbers) and stores them in memory.

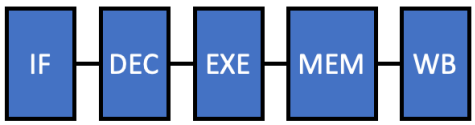
```
        addi t0, zero, 1000  // t0 = 1000
        addi a1, zero, 1     // fib(1) = 1
        sw a1, 0(t1)         // base memory address stored in t1
        addi a2, zero, 1     // fib(2) = 1
        addi t1, t1, 4
        sw a2, 0(t1)

fib:    add a3, a2, a1       // fib(n) = fib(n-1) + fib(n-2)
        addi t1, t1, 4
        sw a3, 0(t1)
        mv a1, a2            // update fib(n-2) to fib(n-1)
        mv a2, a3            // update fib(n-1) to fib(n)
        addi t0, t0, -1
        bnez t0, fib

        xor t2, t3, t4
        not t5, t3
        and t6, t3, t4
```

Ben doesn't want to spend money on a pipelined processor and thinks that a single-cycle RISC-V processor that he found on the black market can get the job done. Alice disagrees and wants to convince Ben to invest in a pipelined processor. Help Alice come up with a convincing argument on why Ben should purchase a pipelined processor.

(A) (1 point) Ben's single-cycle processor has the following propagation delays for each piece of combinational logic.



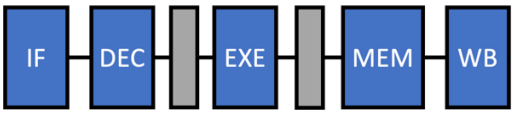
IF	3 ns
DEC	4 ns
EXE	6 ns
MEM	5 ns
WB	3 ns

What is the minimum clock period of this processor for the Fibonacci program to execute correctly? How long does the single-cycle processor take to execute **1 iteration** of the loop?

Processor minimum clock period: _____3 + 4 + 6 + 5 + 3 = 21 ns ____

Time to execute 1 iteration of the loop: _____7(21 ns) = 147 ns_____

(B) (8 points) Alice wants to ease Ben into the world of pipelines, so she designs a simple 3-stage pipelined processor. The three stages are 1) fetch and decode, 2) execute, and 3) memory access and write back. Her pipeline has no bypassing, **branches are always predicted as not taken** with branch annulment available, and branch decisions are available in the EXE. Each component has the same propagation delays as Ben's single-cycle processor (included here for your convenience).



IF	3 ns
DEC	4 ns
EXE	6 ns
MEM	5 ns
WB	3 ns

Fill in the pipeline diagram for cycles 100-113, assuming that at cycle 100 the `add a3, a2, a1` instruction is fetched and registers `a2` and `a1` are up to date.

```
fib:  add a3, a2, a1      // fib(n) = fib(n-1) + fib(n-2)
      addi t1, t1, 4
      sw a3, 0(t1)
      mv a1, a2           // update fib(n-2) to fib(n-1)
      mv a2, a3           // update fib(n-1) to fib(n)
      addi t0, t0, -1
      bnez t0, fib
      xor t2, t3, t4
      not t5, t3
      and t6, t3, t4
```

Cycle	100	101	102	103	104	105	106	107	108	109	110	111	112	113
IF DEC	add	addi	sw	sw	sw	mv	mv	addi	bnez	bnez	bnez	xor	add	addi
EXE		add	addi	NOP	NOP	sw	mv	mv	addi	NOP	NOP	bnez	NOP	add
MEM WB			add	addi	NOP	NOP	sw	mv	mv	addi	NOP	NOP	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: _____12_____

Number of cycles per loop iteration wasted due to stalls: _____4_____

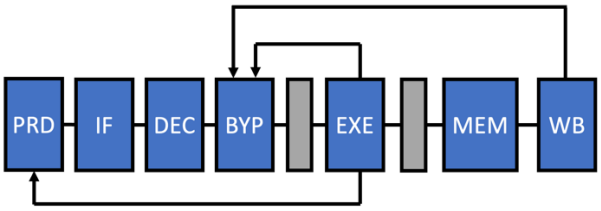
Number of cycles per loop iteration wasted due to annulments: _____1_____

What is the minimum clock period for this processor? How long does 1 iteration of the loop take?

