1University 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 - SOLUTIONS

***Last** Name (Please print clearly)*				
***First** Name (Please print clearly)*				
Student ID Number				
Circle the name of your Lab TA	**Ayush Maganahalli** **John Yang**	**Chenyu Shi** **Lu Yang**	**Gregory Jerian** **Ryan Searcy**	**Jenny Song** **Ryan Thornton**
Name of the person to your: Left | Right				
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)***				

Instructions

- This booklet contains 32 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").`

Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

2. Places the string "cs61c" in RAM.

Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

3. Removes all pseudo instructions.

Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

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

Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

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

Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

6. Stage most often responsible for loop unrolling.

Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ 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 = 0
exp - bias = 2
exp = 2 + 1023 = 1025
significand = 0011

Sign:0b0

Exponent: 0b10000000001

Significand:0b 0011

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.

1 has an exponent value of 1023 because $2^0 = 1$. This means alls the values with a positive sign and exponent less than 1023 are in this range. Because there are 4 significand bits there are 16 numbers per exponent.

= 1023 * 16 = 16368

16368 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

Our largest number in two's compliment is $2^{15} - 1$. The largest number in floating point is $2^{1023} * (1.1111)$, which is clearly much larger.

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

Ⓐ Our Floating Point Scheme Ⓑ Two's Complement

In two's complement there are exactly 64 numbers. For two's compliment we look at each power of 2 (an exponent value) and because there are 4 significand bits there will be 16 per power of 2. Between 1 and 64 there are 6 powers of 2. 6 * 16 = 96.

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

Ⓐ Our Floating Point Scheme Ⓑ Two's Complement

Our floating point scheme only represents 16 numbers (each power of 2 in the range has 16 significand values) while two's complement represents 64.

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 openmp 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

$$True\ Speedup = \frac{old\ time}{new\ time} = \frac{20}{10} = \frac{1}{0.1 + \frac{0.40}{8} + \frac{0.5}{x}}$$

$$2 + 1 + \frac{10}{x} = 10$$

$$x = \frac{10}{7}$$

SIMD SPEEDUP $\frac{10}{7}$

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];
  }
}
```

Ⓐ Always **Ⓑ Sometimes** Ⓒ Never

(Data race on the `sum` variable)

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

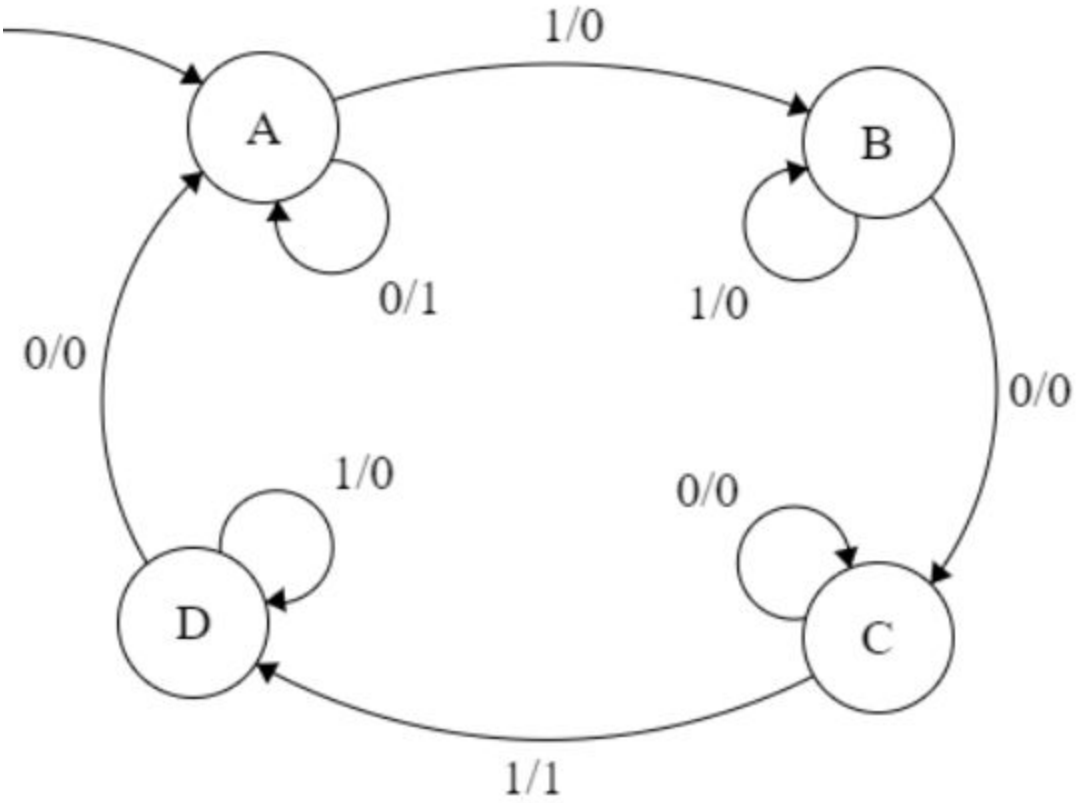
```
// Copies an input array into an output array
void sum_array(int* array, int* output, unsigned int size) {
  #pragma omp parallel for
  for (int i=0; i<size; i++) {
    output[i] = array[i];
  }
}
```

Ⓐ Always Ⓑ Sometimes Ⓒ Never

(Output is only written to at thread-private indices)

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	**B**	**C**	**C**	**D**	**D**	**A**	**A**	**A**
Output	-	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**

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 an 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 i = 0; i < q->occupied_size; i++) {

            int location = q->right_location - i - 1;

            if  (location < 0) {

                  location = location + q->allocated_size;

            }

            printf (“%d\n”, q->data[location]);

      }

}
```

