Nick Weaver CS 61C Sp 2019 Great Ideas in Computer Architecture Final Exam

PRINT your name: _____, _____
(last) (first)

I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that any academic misconduct will be reported to the Center for Student Conduct, and may result in partial or complete loss of credit. I am also aware that Nick Weaver takes cheating personally and, like the Hulk®, you don't want to see him angry.

SIGN your name: _____

PRINT your class account login: `cs61c-`_____ and SID: _____

Your Favourite 61C TA's name: _____

Number of exam of person to your left: _____

Number of exam of person to your right: _____

You may consult three sheets of notes (each double-sided). You may not consult other notes, textbooks, etc. Calculators, computers, and other electronic devices are not permitted. Please write your answers in the spaces provided in the test.

You have 180 minutes. There are 9 questions, of varying credit (180 points total). The questions are of varying difficulty, so avoid spending too long on any one question. Parts of the exam will be graded automatically by scanning the **bubbles you fill in**.

Square boxes indicate you may select more than one option, circular bubbles indicate you should select only one.

Do not turn this page until your instructor tells you to do so.

Question:	1	2	3	4	5	6	7	8	9	Total
Points:	25	21	15	17	26	16	21	19	20	180

Problem 1 ***Potpourri*** **(25 points)**

(a) YFSE (your favorite search engine) corporation has several datacenters (WSCs) that use an interrupt-based approach to handling network-interface events. The context switch from the current process to the interrupt handler takes 15 microseconds of overhead, while the actual handling of the packet takes 45 microseconds. How long does it take us to process one packet given the interrupt model?

Time = _____ microseconds

Solution: 60 ms

(b) The engineers have a new solution that uses polling. Polling has only 1 microsecond of overhead, and the polling interval is every 60 microseconds. How long does it take us to process one packet given the new polling model?

Time = _____ microseconds

Solution: 46 ms

(c) If there is *no* network traffic, what percentage of the CPU will be spent polling?

Overhead = _____ %

Solution: 1.66% or $(1/60) * 100$

(d) The engineers have another, slightly more complicated solution that uses interrupts and polling. Here, the first arriving packet generates an interrupt with 15 microseconds of overhead, and the next 4 packets (which are guaranteed to have arrived) can be polled with a 1 microsecond overhead each. Assume processing cannot happen in parallel. What is the average time to process a single packet in this new model? Leave your answer unsimplified.[1]

Time = _____ microseconds

[1] This is actually what modern OSs do in practice for high performance networking, switching between interrupts and polling based on network activity

Solution: $(15 + 4 * 1 + 5 * 45)/5 = 48.8ms$

(e) YFSE really appreciates your help with fixing their networking stack, but now they are looking to do some hardware upgrades and it is up to you to analyze if the upgrades are worthwhile!

As a datacenter, YFSE spends 4 MW (Mega Watts) on compute, 2 MW on networking, 2 MW on storage to serve their customers. They also spend 2 MW on cooling, and 1 MW on power and other sources.[2]

What is the current PUE? Simplify to a fraction of the from A/B where A and B are single integers.

PUE = _____

Solution: 11/8

(f) If the new networking hardware uses 1/2 the power of the old networking hardware, what will the new PUE be? Simplify to a fraction of the from A/B where A and B are single integers.

PUE = _____

Solution: 10/7

(g) YFSE runs a test program to see how the new system performs. They see the program spends 3% of its time traversing the network (latency), and 7% of its time actually transferring (transmission delay).

If the new networking hardware speeds up our network traversal by a factor of 1.5 and also speeds up transmission by a factor of 1.75, what is the speedup of the test program? Don't simplify.

Speedup = _____

Solution: $\frac{1}{(1-.1)+\frac{.03}{1.5}+\frac{.07}{1.75}}$

[2] We consider networking and storage as "useful work".

(h) YFSW has some FPGA based accelerators, so for this custom hardware they are using a 16 bit floating point scheme which works the same as our in-lecture version, except it has the following breakdown:

Sign: 1 bit, Exponent: 7 bits, Signficand: 8 bits

Represent the given number in our scheme by filling in fields below. If you cannot represent the number **exactly**, you should select NOT REPRESENTABLE

1. -3/64

   ○ Represented as:

   Sign: $0b$_____

   Exponent: $0b$_____

   Singificand: $0b$_____

   ○ Not Representable

   Solution: Sign: $0b1$

   Solution: Exponent: $0b0111010$

   Solution: Significand: $0b10000000$

2. 1/6

   ○ Represented as:

   Sign: $0b$_____

   Exponent: $0b$_____

   Singificand: $0b$_____

   ● Not Representable

3. 1025/1024

○ Represented as:

Sign: $0b$_____

Exponent: $0b$_____

Singificand: $0b$_____

● Not Representable

(i) Find the average memory access time for a system with the following characteristics. For partial credit, write the formula. For full credit, simplify your answer. Assume we check the TLB, then the page table. For this problem, we ignore page faults.

Translation	
TLB Hit Time	0 ps*
TLB Miss Rate	20% of accesses
Page Table Hit Time	60 ps

Data Access	
Data Cache Hit Time	10 ps
Data Cache Miss Rate	25% of accesses
Memory Access time	40 ps

*NOTE: The hardware assumes a TLB hit and is able to overlap the TLB computation with the fetch of the tag from the cache. Both portions arrive at the same time, meaning our TLB hit time is effectively none.

