You are viewing this thread in readonly mode.

[Midterm] Past Exams - 2022 #488

J

Jero Wang STAFF
Last year in Exam – Midterm

1,445
VIEWS

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

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

Semester-Exam-Question Number

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

Anonymous Starling 1y #488eae ✓ Resolved

SU22-MT-Q4.1

Why there is no ***j loop*** in the last line in ***loop***? My understanding is that if the number is odd, then skip that number by skip, and if it is even, add it to sum. But after calculating an even number, how do you go back to the loop to check the next number?

...

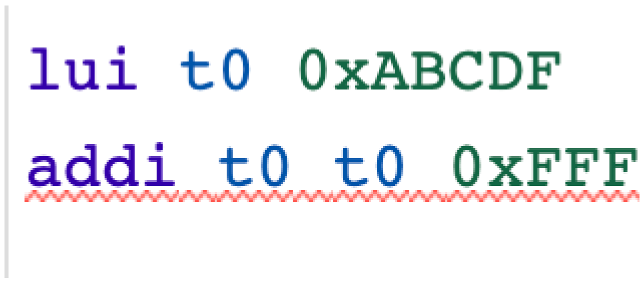
E

Erik Yang STAFF 1y #488eba

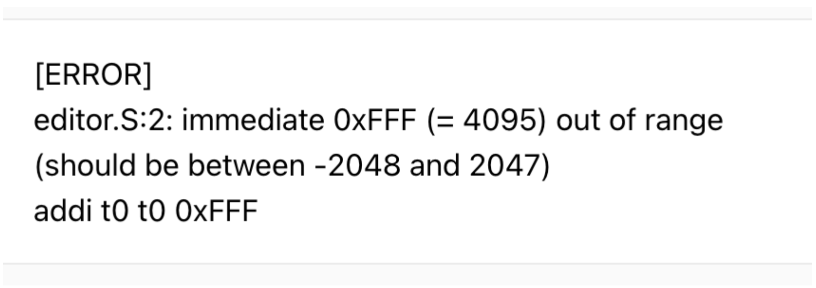
it should be **bne** instead of **beq,** which means it would directly jump to pass

1 ...

Anonymous Alligator 1y #488dfc ✓ Resolved



```
lui t0 0xABCDF
addi t0 t0 0xFFF
```



```
[ERROR]
editor.S:2: immediate 0xFFF (= 4095) out of range
(should be between -2048 and 2047)
addi t0 t0 0xFFF
```

i tried running this command on venus but it seems that it doesn't work because the immediate in addi is getting interpreted as unsigned. is addi unsigned and if i specify only the first 2 bytes (say

0xFF instead of 0xFFF) would addi sign extend?

J Justin Yokota STAFF 1y #488ead

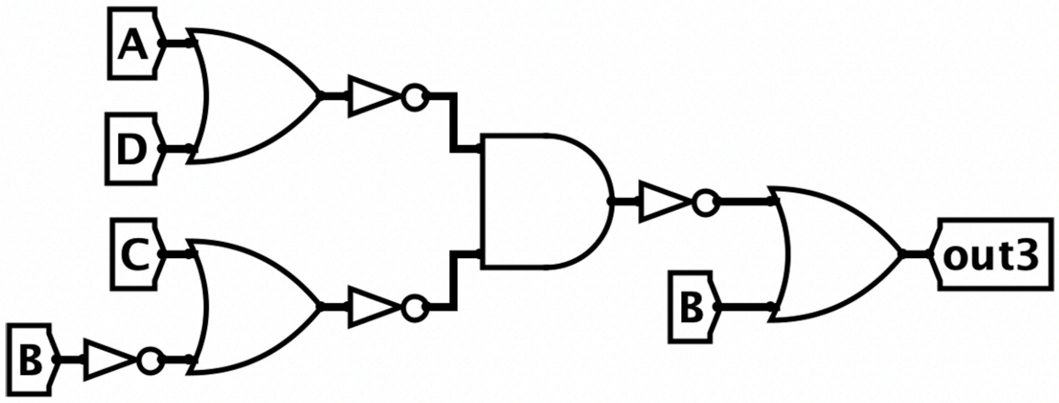
This is an issue with Venus; Venus always interprets hex values as if they were unsigned, even though they shouldn't be. I think it's to prevent programmer mistakes; Venus assumes that if you write 0xFFF, you most likely meant 4095 instead of -1.

Anonymous Rhinoceros 1y #488dea ✓ Resolved

su22-mt-q5.3

I understand how we get A + D + C but how does that become 1?

Q5.3 (4 points)



Solution: ~(~(A + D) * ~(C + ~B)) + B

(A + D) + (C + ~B) + B

A + D + C + (B + ~B)

1

D Darwin Zhang STAFF 1y #488deb

This is Identity Law: X + 1 = 1. Since B + ~B is 1, A + D + C + 1 = 1.

Anonymous Rhinoceros 1y #488dec

Oh shoot ok I see that was slick

1

Anonymous Marten 1y #488dce ✓ Resolved

SP22-MT-Q3.1

Doesn't %d result in printing a signed decimal value so why is it 8?

D Darwin Zhang STAFF 1y #488ddb

We got 8 from a^2, and keep in mind that ^ is the XOR operator.

Anonymous Marten 1y #488dde

but I thought 8 is represented as 1000 in binary and %d prints it as a signed decimal so shouldn't it be negative

D Darwin Zhang STAFF 1y #488ddf

Replying to Anonymous Marten

The entire int should be 32 bits, so 0x00000008 is what we have in the memory.

Anonymous Rhinoceros 1y #488dcb ✓ Resolved

sp22-mt-q3.2

when we do a left bit shift of 4, why do we discard the A in i << 4 that is shifted over four bits. Why aren't we left padding i and then doing the or operation on i and i << 4 to get 0xAE604FFEE

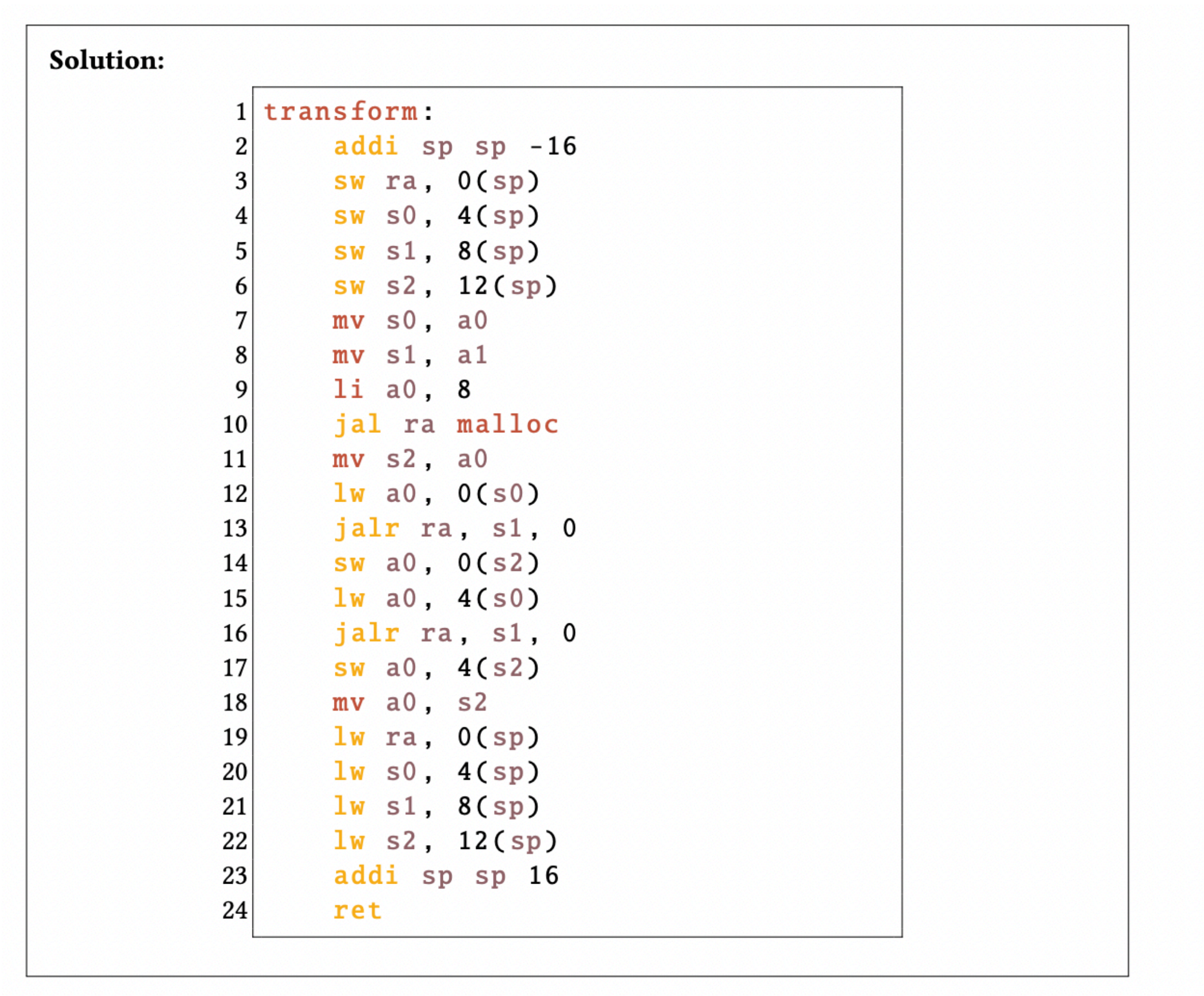
D Darwin Zhang STAFF 1y #488dda

We are doing a logical left shift so the leftmost 4 bits are discarded and the rightmost 4 bits are padded with 0s.

Anonymous Kookabura 1y #488dbd ✓ Resolved

SP22-MT-Q4.2

What is line 17 and 18 doing?



Solution:

```
 1 transform:
 2     addi sp sp -16
 3     sw ra, 0(sp)
 4     sw s0, 4(sp)
 5     sw s1, 8(sp)
 6     sw s2, 12(sp)
 7     mv s0, a0
 8     mv s1, a1
 9     li a0, 8
10     jal ra malloc
11     mv s2, a0
12     lw a0, 0(s0)
13     jalr ra, s1, 0
14     sw a0, 0(s2)
15     lw a0, 4(s0)
16     jalr ra, s1, 0
17     sw a0, 4(s2)
18     mv a0, s2
19     lw ra, 0(sp)
20     lw s0, 4(sp)
21     lw s1, 8(sp)
22     lw s2, 12(sp)
23     addi sp sp 16
24     ret
```

♡ 1 ···

Anonymous Loris 1y #488dbe

Line 17 is storing the new value of a0 into the second slot of memory in s2. Line 18 is setting up the correct output value. Do you happen to know what line 16 is doing?

♡ ···

Eddy Byun STAFF 1y #488dca

Line 16 is jumping to the `f` function, which is stored in our `s0` register

♡ ···

Anonymous Antelope 1y #488eaf

Replying to Eddy Byun

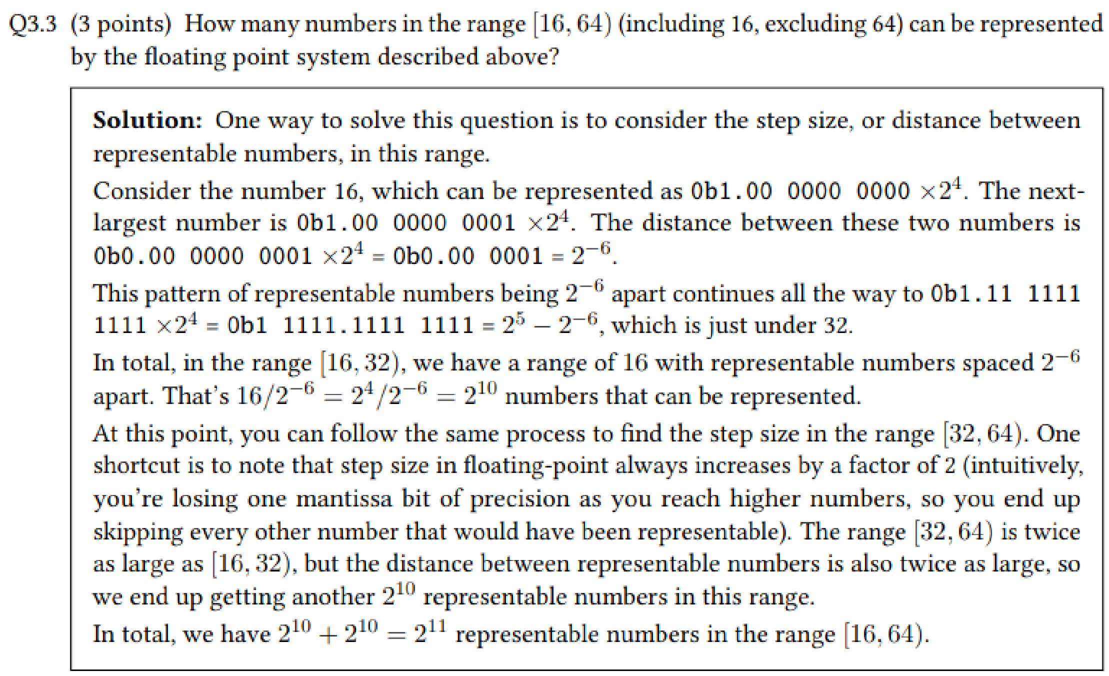
In this question why do you need s2? I understand s0 and s1 are inputs but not seeing where s2 comes in

♡ ···

Anonymous Okapi 1y #488dab ✓ Resolved

SU22-MT-Q3.3

What is the intuition for dividing here our spacing here?



Q3.3 (3 points) How many numbers in the range $[16, 64)$ (including 16, excluding 64) can be represented by the floating point system described above?

