[Midterm] Past Exams - 2023 #487

J Jero Wang STAFF
Last year in Exam – Midterm

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 Alpaca 1y #487abce ✓ Resolved

SP23 MT 1.8 - the reference sheet says we used "signed decimal integer" for %d. I thought this meant we use a sign-magnitude representation. how do we know that we have to use 2s complement?

C Format String Specifiers

Specifier	Output
d or i	Signed decimal integer

Q1.8 (2 points)

Solution: −89
0xA7 interpreted as a signed, two's complement, 8-bit integer is −89.

E Erik Yang STAFF 1y #487abda

twos complement is generally the standard to represent signed numbers

♡ 1 ···

Anonymous Alpaca 1y #487abdc

will the exam specify that, or can we ask a TA during the exam for clarification? some of these instructions seem a bit vague, or like requires us to make some assumptions.

♡ ···

E Erik Yang STAFF 1y #487abde

Replying to Anonymous Alpaca

think you can assume twos complement unless specified

♡ ···

J Justin Yokota STAFF 1y #487abdf

Replying to Erik Yang

Unless otherwise specified, we ALWAYS use 2's complement for signed numbers, because it's so much better than any other option.

♡ ...

Anonymous Alpaca 1y #487abbd ✓ Resolved

SP23 MT 2.13 - is this a valid answer: "Not memory efficient because there's extra memory being used by page->data (since new_data has less elements). Must call realloc first on page->data."

Q2.13 (3 points) Is the following implementation of `update_data` correct (follows the described behavior) and memory efficient?

```
1  int MAX_STR_LEN = 100;
2
3  // You may assume that new_data is stored on the heap
4  // and page is well-formed
5  void update_data(Page* page, char* new_data) {
6      if (strlen(new_data) > MAX_STR_LEN) {
7          return;
8      }
9      page->data = new_data;
10 }
```

○ (A) Yes ● (B) No

If you selected "No", provide a **brief** explanation. If you selected "Yes", leave this box blank. We will only grade the first 15 words of your answer.

Solution: Memory leak. We need to `free` the old `data` before reassigning `page->data` to `new_data`.

Note: There were a couple of alternative answers

1. Mentioning that line 6 is incorrect because `strlen` doesn't include the null terminator. There was ambiguity in what "length" meant (Does length include the null terminator or not?)

2. (Out of scope) Mentioning how comparing `size_t` to an `int` could lead to a security vulnerability where the comparison on line 6 would fail, specifically if how `strlen(new_data)` returned a number where, if represented in binary, had a leading 1. This solution is out of scope for this class, and was only deemed correct if the student's answer got every aspect of it correct.

♡ ...

E Erik Yang STAFF 1y #487abcc

line 9 is where you are just simply replacing page->data with the new data so there isn't a need to realloc

♡ ...

Anonymous Alpaca 1y #487abcd

if you're just replacing the data, why does the answer key say that you need to free the old page->data?

the question also asks about memory efficiency, so I thought that page->data might have a different length than new_data, so a realloc would be memory efficient.

Erik Yang STAFF 1y #487abcf

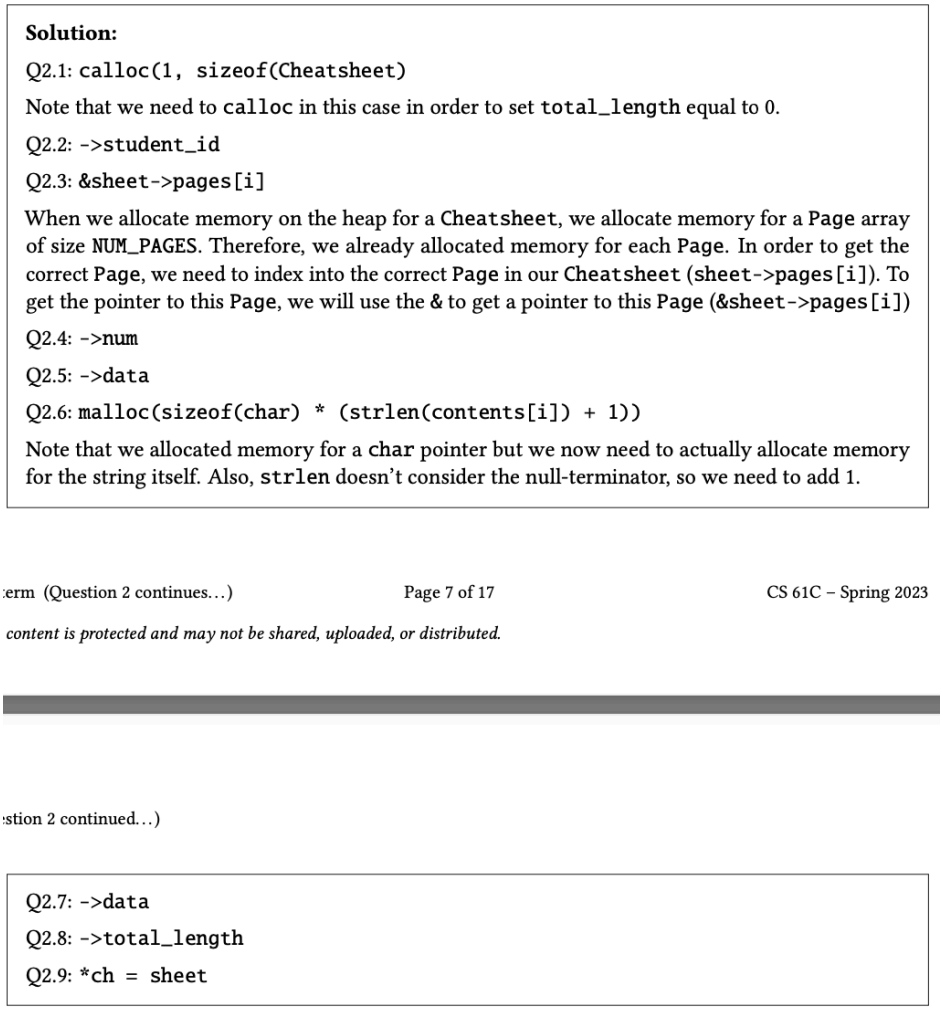
Replying to Anonymous Alpaca

we need to free because we're getting rid of the old_data, which was something that was allocated on the heap and replacing with new_data

Anonymous Alpaca 1y #487abbb ✓ Resolved

SP23 MT 2.3, 2.9

- 2.3 - just wanted to clarify the order of operations for the "*" and "&" operators. is "&sheet->pages[i]" the same as "&(sheet->pages[i]" ?
- 2.9 - could i get an explanation for why "&sheet = ch" would not work? my thought was that i'm setting the memory address of "sheet" to be "ch". how does de-referencing the pointer to the location in memory by doing "*ch" end up saving a pointer to the Cheatsheet sheet at the address ch points to?



Solution:

Q2.1: `calloc(1, sizeof(Cheatsheet)`

Note that we need to `calloc` in this case in order to set `total_length` equal to 0.

Q2.2: `->student_id`

Q2.3: `&sheet->pages[i]`

When we allocate memory on the heap for a `Cheatsheet`, we allocate memory for a `Page` array of size `NUM_PAGES`. Therefore, we already allocated memory for each `Page`. In order to get the correct `Page`, we need to index into the correct `Page` in our `Cheatsheet` (`sheet->pages[i]`). To get the pointer to this `Page`, we will use the `&` to get a pointer to this `Page` (`&sheet->pages[i]`)

Q2.4: `->num`

Q2.5: `->data`

Q2.6: `malloc(sizeof(char) * (strlen(contents[i]) + 1))`

Note that we allocated memory for a `char` pointer but we now need to actually allocate memory for the string itself. Also, `strlen` doesn't consider the null-terminator, so we need to add 1.

stion 2 continued...)

Q2.7: `->data`

Q2.8: `->total_length`

Q2.9: `*ch = sheet`

Erik Yang STAFF 1y #487abbc

first Q: #487aaab

second Q: The problem asks to *save a pointer to that Cheatsheet at the address ch points to*. This means that you need to dereference ch and set it to sheet.

♡ 1 ···

Anonymous Alpaca 1y #487abca

so initially, sheet has some existing random memory address. after you do "*ch = sheet", will that change the value returned by "&sheet" ?

♡ ···

Anonymous Turtle 1y #487abad ✓ Resolved

SP23-MT-Q1.10

just want to clarify that str = 0x6865 6c6c 6f21 2100, and the memory layout is:

address of string: 0x00

+1: 0x21

+2: 0x21

+3: 0x6f,

so the ((int8_t*)str)[1] is actually a+6 instead of a+1, right?

♡ ···

Justin Yokota STAFF 1y #487abae

No; the string is 0x68 0x65 0x6c 0x6c 0x6f 0x21 0x21 0x00, not 0x68656c6c6f212100. The address of string contains the data 0x68, then +1 is 0x65, and so on.

On 1.11, the 32-bit pointer looks for data at +4 - +7, which corresponds to the 0x6F 0x21 0x21 0x00. This, when evaluated as a 32-bit integer in a little-endian system, is 0x0021216F.

♡ 1 ···

Anonymous Snake 1y #487abaa ✓ Resolved

General: When do you use the implicit 1 when doing floating point conversions?

♡ ···

Anonymous Antelope 1y #487abac

I think when you are encountering any normal numbers, any denorms you use the implicit 0 I am pretty sure

♡ ···

Justin Yokota STAFF 1y #487abaf

As above.

♡ ···

Anonymous Vulture 1y #487aafa ✓ Resolved

SP23-MT-Q2.9

Would ch = &sheet also be a valid answer?

♡ ···

E Eddy Byun STAFF 1y #487aafc

#487bff

♡ ···

Anonymous Lyrebird 1y #487aaee ✓ Resolved

SP23-MT-Q6.2

The shortest path between any two timed elements is actually the path from the SEL signal, which changes instantly at the rising edge of the clock, to the right register. This path has only delay 25 ps from the mux.

This was the first time I realized the need to consider non-register elements in the shortest path. That said, is there a more fitting strategy that can be given for examining and finding the shortest path? What other elements like SEL are timed and may have an impact?

♡ ···

E Erik Yang STAFF 1y #487abbf

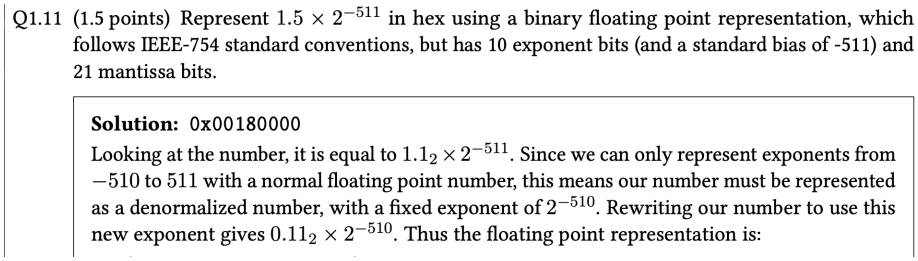
this question is unique in that the SEL signal is actually timed, so you would need to account for that in finding the shortest path

♡ ···

Anonymous Monkey 1y #487aaed ✓ Resolved

SU23-MT-Q1.11

How do we get the range of [-510, 511]? Is there a general formula for getting the range of exponents? Also would the range be different if the standard bias was 511 instead?



Q1.11 (1.5 points) Represent 1.5×2^{-511} in hex using a binary floating point representation, which follows IEEE-754 standard conventions, but has 10 exponent bits (and a standard bias of -511) and 21 mantissa bits.

Solution: 0x00180000

Looking at the number, it is equal to $1.1_2 \times 2^{-511}$. Since we can only represent exponents from -510 to 511 with a normal floating point number, this means our number must be represented as a denormalized number, with a fixed exponent of 2^{-510}. Rewriting our number to use this new exponent gives $0.11_2 \times 2^{-510}$. Thus the floating point representation is:

♡ ···

E Eddy Byun STAFF 1y #487abbe

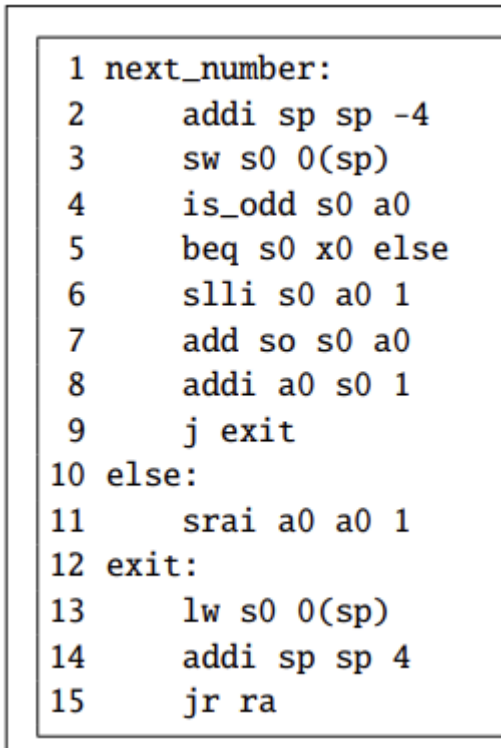
For 10 exponent bits, our normal numbers have exponents in this range: 0000000001 (1) to 1111111110 (1022). When we apply our standard bias of -511, we get a range of [-510,511].

The standard bias for a 10 bit binary number is -511, so I'm a bit confused about your second question. Feel free to follow up!

♡ 1 ···

Anonymous Lion 1y #487aaec ✓ Resolved

SU23-MT-Q4.1



```
1 next_number:
2     addi sp sp -4
3     sw s0 0(sp)
4     is_odd s0 a0
5     beq s0 x0 else
6     slli s0 a0 1
7     add so s0 a0
8     addi a0 s0 1
9     j exit
10 else:
11    srai a0 a0 1
12 exit:
13    lw s0 0(sp)
14    addi sp sp 4
15    jr ra
```

Would it be okay to do

Line 6: add a1, a0, a0

Line 7: add a0, a0, a1

Line 8 addi a0, a0, 1.

Since we don't need to maintain a1 by calling convention, and it is not one of the t registers, is it okay to use it as an intermediate storage register for 2*a0, before completing with 2*a0 + a0 + 1 to get 3*a0 + 1.

♡ ···

J

Justin Yokota STAFF 1y #487abba

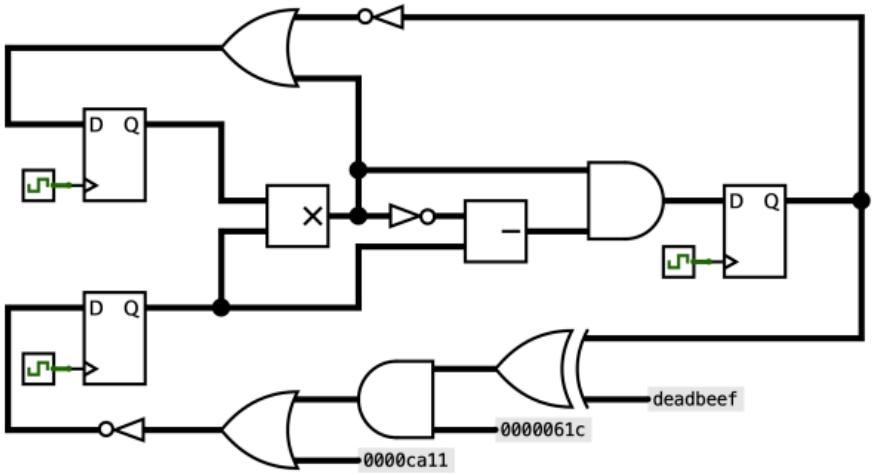
You can use a registers the same way as t registers, so this would technically work. I would hazard a guess that a clarification would have been made that you can't use unused a registers either.