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 (q->data, q->allocated_size * sizeof(int));

      for (int i = 0; i <= q->left_location; i++) {

            q->data[q->occupied_size + i] = q->data[i];

      }

      q->right_location = q->left_location + q->occupied_size + 1;

      if (q->right_location >= q->allocated_size) {

            q->right_location -= q->allocated_size;

      }

}
```

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 s4 s2 LoopEnd
            # Load source value once
            add t0 s0 s4
            lb t0 0(t0)
            # Compute all adds
            add s3 t0 s3
            add t1 s3 s4
            add t1 t1 t0
            # Store in dest
            add t0 s1 s4
            sb t1 0(t0)
            # Make the first call to printf
            la a0 intPrint
            mv a1 s3
            jal printf
            addi s4 s4 1
            j LoopStart
LoopEnd:
            la a0 stringPrint
            mv a1 s1
            jal printf
            la a0 stringPrint
            mv a1 s0
            jal printf
```

2. Complete the prologue and epilogue for the mystery function. Use the calling convention learned in class. You may not need all lines.

```
Prologue:
        addi sp sp -24
        sw s0 0(sp)
        sw s1 4(sp)
        sw s2 8(sp)
        sw s3 12(sp)
        sw s4 16(sp)
        sw ra 20(sp)
mystery: ...
Epilogue:
        lw s0 0(sp)
        lw s1 4(sp)
        lw s2 8(sp)
        lw s3 12(sp)
        lw s4 16(sp)
        lw ra 20(sp)
        addi sp sp 24
        jr ra
```

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 $\sqrt{\texttt{arr[0]}^2 + \cdots + \texttt{arr[n-1]}^2}$, 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; _i<(size/4)*4__; _______i+=4_______) {


            simd_t temp_vec = ____simd_load(&(arr[i]))_____;

            ____________temp_vec = simd_mul(temp_vec, temp_vec)_________;

            ____________sum_vec = simd_add(sum_vec, temp_vec)___________;
      }

      double sum_arr[_______4_______];

      ________simd_store(sum_arr, sum_vec)_____________________;

      double ret_val = ____sum_arr[0] + sum_arr[1] + sum_arr[2] + sum_arr[3]___;

      // Tail Case
      for (int i = ___(size/4)*4_____; ___i<size____; ____i++_____) {