Solution: One way to solve this question is to consider the step size, or distance between representable numbers, in this range.

Consider the number 16, which can be represented as `0b1.00 0000 0000` $\times 2^4$. The next-largest number is `0b1.00 0000 0001` $\times 2^4$. The distance between these two numbers is `0b0.00 0000 0001` $\times 2^4$ = `0b0.00 0001` = 2^{-6}.

This pattern of representable numbers being 2^{-6} apart continues all the way to `0b1.11 1111 1111` $\times 2^4$ = `0b1 1111.1111 1111` = $2^5 - 2^{-6}$, which is just under 32.

In total, in the range $[16, 32)$, we have a range of 16 with representable numbers spaced 2^{-6} apart. That's $16/2^{-6} = 2^4/2^{-6} = 2^{10}$ numbers that can be represented.

At this point, you can follow the same process to find the step size in the range $[32, 64)$. One shortcut is to note that step size in floating-point always increases by a factor of 2 (intuitively, you're losing one mantissa bit of precision as you reach higher numbers, so you end up skipping every other number that would have been representable). The range $[32, 64)$ is twice as large as $[16, 32)$, but the distance between representable numbers is also twice as large, so we end up getting another 2^{10} representable numbers in this range.

In total, we have $2^{10} + 2^{10} = 2^{11}$ representable numbers in the range $[16, 64)$.

I understand the range we are working with, and how the step size goes to 2, but I am lost on why we divided by 2^(-6)?

When we do this division, are we asking how many times can 2^(-6) go into 2^4?

1 ···

Anonymous Antelope 1y #488dcc

i am also confused with this same question. How did you find the representation of 16 to be 1. 000... ? Wouldn't this make it negative? I also don;t understand why this is being split up into 2 ranges

···

Anonymous Okapi 1y #488dcf

The only thing I can think of is:

$$16 \ = \ 2^4 == 10000.000000$$

$$Exponent \ = 10010 => 2^{19-15=4}$$

$$32 = 2^5 \ = \ 100000.00000 \ (slli \ of \ 1 := multiplied \ by \ 2)$$

$$2^{-6} = .000001$$

$$add \ 2^{-6} \ to \ 10000.00000 \ until \ 10000.111111$$

$$rolls \ ove \ r \ to \ 10001.000000 \ \left(now \ 17, \ and \ still \ < 2^5\right)$$

$$Continue \ until \ 11111.111111$$

$$11111.111111 \ + \ 2^{-6} = 2^5 \ and \ (slli \ of \ 1 \ or \ \cdot 2)$$

Now we are in 32 to 64 but stepping by 2 however the range is doubled as well, so we still arrive at the same answer as the first sum.

I believe the crux of the argument, and the confusing part is 2^{-6} being multiplied by 2^4.

I think viewing this as a counting problem is significantly more intuitive than a floating point problem.

we have 16 different representations of the first 5 values because the very first digit is always occupied by 1.

Thus:

$$1xxxx\ example:\ 10100\ or\ 11100\ or\ 10111$$

$$2^4\ 1xxxx's$$

Looking at the places behind the decimal: .000000
we have 2^6 possible values. So there are $2^4 * 2^6$ possible values for [16, 32).

Repeat for 32 to 64

Andrew Liu STAFF 1y #488ded

The idea is that the size of the range divided by the size between numbers in the range gives you the numbers in that range. I.e. up to 1 decimal point, between 0 and 10, there are $\frac{10}{10^{-1}} = 100$ numbers (0.0, 0.1, 0.2, ... 9.9)

1

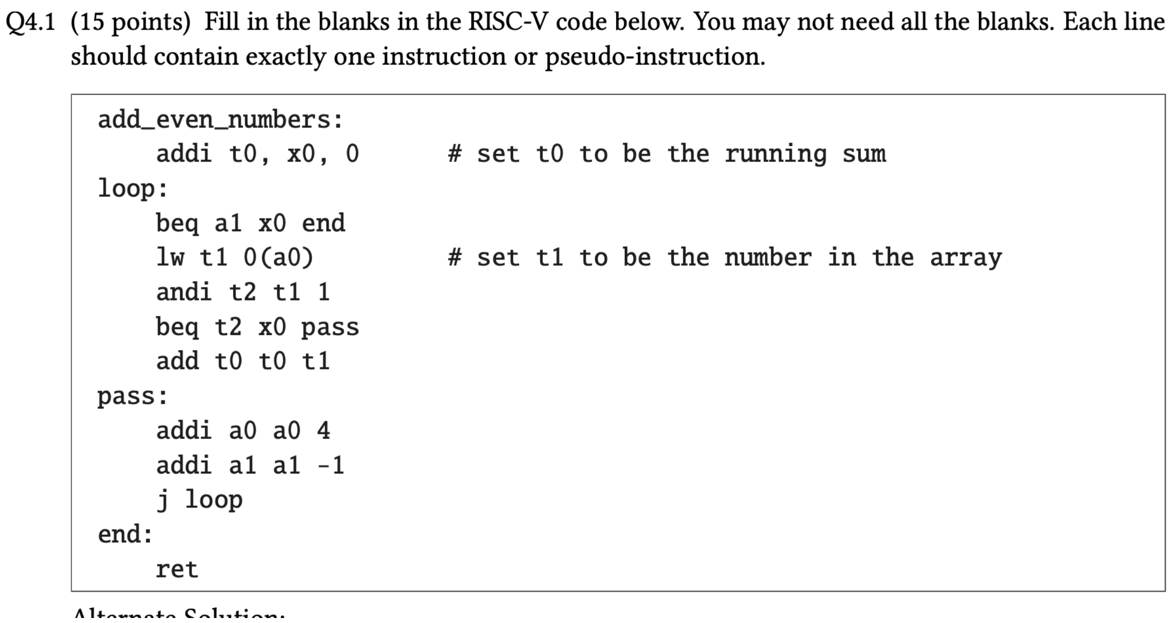
Anonymous Okapi 1y #488dee

Ah sweet, didn't know that.

Anonymous Shark 1y #488daa ✓ Resolved

SU22-MT-Q4.1

I think when we do andi t2,t1,1: when t2 is 0 it means it's even and we want to add it to t0 and when t2 is 1 it means it's an odd number and we don't want to add it to sum, but here when it's equal to zero it go to pass and don't add the even number, why is that?



Q4.1 (15 points) Fill in the blanks in the RISC-V code below. You may not need all the blanks. Each line should contain exactly one instruction or pseudo-instruction.

```
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
```

Alternate Solution:

Eddy Byun STAFF 1y #488dbb

#488a

♡ 1 ···

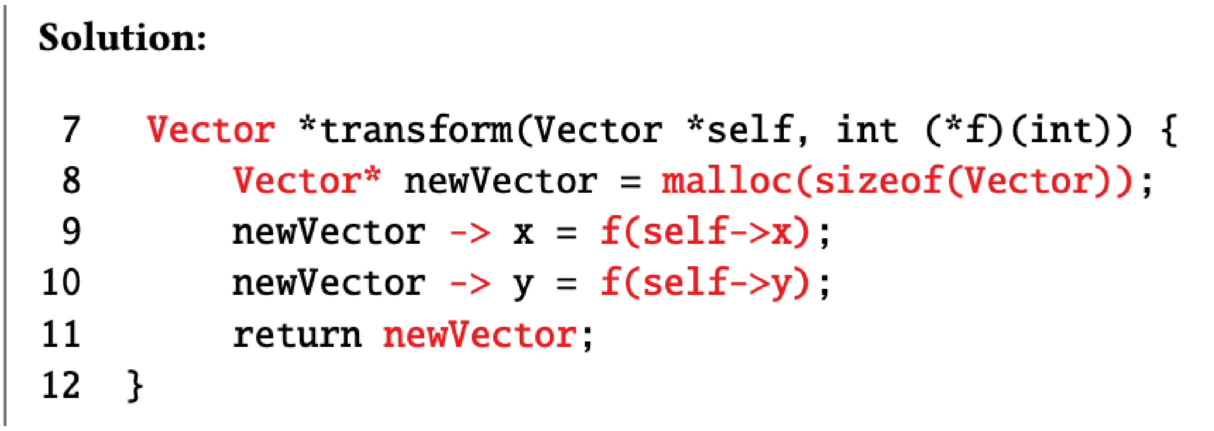
Jiries Kaileh 1y #488eaa

This solution also doesn't seem to assign the final sum to a0

♡ ···

Anonymous Quelea 1y #488cff ✓ Resolved

SP22-MT-Q4.1



Solution:

```
Vector *transform(Vector *self, int (*f)(int)) {
    Vector* newVector = malloc(sizeof(Vector));
    newVector -> x = f(self->x);
    newVector -> y = f(self->y);
    return newVector;
}
```

The function pointer was directly used here as "f(arguments)". But I thought for pointers, we need to deference them first, like "(*f)(arguments)" *for example "(*f)(self->x)"*

♡ ···

E Eddy Byun STAFF 1y #488dae

#488ab

♡ ···

Anonymous Cormorant 1y #488cfe ✓ Resolved

SP22-MT-Q4.2

Why do we li a0 8 (where does the 8 come from). I thought it would be 4 since it's a pointer

♡ ···

E Eddy Byun STAFF 1y #488daf

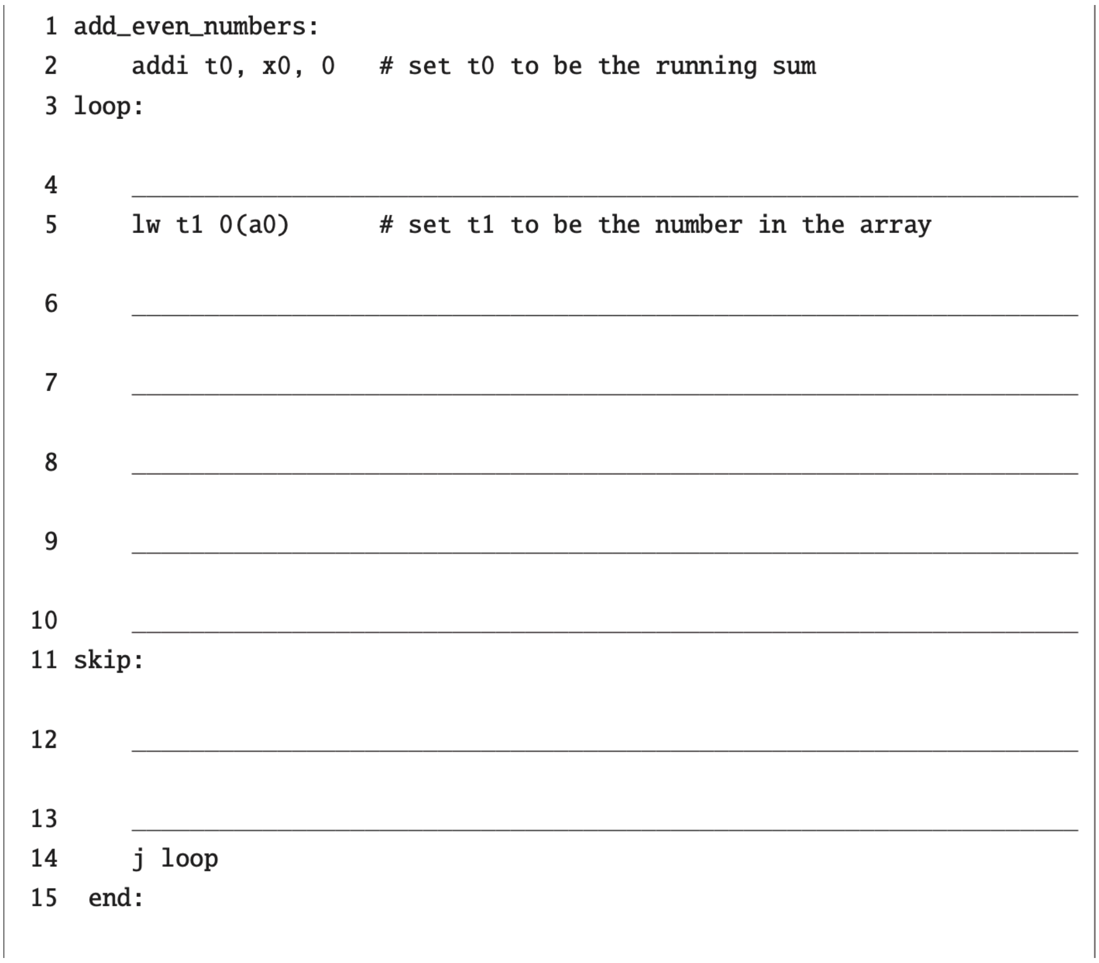
`Vector` is 8 bytes long - 4 bytes for `x` and 4 bytes for `y`

♡ ···

Anonymous Sardine 1y #488cfc ✓ Resolved

SU22-MT-Q4

How do I calculate the offset between lines? Do label names (like "skip") not count as a line? I am wondering how to get -36 from j loop to loop.



```
add_even_numbers:
    addi t0, x0, 0   # set t0 to be the running sum
loop:

    ________________________________________________________
    lw t1 0(a0)      # set t1 to be the number in the array

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________
skip:

    ________________________________________________________

    ________________________________________________________
    j loop
  end:
```

Eddy Byun STAFF 1y #488dad

#488cec and #488adf

Anonymous Sardine 1y #488dbc

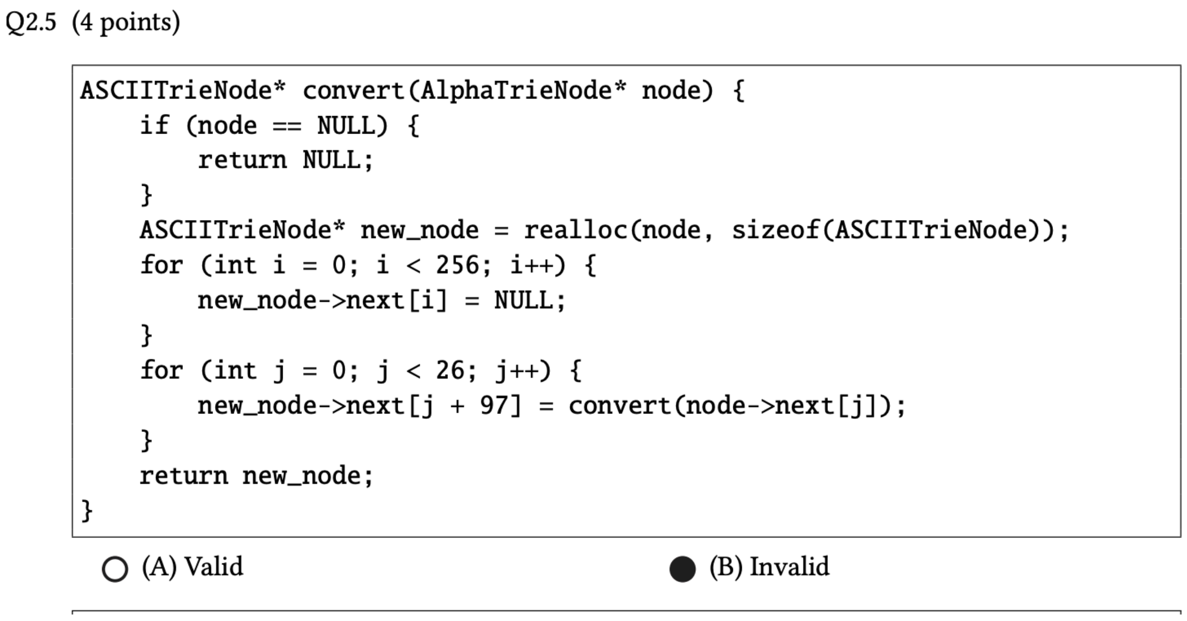
is the offset positive or negative if it is jumping to a line before the current PC?