AMAT = _____ ps

Solution: $0 + .2(60) + 10 + (.25)(40) = 32ps$

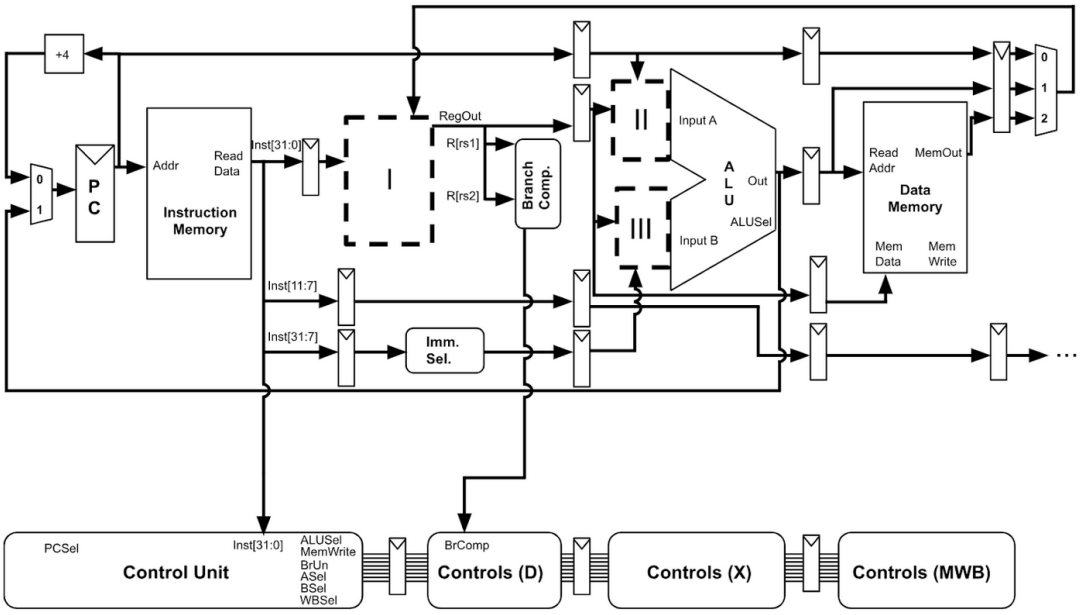
Problem 2 ***Damon-path*** **(21 points)**

In this question, we will incorporate a new instruction ("madd") into our five-stage, pipelined datapath that allows us to perform a multiply and addition in a single instruction. The RTL is written below:

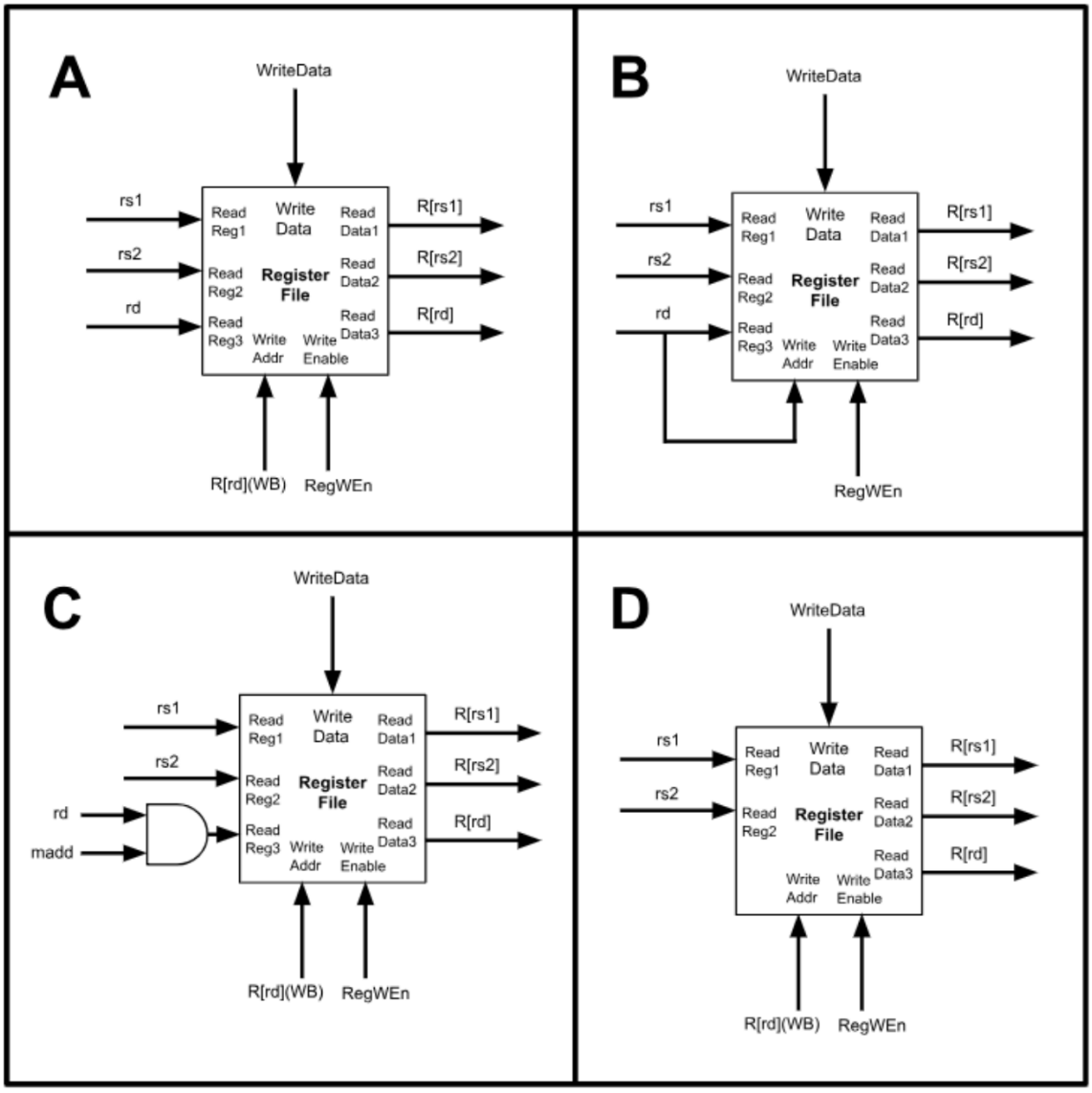
```
madd:  R[rd] = R[rd] + (R[rs1] * R[rs2])
```

Throughout this question, when it is unclear which stage a signal is coming from, we use the syntax `<signal>('stage')`. For example, to specify instruction bits 7 through 11 from the execute stage, we write `inst[11:7](EX)`.

Select the correct options that will implement this instruction with *the least amount of hardware*. Assume we've also added a new control signal `madd` which is 1 when we encounter a `madd` insdtruction and 0 otherwise.

