

You are viewing this thread in readonly mode.

[Midterm] Past Exams - 2020 #881



Jero Wang ADMIN
2 years ago in Exam – Midterm

881
VIEWS



You can find the past exams here: <https://cs61c.org/sp23/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!

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**

[Summer 2020 midterm walkthrough](#)

[Fall 2020 midterm Bit Manipulations Walkthrough](#)

[Fall 2020 midterm 1 Slip Walkthrough](#)



Anonymous Scorpion 2y #881bea Unresolved

Fa20-Final-Section 3 Potpourri Part B

Part B ECC — 1 pt Given the following bitstring `DATASTRING`, calculate the encoded bitstring to perform single-bit error correction with `TYPE` parity. Give your final answer in binary with the fewest number of bits needed to recover the given correct bitstring should a 1-bit error occur in the given bitstring without the `0b` prefix. For example, if your answer is `0b101000101`, you will only receive credit for writing `101000101` and **NOT** `0b101000101`, `0b0000101000101`, and similar answers.

`DATASTRING`: see PrairieLearn for your version; sample solution will use `0b010011000`

`TYPE`: even/odd parity; sample solution will use **odd** parity

Solutions
`101100011000`

I'm not sure how to get the solution. Do we discard prefix 0's when encoding? If so, shouldn't encoded bit 7 (index 1, from left to right, `101100011000`) be 1 according to the 4th data bit `10011...?`



Anonymous Magpie 2y #881bdf ✓ Resolved

SU20-MT1-Q6-a(ii)

Why the answer is not $2^5 - 2$? I think we are not considering 2^1 and 2^3 right?

for 2^3 , I think we can represent as $1.0000 * 2^3$, Why don't we subtract another for this 2^3 ?

ii. (1.5 pt)

How many Floating Point numbers are in the interval of $(2^1, 2^3)$? (Answer in decimal)

31

$$2^5 - 1 = 31$$



Anonymous Peafowl 2y #881bdb

✓ Resolved

su20-final-q4

for 4c why is this the answer?

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

102

$$25 + 50(\text{clock period}) + 4(\text{clk-to-q}) + 14(\text{OR}) + 9(\text{AND}) = 102 \text{ ns}$$



Jero Wang ADMIN 2y #881bdd

Since x_input becomes 1 30ns after the first rising edge of the clock, it becomes 1 at $t=25+30=55\text{ns}$. The next rising edge of the clock at $t=25+50=75\text{ns}$ would store this 1 into Reg0, and also store a 0 into Reg1. Then, the signal has to travel through the OR gate connected to Reg0 and then the AND gate before the value of y_output is updated (and this corresponds to the clk-to-q + OR + AND part of the solution).



Anonymous Stork 2y #881bcd

✓ Resolved

FA20-MT-Q5

Question 5 RISC-V Datapath, Control, and Pipelining

Is this type of problem in scope?



Erik Yang TA 2y #881bce

nope



Anonymous Dolphin 2y #881bbd

✓ Resolved

SP20-MT1-Q6a

Followup to #881db

I have the same question as the original poster. I thought the numbers received were 16 bit floating point numbers so wouldn't we have to do sign, exponent, mantissa conversions? How come here we can just insert 1s?



Erik Yang TA 2y #881bbe

here we're just asking for what the biggest hex value that can be returned, so if you just fill in all 1's it will return the biggest hex number

♡ ...



Anonymous Dolphin 2y #881bbf

Ah, so while the encoding for largest number is 0x7777, the actual number it represents would be 0x7770(not in floating point representation)?

♡ ...

E

Erik Yang TA 2y #881bca

↩ Replying to Anonymous Dolphin

why would the last byte be 0? just asking how you got that

♡ ...



Anonymous Dolphin 2y #881bcb

↩ Replying to Erik Yang

sign: 0

exponent: 11101 = 29

mantissa: 11 0111 0111

value = sign * 1.mantissa * 2^(exponent - 15) = 1.11 0111 0111 * 2^(14)

= 0111 0111 0111 0000 = 0x7770

♡ ...

E

Erik Yang TA 2y #881bcc

↩ Replying to Anonymous Dolphin

oh yeah i see what you mean now ty for that, its just the max for the actual hex number not the fp number

♡ ...



Anonymous Dolphin 2y #881bbb

✓ Resolved

SP20-MT1-Q5b

For double a, double b, could we instead dereference p1,p2 first and then cast to double?

Eg.

double a = (double) *p1

double b = (double) *p2

If not why not? The (double *) cast seems to be slightly repetitive.