1

Anonymous Shark 1y #488cfb ✓ Resolved

SU22-MT-Q2

in this part if in the second loop we had i=26;i<256;i++ and we didn't have the first loop at all and we would assume it had the pointer at node and just expanded the memory, would that work?


Q2.5 (4 points)

```
ASCIITrieNode* convert(AlphaTrieNode* node) {
    if (node == NULL) {
        return NULL;
    }
    ASCIITrieNode* new_node = realloc(node, sizeof(ASCIITrieNode));
    for (int i = 0; i < 256; i++) {
        new_node->next[i] = NULL;
    }
    for (int j = 0; j < 26; j++) {
        new_node->next[j + 97] = convert(node->next[j]);
    }
    return new_node;
}
```

(A) Valid [✓ (B) Invalid]

N

Noah Yin STAFF 1y #488dfd

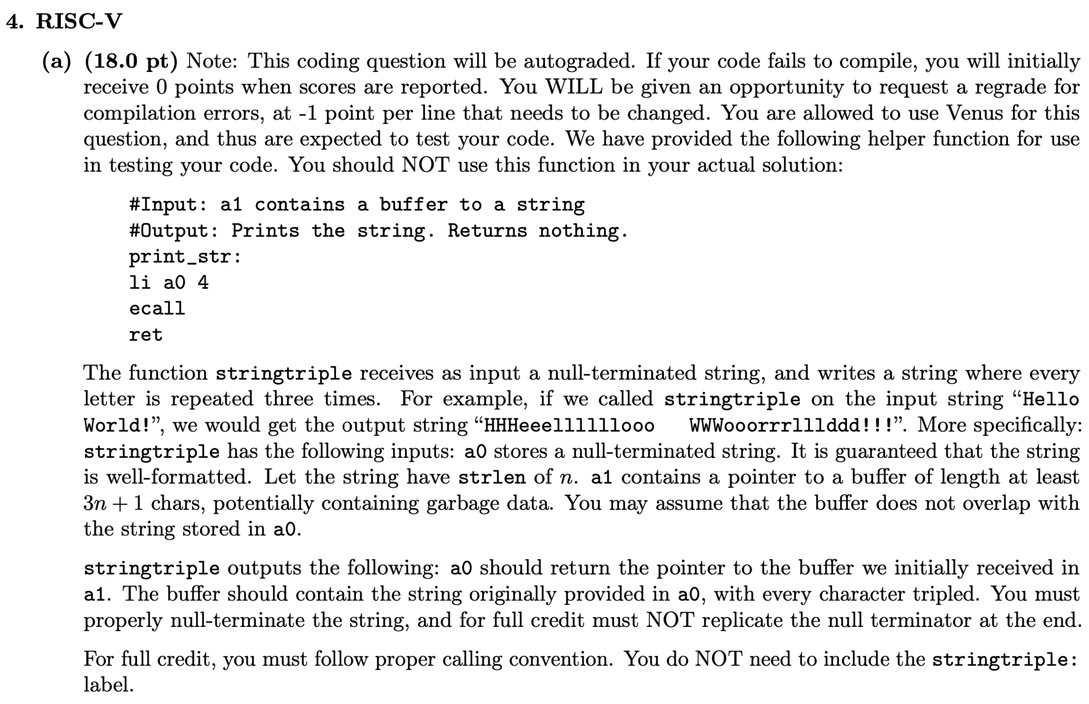
This would not work, since the new_node->next[i + 97] would access the 123rd to 352nd elements, of which all above 255 are out of bounds.

Assuming that it still had the pointer at node, you would still need to initialize the memory from the index 26 to 255 elements as Realloc does not auto initialize its memory.

edit: the old pointer node->next[i] would be able to access the index 26 to 255 elements.

Anonymous Loris 1y #488cfa ✓ Resolved

Spring 2022 MT 2:


4. RISC-V

(a) **(18.0 pt)** Note: This coding question will be autograded. If your code fails to compile, you will initially receive 0 points when scores are reported. You WILL be given an opportunity to request a regrade for compilation errors, at -1 point per line that needs to be changed. You are allowed to use Venus for this question, and thus are expected to test your code. We have provided the following helper function for use in testing your code. You should NOT use this function in your actual solution:

```
#Input: a1 contains a buffer to a string
#Output: Prints the string. Returns nothing.
print_str:
li a0 4
ecall
ret
```

The function `stringtriple` receives as input a null-terminated string, and writes a string where every letter is repeated three times. For example, if we called `stringtriple` on the input string "`Hello World!`", we would get the output string "`HHHeeellllllooo   WWWooorrrlllddd!!!`". More specifically: `stringtriple` has the following inputs: `a0` stores a null-terminated string. It is guaranteed that the string is well-formatted. Let the string have `strlen` of n. `a1` contains a pointer to a buffer of length at least $3n + 1$ chars, potentially containing garbage data. You may assume that the buffer does not overlap with the string stored in `a0`.

`stringtriple` outputs the following: `a0` should return the pointer to the buffer we initially received in `a1`. The buffer should contain the string originally provided in `a0`, with every character tripled. You must properly null-terminate the string, and for full credit must NOT replicate the null terminator at the end.

For full credit, you must follow proper calling convention. You do NOT need to include the `stringtriple:` label.

stringtriple:

```
stringtriple:
            mv t2 a1
    Loop:
            lbu t0 0(a0)
            beq t0 x0 End
            slli t1 t0 8
            add t1 t1 t0
            sh t1 0(a1)
            sb t0 2(a1)
            addi a0 a0 1
            addi a1 a1 3
            j Loop
    End:
            sb x0 0(a1)
            mv a0 t2
            jr ra
```

why are we multiplying the value of t1 by 16? (slli t1 t0 8) . Also I don't understand what sh and sb are doing visually.

Andrew Liu STAFF 1y #488def

A character is 1 byte, or 8 bits. For sake of example, let's use `'A' = 0x65`. What this code does then is this:

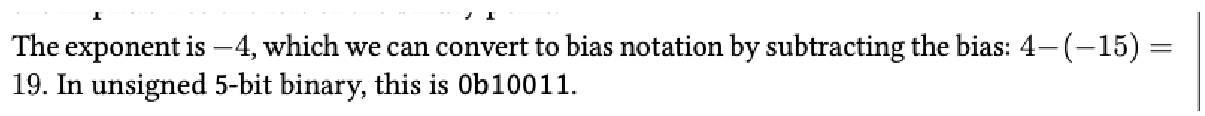
```
lbu t0 0 (a0)       # stores 0x65 in t0

t0: 0x00 00 00 65   # Note: this represents 'A'


beq t0 x0 End       # check for null terminators

t0: 0x00 00 00 65


slli t1 t0 8        # shift t0 by 1 byte, and store in t1.

t0: 0x00 00 00 65
t1: 0x00 00 65 00   # Note: this kinda (not really) represents "A\0"


add t1 t0 8        # shift t0 by 1 byte, and store in t1.

t0: 0x00 00 00 65
t1: 0x00 00 65 65 # Note: now this represents "AA"
```

Now, when you `sh`, you're storing `"AA"`, and then when you `sb`, you're storing `"A"`, which triples each character of the string.

Anonymous Sardine 1y #488cef ✓ Resolved

SU22-MT-Q3:

Shouldn't this be -4 + 15, not +4? I was wondering what the right convention is, I feel like it changes every test?



The exponent is -4, which we can convert to bias notation by subtracting the bias: $4-(-15) = 19$. In unsigned 5-bit binary, this is 0b10011.

♡ ···

E Eddy Byun STAFF 1y #488dac

Yes, sorry the -4 is a typo

♡ 1 ···

Anonymous Fox 1y #488cea ✓ Resolved

Hi,

For **SU22-MT-Q4.2,** do labels count as lines that are accounted for by the PC? I noticed the offset provided is -36 so 9 lines, but looking at the first sample solution, I only count 9 lines from j loop if I also consider the labels to be lines.

♡ ···

Anonymous Dugong 1y #488ceb ENDORSED

Correct me if i'm wrong, but I didn't count labels and that's how I got 9.

♡ 1 ···

E Eddy Byun STAFF 1y #488cec

Yea, what Anon Dugong said. Labels are not instructions so we don't count them.

If it's easier to think about it this way, you can write RISC-V code like this:

```
label_1: addi t0 x0 1
addi a0 x0 2
addi t0 t0 3
label_2: addi t1 x0 54
xor t2 t0 t1
```

where the instruction and the label are on the same line

♡ ···

Anonymous Fox 1y #488cfd

Got it, thank you both. For this question, when the solutions say the offset is 9 lines away from the j loop instruction, are we defining the PC to be pointing at the line right after j loop (so we would have to move PC up 9 lines for it to point to the `loop: beq a1, x0, end` line)?

If so, is this because the PC points to the line it is about to execute but hasn't yet? So since j loop is currently being executed, the PC is pointing to the line right after j loop just before it's moved 9 lines back, and moving the PC 9 lines back causes it to point to the beq a1 x0 end line, meaning it is about to, but hasn't executed the beq instruction yet?

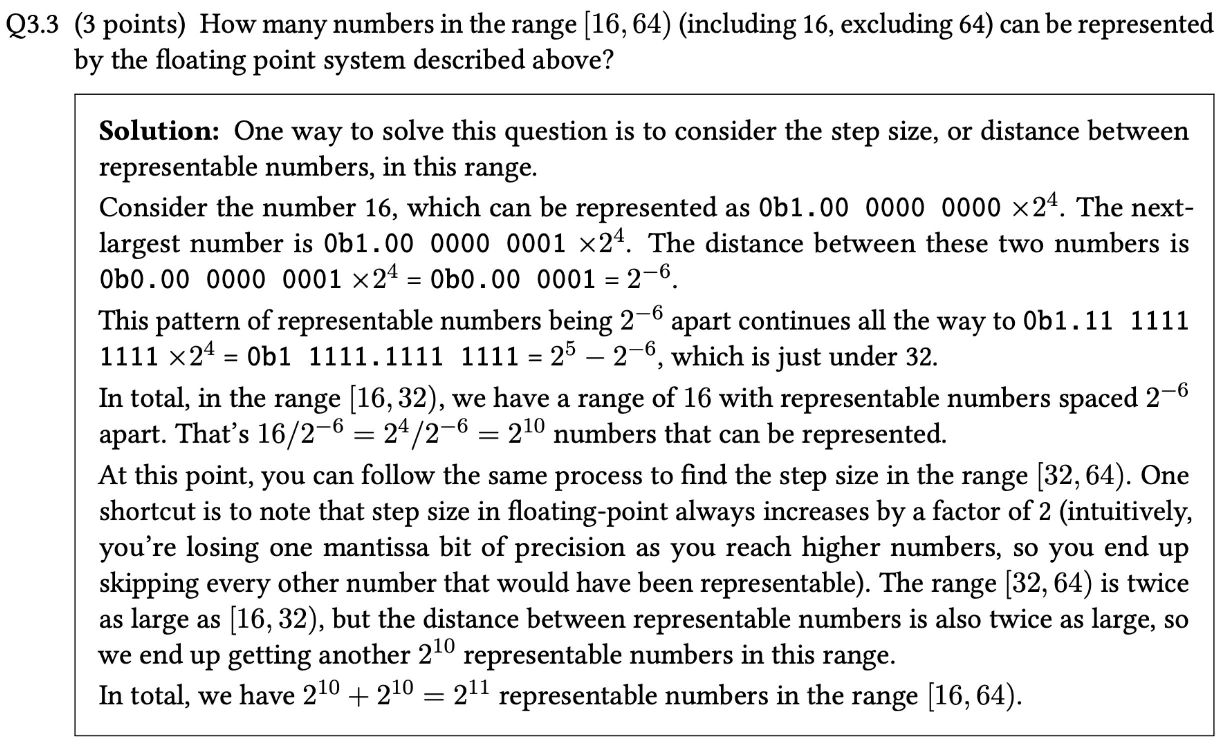
(Also, as an aside, are commas required in RISC-V?)