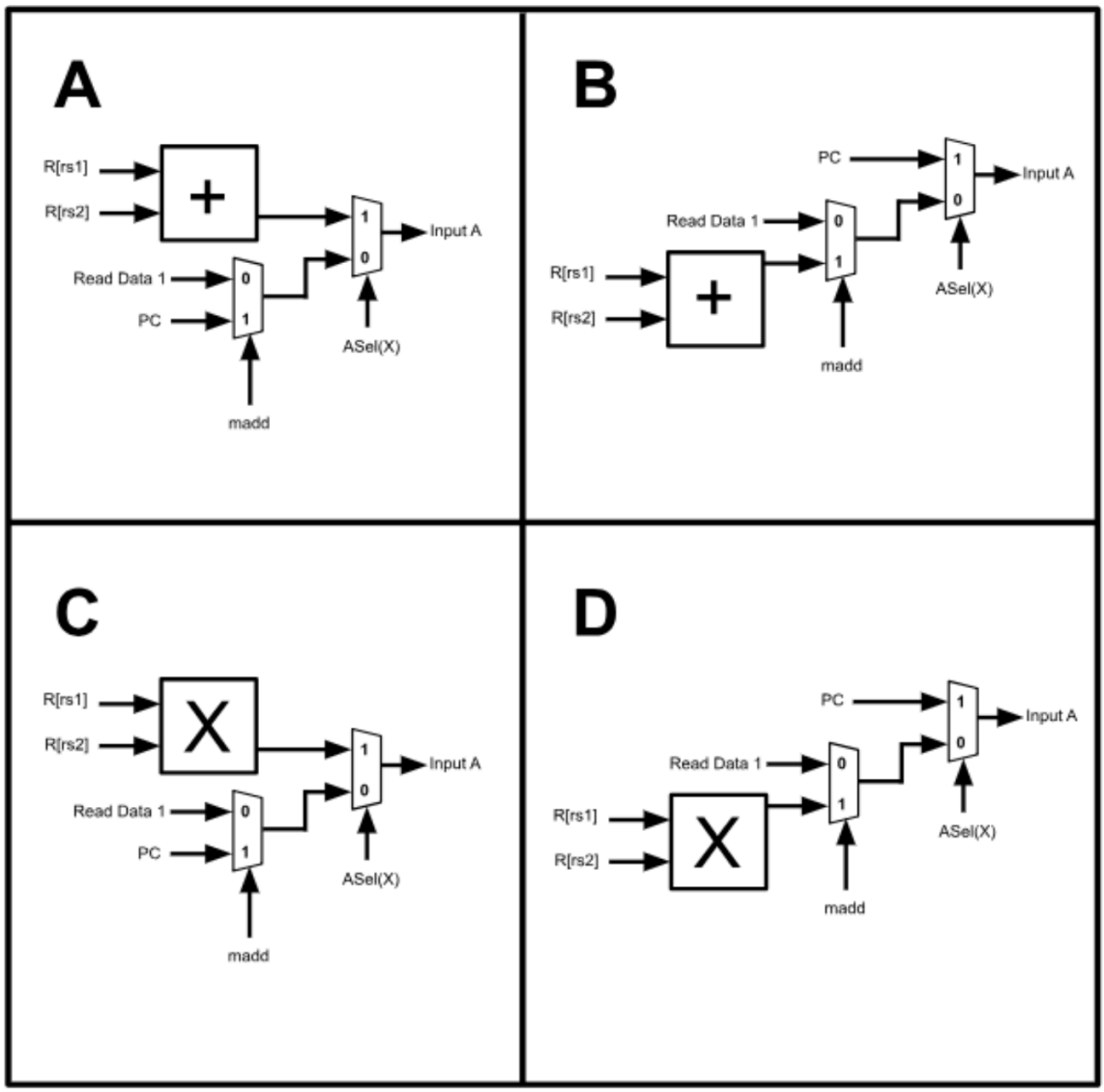


(a) Which of the options below is the best fit for box I (1) on the previous page? Fill in the multiple choice bubbles below.



