[Midterm] Past Exams - 2020 and Older #490

J Jero Wang STAFF
Last year in Exam – Midterm

925 VIEWS

You can find the past exams here: https://cs61c.org/fa23/resources/exams/. Please check the linked past Piazza/Ed Q&A PDFs first before asking here. Many of the questions are already answered in those! Video walkthroughs (if available), are also linked on that page!

When posting questions, please reference the semester, exam, and question in this format so it's easier for students and staff to search for similar questions:

Semester-Exam-Question Number

For example: **SP22-Final-Q1**, or **SU22-MT-Q3**

Anonymous Elk 1y #490beb ✓ Resolved

Summer 2020 mt1

4d. How did they find the immediate?

Also, can someone explain the logic behind the ans to 4e

♡ ···

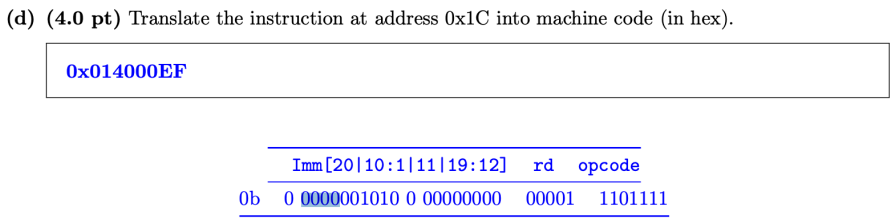
E Erik Yang STAFF 1y #490bfc

#490bad

♡ ···

Anonymous Elk 1y #490bfe

i dont think that's the answer to my q

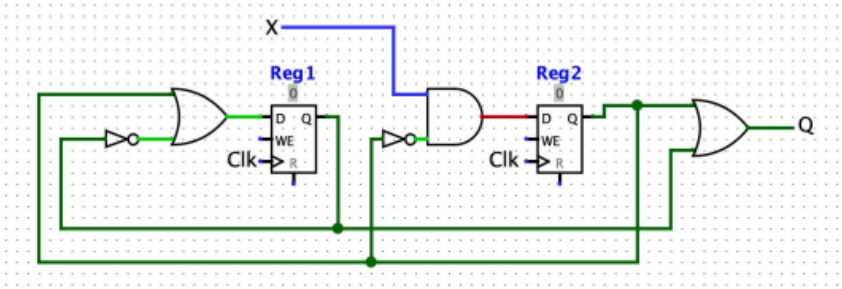


♡ ···

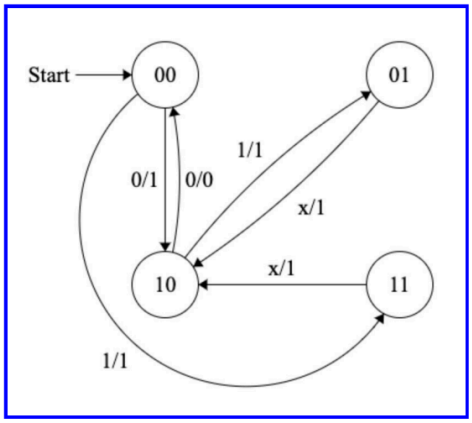
Anonymous Llama 1y #490bde ✓ Resolved

FA19-Final-Q5c, could anyone explain how we can assign values in registers in circuits? I think we can only control the input X not R1 and R2, otherwise there will be discrepancies in the wire flow?

Consider the following circuit:



c) Represent the circuit above using an equivalent FSM, where X is the input and Q is the output, with the state labels encoding Reg1Reg2 (e.g., "01" means Reg1=0 and Reg2=1). We did one transition already.



♡ ···

S Sam Xu STAFF 1y #490bed

We can assign initial value to the register. After the first time clock goes high, the registers value are up to the source. In logism, you can click the register to set up the inital value

♡ 1 ···

Anonymous Llama 1y #490bdd ✓ Resolved

For FA20-Final-Q3B version 1, I think the FSM will output 1 when we see 3 consecutive 1s and 0 otherwise, but I think the answer table is not corresponding to this logic?

Solutions
Version 1
Input 1: 111111111111111
Output 1: 001001001001001
Input 2: 001011011101111

5

Output 2: 000000000100010

current state	input	next state	output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

♡ ···

E Erik Yang STAFF 1y #490bfa

it sees 3 1's in a row, and then goes back to the initial state after outputting 1. The answer table corresponds to the logic

♡ ···

Anonymous Llama 1y #490bfd

Sorry I am still confused...if that's the case, then when the current state is 01 and the input is 0, the next state should be 10 instead of 00, right?

♡ ···

Anonymous Hawk 1y #490bcc ✓ Resolved

SP18-Final-Q9

Problem 9 ***[F-1] Floating Point*** **(13 points)**

IEEE 754-2008 introduces half precision, which is a binary floating-point representation that uses 16 bits: 1 sign bit, 5 exponent bits (with a bias of 15) and 10 significand bits. This format uses the same rules for special numbers that IEEE754 uses. Considering this half-precision floating point format, answer the following questions:

(a) For 16-bit half-precision floating point, how many different valid representations are there for NaN?

Solution: $2^{11} - 2$

(b) What is the smallest positive non-zero number it can represent? You can leave your answer as an expression.

Solution: bias $= 2^{5-1} - 1 = 2^4 - 1 = 15$
Binary representation is: 0 00000 0000000001
$= 2^{-14} * 2^{-10} = 2^{-24}$

Why is the smallest non-zero positive number not 2^-15*2^-10 since the exponent should be 0, then adding the bias makes it 2^-15, and the smallest mantissa is 2^-10?

♡ ···

V Vinay Agrawal STAFF 1y #490bdb

For denormalized fp, we need to add one to the bias expression, so (0 - 15 + 1) = -14

♡ ···

Anonymous Pig 1y #490bcb ✓ Resolved

SU20-MT1-Q4c:

I'm confused as to how they got 14 bits as the range. The regular branch instruction has [1:12] bits plus the implicit bit, so 13 bits. Then, 2 more bits can be added on top for the 16-bit branch instruction, one for each register, so that would be 15 bits in total. Doing the same counting, there are [-2^14, 2^14 - 2] memory address bytes can be represented via 15 bits, and dividing this by 2 gives [-2^13, 2^13 - 1]

A. (0.75 pt) `<lower bound>`

-4096

$[-2^{12}, 2^{12} - 1]$ Since we now only need 4 bits for the register fields, for Branch instructions, the immediate field will have 14 bits.

14 bits -> range = [-2^13, 2^13 - 2 (since imm[0] is set to 0)] -> divide each bound by 2 since half-word (2-byte) instructions

B. (0.75 pt) `<upper bound>`

4095

$[-2^{12}, 2^{12} - 1]$ Since we now only need 4 bits for the register fields, for Branch instructions, the immediate field will have 14 bits.

♡ 4 ···

Anonymous Llama 1y #490bdf

^^Same question here! I wonder why this problem is actually SU20-MT1-Q3C and the staff solution provided answers as -8192, 8191 instead of -4096, 4095 as given in the review session...

3. RISC-V!

For each of the following, write a simple RISC-V function with one argument. Follow calling convention, use register mnemonic names (e.g., refer to t0 rather than x6), and add commas and a single space between registers/arguments (e.g. `addi a0, a1, 2`). If you do not follow this, you may be misgraded!

(a) Leave your answers fully simplified as integers. **Do not leave powers of 2 in your answer!** Feel free to use a calculator to simplify your answer.

