

University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Summer 2019 Instructors: Branden Ghena, Morgan Rae Reschenberg, Nicholas Riasanovsky 2019-08-15

CS61C FINAL

<i>Last Name (Please print clearly)</i>				
<i>First Name (Please print clearly)</i>				
<i>Student ID Number</i>				
<i>Circle the name of your Lab TA</i>	Ayush Maganahalli	Chenyu Shi	Gregory Jerian	Jenny Song
	John Yang	Lu Yang	Ryan Searcy	Ryan Thornton
<i>Name of the person to your: Left Right</i>				
<i>All my work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS61C who haven't taken it yet. (please sign)</i>				

Instructions

- This booklet contains **30** pages including this cover page. **The back of each page of this exam is blank and can be used for scratch work, but will not be graded.**
- Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats and headphones. Place *everything* except your writing utensil(s), cheat sheet(s), and beverage underneath your seat.
- You have 170 minutes to complete this exam. The exam is closed book: no computers, tablets, cell phones, wearable devices, calculators, or cheating. You are allowed three pages (US Letter, double-sided) of *handwritten* notes.
- There may be partial credit for incomplete answers; write as much of the solution as you can.
- Please write your answers within the boxes and blanks provided within each problem!

Question	1	2	3	4	5	6	7	8	9	10	11	Total
Possible Points	19	8	24	25	18	12	18	10	17	8	21	180

If you have the time, **feel free to doodle on the front page!**

Question 1: Potpourri - 19 pts

Select which stage of CALL (compiler, assembler, linker, loader) is responsible for the following actions:

1. Provides the address printed by: `printf("%p", "cs61c")`.

- (A) Compiler (B) Assembler (C) Linker (D) Loader

2. Places the string "cs61c" in RAM.

- (A) Compiler (B) Assembler (C) Linker (D) Loader

3. Removes all pseudo instructions.

- (A) Compiler (B) Assembler (C) Linker (D) Loader

4. Can always provide the correct immediate value when translating all `l1a` instructions.

- (A) Compiler (B) Assembler (C) Linker (D) Loader

5. Can always provide the correct immediate value when translating all `l1i` instructions.

- (A) Compiler (B) Assembler (C) Linker (D) Loader

6. Stage most often responsible for loop unrolling.

- (A) Compiler (B) Assembler (C) Linker (D) Loader

You propose a new 16 bit floating point number. It has:

- 1 sign bit
- 11 exponent bits
- 4 significand bits
- A bias of 1023
- All other rules consistent with IEEE 754 floating point.

7. Represent 4.75 in our new floating point scheme

Sign:0b _____

Exponent: 0b _____

Significand:0b _____

8. How many numbers does our floating point scheme represent in the range $[0, 1)$ (the range 0 to 1, where 0 is included and 1 is not)? For this question assume -0 is not in this interval. You may leave your answer unsimplified.

_____ numbers

Now let's compare to a 16 bit two's complement number.

9. Which can represent a larger number (ignore infinities)?

Ⓐ Our Floating Point Scheme

Ⓑ Two's Complement

10. Which scheme represents more numbers in the range [1, 64)?

Ⓐ Our Floating Point Scheme

Ⓑ Two's Complement

11. Which scheme represents more numbers in the range [64, 128)?

Ⓐ Our Floating Point Scheme

Ⓑ Two's Complement

12. You are doing an internship project for a big tech company and need to speed up your program. You find that your program calls easily parallelizable code 40% of the time, so you use `#pragma omp parallel for` to split up that work into 8 threads. You also implement SIMD for other sequential functions run in a single thread, which are called 50% of the time. If initially your program takes 20s to run and you want it to take 10s to run, how much speedup is needed from your SIMD functions to achieve it? Leave your answer as a fraction

SIMD Speedup: _____

13. The following OpenMP code will properly sum an input array:

```
// Sums the elements of the array
void sum_array(int* array, unsigned int size) {
    int sum = 0;
    #pragma omp parallel for
    for (int i=0; i<size; i++) {
        sum += array[i];
    }
}
```