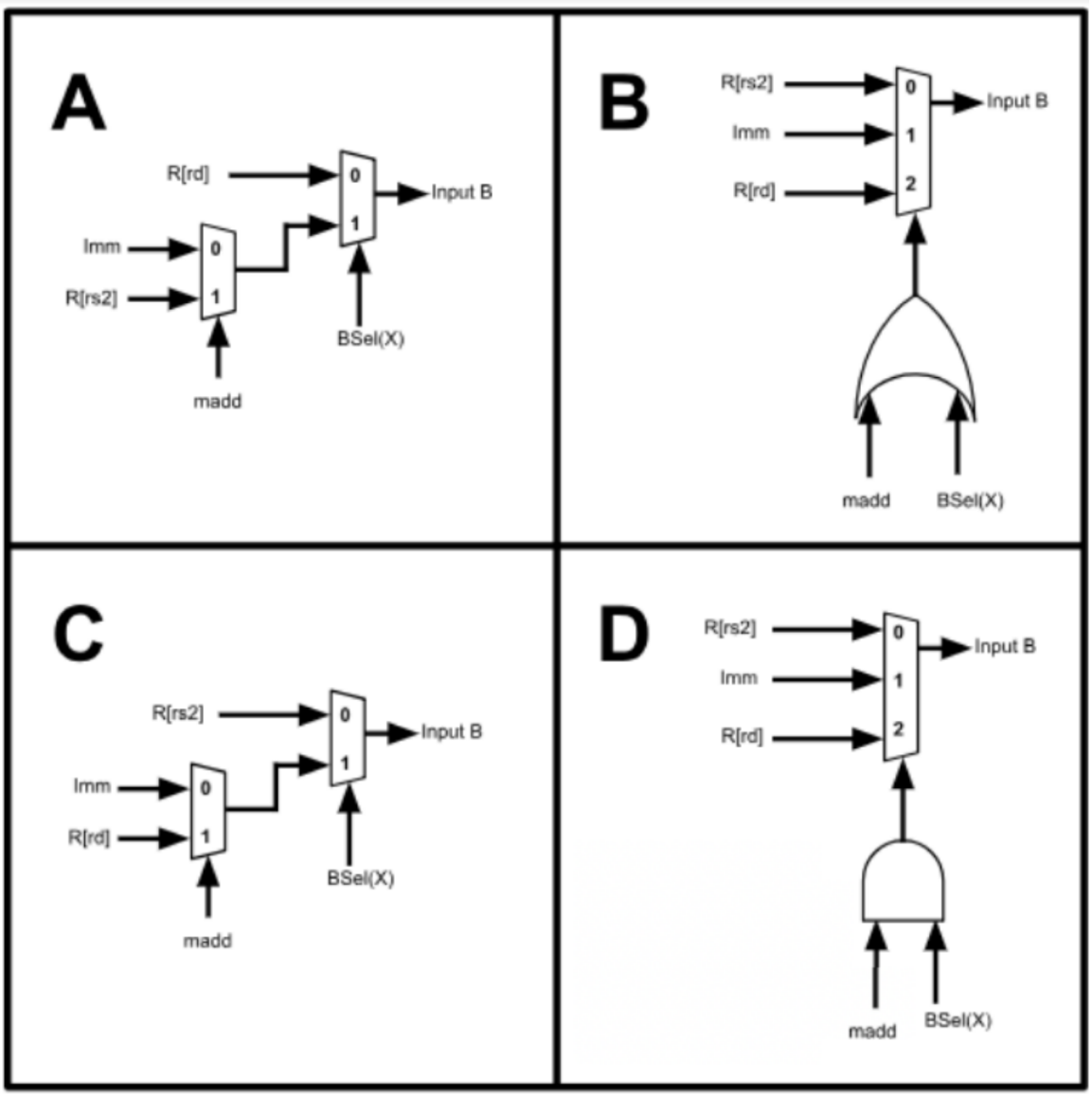
- (●) A
- ( ) B
- ( ) C
- ( ) D

(b) Which of the options below is the best fit for box II (2) on the previous page? Fill in the multiple choice bubbles below.



- ○ A
- ○ B
- ○ C
- ● D

(c) Which of the options below is the best fit for box III (3) on the previous page? Fill in the multiple choice bubbles below.



- ○ A
- ○ B
- ● C
- ○ D

(d) Fill in the correct values for the control signals below to correctly execute a madd instruction. Assume other control bits have been done for you and are correct. For ALUSel, please write an operation name (ie. add) not a numeric value. For RegWEn and WBSel, use integer, decimal numbers.

Signal	ALUSel	RegWEn	WBSel
Value			

Solution:

Signal	ALUSel	RegWEn	WBSel
Value	add	1	1

(e) Given the timing specifications below, calculate the critical path of the new pipelined datapath.

ClkQ	Setup	RF Setup	RF Read	ALU	MUX	Mem Read	Mem Write	Branch Comp
30ps	20ps	20ps	150ps	200ps	25ps	250ps	200ps	75ps

Do NOT use your answers in parts (A / B / C) to do this calculation. Instead assume the following delays for each box on the datapath (which may or may not be correct):

I : 160 ps
II : 75 ps
III : 75 ps

For full credit, simplify your answer to a single number (ie. do NOT leave your answer as a formula). If a component is not listed in the table above, assume its time is negligible.

New Critical Path = _____ ps

Solution: 325 ps

Problem 3 ***Virtual Memory*** **(15 points)**

Assume we're working on a machine which has the following parameters:

- 16GiB of physical memory
- 22 bit virtual addresses
- 128B pages
- Each PTE in our single level table is 4B

(a) How many bits are in the Physical Address Offset?

Offset = _____ bits

Solution: 7 bits

(b) How many bits are in the PPN? the VPN?

VPN = _____ bits

Solution: 15 bits

PPN = _____ bits

Solution: 27 bits

Consider the following snippet of code. Assume the following:

- Arrays begin on page boundaries.
- We have a function `initialize` that creates an array containing `size` integers.
- A starts at 0x20000 and B starts at 0x30000
- `sizeof (int)` == 4

```
int size = 256;
int32_t *A = initialize(size);
int32_t *B = initialize(size);
for (int i = 0; i < size; i += 32) {
    B[i] = B[size - i - 1] * A[i];
}
```

For the following parts, assume "data pages" refers only to pages containing elements of A and B (ie. not pagetable pages). Remember that we have 128B pages and each PTE is 4B.

(c) How many unique DATA pages does this access pattern traverse?

- ○ 0 pages
- ○ 2 pages
- ○ 3 pages
- ○ 8 pages
- ○ 12 pages
- ○ 13 pages
- ● 16 pages

(d) How many unique NON-DATA, NON-CODE pages does this access pattern traverse? (ie. page table pages)

- ○ 0 pages
- ● 2 pages
- ○ 3 pages
- ○ 8 pages
- ○ 12 pages
- ○ 13 pages
- ○ 16 pages

(e) Suppose this process has a single-level page table that starts out empty and we run through this access pattern. How much space would the page table take in memory? Give your answer in units of PAGES (*NOT BYTES*).

Size = _____ pages.

Solution: 2^{10} pages

Problem 4 ***Nick Goes Nuclear - Atomics*** **(17 points)**

In class you learned about OpenMP and got to experience speedups on the Hive Machine. However after your time in 61C you developed a deep-seated hatred for X86 and have determined that you want to employ OpenMP on RISC-V machines using atomic instructions.

You decide to start small and you seek to implement the following parallelization of summing a loop.

```
int sum = 0;
#pragma omp parallel for {
for (int i = 0; i < n; i++) {
     #pragma omp critical
     sum += A[i];
}
```

When executing the for loop, each thread holds its local starting and terminating byte offset in t0 and t1 respectively. You store the address of `sum` in s1 and the address of `A` in s2. Now you are tasked with implementing the actual `sum` update. You develop the following code which WORKS:

```
loop_start:
     beq t0 t1 end
     add t2 s2 t0
     lw t2 0(t2)
retry:
     lr.w t3 (s1) # Load sum and place our reservation
     add t3 t3 t2
     sc.w t4, t3 (s1)
     bne t4 x0 retry # Check if our store failed
     addi t0 t0 4
     j loop_start
```

(a) Your friend, however, took 61C back in Fall 2017, so he only understands amoswap. Your friend asks if you could reimplement the same piece of coding using amoswap instead, without needing any values other than those in t0, t1, s1, and s2. Is this possible? Why or why not?