You want to build a mini RISC-V instruction architecture that only supports 16 registers, which allows the length of the register fields to be shortened. Assuming that you use the extra bits to extend the immediate field, what is the range of half-word instructions that can be reached using a branch instruction in this new format? [`<lower bound>`, `<upper bound>`]

i. (0.75 pt) `<lower bound>`

-8192

$[-2^{13}, 2^{13} - 1]$ Since we now only need 4 bits for the register fields, for Branch instructions, the immediate field will have 14 bits.

ii. (0.75 pt) `<upper bound>`

8191

$[-2^{13}, 2^{13} - 1]$ Since we now only need 4 bits for the register fields, for Branch instructions, the immediate field will have 14 bits.

♡ 2 ···

Anonymous Partridge 1y #490bea

I'm also confused of why we're dividing by 2, isn't a half-word 16 bits, so we should be dividing by 16?

♡ ···

Justin Yokota STAFF 1y #490bec

Replying to Anonymous Partridge

How much data is stored at each memory address?

♡ ···

Anonymous Partridge 1y #490bee

Replying to Justin Yokota

1 byte, got it. So we divide by 2 since we need 2 bytes per half-word. This still doesn't clarify which answer is correct though, the one in the review slides [-4096, 4095] or the one in the solutions [-8192, 8191]?

♡ ···

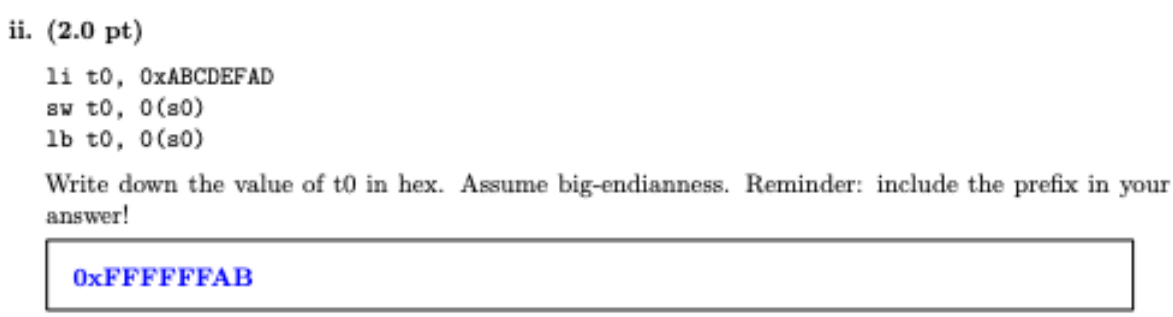
N Nikhil Kandkur STAFF 1y #490bef

#607aad I made a mistake during the review session. Here is a full explanation for the problem.

♡ 1 ···

Anonymous Shrew 1y #490bbe ✓ Resolved

SU20-MT1-Q3d: What is s0 in this context? and why is it that we only have the AB portion in the solution?



ii. (2.0 pt)

```
li t0, 0xABCDEFAD
sw t0, 0(s0)
lb t0, 0(s0)
```

Write down the value of t0 in hex. Assume big-endianness. Reminder: include the prefix in your answer!

0xFFFFFFAB

♡ ···

S Su-Ann Ho STAFF 1y #490bbf

You can assume `s0` contains a valid memory address we can load/save to. According to the reference sheet, `lb` loads **one byte** of memory at address `rs1 + imm` (in this case, `s0 + 0`) and **sign extends it**. The `sw` instruction saved the value "0xABCDEFAD" into `0(s0)`.

This means that the byte `0xAB` is at `0(s0)`, `0xCD` is at `1(s0)`, `0xEF` is at `2(s0)`, and `0xAD` is at `3(s0)`.

When we load the byte at memory address `0(s0)`, we get `0xAB`. 0xA = 0b1010, and because the most significant bit is 1, we sign-extend our value in `t0` with 1s, giving us the value `0xFFFFFFAB`.

♡ ···

Anonymous Llama 1y #490baa ✓ Resolved

It seems that there's no solution to Fall 2020 Midterm Q3A?

♡ ···

J Jero Wang STAFF 1y #490bab

#490aea

Take a look at the rewritten solutions.

♡ ···

Anonymous Llama 1y #490bac

I think #490aea is Q2 and the problem in the mt2 rewritten solution is different from that in the blank sheet.

♡ ···

J Jero Wang STAFF 1y #490bba

Replying to Anonymous Llama

The blank sheet does not have any specific variant; my comment applies to all of the FA20 exams.

I would recommend working through the rewritten solution, since the raw FA20 exams are extremely difficult to read.

♡ 1 ···

Anonymous Llama 1y #490bbb

Replying to Jero Wang

I have read through the rewritten solution for FA20-MT2-Q3A, but I think the question in the blank sheet is slightly different, asking for the machine code of a string instead of a RISC-V instruction.

For this question, do we translate each letter in "INSTRUCTION" into a hex number using the ASCII table, and concatenate them together? Then what if the total number of bits is greater than 32? Do we split up to take only the hex numbers of only "INST"?

Part A — 2 pts

What is the machine code (in hex) of `INSTRUCTION`? **Do NOT prefix your solution with `0x`.** Please pad your answer to a full 4 bytes when submitting if necessary. **See Gradescope for your specific instruction.**

♡ ···

Justin Yokota STAFF 1y #490bda

Replying to Anonymous Llama

INSTRUCTION was replaced with a randomly-generated instruction for each student in the real exam (this exam was run on PrairieLearn, so we played around with random question generation. Any similar ALL CAPS likely reflects randomized values that were given per student.

♡ ···

Anonymous Llama 1y #490bdc

Replying to Justin Yokota

oops, I guess I understand this wrong. Thanks for the reply!

♡ ···

Anonymous Elk 1y #490aed ✓ Resolved

sp20 mt1

q3 part c

I ran the calculations over and over but still don;t get how after negationwe get 0xFFFFFF81.

Shouldn't it be 0xFFFFFF80 instead.

My logic is

-2 in binary is 1111 1110

On negation, this part becomes= 0000 0001, the rest is 111....

on shifting we get

1111 1111 1111 1111 ..... 1000 0000

since we took the 1 at the end and put it in the beginning.

This is = 0xFFFFFF80 right?

Andrew Liu STAFF 1y #490afc

We start with:

```
fun->i[0] = -1

[0] 0b1111 1111 = -1 (signed) = 2^8 - 1 (unsigned)
[1] 0b0000 0000 = 0
[2] 0b0000 0000 = 0
[3] 0b0000 0000 = 0
```

Then

```
fun->u[0] *= 2

[0] 0b1111 1110 = -2 (signed) = 2^8 - 2 (unsigned)
[1] 0b0000 0000 = 0
[2] 0b0000 0000 = 0
[3] 0b0000 0000 = 0
```

Then

```
fun->t *= -1
fun->t >>= 1

[0] 0b1111 1110 = 0
[1] 0b0000 0000 = 0
[2] 0b0000 0000 = 0
[3] 0b0000 0000 = 0

[int] 0b 1111 1110 0000 0000 0000 0000 0000 0000 (little endian in memory)
->    0b 0000 0000 0000 0000 0000 0000 1111 1110 (actual representation)
->    0b 1111 1111 1111 1111 1111 1111 0000 0010 (multiply by -1)
->    0b 1111 1111 1111 1111 1111 1111 1000 0001 (right shift by 1)
->    0b 1000 0001 1111 1111 1111 1111 1111 1111 (back to little endian)

[0] 0b 1000 0001
[1] 0b 1111 1111 = -1 (signed)
[2] 0b 1111 1111
[3] 0b 1111 1111
```

Lastly (for sake of completeness)

```
s[0] = '\0'

[0] 0b 0000 0000
[1] 0b 1111 1111 = -1 (signed)
[2] 0b 1111 1111
[3] 0b 1111 1111

-> 0b 0000 0000 1111 1111 1111 1111 1111 1111 (little endian in memory)
-> 0b 1111 1111 1111 1111 1111 1111 0000 0000 (actual binary representation)
= -0b 0000 0000 0000 0000 0000 0001 0000 0000 (negate for ease of reading)
= -256
```


Anonymous Shrew 1y #490bca

When we multiply by -1, do we just do negation, or do we need to do negation and add 1?