♡ ···

Anonymous Lion 1y #487aaeb ✓ Resolved

SU23-MT-Q6.1

I understand that the top right reg to top left reg has a 25 ps delay path, but since it is asking for the smallest combinatorial delay, shouldn't that mean the shortest time it takes for the input to propagate to the output according to the intended combinatorial behavior? In that case, wouldn't the 50ps delay of the multiplier matter, and cause the shortest delay to be 70 ps?



$t_{\text{AND}} = 20\text{ps}$
$t_{\text{NOT}} = 5\text{ps}$
$t_{\text{OR}} = 20\text{ps}$
$t_{\text{XOR}} = 25\text{ps}$
$t_{\text{multiplier}} = 50\text{ps}$
$t_{\text{subtract}} = 35\text{ps}$
$t_{\text{clk-q}} = 5\text{ps}$
$t_{\text{setup}} = 2\text{ps}$

Q6.1 (2 points) What is the smallest combinational delay of all paths in this circuit, in picoseconds?

Solution: 25ps
The shortest CL path is between the right register and the top left register, consisting of a NOT gate and an OR gate, for a total delay of 25ps.
Grading: All-or-nothing.

♡ 1 ···

D Darwin Zhang STAFF 1y #487abab

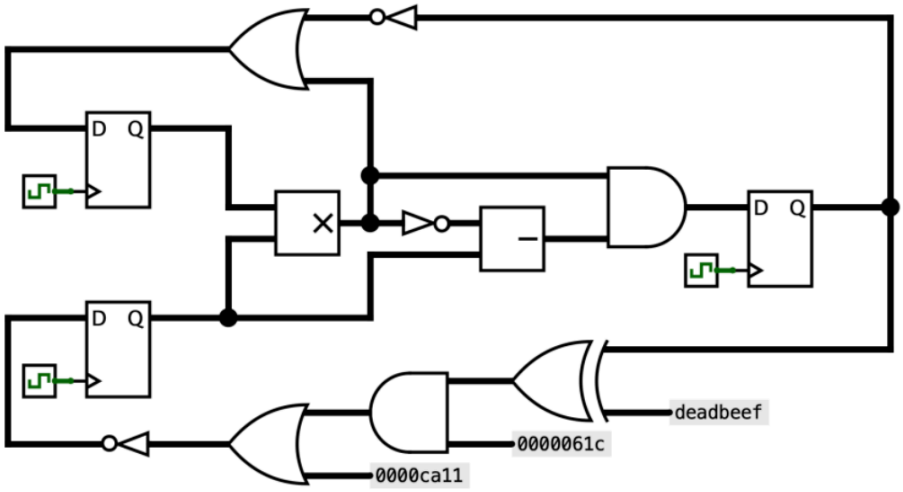
The shortest path is the path that has the smallest delay between two registers. In this case, the path is from the right register to the top left.

♡ ···

Anonymous Lobster 1y #487aadd ✓ Resolved

SUM-MT-Q6.1

Since one of the inputs to the top OR gate is the output of the multiplier, why don't we account for the multiplier time when calculating the length of this path?



$t_{AND} = 20ps$
$t_{NOT} = 5ps$
$t_{OR} = 20ps$
$t_{XOR} = 25ps$
$t_{multiplier} = 50ps$
$t_{subtract} = 35ps$
$t_{clk\text{-}q} = 5ps$
$t_{setup} = 2ps$

Q6.1 (2 points) What is the smallest combinational delay of all paths in this circuit, in picoseconds?

Solution: 25ps
The shortest CL path is between the right register and the top left register, consisting of a NOT gate and an OR gate, for a total delay of 25ps.
Grading: All-or-nothing.

♡ ···

E Ekansh Agrawal STAFF 1y #487aadf

We can assume that a value is already present when we calculate the output of the top left OR gate. Since we are calculating a path to a register to register, we don't need to wait on any gates, we simply calculate the blocks in our direct path.

♡ ···

Anonymous Chicken 1y #487aadb ✓ Resolved

SP23-MT-Q6.1

Q6.1 (3 points) What is the minimum clock period for the circuit above such that it will always result in well-defined behavior?

Solution: 1075 ps
The longest path goes through the multiplier, by far the slowest block in the circuit.
From the rising edge of the clock, we have to wait 30 ps for the signal to show up at the register output. Then, we have to wait 1000 ps for the signal to move through the multiplier, and another 25 ps for the signal to move through the mux. Finally, we have to reach the rightmost register 20 ps early (before the next rising edge) to account for the setup time.
In total, this is $30 + 1000 + 25 + 20 = 1075$ ps.

I am confused why the answer for 6.1 starts at the rising edge of the clock. I thought when it came to finding combinatorial paths, you were always supposed to find the delays between two registers?

♡ ···

Anonymous Kangaroo 1y #487aadc ENDORSED

its asking for the minimum clock period, not just the longest combinatorial path.

the formula for clock period is:

$$t_{clock} \geq t_{CLK-TO-Q} + t_{longest-combinatorial} + t_{setup}$$

so, you include the clock-to-q, which is the time needed to update the register output starting from the rising edge of the clock

the explanation doesn't really separate these out into different components, so that's probably why it was confusing. but yes, if you were just asked to calculate the longest combinatorial path, it would be 1000 + 25 = 1025.

♡ ···

Anonymous Kangaroo 1y #487aada ✓ Resolved

for **sp23-mt-q4.1** can we do:

lw rd imm(rs1)

srli rd rd 24
basically we load a whole word into rd, but then shift right by 24 bits, effectively taking only the upper 8 bits, which is a byte.

the only reason why i think it might not work is that it depends on the endianness of the system. if the system is little-endian, then, the 8 bits at rs1 will be considered the least significant bits when we do lw rd imm(rs1).

this would mean that by shifting right, we would get rid of the 8 bits that we were supposed to look at. (essentially, this alternative would take the 8 most significant bits of the word instead of the 8 least significant bits).

edit: i guess this also wouldn't sign extend, but is my logic about the endianness correct?

♡ 1 ···

M Minyi Liu 1y #487aafe

same question here 😂

♡ ···

S Sam Xu STAFF 1y #487abdd

Good question!

The reason why `lw rd imm(rs1) srli rd rd 24` does not work is `lw` instruction must takes in 4-byte aligned address. However, `lb` and `lbu` insturctions take 1-byte aligned address. If we try to `lbu` an address not 4-byte aligned, such as 0x0000003, we cannot `lw` this address.

♡ ···

Anonymous Gerbil 1y #487aacc ✓ Resolved

SU23-MT-Q4.2

should be add a0 s0 x0 right? not a0 t0 x0

s0 stores the counter & t0 just stores 0 before you jump to loop_end

♡ ···

E Eddy Byun STAFF 1y #487aacd

#487fb

♡1 ···

Anonymous Gerbil 1y #487aace

thank u sm!

♡ ···

Anonymous Octopus 1y #487aabf ✓ Resolved

SU23-MT-Q2.2-3

For this question, if we assume that there was only memory allocated for the Library struct and not for the `users` or `books` array, then that would mean that the address `users` holds is garbage, right (since the memory lib points to hasn't been initialized yet)? If this is true, then `users` points to some random memory location before initialization, that may or may not be available to the user to use. Can we call `realloc` on such a memory location, or will we get an error? In other words, can `realloc` only be used on memory addresses that the programmer already has control over?

If so, is this why this question requires us to assume that `users` also has some unitialized block of memory already created for the programmer to use?

♡ ···

E Eddy Byun STAFF 1y #487aacf

Yea, you're right that in this question we never specified that `users` is `NULL` or has been allocated. As a result, it contains garbage values. If you call `realloc` on garbage values/a pointer that hasn't been allocated yet, you get undefined behavior. We realized this after the midterm, which is why we awarded full credit for Q2.2, Q2.3, and Q2.4 to everyone.

♡ ···

Anonymous Turtle 1y #487abcb

In this case, if we assume that users is NULL, can we use malloc to initialize it?

♡ ···

Anonymous Octopus 1y #487aabe ✓ Resolved

SU23-MT-Q2.6

Hi, for this question, is calloc instead of malloc acceptable, where the second parameter is the same as the parameter for malloc and the first parameter is just 1?

♡ ···

N Noah Yin STAFF 1y #487aaca

Yes, that should be acceptable.

♡ ···

Anonymous Reindeer 1y #487aaae ✓ Resolved

SP23-MT-Q1.2

The explanation for the solution to this question confuses me. Shouldn't an n-bit signed number represent more unique numbers values than an n-bit two's-complement number? I'm asking this

because I believe n-bit two's complement also represents NaN values, which aren't unique.

♡ ···

S Sam Xu STAFF 1y #487aaba

n-bit two's complement does not represent NaN value.
n-bit signed system and n-bit two's complement system both can represent 2^n values. However, in n-bit signed system, `0b000..00` represent 0 and `0b100...00` represent -0, which are the same value. Therefore n-bit signed system represent 1 less unique value than 2's complement

♡ ···

Anonymous Reindeer 1y #487aabb

oh wait, I somehow thought 2's-complement was floating point. sorry. thanks for the clarification!

♡ ···

Anonymous Ibis 1y #487aaab ✓ Resolved

SP23-MT-Q2.3
Would sheet->pages+i also do the same thing as &sheet->pages[i]?

(Question 2 continued...)

(15 points) Fill in `cheatsheet_init` so that it matches the described behavior.

```
1  void cheatsheet_init(Cheatsheet** ch, int student_id, char** contents) {
2      Cheatsheet* sheet = _____Q2.1_____;
3      sheet_____Q2.2_____ = student_id;
4      for (int i = 0; i < NUM_PAGES; i++) {
5          Page* page = _____Q2.3_____;
6          page_____Q2.4_____ = i;
7          page_____Q2.5_____ = _____Q2.6_____;
8          strcpy(page_____Q2.7_____, contents[i]);
9          sheet_____Q2.8_____ += strlen(contents[i]);
10     }
11     _____Q2.9_____;
12 }
```

Solution:

Q2.1: `calloc(1, sizeof(Cheatsheet)`

Note that we need to `calloc` in this case in order to set `total_length` equal to 0.

Q2.2: `->student_id`

Q2.3: `&sheet->pages[i]`

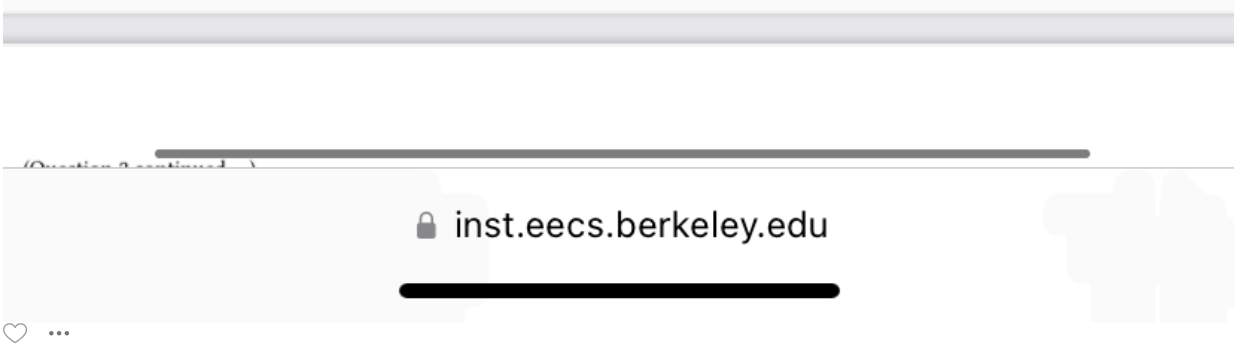
When we allocate memory on the heap for a `Cheatsheet`, we allocate memory for a `Page` array of size `NUM_PAGES`. Therefore, we already allocated memory for each `Page`. In order to get the correct `Page`, we need to index into the correct `Page` in our `Cheatsheet` (`sheet->pages[i]`). To get the pointer to this `Page`, we will use the `&` to get a pointer to this `Page` (`&sheet->pages[i]`)

Q2.4: `->num`

Q2.5: `->data`

Q2.6: `malloc(sizeof(char) * (strlen(contents[i]) + 1))`

Note that we allocated memory for a `char` pointer but we now need to actually allocate memory for the string itself. Also, `strlen` doesn't consider the null-terminator, so we need to add 1



♡ ···

N Noah Yin STAFF 1y #487aacb

Yes, sheet->pages+i would do the same thing as &sheet->pages[i].

In sheet->pages + i, you are first getting the pages pointer from sheet then using pointer arithmetic to increment pages by i * size of Page, giving a pointer to the i-th element.

In &sheet->pages[i], you are using array indexing to access the i-th pages element, then using the & operator to get a pointer to that i-th element.

♡ ···

Anonymous Weasel 1y #487aaef

&(sheet->pages[i]) would also be correct, right?

♡ ···

N Noah Yin STAFF 1y #487aafb

Replying to Anonymous Weasel

Yeah that is equivalent to &sheet->pages[i].

♡ ···

Anonymous Heron 1y #487fea ✓ Resolved

Q4.2 (5 points) Translate the `j loop` instruction under the **skip** label to hexadecimal. Assume that every line in the above code is filled with exactly one instruction (or pseudo-instruction that expands to one instruction).

Solution: `0xFDDFF06F`

SU23-MT-Q4.2

Why would the third bit be D? The 2's complement representation of -36 is

1111111111111011**100**

The third D is essentially the imm[3:1] + imm[1] so shouldn't the binary rep of the third MSB be 0b1001 = 0x9 making the total answer to be 0xFD9FF06F

♡ ···

Andrew Liu STAFF 1y #487fec

Yep, so we have

```
-36 = 1 1111 1111 1111 1101 1100
```

Then, the immediate field should be:

```
imm[20]    = 1             : 1 1111 1111 1111 1101 1100
imm[10:1]  = 111 1101 110  : 1 1111 1111 1111 1101 1100
imm[11]    = 1             : 1 1111 1111 1111 1101 1100
imm[19:12] = 1111 1111     : 1 1111 1111 1111 1101 1100
```

Stringing this together, we have

```
imm[20|10:1|11:19:12] = 1111 1101 1101 1111 1111 (0) <- implicit bc j type
```

Which translates to `0xFDD...`

I think the error you made was that you highlighted `imm[2:0]` and not `imm[3:1]` in your explanation.

♡ ···

Anonymous Duck 1y #487fcd ✓ Resolved

For question 2.6, could you do **sizeof(contents[i])** instead of multiplying by the num of characters and the way the solution did it?

♡ ···

M Mira Bali STAFF 1y #487fda

Could you also specify the exam where this question was asked?

♡ ···

Andrew Liu STAFF 1y #487fef

No, since `content` is of type `char**`, so `contents[i]` is of type `char*`, meaning that trying to get `sizeof(contents[i])` gives you `sizeof(char*) != strlen(contents[i])`. (unless you have a length 4 string on a 32-bit system)

You would be able to do this if you knew the size of the array beforehand (e.g. `sizeof(char*) = 4`, but `sizeof(char[12]) = 12`)

♡ ···

Anonymous Ibis 1y #487fcc ✓ Resolved

SU23-MT-Q2
Why do we not need to malloc borrow_books before passing it into memset?