            _______ret_val += arr[i]*arr[i]_________;


            ____________________________________________________________________________;
      }
      // Square root
      return sqrt(___________ret_val_____________________________);
}
```

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	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		...
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

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.

1. 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 = 0 P2 = 0 P4 = 0 P8 = 1 P16 = N/A

2. 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

3. 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?

7 Bits are necessary (Can cover up to 120 Data Bits). 6 Bits can only surveil 57 data bits at most.
Explanation: Given *m* parity bits

- The number of possible data bits it can cover is given by $k = 2^m - m - 1$
- The total number of bits needed is given by *data + parity*, or $n = 2^m - 1$

7 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.

Ⓐ Striping Ⓑ Mirroring Ⓒ Both Ⓓ Neither

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

Ⓐ Striping Ⓑ Mirroring Ⓒ Both Ⓓ Neither

Striping makes read and write operations better. Mirroring also improves reads without harming writes.

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

Ⓐ Striping Ⓑ Mirroring Ⓒ Both Ⓓ Neither

7. RAID 0

Ⓐ Striping Ⓑ Mirroring Ⓒ Both Ⓓ Neither

8. RAID 1

Ⓐ Striping Ⓑ Mirroring Ⓒ Both Ⓓ Neither

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

Ⓐ True Ⓑ 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: $\log_2 2^{20} - 3 - 4 = 20 - 7 = 13$

Index: $\log_2(256 / (2 * 16)) = \log_2 8 = 3$

Offset: $\log_2 16 = 4$

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

First we notice that we are only accessing 2 blocks at a time and they are contiguous. Thus we expect only to have a cache miss when we load in a block during
either initialization or the inner loop.

Now let's calculate our hit rate in a very general manner. We first look at our access pattern. Let's consider the access pattern for fib[2], fib[3], fib[4], fib[5]

fib[2] = fib[1] + fib[0]

fib[3] = fib[2] + fib[1]

fib[4] = fib[3] + fib[2]

fib[5] = fib[4] + fib[3]

By looking at fib[2] we notice that an element will be written to once and then read twice, leading to roughly 3 elements per access. The only exceptions to this will be at the boundaries. Fib[0] is accessed twice, once on initialization and once on fib[2]. Fib[1] is accessed 3 times. Fib[n - 1] is accessed twice, once on itself and once on fib[n]. Finally fib[n] is accessed twice, once in the loop and once in the return statement.