- (A) Always (B) Sometimes (C) Never

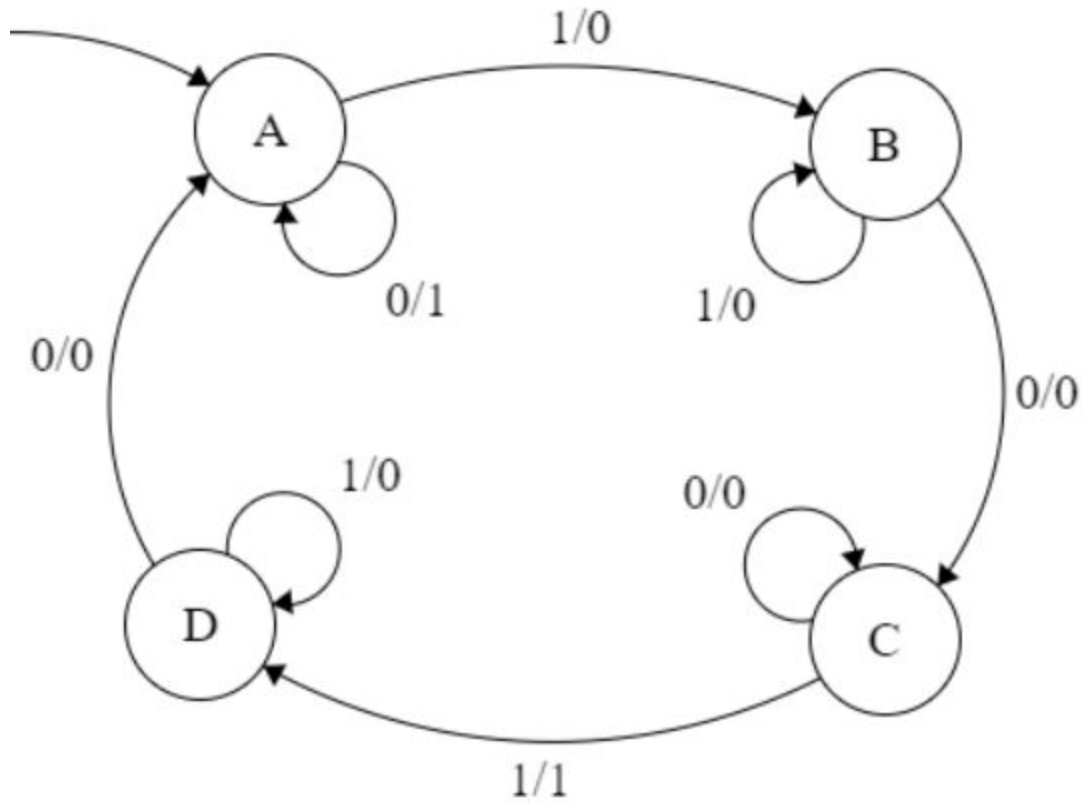
14. The following OpenMP code will properly copy an input array into an output array:

```
// Copies an input array into an output array
void sum_array(int* array, int* output, unsigned int size) {
    #pragma omp parallel for
    for (int i=0; i<size; i++) {
        output[i] = array[i];
    }
}
```

- (A) Always (B) Sometimes (C) Never

Question 2: FSM - 8 pts

FSM Question For the following Finite State Machine, fill out the remainder of the table.



Input	-	1	0	0	1	1	0	0	0
Next State	A								
Output	-								

Question 3: C Coding - 24 pts

In this question we are going to implement a double-ended queue data structure, which is a data structure in which you can insert to either end. To do so we will allocate a single array to store all data contiguously, but because we need to append to both ends we will implement our array as a circular buffer. A circular buffer is a way of wrapping around an array while maintaining the ordering. For example imagine the following implementation where we append to the left of our queue with an initial value of 3.

```
// Initially q->data = [garbage, 3, garbage];
append_left (q, 2) // Now q->data = [2, 3, garbage]
append_left (q, 1) // Now q->data = [2, 3, 1];
// We keep track of the order with additional struct fields.
```

Notice that we fill the array entirely and move from one end to the other when we run out of space. To implement our queue we have provided a struct and a constructor on the handout.

1. Implement `print_reverse_dqueue` which prints each valid element in the array from the end to the front (left to right) with each element on a newline. You may not need all lines.

```
#include <stdio.h>
void print_reverse_dqueue (int_dqueue_t* q) {
    for ( _____ ) {
        int location = _____;
        if ( _____ ) {
            _____;
        }
        printf ( _____, _____);
    }
}
```

One issue that complicates our queue is what happens when we need to resize. With other data structures we can use `realloc`, but imagine we have the following full data where the actual order of the data is 1, 2, 3, 4.

```
q->data = [3, 4, 1, 2];
```

If we were to reallocate the queue to size we would then get:

```
q->data = [3, 4, 1, 2, garbage, garbage, garbage, garbage]
```

Now if we only `realloc` we can't maintain our ordering, so we need to do some extra work when resizing.

2. Implement `expand_buffer` which takes in a queue that is **full** and reallocs circular buffer while maintaining the previous ordering. You can assume all calls to `realloc` succeed and you may not need all lines.