Solution:

```
1 void init_users(Library* lib, char** user_ids) {
2   int i = 0;
3   while (user_ids[i] != NULL) {
4     lib->users = realloc(lib->users, sizeof(User) * (i + 1));
5     User* cur_user = &lib->users[i];
6     cur_user->user_id = malloc((strlen(user_ids[i]) + 1) * sizeof(char));
7     strcpy(cur_user->user_id, user_ids[i]);
8     memset(cur_user->borrowed_books, 0,
                    MAX_BORROWS * sizeof(Book));
```


(Question 2 continued…)

```
 9     i++;
10   }
11   lib->users_len = i - 1;
12 }
```

Line 11 should have been `i`, not `i - 1`. This was given as a clarification during the exam, and no grading adjustment has been made.

It was ambiguous whether or not that memory was allocated for the `Library` struct pointer's members, such as `users`. Our solution relies on `users` being a pointer returned by `malloc`. As a result, we've awarded full credit for Q2.2, Q2.3, and Q2.4 to everyone.

♡ ···

Andrew Liu STAFF 1y #487ffb

Borrowed books is an array type of `Book*` with length `BORROWED_BOOKS`, so there's space in the struct of size `sizeof(Book*) * BORROWED_BOOKS`, which is allocated on line 4.

♡ ···

Anonymous Ibis 1y #487fff

If I'm understanding correctly, does space for the entire array of Book*'s get allocated when the size of the struct is malloc'ed? While if the type was Book**, we would have to malloc separately to have the same effect?

♡ ···

Anonymous Ibex 1y #487fcb ✓ Resolved

SU23-MT-Q4.1

What is the solution to this question? It doesn't seem to be in the solutions PDF.

Q4 ***RISC-V, RISC-XVI, RISC-VIII*** **(25 points)**

A *wondrous* sequence of positive integers is defined as follows: If n is even, then the next number is $\frac{n}{2}$. Otherwise, the next number is $3n + 1$.

Q4.1 (2 points) We want to create a pseudoinstruction to check whether a number is odd or not. This instruction, written `is_odd rd rs1`, will put the value 1 in `rd` if the value in `rs1` is odd, and the value 0 otherwise. What is the RISC-V instruction that `is_odd rd rs1` would translate to? You may only use one instruction, and you may not use any pseudoinstructions.

Note: Your solution may include `rd` and `rs1`.

♡ 1 ···

M Mira Bali STAFF 1y #487fdb

Oops, I think I attached the answers for 4.2 essentially. I would just check the post that Eddy linked below!

♡ ···

E Eddy Byun STAFF 1y #487fdc

#487bd

♡ ···

Anonymous Manatee 1y #487fbf ✓ Resolved

SU23-MT-Q1.3:

What does it mean by, in bias notation with bias = 2^{2n}, we only need one bit to represent 2^{2n}?

♡ 1 ···

Andrew Liu STAFF 1y #487feb

Suppose we have a 1 bit number with bias 2^{2n}. Then, `0b0` $= 0 + 2^{2n}$, and `0b1` $= 1 + 2^{2n}$.

♡ ···

Anonymous Manatee 1y #487fbc ✓ Resolved

SU23-MT-Q2.11:

Why is it `MAX_BORROWS * sizeof(Book)` and not `MAX_BORROWS * sizeof(Book*)` ? Isn't `Book* borrowed_books[MAX_BORROWS]` an array of pointers?

♡ ···

Anonymous Mole 1y #487fca

+1, I thought that borrowed_books is an array of pointers, and wrote MAX_BORROWS * 4.

♡ ···

M Mira Bali STAFF 1y #487fee

#487aaad

♡ ···

M Mira Bali STAFF 1y #487fed

Borrowed_books is an array of Book objects, so essentially a pointer to Book objects. The Book* specifies that we have a pointer that points to a Book object(s). This is similar to how int* refers to a pointer to an int. So, when you have an int array, to malloc size for it, you would malloc the "# of elements in the array * sizeof(int)"; that's also why we do "MAX_BORROWS * sizeof(Book)" here.

♡ ···

Anonymous Manatee 1y #487ffa

I believe that there is a distinction between `Book* borrowed_books[MAX_BORROWS]` and `Book borrowed_books[MAX_BORROWS]` , just like there is a distinction between `int* arr[5]` and `int arr[5]` . In the int case, the former would be an array of 5 int pointers (i.e. arr[0] would be an int* type and point to an int), while the latter would be an array of 5 ints (i.e. arr[0] would be an int type). I scoured the web and seems like this is the case. If we apply the same logic to the Book case, the former should be an array of pointers to Books while the latter simply an array of Books.

In this case, if we are performing a malloc on int* arr[5], we would have to allocate space for 5 **pointers,** i.e. `malloc(sizeof(int*) * 5)` .For int arr[5], we would allocate space for 5 **ints,** i.e. `malloc(sizeof(int) * 5)` . The same logic applies for Books, hence we are malloc-ing using `sizeof(Book*)` . Is there anything wrong with my logic? Not sure why the answer key says `sizeof(Book)` .

♡ ···

M Mira Bali STAFF 1y #487aaaa

Replying to Anonymous Manatee

Oh I see what you mean, you're right. Since in this case, we do have an array of Book*. Let me look into it and see if this is a memset() thing.

♡ ···

M Mira Bali STAFF 1y #487aaad

Replying to Mira Bali

Ok so I checked and turns out that this was a typo, we should have "MAX_BORROWS * sizeof(Book*)" here. Sorry for the confusion!

♡ ···

Anonymous Manatee 1y #487fbb ✓ Resolved

SU23-MT-Q2.13:

Is saying lib->users[i] is a User struct but not a pointer a valid solution?

♡ ···

Andrew Liu STAFF 1y #487ffc

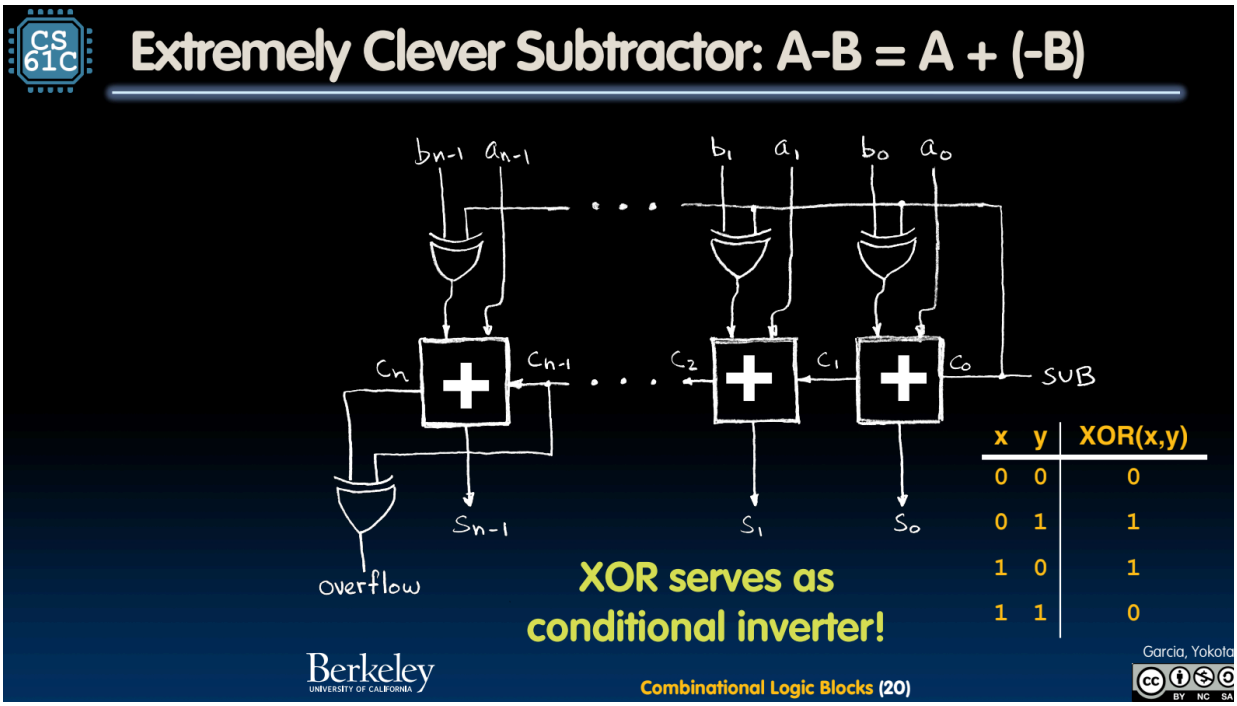
Yes

♡ ···

Anonymous Manatee 1y #487fba ✓ Resolved

SU23-MT-Q6.4:

Lecture 17 states that we can build a subtractor by simply performing a NOT on B and passing a 1 into the input C_0. In essence we only need one adder. Why does the answer key state that we need two?



♡ 1 ···

Anonymous Heron 1y #487fdf

curious as well

♡ ···

Andrew Liu STAFF 1y #487ffd

We assumed adders without carry inputs on the exam.

♡ ···

Anonymous Manatee 1y #487aaac

Can we always make this assumption? Or will it be specified?

♡ ···

Justin Yokota STAFF 1y #487abea

Replying to Anonymous Manatee

This specifically is how you might create a combined adder-subtractor block (which MAY be the adder block we use, but may also not be). In the mentioned exam, we just gave you an adder block; you only know that it takes in two inputs and outputs their sum, and your circuit should be able to work regardless of how the adder works.

In general, if we black-box something, you should be able to make it work regardless of how that black box is implemented.

♡ ···

Anonymous Mandrill 1y #487faf ✓ Resolved

SP23-MT-Q2

When a question says the output "must persist through function calls," what does this mean? Is it implying that we must allocate space?

♡ ···

E Eddy Byun STAFF 1y #487fbd

It implies that we must allocate space on the heap since our stack frame gets deleted when we return from a function

♡ ···

Anonymous Armadillo 1y #487eff ✓ Resolved

SP23-MT-Q2.10

In general, do global variables in the form int x = *some integer* live in the code?

Q2 ***I Can't C My Cheatsheet;*** **(24 points)**

```
1  #define NUM_PAGES 8
2
3  typedef struct Page {
4      int num;
5      char* data;
6  } Page;
7
8  typedef struct Cheatsheet {
9      int student_id;
10     int total_length;
11     Page pages[NUM_PAGES];
12 } Cheatsheet;
```

The question asked where num_pages lives in memory and the answer was code

♡ ···

E Ekansh Agrawal STAFF 1y #487fac

`num_pages` is a macro which means that the compiler basically replaces every invocation of that variable with the value that is defined, in this case 8. The definition lives within the code segment.

♡ ···

Anonymous Armadillo 1y #487fad

ok thanks, what about something like int x = 8? Where would x live if this line was a global variable?

♡ ···

E Ekansh Agrawal STAFF 1y #487fae

Replying to Anonymous Armadillo

It would live in the static memory section (often times this is a shared memory space with the code section).

♡ ···

Anonymous Chicken 1y #487eef ✓ Resolved

Q3.3 (5 points) Consider the floating point number 7.625. What is the largest (closest to $+\infty$) possible value we can represent by modifying a single bit of the floating point representation of this number? Write the binary representation of each component of your answer.

Solution: Sign bit: 0b0

Exponent bits: 0b11001

Mantissa bits: 0b1110100000

$7.625 = 61/8 = 61 \times 2^{-3} = \texttt{0b}111101 \times 2^{-3} = (\texttt{0b}1.11101 \times 2^5) \times 2^{-3} = \texttt{0b}1.11101 \times 2^2$

Sign bit: 0 (positive). We know flipping the sign bit will just make the number negative, which isn't helpful.

Mantissa bits: 0b11101 00000. To increase the number, the most-significant bit we could flip is the 0 to a 1, which would produce $\texttt{0b}1.11111 \times 2^2$. The difference between this number and the original number is $\texttt{0b}0.00010 \times 2^2 = \texttt{0b}0.01 = 1/4$. Flipping any of the less-significant 0s would increase the number by even less.

Exponent bits: $2 - (-15) = 17$, which in unsigned 5-bit binary is 0b10001. We can increase the number by flipping the most-significant 0 to a 1, which would produce 0b11001.

The overall solution is to leave the sign and mantissa bits unchanged, and flipping the most-significant zero bit in the exponent.

Can someone give a bit more detail on how they converted 7.625 into floating point. The exam's solution makes no sense to for me.

♡ ···

E Eddy Byun STAFF 1y #487efe

Have you taken a look at this: #487cca?

♡ ···

Anonymous Stork 1y #487eeb ✓ Resolved

SP23-MT-Q6.4

If we treat the substractor as a black box, then a - b = a + (flip b) + 1. Can we change it to (a+1) + (flip b)? Now the longest path only has two adders, so the maximum delay of an adder can be 17.5ps instead of 15 ps.

♡ ···

E Eddy Byun STAFF 1y #487fbe

Yea, 17.5 was also an acceptable solution for this problem.

♡ ···

Anonymous Gaur 1y #487edc ✓ Resolved

sp23-mt-q1.6

converting jal s3 588

My work:

opcode: 1101111

rd: 11001

label = imm = 588 = 1001001100

I left padded the label with zeros until it was 20 bits.

final entire binary:
imm[20|10:1|11|19:12] rd opcode

0 1001001100 0 00000000 11001 1101111

0x49800CEF

Correct Answer: 0x24C009EF

Where am I going wrong?

♡ ···

E Eddy Byun STAFF 1y #487efa

#487aae

♡ ···

Anonymous Badger 1y #487ecf ✓ Resolved

Solution:

```
 1 num_steps:
       # Prologue
       # Omitted
 2     addi s0 x0 0
 3 loop_start:
 4     addi t0 x0 1
 5     beq a0 t0 loop_end
 6     jal ra next_number
 7     addi s0 s0 1
 8     j loop_start
 9 loop_end:
10     add a0 t0 x0
       # Epilogue
       # Omitted
11     jr ra
```

Grading: Credit was given for all equivalent answers, with points deducted for using `s` registers or breaking calling convention.

for the loop_end part, shouldn't it be add a0 s0 x0 instead of t0 since s0 is the counter for number of steps taken?

♡ ···

E Eddy Byun STAFF 1y #487ede

Yea, this is a typo in the solution. It should be `add a0 s0 x0` or an equivalent instruction

♡ ···

Anonymous Newt 1y #487ece ✓ Resolved

There is an error in the solutions for the alternative answer on SP23-MT-Q2.13.

The given alternative solution is:

> (Out of scope) Mentioning how comparing size_t to an int could lead to a security vulnerability where the comparison on line 6 would fail, specifically if how strlen(new_data) returned a number where, if represented in binary, had a leading

> 1. This solution is out of scope for this class, and was only deemed correct if the student's answer got every aspect of it correct

But this justification is incorrect. There is no situation where the code could lead to a security bug because signed integers are converted to unsigned integers before comparison (assuming `sizeof(size_t) >= sizeof(int)`, and `MAX_STR_LEN` is non-negative.