○ Yes ● No

Solution: To increment the sum atomically using `amoswap` you need to already know the old value of `sum` (otherwise you cannot replace it with the incremented value). This requires another atomic access for the read, most likely through a lock.

(b) You decide to stick with your existing implementation, but you discover your code is much slower than expected. In fact, it is almost as though you are getting no parallelism at all. You remember OpenMP uses the reduction keyword to solve that problem.

Transform the code above to perform a reduction before updating the sum using `lr.w` and `sc.w`. You may store any "private" variables in registers directly rather than memory. You may not need all lines.

```
    add t5 x0 x0
label1:
    beq t0 t1 label2
    _____________________________________________
    _____________________________________________
    _____________________________________________
    addi t0 t0 4
    j label1
label2:
    _____________________________________________
    _____________________________________________
    _____________________________________________
    _____________________________________________
```

Solution:

```
    add t5 x0 x0
label1:
    beq t0 t1 label2
    add t2 s2 t0
    lw t2 0(t2)
    add t5 t5 t2
    addi t0 t0 4
    j label1
label2:
    lr.w t3 (s1)
    add t3 t5 t3
    sc.w t4, t3 (s1)
```

```
    bne t4 x0 label2
```

(c) You talk to another former 61C student who took the class back in Summer 2018 and that friend tells you about `amoadd.w`, it works as follows:

```
amoadd.w rd, rs2, (rs1)
# Loads the value at the address rs1, adds the result to rs2, and stores
it back in rs1
# Returns the result in rd
# All happens ATOMICALLY!!!!  Aka one instruction.
```

Using `amoadd.w` you can rewrite your original code to be:

```
loop_start:
    beq t0 t1 end
    add t2 s2 t0
    lw t2 0(t2)
    amoadd.w x0, t2, (s1) #atomically adds t2 to s1
    addi t0 t0 4
    j loop_start
```

Select the most correct option:

Your new code will be _____ than the reduction you implemented part (b).

● slower ○ faster

Problem 5 ***SIMD*** **(26 points)**

In this question, you will implement a vectorized max function. The goal is to find the maximum element in an array of `n` signed **8-bit integers**. You will need to compute partial maxima that are stored in a vector register and finally reduce it down to a single element. You may ONLY use the intrinsics on the cheat sheet we have provided.

```
#include <immintrin.h>

int8_t fast_max(size_t n, int8_t a[]) {
    // Init elements to minimum value
    __m128i max_vec = _mm_set1_epi8(-128);

    for (size_t i=0; i < ____________________; i+= ___________) {

        __m128i temp_vec = ____________________________________________;

        ____________________________________________________________________;
    }
    // Reduction step
    max_vec = _mm_max_epi8(_________,___________(_____________________));

    max_vec = _mm_max_epi8(_________,___________(_____________________));

    max_vec = _mm_max_epi8(_________,___________(_____________________));

    max_vec = _mm_max_epi8(_________,___________(_____________________));

    int8_t ret_val, result[____________________];

    ____________________________________________________________________;

    ____________________________________________________________________;


    // Tail case
    for (size_t i = ___________; ___________; ___________) {

        ret_val = ___________ > ___________ ?  ___________ :  ___________;
    }
    return ret_val;
}
```

Solution:

```
#include <immintrin.h>

int8_t fast_max(size_t n, int8_t a[]) {
    // Init elements to minimum value
    __m128i max_vec = _mm_set1_epi8(-128);

    for (size_t i=0; i < n / 16 * 16; i+= 1) {

        __m128i temp_vec = _mm_loadu_si128((__m128i *)(a+i));;

        max_vec = _mm_max_epi8(max_vec, temp_vec);
    }
    // Reduction step
    max_vec = _mm_max_epi8(max_vec, _mm_alignr_epi8(max_vec, max_vec, 8));
    max_vec = _mm_max_epi8(max_vec, _mm_alignr_epi8(max_vec, max_vec, 12));
    max_vec = _mm_max_epi8(max_vec, _mm_alignr_epi8(max_vec, max_vec, 14));
    max_vec = _mm_max_epi8(max_vec, _mm_alignr_epi8(max_vec, max_vec, 15));
    int8_t ret_val, result[16];

    _mm_storeu_si128((__m128i *)result, max_vec);
    ret_val = result[15];

    // Tail case
    for (size_t i = n / 16 * 16; i < n; i++) {
        ret_val = a[i] > ret_val ?  a[i] :  ret_val;
    }
    return ret_val;
}
```

Problem 6 ***C Reading*** **(16 points)**

The function `parse_message` takes two inputs: an array of strings, and the length of the array. It copies the strings from the input array into a new buffer, ending the buffer with a NULL ptr rather than specifying a size. However if any of the strings are the string "STOP", then it terminates early and returns only strings before the stop message, again ending with a NULL terminator.

(a) The function below contains *at most 5 bugs* which cause the function to nondeterministically exhibit incorrect behavior. Bubble in the lines of code that may produce errors. **You may select more than one line.**

You may assume all calls to malloc succeed, `arr` and its contents are never NULL, `arr` always has at least `size` allocated, and we are using C99.

```
☐ 1.  char** parse_message (char** arr, size_t size) {
☐ 2.    int init_size = 8;
☐ 3.    char **output = malloc (sizeof (char *) * init_size);
☐ 4.    int i;
☐ 5.    for (i = 0; i < size; i++) {
☐ 6.        char *pointer = * arr + i;
☐ 7.        if (pointer == "STOP") {
☐ 8.            break;
☐ 9.        } else if (init_size == i - 1) {
☐ 10.           init_size *= 2;
☐ 11.           realloc (output, sizeof (char *) * init_size);
☐ 12.       }
☐ 13.       output[i] = malloc (sizeof (char) * strlen (pointer));
☐ 14.       strcpy (output[i], pointer);
☐ 15.   }
☐ 16.   output[i] = NULL;
☐ 17.   return output;
☐ 18. }
```

Solution: Lines 6, 7, 11, 13 have errors

(b) For each of the following addresses in `parse_message`, indicate what regions of memory the address *could* refer to when execution of code reaches line 16. Assume `size > 0` and `arr`'s contents are valid C strings. **You may select more than one option.**

1. `arr[0]`

■ Heap ■ Static

■ Stack ☐ Code

2. `"STOP"`

☐ Heap ■ Static

☐ Stack ☐ Code

3. `output[0]`

■ Heap ☐ Static

☐ Stack ☐ Code

4. `output`

■ Heap ☐ Static

☐ Stack ☐ Code

5. `&output`

☐ Heap ☐ Static

■ Stack ☐ Code

6. `&parse_message`

☐ Heap ☐ Static

☐ Stack ■ Code

Problem 7 ***Go Go Power Potpourri*** **(21 points)**

For each of the following scenarios, mark the type of parallelism best suited to the problem.