Anonymous Elk 1y #490abf ✓ Resolved

su20 mt1

ii. (6.0 pt)

```
GenericLink* store_string(char* str) {
    /* store_string takes in a string, and returns a
  pointer to the "head" of the GenericLinkedList
  holding the string, i.e. the link containing the first char*/

  <YOUR CODE HERE>
}
```

```
GenericLink* string = store_char(str++); GenericLink curr = string; while (str)
{ curr->next = store_char(str); curr = curr->next; str++; } return string;
```

why do we do s++ in line 1?

shouldn't it be something like *str[0]?

also when we do store_char(str), shouldn't we dereference str?

Jero Wang STAFF 1y #490aeb

The `str++` just increments `str` so that it points to the second character of the string (since you've just stored the first character in the first node of the linked list). Yes, you should dereference `str`.

Anonymous Elk 1y #490aef

but we haven't stored the first char yet .

I understand incrementing in the while loop but the very first line? Won't we end up not accounting for the char at the 0th index?

Jero Wang STAFF 1y #490afd

Replying to Anonymous Elk

It's a postfix operator. To expand it, what really happens is this

```
GenericLink* string = store_char(str);
str++;
```

In other words, it increments `str` by 1, but it "returns" the old value and passes that to `store_char`.

Anonymous Elk 1y #490afa

also, just to confirm, we'd do store_char(*str[0]) instead of store_char(str) in order to dereference it and get exactly the first char, right?

Jero Wang STAFF 1y #490afe

Replying to Anonymous Elk

You should do `*str` or `str[0]`, but not both. Keep in mind that `str[0]` is just shorthand for `*(str + 0)`, so if you have `*str[0]`, you're really doing `**(str + 0)` or `**str`.

Anonymous Elk 1y #490abe ✓ Resolved

FA20 Q2

I'm really confused. How do I attempt this question?

There is so much information and I don't understand the solution.

Jero Wang STAFF 1y #490aea

Which exam is this? MT1 (Quest) or MT2?

Also, FA20 questions are really hard to read since they were randomized (so the generated PDFs contain a lot of variable names that were replaced in the actual questions). If you want to see an example of what a question would actually look like, take a look at the "rewritten solutions" (courtesy of Peyrin).

Anonymous Elk 1y #490afb

mt2

Jero Wang STAFF 1y #490aff

Replying to Anonymous Elk

#490acb

Anonymous Hawk 1y #490abd ✓ Resolved

SP18 MT1

Problem 3 *C Analysis* **(10 points)**

The CS61C Staff is creating songs in preparation of the grading party. Consider the following program:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Song {
    char *title;
    char *artist;
} Song;

Song * createSong() {
    Song* song = (Song*) malloc(sizeof(Song));
    song->title = "this old dog";
    char artist[100] = "mac demarco";
    song->artist = artist;
    return song;
}

int main(int argc, char **argv) {
    Song *song1 = createSong();
    printf("%s\n", "Song written:");
    printf("%s\n", song1->title); // print statement #1
    printf("%s\n", song1->artist); // print statement #2

    Song song2;
    song2.title = malloc(sizeof(char)*100);
    strcpy(song2.title, song1->title);
    song2.artist = "MAC DEMARCO";
    printf("%s\n", "Song written:");
    printf("%s\n", song2.title); // print statement #3
    printf("%s\n", song2.artist); // print statement #4

    return 0;
}
```

malloc

Immutable string

Stack variable (createSong)

Stack variable (main)

Malloc (main)

(a) What type of address does each value evaluate to? Fill in the entire bubble.

i. song1
- ○ Stack address
- ○ Static address
- ● Heap address
- ○ Code address

ii. song1->title
- ○ Stack address
- ● Static address
- ○ Heap address
- ○ Code address

iii. song1->artist
- ● Stack address
- ○ Static address
- ○ Heap address
- ○ Code address

iv. &song2
- ● Stack address
- ○ Static address
- ○ Heap address
- ○ Code address

v. song2.title
- ○ Stack address
- ○ Static address
- ● Heap address
- ○ Code address

(b) Will all of the print statements execute as expected?
- ○ Yes
- ● No

If you answered yes, leave this blank. If you answered no, write the number(s) of the print statement(s) which will not execute as expected.

Solution: print statement #2

why does the 2nd print statement fail?

♡ ···

J Jero Wang **STAFF** 1y #490adf

The second one fails because `song->artist` points to the stack array `artist`, and accessing values on the stack after the function returns may cause undefined behavior.

♡ ···

Anonymous Llama 1y #490abc ✓ Resolved

For fa20-Final-S2-Q2A, the floating number is altogether 33 bits instead of 36 bits, then how can we use the hexadecimal numbers as an intermediate step to represent?

Part A — 3 pts What is the median of the positive non-NaN floats? (including +0, denorms, and ∞, which is an odd number of numbers) Write your answer as a decimal number, like 7.65; something a first grader could understand.

Solutions

1.5

Explanation

One useful thing to note is that floating point numbers maintain the order of sign-magnitude numbers; for example, 0x00000000 < 0x00000001 < 0x00000002 < 0x0FFFFFFF< 0x10000000 < 0x10000001, if we interpret the hexadecimal as an IEEE standard floating point number. We can thus find the median by looking at the bit patterns, finding the median of a sign-magnitude number, and converting that to floating point. The smallest number is 0x000000000, and the largest number is 0x0FFF80000 (+∞). The median of these two numbers (treating them as sign-magnitude numbers) is (0x000000000+0x0FFF80000)/2 = 0x07FFC000, which we convert to floating point as 1.5.

♡ ···

J Jero Wang **STAFF** 1y #490ade

A 33-bit hex number is similar to a 36-bit hex number, except the top 3 bits are always 0. You would still have 9 hex characters to represent this number.

♡ ···

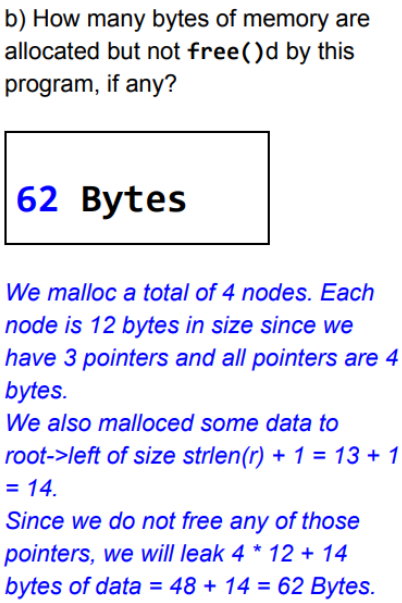
Anonymous Llama 1y #490bcf

For the top 3 bits do you mean the most significant ones or the least significant ones? I am still a little confused as to why we can equivalently treat the 33 bits as 36 bits to find the median. Could you explain a little more about this?

♡ ···

Anonymous Llama 1y #490abb ✓ Resolved

For SP20-MT1-Q4b), why don't we need to take the null terminator at the end of the character array r into account? I thought this question is similar to that of SU20-MT1-Q2b, but that one takes the \0 into account...



♡ ···

Jero Wang STAFF 1y #490add

From what I'm reading, it seems like `strlen(r)` is 13 here, and the solution adds 1 to account for the null terminator (the code also allocates `strlen(r) + 1` bytes of memory). If I'm misunderstanding your question, can you clarify where it's not taking the null terminator into account?

♡ ···

Anonymous Llama 1y #490bce

Sorry for the confusion! I mean the `malloc(strlen(r)+1)` takes 13 bytes for the string `r` + 1 byte for the +1 in `malloc()` and +1 byte for the null terminator = 15 bytes. Then why is it 14 bytes instead of 15 bytes?

Because in SU20-MT1-Q2b, `calloc(sizeof(s) + 1, sizeof(char))` takes 15 bytes, " (Since the compiler knows the length of the s array since it is stored on the stack which is 13 characters plus a null terminator so 14 bytes long)."

♡ ···

Anonymous Llama 1y #490aba ✓ Resolved

For SP20-Final-Q6, &w evaluates to w, which is also a hard-coded integer value, then why it is not a static address?