To be precise, here is the relevant portion of the standard:

> [When] both operands [to a relational operator] are integers, both operands undergo *integer promotions* (see below); then, after integer promotion, one of the following cases applies:

So there are three cases:

1. If `size_t` is smaller than `int`, it is promoted to an `int`. Then we have an integer vs integer comparison, which is logically correct.
2. If `size_t` is equal to an int, then this bullet applies, and `int` is converted to `unsigned int`. So `MAX_STR_LEN` is promoted to `unsigned int`. Since `MAX_STR_LEN` is non-negative, the comparison is again logically correct.
3. If `size_t` is greater than an int, then `MAX_STR_LEN` is promoted to `size_t`, and again the comparison is logically correct since `MAX_STR_LEN` is non-negative.

♡ ···

Anonymous Swan 1y #487ecc ✓ Resolved

SP23-MT-Q1.10:

Since this is little endian i found the corresponding addresses of the word hello!! to make it !!olleh. I thought the str[1] is the first element and would return the address of o. Why does the answer key return the address of the letter e?

♡ 2 ···

Anonymous Duck 1y #487faa

i'm stuck on this as well, especially since the next part acts treats it differently with it having **/n!!o** as the 0 element. why is that? I would appreciate a detailed response, thanks!

♡ ···

J Justin Yokota STAFF 1y #487fdd

Arrays don't change the order of their elements; only the order of the bytes within each element get reversed. Since each char is a single byte, that doesn't affect the order of the bytes in the string.

♡ ···

Anonymous Manatee 1y #487ecb ✓ Resolved

SP23-MT-Q1.3

Does gcc only act as the compiler or is it the compiler, assembler, and linker all in one?

♡ ···

C Catherine Van Keuren STAFF 1y #487eda

It's the compiler, assembler and linker.

♡ ···

Anonymous Manatee 1y #487eca ✓ Resolved

SP23-MT-Q2.1:

Can we always use calloc in place of malloc?

♡ ···

E Eddy Byun STAFF 1y #487edd

Yes, the only difference between `calloc` and `malloc` is that `calloc` is going to zero-out the memory that you allocated.

For this question, `calloc` was the only answer that got full credit.

♡ ···

Anonymous Antelope 1y #487ebe ✓ Resolved

SP23-MT2-Q2.10-2.12

Hi, I am pretty confused why sheet would be on the stack instead of the heap since it is a pointer? I thought pointers were supposed to be on the heap? And also *sheet seems like it would point inside a struct which I thought would be on the stack?

Q2.10 (2 points) `NUM_PAGES`

○ (A) Stack ○ (B) Heap ● (C) Code ○ (D) Data/Static

Q2.11 (2 points) `sheet`

● (A) Stack ○ (B) Heap ○ (C) Code ○ (D) Data/Static

Q2.12 (2 points) `*sheet`

○ (A) Stack ● (B) Heap ○ (C) Code ○ (D) Data/Static

♡ ···

E Eddy Byun STAFF 1y #487eea

`calloc` and `malloc` will allocate memory in the heap and then return a pointer to the allocated memory on the heap. `sheet` itself is a pointer to the `Cheatsheet` that we allocated, and it's a local variable on the stack. Dereferencing `sheet` (`*sheet`) is going to be the `Cheatsheet` that we allocated on the heap.

♡ 1 ···

Anonymous Antelope 1y #487aabd

thank you :)

♡ ···

Anonymous Sardine 1y #487ebc ✓ Resolved

SP23-MT2-Q2.1 - Q2.4

For question 2.1, I am wondering why we made a whole new struct pointer, if we are already given ch?

My initial thought was to do Cheatsheet* sheet = &ch

For Q2.4, I am wondering why we didn't put parenthesis around the values and the ampersand outside of it: &(sheet->pages[i])?

> Q2.1: `calloc(1, sizeof(Cheatsheet)`
>
> Note that we need to `calloc` in this case in order to set `total_length` equal to 0.
>
> Q2.2: `->student_id`
>
> Q2.3: `&sheet->pages[i]`

♡ ···

E Eddy Byun STAFF 1y #487eec

For 2.1: "It should create a well-formed Cheatsheet with the following properties, and save a pointer to that Cheatsheet at the address ch points to" - we need to allocate memory for a `Cheatsheet` and store this pointer at the address `ch` points to.

`&ch` returns the address of `ch`, which would be of type `Cheatsheet***`

2.4: https://en.cppreference.com/w/c/language/operator_precedence - the order of operands state that we first do `->` and `[]` and comes the `&` operand

♡ 1 ···

Anonymous Sardine 1y #487eed

Thank you. I was not aware about the order of operands either.

♡ ···

E Eddy Byun STAFF 1y #487eee

Replying to Anonymous Sardine

Yea, you can also do `&(sheet->pages[i])`

♡ ···

Anonymous Spoonbill 1y #487eaf ✓ Resolved

SP23-MT-Q1.11

does str[1] contain 'o!!\0' because str[0] is going to contain the fist 4 letters ('hell')?

♡ ···

J Justin Yokota STAFF 1y #487ebf

Yes, since str is being interpreted on that line as an int*

♡ ···

Anonymous Octopus 1y #487eac ✓ Resolved

Hi,

For **SP23-MT1-Q4.3**, RISC-V labels count as lines? For example, does `temp_label:` count as its own line, and do we have to account for it when incrementing the PC?

♡ ···

E Eddy Byun STAFF 1y #487eba

#488cec

♡ ···

Anonymous Reindeer 1y #487ddb ✓ Resolved

Sp23-MT-Q1.11

Shouldn't the answer be 0x0021216F instead of 0x00212165 since 'o' is 0x6F in ASCII and not 0x65?

♡ 1 ···

Jero Wang STAFF 1y #487dea

Yes, that's a typo, sorry.

♡ 1 ···

Anonymous Butterfly 1y #487ecd

Could you please explain why we skip 'l', 'l', and 'o' and go straight to '!'?

♡ ···

Anonymous Chamois 1y #487dda ✓ Resolved

su22-MT2-q4

```
add_even_numbers:
    addi t0, x0, 0      # set t0 to be the running sum
loop:
    beq a1 x0 end
    lw t1 0(a0)         # set t1 to be the number in the array
    andi t2 t1 1
    beq t2 x0 pass
    add t0 t0 t1
pass:
    addi a0 a0 4
    addi a1 a1 -1
    j loop
end:
    ret
```

How come for the line add t0 t0 t1, t0 isn't garbage once j loop is called? We never stored it to the stack.

♡ ···

Andrew Liu STAFF 1y #487ddd

`loop` is a label to help organize our function, not a function call. (note the lack of a `al` as well)

♡ ···

Anonymous Chamois 1y #487dcb ✓ Resolved

Q1.6 (3 points) Which program initializes registers to their default value?

- ○ (A) Assembler
- ○ (B) Compiler
- ○ (C) Interpreter
- ○ (D) Linker
- ● (E) None of the above

Solution: None of these choices actually start and run the program, so they don't initialize registers.

su22-mt-q1.6, i was wondering what choice would actually start and run the program?

♡ ···

S Sam Xu STAFF 1y #487dcd

loader starts and runs the program

♡ ···

Anonymous Penguin 1y #487cfd ✓ Resolved

su23-mt-q2.13

Would this be a valid justification for "No"?

The user_id string's memory for each user was malloc-ed but not deallocated before this user's memory was freed, leading to memory leak.

♡ 1 ···

E Eddy Byun STAFF 1y #487ddf

Yes, this was another error in the function, and we accepted this answer as well.

♡ 1 ···

Anonymous Armadillo 1y #487cfa ✓ Resolved

conceptual question that I see on pretty much every test: whats the real difference between static and code that differentiates them? I was told in riscv, something is code if theyre an immediate but I am still confused about their true distinction

♡ ···

J Jero Wang STAFF 1y #487dec

In most cases, code is executable, and static is not executable. Code generally contains the program itself, while static contains any data the program may need to use during its lifecycle.

In RISC-V, the immediate is code because it's literally embedded in the instruction itself. Take `addi a0 x0 1`, it gets written to the executable as `0x00100513`, and the immediate is within the code. However, if you have something like a string literal (you can't really put a string in an instruction), you need to store the string literal somewhere else (like the data segment).

♡ ···

Anonymous Armadillo 1y #487ead

thanks!

♡ ···

Anonymous Armadillo 1y #487eae

Does executable mean the same thing as read-write, and not executable = read only?

♡ ···

Anonymous Cheetah 1y #487cef ✓ Resolved

SP23-MT2-Q4.1

Why use s0? From my perspective other registers like t0 could also be the counterpart. Is it only because s0 was shown up in the context?

♡ ···

E Eddy Byun STAFF 1y #487dca

Is this from the SU midterm? You need to use `s0` because it was shown up in the context.

♡ ···

Anonymous Cod 1y #487ced ✓ Resolved

SP23-MT-Q1.7

How was this simplified?

Q1.7 (3 points) Write a Boolean expression that evaluates to the truth table below. You may use at most 2 Boolean operations. ~ (NOT), | (OR), & (AND) each count as one operation. We will assume standard C operator precedence, so use parentheses when uncertain.

W	Y	Z	Out
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Solution: ~Y | Z
Other solutions may exist.

♡ ···

E Eddy Byun STAFF 1y #487dbd

!W*!Y*!Z + !W*!Y*Z + !W*Y*Z + W*!Y* !Z + W*!Y*Z + W*Y*Z =

!W!Y * (!Z + Z) + YZ * (!W + W) + W!Y (!Z + Z)

= !W!Y + YZ + W!Y

= !Y (!W +W) + YZ

= !Y + YZ

!Y = !Y + !YZ using the absorption law; we can plug this into !Y

= !Y + !YZ + YZ

= !Y + Z*(!Y + Y)

= !Y + Z

♡ 1 ···

Anonymous Chamois 1y #487cde ✓ Resolved

SU23-MT-Q2.10

Can we say memset(cur_user->borrowed_books, NULL (instead of 0), MAX_BORROWS * sizeof(Book));

♡ ···

E Eddy Byun STAFF 1y #487dbc

Yea, NULL is fine.

♡ ···

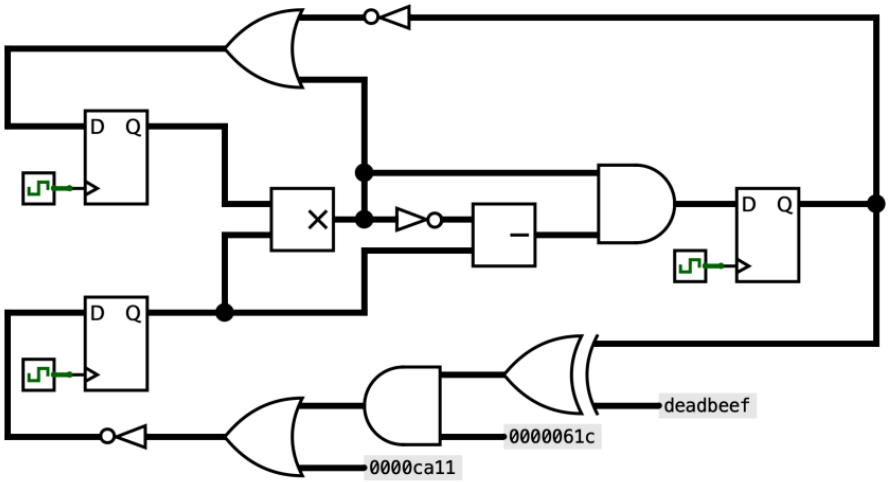
Anonymous Crab 1y #487cdc ✓ Resolved

SU23-MT-Q6

in this question part 6.3, why couldn't it be 25 ps? because if we consider the path from input 0000ca11 to D of the register on left it takes 25ps and since input doesn't have clk-to-q so the hole should be less than or equal to 25.

Q6 *SDS* (8 points)

Consider the following circuit diagram and component delays:



$t_{\text{AND}} = 20\text{ps}$
$t_{\text{NOT}} = 5\text{ps}$
$t_{\text{OR}} = 20\text{ps}$
$t_{\text{XOR}} = 25\text{ps}$
$t_{\text{multiplier}} = 50\text{ps}$
$t_{\text{subtract}} = 35\text{ps}$
$t_{\text{clk-q}} = 5\text{ps}$
$t_{\text{setup}} = 2\text{ps}$

Q6.1 (2 points) What is the smallest combinational delay of all paths in this circuit, in picoseconds?

Solution: 25ps
The shortest CL path is between the right register and the top left register, consisting of a NOT gate and an OR gate, for a total delay of 25ps.
Grading: All-or-nothing.

Q6.2 (2 points) What is the minimum allowable clock period for this circuit to function properly, in picoseconds?

Solution: 117ps
The longest path between any two registers is between the top left register and the register, consisting of a multiplier, a NOT gate, a subtractor, and an AND gate, for a total of 110ps. Additionally, we need to account for clk-to-q and setup, which gives us 117ps.
Grading: All-or-nothing.

Q6.3 (2 points) What is the maximum hold time the registers can have so that there are no hold time violations in the circuit above?

Solution: 30ps
The shortest CL path is 25ps (see Q6.1), and the maximum hold time is the shortest CL path + clk-to-q, which gives us 30ps.
Grading: All-or-nothing, except full credit was given for Q6.1 + 5 to avoid double jeopardy

♡ ···

Anonymous Sardine 1y #487cee

You have to look at is as a register to register value when it comes to CL max and min.

♡ ···

E Eddy Byun STAFF 1y #487dbb

`0000ca11` is a constant; it is not an input. The other input that is above `0000ca11` only gets updated once the AND gate to the right gets changed. The input to the AND gates are also a constant `0x0000061C` and the output of an XOR gate. The output of the XOR gate only gets changed after clk-to-q. The sum of all these delays is >30.

♡ ···

Anonymous Heron 1y #487cdb ✓ Resolved

SP23-MT-Q2.10-12

Why would sheet be on the stack and *sheet be on the heap? Shouldn't their data types align if *sheet is a pointer to sheet? Shouldn't they both be on the heap?

♡ 1 ···

Anonymous Badger 1y #487cdf

was also confused abt this, the second answer in this article helped for me: https://stackoverflow.com/questions/14588767/where-in-memory-are-my-variables-stored-in-c (the one by hagrawal7777)

♡ ···

Anonymous Heron 1y #487cfc

Thank you! This is really helpful

♡ ···

Anonymous Chamois 1y #487ccf ✓ Resolved

SPRING23-MT2-Q2

- for Q2.1 can we write malloc(sizeof(cheatsheet)) as calloc(1, sizeof(Cheatsheet) would basically malloc space for 1 * sizeof(cheatsheet)
- Q2.3: Just to clarify would it be &(sheet->pages[i]), so first we would find the relevant pages[i] and then take the address of it
- Q2.11 and 12: I get that *sheet would be a pointer and you need to malloc space for it which is what we did for Q2.1 but what is sheet?

♡ ···

E Eddy Byun STAFF 1y #487dac

1. `calloc` is the only solution that would get full points because `calloc` 0's out the allocated memory, and we need to initialize `total_length` to be 0

2. Yes

3. `sheet` is a local variable that contains the pointer to the allocated `Cheatsheet` on the heap. While `*sheet` is on the heap, `sheet` itself will live in the stack.