Thus we find that we perform (n - 2) * 3 + 3 * 2 = 3n - 6 + 6 = 3n memory accesses.
We know we will only have compulsory misses, so we need to figure out how many blocks we span.
To determine the number of blocks let's calculate the total amount of memory.

Memory = num_elems * sizeof (elem) = (n + 1) * sizeof (elem)
Now to determine the number of misses we convert this to a number of blocks access.

#blocks = ceil (Memory / blocksize) = ceil ((n + 1) * sizeof (elem) / blocksize) = # misses

Plugging in sizeof(elem) = 4, blocksize = 16, n = 32 we get:

misses = ceil (33 * 4 / 16) = ceil (33 / 4) = 9

HR = (#access - #misses) / #accesses = (3*32 - 9) / (3 *32) = 87 / 96 = 29 / 32

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

Notice our first access to every block is a write. Thus we would have at least one extra miss per block as opposed to the write allocate miss policy.

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)?

We can build off our general solution in part 2 and just plug in. Recall from question 2 we have

misses = ceil ((n + 1) * sizeof (elem) / blocksize)

accesses = ceil ((n + 1) * sizeof (elem) / blocksize)

HR = (#access - #misses) / #accesses

We can plug in sizeof(elem) = 8, blocksize = 16, n = 64 and we get:

misses = ceil (65 * 8 /16) = ceil (65 / 2) = 33

HR = (64 * 3 - 33) / (64 * 3) = 159 / 192 = 53 / 64

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

We saw in part 2 that because only use the two most recent blocks and then never revisit we only have compulsory misses.

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

We saw in part 2 that because only use the two most recent blocks and then never revisit we only have compulsory misses

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(_____tokens[2]_____,_____tokens[3]_____)
    result.append(tuple(tuple(tokens[0], tokens[1]), timediff))
  return result

def count_time(v1, v2):


  return _____v1+v2_____

def group_by_type(k, v):


  return _____tuple(k[0], tuple(v, 1))____

def count_ids(v1, v2):


  return _____tuple(v1[0]+v2[0], v1[1]+v2[1])_____

def average(k, v):


  return _____tuple(k, v[0]/v[1])_____

# 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 [x] destination

B) [ ] 1 [ ] 2 [ ] 3 [x] 4 [x] 5

C) [x] source [ ] destination

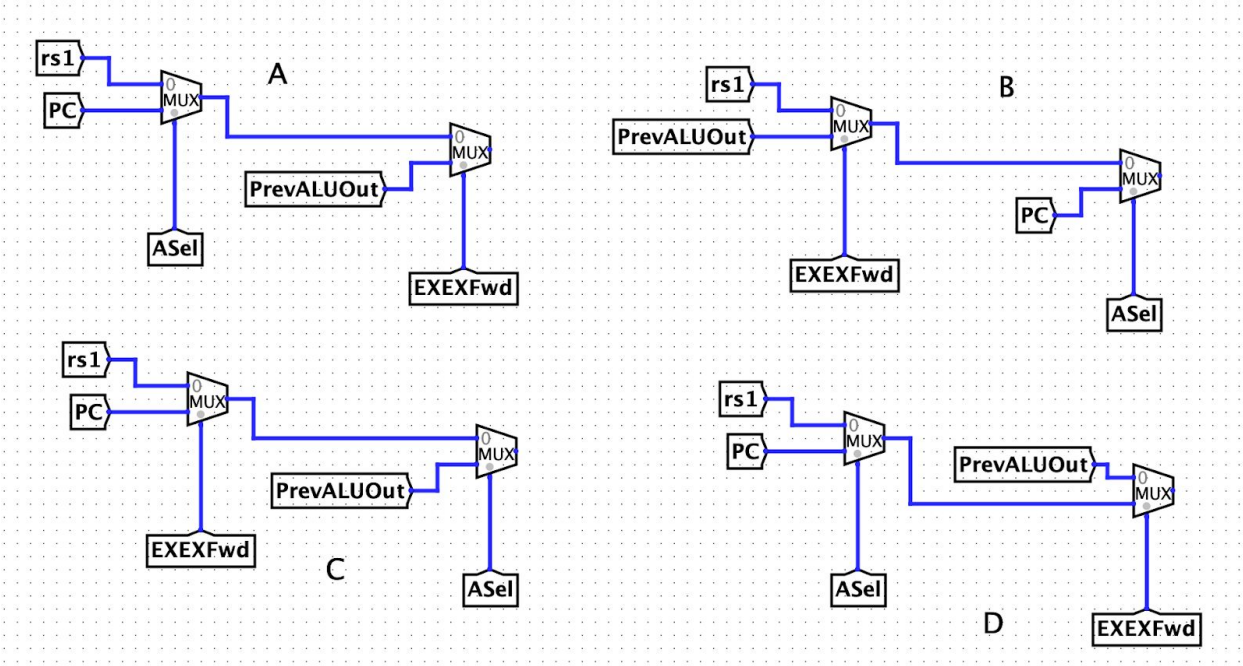
2. Feeling a little overwhelmed with forwarding, you try to break the problem down into small pieces. First you consider the case where we need to forward from our ALU output to the next EX stage as an argument:

```
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 Ⓑ B Ⓒ C Ⓓ 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

We've only modified ASel, not BSel, so our rs2 cannot be forwarded to.

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.

1 WB to EX

3 EX to EX

2 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    EX    MEM    WB
add t3 t0 t2          IF    ID    EX     MEM    WB
```

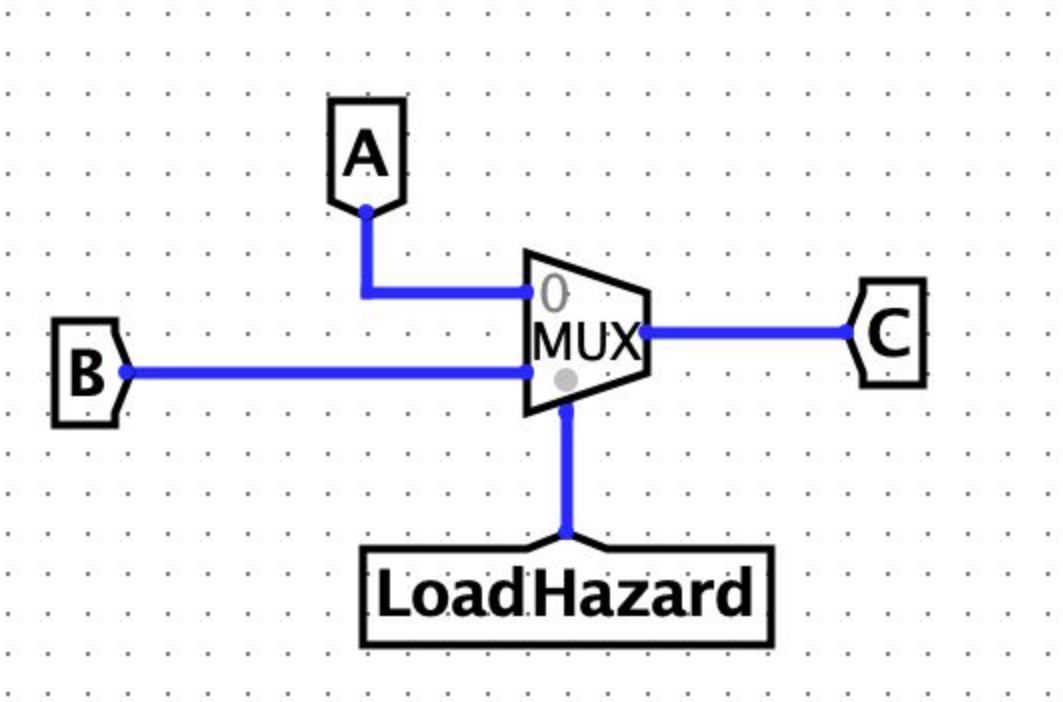
a. What is the *earliest* stage at which the load data is ready/available to forward? Circle one stage.

`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 t2` 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:

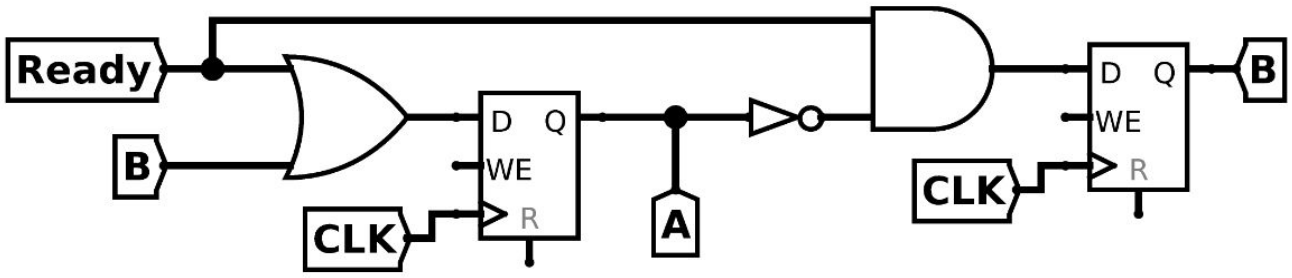
Ⓐ IMEM output Ⓑ ID input (RegFile parser input) Ⓒ PC Ⓓ NOP instruction

C:

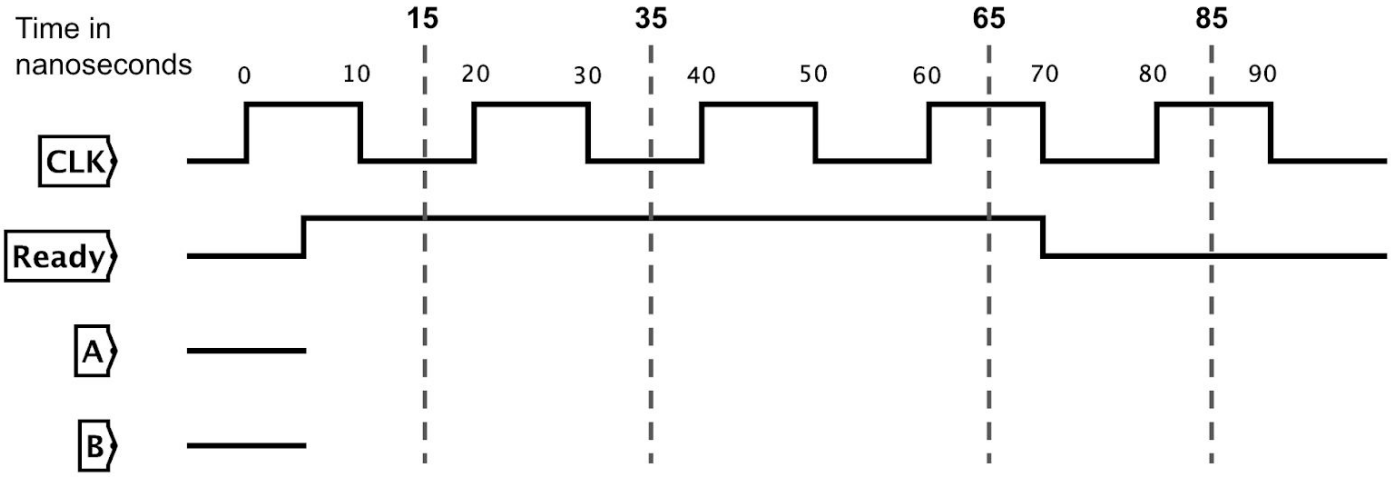
Ⓐ IMEM output Ⓑ ID input (RegFile parser input) Ⓒ PC Ⓓ NOP instruction

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

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

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

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

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

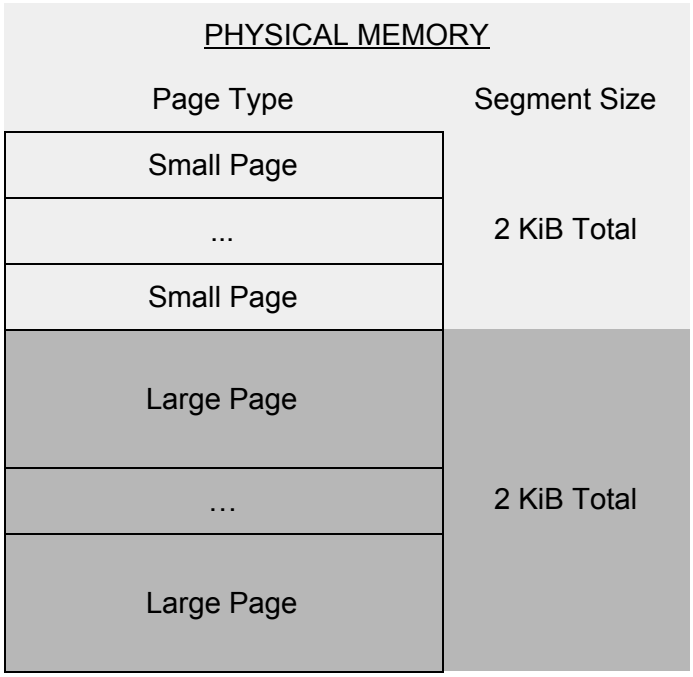
What is the value of the output signals at time **85 ns**? (circle the correct answer for each signal)

7. **Signal A**: Ⓐ High **Ⓑ Low** Ⓒ Undefined
8. **Signal B**: Ⓐ High **Ⓑ Low** Ⓒ 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 KiB = 2 * 2^{10} = 2^{11} B in small segment
   2^{11} / 2^{8} = 2^{11-8} = 2^{3} → 8 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
   2 KiB = 2^{11} B in large segment
   2^{11} / 2^{2} = 2^{11-2} = 2^{9} B per large lage

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 with 16 small pages, 8 large pages
- 8 KiB of VM with 4 KiB of small, 4 KiB of large pages