Recall the header for `realloc` is:

```
void* realloc (void* ptr, int size);
```

Hint: You probably only want to change either `left_location` or `right_location`, not both

```
#include <stdlib.h>
void expand_buffer (int_dqueue_t* q) {
    q->allocated_size *= 2;
    q->data = realloc (_____, _____);
    for (_____ ) {
        _____;
    }
    _____
    _____
    _____
    _____
}
```


Question 4: RISC-V - 25 pts

1. Translate the body of mystery from the handout from C to RISC-V. Assume that a correct prologue and epilogue that adheres to the calling convention learned in class is provided. You may not need all lines. You may only use registers a0-a7, t0-t5, s0-s4, ra, and sp.

```
.data
stringPrint: .asciiz "%s\n"
intPrint:    .asciiz "%d\n"
.text
mystery:    mv s0 a0 #src
            mv s1 a1 #dest
            mv s2 a2 #length
            add s3 x0 x0 #charSum
            add s4 x0 x0 #encryptCircular

LoopStart:  bge _____ LoopEnd
            # Load source value once
            _____
            _____

            # Compute all adds
            _____
            _____
            _____

            # Store in dest
            _____
            _____

            # Make the first call to printf
            _____
            _____

            jal printf
            addi s4 s4 1
            j LoopStart

LoopEnd:    _____
            _____

            jal printf
            _____
            _____

            jal printf
```


Question 5: Data-Level Parallelism - 18 pts

Help John write a program that will take the **norm** of an array using SIMD instructions. The norm of an array is defined as the square root of the sum of the squared elements of the array. In other words, the norm is equal to $\sqrt{\text{arr}[0]^2 + \dots + \text{arr}[n-1]^2}$, where n is the size of the array.

To make this calculation fast, we will use SIMD instructions. However, instead of the nonsense Intel SIMD instructions, you can (and must) use any of the functions on your handout. Fill in the following C code. You may not need all lines.

```
// Returns the norm of ARR, which is an array of length SIZE
```

```
double norm(double arr[], unsigned int size) {
    simd_t sum_vec = simd_set_value(0);

    // SIMD Code
    for (int i = 0; _____; _____) {

        simd_t temp_vec = _____;
        _____;
        _____;
    }

    double sum_arr[_____];
    _____;

    double ret_val = _____;

    // Tail Case
    for (int i = _____; _____; _____) {
        _____;
        _____;
    }
    // Square root
    return sqrt(_____);
}
```

Question 6: RAID + ECC - 12 pts

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Encoded data bits	p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15
Parity bit coverage	p1	X		X		X		X		X		X		X		X		X		X
	p2		X	X			X	X			X	X			X	X			X	X
	p4				X	X	X	X				X	X	X	X					X
	p8								X	X	X	X	X	X	X					
p16																X	X	X	X	X

For the following ECC questions, assume that the parity is calculated using *ODD* parity (ie. the opposite of the even parity we learned in lecture). Use the above Hamming Code table to locate parity and data bits within a codeword string.

- Given the following string of data bits (from left to right), what should our parity bits be? If a parity bit is unnecessary for this data string, write N/A in the blank.

Data: 0 0 1 1 0 1 0 1

P1 = _____ P2 = _____ P4 = _____ P8 = _____ P16 = _____

- We store the data in memory and read it out moments later as 01110101. The underlined bit differs. When we re-do our parity calculations, which bits can we expect to be incorrect due to this error? Mark all that apply.

P1 P2 P4 P8 P16

- Given a data string that is 97 bits long, how many parity bits must we use to provide single error detection and single error correction?

_____ parity bits

For the questions below, identify the type of disk system being described, both or neither.

4. Provides Fault Tolerance. If a disk suffers a failure and the data on it is lost, it can be recovered.

- A Striping B Mirroring C Both D Neither

5. Provides a performance improvement (i.e. faster read and write operations)

- A Striping B Mirroring C Both D Neither

6. Requires more than one disk or storage device to implement in practice.

- A Striping B Mirroring C Both D Neither

7. RAID 0

- A Striping B Mirroring C Both D Neither

8. RAID 1

- A Striping B Mirroring C Both D Neither

9. True or False: "RAID 0 is more capable of tolerating disk failures than RAID 1"

- A True B False

Question 7: Caches - 18 pts

Dynamic Programming is an algorithm used to reduce the runtime of recursions by storing intermediate results to an array. `fib_dynamic` below is an example of calculating Fibonacci numbers using dynamic programming:

```
int fib_dynamic(int number) {
    /* Declare an array to store Fibonacci numbers. */
    int f[number+1];
    int i;

    /* 0th and 1st number of the series are 0 and 1*/
    f[0] = 0;
    f[1] = 1;

    for(i = 2; i <= number; i++) {
        /* Add the previous 2 numbers in the series
           and store it */
        f[i] = f[i-1] + f[i-2];
    }
    return f[number];
}
```

We have a **2-way set associative cache with 256 total bytes and 16 bytes per block**. The cache is write back with a write allocate on miss policy. Assume `sizeof(int) == 4`, `sizeof(long) == 8`, and that `f` is at a **block-aligned address**. We also have **1 MiB** of physical memory and no virtual memory. Assume that for all questions the **cache begins cold** and that all questions are independent. You should assume `i` and `number` are optimized into registers.