♡ ...



Rosalie Fang ADMIN 2y #881bbc

No, because p1 and p2 are (void *) pointers, if you just try to dereference it, it will not know how many bytes to read when you dereference. Even if p1 started out as (int *) pointers, say it 's a list of integers looking like {1, 2, 3, 4...}, so if you dereference *p1, it would only read the first int, which is 1, and not read the second 4 bytes as a part of the double.

♡ ...



Anonymous Red deer 2y #881afe

✓ Resolved

SU20-MT2-Q5b.ii.A

I am not sure how to calculate the maximum clock frequency without a calculator. Is there a process of going from nanoseconds to MHz without a calculator?

♡ ...

Jero Wang ADMIN 2y #881baa

We will not expect this level of calculations since we do not allow calculators this semester. If we have questions like these, the answer can be calculated by hand.

♡ ...

Anonymous Lark 2y #881afd

✓ Resolved

SP20-Final-Q7 -> B -> iii

The screenshot shows a PDF document titled "Sp20_Final_Solutions.pdf" with page 19 of 24. It contains three questions, each with a code input field:

- A. (1.0 pt) (CODE INPUT 1):
`b->itercount`
- B. (3.0 pt) (CODE INPUT 2):
`b->hash(element, (uint16_t) i) % b->size`
- C. (3.0 pt) (CODE INPUT 3):
`b->data[bitnum >> 3] | (1 << (bitnum & 0x7)) (or equivalent)`

I'm really confused. Why are we doing the right shift by 3 positions? Also, why are we using 0x7?

♡ ...

Jero Wang ADMIN 2y #881bab

We need to set the `bitnum`th bit of `data`, and `data` is stored as an array of 8 bit integers. The right shift 3 allows us to figure out which integer in `data` to modify, and we AND it by `0x7` to figure out which bit within the 8 bit integer to modify.

♡ ...

Anonymous Mink 2y #881aee

✓ Resolved

Sp 20 q3 does the union fields `uint8_t u[4]`, etc create an array of 4 `uint8_t` or is it just breaking u into 4 blocks of 2 bits?

```
union Fun {
    uint8_t u[4];
    int8_t i[4];
    char s[4];
    int t;
};
```

♡ ...

N **Nikhil Kandkur** TA 2y #881aef
uint8_t u[4] creates an array of 4 uint8_t
♡ ...

Anonymous Mink 2y #881afa
so how come part c is -1? isn't i[1] just 0x00000000 after part b and then all of the bits in t are negated and shifted right by 1 so i[1] becomes 0x1fffffff since i[0] was 0x11111110 after b and before the negation and shift?
♡ ...

N **Nikhil Kandkur** TA 2y #881aff
← Replying to Anonymous Mink
I think the author of this question assumed it was an arithmetic shift instead of a logical shift. Also, remember that i[1] is 1 byte, so it would be something like 0xFF instead of 0xFFFFFFFF
♡ ...

Anonymous Chough 2y #881aeb ✓ Resolved
sp20-mt1-q6

How did they reduce: mantissa (1).1111111111 to $2 \cdot 2^{-10}$?

b) You receive the data "0b1110xxxxxxxxxxxx". What is the **decimal value** of the **smallest number** the sender could have sent (i.e. it is less than all of the other possibilities)? You must provide the decimal form, **do not leave as a power of 2**.

-8188

Answer: -8188. By the previous observation, the smallest number is encoded by 0xEFFF. This has sign bit 1, exponent $0b11011 - 15 = 12$, and mantissa $(1).1111111111 = 2 \cdot 2^{-10}$. Our answer is thus $-4096 \cdot (2 \cdot 2^{-10}) = -8192 + 4 = -8188$

♡ ...

Y **Yile Hu** TUTOR 2y #881aec

$0b1.1111111111 = 0b10.0000000000 - 0b0.0000000001 = 2 - 2^{-10}$

♡ ...

Anonymous Chicken 2y #881aea ✓ Resolved

FA20-Final-Q3b: For version 1, I don't get how the next state was found and why for the last row with a current state of 10 and input 1 outputs a 1.

♡ ...

R **Rosalie Fang** ADMIN 2y #881afb

The names of the states are not related to what you have seen so far. To approach this question, I'd think about which states we need to keep track of. Currently since we know we'd only output a 1 if we've seen '111' and we'd restart afterwards, the states will be "fresh start", "seen a 1", "seen 11", we label these 0, 1, 2, that's why "seen 11", the last state, is 10. At this state, if we receive an input of 1, we want to restart our count, so we go back to "fresh start", or 00, and output a one because we've now seen '111'.