♡ ···

Anonymous Chamois 1y #487dba

does calloc 0's out all the parameters of Cheatsheet. so once we calloc, student_id, total_length would be 0 and Page pages[NUM_PAGES] would have the appropriate memory space malloced?

♡ ···

E Eddy Byun STAFF 1y #487dee

Replying to Anonymous Chamois

Page pages[NUM_PAGES] would be allocated on the heap as well and will also be zero'd out. Both `calloc` and `malloc` allocate memory on the heap. `calloc` will initialize all the values that you allocate to 0 while `malloc` will not initialize the values to anything. As a result, when you call `malloc` , you can have garbage values.

♡ ···

Anonymous Pony 1y #487edb

Replying to Eddy Byun

Why do we need to initialize total_length to be 0?

♡ ...

Anonymous Heron 1y #487aade

Replying to Eddy Byun

Why would the last line be *ch = sheet if sheet is on the stack so the information would disappear after the function call ends?

Since the stack memory of a function gets deallocated after the function returns, there is no guarantee that the value stored in those area will stay the same. **A common mistake is to return a pointer to a stack variable in a helper function**. After the caller gets this pointer, the invalid stack memory can be overwritten at anytime. The following figures demonstrate one example of such scenario. Assume there is a `Cube` class that has methods `getVolume` and `getSurfaceArea`, as well as a private variable `width`.

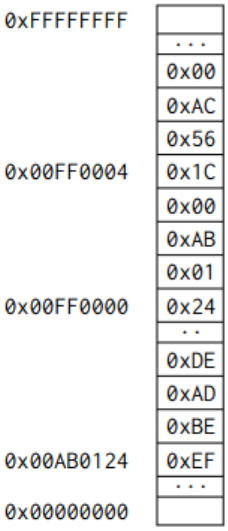
♡ 1 ...

Anonymous Sardine 1y #487cce ✓ Resolved

SU23-MT2-Q1.7

I am confused as to how we got 2 for this question and 0 for 1.8 the other because it seems to contradict this solution from discussion 4:

```
1  li x5 0x00FF0000
2  lw x6 0(x5)
3  addi x5 x5 4
4  lhu x7 1(x5)
5  lh x8 1(x5)
6  lb x9 3(x6)
7  sh x8 2(x5)
```



- Line 1: `x5` will hold `0x00FF0000`
- Line 2: `x6` will hold 0x00AB0124, the word at the address `0x00FF0000` + 0

It's little endian or even:

- Line 4: `x7` will hold `0x0000AC56`. `0xAC56` is the 2 bytes of data stored starting at address `0x00FF0004` + 1. Because the instruction is lhu, `x7` will hold `0xAC56` zero-extended. Recall, registers store 32 bits

This is picking locations starting from the MSB side instead, so applying that same logic to Q1.7 And Q1.8 we wouldn't get 2 and 0 respectively, we'd get 0(x) being 00 and thus 0 for 1.7, and big endian would be 2 because 0(x) = 01 and 1(x) = 01 and then finally 2(x) = 00

♡ ...

E Eddy Byun STAFF 1y #487dae

Sorry, what do you mean by "picking locations starting from the MSB side"?

When we store 257 (0x00000101) in a little endian system, this is the memory layout for a little endian system:

Memory address a: 0x01

Memory address a + 1: 0x01

Memory address a + 2: 0x00

Memory address a + 3: 0x00

`strlen((char *) &x);` will return 2 in this case.

For a big endian system, this is the memory layout:

Memory address a: 0x00

Memory address a + 1: 0x00

Memory address a + 2: 0x01

Memory address a + 3: 0x01

`strlen((char *) &x);` will return 0 in this case.

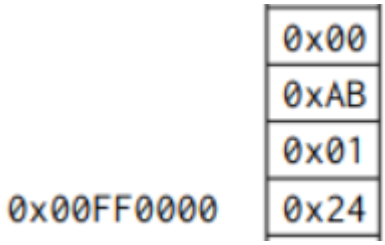
♡ ···

Anonymous Sardine 1y #487daf

From discussion when we loaded line 2:

```
2   lw x6 0(x5)
```

We loaded from the highest point in memory, down:



- Line 2: `x6` will hold 0x00AB0124, the word at the address `0x00FF0000` + 0

Does this mean I am reading the discussion sheet wrong?
Because applying the same logic from this sheet to this problem would leave me to believe that we should be reading 0x00 first and not 0x01in little endian and the opposite in big endian

♡ ···

E Eddy Byun STAFF 1y #487dbe

Replying to Anonymous Sardine

The memory diagram in the discussion goes from smallest memory address at the bottom to largest memory address at the top.

♡ ···

Anonymous Sardine 1y #487dbf

Replying to Eddy Byun

That's where my confusion is.

Because for the problem, the 01's are stored at the lowest memory address; YET, in discussion we do not pull from the lowest memory address, we pull from the highest, so if I apply the same logic from discussion, to this question, I would pull 00 from the top because they are in the highest memory address not the 01's

If we pulled from lowest memory address in discussion first, we'd have the same answer but flipped.

♡ ···

E Eddy Byun STAFF 1y #487dcc

Replying to Anonymous Sardine

We load a word from 0x00FF0000 since `x5` is equal to 0x00FF0000. A word is 4 bytes, so we get

0x00FF0000: 0x24

0x00FF0001: 0x01

0x00FF0002: 0xAB

0x00FF0003: 0x00

Since we're working with a little endian system, the least significant byte is stored at the smallest memory address. Therefore, `x6` is going to contain `0x00AB0124`

♡ ···

Anonymous Sardine 1y #487dce

Replying to Eddy Byun

Ah, when viewed that way it makes sense, thank you.

♡ ···

Anonymous Crab 1y #487ccc ✓ Resolved

SU23-MT-Q3

When a question says write it in a floating point value, does that mean to write it like 1.1 or 1.5?

Like in here I though because question said return the result as a floating point value, we should say it's 1.1? i though 1.5 is the decimal representation of 1.1.

Q3.3 (2 points) `mystery(∞)`

Solution: 1.5
Positive infinity has representation
`0b0 11111111 00000000000000000000000`
and thus is
`0b0 01111111 10000000000000000000000`
after the shift. This is a positive number with an exponent of 127 pre-bias (0 post-bias).
$1.1_2 \times 2^0 = 1.5$
Grading: All-or-nothing.

♡ ···

E Eddy Byun STAFF 1y #487dab

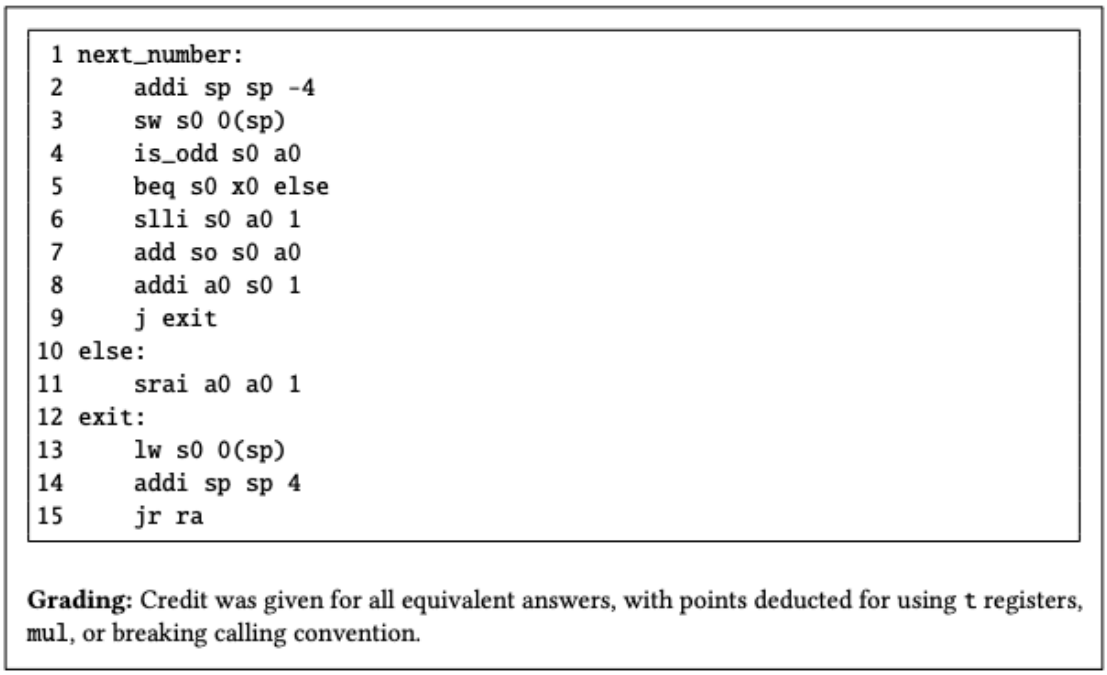
We'll try to explicitly state the form that we want you to express your answer as. I think the intent was to have this written in decimal form

♡ ...

Anonymous Armadillo 1y #487cbb ✓ Resolved

SU23-MT-Q4

Why do we not save ra to the sp?



```
1 next_number:
2     addi sp sp -4
3     sw s0 0(sp)
4     is_odd s0 a0
5     beq s0 x0 else
6     slli s0 a0 1
7     add so s0 a0
8     addi a0 s0 1
9     j exit
10 else:
11    srai a0 a0 1
12 exit:
13    lw s0 0(sp)
14    addi sp sp 4
15    jr ra
```

Grading: Credit was given for all equivalent answers, with points deducted for using `t` registers, `mul`, or breaking calling convention.

♡ ...

E Eddy Byun STAFF 1y #487daa

We never overwrite the `ra` register, so we don't need to save `ra` onto the stack

♡ ...

Anonymous Wolf 1y #487caa ✓ Resolved

SP23-MT-Q2

Can we also `calloc(1, sizeof(struct Cheatsheet))` instead of `calloc(1, sizeof(Cheatsheet))`?

♡ 1 ...

Andrew Liu STAFF 1y #487ddc

Yes, `Cheatsheet` is `typedef`'d as `struct Cheatsheet`.

♡ ...

Anonymous Wolf 1y #487bff ✓ Resolved

SP23-MT-Q2

Could `ch = &sheet` work instead of `*ch = sheet`?

♡ ...

Andrew Liu STAFF 1y #487cba

No, since C is pass by value, what `ch = &sheet` does is change your local copy of `ch` without affecting the value passed in by the parent.

♡ ...

Anonymous Ferret 1y #487edf

Hi, if you were to do *ch = sheet then ran print(ch) would this print out &sheet?

♡ ···

Anonymous Crab 1y #487bfc ✓ Resolved

SP23-MT-Q3

in this question, how did they found 61?

Q3.3 (5 points) Consider the floating point number 7.625. What is the largest (closest to $+\infty$) possible value we can represent by modifying a single bit of the floating point representation of this number? Write the binary representation of each component of your answer.

Solution: Sign bit: 0b0

Exponent bits: 0b11001

Mantissa bits: 0b1110100000

$7.625 = 61/8 = 61 \times 2^{-3} = \texttt{0b}111101 \times 2^{-3} = (\texttt{0b}1.11101 \times 2^5) \times 2^{-3} = \texttt{0b}1.11101 \times 2^2$

Sign bit: 0 (positive). We know flipping the sign bit will just make the number negative, which isn't helpful.

Mantissa bits: 0b11101 00000. To increase the number, the most-significant bit we could flip is the 0 to a 1, which would produce $\texttt{0b}1.11111 \times 2^2$. The difference between this number and the original number is $\texttt{0b}0.00010 \times 2^2 = \texttt{0b}0.01 = 1/4$. Flipping any of the less-significant 0s would increase the number by even less.

Exponent bits: $2 - (-15) = 17$, which in unsigned 5-bit binary is 0b10001. We can increase the number by flipping the most-significant 0 to a 1, which would produce 0b11001.

The overall solution is to leave the sign and mantissa bits unchanged, and flipping the most-significant zero bit in the exponent.

♡ ···

E Eddy Byun STAFF 1y #487cca

I think a better way to do the conversion is to do the following:

7 -> 111_2

0.625 -> $.101_2$

-> 7.625 = 111.101_2 = $1.11101 * 2^2$

♡3 ···

Anonymous Grouse 1y #487bef ✓ Resolved

SU22-MT-Q4.1

For line 11, what's the difference between srai and srli when dividing? Why does it matter that we sign extend if all the numbers are going to be positive, so would srli work?

```
 1 next_number:
 2     addi sp sp -4
 3     sw s0 0(sp)
 4     is_odd s0 a0
 5     beq s0 x0 else
 6     slli s0 a0 1
 7     add so s0 a0
 8     addi a0 s0 1
 9     j exit
10 else:
11     srai a0 a0 1
12 exit:
13     lw s0 0(sp)
14     addi sp sp 4
15     jr ra
```

Grading: Credit was given for all equivalent answers, with points deducted for using `t` registers, `mul`, or breaking calling convention.

♡ 1 ···

Andrew Liu STAFF 1y #487cae

Having either `srai` or `srli` were considered as equivalent solutions for this question.

♡ 1 ···

Anonymous Ram 1y #487bee ✓ Resolved

SU23-MT-Q4.16

I am unsure as to why the answer to the question is add a0 t0 x0, wouldn't that mean a0 would always be 1 no matter how many steps were taken? I thought that s0 was holding the number of steps and therefore we were supposed to move the value of s0 to a0.

♡ 2 ···

Andrew Liu STAFF 1y #487caf

You're correct, there's a typo in the solutions, that line should be `add a0 s0 x0` or equivalent.

♡ ···

Anonymous Swan 1y #487bed ✓ Resolved

SU23-MT-Q1.11

I don't understand how the exponent value was found. I thought there is a bias of -511 this would be deormanlized so the exponent would become 2^(-511+511+1) -> 2^1. Hence, I don;t get why the 10 bits of the exponents are all 0s.

(1.5 points) Represent 1.5×2^{-511} in hex using a binary floating point representation, which follows IEEE-754 standard conventions, but has 10 exponent bits (and a standard bias of -511) and 21 mantissa bits.

Solution: 0x00180000

Looking at the number, it is equal to $1.1_2 \times 2^{-511}$. Since we can only represent exponents from -510 to 511 with a normal floating point number, this means our number must be represented as a denormalized number, with a fixed exponent of 2^{-510}. Rewriting our number to use this new exponent gives $0.11_2 \times 2^{-510}$. Thus the floating point representation is:

```
  sign  exponent   mantissa
  0     0000000000 11000000000000000000000
  0000 0000 0001 1000 0000 0000 0000 0000
0x0    0    1    8    0    0    0    0
```

♡ ···

Anonymous Sardine 1y #487ccd

$$1.5_{10} \cdot 2^{-511} == 1.1_2 \cdot 2.^{-511}$$

$$denormalized = 1^{(sign)} \cdot 2^{bias+1} \cdot 0.significand$$

$$0.11 \cdot 2 == 1.1 => 1.1 \cdot 2^{-511} == 0.11 \cdot 2^{-510}$$

$$2^{-510} == 2^{-511+1}$$

♡ ···

Andrew Liu STAFF 1y #487dde

The exponent for a denormalized number is going to be `bias + 1`, and a denormalized number is represented by a fully-zero exponent.

♡ ···

Anonymous Eland 1y #487beb ✓ Resolved

SP23-MT-Q6

This is conceptual but for finding the shortest and longest combinational block, does this basically mean the shortest and longest time between two timed elements (i.e. clocks)?