1. How many bits are in the tag, index and offset fields?

Tag: _____

Index: _____

Offset: _____

2. What is the hit rate if we run fib_dynamic(32)?

HR: _____

3. Would our hit rate increase, decrease or stay the same if instead we had a write through cache with a no write allocated on miss policy?

Ⓐ Increase

Ⓑ Decrease

Ⓒ Stay the same

Noticing that int can only accommodate the first 47 Fibonacci number without overflowing, we change the type of array f in which we store the intermediate result to be long f[n+1] instead. Assume our cache is still **2-way set associative cache with 256 total bytes and 16 bytes per block and write back with a write allocate miss policy**.

4. What is the hit rate if we run fib_dynamic(64)?

HR: _____

5. What is the smallest value of number that causes a capacity miss? Select N/A if there is never a capacity miss.

Ⓐ 8 Ⓑ 16 Ⓒ 32 Ⓓ 64 Ⓔ 128 Ⓕ 256 Ⓖ 512 Ⓗ 1024 ⓘ N/A

6. What is the smallest value of number that causes a conflict miss? Select N/A if there is never a conflict miss.

Ⓐ 8 Ⓑ 16 Ⓒ 32 Ⓓ 64 Ⓔ 128 Ⓕ 256 Ⓖ 512 Ⓗ 1024 ⓘ N/A

Question 8: Spark - 10 pts

Map-Reduce & Spark

We are given a dataset from a gym and we want to find the **average use time for each type of machine**. Fill in the blanks for the python pseudocode using map-reduce ideas. (Your specific python syntax is not important as long as your answer is clear.) Assume each machine works independently and there is no time overlap for one machine.

Sample Input (MachineType, MachineID, start_time, end_time):

Treadmill 1 8:00 8:30

Treadmill 1 8:32 8:42

Treadmill 2 10:05 10:25

Seated_overhead_press 1 14:05 14:17

Sample Output (MachineType, average_use_time):

(Treadmill, 30)

(Seated_overhead_press, 12)

Explanation: Treadmill 1 is used for 40 minutes and Treadmill 2 is used for 20 minutes, so the average Treadmill use time is 30 minutes.

Refer to the Spark section of the handout for a list of helper functions you can use.

The code to fill in is on the next page.

```

def parseInput(lines):
    result = []
    for line in lines:
        tokens = line.split(" ")
        timediff = time_elapse(_____, _____)
        result.append(tuple(tuple(tokens[0], tokens[1]), timediff))
    return result

```

```

def count_time(v1, v2):

    return _____

```

```

def group_by_type(k, v):

    return _____

```

```

def count_ids(v1, v2):

    return _____

```

```

def average(k, v):

    return _____

```

You do not need to edit this function, but it may be helpful to reference
Assume Spark has been properly configured and the return is written to a file

```

def main(rsfData):
    out = rsfData.flatMap(parseInput) \
        .reduceByKey(count_time) \
        .map(group_by_type) \
        .reduceByKey(count_ids) \
        .map(average)
    return out

```

Question 9: Datapath - 17 pts

Now that you've (almost) finished CS61C, you decide to spend your free time beefing up your favourite project: our RISC-V CPU! After the quick work of changing your datapath from a 2-stage to 5-stage pipeline, you're interested in adding forwarding.

1. Before adding forwarding logic, we need to change our CPU to detect hazards that can be solved by forwarding. Fill in the blanks in the following statement to describe which instruction fields should be compared to identify forwarding cases. You may select more than one option if necessary.

Assume our pipeline currently contains the following instructions:

IF	ID	EX	MEM	WB
Inst 1	Inst 2	Inst 3	Inst 4	Inst 5

We need to check for equality between the __A__ register(s) of inst(s) __B__ and the __C__ register(s) of inst 3.