♡ 1 ...



Anonymous Chicken 2y #881afc

thank you! so how is the state 2 represented here? I think I am still confused on what the current column is saying. Like if the current state is 01 (row 4) and input is 1, does that mean based on the labeling you said: "fresh start" --> saw a 1 and now since input is 1 we have seen 2 one's so technically we are in state 2? But, how is state 2 reflected in the chart bc 0 is still outputted?

♡ ...



Yile Hu TUTOR 2y #881bac

↩ Replying to Anonymous Chicken

For version 1, we know that we want to output 1 only if we have seen 3 consecutive 1's.

Therefore, we have three states:

00 -> the "fresh start".

01 -> the last input bit is 1.

10 -> the last two input bits are 1.

If the current state is 01, this means the last input is 1. Then, the current input is also 1. Therefore in the new state the last two input bits are 1, and we are in state 10.

If we are in state 10, this means the previous two input bits are both 1, and if the current input is 0, unfortunately we do not have three consecutive 1, so we have to start from the beginning, thus the state transitions into 00, and the output is still 0. On the other hand, if the input is 1, we have three consecutive 1's. We still transition into state 00, but this time outputting 1.

♡ 1 ...



Anonymous Lark 2y #881ade

✓ Resolved

SP20 - MT1 Q 5.B

Isn't it 6? 3 (clk-to-q) + 3 (AND -> NOT)

♡ ...



E Eric Kusnanto TA 2y #881adf

I'm assuming you meant Su20 - MT2 5(b).ii.B, asking for hold time (idk who decided to number these problems) . The shortest path possible is clk-to-q + AND because we don't need to wait for the other input to the AND gate to stabilize, it may have values from the previous cycle, but all that matters for calculating hold time is that some new value has been propagated to a register input after a rising edge.

♡ 1 ...



Anonymous Goldfinch 2y #881adb

✓ Resolved

SU20-MT1-Q6aiii

How did we get this answer for part iii? Thank you

6. Don't Float Away!

Suppose we use an 8-bit floating point format similar to IEEE-754, with 1 sign bit, 3 exponent bits, and 4 significand bits. Assume the bias is -3 and we add the bias. For ALL parts of this question, express your answer a) in decimal, and b) in hex. Make sure you add the prefix to your hex value, fully simplify your answers, and do NOT leave them as fractions. Feel free to plug your fraction into Google to turn it into a decimal value. For all answers, write the exact decimal value, not a rounded one. All solutions have a finite number of decimal digits without rounding!

iii. (1.0 pt)

How many positive non-zero denormalized Floating Point numbers can we represent? (Answer in decimal)

15


$$2^4 - 1 = 15$$

♡ ...

 Erik Yang TA 2y #881adc

Denorm means exponent is all zero and sign must be positive so only thing you can change is the mantissa bits. There are 4 bits so there is 2^4 representations but you can't have mantissa be all 0 since you don't want non zero numbers. That's how you get $2^4 - 1$

♡ 1 ...

 Anonymous Chough 2y #881acd ✓ Resolved

SU20-MT1-Q2

For all the subproblems of part (a), I'm struggling to understand why each multiple choice solution is what it is. I've only ever encountered problems that were more straightforward in discussions/worksheets, and am struggling to understand the nuances here. Would someone be able to explain each of them, or at least some?

♡ ...

 Jero Wang ADMIN 2y #881acf

i. `sentinel` is a local variable on the stack, the address of it is a stack address

ii. `sentinel.next->data` is the struct containing `s`, which is an array on the stack, which has a stack address

iii. `push_back` is a function, therefore code

iv. `sentinel.next` is the struct containing `r`, which is a pointer to a string literal, which is stored in static memory.

v, vi, vii. see https://inst.eecs.berkeley.edu/~cs61c/sp22/pdfs/forum-threads/sp22_mt_past_exam_questions_2020.pdf

♡ 1 ...

 Anonymous Goldfinch 2y #881acb ✓ Resolved

Su20-MT1-Q3c

Hello, how did we get this answer? Also, what is the difference between `auipc` and `lui`? Thank you

(d) i. (1.0 pt)

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

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

`0xABCDDBBC`

♡ ...

 **Jero Wang** ADMIN 2y #881ace

$0xABCDE000 + 0x100 = 0xABCDE100$ for the `auipc`, then $0xABCDE100 + 0xABC = 0xABCDEBBC$ for the `addi`.

`auipc` adds the immediate to the upper 20 bits of the current PC, whereas `lui` adds it to 0.