```
void foo() {
    int64t w = 4;
    int64t* u = malloc(100 * sizeof(w));
    int64t** v = &u;
    ...
}
```

Q6.7 (2 points) What type of value does `&w` evaluate to?

- ● **Stack Address**
- ○ Static Address
- ○ Heap Address
- ○ Not an Address

Solution: `&w` is the address of the local variable `w`, and we know that local variables are stored on the stack. Thus the value `&w` is a stack address.

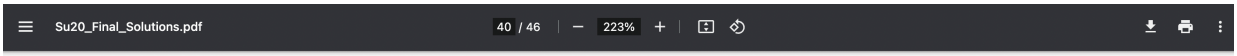
♡ ···

Jero Wang STAFF 1y #490adc

Unlike strings (which are technically an array of characters), integers are more like characters, in that they are stored directly. In general, `&x` is not the same as `x`. In this case, `&w` is a pointer to the variable `w`, which is stored on the stack since it's a local variable in a function.

♡ 1 ···

Anonymous Cattle 1y #490aaf ✓ Resolved



Mantissa: 0201

Together: 0 111 0201

ii. (2.5 pt) What is the decimal value of the largest positive normalized float? Express your answer in terms of powers of 3 from largest to smallest. Ex: `2*3^8+1*3^2+2*3^0`. Leave out the zero bits and do NOT add spaces. Do not add parenthses for powers!

`1*3^15+2*3^14+2*3^13+2*3^12`

Sign : 0

exp digits: $221 = 2 * 3^2 + 2 * 3 + 1 = 25 \rightarrow 25 - 10 = 15$

Mantissa: 2222

$3^{15} * (1.2222_3) = 1 * 3^{15} + 2 * 3^{14} + 2 * 3^{13} + 2 * 3^{12} = 31355019$

how come the final answer doesn't include +2*3^11 since there are 4 2's in the mantissa?

♡ ···

Jero Wang STAFF 1y #490adb

I think you're right, there should be a $2 \cdot 3^{11}$ term in the answer since there are 4 mantissa bits.

♡ ···

Anonymous Llama 1y #490aae ✓ Resolved

For Su20-MT1-Q2(a), is there a more thorough solution to approach the answer, or are there any notes/slides I can particularly refer to?

♡ ···

Jero Wang STAFF 1y #490ada

Not really...in general, here are a few things:

- String literals are always in static
- Local variables (variables in functions) are always on the stack

- Arguments are always on the stack
- Stack arrays `some_var[]` are on the stack, even when they are set equal to a string literal

I'd also recommend watching out for what the question asks for. Sometimes, we ask where the value of a variable lives, while other times, we ask where a memory address points to in memory.

♡ 1 ···

Anonymous Llama 1y #490bcd

I kind of figured it out. But can I still ask about the last two questions of this part? On what conditions are the address on the heap?

```
typedef struct node {
    void *data;
    struct node *nxt;
    struct node *prv;
} node;

void push_back(node *list, void *data) {
  node *n = (node *) malloc(sizeof(node));
  n->data = data; n->nxt = list; n->prv = list->prv;
  list->prv->nxt = n; list->prv = n;
}

int main() {
  char *r     = "CS 61C Rocks!";
  char s[]    = "CS 61C Sucks!";
  node sentinel; sentinel.nxt = &sentinel; sentinel.prv = &sentinel;
  push_back(&sentinel, r);
  push_back(&sentinel, s);
  push_back(&sentinel, &sentinel);
  push_back(&sentinel, calloc(sizeof(s) + 1, sizeof(char)));
}
```

vi. (0.75 pt) `sentinel.prv->data`

- ○ Static
- ● Heap
- ○ Stack
- ○ Code

vii. (0.75 pt) `sentinel.prv->prv`

- ○ Code
- ● Heap
- ○ Stack
- ○ Static

♡ ···

Anonymous Elk 1y #490fd ✓ Resolved

7. C Programming

(a) Consider the following structure definition. Assume we are using a 32-bit machine.

```
struct foo {
  char a;
  char *b;
}
```

And the following C code

```
void bar(struct foo *f){
  int i;
  ....
  for(i = 0; i < 5, ++i){
    baz(f[i].b);
  }
  ....
}
```

i. (2.0 pt) What is `sizeof(struct foo)`?

> **8**

ii. (2.0 pt) What is `sizeof(f)`?

> **4**

iii. (4.0 pt) Translate the line baz(f[i].b) into RISC-V assembly. Assume that f is in S5 and i is in S6. You should use only 4 instructions and you can only use a0 as a temporary. **You may NOT use mul, div, or rem!**

```
sll a0 s6 3
add a0 a0 s5
lw a0 4(a0)
jal baz
```

Can someone explain why in (iii) we did sll and then added in the next line? I don't see how sll works in this case.

♡ ...

Jero Wang STAFF 1y #490acf

Which exam is this?

The first line should be `slli` , but in general, this is what it does:

`slli a0 s6 3` : gets `i` and calculates the offset from the start of the array. Each struct in the array is 8 bytes (see part i), so we multiply the index by 8 (aka left shift by 3).

`add a0 a0 s5` : adds the offset to the start of the array

`lw a0 4(a0)` : loads the `b` member of the correct element within the array

`jal baz` : call `baz`

♡ ...

Anonymous Llama 1y #490fc ✓ Resolved

For Fa20-Quest-Q1, I cannot find the solution in the answer key???

♡ ...

E Eddy Byun STAFF 1y #490aad

Sorry, it's there in the Rewritten solutions:
https://inst.eecs.berkeley.edu/~cs61c/fa22/pdfs/exams/fa20-quest-sols-rewritten.pdf

♡ ···

 Elana Ho 1y #490fb ✓ Resolved

Q1.3 (3 points) The following code is executed on a 32-bit little-endian system.