A) source destination

B) 1 2 3 4 5

C) source destination

2. Feeling a little overwhelmed with forwarding, you try to break the problem down into small pieces. First you consider the case where we need to forward from our ALU output to the next EX stage as an argument:

```

addi t0 t1 10      IF   ID   EX   MEM   WB
add  s0 t0 t3      IF   ID   EX   MEM   WB
    
```

Assume you've been able to implement the logic described in part 1, and this logic exists as a control bit EXEXFwd, which is 1 when we should forward from EX to EX and 0 otherwise.

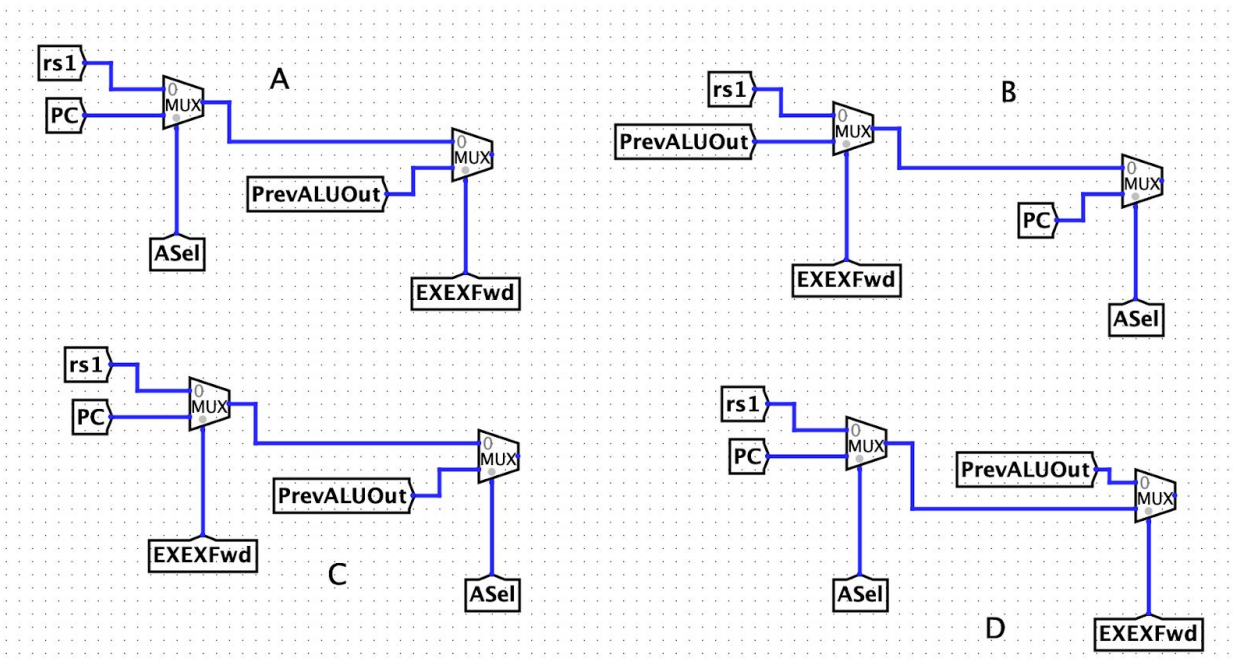
Which ASel model correctly uses this new control bit? (circle the correct choice)

(A) A

(B) B

(C) C

(D) D



3. Given the change to ASel you picked above, will the following chunk of code execute correctly? Why or why not?

```

slli t0 t1 10      IF   ID   EX  MEM  WB
add  s0 t3 t0      IF   ID   EX  MEM  WB

```

- Ⓐ Yes, it will execute correctly Ⓑ No, it will not execute correctly

4. After some time, you get your EX to EX forwarding working correctly, but you start to realise you need to forward from *other* locations to EX as well (ie. MEM to EX):

```

addi t0 t1 6      IF   ID   EX   MEM  WB
slli t0 t0 2      IF   ID   EX   MEM   WB
slti t0 t0 8      IF   ID   EX  MEM   WB

```

You'd like to chain your EXEXFwd sub-circuit together with your other forwarding logic such that changes to a register prioritize forwarding from the most recent instruction. Order the following sub circuits from 1 to 3 with 1 being leftmost (lowest priority) and 3 being rightmost (highest priority) such that the subcircuits will always output the most current value to forward.