♡ ...

 **Anonymous Goldfinch** 2y #881ada

Thank you

♡ ...

 **Anonymous Chough** 2y #881add

<code>auipc rd imm</code>	Add Upper Immediate to PC	<code>rd = PC + (imm << 12)</code>
<code>lui rd imm</code>	Load Upper Immediate	<code>rd = imm << 12</code>

How does `lui` not also load the immediate to the upper 20 bits?

♡ ...

 **Anonymous Penguin** 2y #881aed

Is the solution to this problem wrong? I also got `0xABCDEBBC`, but the solution is `0xABCDDBBC`.

♡ 1 ...

 **Anonymous Giraffe** 2y #881bde

↩ Replying to Anonymous Penguin

^^^

♡ ...

 **Anonymous Goldfinch** 2y #881abe ✓ Resolved

SU20-MT1-Q2

How would we free the node that is on the heap instead of freeing the one on the stack? Thank you

2. Doubly Linked Trouble!

For this problem, assume all pointers and integers are **four bytes** and all characters are **one byte**. Consider the following C code (all the necessary `#include` directives are omitted). C structs are properly aligned in memory and all calls to `malloc` succeed. **For all of these questions, assume we are analyzing them right before `main` returns.**

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

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

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

(c) (1.75 pt) Say we had this free function:

```
void free_list(node *n) {
    if (n == NULL) return;
    node *c = n->nxt;
    for (; c != n;){
        node *tmp = c; c = c->nxt;
        free(tmp);
    }
}
```

Given this free function, if we called `free_list(&sentinel)` **after all** the code in `main` is executed, this program would have well defined behavior.

- False
 True

This free function would free the `sentinel` node which is stored on the stack not the heap! This would result in undefined behavior.

♡ ...

↳ E Erik Yang TA 2y #881abf

It seems like free list already frees everything on the heap so you don't need to free anything in main since sentinel is on the stack

♡ ...

↳ Anonymous Goldfinch 2y #881aca

Hi, thanks for the response. In the explanation it says that the free function frees the sentinel node on the heap. Would you mind explaining in which line the free function frees what is on the heap? Thank you

♡ ...

E Erik Yang TA 2y #881acc

↳ Replying to Anonymous Goldfinch

The line `free(tmp)` frees the node that was created in push back which was on the heap

♡ ...

 **Anonymous Hawk** 2y #881bae

↩ Replying to Erik Yang

When does temp ever point to the sentinel node? at the last iteration of the for loop, c is the node just before the sentinel node which gets assigned into temp, then c becomes the sentinel node. Temp gets freed and the loop terminates, which means the current C (sentinel node) is never freed.

♡ ...

E **Erik Yang** TA 2y #881baf

↩ Replying to Anonymous Hawk

Tmp is the current node and then u set c to be the next node. Then you free tmp which is the current node. This process goes on until c is null meaning you do end up freeing everything in the node list

♡ ...

 **Anonymous Hawk** 2y #881bba

↩ Replying to Erik Yang

I just looked at the walkthrough and the TA had the same issue as me. The function loops through the list until it goes back to n (sentinel node), however on the last iteration of the for loop, the loop stops because $c == n$. That means the sentinel node never gets freed which means the function has well-defined behavior.

♡ ...

E **Erik Yang** TA 2y #881bcf

↩ Replying to Anonymous Hawk

you want to free everything except for the sentinel right? Because the sentinel does not exist on the heap, but everything else does because of the malloc call in push_back. That's why if you call free(sentinel) then there is an error because sentinel was only ever initialized on the stack in line 3 of main

♡ ...

 **Anonymous Hawk** 2y #881bda

↩ Replying to Erik Yang

The free_list function as defined when called on &sentinel, never actually calls free on sentinel. I agree that because sentinel hasn't been malloced and isn't on the heap that calling free on sentinel would produce undefined behavior, but my point is that the function never calls free on sentinel. The TA solving the test in the walkthrough ran into the same problem and said that the solution is probably wrong.

♡ ...

E **Erik Yang** TA 2y #881bdc

↩ Replying to Anonymous Hawk

Oh I see what you're saying now. You free until you go back to sentinel and then the for loop ends. I'll bring this up

♡ 1 ...

 **Anonymous Goshawk** 2y #881abc

✓ Resolved

Fa20-MT-Q2B

Why is it $q \rightarrow R = \text{map_tree}(p \rightarrow R)$ instead of $q \rightarrow R = \text{incr_tree}(p \rightarrow R)$?