- 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
   Large page size = (½ PM) / 2^3 = 2^{11} / 2^3 = 2^8 B per large page
   Num large pages = 4 KiB / 2^8 = 2^{12} / 2^8 = 2^4 = 16 large pages
   log_2(2^4) = 4 bits for large page VPN (or 5, if including topmost bit)

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

   _____ bits
   Small page size = (½ PM) / 2^4 = 2^{11}/2^4 = 2^7 B per small page
   Num small pages = 4 KiB / 2^7 = 2^{12} / 2^7 = 2^5 = 32 small pages

log_2(2^5) = 5 bits for small page VPN (or 5, if including topmost bit)

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

   _____ bits
   log_2(2^3) = 3 bits
4. How many bits (at most) does it take to represent the PPN of a SMALL page?

   _____ bits
   log_2(2^4) = 4 bits
5. How many rows must our page table contain?

   _____ rows
   Total num virtual pages = num_small_VM + num_large_VM
   = 2^4 + 2^5 = 16 + 32 = 48 rows

For each of the following accesses, find the topmost bit, VPN, 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	1
0	0x3	0x5	0

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

Virtual Address	Topmost bit	PPN	Offset	Result of Access
0b0 00011 0000110	0	0x5	0x6	[ ] TLB Hit [x] Page Table Hit [ ] Page Fault
0b1 0011 10101010	1	0x9	0xAA	[x] TLB Hit [ ] Page Table Hit

				[ ] Page Fault
0b0 00011 1101101	0	0x5	0x6D	[x] TLB Hit [ ] Page Table Hit [ ] 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% of total faults occurring on small pages and 4% of total faults 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 and that we do not have a TLB.

_____ ns

AMAT = page table check + .9(memory hit) + .1(small_fault_rate(small_fault_cost) + large_fault_rate(large_fault_cost))
= 100ns + .9(100ns) + .1(.6(1000ns + 100ns) + .4(5000ns + 100ns))
= 100ns + 90ns + .1(.6(1100) + .4(5100))
= 100ns + 90ns + .1(660ns + 2040ns)
= 190ns + 270ns
= 460ns