___ WB to EX

___ EX to EX

___ MEM to EX

5. You finish installing hardware for forwarding EX to EX, MEM to EX, and WB to EX, but find this isn't sufficient to allow all combinations of instructions to execute correctly in your five stage pipeline; you still experience load hazards. Answer the following questions to prove why forwarding is impossible for load hazards.

lw	t0	0(a0)	IF	ID	<u>EX</u>	MEM	WB	
add	t3	t0		IF	ID	<u>EX</u>	MEM	WB

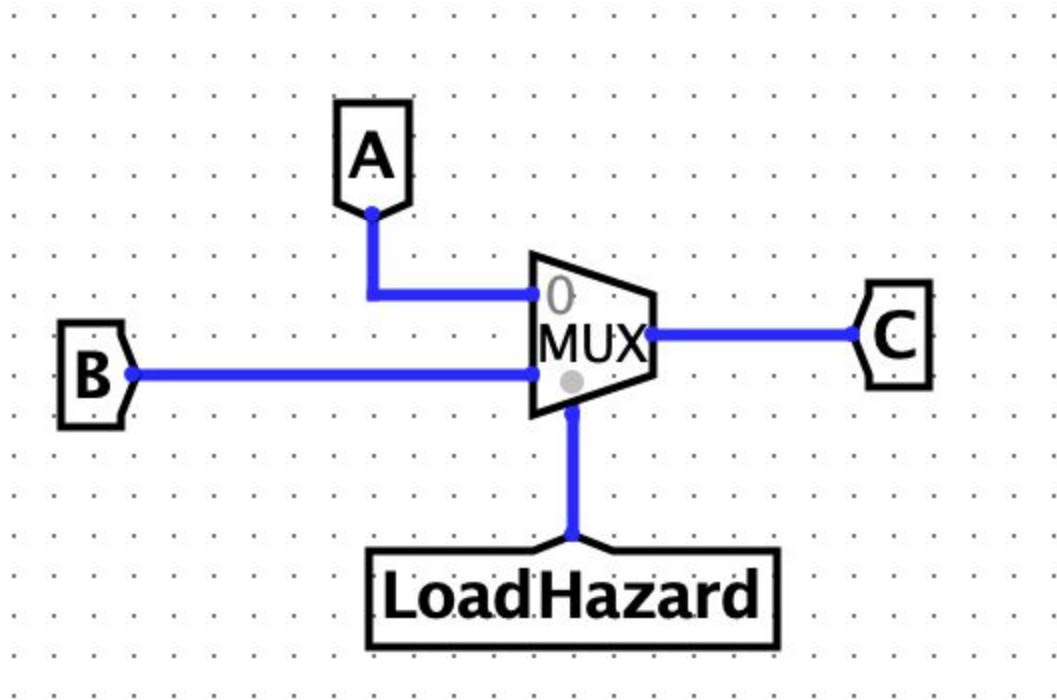
a. What is the *earliest* stage at which the load data is ready/available to forward? Circle one stage.

lw	t0	0(a0)	IF	ID	EX	MEM	WB
----	----	-------	----	----	----	-----	----

b. Where is the *latest* stage by which the load data could be consumed/received from forwarding? Circle one stage.

add	t3	t0	IF	ID	EX	MEM	WB
-----	----	----	----	----	----	-----	----

6. We can detect a load hazard after we have fetched the dependent instruction (add, in our previous example), and so this is the earliest point at which we can stall. We'd like to add a MUX between our ID and IF stages. This MUX should current instruction to a NOP if a load hazard exists. Assume we have a new control bit LoadHazard which is 1 when a load hazard is present and 0 otherwise. Where should we connect tunnels A, B, and C? Select one option for each letter.



A:

- Ⓐ IMEM output Ⓑ ID input (RegFile parser input) Ⓒ PC Ⓓ NOP instruction

B:

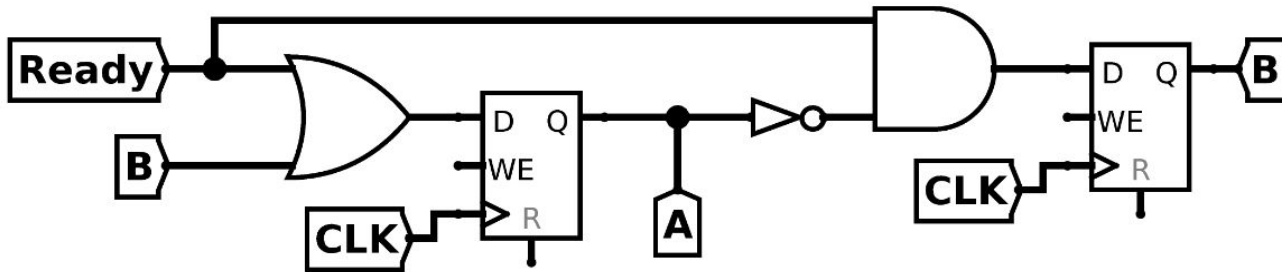
- Ⓐ IMEM output Ⓑ ID input (RegFile parser input) Ⓒ PC Ⓓ NOP instruction

C:

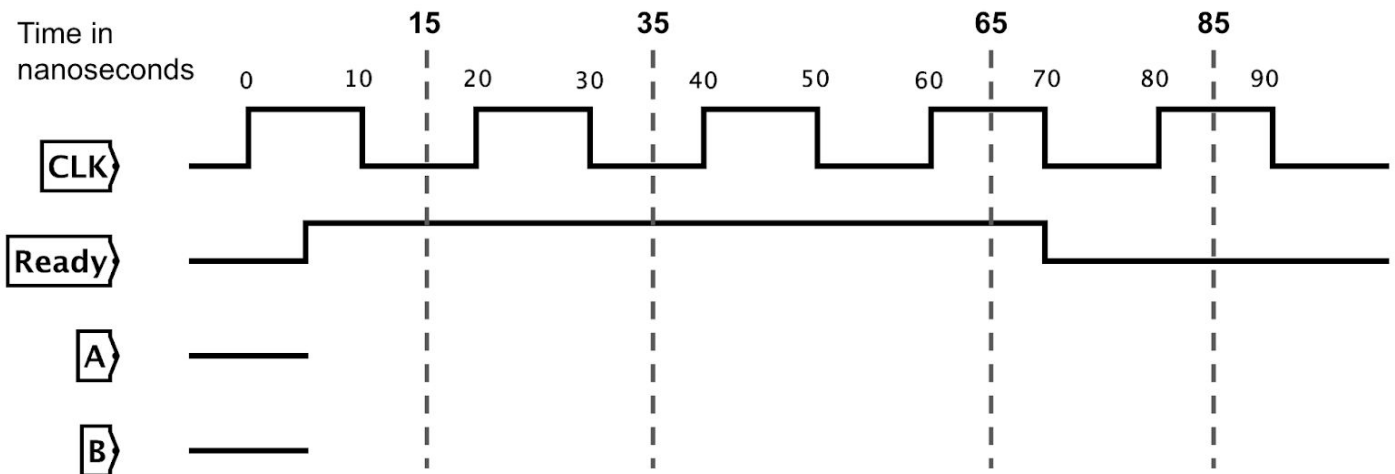
- Ⓐ IMEM output Ⓑ ID input (RegFile parser input) Ⓒ PC Ⓓ NOP instruction

Question 10: Digital Logic - 8 pts

Determine the value of the signals A and B from the following circuit given the waveform diagram below. All registers are rising-edge triggered, have a setup time of 1 ns, a hold time of 1 ns, and a clock-to-q delay of 3 ns. The propagation delay through AND and OR gates is 4 ns, and the propagation delay through NOT gates is 2 ns.



Both output signals start low while the value of Ready changes as shown. You may fill out the waveform diagram if you find it helpful, but you will only be graded on your answers to the multiple choice questions which begin on the next page.



What is the value of the output signals at time **15 ns**? (circle the correct answer for each signal)

1. **Signal A:** A High B Low C Undefined

2. **Signal B:** A High B Low C Undefined

What is the value of the output signals at time **35 ns**? (circle the correct answer for each signal)

3. **Signal A:** A High B Low C Undefined

4. **Signal B:** A High B Low C Undefined

What is the value of the output signals at time **65 ns**? (circle the correct answer for each signal)

5. **Signal A:** A High B Low C Undefined

6. **Signal B:** A High B Low C Undefined

What is the value of the output signals at time **85 ns**? (circle the correct answer for each signal)

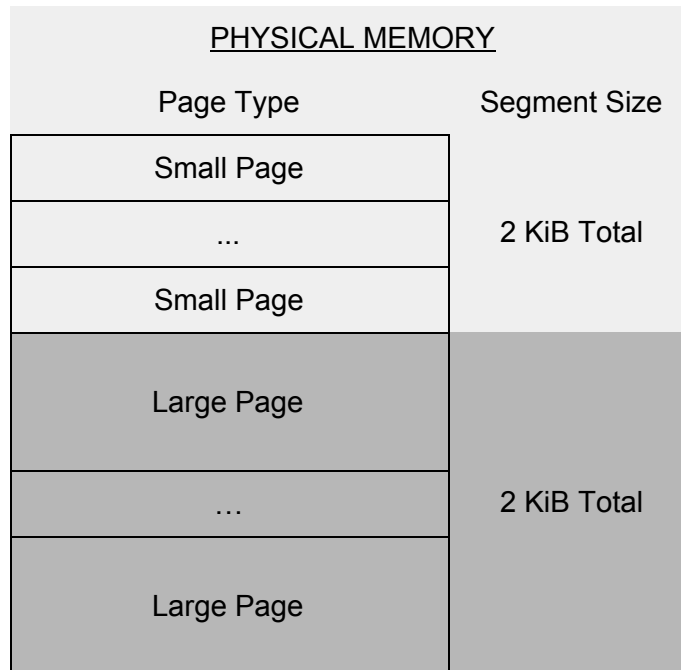
7. **Signal A:** A High B Low C Undefined