Anonymous Dugong 1y #488cab ✓ Resolved

SU22-MT-Q3.4

1. How do we build intuition around this?

2. I understand calculating 16, but I would never think until when the pattern would continue. I don't understand how they got this: "This pattern of representable numbers being 2^−6 apart continues all the way to 0b1.11 1111 1111 ×2^4 = 0b1 1111.1111 1111 = ^5 − 2^−6 , which is just under 32."



Q3.3 (3 points) How many numbers in the range $[16, 64)$ (including 16, excluding 64) can be represented by the floating point system described above?

Solution: One way to solve this question is to consider the step size, or distance between representable numbers, in this range.

Consider the number 16, which can be represented as `0b1.00 0000 0000` $\times 2^4$. The next-largest number is `0b1.00 0000 0001` $\times 2^4$. The distance between these two numbers is `0b0.00 0000 0001` $\times 2^4$ = `0b0.00 0001` = 2^{-6}.

This pattern of representable numbers being 2^{-6} apart continues all the way to `0b1.11 1111 1111` $\times 2^4$ = `0b1 1111.1111 1111` = $2^5 - 2^{-6}$, which is just under 32.

In total, in the range $[16, 32)$, we have a range of 16 with representable numbers spaced 2^{-6} apart. That's $16/2^{-6} = 2^4/2^{-6} = 2^{10}$ numbers that can be represented.

At this point, you can follow the same process to find the step size in the range $[32, 64)$. One shortcut is to note that step size in floating-point always increases by a factor of 2 (intuitively, you're losing one mantissa bit of precision as you reach higher numbers, so you end up skipping every other number that would have been representable). The range $[32, 64)$ is twice as large as $[16, 32)$, but the distance between representable numbers is also twice as large, so we end up getting another 2^{10} representable numbers in this range.

In total, we have $2^{10} + 2^{10} = 2^{11}$ representable numbers in the range $[16, 64)$.

Andrew Liu STAFF 1y #488ccf

1) I would say that the intuition for this comes from working with floating point more and become more comfortable with it. Everyone learns at their own rate, and it's ok if you don't feel super comfortable with it yet

2) So you have that 16 is representable and that $16 + 2^{-6}$ is the next smallest representable number, and they correspond to

```
16        = 0b 0 10011 0000000000
16 + 2^-6 = 0b 0 10011 0000000001
...
32 - 2^-6 = 0b 0 10011 1111111111
```

(One way to think about this is that the mantissa works kinda like a scaled and translated version of unsigned numbers, so the smallest mantissa is `00...00` and the largest is `11...11`)

What I mean by the above is that the mantissa is in essence scaled (multiplied) by the exponent, and translated (added) by the exponent as well.

To count the numbers in this range, you count over all possible arrangements of bits in the last 10 bits, which is 2^{10}. Then, you repeat again, starting at 32 and ending at $64 - 2^{-5}$ (do this yourself for practice!)

Anonymous Dugong 1y #488cdd

Thank you so much for this explanation! I think I'm still confused on how we know the step size is 2^-6 until 32? Doesn't step size change?
How do we know it will change after 32?

Also, shouldn't 32 - 2^-6 be:

```
0b 0 11111 1111111111
```

1

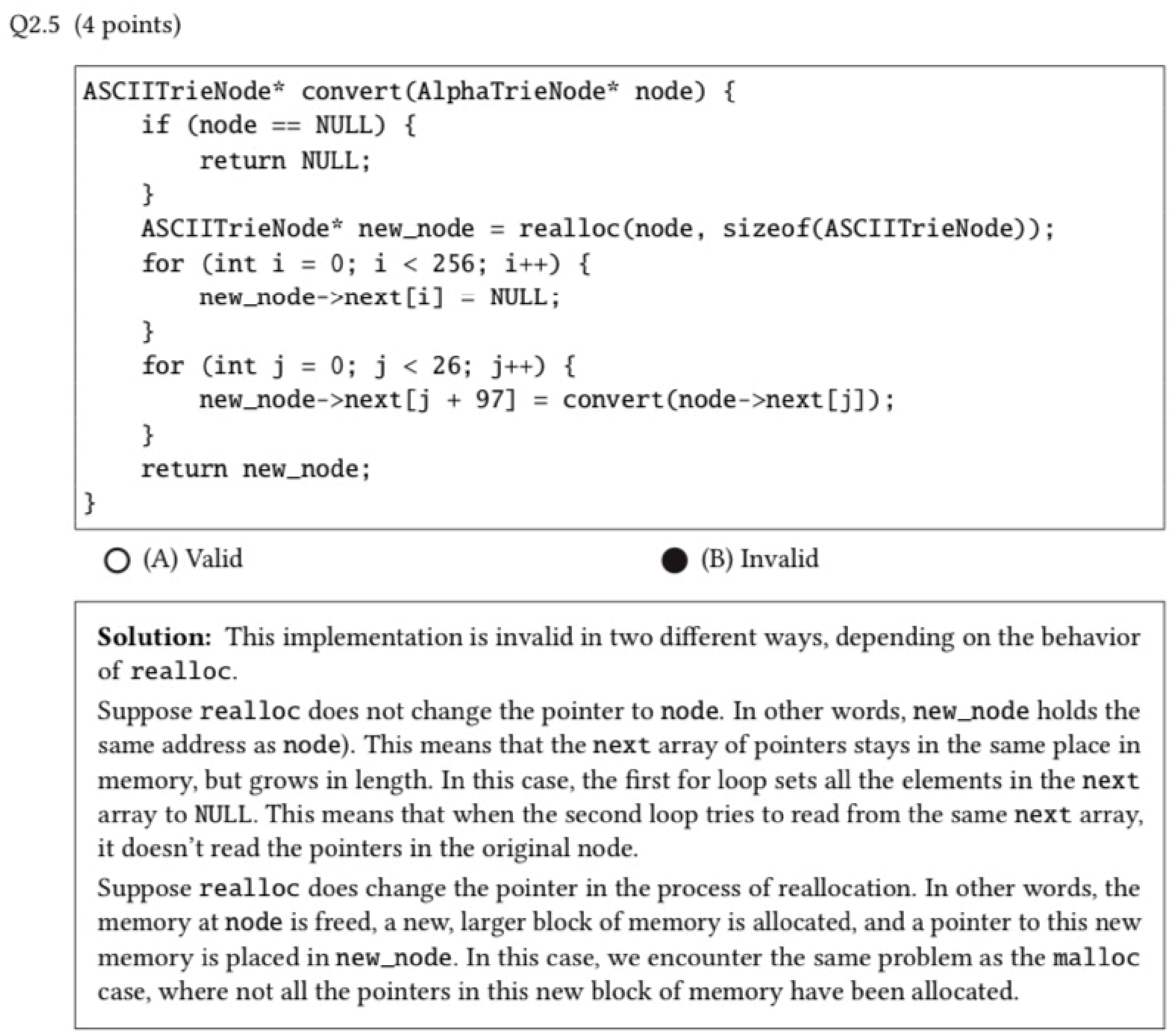
Andrew Liu STAFF 1y #488dfa

Replying to Anonymous Dugong

That number is a `NaN`. Step size in normal floating point numbers is always `value of last bit in mantissa = 2^-#num of mantissa bits * 2^exponent`, and since the number of mantissa bits doesn't change, the step size changes only when the exponent changes.

Anonymous Mole 1y #488beb ✓ Resolved

SU22-MT-Q2.5
For the second case, would trying to access freed memory also cause an error?


Q2.5 (4 points)

```
ASCIITrieNode* convert(AlphaTrieNode* node) {
    if (node == NULL) {
        return NULL;
    }
    ASCIITrieNode* new_node = realloc(node, sizeof(ASCIITrieNode));
    for (int i = 0; i < 256; i++) {
        new_node->next[i] = NULL;
    }
    for (int j = 0; j < 26; j++) {
        new_node->next[j + 97] = convert(node->next[j]);
    }
    return new_node;
}
```

(A) Valid [✓ (B) Invalid]

Solution: This implementation is invalid in two different ways, depending on the behavior of `realloc`.

Suppose `realloc` does not change the pointer to `node`. In other words, `new_node` holds the same address as `node`). This means that the `next` array of pointers stays in the same place in memory, but grows in length. In this case, the first for loop sets all the elements in the `next` array to NULL. This means that when the second loop tries to read from the same `next` array, it doesn't read the pointers in the original node.

Suppose `realloc` does change the pointer in the process of reallocation. In other words, the memory at `node` is freed, a new, larger block of memory is allocated, and a pointer to this new memory is placed in `new_node`. In this case, we encounter the same problem as the `malloc` case, where not all the pointers in this new block of memory have been allocated.

Sam Xu STAFF 1y #488bef

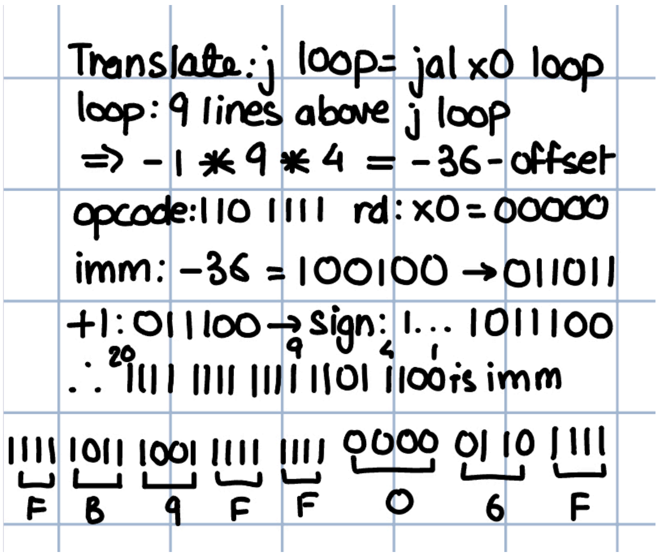
Yes, I agree.

Anonymous Dugong 1y #488bdf ✓ Resolved

SU22-MT-Q4.2:

Here is my working. I get FB9FF06F and the answer is FDDFF06F.

I am not sure what I do wrong/why I get this wrong. I know my immediate is wrong, but I checked the value of -36 online (using 2's comp.) so I know I calculated it correctly. I also am confident that I followed the j-type instruction pattern correctly. So i'm not sure what I did wrong. **Any help would be greatly appreciated :)**



Translate: j loop = jal x0 loop
loop: 9 lines above j loop
$\Rightarrow -1 * 9 * 4 = -36$ - offset
opcode: 110 1111 rd: x0 = 00000
imm: -36 = 100100 $\rightarrow$ 011011
+1: 011100 $\rightarrow$ sign: 1... 1011100
$\therefore$ 20 9 4 1
1111 1111 1111 1101 1100 is imm

$\underbrace{1111}_{F}\ \underbrace{1011}_{B}\ \underbrace{1001}_{9}\ \underbrace{1111}_{F}\ \underbrace{1111}_{F}\ \underbrace{0000}_{0}\ \underbrace{0110}_{6}\ \underbrace{1111}_{F}$

♡ ···

E Eddy Byun STAFF 1y #488cae

I walked through it here: #488cad - let me know if you have any questions!

♡ ···

Anonymous Fox 1y #488bdd ✓ Resolved

Hi,

For **SU22-MT1-Q4.3**, one of the correct choices is to save some values in s registers instead of t registers. However, the function also makes use of a registers to hold the array and the number of elements in the array. Don't these also need to be saved in s registers since is_prime could change them?

♡ ···

Anonymous Dugong 1y #488bea

I think the reason we don't choose that is because the option is "Save used a registers onto the stack at the beginning of the function." This won't work because we need to save a registers every time in the loop and not just at the start of the function.

♡ ···

Anonymous Fox 1y #488bec

Thanks! I was thinking the same thing. I was actually wondering why what you said isn't an answer choice, since I thought you would need to save the a registers before calling is_prime in the loop, and the answer choice I referenced in my question seemed to pertain just to t registers.

♡ 1 ···

Andrew Liu STAFF 1y #488ccd

Yeah, if "Save some values in s registers instead of a registers" was an answer choice, it would be correct, but in this case, since the answer choice isn't there, it is not a correct answer choice (or even an answer choice at all)

♡ ···

Anonymous Louse 1y #488bdb ✓ Resolved

SP22-MT-Q3

is sp22 Q3 define function in scope?

Eddy Byun STAFF 1y #488cbf

Yes, it's in scope

Taeyoung Kim 1y #488bcf ✓ Resolved

SP22-MT-Q4.2

The solution uses saved registers, but would it be fine if we used temporary registers so we don't have to deal with calling conventions?

Eddy Byun STAFF 1y #488cba

No, because we would then need to save the temporary registers on the stack before calling functions and load them back in after calling the functions and we don't have enough space to do that for each `jal` or `jalr` instruction

Jinjian Liu 1y #488bce ✓ Resolved

SP22-MT-Q3.6

#define f(a,b) a*b/4 is tested for the special case when a is passed in a "ptr+". I don't think this is in scope this sem right?

Eddy Byun STAFF 1y #488cbe

It's in scope

Anonymous Fox 1y #488bcb ✓ Resolved

Hi,

For **SU22-MT1-Q1.6**, would the loader be the correct choice here if it was an option?