Processor minimum clock period: _____8 ns (MEM+WB)_____

Time to execute 1 iteration of the loop: _____12(8 ns) = 96 ns_____

(C) (7 points) Ben still doesn't seem convinced, so Alice decides to add bypass and branch prediction to her pipeline. Her pipeline has full bypassing to IFDEC (i.e., EXE->IFDEC and MEMWB->IFDEC), **branches are predicted as taken** with branch annulment available, and branch decisions are available in the EXE stage. She adds bypass and prediction logic with the following propagation delays:



IF	3 ns
DEC	4 ns
EXE	6 ns
MEM	5 ns
WB	3 ns
PRD	**1 ns**
BYP	**1 ns**

Fill in the pipeline diagram for cycles 100-109, assuming that at cycle 100 the `add a3, a2, a1` instruction is fetched and registers `a2` and `a1` are up to date. **Indicate all bypass paths with vertical arrows.**

```
fib:  add a3, a2, a1      // fib(n) = fib(n-1) + fib(n-2)
      addi t1, t1, 4
      sw a3, 0(t1)
      mv a1, a2           // update fib(n-2) to fib(n-1)
      mv a2, a3           // update fib(n-1) to fib(n)
      addi t0, t0, -1
      bnez t0, fib
      xor t2, t3, t4
      not t5, t3
      and t6, t3, t4
```

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

How many cycles does each iteration of the loop take?

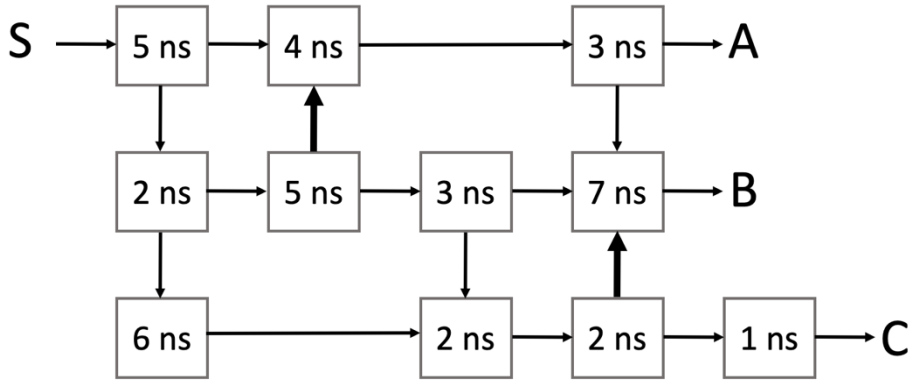
Number of cycles per loop iteration: _____7_____

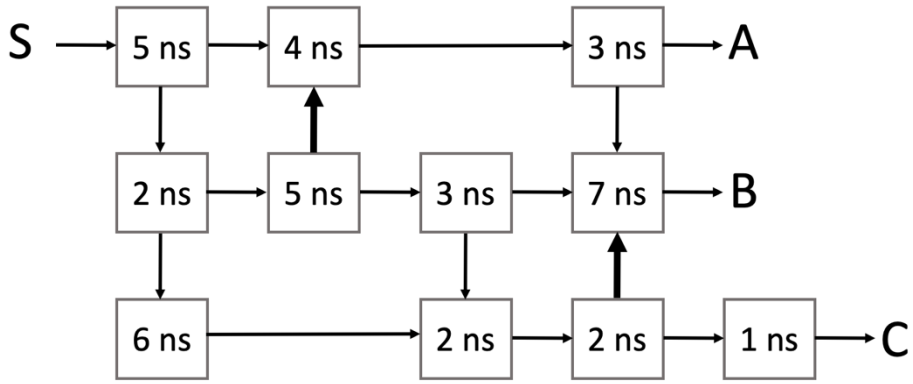
Assuming the delays provided for the bypass and branch prediction components, what is the minimum clock period for this processor? How long does 1 iteration of the loop take?

Processor minimum clock period: _____15 ns (EXE+PRD+ IF+DEC+BYP)_____

Time to execute 1 iteration of the loop: _____7(15 ns) = 105 ns_____

Extra diagrams for problem 2:





Extra diagrams for problem 5:

	100	101	102	103	104	105	106	107	108	109	110
IF	`lw`										
DEC											
EXE											
MEM											
WB											

	200	201	202	203	204
IF	`blt`				
DEC					
EXE					
MEM					
WB					

Extra diagrams for problem 6:

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

Cycle	100	101	102	103	104	105	106	107	108	109
IF DEC	add									
EXE										
MEM WB										

END OF QUIZ 2!