```
#include <stdio.h>
int main() {
    int doThis = 0x6C697665;
    char *dont = (char *)(&doThis);
    printf("A: ");
    for (int i = 0; i < 4; i++) {
        printf("%c", dont[i]);
    }
    printf("\n");
}
```

What is printed when this program is run? If it crashes/segfaults, write **n/a**.

Solution: The program prints out `A: evil`.
The program doesn't segfault because C treats data as bits to be interpreted with regards to their type and trusts that the programmer's type casts are correct. This means that we can treat an `int` as a series of `char`s. Little-endian means the least significant byte (0x65) is located at the lowest memory address, which is accessed first, so the program prints `evil`, not `live`.

for spring 2020 midterm 1, where did "evil" and "live" come from??

♡ ···

E Eddy Byun STAFF 1y #490aac

We store doThis as followed in a little endian system

Memory address a: 0x65

Memory address a + 1: 0x76

Memory address a + 2: 0x69

Memory address a + 3: 0x6C

The ASCII values for 0x65 is 'e', 'v' for 0x76, 'i' for 0x69, and 'l' for 0x7C, which is how we print out "evil"

♡ ···

Anonymous Finch 1y #490ed ✓ Resolved

why does calloc allocate 15 instead of 14 bytes?

(b) (3.0 pt) How many bytes of memory are allocated but not `free()`d by this program, if any? (assuming we have not called `free_list`) (Leave your answers as an integer. Do not include the units, we are telling you it's bytes after all!)

63

Each node will be `sizeof(node) == 12` bytes. We have allocated 4 nodes so 48 bytes.

We also made a `calloc` of 15 bytes (Since the compiler knows the length of the `s` array since it is stored on the stack which is 13 characters plus a null terminator so 14 bytes long). This means that we will leak 63 bytes.

```
int main() {
  char *r     = "CS 61C Rocks!";
  char s[]    = "CS 61C Sucks!";
  node sentinel; sentinel.nxt = &sentinel; sentinel.prv = &sentinel;
  push_back(&sentinel, r);
  push_back(&sentinel, s);
  push_back(&sentinel, &sentinel);
  push_back(&sentinel, calloc(sizeof(s) + 1, sizeof(char)));
}
```

also, for SU-MT1-2c,

wouldn't it just stop before hitting sentinel because we start from node 2 and keep checking if our new c is equal to the first node each loop?

♡ ···

Andrew Liu STAFF 1y #490fe

This is dealing with array types in C. Since we have `s` being an array type, `sizeof s` will tell us how many bytes are allocated to the array, which is 13 chars + 1 null term = 14 bytes. Adding 1 to that size of gives us the `15` bytes.

♡ ···

Anonymous Finch 1y #490de ✓ Resolved

i'm so lost, why does the "smallest positive denorm" have 2^-3+1 as an exponent? where did the +1 come from?

What's the gap (aka absolute value of the difference) between the **smallest positive** non-zero denorm and **smallest positive** non-zero norm? (Answer in decimal)

0.234375

$1.0000_2 * 2^{1-3} - 0.0001_2 * 2^{0-3+1} = \frac{1}{4} - \frac{1}{64} = \frac{15}{64} = 0.234375$

♡ ···

N Nikhil Kandkur STAFF 1y #490df

For denormalized floats:

$$\textbf{Value} = (-1)^{\textbf{Sign}} * 2^{\textbf{Exp+Bias+1}} * 0.\textbf{significand}_2$$

Exponent	Significand	Meaning
0	Anything	Denorm
1-254	Anything	Normal
255	0	Infinity
255	Nonzero	NaN

According to this photo from the FP discussion worksheet, we see that the bias here is -3, and we since we want a denorm number, we need to have our exponent = 0, but we still have to add 1 to what we get after `exponent + bias`, so this gives us `0 + -3 + 1` as our power of 2.

♡ ···

Anonymous Finch 1y #490eb

oh, i see, thank you

♡ ···

Anonymous Finch 1y #490dd ✓ Resolved

why is this not ABCDEBBC?

ABCDE << 12 = ABCD E000

+ x100 = ABCD E100

+xABC = ABCD EBBC

(d) i. (1.0 pt)

```
auipc t0, 0xABCDE # Assume this instruction is at 0x100
addi t0, t0, 0xABC
```

Write down the value of t0 in hex. Reminder: include the prefix in your answer!

0xABCDDBBC

♡ ···

N Nikhil Kandkur STAFF 1y #490ea

Remember that we work with 2's complement signed integers for addi, and since 0xABC has an MSB of 1, we sign extend that 1 into a 32 bit immediate that we then add to 0xABCDE.

♡ ···

Anonymous Finch 1y #490ec

So then wouldn't ABC become FFFF FABC? and wouldn't we get a totally different answer?

♡ ···

N Nikhil Kandkur STAFF 1y #490ee

Replying to Anonymous Finch

Hmm I think you're right, this might be an error in the solutions

♡ ···

Anonymous Finch 1y #490ef

Replying to Nikhil Kandkur

yeah i'm really confused and i'm not sure how to do this type of adding now

♡ ...

J Jero Wang STAFF 1y #490bad

Replying to Anonymous Finch

I think the printed answer is correct, here's my work:

```
  0xABCD E100 + 0xFFFF FABC
= 0xABCD E100 + (0xFFFF FABC + 0x0000 1000 - 0x0000 1000)
= 0xABCD E100 + (0x0000 0ABC - 0x0000 1000)
= 0xABCD EBBC - 0x0000 1000
= 0xABCD DBBC
```

♡ ...

Anonymous Finch 1y #490bbc

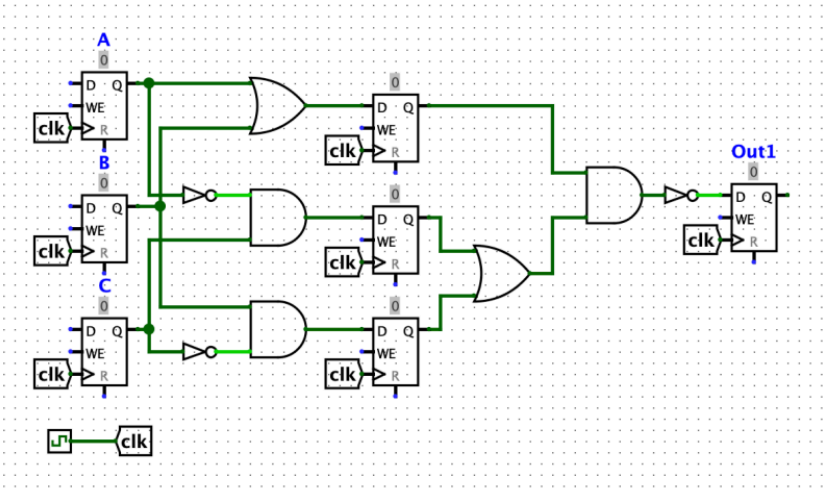
Replying to Jero Wang

how does adding 0000 1000 turn all the F's to 0's?

♡ ...

Anonymous Elk 1y #490dc ✓ Resolved

ii. Regardless of our previous answer, we decide, in order to speed our circuit up, to pipeline it by adding registers in the following places.



Pipelined Circuit

A. **(3.0 pt)** What is the maximum clock frequency of the circuit? You should leave your answer as a decimal value in terms of MHz. Please round decimal answers to 2 decimal places. Please keep both decimal places even if they are 0!

76.92

The critical path is now between the middle and end registers. It is CLK-2-Q + OR + AND + NOT + SETUP = 3 + 5 + 2 + 1 + 2 = 13 ns. Therefore, our maximum clock frequency is 1/13 ns ≈ 76.92 MHz.

B. **(4.0 pt)** What is the maximum possible hold time (in ns), so that our circuit will function as expected?

5

To find the maximum hold time, we need to find the shortest path between 2 registers: CLK-2-Q + AND = 3 + 2 = 5 ns.

Can someone explain both answers.

Especially part B. Why is it only clk-to-q + AND and not clk-to-q + AND +NOT

Andrew Liu STAFF 1y #490ff

Pipelining is not in scope for this midterm, but the maximal path is now from a middle register to Out1, and the minimal parth is from A to a middle register through an AND gate.

Anonymous Elk 1y #490aab

but isn't there a NOT gate present as well in part B

and isn't the minimal parth from A to a middle register through an OR gate since an AND Gate will have aNOT gate as well.

Jero Wang STAFF 1y #490bae

Replying to Anonymous Elk

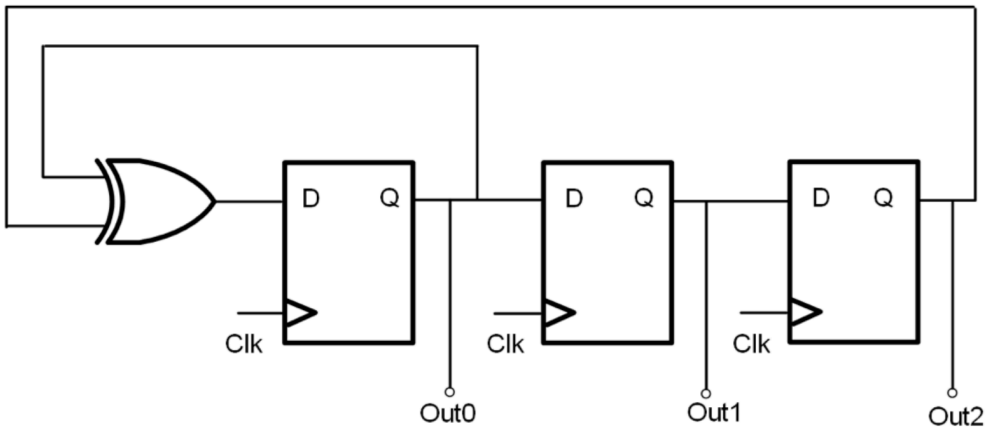
I think Andrew might be referring to register B, not register A, since there's no NOT gate for register B and it only has an AND gate before reaching one of the middle registers.

Anonymous Elk 1y #490cf ✓ Resolved

in this question

Q4: SDS

Version 1 SDS



Part B — 3 pts
In the figure in the Appendix and from Part A, the flip-flop clk-to-q delay is `CLK` ps, the setup time is `SETUP` ps, the XOR delay is `XOR` ps, and the inverter time if it applies is `INVERTER` ps. What is the minimum cycle of operation?

Solutions

Part C — 2 pts In the above figure, what is the longest hold time for the flip-flop that allows for correct operation?

why is the answer to Part B

CLK + XOR + SETUP

and not 3*CLK + XOR + SETUP

and why is the answer to C

CLK and not 3*CLK + XOR

Andrew Liu STAFF 1y #490aaa

Part B — Remember that combinational paths are only from Q-D, without passing *any* registers in the middle. Passing through a register "resets" the path in terms of calculation. So, the shortest path could be from Q (Reg1) - Out0 - D(Reg2) = 0ns, and the longest is from Q(Reg1) - Out0 - XOR - D(Reg1).

S Saloni Khule 1y #490cc ✓ Resolved

How do we solve this question. There is no solution on the solution page.

Q3: RISC-V (*10 pts*)

Part A — 2 pts

What is the machine code (in hex) of `INSTRUCTION`? *Do NOT prefix your solution with* `0x`. Please pad your answer to a full 4 bytes when submitting if necessary. See Gradescope for your specific instruction.

Solutions

Part B — 8 pts

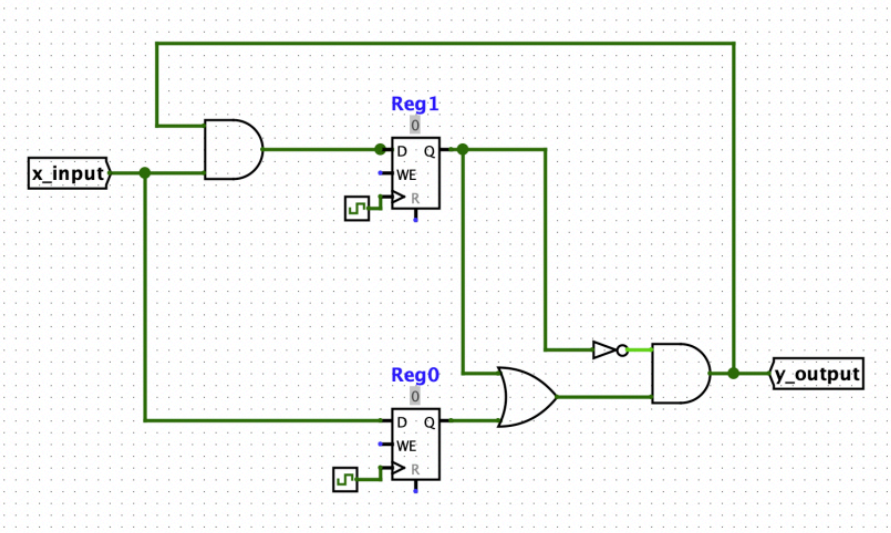
J Justin Yokota STAFF 1y #490cd

On Gradescope, each student received a different instruction for this question. As such, it is not be possible to provide a solution to this question.

Anonymous Cattle 1y #490ca ✓ Resolved

4. SDS, Logic

We will be analyzing the following circuit:



Circuit

Given the following information:

- **AND** gates have a propagation delay of 9ns
- **OR** gates have a propagation delay of 14ns
- **NOT** gates have a propagation delay of 5ns
- **x_input** switches value(i.e. 1 to 0, 0 to 1) 30 ns after the rising edge of the clk
- **y_output** is directly attached to a register
- **Setup** time is 3ns
- **Clk-to-q** delay time: 4ns

(a) **(2.0 pt)** What is the max hold time in ns?

18

Shortest CL: NOT -> AND = 5 + 9 = 14ns clk-to-q + shortest CL = 4ns+14ns = 18ns

(b) **(2.0 pt)** What is the minimum clock period in ns?

42

Critical path = clk-to-q + longest CL + setup = 30ns for x_input to change (includes clk-to-q) + 9 AND + 3 setup = 42 ns

(c) **(3.0 pt)** Regardless of your previous answers, assume the clock period is 50ns, the first rising edge of the clock is at 25 ns and x_input is initialized to 0 at 0ns. At what time in ns will y_output become 1?

102

25+50(clock period)+4(clk-to-q)+14(OR)+9(AND) = 102 ns

why is the minimum clock period 42?

isn't the longest path R0->R1, which makes the answer 14+9+9 +3(setup)+ 4(clktoq) = 39?

♡ ...

E Eddy Byun STAFF 1y #490ce

The longest path is from x_input to the input of reg1.

30ns (for x_input to switch) + 9ns (for AND gate) + 3ns (for setup) = 42ns

♡ ...

Anonymous Cattle 1y #490fa

thank you. Another general question, when doing these min clock period and max hold time questions. To find the min, max combinational path, I know we compare the paths

for registers to registers, but do we also include and compare input to register or like register to output?

♡ ···

Jero Wang STAFF 1y #490baf

Replying to Anonymous Cattle

In general, you should assume inputs and outputs would affect the min clock period and max hold time if we mention the timings of inputs/outputs, but generally, we will say to ignore them unless we're explicitly testing for it.

♡ ···

Anonymous Cormorant 1y #490bf ✓ Resolved

SP18_MT2-Q3e4

4. BSel:

○ 1 ○ 0 ● X

Solution: We want our ALU to produce rs1 as its output. We do not care what the value of our second operand is (because regardless, it isn't the output we want) and therefore it doesn't matter if we pass in the immediate or DataB.

If we want the ALU to output rs1 (and select ASel accordingly) won't the value of BSel matter? I thought we would have to set it so that the immediate (which contains 0) is added to Reg[rs1] to produce the ALU output of Reg[rs1] instead of having BSel sometimes adding whatever is in Reg[rs2], which may or may not be 0.

♡ ···

Jero Wang STAFF 1y #490ace

It doesn't matter here because the output of the ALU only matters if the condition is true (otherwise the value isn't written back to the regfile). In that case, rs2 will be 0 (that's what the instruction does), and the immediate will be 0 ("the immediate generator outputs ZERO when we have a SIZ instruction"), so it doesn't matter which value it takes here.

♡ ···

Anonymous Cormorant 1y #490be ✓ Resolved

SP18_MT2-Q1f

(f) Complete the following RISC-V procedure `jal_address_fixing` that handles address relocation for all `jal` instructions. It first calls `find_next_jal` to find a `jal` instruction that does not yet have its offset filled in (the immediate bits are all zeroes), calculates the jump offset, and fills the immediate field of the `jal` instruction.

- Fill in **one** instruction for each of the 5 blanks.
- You can assume `jal_address_fixing` has the ability to modify text segment instruction memory.
- The function `find_next_jal` returns two values: the first is the address of a `jal` instruction stored in `a0`; the second is the address of the target instruction this `jal` instruction is jumping to stored in `a1`. If there are no more `jal` instructions to fill in offsets for, it returns 0 and 0.