♡ ...

Erik Yang TA 2y #881abd
If you take a look at the rewritten solns, it's inc_tree
♡ ...

Anonymous Goshawk 2y #881aaf ✓ Resolved
SP20-Final-Q1c

The value is 0xFA000003, with type I. The immediate value is
1111 1010 0000. How does that equal to -96?

(c) (2.0 pt) a RISC-V instruction? If there's an immediate, write it in decimal. If it is an invalid instruction, write INVALID INSTRUCTION (in all caps).

lb x0, -96(x0)

♡ ...

Erik Yang TA 2y #881abb
The number is in 2s complement
♡ ...

Anonymous Goshawk 2y #881aae ✓ Resolved
SP20-Final-Q8

Why is this the case? What does it mean by "static data reference" and "PC-relative addressing"? I thought things that needs to be relocated is when you need to jump to a label address.

(d) CALL me maybe

i. For the following, please indicate if they always or never need to be relocated.

A. (2.0 pt) PC-Relative Addressing

- Never Relocate
 Always Relocate

B. (2.0 pt) Static Data Reference

- Never Relocate
 Always Relocate

♡ ...

Erik Yang TA 2y #881aba
Static data reference is all the data stored in .data that is used in your .text. So a matrix m0 could be stored in .data and is referenced in .text. PC addressing is the offset in which you need to jump by to get to some line of code. Since it is just an imm I think you don't have to relocate it
♡ 1 ...

Anonymous Goshawk 2y #881aaa ✓ Resolved
SP20-Final-Q6

Why is it not an address?

```
void foo() {
    int64_t w = 4;
    int64_t* u = malloc(100 * sizeof(w));
    int64_t** v = &u;
    ...
}
```

vi. (2.0 pt) What type of value does `*(u + 1)` evaluate to?

- Not an Address
- Stack Address
- Static Address
- Heap Address

♡ ...

J Jero Wang ADMIN 2y #881aad

`u + 1` points to the element in the `u` array at index 1, and dereferencing gives the value at that memory address, not an address.

♡ 1 ...

Anonymous Goldfish 2y #881fe ✓ Resolved

SP20- MT1-Q1

a) Correctly gets the number of bytes in a string, including the null-terminator (Mark all that apply)

```
int get_strlen(char* str) {
     return strlen(str);
     return strlen(str) + 1;
     return sizeof(str);
     return sizeof(str) + 1;
     return str[strlen()] + 1;
     None of the above
}
```

Why the third option isn't right?

♡ ...

Y Yile Hu TUTOR 2y #881ff

`str` is a pointer, so `sizeof(str)` will return 4 bytes (assuming 32-bit system), regardless of the length of the string it is pointing to

♡ ...

Anonymous Goldfish 2y #881aab

I see. Thanks. If it is `char[] str`, then we can use `sizeof(str)` to get the length, right?

♡ ...

J Jero Wang ADMIN 2y #881aac

↩ Replying to Anonymous Goldfish

`char[] str` isn't really a type you can have for a function, but if you have the following code, it would work

```
char str[10];
sizeof(char) + 1;
```

♡ ...

Anonymous Giraffe 2y #881fb ✓ Resolved

SP20-MT1-Q5b: why does doing $a-b$ guarantee that the number will be 1 if $a>b$? for example, what if $a=5.2$, $b=3.2$, $a-b=2!=1$?

♡ ...

E Erik Yang TA 2y #881fd

The comparison function should return negative if the first element is less than the second, 0 if they are the same, or positive if the first element is bigger

♡ 1 ...

Anonymous Giraffe 2y #881fa ✓ Resolved

SP20-MT1-Q4c: I'm confused as to why freeing root would lead to undefined behavior. which address on the stack are we freeing?

♡ ...

E Erik Yang TA 2y #881fc

since `root->left` gets set to `&nl` on line 23, this means it lives on the stack, so when you try to call it on `free(root)`, it will execute `free(root->left)`, which will cause an error

♡ ...

Anonymous Giraffe 2y #881ee ✓ Resolved

is sp20 q3 (unions) in scope for the exam?

♡ ...

E Erik Yang TA 2y #881ef

Yes

♡ ...

Anonymous Jaguar 2y #881dc ✓ Resolved

SU 2020-MT1-Q3a

I don't think I quite understand the solution. Since there are two registers in a branch instruction and we are taking off one bit for each register, this means the immediate has 15 bits with an implicit 0 at the 0th bit. Doesn't this mean the lower bound and upper bound is -2^{14} and $2^{14} - 1$?