Eddy Byun STAFF 1y #488cbb

Yes

Elana Ho 1y #488bca ✓ Resolved

Solution:

```
 1 transform:
 2     addi sp sp -16
 3     sw ra, 0(sp)
 4     sw s0, 4(sp)
 5     sw s1, 8(sp)
 6     sw s2, 12(sp)
 7     mv s0, a0
 8     mv s1, a1
 9     li a0, 8
10     jal ra malloc
11     mv s2, a0
12     lw a0, 0(s0)
13     jalr ra, s1, 0
14     sw a0, 0(s2)
15     lw a0, 4(s0)
16     jalr ra, s1, 0
17     sw a0, 4(s2)
18     mv a0, s2
19     lw ra, 0(sp)
20     lw s0, 4(sp)
21     lw s1, 8(sp)
22     lw s2, 12(sp)
23     addi sp sp 16
24     ret
```

spring 2022 midterm Q4.3 on line 10, why is it `jal ra malloc` instead of `jal malloc`? what do lines 13 and 16 do?

♡ 1 ···

R Raiyan Rizwan 1y #488bed

Also curious about jal malloc vs jal ra malloc. For lines 13 and 16, I believe they are jumping to the function stored at address s1, which is *f* (original argument a1), in order to compute f(self.x) and f(self.y).

♡ ···

S Sam Xu STAFF 1y #488bfb

`jal malloc` is the same as `jal ra malloc`.

`jalr ra s1 0` means jump to the instruction at address `s0+0`, and make ra = next line of code. I think the 61C reference sheet is very helpful to understand the instructions.

♡ 1 ···

Anonymous Gnat 1y #488bbf ✓ Resolved

SU22-MT-Q4.2

Can the answer please be explained? Not sure how to arrive to the hexadecimal solution

♡ 1 ···

Eddy Byun STAFF 1y #488cad

Our immediate is going to be -36 (detailed explanation here: #488adf)

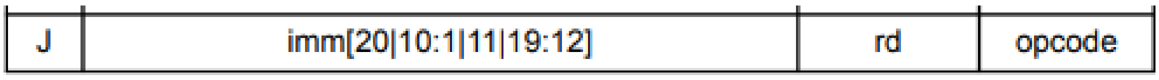
Remember that `j label` is a pseudoinstruction for `jal x0 label`

Opcode: 0b110 1111

Immediate: -36 = 1 1111 1111 1111 1101 1100

rd = x0 = 00000

Put it all together!



J	imm[20|10:1|11|19:12]	rd	opcode

1 1111101110 1 11111111 00000 1101111

1111 1101 1101 1111 1111 0000 0110 1111

= 0xFDDFF06F

1 ···

Anonymous Peafowl 1y #488aee ✓ Resolved

SP22-MT-Q4

I'm a bit confused about all the sw, lw, and mv operations and how that translates to the C code. What would be the general approach to this problem?

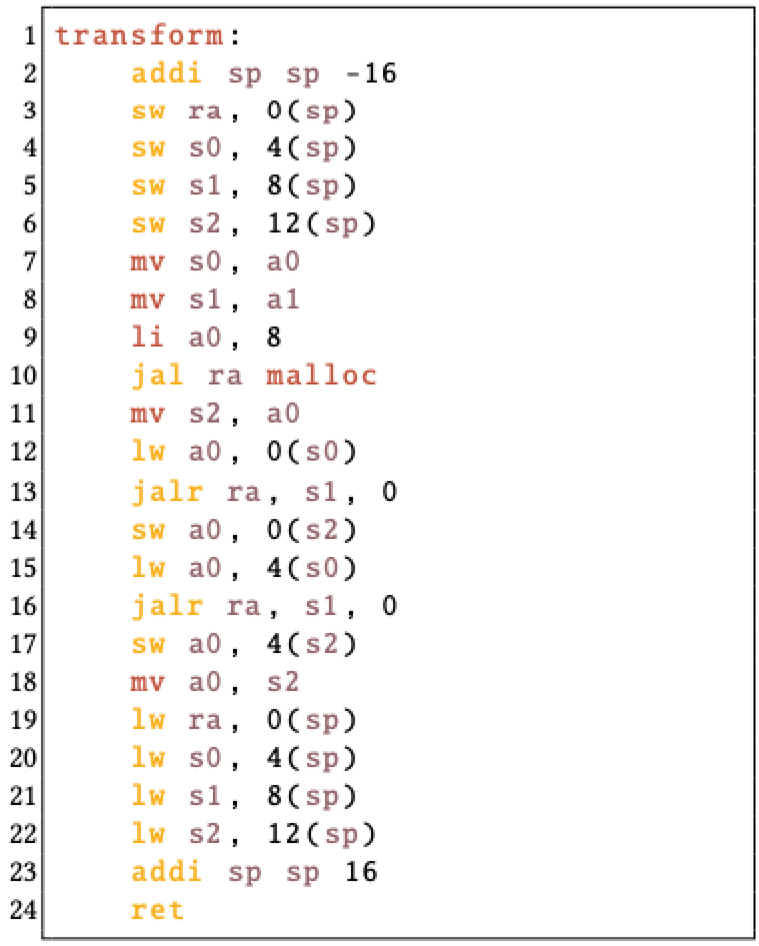
```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def transform(self, f):
        return Vector(f(self.x), f(self.y))
```

Q4.1 (20 points) We want to translate this code to C. Fill in the following C code. Assume all allocations succeed. For full credit, your solution must use the minimum amount of memory required.

```c
1   #include <stdlib.h>
2
3   typedef struct Vector {
4       int x;
5       int y;
6   } Vector;

7   ____________________ *transform(Vector *self, int (*f)(int)) {

8       ________________ newVector = ______________________________;

9       newVector ______ x = _________________________;

10      newVector ______ y = _________________________;

11      return _________________________;
12  }
```

Solution:

```c
7   Vector *transform(Vector *self, int (*f)(int)) {
8       Vector* newVector = malloc(sizeof(Vector));
9       newVector -> x = f(self->x);
10      newVector -> y = f(self->y);
11      return newVector;
12  }
```



```
1  transform:
2      addi sp sp -16
3      sw ra, 0(sp)
4      sw s0, 4(sp)
5      sw s1, 8(sp)
6      sw s2, 12(sp)
7      mv s0, a0
8      mv s1, a1
9      li a0, 8
10     jal ra malloc
11     mv s2, a0
12     lw a0, 0(s0)
13     jalr ra, s1, 0
14     sw a0, 0(s2)
15     lw a0, 4(s0)
16     jalr ra, s1, 0
17     sw a0, 4(s2)
18     mv a0, s2
19     lw ra, 0(sp)
20     lw s0, 4(sp)
21     lw s1, 8(sp)
22     lw s2, 12(sp)
23     addi sp sp 16
24     ret
```

Eddy Byun STAFF 1y #488cbd

This was some general advice I gave for translating from C to RISC-V: #488bee

`lw` is used in this case to load `x` and `y` from `self` into `a0` before calling `f`

`sw` is used to store the result of `f` into `s2`, which is `newVector`

We used `mv` to move `self` to `s0`, `f` into `s1`, and `newVector` into `s2`

Anonymous Peafowl 1y #488cdf

On which line do we call the f function in this case?

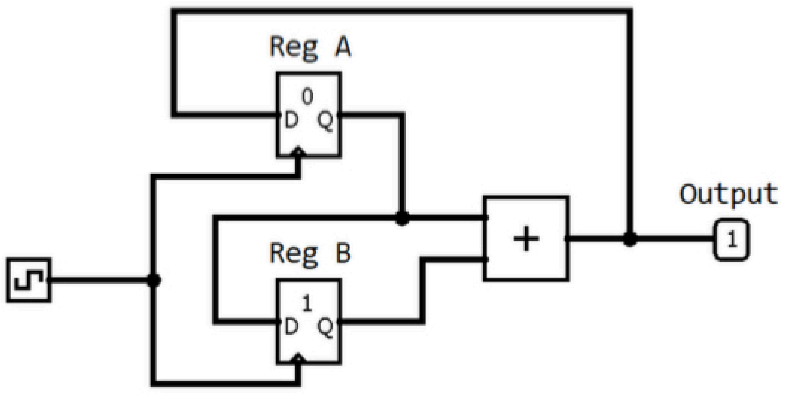
Eddy Byun STAFF 1y #488ced

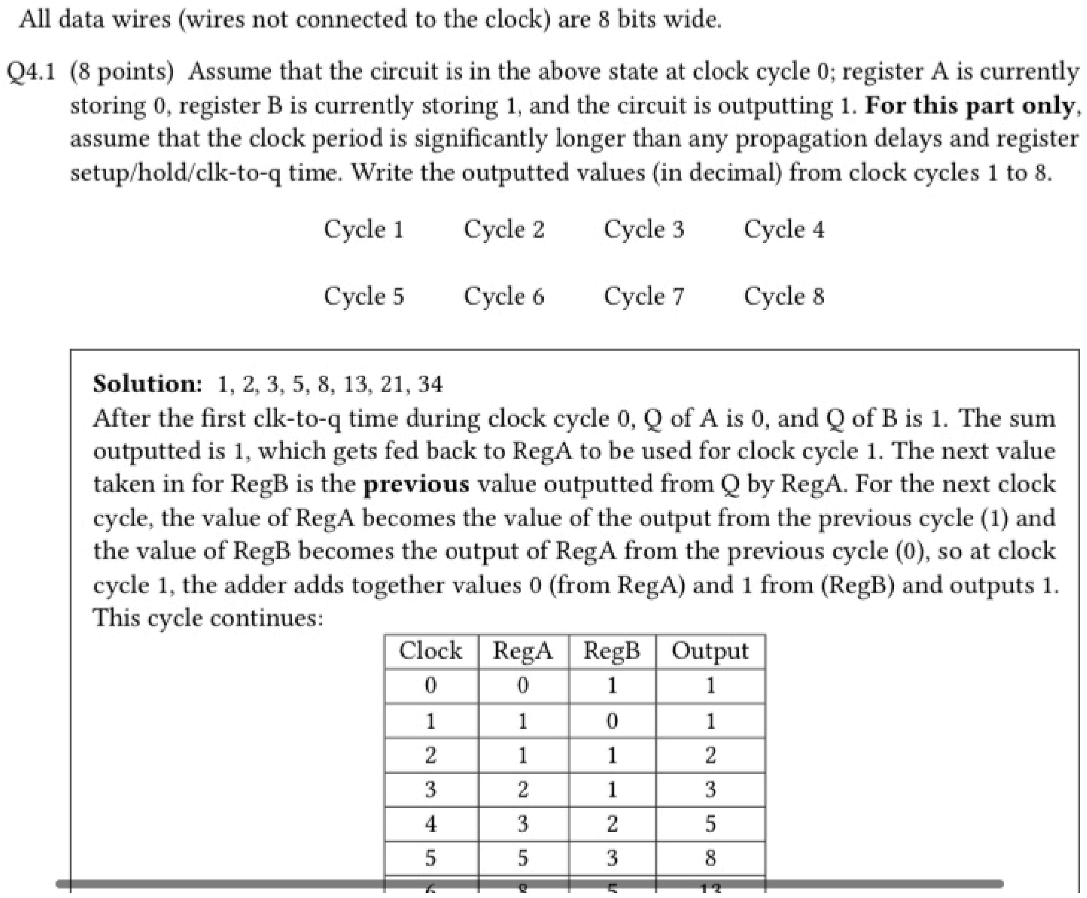
Replying to Anonymous Peafowl

Lines 13 and 16 - note that we moved `f` from `a1` to `s1`

Anonymous Mole 1y #488aeb ✓ Resolved

For SP22-MT-Q5, when the clock cycle changes, are Reg A and Reg B both simultaneously changing to their input values before their output changes? Because if A's output changes first, then the signal going into B would change.





All data wires (wires not connected to the clock) are 8 bits wide.

Q4.1 (8 points) Assume that the circuit is in the above state at clock cycle 0; register A is currently storing 0, register B is currently storing 1, and the circuit is outputting 1. **For this part only**, assume that the clock period is significantly longer than any propagation delays and register setup/hold/clk-to-q time. Write the outputted values (in decimal) from clock cycles 1 to 8.

Cycle 1	Cycle 2	Cycle 3	Cycle 4
Cycle 5	Cycle 6	Cycle 7	Cycle 8

Solution: 1, 2, 3, 5, 8, 13, 21, 34
After the first clk-to-q time during clock cycle 0, Q of A is 0, and Q of B is 1. The sum outputted is 1, which gets fed back to RegA to be used for clock cycle 1. The next value taken in for RegB is the **previous** value outputted from Q by RegA. For the next clock cycle, the value of RegA becomes the value of the output from the previous cycle (1) and the value of RegB becomes the output of RegA from the previous cycle (0), so at clock cycle 1, the adder adds together values 0 (from RegA) and 1 from (RegB) and outputs 1. This cycle continues:

Clock	RegA	RegB	Output
0	0	1	1
1	1	0	1
2	1	1	2
3	2	1	3
4	3	2	5
5	5	3	8

E

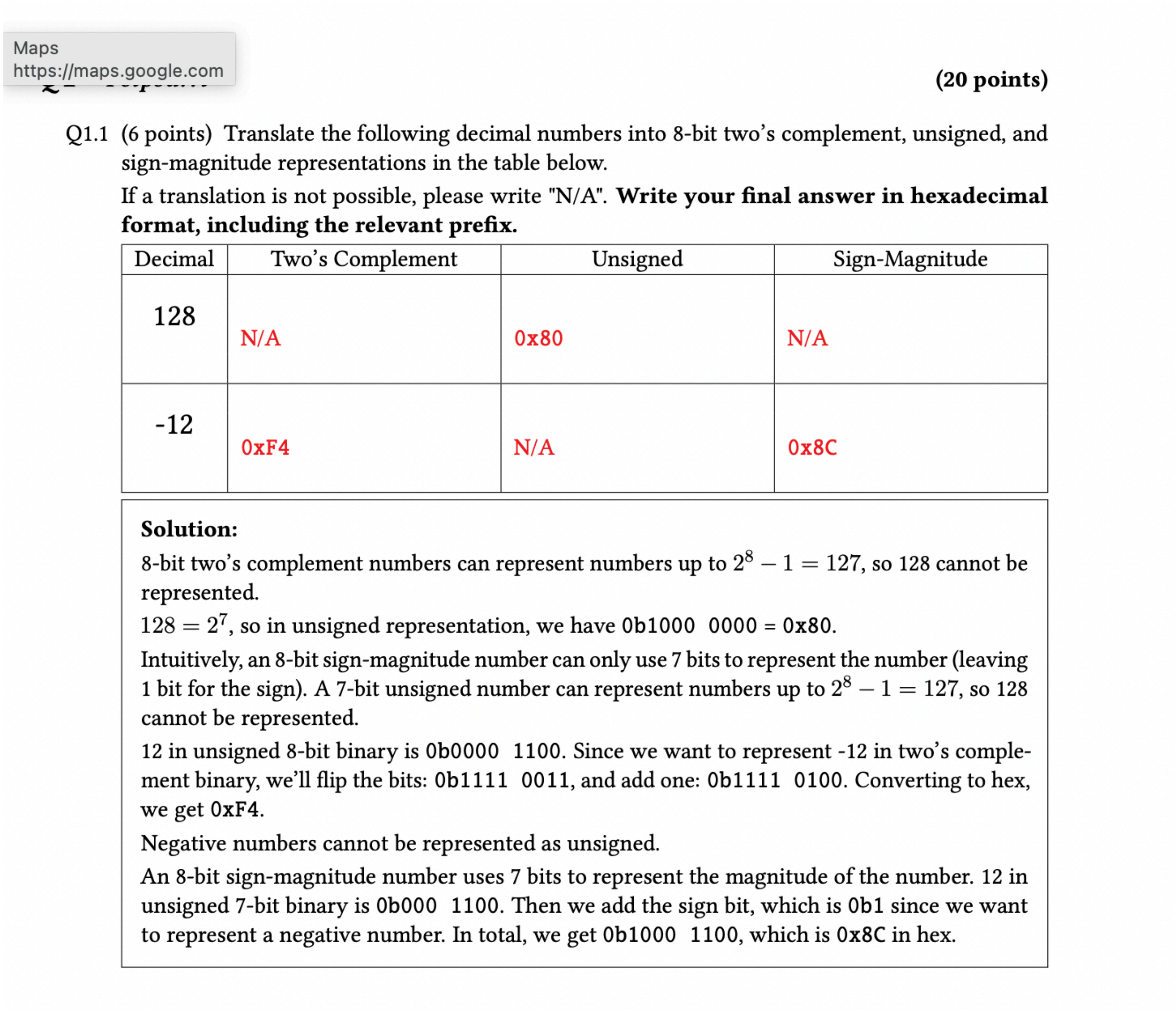
Eddy Byun STAFF 1y #488cac

Yes, both A and B are connected to the same clock, which means that on the rising edge of the clock, both the D input of registers A and B work their way to their respective Q outputs

Anonymous Kookabura 1y #488aea ✓ Resolved

SU22-MT-Q1.1

Why is twos complement numbers only represented up until 2^n -1?



Maps
https://maps.google.com

(20 points)

Q1.1 (6 points) Translate the following decimal numbers into 8-bit two's complement, unsigned, and sign-magnitude representations in the table below.
If a translation is not possible, please write "N/A". **Write your final answer in hexadecimal format, including the relevant prefix.**

Decimal	Two's Complement	Unsigned	Sign-Magnitude
128	N/A	`0x80`	N/A
-12	`0xF4`	N/A	`0x8C`

Solution:

8-bit two's complement numbers can represent numbers up to $2^8 - 1 = 127$, so 128 cannot be represented.

$128 = 2^7$, so in unsigned representation, we have `0b1000 0000` = `0x80`.

Intuitively, an 8-bit sign-magnitude number can only use 7 bits to represent the number (leaving 1 bit for the sign). A 7-bit unsigned number can represent numbers up to $2^8 - 1 = 127$, so 128 cannot be represented.

12 in unsigned 8-bit binary is `0b0000 1100`. Since we want to represent -12 in two's complement binary, we'll flip the bits: `0b1111 0011`, and add one: `0b1111 0100`. Converting to hex, we get `0xF4`.

Negative numbers cannot be represented as unsigned.

An 8-bit sign-magnitude number uses 7 bits to represent the magnitude of the number. 12 in unsigned 7-bit binary is `0b000 1100`. Then we add the sign bit, which is `0b1` since we want to represent a negative number. In total, we get `0b1000 1100`, which is `0x8C` in hex.

♡ 1 ···

S Sam Xu STAFF 1y #488bfc

Sorry the solution has typo, the correct answer is 2^7 - 1.

For 8-bit 2-complement integer, the largest number you can have is `0b01111111` , which is 2^0 + 2^1 + 2^2 + .. + 2^6. If you plus one to it, you will have `0b10000000 (unsigned)` , which is 2^7. Therefore. `0b01111111` = 2^7 - 1.

♡ ···

Anonymous Kookabura 1y #488cdc

I'm not conceptually understanding, wouldn't 0b10000000 be signed because the MSB is a 1? Also I don't understand why twos complement can't go up to 128 and can only go up to one less than the number? Is is always the case that a twos complement number can only go up to one less than the number?

♡ 1 ···

Anonymous Reindeer 1y #488adf ✓ Resolved

SU-MT-Q4.2

How do you get from j loop to identifying there is an immediate = -36 (don't quite understand usage of immediate as jal is different than jalr). How do we get the final instruction before converting to Hex?

♡ ···

E Eddy Byun STAFF 1y #488baf

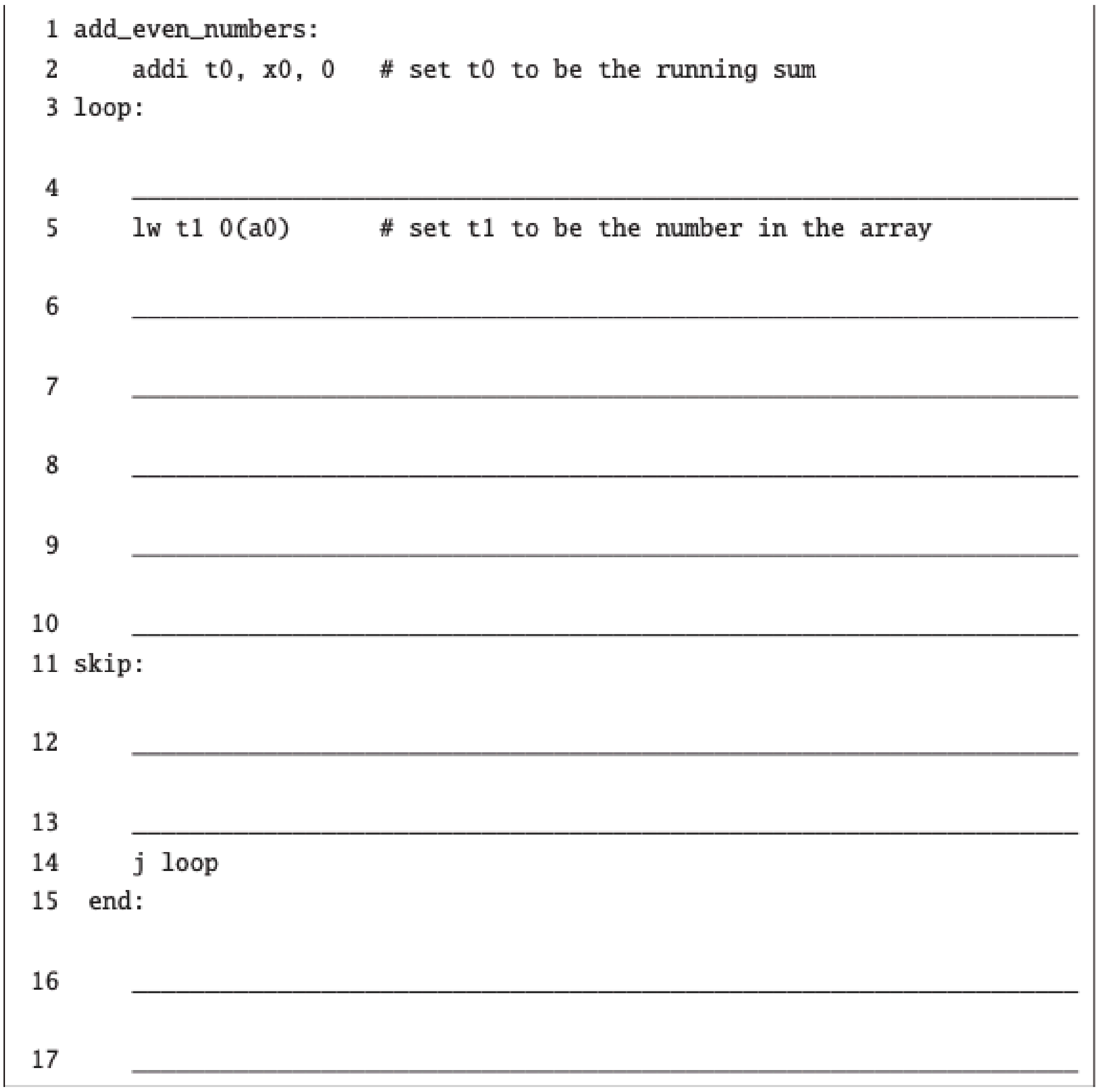
#488ea

Anonymous Reindeer 1y #488bba

I looked at that thread but am still stuck as it only shows how to get the -36 but but not what the instruction is and steps & reasoning to get there.

E Eddy Byun STAFF 1y #488bfa

Replying to Anonymous Reindeer



```
 1 add_even_numbers:
 2     addi t0, x0, 0    # set t0 to be the running sum
 3 loop:

 4     ____________________________________________________________
 5     lw t1 0(a0)       # set t1 to be the number in the array

 6     ____________________________________________________________

 7     ____________________________________________________________

 8     ____________________________________________________________

 9     ____________________________________________________________

10     ____________________________________________________________
11 skip:

12     ____________________________________________________________

13     ____________________________________________________________
14     j loop
15   end:

16     ____________________________________________________________

17     ____________________________________________________________
```

The problem assumes that we've filled in an instruction at each blank. How far away is the instruction that is at the `loop` label (line 4) from `j loop`? Remember that labels aren't instructions! We see that those are going to be instructions at lines 13,12,10,9,8,7,6,5,and 4, which are 9 instructions. Since we have to go backwards 9 instructions and each instruction is 4 bytes, we do 4*-9 = -36 to get our immediate.

Anonymous Antelope 1y #488ddd

Replying to Eddy Byun

what do you mean by backwards 9 instructions? is it because all the calls requiring loop are above it? is that why this I'm is -36 and not 36?

T Taeyoung Kim 1y #488ade ✓ Resolved

SP22-MT-Q2.6: The solution has two different answers, the top one says FF44 6793 but the bottom one with the explanation says FF44 6F93, I'm guessing the bottom one is the correct one, so is it just a typo?

♡ 1 ···

E Eddy Byun STAFF 1y #488bad

Yea, this is a typo. The correct answer is `0xFF446F93` . Sorry for the confusion!

♡ ···

R Robert Yang 1y #488ebb

I also noticed this and was confused

♡ 1 ···

Anonymous Fish 1y #488add ✓ Resolved

SP22-MT-Q4.2

What's the difference between jalr and jal. Why do we need to use jal when calling the inputted function but not malloc?

♡ ···

E Eddy Byun STAFF 1y #488bab

`jalr rd rs1 imm`

`jalr` sets the PC to `rs1 + imm` and sets the `rd` to `PC + 4` - in other words, go to the instruction at instruction `rs1 + imm`

`jal rd label` is going set the PC to `PC + offset` and set the `rd` to `PC` + 4.

We use `jalr` for calling the inputted function because we're told that the `f` is in `a1` , so we can use `jalr` to call `f`

We use `jal` for calling `malloc` because we've defined `malloc` in some external library and the linker will resolve the address of `malloc` for us.

♡ ···

Anonymous Wren 1y #488bda

Also, in this question, shouldn't there be a need to free(s2) once s2 is assigned to a0?