(a) You would like to find out what fraction of words in all of Wikipedia are adjectives.

○ Go concurrency ● MapReduce

(b) Given a server which uses AI to classify images as Dogs or Not Dogs and a cache of previous requests, you'd like to classify incoming requests using the classifier or serve old results from the cache.

● Go concurrency ○ MapReduce

(c) We attempt to send a 6 data string using Hamming ECC with single error correction, but no double error detection. We recover the following message when using even parity:

$0b1010010011$

Using the table on your cheat sheet, what is the correct data? Do not include any parity bits. Write one DATA bit per line. If you do not need all lines, align your answer to the right and fill other lines with zeros.

$0b$ _____ _____ _____ _____ _____ _____

Solution: $0b101111$

(d) Select **all** versions of RAID for which the given statement is true. **You may select more than one option.**

1. Can recover from a single disk failure.

   ☐ RAID 0 ■ RAID 5

   ■ RAID 1 ☐ None of the above

2. Can never do 2 small writes without writing to the same disk twice

   ☐ RAID 0 ☐ RAID 5

   ☐ RAID 1 ■ None of the above

3. Use the fewest number of disks for a given amount of available storage

   ■ RAID 0 ☐ RAID 5

   ☐ RAID 1 ☐ None of the above

(e) Express the value 0b10011101 in:

1. Hexadecimal

   $0x$ _____

   Solution: $0x9D$

2. Decimal integer, interpreting this as 8 bit two's complement

   _____

   Solution: -99

3. Decimal integer, assuming it was first cast to a 4 bit unsigned number

   _____

   Solution: 13

(f) For each of the following questions, determine what stage(s) of Compiler, Assembler, Linker, Loader the follow actions can happen. Assume static linking. **You may select more than one option.**

1. The imm in `la t0 LABEL` gets replaced with its final value

☐ Compiler ■ Linker

☐ Assembler ☐ Loader

2. The imm in `beq x0 x1 LABEL` gets replaced with its final value

☐ Compiler ☐ Linker

■ Assembler ☐ Loader

3. Pseudo instructions are outputted

■ Compiler ☐ Linker

☐ Assembler ☐ Loader

4. Physical addresses are assigned

☐ Compiler ☐ Linker

☐ Assembler ■ Loader

5. The symbol table is read

☐ Compiler ■ Linker

■ Assembler ☐ Loader

Problem 8 ***Gotta Cache 'em All*** **(19 points)**

Consider a 8-way set associative cache with 64 B blocks, and 64 total blocks as part of a 16 bit physical address

(a) Given the machine specs above, how big is each field?

Tag: _____ bits

Solution: $16 - 6 - 3 = 16 - 9 = 7$ bits

Index: _____ bits

Solution: 3 bits

Offset: _____ bits

Solution: 6 bits

Now imagine we use the same cache on the following RISC-V code:

```
.data:
    arr:  .byte 0, 1, 2 , ...  255 # All values from 0 to 255
.text:
         # ASSUME A WORKING PROLOGUE
    la a0 arr
    li a1 256
         #Scramble randomizes the elems of arr
    jal scramble
         # Assume t0 = 0, t1 = 256, s0 = A, s1 = B, s2 = C
         # START OF HIT RATE
Start:
    beq t0 t1 End      # Iterate 256 times
    add t2 a0 t0
    lbu t2 0(t2)      # t2 = arr[t0]
    add t3 s0 t2
    lw t3 0(t3)      # t3 = A[t2]
    add t4 s1 t2
    lw t4 0(t4)      # t4 = B[t2]
```

```
        add t3 t3 t4
        add t4 s2 t0
        sw t3 0(t4)      # C[t0] = t3 + t4
        addi t0 t0 1
        j Start
            # END OF HIT RATE
End:
            # ASSUME A WORKING EPILOGUE
```

Let `scramble` be a function that randomly sorts the elements of an array. Additionally assume that:

- A is located at 0x1000
- B is located at 0x2000
- C is located at 0x3000
- arr is located at 0x4000
- Our cache is empty when reaching `# START OF HIT RATE`

(b) What is the best case hit rate for this code? Write your answer as a fraction.

Hit Rate = _____ / _____

Solution: $\frac{63}{64}$

(c) What is the worst case hit rate? Write your answer as a fraction.

Hit Rate = _____ / _____

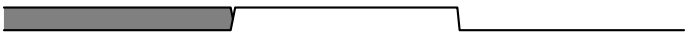
Solution: $\frac{63}{64}$

(d) Now assume that we can modify the associativity without changing any other property. What is the minimum associativity for which the best case hit rate can equal the best case hit rate with 8 way set associative?