♡ ...

E Erik Yang TA 2y #881de

Since there's `rs2` and `rs1` for branch, we take off 1 bit for each reg which gives us 2 extra bits to the imm, which is now 14 bits. This means we have bounds of -2^{13} and $2^{13} - 1$

♡ ...

Anonymous Moose 2y #881ea

But why wouldn't the upper bound be $2^{13} - 2$? We didn't have a place to store the last bit right?

♡ ...

J Jero Wang ADMIN 2y #881eb

↩ Replying to Anonymous Moose

Could you elaborate on why $2^{13} - 2$? The implicit bit 0 is already incorporated in this calculation.

♡ ...

Anonymous Moose 2y #881ec
↩ Replying to Jero Wang

I think because we have an implicit 0, all the addresses we can represent must be even
♡ ...

Anonymous Goshawk 2y #881ce ✓ Resolved
SP 2020-MT-Q6b

Why does $1.11\ 1111\ 1111 = 2 \cdot 2^{-10}$? Where does the (2-) part come from?

b) You receive the data "0b110xxxxxxxxx". What is the **decimal value of the smallest number** the sender could have sent (i.e. it is less than all of the other possibilities)? You must provide the decimal form, **do not leave as a power of 2**.

-8188

Answer: -8188. By the previous observation, the smallest number is encoded by 0xEFFF. This has sign bit 1, exponent 0b11011 - 15 = 12, and mantissa (1)111111111 = $2 \cdot 2^{-10}$. Our answer is thus $-4096(2 \cdot 2^{-10}) = -8192 + 4 = -8188$

♡ ...

Rosalie Fang ADMIN 2y #881dd
You can think about it in terms of stride.

If we were to add $0.00\ 0000\ 0001$ to $1.11\ 1111\ 1111$, we would get $10.00\dots0 = 2 \cdot 0.00\ 0000\ 0001 = 2^{-10}$, so $1.11\ 1111\ 1111 = 2 - 2^{-10}$

♡ ...

Anonymous Goshawk 2y #881cd ✓ Resolved
SP 2020-MT-Q6a

Why is it 0x7777 instead of 0x7770? Where did I make a mistake in my work?

Q6) Fixing Point Numbers (9 pts = 3 * 3)

You received a sequence of IEEE standard 16-bit floating point numbers from your friend. So you don't need to look it up on your green sheet, we will remind you that a 16-bit floating point is **1 sign bit, 5 exponent bits, and 10 mantissa bits**. The bias for the exponent is **-15**.

Unfortunately, cosmic rays corrupted some of the data, rendering it unreadable. For the following problems, we will use "x" to refer to a bit that was corrupted (in other words, we don't know what the sender wanted that bit to be). For example, if I received the data "0b0xx1", the sender sent one of "0b0001", "0b0101", "0b0011", or "0b0111".

a) You receive the data "0b0x1x0x1x0x1x0x1x". What is the **hexadecimal encoding of the biggest number** the sender could have sent?

0x _ _ _ _

1. pos
2. $x1x0x \rightarrow 11101 \rightarrow 16+15 = 27$
3. $1x0x1x0x1x \rightarrow 01101101110000$
exp bias
 $27 - 15 = 5 + 9 = 14$
7 7 7 6

♡ ...

Erik Yang TA 2y #881db
if you fill in all the x's with 1's you get: $0b\ 0111\ 0111\ 0111\ 0111 = 0x7777$
♡ ...

Anonymous Goshawk 2y #881cc ✓ Resolved
SP 2020-MT-Q5a

For this question, I'm trying to understand why we are shifting to the left by 3 bytes.

- The character is stored in the most significant byte of the ith integer in info
- If we had FE 00 00 01, then the character is FE. If we did a right shift then the pointer will point to the beginning of 01 right? Then when we cast with a character, it will only read 1 byte because a char reads 1 byte. My question is why does it read the most significant bit (FE) instead of the least significant (01) bit because the pointer currently points to 01.