♡ ···

E Eddy Byun STAFF 1y #488ccb

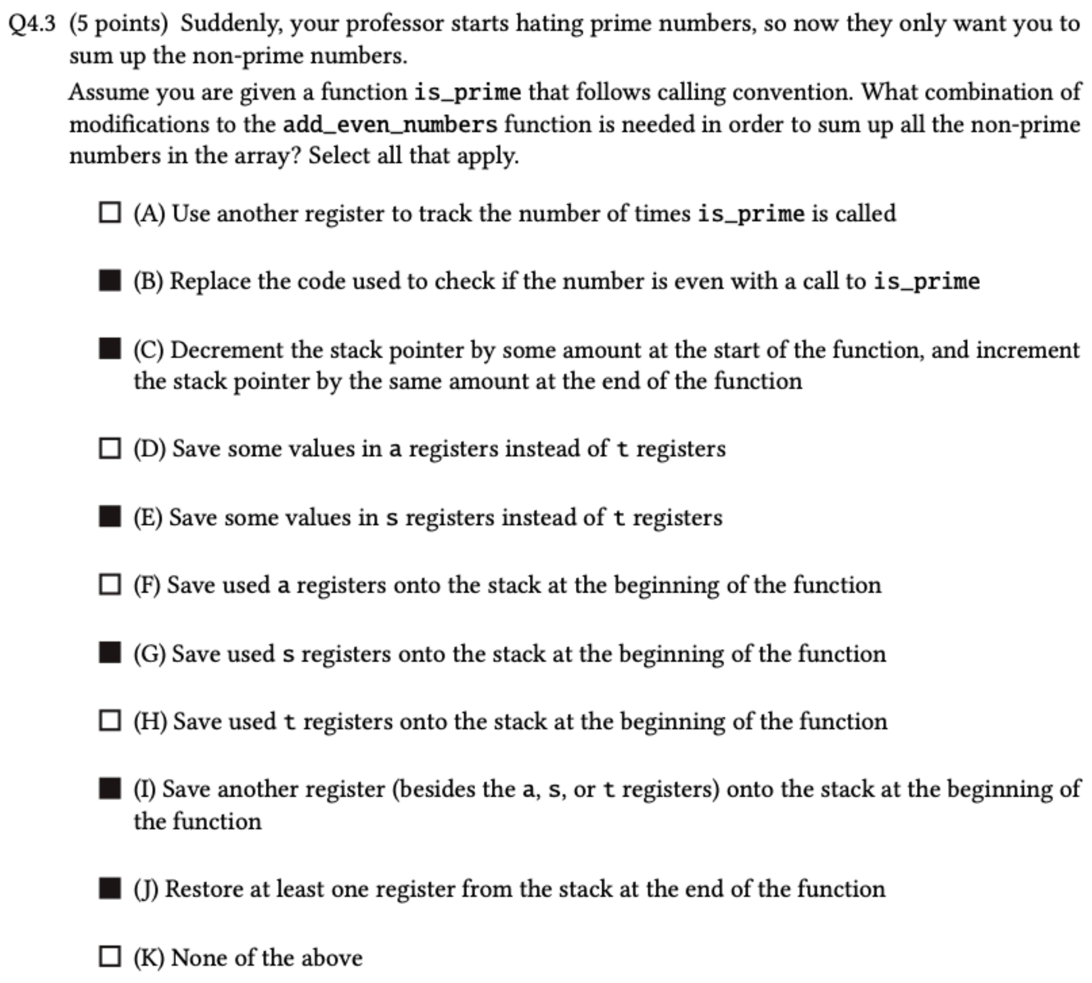
Replying to Anonymous Wren

No, `s2` and `a0` point to the same memory that's been allocated on the heap. If we `free(s2)` and try to access `a0` outside of `transform`, we would get an error.

♡ ···

Anonymous Rook 1y #488adc ✓ Resolved

For **SU22-MT1-Q4.3,** I was wondering why Option D is false. My thought process is that when we get each value in the numbers array, we would need to lw into a0 rather than t1, since we need to check if the number is prime by calling a function. To call a function, I thought you need to save the argument you pass into it with a0. Isn't that technically saving some values in a registers instead of t registers?



Q4.3 (5 points) Suddenly, your professor starts hating prime numbers, so now they only want you to sum up the non-prime numbers.

Assume you are given a function `is_prime` that follows calling convention. What combination of modifications to the `add_even_numbers` function is needed in order to sum up all the non-prime numbers in the array? Select all that apply.

- □ (A) Use another register to track the number of times `is_prime` is called
- ■ (B) Replace the code used to check if the number is even with a call to `is_prime`
- ■ (C) Decrement the stack pointer by some amount at the start of the function, and increment the stack pointer by the same amount at the end of the function
- □ (D) Save some values in `a` registers instead of `t` registers
- ■ (E) Save some values in `s` registers instead of `t` registers
- □ (F) Save used `a` registers onto the stack at the beginning of the function
- ■ (G) Save used `s` registers onto the stack at the beginning of the function
- □ (H) Save used `t` registers onto the stack at the beginning of the function
- ■ (I) Save another register (besides the `a`, `s`, or `t` registers) onto the stack at the beginning of the function
- ■ (J) Restore at least one register from the stack at the end of the function
- □ (K) None of the above

Eddy Byun STAFF 1y #488afe

`a` registers are caller-saved registers, which means that when we return from `is_prime` , we cannot assume that the value that was previously in the `a` register before we called `is_prime` is the same when we return from `is_prime`

Anonymous Fox 1y #488adb ✓ Resolved

Hi,

For **SU22-MT1-Q2.4,** I thought this would be invalid because we are only freeing the memory used to hold the node. However, since the node has an array of pointers to other nodes, we are only freeing the memory addresses from the root node's memory, but we aren't actually freeing the memory addresses' own memory blocks (that correspond to other nodes we created later).

Could you please tell me what the error is in my thinking? Is it that because this is a recursive function, free will be called on each node we visit?

Eddy Byun STAFF 1y #488bae

Yea, it's because the function is recursive. If `node == NULL` , this means we haven't allocated memory for the `node` , so we just return. Otherwise, we iterate through all the elements in our `next` array and recursively call `convert` where we will free all the allocated nodes inside of `next` before `free` -ing the root node.

Anonymous Fox 1y #488ada ✓ Resolved

Hi,

For **SU22-MT1-Q2.1-2,** is it acceptable to write, on line 3, just `word[i]` and then on lines 4 and 7 (for 2.1) and lines 4,5, and 7 (for 2.2) to write `next[char_to_ascii - 97]` to avoid the type cast?

E Eddy Byun STAFF 1y #488bbe

Yea, that's fine.

Anonymous Fox 1y #488acf ✓ Resolved

Hi,

For **SU22-MT1-Q1.4,** is the assembler's output (the object file) machine code with unresolved references, or is it something else?

J Jero Wang STAFF 1y #488caf

Could you clarify what you mean by unresolved references/how it relates to Q1.4? Yes, the assembler's output may have unresolved references if it calls functions or references labels in other object files.

Anonymous Fox 1y #488cde

Sure, I was really just wondering if the assembler's output (the object file) is machine code, or if it is written in some other language. I think my question was tangentially related to the question's solution, in which it says what the assembler outputs.

Anonymous Elk 1y #488acb ✓ Resolved

SU22-MT-Q4.2

Do labels like loop and pass also count as instructions? Also why is the offset the same for both of the solutions in Q4.1?

E Eddy Byun STAFF 1y #488afb

No labels don't count as instructions. The offsets are the same in both solutions because we move the pointer to the array by 4 bytes (1 index) in `pass` for both solutions. As a result, our offset is going to be 0

Anonymous Elk 1y #488bcd

I'm sorry, I should've phrased my question clearer. The answer key says that the offset for the j instruction is -36 bytes. Shouldn't it be -28 for the first solution and -32 for the second if we're not treating labels as instructions?

E Eddy Byun STAFF 1y #488cda

Replying to Anonymous Elk

> Assume that every line in the above code is filled with exactly one instruction (or pseudo-instruction that expands to one instruction).

Anonymous Elk 1y #488cee

Replying to Eddy Byun

So the blank lines are also considered instructions?

E Eddy Byun STAFF 1y #488dba

Replying to Anonymous Elk

We assume that every blank line has exactly one instruction.

Anonymous Elk 1y #488aca ✓ Resolved

SU22-MT-Q4.1

Is the first solution missing a mv a0, t0 and the second solution missing a ret?

Additionally, is ret the same as jr ra?

E Eddy Byun STAFF 1y #488afa

Yea, you're right. We need a `mv a0 t0` in the first solution and the second solution is missing a `ret`. Sorry for the confusion!

`ret` and `jr ra` are the same

Anonymous Elk 1y #488abf ✓ Resolved

SU22-MT-Q2.4

Why is it acceptable to not check if new_node is NULL or not after calloc? The same issue occurs for both Q2.3 malloc and Q2.5 realloc.

E Eddy Byun STAFF 1y #488aef

I think we assumed that the allocation will not return NULL for these problems.

Anonymous Elk 1y #488dbf

Should we always assume this is the case for future problems? Or will it be specified?

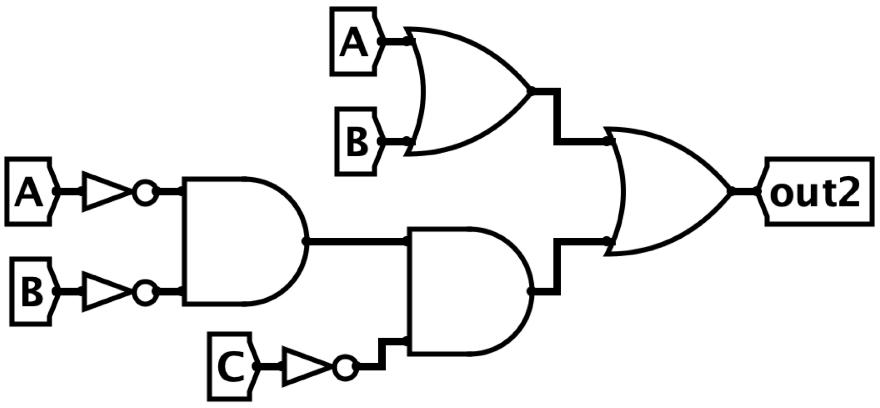
J Justin Yokota STAFF 1y #488ddc

Replying to Anonymous Elk

If you may make that assumption, it will be stated on the exam, or via clarification.

Anonymous Loris 1y #488abd ✓ Resolved

Q5.2 (4 points)



Solution: `(A + B) + ((~A * ~B) * ~C)` (direct translation of circuit)
`(A + B) + (~(A + B) * ~C)`
`A + B + ~C`

Summer 2022 MT:

How did we simplify (~ (A + B) * ~C) to just ~C?

Anonymous Elk 1y #488abe

(A + B) + (~(A + B) * ~C)

= ((A + B) + ~(A + B)) * ((A + B) + ~C)

= 1 * ((A + B) + ~C)

= A + B + ~C

Anonymous Loris 1y #488acd

= ((A + B) + ~(A + B)) * ((A + B) + ~C)

How did you get (A +B) + ~ C? Shouldn't it be ~(A + B) + ~C?

Anonymous Elk 1y #488ace

Replying to Anonymous Loris

X + (Y * Z) = (X + Y) * (X + Z)
Not sure if this makes sense.

1

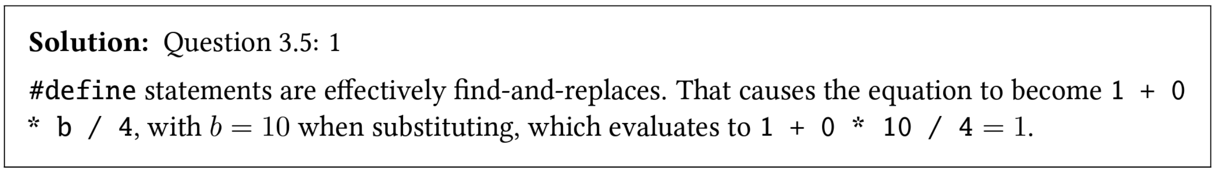
Andrew Liu STAFF 1y #488ccc

#488afc

Anonymous Flamingo 1y #488abc ✓ Resolved

SP22-Midterm-Q3

Question 3.5



Solution: Question 3.5: 1

`#define` statements are effectively find-and-replaces. That causes the equation to become `1 + 0 * b / 4`, with $b = 10$ when substituting, which evaluates to `1 + 0 * 10 / 4` $= 1$.

Should this not be 1/4, which becomes rounded down to 0?

♡ ···

E Eddy Byun STAFF 1y #488aec

It should be 1. 1 + 0*10/4 = 1 + 0/4 = 1

♡ 1 ···

Anonymous Flamingo 1y #488bde

Ah, I understand now.. Thank you!

♡ ···

Anonymous Heron 1y #488cbc

Why isn't it (1+0)*10/4?

♡ ···

E Eddy Byun STAFF 1y #488cca

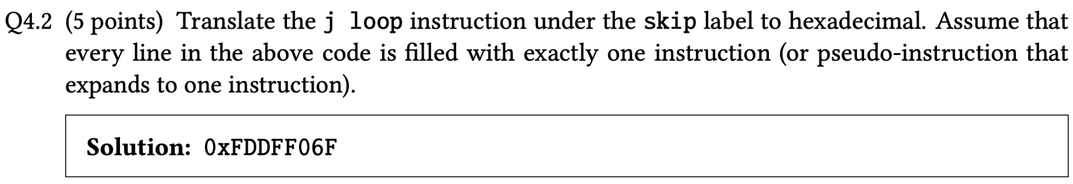
Replying to Anonymous Heron

define does find and replace and the "arguments" are not evaluated first

♡ ···

Anonymous Loris 1y #488aaf ✓ Resolved

Summer 2022 MT 2: I understand that j loop can be translated to jal x0 loop. However, how are we supposed to get intermediate bits from the loop label?



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`

♡ ···

E Eddy Byun STAFF 1y #488aba

#488ea

♡ ···

Anonymous Fish 1y #488aae ✓ Resolved

LOGIC CIRCUITS: Could someone explain in general how to find the longest and shortest combinational path in a circuit? I am getting these questions wrong and often factor in the wrong times, are there specific steps I should follow?

♡ 1 ···

Andrew Liu STAFF 1y #488afd

The main idea is that a combinational path always starts at any register's Q side and goes to any register's D side. So, you would draw every possible line between these things and then figure out the delays across each path, and then the longest and shortest can be found from all of those.

♡ ···

Anonymous Starling 1y #488aad ✓ Resolved

SU22-Q4.1

In *loop*, why it doesn't need *addi a0, a0, 4* to move the pointer?

Eddy Byun STAFF 1y #488abb

We move the pointer inside of `pass`

Anonymous Cod 1y #488aac ✓ Resolved

su22 3.1) aren't we supposed to add the bias (-15) instead of subtract? also, how did an exponent of 4 become -4 then become 4 again?

The exponent is -4, which we can convert to bias notation by subtracting the bias: $4-(-15) = 19$. In unsigned 5-bit binary, this is $0b10011$.

Eddy Byun STAFF 1y #488aed

"The exponent is -4" is a typo. Remember that for bias notation,

- To interpret stored binary: Read the data as an unsigned number, then add the bias
- To store a data value: Subtract the bias, then store the resulting number as an unsigned number

In this instance, we want to store the data value 4. To do that, we take 4, subtract the bias, and store the resulting number (19) as an unsigned number.