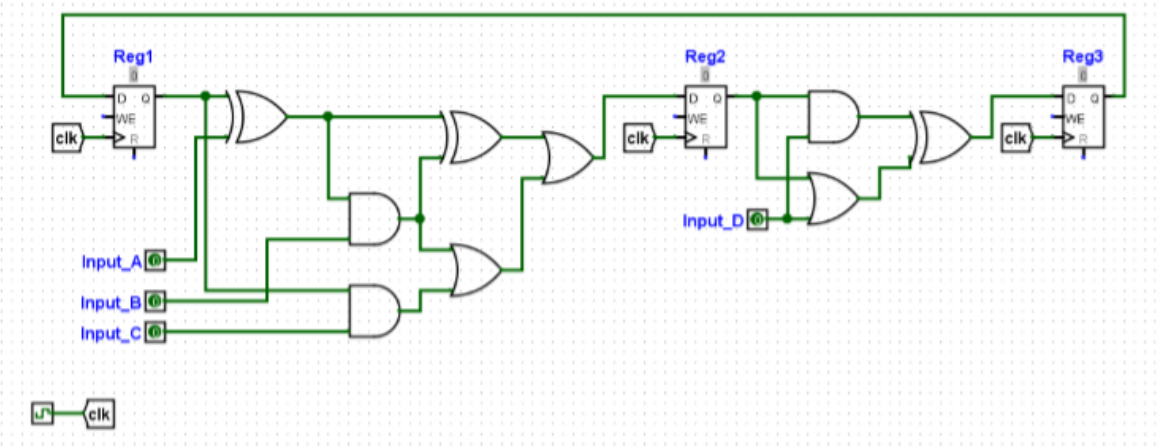
- ○ 1-way (Direct Mapped)
- ● 2-way
- ○ 4-way
- ○ 8-way
- ○ 16-way
- ○ Fully Associative

Problem 9 ***SDS*** **(20 points)**

For the following question, you'll be asked to draw waveform diagrams. For reference, in the diagram below, the first region indicates an "undefined" signal, the second region indicates a signal of "high" or 1, and the third region indicates a signal of "low" or 0.

Undefined → High → Low 

Take a look at the following circuit:



We have a register clk-to-Q time of 5ps, a hold time of 2ps, and a setup time of 3ps. AND and NAND gates have a delay of 5ps, OR and XOR gates have a delay of 6ps, and NOT gates have a delay of 1ps. Assume that our inputs A, B, C, and D arrive **on the rising edge of the clock**.

Which gates make up the critical path in the circuit above? Your answer should be correctly ordered from left to right, e.g. NOT → OR → NAND.

- ○ OR → XOR
- ○ AND → XOR
- ○ XOR → XOR → OR
- ○ AND → XOR → OR
- ○ AND → OR → OR
- ● XOR → AND → OR → OR
- ○ XOR → XOR → OR → AND → XOR
- ○ AND → OR → OR → OR → XOR

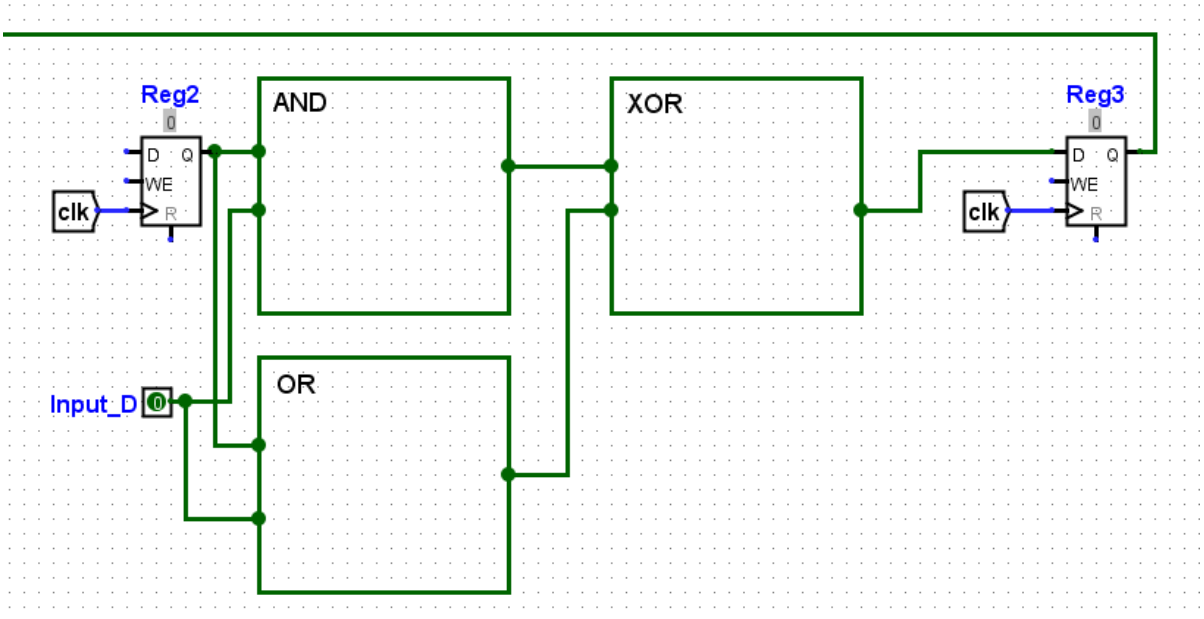
What is the critical path delay in the circuit?

- ○ 11 ps
- ○ 17 ps
- ○ 19 ps
- ○ 23 ps
- ○ 25 ps
- ○ 26 ps
- ○ 28 ps
- ● 31 ps
- ○ 40 ps
- ○ 42 ps

Of the clock frequencies below, select the **highest** frequency that will meet the timing requirements of the circuit.

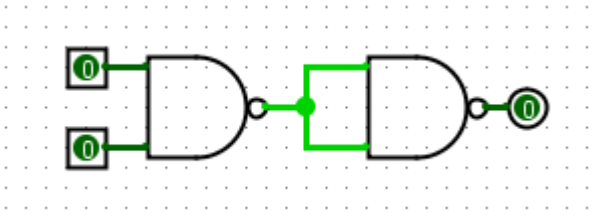
- ○ 100 GHz
- ○ 50 GHz
- ● 25 GHz
- ○ 20 GHz
- ○ 10 GHz
- ○ 5 GHz

There's a big sale on 2-input NAND gates, and we decide to take advantage of it. We'd like to convert the portion of the circuit on the previous page **between Reg2 and Reg3** so that the logical result is equivalent, but it uses exclusively 2-input NAND gates. In the boxes on the following page, convert each of the three logic gates to a NAND-gate only representation individually. Recall that with inputs A and B, a NAND gate will return 0 if A == 1 && B == 1 and 1 otherwise. **Hint:** it may be useful to also think about how to make a NOT gate.