♡ ···

N Nikhil Kandkur STAFF 1y #487bfa

Yup!

♡ ···

Anonymous Monkey 1y #487bcc ✓ Resolved

SP23-MT2-Q3.4

Given our binary floating point representation, with 5 exponent bits (and a standard bias of -15) and 10 mantissa bits, how do we calculate to get that the total number of floating-point numbers is 2^16?

♡ ···

E Eddy Byun STAFF 1y #487bdc

We have 16 total bits and each bit can be either a 1 or a 0. Thus, the total number of floating point numbers would be 2*2*2....*2 (16 multiplications) which is 2^{16}.

Anonymous Chamois 1y #487bca ✓ Resolved

SUM-MT-Q6.4

I understand the logic behind that A - B = A + (~B+1) but I'm still not sure how that corresponds to 35. Is it because A would be the output from the multiplier and while that is running you can compute (~B+1) which would take 5 + 20 (25 ps) then when adding them together it would take 20 (given tAND is 20), so we could still hold it for 35 without altering the behavior since we want maximum delaY?

Andrew Liu STAFF 1y #487cbc

The idea is the the first adder and not gate will run in parallel with the multiplier, and since the delay of the first adder and not gate is less than that of the multiplier, the only delay that adds to the path is that of the second adder.

Anonymous Armadillo 1y #487bbd ✓ Resolved

SP23-MT1-Q6.2

Q6.2 (3 points) What is the maximum hold time the registers can have so that there are no hold time violations in the circuit above? Reminder: you may assume that Input will not cause any hold time violations.

Solution: 25 ps
The shortest path between any two timed elements is actually the path from the SEL signal, which changes instantly at the rising edge of the clock, to the right register. This path has only delay 25 ps from the mux.
If you didn't see this path, the next-shortest path starts from the rightmost register and goes around, through the NOT gate, to the top-left register. This path has a delay of 30 ps (clk-to-q from the rightmost register) and 8 ps (from the NOT gate), for a total of 38 ps. Partial credit was given for this answer.

I dont really understand the explanation for this question. I thought the maximum hold time would is looking at the time from one register to the other? Why can we consider the time from the SEL signal to the right register? Is mux a register?

E Eddy Byun STAFF 1y #487bcd

#487afb

Anonymous Armadillo 1y #487bce

I see thanks, so essentially the SEL doesn't have a clk-to-q because it updates instantaneuously? Also I am still confused as to why we can count that as the lower bound, since arent we usually supposed to look at the minimum time from one register to the other when upperbounding the hold time?

Andrew Liu STAFF 1y #487cbd

Replying to Anonymous Armadillo

The hold time is usually thought of as that when all inputs go through registers, but more generally, one should think about "what is the minimum time after a clock tick

that all signals are stable." The easiest way to think about this is to treat any input that comes from a tunnel that updates at the rising edge of the clock as coming from a register with `clk-to-q = 0`.

♡ ···

Anonymous Chamois 1y #487bba ✓ Resolved

SUM-MT-Q1.9

$$= (\sim A\&C) \mid ((\sim A \ \& \ B) \mid B) \qquad \text{Idempotence (NOT)}$$
$$= (\sim A\&C) \mid B \qquad \text{Absorption (AND)}$$

how does the second part simplify to B

♡ ···

E Eddy Byun STAFF 1y #487bcb

Recall the absorption law says the following: A + AB = A.

!A * B + B can be simplified into B by the absorption law.

♡ ···

Anonymous Chamois 1y #487bae ✓ Resolved

SUM-MT-Q1.7

I get that for little endian it would be stored as 00000101 but when we are taking the strlen of it, we would count by char to read '1' and then stop because of the next number '0' (null). I would think it is strlen of 1 since null terminators don't count as the length?

"When strlen interprets this as a string, it will count length until the first null byte"

♡ ···

E Eddy Byun STAFF 1y #487bbf

Little endian stores 0x00000101 like this

Address a: 0x01 (least significant byte)

Address a + 1: 0x01

Address a + 2: 0x00

Address a + 3: 0x00

`strlen((char *) &x)` will see the value at addresses `a` and `a+1` are both not-NULL but see that the value at address `a+2` is NULL, so it will return 2.

♡ ···

Anonymous Swan 1y #487bac ✓ Resolved

SUM-MT-Q1.7: I got the binary rep of 257 to be 100000001 but i'm not sure how to get the byte rep from this. how did you get the four bytes representation as 00 01 01 01.

For Q1.7 and Q1.8, consider the following code snippet and assume that `int`s are 4 bytes.

```
int x = 257;
int y = strlen((char *) &x);
```

Q1.7 (1.5 points) In a little-endian system, what will `y` contain?

Solution: 2

`x` contains the four bytes `0x00 00 01 01`. Thus on a little-endian machine, the bytes will be stored with the nonzero bytes at lower addresses. When `strlen` interprets this as a string, it will count length until the first null byte - in this case, it will count both `0x01` bytes and report a length of 2.

Grading: All-or-nothing, except partial credit was given for interpreting it as a big-endian system.

♡ ···

Anonymous Chamois 1y #487bad ENDORSED

left pad 100000001 with 0s to get 32 bits and then find the hex of that i think

♡ ···

E Eddy Byun STAFF 1y #487bbe

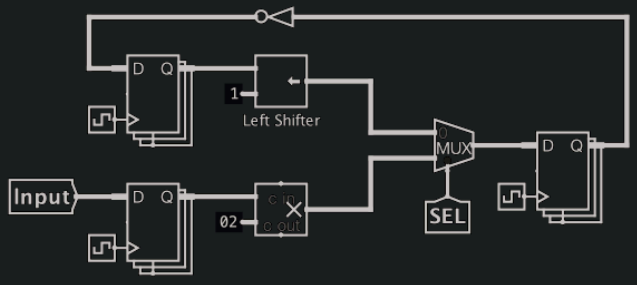
Yep as Anon Chamois says, we 0 pad 100000001 with 0's and then convert to hex.

♡ ···

Anonymous Cormorant 1y #487afd ✓ Resolved

Q6 *SDS* **(12 points)**

Consider the following circuit diagram. SEL is a single bit control signal that updates instantaneously at the rising edge of every clock cycle and remains stable during any given clock cycle. You may assume that `Input` will not cause any hold time violations.



$t_{NOT} = 8$ ps
$t_{mux} = 25$ ps
$t_{multiplier} = 1000$ ps
$t_{shifter} = 2$ ps
$t_{clk\text{-}q} = 30$ ps
$t_{setup} = 20$ ps

The left shifter combinational logic block shifts the top input by the number of bits indicated by the bottom input. The shifter in the diagram shifts the output of the connected register left by 1 bit.

Q6.1 (3 points) What is the minimum clock period for the circuit above such that it will always result in well-defined behavior?

ps

Q6.2 (3 points) What is the maximum hold time the registers can have so that there are no hold time violations in the circuit above? Reminder: you may assume that `Input` will not cause any hold time violations.

ps

SP23-MT-Q6.2

- The solution states that the next max hold time would be 38, with the rightmost register. But wouldent the next max hold time be 32, with the top left register getting an input signal having a

clk-q of 30, then going through the shifter with 2 ps. Thus needing max hold time to be 32 if we pretend the SEL is not there? Or Am I missing something

♡ ···

E Eddy Byun STAFF 1y #487baf

After going through the shifter, the signal has to go through the mux which adds another 25ps, so the path you described is not the next max hold time.

♡ ···

Anonymous Cormorant 1y #487bdb

Oh I thought that logic min, was the first place/time when the logic changed, which would be the leftshifter. I guess this assumption was wrong. Thank you!

♡ ···

Anonymous Grouse 1y #487adf ✓ Resolved

SP23-MT-Q6.1

More of a conceptual question, but if a problem asks us to find the minimum clock period, can we assume that means finding the longest path? If so, why?

Also, how do we know when to factor in the setup time? Do we add it in every time when approaching the input of a register?

♡ 1 ···

E Eddy Byun STAFF 1y #487bbc

Clock period >= clk-to-q delay + longest combinational delay + setup time. If we set our clock period to be clk-to-q delay + non-longest combinational delay + setup time, then our clock isn't going to give enough time for the longest combinational delay path to finish its computation. The clock will tick before the longest combinational path is done computing, and this may lead to unexpected behavior for our circuit.

A little bit confused about the setup time question. What do you mean by "factor in the setup time"?

♡ ···

Anonymous Grouse 1y #487bcf

Thanks for the response! As for the second question, I guess for an example, if we took the second shortest path between two timed elements, it would be the right most register that goes through the NOT gate, which is 38ps. However, why wouldn't we add an additional 20ps to 38ps to account for the setup time like we did in part 6.1?

♡ 1 ···

E Eddy Byun STAFF 1y #487def

↩ Replying to Anonymous Grouse

Recall these equations:

hold time <= clk-to-q + shortest combinational path

clock period >= clk-to-q + longest combinational path + setup time

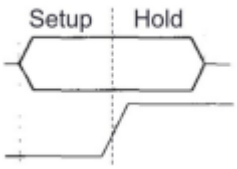
In 6.1, we were asked to find the minimum clock period, so we need to add the setup time.

In 6.2, we were asked for the maximum hold time, and from our equation we can see we don't need to add the setup time.

Beyond equations, why do we account for setup time for calculating the min clock period but we don't in calculating max hold time?

Remember that the setup time is time that you give for the signal that goes into the input of the register to become stable **before the rising edge of the clock**. The value that goes through the longest combinational path needs to become stable so we add the setup time.

Hold time is the amount of time we can keep the value at the input of the register **after the rising edge of the clock**. We can keep the input to the register after the rising edge of the clock for as long as the shortest combinational path + clk to q because beyond this time frame, the input to a register may change! Here's a diagram that I found that shows the difference b/w setup time and hold time (you can also find this diagram on the Discussion 6 worksheet!)



♡ 1 •••

Anonymous Grouse 1y #487adb ✓ Resolved

SP23-MT-Q4.3

I'm confused on the concept of the PC and what line 2 is and how we got the address for line 2.

Solution:

```
jal rd temp_label
addi rd rd 8
lui t0 imm
add rd rd t0
```

This one is tricky. The first thing to remember is what `auipc` does. First, it takes a 20-bit `imm`, and creates an immediate with these 20 bits as the upper 20 bits and 0s as the lower 12 bits. Then, it adds this new immediate with the current PC.

First, we have to get the value of the current PC. Looking through the reference card, the only instructions that put the PC in a register are the jump instructions. Here, we use `jal` to get the address of the `jal` instruction, plus 4 (i.e. the address of Line 2 here) into register `rd`.

However, the question says that we should be adding to the PC of the final instruction in our answer. Since our answer uses all 4 lines, and we got the address of Line 2, we need to add 8 bytes = 2 instructions to the PC we got, to find the PC of the final instruction.

Now, we can take `imm` and build the 32-bit value we'll be adding to PC. Note that `imm` is 20 bits, so an `addi` instruction (with only a 12-bit immediate) cannot handle this number. We have to use `lui` to put these 20 bits into the top 20 bits of a register. We choose to use `t0` because that's the only other register we can modify, and `rd` is already holding our current PC. (If we `lui`'d into `rd`, we'd mess up the PC we found.)

Finally, we use `add` to add the PC and the immediate.

Other answers are possible here, e.g. putting the PC in `t0` and the immediate in `rd` before adding.

♡ 2 ···

E Eddy Byun STAFF 1y #487bde

PC is the program counter, and it tell us which instruction we're currently executing. When we do `jal rd temp_label`, we will set `rd` equal to `PC + 4`. Notice that the last instruction `add rd rd t0` is 3 instructions away from `jal rd temp_label`, which means it's `PC + 12` away from `jal rd temp_label`. Since we've set `PC+4` to our `rd` already, we need to add `8`, which is why the second line is `addi rd rd 8`

♡ ···

Anonymous Manatee 1y #487ebb

A couple of follow-up questions:

1) When we're executing jal rd temp_label, isn't PC+4 temp_label?

2) The question says that we should use the PC of the fourth line. However, when we're executing the last line (add rd rd t0), the PC points to the instruction after that. Let final_pc be the address of the instruction below add rd rd t0, which is the PC value we want. Then add is final_pc - 4, lui is final_pc - 8, addi is final_pc - 12. Therefore, we are loading final_pc - 12 into rd via the jalr command. Shouldn't we be adding 12 instead of 8?

♡ ···

Anonymous Gerbil 1y #487ada ✓ Resolved

SP23-MT-Q1.11

isn't lowercase 'o' 0x6F in ASCII and not 0x65?

♡ 3 ···

J Jero Wang STAFF 1y #487deb

#487dea

♡ 1 ···

Anonymous Gerbil 1y #487ded

thanku!

♡ ···

Anonymous Stingray 1y #487ace ✓ Resolved

Q3.4 (5 points) How many non-zero numbers **x** are there in this floating point system where **x** and **2x** differ by exactly 1 bit?

Write your answer as a sum or difference of unique powers of 2 (e.g. $2^3 - 2^2 + 2^1$).

Solution: $2^{15} - 2^{12}$

Note that to double a floating-point number, we have to increase the exponent by 1.

When we increase the exponent by 1, what could happen? If the least-significant bit of the exponent (as represented in bias notation) is 0, then the 0 gets flipped to a 1. For example, `0b11010 + 0b1 = 0b11011`.

If the least-significant bit of the exponent is 1, then the 1 flips to a 0, a 1 carries over into the next place, and other bits must change. For example, `0b11011 + 0b1 = 0b11100`, which changed 3 bits.

In summary, we need to figure out how many floating-point numbers have a least-significant exponent bit of 0. This is half of the floating-point numbers (if you just wrote out all the bit representations, half of them would have a 0 in the exponent LSB). There are 2^{16} floating-point numbers, and 2^{15} of them have a 0 in the exponent LSB.

The last thing we need to do is remove the infinities, NaNs, and denorms, because adding 1 to the exponent does not double these numbers. (In the case of denorms, changing the exponent also introduces the implicit 1, which changes the number in other ways than just a simple doubling.)

Denorms: Exponent is all 0s. The 11 sign/mantissa bits could be anything, so there are 2^{11} denorms we have to remove from our final total.

Infinities and NaNs: Exponent is all 1s. Just like the denorms, there are 2^{11} more values we have to remove.

In total, we throw out $2 \times 2^{11} = 2^{12}$ values from our original total of 2^{15}.

The original idea for this question came from an ex-TA who went on to teach other classes, so you can't blame anyone on the current staff for it. It's a tricky question!

for this problem, don't you need to re-add the denormalized numbers with a least significant bit of 0 in the significant?

for ex.
0.0001 * 2^-14 multiplied by 2 would give 0.0010 * 2^-14 so we would have a final answer of 2^15 - 2^12 + 2^10

♡ ···

E Eddy Byun STAFF 1y #487aef

The example that you gave changes two bits. Also, adding changing the least significant bit of the mantissa from 0 to 1 does not double the denorm number.

Imagine our mantissa is 0000010000 - this is going to be $0.0000010000_2 * 2^{-14}$

If I change the least significant mantissa bit to a 1, I get $0.0000010001_2 * 2^{-14}$, and this will not double the number.

♡ 1 ···

Anonymous Heron 1y #487acd ✓ Resolved

SU23-MT-Q2

What is the answer to Q4.1 --> the RISC-V instruction that is_odd would translate to?