8. **Signal B:** A High B Low C Undefined

Question 11: Virtual Memory - 21 pts

Morgan wonders if she can decrease the overall cost of virtual memory by changing the page size of *some* pages on her machine. To do this, she combines ideas from both segmented and paged memory models creating a scheme she calls “Page-mented Virtual Memory”. It works as follows:

Morgan divides 4 KiB of physical memory such that there are two evenly sized segments. One contains “small pages” and the other contains “large pages”. In our physical memory model, pages are organised contiguously as follows, with small pages on top at smaller addresses and large pages at higher addresses:



Considering only the physical memory model, answer the following questions:

1. Morgan wants a small page to have a size of 256B. How many small pages fit in the small page segment?

_____ Small Pages

2. Morgan wishes to have a total of 4 large pages in her large page segment. How big must a large page be to have 4 of them in total?

_____ Bytes per Large Page

Because her scheme has variable page sizes (and variable offsets), Morgan realises she'll have to be creative about how she finds the VPN and offset of a given virtual address. She proposes numbering pages within their "small" or "large" segment, as shown below. Note that page numbers are not unique.

To decide how to break down the address, Morgan refers to the topmost virtual address bit: small-page addresses are 0 at this bit while large-page addresses are 1.

VIRTUAL MEMORY			
Topmost bit value	VPN value	Page Type	Segment Size
0	0	Small Page	4 KiB Total
...	
0	num_small - 1	Small Page	
1	0	Large Page	4 KiB Total
...	
1	num_lrg - 1	Large Page	

For the remainder of this problem, you may make the following assumptions which may differ from your calculated answers above:

- 4 KiB of PM (2 KiB small segment, 2 KiB large segment) with 16 small pages, 8 large pages
- 8 KiB of VM (4 KiB small segment, 4 KiB large segment)
- sizeof(small segment) == sizeof(large segment)
- sizeof(large page in VM) == sizeof(large page in PM)
- sizeof(small page in VM) == sizeof(small page in PM)

1. How many bits (at most) does it take to represent the VPN of a LARGE page?

_____ bits

2. How many bits (at most) does it take to represent the VPN of a SMALL page?

_____ bits

3. How many bits (at most) does it take to represent the PPN of a LARGE page?

_____ bits

4. How many bits (at most) does it take to represent the PPN of a SMALL page?

_____ bits

5. How many rows must our page table contain?

_____ rows

For each of the following accesses, find the topmost bit, PPN, and offset. Then, decide whether the address results in a TLB hit, page table hit, or page fault. Assume the accesses happen in order and that they modify the TLB, page table, and physical memory as they are executed. Assumptions from the previous portion still hold. You do not need to change/mark the TLB or page table for credit.

Free Page List
0x17 (small)
0xC (large)

LRU = 1 → Replace me! I am the “least recently used” item :)

TLB			
Topmost bit	VPN	PPN	LRU
1	0x3	0x9	0
0	0x1	0x2	1

Assume shown entries are valid, omitted entries are invalid, and that the page table is of proper size given the VM/PM specifications

Page Table		
Topmost bit	VPN	PPN
0	0x1	0x2
0	0x3	0x5
0	0x6	0x4
1	0x1	0x7
1	0x3	0x0
1	0x7	0x6

Please write your answers in HEX.

Virtual Address	Topmost bit	PPN	Offset	Result of Access
0b0000110000110				<input type="radio"/> TLB Hit <input type="radio"/> Page Table Hit <input type="radio"/> Page Fault
0b1001110101010				<input type="radio"/> TLB Hit <input type="radio"/> Page Table Hit <input type="radio"/> Page Fault
0b0000111101101				<input type="radio"/> TLB Hit <input type="radio"/> Page Table Hit <input type="radio"/> Page Fault

Morgan simulates her virtual memory design and finds it takes 1000ns to fetch one small page from disk and 5000ns to fetch one large page. It takes 100ns to do a single memory access. On a set of benchmarks, she also find programs experience page faults 10% of the time with 6% occurring on small pages and 4% occurring on large pages.

Assuming the page table fits completely in one large page (and that the table is loaded before the program runs, but memory is otherwise cold), what is the average time taken to complete a memory access in this scheme?

Assume nothing is cached, that we do not have a TLB, and that updates to the page table require a separate memory access.

_____ ns