- Does this have to do with the current system we are in? If it is in little endian then it will read 01, but if its in big endian then will it read FE?

```
(a) Your first clue is a string encoded in an integer array info of length len. We encoded the null-terminated string by placing the ith character in the most significant byte of ith integer in info. Modify the code below so that the original string is properly printed and so that there are no memory leaks or undefined behavior. For this question, len is the size of all the characters needed for a properly formatted string (all the letters and the null terminator). The first step is to allocate the memory for this buffer on the heap, using malloc or calloc. Next, since the information is encoded in the most significant bit, a right shift will move the character into the least significant bit so that a cast to char type will keep the data. Since ints are 4 bytes, the left shift must be by three bytes (24 bits). Finally, the buffer created to print the information must be freed to avoid a memory leak.
void clue1(unsigned int* info, int len) {
    char* info_to_print = malloc (len * sizeof(char));
    for (int i = 0; i < len; i++) {
        info_to_print[i] = (char) info[i]>>24 /* Others possible as well */;
    }
}
```



Erik Yang TA 2y #881da

This one is tricky because it's saying they encoded the null terminated string in the MSB of every integer in `info`. This means that given an array of integers, the string is located at the MSB at every index so that's why we left shift by 24 -> to get the last byte. And print this until the end.



Anonymous Goshawk 2y #881bd ✓ Resolved

SP2020-MT-Q2

I'm having a hard time understanding what exactly an Octal base 8 means. If you have 131 how do you convert it to binary? How does it equal 0b01011001?

How do you convert -29 into base 8?

Decimal	Binary (Two's complement)	Octal (base 8, two's complement)	Hex (two's complement)	Binary (Biased w/ added bias of -127)
-29	We first find the binary representation of 29, which is 0b00011101. We then flip all the bits and add 1, and that gives us 0b11100011	Notice that $8 = 2^3$. Thus, we can group 3 binary digits at a time and represent them as one octal digit. Since 8 is not a multiple of 3, we can add a zero at the beginning of our binary number as this will not change the value. Now 0b 011 100 011 = 343_8 .	We group the binary digits in groups of 4. 0b 1110 0011 = 0xE3	To go from a decimal value to its biased notation, we first subtract the bias from -29, and then represent the resulting number $-29 - (-127) = 98$ in its binary representation 0b01100010.
$1 * 8^0 + 3 * 8^1 + 1 * 8^2 = 89$	We will take each octal digit and replace it with 3 binary digits, this gives us $131_8 = 0b 001 011 001$. Then we take the least significant 8 bits and get 0b01011001	131	We group the binary digits in groups of 4. 0b 0101 1001 = 0x59	We first subtract the bias -127 from 89 and get 216. 216 represented as an unsigned binary number is 0b11011000



Erik Yang TA 2y #881bf

Base 8 is similar to all other bases - think of binary 1101 as $1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$. In octal, you can have up to 8 numbers for each bit. For example, $713 = 7 * 8^2 + 1 * 8^1 + 3 * 8^0$. hexadecimal is jsut base 16.

To convert 131, usually I jsut find the biggest power of 2 that can fit in 131, which in this case is 128. Then find the difference which is 3, and find the biggest power of 2 that can fit in 3, etc...



Anonymous Goshawk 2y #881cb

When converting 131 octal base into binary, why are we able to just convert each digit into binary? $1 = 0001$, $3 = 0011$, $1 = 0001$?

Is this the same concept as hexadecimal, which is in base 16? Where if you have $0xFE = 1111\ 1110$?

How does this work for base 2 then? If I have 34 in base 2 then is that 0011 0100?

This doesn't seem to work for base 2 hmmm but why does this work for base 8 and base 16?

♡ ...

E Erik Yang TA 2y #881cf

↩ Replying to Anonymous Goshawk

To convert 131 base 8 to binary, you split each digit into binary bits of 3, because of $2^3 = 8$. This means that this would be 001 011 001. This is very similar to hex.

34 in base 2 is a little different. You can't really split the digits apart like how we did in base 8 and hex bc base 2 is the baseline with which we work with in binary. So how I think of it is the biggest power of 2 that goes in 34 is 32. $34 - 32 = 2$, and 2 fits into 2. That means this would be 0100010, where you just turn "on" the bits that fit into the final number.

♡ ...



Anonymous Goshawk 2y #881bc

✓ Resolved

SP20-MT-Q1.C

For this question, it's in little indian system. When storing the address of doThis converted into a character. It will store it like

6C 69 76 65

l i v e

Then will the pointer start at the beginning of the "e" which will be dont[0]? I'm just a bit confused by little endian.

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

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

A: evil

Doesn't segfault b/c C treats data as bits to be i
their type and trusts that the programmer's type c
Little-endian means least significant byte (0x65)
address, so the answer is evil instead of live.

♡ ...



E Erik Yang TA 2y #881ca

Since it is little median, you read the most significant byte from right to left.. this means that in memory, doThis will be stored as 0: 65 1: 76 2: 69 3:6C. Each index represents one byte. When you print don't out you get evil

♡ 1 ...