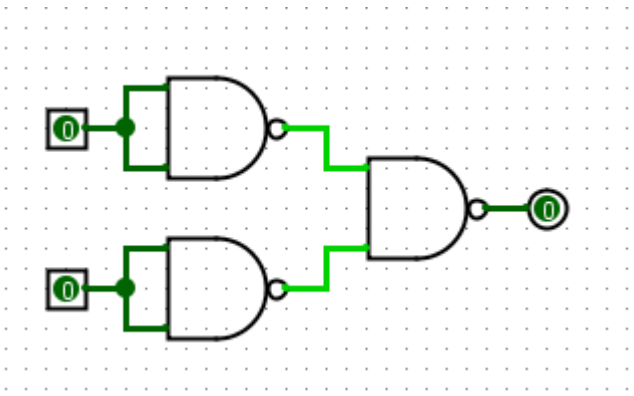
AND

Solution:



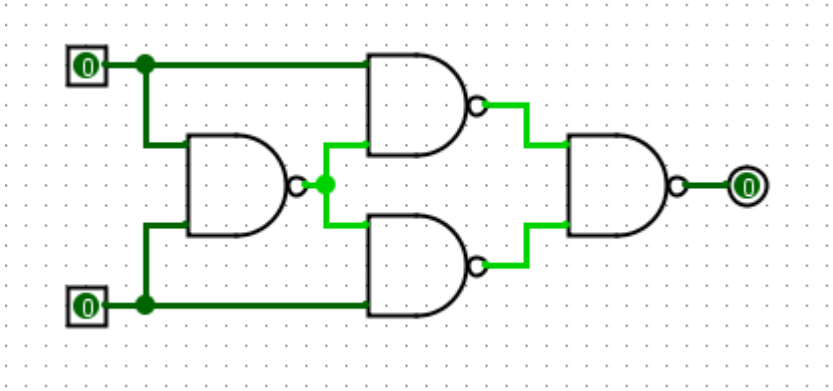
OR

Solution:



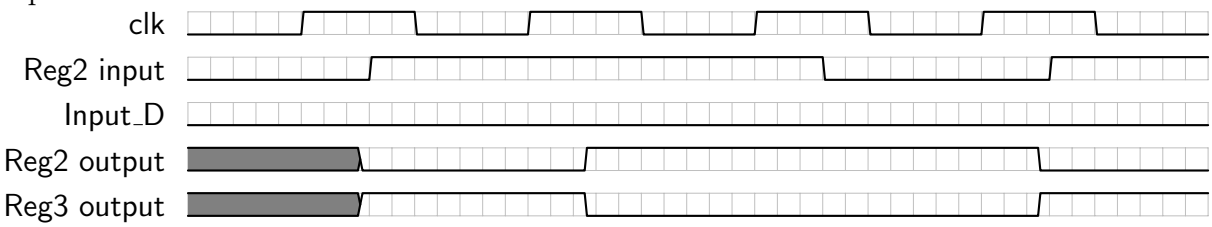
XOR

Solution:



Let us now consider only the portion of the circuit between Reg2 and Reg3. Assume that the clock period (rising edge to rising edge) is 100 ps, registers have a clk-to-Q delay of 25ps and a setup and hold time of 20ps, and all gates have a delay of 5ps. Choose the waveform with the correct outputs for Reg2 and Reg3.

Option A:



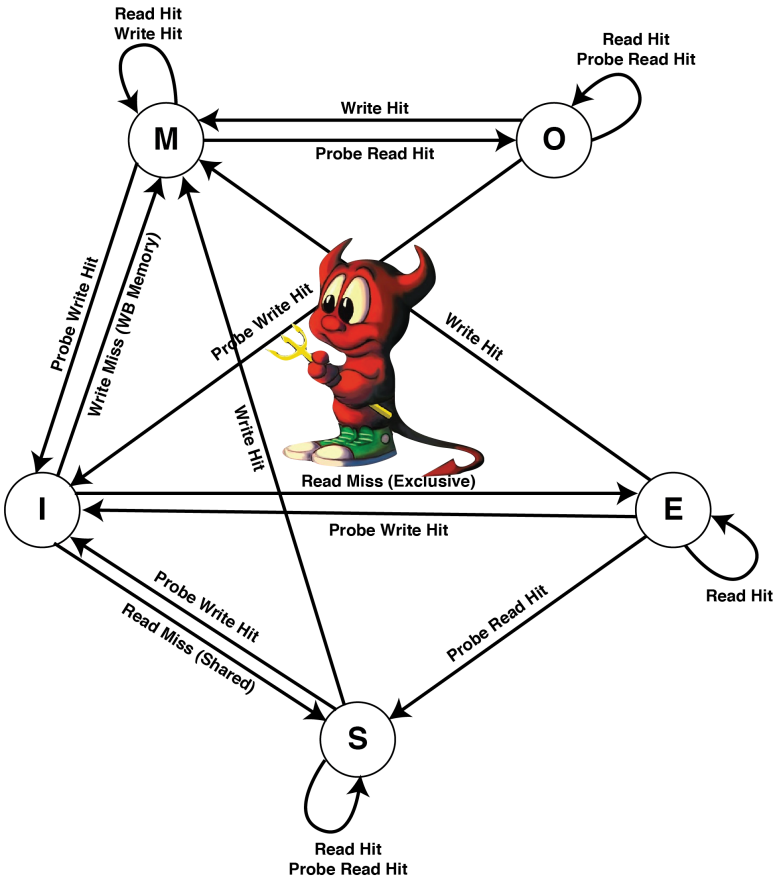
Option B:

Option C:

Option D:

○ A ○ C

● B ○ D


Read Hit
Write Hit
Read Hit
Probe Read Hit
Write Hit
Probe Read Hit
M
O
Probe Write Hit
Write Miss (WB Memory)
Probe Write Hit
Write Hit
Write Hit
Read Miss (Exclusive)
I
E
Probe Write Hit
Read Hit
Probe Write Hit
Read Miss (Shared)
Probe Read Hit
S
Read Hit
Probe Read Hit