```
jal_address_fixing:
    jal ra, find_next_jal
    beq a0, x0, DONE
    sub a1, a1, a0          # set a1 as the jump offset
IMM_20:                       # sets imm[20]
    srai a5, a1, 20          # now a5 has imm[20]
    slli a5, a5, 31      # a5 has imm[20]
IMM_19_12:     # sets imm[19:12]
    li a3, 0xFF000
    and a3, a1, a3
    or a5, a5, a3      # now a5 has imm[20] and imm[19:12]
IMM_10_1:     # sets imm[10:1]
    li a3, 0x7FE
    and a3, a1, a3
    slli a3, a3, 20
    or a5, a5, a3      # now a5 has imm[20], imm[10:1], and imm[19:12]
IMM_11:     # sets imm[11]
```

Could slti a5 a1 0 also work for the blank #2 (currently srai a5, a1, 20)? Wouldn't both set the low bit of a5 to 1 if the number in a1 was negative?

♡ ···

J Jero Wang STAFF 1y #490acd

Yes, that should work as well.

♡ ···

Anonymous Hawk 1y #490bd ✓ Resolved

FA19-MT2-Q4

How do we figure out how much an instruction moves the PC by? Would we add the hex value of the instruction to the address of reverse after we convert it to find the hex value of the machine code instruction?

Now assume all blanks above contain a single instruction (no more, no less).

b) The address of `reverse` is `0x12345678`.

What is the hex value for the machine code of `beq x0, t0, returnnode`? **0x02500063**

c) The user adds a library and this time the address of `reverse` is `0x76543210`.

What is the hex value for the machine code of `beq x0, t0, returnnode`? **0x02500063**

Part B

`beq x0, t0, returnnode` moves the PC (program counter) forward by 8 instructions, each of which is 1 word or 4 bytes. This means the immediate = 8 * 4 = 32.

In binary, we represent 32 as 0b100000. However, since we know branch/jump immediates are always even numbers, we don't store bit 0.

imm[12]	imm[10:5]	rs2	rs1	func3	imm[4:1]	imm[11]	opcode
31	30 - 25	24 - 20	19 - 15	14 - 12	11 - 8	7	6 - 0
0	000001	00101	00000	000	0000	0	1100011

0000	0010	0101	0000	0000	0000	0110	0011
0	2	5	0	0	0	6	3

Part C

Since the number of instructions between the branch and its label does not change, the immediate value should not change regardless of the location since branches' immediates are relative to the current instruction.

♡ ···

Anonymous Lark 1y #490cb

Also, why do we move the PC forward by 8 instructions? I thought we only moved it till we reached the line with the label so that would be 7 instructions. And why do we multiply by 4? I saw earlier calculations where immediate is just the value of the offset, not the number of bytes we move forward by.

♡ ···

E Eddy Byun STAFF 1y #490db

There are 8 instructions starting from `beq x0, t0, returnnode`. `mv a1, a0`, `mv a0, t0`, `addi sp, sp, -4`, `sw ra, 0(sp)`, `jal ra reverse`, `lw ra, 0(sp)`, `addi sp, sp, 4`, `jr ra`. We multiply by 4 because each instruction is 4 bytes long.

♡ ···

E Eddy Byun STAFF 1y #490da

The PC moves by 32 because `return_node` is 8 instructions from `beq x0, t0, returnnode`. Each instruction is 4 bytes, and 8*4 = 32, which is why we move the PC by 32.

♡ ···

Anonymous Hawk 1y #490bc ✓ Resolved

SP19-MT1-Q2

I am confused on why each variable is stored in the specific place in memory.

a. zero is stored on the stack, but its value is NULL and shouldn't that be static?

b. I just don't get why this is static at all

c. I'm assuming this is stored on the heap because you malloc'ed space for it in the main function

d. I'm assuming this is code because all functions are stored in code (?)

Please explain and double check my assumptions.

♡ ···

J Jero Wang STAFF 1y #490acc

a. We're looking at where the value of `zero` is stored, not where the pointer `zero` points to. In this case, since it's a local variable within a function, it's stored on the stack/

b. It's static because `map[0].key` points to the string literal `"k"`, and dereferencing it gives you the character `k` in static memory (since all string literals are in static).

c. Yes

d. Yes

♡ ···

Anonymous Skunk 1y #490af ✓ Resolved

are all questions from the Midterm 1 exams (2020 and prior) in-scope?

which questions from the Midterm 2 exams (2020 and prior) are in scope?

♡ ···

E Eddy Byun STAFF 1y #490bb

#491d

♡ ···

Anonymous Llama 1y #490ac ✓ Resolved

FA 20 Q2: what is this question asking about?

Q2: Quest Clobber (*10 pts*)

Part A — 3 pts

Help! We have two robots, Alexa and Siri, but they're speaking the wrong languages! Alexa sends sensor data with the bits as `uint8_t`s encoding `ENCODING TYPE 1` which would later be read by Siri. However, Siri can only understand `ENCODING TYPE 2`. Your job is to write a simple function `ENCODING TYPE 1 TO 2` so that Siri can correctly convert messages received from Alexa. (E.g. if Alexa saw "-3" and encoded it as `ENCODING TYPE 1`, you'd need to return a new set of bits such that Siri would interpret those bits in `ENCODING TYPE 2` as "-3" as well.) Note that the range of the sensor is such that it would never produce a number that Siri couldn't handle. If you're ever given the choice between storing +0 or -0, store +0. If you are not able to complete this part, you can still receive full credit on Parts B and C.

Solutions

bias2ones