 **Anonymous Goshawk** 2y #881bb ✓ Resolved

SP20-MT-Q1b

For the $*(arr + i)$, how come we don't need to multiply i by 4 because each integer is 4 bytes?

b) Gets the i th element of an array

```
int get_elem(int* arr) {  
     return arr[i];  
     return arr + i;  
     return *(arr + i);  
     return arr.get(i);  
     return *arr + i;  
     None of the above  
}
```

♡ ...

 **Erik Yang** TA 2y #881be

When u do ptr notation, it moves the ptr by one index automatically so you don't have to worry about what each type of the element in the array

♡ 1 ...

 **Anonymous Scorpion** 2y #881ac ✓ Resolved

Fa20-MT-Q1A

Based on the solution, I'm confused on why the mantissa bits are 0b1100... instead of 0b0011... as from $-0b1.001 \cdot 2^2$

First write the number in binary: $-4.75 = -0b0100.11$

Then write the number in scientific notation by moving the binary point so that only the implicit 1 is on the left of the binary point: $-0b1.0011 \times 2^2$

Sign bit: 0b1 (number is negative)

Exponent is 2. Subtract the bias to get $2 - (-1023) = 1025$. Write 1025 in 11-bit unsigned binary to get 0b1000 0000 001

Mantissa bits are 0b1100 0000 0

♡ ...

 **Eric Kusnanto** TA 2y #881ae

You're right, mantissa should be 0011_0000_0, I'll look into it thanks for the catch!

♡ 1 ...

 **Eric Kusnanto** TA 2y #881af

Note the disclaimer at the top that these rewritten solutions haven't been thoroughly verified as a project from Su22, so take the solutions with a grain of salt. Sorry about that :/

♡ 1 ...

 **Anonymous Scorpion** 2y #881aa ✓ Resolved

SP20-MT1-Q3c

I'm not sure how starting with -2 and "negate all the bytes in the union and right shift it by one" gives us 0xfffff81.

♡ ...


 R **Rosalie Fang** ADMIN 2y #881ab

We start with -2 in the first byte of the union. So at the beginning, this is what it looks like in memory:

0x000000FE.

Then we multiplied this value by -1, which, in 2's complement, should negating all the bits than add 1, so we'd get negate -> 0xFFFFF01; add 1 -> 0xFFFFF02, right shifted by 1 should give 0xFFFFF81.

♡ 2 ...

 **Anonymous Jaguar** 2y #881e ✓ Resolved

SP20-MT1-Q5b

For the comp function, we take in two void pointers -- p1 and p2 -- as the parameter. The solution says that we should assign `double a = *((double *) p1)`. Are we casting p1 to a double pointer since p1 is a void pointer?

♡ ...

 E **Eric Kusnanto** TA 2y #881f

Correct, we are casting the void pointers to be a 'double' type pointer (not to be confused with a double pointer `char **`), so that when we dereference p1 and p2, they will be types that can be compared.

♡ ...

 **Anonymous Gull** 2y #881c ✓ Resolved

SP20-MT1-Q4


I'm confused why Q4.4 is static but Q4.5 is stack. In Q4.4, the reasoning seems to be r was declared and initialized as a static string, so it is static, but in 4.5, s was also initialized and declared in a very similar way, so why is it considered stack and not static?

♡ ...

 E **Erik Yang** TA 2y #881d

take a look at the input of each newNode as the argument passed in = what data is and char arrays and char pointers live in different parts of memory

♡ ...

 **Anonymous Gull** 2y #881a ✓ Resolved

SP20-MT1-Q3

I'm pretty confused for this question. In part 3.1, we haven't set `u[0]` to be anything, we've only set `i[0]` to be -1. Shouldn't the value `u[0]` then be garbage or 0? Why is it 255 if we haven't given it a value yet?

♡ ...

 C **Catherine Van Keuren** TA 2y #881b

In this question 'Fun' is a union rather than a struct, meaning that all of its attributes share the same place in memory. So, when we assign `i[0]` to -1, we're assigning the same place in

memory where u[0] would read from. However since i is an array of signed ints, but u is an array of unsigned ints, we would read -1 as unsigned, thus 255.

♡ 1 ...



Anonymous Giraffe 2y #881ed

are unions in scope for the exam?

♡ ...



This comment was deleted



E Eric Kusnanto TA 2y #881ba

12 has a mantissa of 1000_0000_0, so any float with mantissa 0000_0000_0 to 0111_1111_1 and the same exponent and sign will be smaller than 12. This means there are 8 bits that can be freely manipulated.

♡ ...