Doesn't appear in the solutions

♡ ···

E Eddy Byun STAFF 1y #487aec

#487bf

♡ ···

Anonymous Heron 1y #487acc ✓ Resolved

SU23-MT-Q2

Why is it

User* cur_user = &lib->users[i]

With the & instead of just lib->users[i]?

♡ 1 ···

E Eddy Byun STAFF 1y #487afe

`lib->users[i]` is of type `User` . Note that on the left hand side, we want a `User*` (a pointer to a User struct). Since I want a pointer to the `User` instead of the `User` itself, I get the address of `lib->users[i]` , which is going to be `&lib->users[i]`

♡ ···

Anonymous Grouse 1y #487acb ✓ Resolved

SP23-MT-Q2.3 and 2.9

What's the difference again between using an & in front of a pointer vs using * in front, for example between 2.3 and 2.9?

Solution:

Q2.1: `calloc(1, sizeof(Cheatsheet)`

Note that we need to `calloc` in this case in order to set `total_length` equal to 0.

Q2.2: `->student_id`

Q2.3: `&sheet->pages[i]`

When we allocate memory on the heap for a `Cheatsheet`, we allocate memory for a `Page` array of size `NUM_PAGES`. Therefore, we already allocated memory for each `Page`. In order to get the correct `Page`, we need to index into the correct `Page` in our `Cheatsheet` (`sheet->pages[i]`). To get the pointer to this `Page`, we will use the `&` to get a pointer to this `Page` (`&sheet->pages[i]`)

Q2.4: `->num`

Q2.5: `->data`

Q2.6: `malloc(sizeof(char) * (strlen(contents[i]) + 1))`

Note that we allocated memory for a `char` pointer but we now need to actually allocate memory for the string itself. Also, `strlen` doesn't consider the null-terminator, so we need to add 1.

Q2.7: `->data`

Q2.8: `->total_length`

Q2.9: `*ch = sheet`

♡ 1 ···

E Eddy Byun STAFF 1y #487bbb

The `&` operator takes the address of some variable. The `*` in front of a pointer retrieves the value that is pointed by the pointer.

♡ ···

Anonymous Grouse 1y #487bda

Why wouldn't we be using * in front of sheet in 2.3? Is it because we are trying to get the specific address of that specific sheet to point to a specific page? I originally used * to dereference since we initially created a pointer to sheet?

♡ 1 ···

E Eddy Byun STAFF 1y #487dff

Replying to Anonymous Grouse

*(sheet->pages[i]) is going to treat the value of (sheet->pages[i]) as an address and dereference that address. We see that we want `Page* page` on the left. sheet->pages[i] is going to give us the `Page` struct, but we want a `Page*` (a pointer to the `Page`. The `&` operand is going to give me the address of (`sheet->pages[i]`), which is the `Page*` that I want.

♡ 1 ···

Anonymous Bee 1y #487fde

could we do ch = &sheet?

♡ 1 ···

Anonymous Gerbil 1y #487aca ✓ Resolved

SP23-MT-Q1.7

How do you simplify !Y + Y*Z into !Y + Z?

♡ ···

E Eddy Byun STAFF 1y #487afc

I used the absorption rule: A + AB = A

In this case, you have !Y + !Y*Z = !Y

Plug in !Y + !Y*Z into !Y to get !Y + !Y*Z + Y*Z = !Y + Z * (Y + !Y) = !Y + Z

♡ 2 ···

Anonymous Gerbil 1y #487bab

THANK YOU SO MUCH

♡ ···

Anonymous Snake 1y #487abc ✓ Resolved

SP23-MT-Q6.3

How does replacing multiplier with shifter (Switch constant from 2 to 1) not affect the behavior of the circuit? I don't understand the usage of the shifter. Would appreciate any examples as well.

♡ ···

E Eddy Byun STAFF 1y #487afa

Note how the bottom multiplier is multiply the top input into the multiplier block by 2. This is the same as left shifting by 1. As an example, consider this binary number: 0b0011 (3 in decimal). Shifting it left by 1 gives us 0b0110 (6 in decimal). A shift left operation by one is similar to multiplying the number by 2, which is why we can replace the multiplier with a shifter.

♡ ···

Anonymous Snake 1y #487abb ✓ Resolved

SP23-MT-Q6.2

Why do we look at the path from the SEL signal to the right register which has a hold time of 25ps instead of right register to top left register which has a hold time of 8 (from NOT gate)? Also what is an SEL gate and timed element? And what are other timed elements?

♡ ···

E Eddy Byun STAFF 1y #487afb

The SEL signal is "a single bit control signal that updates instantaneously at the rising edge of every clock cycle and remains stable during any given clock cycle" (from the exam). Since SEL will update instantaneously, we consider the delay for the mux (which is 25 ps), which may change the output of the mux and this in turn changes the input to the register. The delay from the right register to the left register is 38ps (clk-to-q + not delays), and this delay is shorter than the delay from the SEL to the input of the right register.

♡ ···

Anonymous Swan 1y #487dad

Is SEL treated like a register? i thought to find the path you must go from the output of a register to input of another

♡ ···

E Eddy Byun STAFF 1y #487dfe

↩ Replying to Anonymous Swan

I would say it's like the input that we saw from HW 5.6 (the X and Y inputs)

♡ ...

Anonymous Grouse 1y #487aba ✓ Resolved

SP23-MT2-Q1.6

As a conceptual question, when do we know when to sign extend? Because when I was calculating the imm, I took the MSB and sign extended with 1's since 2^9 was a 1.

♡ 1 ...

E Eddy Byun STAFF 1y #487bdf

Note that the immediate is positive. Remember that for 2's complement, the most significant bit (leftmost bit) tells us both the sign and magnitude of the binary number (if the leftmost bit is a 1, we have a negative number if it's 0, it's a positive number). We see that the number we have is positive, which means we sign extend by 0 since our number is positive.

♡ ...

Anonymous Grouse 1y #487cab

Wouldn't the leftmost bit be a 1 since 588=512+64+8+4 which means 2^9 would be the leftmost bit which is a 1?

♡ 1 ...

E Eddy Byun STAFF 1y #487cfe

Replying to Anonymous Grouse

We have 21 bits for the immediate for J instructions, so the leftmost bit would be the bit at position 2^{20}. This bit is going to be a 0 because our immediate is positive.

♡ ...

Anonymous Swan 1y #487dcf

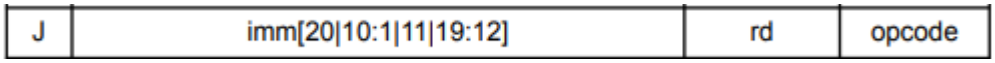
Replying to Eddy Byun

for the sign extending the imm, I found the imm of 588 to be 0010 0100 1100 -> 24C in hex. Would sign extending this be to pad the left with 0000 0000 to make this 32 bits? I am confused because I see the answer is 0x24C009EF instead of 0x0024C9EF as with the sign extending i mentioned

♡ ...

E Eddy Byun STAFF 1y #487eaa

Replying to Anonymous Swan



Careful with how we organize our immediate for J type instructions!

♡ ...

Anonymous Octopus 1y #487aaf ✓ Resolved

Hi,

For **SP23-MT1-Q6.2-6.4**, for 6.3 and 6.4, would adding another register between the multiplier and the mux be acceptable, along with a new minimum clock period of 1050ps?

And for 6.2, why do we consider SEL? I thought that when determining minimum clock period, we only consider any path between any two registers. Is SEL acting like the output signal from a register since it is timed with the clock, and that's why it's considered?

♡ ...

E Eddy Byun STAFF 1y #487baa

Adding a register between the multiplier and the mux is going to change the behavior of the circuit, so this would not be a valid modification.

For 6.2, we are trying to find the maximum hold time. Recall that the general equation for the hold time is Hold time ≤ clk-to-q delay + shortest combinational delay. There is no clk-to-q delay for SEL to be updated since it gets updated instantaneously. It takes 25ps for the mux to get updated, and updating the SEL bit may change the output for the mux, which is why the max hold time is 25 ps.

♡ ···

Anonymous Chicken 1y #487ff ✓ Resolved

Q1.11 (1.5 points) Represent 1.5×2^{-511} in hex using a binary floating point representation, which follows IEEE-754 standard conventions, but has 10 exponent bits (and a standard bias of -511) and 21 mantissa bits.

Solution: 0x00180000

Looking at the number, it is equal to $1.1_2 \times 2^{-511}$. Since we can only represent exponents from -510 to 511 with a normal floating point number, this means our number must be represented as a denormalized number, with a fixed exponent of 2^{-510}. Rewriting our number to use this new exponent gives $0.11_2 \times 2^{-510}$. Thus the floating point representation is:

```
  sign  exponent   mantissa
  0     0000000000 11000000000000000000
  0000 0000 0001 1000 0000 0000 0000 0000
0x0    0    1    8    0    0    0    0
```

Grading: Partial credit was awarded for having the correct sign bit, having the correct exponent bits, and having the correct mantissa.

Q1.11

I don't understand why the Mantissa is 11? I understand why -511 doesn't work, but I also don't understand why the number converts to 0.11 * 2^-510. I thought moving the decimal place over would mean it would be 0.15 * 2^-510?

♡ ···

E Eddy Byun STAFF 1y #487aed

$1.5 * 2^{-511} = 1.1_2 * 2^{-511} = 0.11_2 * 2^{-510}$. Translate 1.5 to binary first. Afterwards, moving the decimal point to the left by one will cause us to increase the exponent by 1.

♡ ···

Anonymous Snake 1y #487fe ✓ Resolved

SP23-MT-Q3.4

I want to double check if my understanding of the solution is correct. The solution says the total number of floating points that has its least significant exponent bit as 0 is **half** of the total number of floating points (2^16). My understanding is that at that last significant exponent bit, a floating point can have either "0" or "1", so we can partition all of the possible fps into two groups, where half of them is in "team last bit 0" and the other half in "team last bit 1".

If my understanding is correct, why do we still have to remove NaN/infinities? Since for these representations all exponent bits are 1s, wouldn't they be included in "team last bit 1" already? And at the end we just have to subtract the denorms from 2^15?

I also don't really understand what "introduces implicit 1" mean in the explanation for why adding 1 to the exponent bit for denorms is not the same as doubling the value.

♡ ···

E Eddy Byun STAFF 1y #487aee

We have to remove NaN/infinities for the following case: Our original value exponent bits are 0b11110, and we change the least significant exponent bit so we now have 0b11111. In this case, our number would be a NAN or infinity.

Regarding the denorm question, let's consider the following denorm number:

0 00000 0100000000, this would be $0.01_2 * 2^{-14} = 0.25 * 2^{-14}$

If I change my least significant exponent bit so now I have

0 00001 0100000000, this would be $1.01_2 * 2^{-14} = 1.25 * 2^{-14}$.

$1.25 * 2^{-14}$ is not twice as large as $0.25 * 2^{-14}$. Denorms have implicit 0's in front of the mantissa while normalized numbers have implicit 1's in front of the mantissa. As soon as the exponent changes from 00000 to 00001, we go from a denorm number to a normalized number, and this is going to change between having an implicit 0 in front of the mantissa to having an implicit 1.

♡ 1 ···

Anonymous Badger 1y #487fd ✓ Resolved

offsets for jump instruction will always be resolved in the linker step. why is this statement false?

♡ ···

E Eddy Byun STAFF 1y #487aeb

It's because the offsets can be resolved in both the assembler and linker stages.

♡ ···

Anonymous Camel 1y #487fb ✓ Resolved

SU23-MT-Q4.1

Solution:

```
1 num_steps:
      # Prologue
      # Omitted
2     addi s0 x0 0
3 loop_start:
4     addi t0 x0 1
5     beq a0 t0 loop_end
6     jal ra next_number
7     addi s0 s0 1
8     j loop_start
9 loop_end:
10    add a0 t0 x0
      # Epilogue
      # Omitted
11    jr ra
```

Grading: Credit was given for all equivalent answers, with points deducted for using `s` registers or breaking calling convention.

On line 10 of the solution, isnt it supposed to be "add a0 s0 x0" instead of "add a0 t0 x0"?

♡ 1 ···

E Eddy Byun STAFF 1y #487aea

Yea, it should be `add a0 s0 x0` or equivalent. Sorry for the confusion!

♡ ···

Anonymous Grouse 1y #487bdd

would mv a0 s0 work?

♡ 1 ···

E Eddy Byun STAFF 1y #487dfc

Replying to Anonymous Grouse

Yes, since it's equivalent to `add a0 s0 x0`

♡ 1 ···

Anonymous Monkey 1y #487cec

Would the #omitted part for both the prologue and epilogue be the logic to do calling convention for saving and loading s0 to and from the stack?

♡ ···

E Eddy Byun STAFF 1y #487dfd

Replying to Anonymous Monkey

Yes, Decrementing/Incrementing stack, storing and loading `s0` and `ra` for CC purposes.

♡ ···

Anonymous Sandpiper 1y #487fa ✓ Resolved

SU23-MT1-Q2.13

I don't understand the reasoning behind the solution where it says `lib->users[i]` cannot be freed since it was not created using malloc. How else is the space being allocated?

♡ 1 ···

Anonymous Dove 1y #487fc

for this question is the fact that you are freeing all of the malloc space within users but not freeing the users array a valid response?

♡ ···

E Eddy Byun STAFF 1y #487ade

No, see #487add on how we should free the `users` array.

♡ ···

E Eddy Byun STAFF 1y #487add

We can't free each index in `lib->users` which is what `free(lib->users[i])` attempts to do. When we called `realloc`, it returned a pointer to the `users` array, so if we want to free the `users` array, we have to do `free(lib->users)`. It's similar to how in snek, we free'd our snake array like this: `free(state->snakes)` - we couldn't iterate through the snakes array and do `free(state->snakes[i])`

♡ ···

Anonymous Spoonbill 1y #487bfd

no but i dont understand why can we not iterate through the array and do that. is it because only outputs of malloc/realloc/calloc can be freed?

♡ ···

E Eddy Byun STAFF 1y #487dfb

Replying to Anonymous Spoonbill

Calling `free` on something that was not returned by `malloc`, `calloc`, or `realloc` is undefined behavior.

♡ ···

Anonymous Octopus 1y #487ef ✓ Resolved

Hi,

For **SP23-MT1-Q4.1**, is `lhu rd imm(rs1)` followed by `andi rd rd 00001111` acceptable?

♡ ···

E Eddy Byun STAFF 1y #487adc

It would have to be `lhu rd imm(rs1)` followed by `andi rd rd 0x000000FF` but yes you could've used a different load instruction.

♡ ···

Anonymous Raccoon 1y #487ee ✓ Resolved

Sp 23 -MT 2

Q1.6 (3 points) Translate the following RISC-V instruction to its hexadecimal counterpart.

```
jal s3 588
```

Hint: 588 = 512 + 64 + 8 + 4

Solution: `0x24C009EF`

Confused how to arrange the imm field. I keep getting the wrong answer and am unsure (I get 0x0024C9EF).