```
uint8_t bias2ones(uint8_t bias) {
    return (bias >= 0x7F) ? bias - 127 : bias + 128;
}
```

♡ ···

J Jero Wang STAFF 1y #490acb

It's asking you to convert between different number representations. For example, ENCODING TYPE 1 could be sign-magnitude and ENCODING TYPE 2 could be two's complement. For reference, these exams were conducted on PrairieLearn, so the formatting may be a bit weird and hard to understand.

♡ ···

Anonymous Llama 1y #490ab ✓ Resolved

For FA 20 Q1, why is the exponent bin_int log2(left of decimal point) - 1 instead of log2(left of decimal point)? For example, if convert to binary, n is 111.001, wouldn't this be a left logical shift of 2, which is int(log2(7))? instead of log2(7) - 1?

Part A — 3 pts
What is the bit representation (in hex) of the floating-point number `n`? **Do NOT include the `0x` prefix when writing your answer.**

Solutions

1. `bias = - ((1 << (e - 1)) - 1)`
2. `bin_int` $= \log_2(leftofdecimalpoint) - 1 =$ num shifts left for floating point needed to represent whole part of number based on FP equation (remember there's an implicit 1)

♡ ···

Andrew Liu STAFF 1y #490aee

You're right here, nice catch! The proper expression to use should be as you said `int(log_2(n))`, or mathematically

$$\lfloor \log_2 (\lfloor n \rfloor) \rfloor$$

♡ ···

Anonymous Wolverine 1y #490f ✓ Resolved

FA18 MT2, Q1: I'm a little confused by this solution. What does this have to do with counting by 2s? Wondering if I could get a more thorough explanation regarding the logic behind this question.

d) What does **`should_be_a_billion()`** return? *(assume that we always round down to 0)* ___8.0____

This is basically the value when you start counting by 2s, since once you start counting by 2s and always round down, your sum doesn't increase. There are 2 mantissa bits, so there are always 4 numbers in the range [2^i, 2^{i+1}). So you ask yourself, what gap has exactly 4 numbers between consecutive values of [2^i, 2^{i+1}), meaning when are you counting by 1? Easy, [4-8) ⇒ {4, 5, 6, 7}. When you get to 8 you're counting by 2s since you have to cover 8 to 16 with only 4 numbers: {8, 10, 12, 14}. **So it's 8.0** and you didn't have to do any work trying to encode numbers in and out of minifloats, since that was what question c was supposed to be about.

```
minifloat should_be_a_billion() {
      minifloat sum = 0.0;
      for (unsigned int i = 0; i < 1000000000; i++) { sum = sum + 1.0; }
      return(sum);
}
```

♡ ···

Justin Yokota STAFF 1y #490ad

If we keep adding by 1, we eventually reach a point where x was representable, but x+1 wasn't. Since we add 1 each step in the loop, we round to a representable number each step, so once we hit the point where x+1 isn't representable, each step from then on ends up rounding back to x. Thus, we get "stuck" at the point where we can no longer represent x+1. That happens to be exactly the point where we start counting by 2s, since that's the first point we find an integer that can't be represented.

♡ ···

Anonymous Marten 1y #490c ✓ Resolved

Sp20-MT1-Q5C

(c) You arrive at the room, only to find a door locked with a keycode. Spray painted on the wall, you see

*"How many stairwells have a power-of-two number of steps? Print the answer **in hex**..."*

So close to your goal, you crowdsource this question to your favorite social media. Enlisting a friend taking CS 186, you end up with an array of step counts for all stairs which are all positive integers. Create a function to see the total number of stairwells with exactly a power of 2. Hint: you know ***X*** is a power of 2 if and only if ***X*** and ***X-1*** have no bits in common and ***X*** is nonzero. You do not need to use all the lines.

There are multiple valid methods to approach this question. The staff solution requires the least number of lines, and uses the hint that x and x-1 have no bits in common for a power of 2. A power of 2 is found for any non-zero value where the logical and of x and x-1 results in a zero value (thus `stairs[i]` is checked to ensure a non-zero value, then `(stairs[i] & (stairs[i]-1))` is tested for a zero value. Another way to check if the entry is a power of two which is possible with the given lines is to test each bit within the entry and make sure only one bit has a 1 value. Finally, `printf` is called with %x, printing the number of entries in hexadecimal (given by the chart below).

```
void pows_of_2(unsigned int* stairs, int len) {
    int matching_entries = 0;
    _____________________________________________;
    _____________________________________________;
    for (int i = 0; i < len; i++) {
        _____________________________________________;
        if (stairs[i] && (stairs[i] & (stairs[i]-1)) == 0) {
            matching_entries += 1;
        }
        _____________________________________________;
    }
    printf("%x\n", matching_entries );
}
```

Why is it stairs[i] AND stairs[i - 1]? The instructions say that X is a power of 2 iff X and X-1 have no bits in common. So this should mean an XOR right? Since XOR says only 1 or the other, but not both?

♡ ···

E Erik Yang STAFF 1y #490d

the first stairs[i] checks if it's a non-zero value, and then if it is a 1, you can check it with (i - 1) to make sure (i - 1) is not a 1.

♡ ···

Anonymous Marten 1y #490e

I think I still might be misinterpreting the question, so correct me if I'm wrong. When they say X is a power of 2 iff X and X-1 have no bits in common, this should mean that the bit representations of X and X - 1 should **NOT** have any bits in common at the corresponding bit places. For example, if X = 2 and X-1 = 1, then their respective representations are X = 0010 and X-1 = 0001. Then to check that these two have **no** bits in common, we would use the XOR bitwise operator (X ^ (X-1)) and check if this equals to 1.

However, the solution uses **& bitwise** operator, which I still do not understand because that would mean if the bits were both 0 and 0 at the i^{th} bit position in X and X-1, then [(stairs[i] & stairs[i] - 1) == 0] would be true, which contradicts the point of X and X-1 having no bits in common.

♡ ···

E Erik Yang STAFF 1y #490aa

Replying to Anonymous Marten

yeah you're right in that being true, but they check first if stairs[i] is = 1, if not then the logic short circuits and False is returned, if it is a 1, then we know it would only be comparing 1 to 0/1 from stairs[i-1]

♡ ···

Anonymous Camel 1y #490ae

Replying to Erik Yang

would it be correct to instead say stairs[i]^stairs[i-1]? Also, why do we have to check if stairs[i] is zero? The problem statement says that "you end up with an array of step counts for all stairs which are all positive integers"

♡ ···

Anonymous Finch 1y #490a ✓ Resolved

FA17-Final-Q1

Can someone please explain this solution, I'm very confused

2. Please fill in `power_of_16` to calculate whether the given integer is a power of 16.

   Be sure to only use bitwise operators, == and != and up to 1 subtraction. You may introduce constants but not any new variables.

   Return 0 if it is not a power of 16, or 1 if it is. (**Note: 0 is not a power of 16**).

```
// sizeof(int) == 4
int power_of_16(int m) {
      # check sign bit & # of 1-bit in binary representation
      if (0 != ((m >> 31) & 1) || (m & (m-1)) != 0) {
            return 0;
      } else {
            return m != 0 & (m & 0xEEEEEEEE) == 0;
      }
}
```

♡ ...

Andrew Liu **STAFF** 1y #490b

m >> 31& 1 will give you 1 if the top bit is a 1, and a 0 if the top bit is a 0.

This is useful because negative numbers cannot be powers of 16.

Then, (m & (m-1)) gives you 0 if the number of 1's in the binary representation is exactly 1.

This is important because all powers of 16 look like 0x00010000 with a 1 somewhere.

Then, if m is 0, m is not a power of 16, and if it got to the else, we know that it has only one 1 in it, so we AND it with the mask 0xEEE... = 0b11101110..., and if a 1 is somewhere that aligns with this mask, then we have a non power of 16.

♡ ...