♡ ···

E Eddy Byun STAFF 1y #487aae

Opcode: 110 1111

`s3 = x19` = 10011

588 = 0b0 0000 0000 0010 0100 1100 but we don't write the last bit since it's implicitly 0 so bits [20:1] would be 0b0000 0000 0001 0010 0110

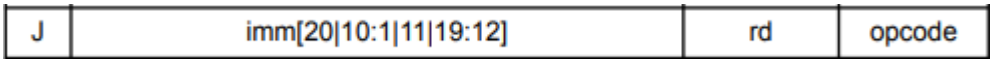
Bit 20: 0

Bit [10:1]: 0100100110

Bit 11: 0

Bit [19:12]: 00000000

Put it all together!



0 0100100110 0 00000000 10011 1101111

0010 0100 1100 0000 0000 1001 1110 1111 = 0x24C009EF

♡ 1 ···

Anonymous Horse 1y #487ccb

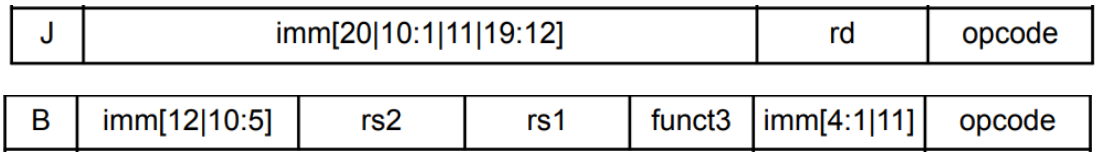
Why do we not write the last bit? Is this true in all labels or constants?

♡ ···

E Eddy Byun STAFF 1y #487efb

Replying to Anonymous Horse

For B and J type instructions, we don't write the last bit because it will always be a 0



The RISC-V instructions that we teach are 32 bits (4-bytes), but RISC-V also supports 16 bits (2 bytes), which is why we can omit the rightmost bit.

♡ ···

Anonymous Gaur 1y #487efc

Replying to Eddy Byun

When we have x19, isn't it 25 in decimal and thus in binary it would be 11001? Or is the x19 just referring to 19 in decimal?

♡ ···

Anonymous Octopus 1y #487ed ✓ Resolved

Hi,

For **SP23-MT1-Q2.6**, I know it's better to include sizeof(char) but would it be wrong to omit it since sizeof(char) is just 1 byte?

♡ ···

E Eddy Byun STAFF 1y #487aad

Writing 1 isn't wrong but I'd recommend writing `sizeof(char)`

♡ ···

Anonymous Octopus 1y #487ec ✓ Resolved

Hi,

For **SP23-MT1-Q2.1**, would `realloc(*ch, sizeof(Cheatsheet))` be acceptable, ignoring the fact that this doesn't set total_length to 0 (in other words, does this allocate memory correctly)?

♡ ···

E Eddy Byun STAFF 1y #487aac

No, this would be undefined behavior. `realloc(*ch, sizeof(Cheatsheet))` would only work if we knew that `*ch == NULL` since `realloc` behaves like `malloc` if the pointer that you pass in is equal to `NULL`. In this problem, we don't say anything about the value of `*ch` Also, like you said, this won'g set `total_length` to 0.

♡ ···

Anonymous Reindeer 1y #487bec

why does calloc set the total_length = 0; how does it know which variable to set to 0 through a call to calloc?

♡ ···

E Eddy Byun STAFF 1y #487dfa

Replying to Anonymous Reindeer

calloc sets all allocated memory to 0:
https://www.tutorialspoint.com/c_standard_library/c_function_calloc.htm

♡ ···

Q Quan Nguyen 1y #487dc ✓ Resolved

SU23-Midterm-Q1.1

Does the term "the same range of numbers" refer to the same amount of numbers represented or the same values of numbers?

♡ ···

J Justin Yokota STAFF 1y #487dd

Neither; this refers to the same distance between the largest representable number and the smallest representable number.

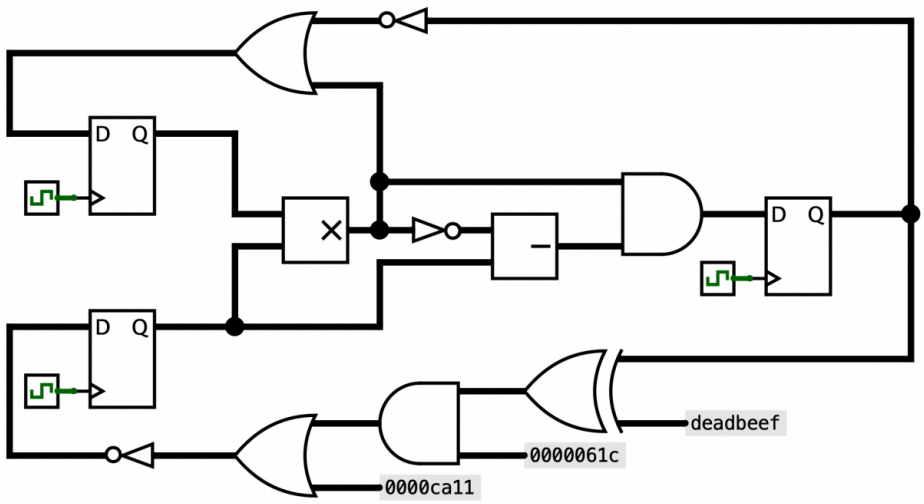
♡ ···

Anonymous Rook 1y #487db ✓ Resolved

SU23-MT-Q6.4

Q6 *SDS* **(8 points**

Consider the following circuit diagram and component delays:



$t_{\text{AND}} = 20\text{ps}$
$t_{\text{NOT}} = 5\text{ps}$
$t_{\text{OR}} = 20\text{ps}$
$t_{\text{XOR}} = 25\text{ps}$
$t_{\text{multiplier}} = 50\text{ps}$
$t_{\text{subtract}} = 35\text{ps}$
$t_{\text{clk-q}} = 5\text{ps}$
$t_{\text{setup}} = 2\text{ps}$

Q6.4 (2 points) Suppose this circuit only deals with two's complement integers. Currently, the subtractor component has a delay of 35ps. What is the maximum delay an adder component can have such that we could replace the subtractor component with adders, NOT gates, and constants to achieve the same delay as the subtractor while maintaining the same behavior? You may assume that constants have no delay.

As a reminder, the subtract component does the following operation:
output = top input - bottom input

Solution: 35ps

Since we're dealing with two's complement numbers, subtracting by x is equivalent to adding $\sim x + 1$, where $\sim x$ flips all of the bits of x. As a result, we can chain together a NOT gate, an adder with a constant 1, and another adder (to add the output of the previous adder and the top input of the subtractor) to achieve the same behavior.

The intent of the question is for students to realize that the NOT gate and the first adder does not actually add any additional delay, since the multiplier/NOT gate combo of the top input of the subtractor takes more time than the NOT gate/adder combo for the bottom input. Therefore, the adder can have a delay of 35ps (the same as the existing subtractor) for the circuit to maintain the same timing behavior.

Grading: All-or-nothing, except 15ps was also awarded full credit due to ambiguity raised within this question (assuming that the subtractor should be treated as a black box, and replaced with a black box consisting of two adders, a NOT gate, and a constant).

The solution says that the NOT gate and the ADDER will not add any additional delay, is this because the NOT gate is placed before the bottom input of the ADDER?

♡ ···

J Justin Yokota STAFF 1y #487de

Yup. And that bottom input already arrives way before the top input. So it can be delayed a bit.

♡ ···

Anonymous Swan 1y #487da ✓ Resolved

SU23-MT-Q6.4

I understand that by defintion two's complement ~X+1 but how does that have anything to do with the answer 35? i don't understand why the adder delay is the same as the subtractor delay

♡ ···

J Justin Yokota STAFF 1y #487df

Since A-B = A+(-B) = A+(~B+1), we can replace the subtractor with a NOT on B, then an ADD to 1, then an ADD to A. The goal, then, is to determine the maximum delay on ADD that lets the circuit maintain the same clock period.

♡ ···

Anonymous Wolf 1y #487ce ✓ Resolved

SU23-MT-Q2

Could `free(&lib->users[i])` also work in the for loop?

♡ ···

E Eddy Byun STAFF 1y #487aab

No, the only way to correctly free the `users` array is to do `free(lib->users)`.

♡ ···

Anonymous Wolf 1y #487cd ✓ Resolved

SU23-MT-Q2

Why isn't the memset part `memset(curr_user->borrowed_books, NULL, MAX_BORROWS * sizeof(Book*))`, with the pointer `Book*` instead of `Book`? I thought `borrowed_books` in struct `User` was defined as an array of `Book*`.

♡ ···

E Eddy Byun STAFF 1y #487aaa

Yea, you're right it should be `Book*` - sorry for the typo!

♡ ···

Anonymous Turtle 1y #487cb ✓ Resolved

SP23-MT-Q1.7

Why does x=257 contain the four bytes 0x00 00 01 01?

♡ ···

 Andrew Liu STAFF 1y #487cf

Integers are 4B types, so we know there are 4 bits. Then, 257 = 256 + 1 = 2^8 + 2^0 = 0b1 0000 0001 = 0b 0000 0000 0000 0000 0000 0001 0000 0001 = 0x00000101

♡ ···

Anonymous Goat 1y #487be ✓ Resolved

SP23-MT-Q2.10-12
Why is num pages on code? I think it's previously defined outside of the function so it should be on static. And sheet is on stack because it's a pointer defined in function. The content of sheet is on heap because it allocates memory. Are my reasonings correct?

♡ ···

E Eddy Byun STAFF 1y #487ca

Take a look at Homework 2, Memory Alpha Model! It's the same reason that SPOCK is in the code segment; all instances of NUM_PAGES will be replaced by 8 by the compiler so it is

going to be in the code segment.

♡ ···

Anonymous Cod 1y #487cfb

What do you mean replaced by 8 in the compiler?

♡1 ···

Anonymous Reindeer 1y #487bd ✓ Resolved

SU23-MT-4.1: what is the answer to this question?

♡ ···

E Eddy Byun STAFF 1y #487bf

Sorry, it should be `andi rd rs1 1`

♡ ···

Anonymous Spoonbill 1y #487bfe

how? dont we want to store 1 in rd if rs1 is an odd number? how does this achieve that

♡ ···

Anonymous Aardvark 1y #487cac

Replying to Anonymous Spoonbill

andi compares bitwise rs1 and 1, for a binary value to be odd, the last bit has to be 1. therefore, if rs1 is odd, andi would compute 1 and 1 which stores 1 in rd. if rs1 is even, it would've been 0 and 1 which is 0.

♡ ···

Anonymous Spoonbill 1y #487cbe

Replying to Anonymous Aardvark

Ok, then does andi only compare the last digit of rs1 or does it compare every digit and then just store the last value

♡ ···

Anonymous Aardvark 1y #487cbf

Replying to Anonymous Spoonbill

it compares every bit but technically the immediate 1 is 000001 with arbitrary leading 0s

♡ ···

Anonymous Sardine 1y #487ceb

Replying to Anonymous Spoonbill

andi'ing:

0b1101100001

and

0b0000000001

looks at the first starting bit of each and puts that in that digits place.

So the MSB of both would be 0, because 1and0 = 0. The next bit after the MSB looks at 1and0 again. Only the LSB of both has 1and1

♡ ···

Anonymous Reindeer 1y #487ab ✓ Resolved

SU23-MT-2.16:

why is init_users on stack, even though it is a pointer (I thought it would be on the heap)?

♡ ···

Andrew Liu STAFF 1y #487ae

Not all pointers are on the heap, for example

```
foo() {
 int x = 3;
 int* y = &x;
}
```

creates the pointer `y` on the stack.

For this case in particular, remember that function arguments are pushed onto the stack by a program, so `lib` is on the stack.

♡ 1 ···

Anonymous Reindeer 1y #487bc

Could you explain why in the example you gave, pointer y is on the stack?

♡ ···

Justin Yokota STAFF 1y #487ea

Replying to Anonymous Reindeer

x is on the stack

y points to x

Therefore, y points to the stack.

♡ 2 ···

Anonymous Reindeer 1y #487eb

Replying to Justin Yokota

Thank you!

♡ ···

Anonymous Reindeer 1y #487aa ✓ Resolved

SU23-MT-2:

what does &lib->users[i] mean, and how is it different from lib->users[i], and lib->users?

♡ ···

Andrew Liu STAFF 1y #487af

`&lib->users[i]` = "The address of the `i` th element of the `users` array in the structure `lib`"

`lib->users[i]` = "The `i` th element of the `users` array in the structure `lib`"

`lib->users` = "The `users` array in the structure `lib`"

♡ 2 ···

Anonymous Reindeer 1y #487bb

Thank you!

♡ ···

A Abhi Pomalapally 1y #487f ✓ Resolved

SU23-MT-Q1.8:

In big endian, would it change the way x is stored, and put the 01 on the left? Why is x still represented the same way in 1.8 (big-endian) as it is in 1.7 (little endian)?

♡ ···

 Andrew Liu STAFF 1y #487ad

Yep! Endianness is all about the ordering in memory; x will always be 0b00000101, but the memory addresses assigned to each byte are different between systems.

♡ ···

A Abhi Pomalapally 1y #487ba

Thank you!

♡ ···

Anonymous Swan 1y #487aff

How did you find the memory bytes of x to be: 0b00000101

♡ ···

Anonymous Snake 1y #487cad

For this problem in particular,

it shouldn't matter if it's little endian or big endian right? Since 0xA7 is 'one byte'?

♡ ···

Anonymous Armadillo 1y #487c ✓ Resolved

SP23-MT-Q1.11

for the representation of "o!!\00" shouldnt \00 be the least significant byte, since its on the right? Im just confused as to how we got the reverse order, rather than 0x65212100

♡ ···

J Justin Yokota STAFF 1y #487d

Remember that the system is little endian!

♡ ···

Anonymous Goat 1y #487e

Shouldn't that be 6F? I think o is 0x6F in ASCII though while 65 is for e right?

♡ 2 ···

Anonymous Cat 1y #487cc

↩ Replying to Anonymous Goat

yeah I think it's another mistake in the solutions

♡ 1 ···

R Robert Yang 1y #487acf

Replying to Anonymous Goat

Yeah I got 6F as well I was so confused lol

♡ 1 ···

Anonymous Dove 1y #487bea

Replying to Robert Yang

Regarding 1.10, are you including "!" in the calculations if so wouldn't that be the correct answer since we are storing in little-endian? Or is this another mistake within the solutions?

♡ ···

Anonymous Dragonfly 1y #487bfb

So is 0x0021216F the correct order or 0x6F212100? I know we're in little endian but I still don't see why 0x0021216F is correct

♡ 2 ···

Anonymous Duck 1y #487fab

Replying to Anonymous Dragonfly

didn't they treat it as big endian in the part before that?

♡ ···

Anonymous Pheasant 1y #487aabc

Replying to Anonymous Dragonfly

The correct answer is 0x0021216F.

In little endian the lowest sig byte is stored at the lowest address.

Notice that when we write:
"hello!!\0", we write it in such a way that 'h' is stored at the lowest address. (and the null terminator is at the highest)

So, in our sequence "o!!\0", we get that 'o' is the least sig digit = 0x6F.

'\0' is at the highest address, so it's the most sig digit in our number = 0x00.

♡ ···

Anonymous Armadillo 1y #487a ✓ Resolved

Where can I look for past Sp23 midterm 1 ?

♡ ···

J Justin Yokota STAFF 1y #487b

SP 23 had only one midterm (as with this semester). We've placed that midterm under the column "midterm 2".

♡ ···