Anonymous Penguin 1y #488aab ✓ Resolved

SU22-MT-Q2.5

I don't quite understand the second half of the explanation: "Suppose realloc does change the pointer in the process of reallocation. In other words, the memory at node is freed, a new, larger block of memory is allocated, and a pointer to this new memory is placed in new_node. In this case, we encounter the same problem as the malloc case, where not all the pointers in this new block of memory have been allocated."

I'm confused because I feel like all the pointers in this new block memory will be allocated after we go through the for loops. Is it because the pointer to node was freed so we can't access it anymore to use in the second for loop?

Andrew Liu STAFF 1y #488aff

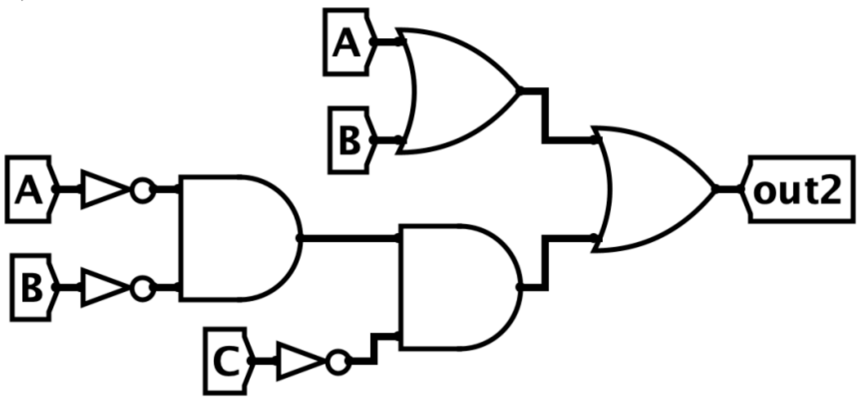
It's because the first `for` loop NULL's out all the pointers, which mean that when the second `for` loop runs, all it sees are `NULL` values.

Anonymous Wren 1y #488ee ✓ Resolved

SU22-MT-Q5.2

Hi, how did they simplify (A+B) + (~(A+B) * ~C) to A + B + C?

Q5.2 (4 points)



Solution: `(A + B) + ((~A * ~B) * ~C)` (direct translation of circuit)
`(A + B) + (~(A + B) * ~C)`
`A + B + ~C`

♡ 3 ...

Andrew Liu STAFF 1y #488afc

Let `D = (A + B)`. Then,

```
(A + B) + (~(A + B) * ~C) = D + (~D * ~C)
                          = D + ~C          (Absorption law)
                          = (A + B) + ~C
```

I would do case work here to simplify this pattern. I.e if `D` is true, then the whole statement is true, and if `D` is false, it relies entirely on the value of `~C`, so this simplifies to `D + ~C`. Alternatively,

```
(A + B) + (~(A + B) * ~C) = D + (~D * ~C)
                          = (D + ~D) * (D + ~C) # Distributive (OR)
                          = 1 * (D * ~C)        # Identity     (OR)
                          = (D + ~C)            # Identity     (AND)
                          = (A + B) + ~C        # Substitution
                          = A + B + ~C          # Associativity (OR)
```

♡ ...

Anonymous Starling 1y #488ed ✓ Resolved

SP22-Q4.2

Hi,

I'm not very familiar with ***converting a C program to a RISC-V***. I feel that my thoughts break down when I do this kind of problem. Can someone give me some key ideas or train of thought?

Thanks a lot!

♡ ...

E Eddy Byun STAFF 1y #488bee

Yea, that's completely understandable - it's a really tricky question.

When I see this problem, the first thing that caught my eye were the `jal` and `jalr` instructions. That got me to think that we needed to use `s` registers (which is also indicated by the fact that we see `addi sp sp` on the very first line). The question then becomes which registers and how many registers do I need to save on the stack? I think that as you do the problem, you may see that you need to use more `s` registers or you need to save `ra`, so I wouldn't be too worried about getting this right the first time you're going through the problem.

The second part of the problem is to know what needs to go in between the `jal` and `jalr` calls. We know that we need to send in arguments to `malloc` and to `f`, but what else do we need to do? Do we need to save the values that either `malloc` or `f` returns? If so, where do we need to save these values? How do we save these values?

And finally, the last part of the problem is to put any return values in our `a` registers, do our epilogue and then return.

This was my train of though when I saw this question

Anonymous Grouse 1y #488df ✓ Resolved

SU - 2.4

Does realloc create an array and intializes it to 0 or null or are they the same thing as I'm getting mixed answers online.

E Eddy Byun STAFF 1y #488ef

If we attempt to `realloc` a pointer to allocate more space than it currently has, then we fill up the newly allocated space with garbage values; `realloc` will not zero out the newly allocated space.

Anonymous Grouse 1y #488fa

would calloc initialize the new values with null or 0 ?

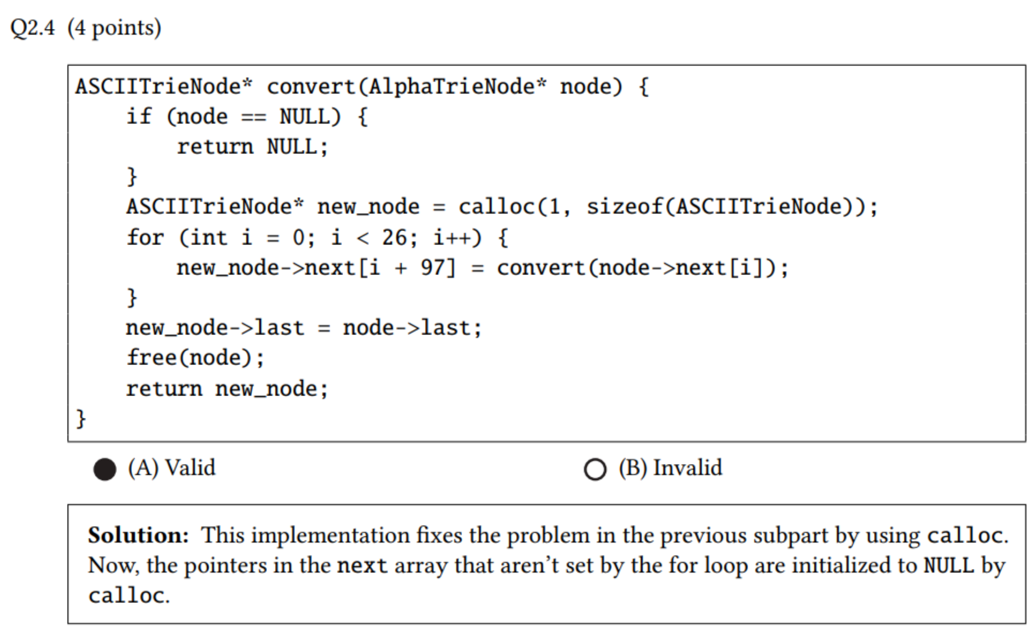
E Eddy Byun STAFF 1y #488fb

Replying to Anonymous Grouse

`calloc` would allocate new space on the heap and zero's out the memory that you allocate.

Anonymous Grouse 1y #488fc

Replying to Eddy Byun



Q2.4 (4 points)

```
ASCIITrieNode* convert(AlphaTrieNode* node) {
    if (node == NULL) {
        return NULL;
    }
    ASCIITrieNode* new_node = calloc(1, sizeof(ASCIITrieNode));
    for (int i = 0; i < 26; i++) {
        new_node->next[i + 97] = convert(node->next[i]);
    }
    new_node->last = node->last;
    free(node);
    return new_node;
}
```

[✓ (A) Valid] (B) Invalid

Solution: This implementation fixes the problem in the previous subpart by using `calloc`. Now, the pointers in the `next` array that aren't set by the for loop are initialized to `NULL` by `calloc`.

For 2.4, they said calloc would be valid. But it also says that it would be initialized to Null. So wouldn't calloc work as well or ?

E Eddy Byun STAFF 1y #488fd

Replying to Anonymous Grouse

Ah sorry, I confused this problem with another problem. Yes, `calloc` would work here since we zero out the memory that we allocate. Apologies for the confusion!

Anonymous Grouse 1y #488ff

Replying to Eddy Byun

Is zeroing out the memory the same thing as setting the pointers to NULL

E Eddy Byun STAFF 1y #488aaa

Replying to Anonymous Grouse

Yes

Anonymous Gnat 1y #488baa

Replying to Eddy Byun

Why does the solution write calloc(1, sizeof(ASCIITrieNode)) for 2.4? If writing calloc(sizeof(ASCIITrieNode) achieves the same purpose, using the value 1 seems arbitrary, if not wrong

E Eddy Byun STAFF 1y #488bac

Replying to Anonymous Gnat

`calloc` takes in two arguments: void *calloc(size_t nitems, size_t size), so `calloc(sizeof(ASCIITrieNode)` would raise a compilation error

Anonymous Fish 1y #488de ✓ Resolved

SP22-MT-Q2.5:

To find the smallest positive number that can be represented by floating point, why can I not set all of the mantissa bits to 0? On the chart it says that a normal number has its exponent 1-254, so I understand for the smallest positive number the exponent must be 1. But it says the mantissa can be anything so I thought I could set it to 0. Also, I thought for floating point you always have to add a 1 to the significand. How do you get a 0.___ instead for this question? Also why did they add 1 to the exponent? I thought the exponent is 0+(-3) but where does the 1 come from?

Screen Shot 2023-10-07 at 7.55.22 PM.png

Eddy Byun STAFF 1y #488ec

Your exponent bits can be all 0's (remember denorm numbers!), but if your mantissa bits are also all 0, we have either + or - zero depending on your sign bit. Recall that for denorm numbers, we represent them as 0.mantissa$_2$ * 2$^{\text{bias + 1}}$

Anonymous Rook 1y #488dd ✓ Resolved

For SP22-MT-Q2.4, why isn't the largest exponent 111 instead of 110?

Eddy Byun STAFF 1y #488eb

If our exponent is 111, that means we have NaN or + or - infinity

Anonymous Narwhal 1y #488dc ✓ Resolved

SU-MT-Q4.2

How to do you calculate the immediate value for j loop which just translates to jal x0 loop in this question?

Eddy Byun STAFF 1y #488ea

`loop` is 9 instructions above the `j loop` instruction (we assume that all the blanks are filled it). This means we have to go back 9 instructions from `j loop` to get to `loop`, which is 36 bytes. Our immediate value is going to be -36

Anonymous Antelope 1y #488dcd

Why -36 and not 36?

Anonymous Narwhal 1y #488db ✓ Resolved

SU22-MT-Q2.3

Why don't we need to free node->next before freeing node? I'm a bit confused when exactly we need to free struct variables and when we can just free the struct it self.

Eddy Byun STAFF 1y #488fe

Note that we recursively call `convert(node->next[j])`, which also means we recursively `free` all the struct elements that we previously allocated before `free`-ing the struct itself.

Anonymous Rook 1y #488cd ✓ Resolved

For Su22 #4.3, why is it correct to save items onto the s register, but incorrect to save all (used) values onto the stack and restore them afterward?

Shouldn't both approaches be fine for calling convention?

S Sam Xu STAFF 1y #488ce

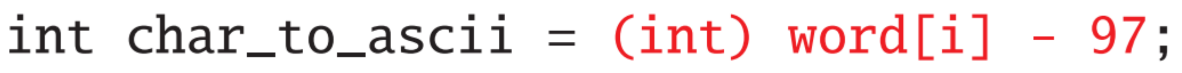
I assume you are asking for option FH:

- (F) Save used a registers onto the stack at the beginning of the function
- (H) Save used t registers onto the stack at the beginning of the function.

You are right that we can save used values to the stack before we jump to other function, and restore them after jump back from other functions. Here the options are not correct because they said save onto stack at the beginning of the function, not before jump to other function.

Anonymous Rook 1y #488ca ✓ Resolved

For Su22 #2.1, why do we subtract by 97?



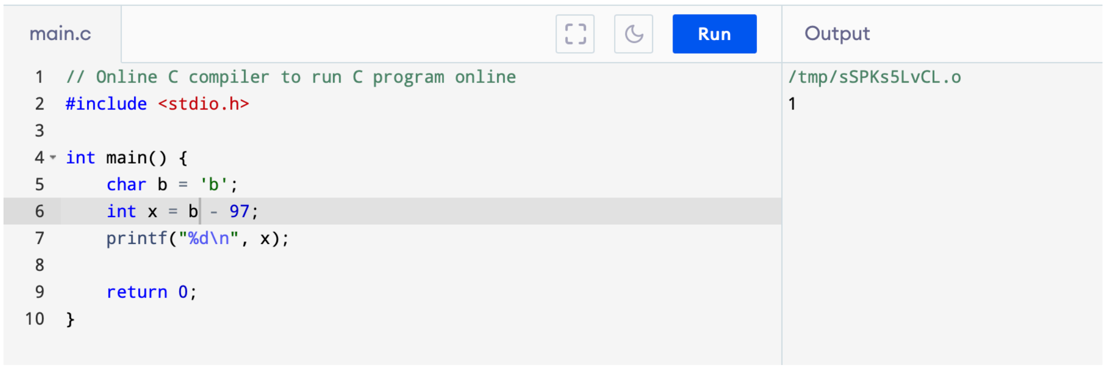
```
int char_to_ascii = (int) word[i] - 97;
```

E Eddy Byun STAFF 1y #488cb

It's because the ASCII character 'a' starts at ASCII value 97 and we want index 0 to represent the 'a' node so we subtract 97.

Anonymous Mantis 1y #488cf

For midterms, do we need to type cast? The above example works without `(int)`, as shown below, yet the answers still include `(int)`.



```c
// Online C compiler to run C program online
#include <stdio.h>

int main() {
    char b = 'b';
    int x = b - 97;
    printf("%d\n", x);

    return 0;
}
```

```
/tmp/sSPKs5LvCL.o
1
```

E Eddy Byun STAFF 1y #488bbc

Replying to Anonymous Mantis

You didn't need to cast word[i] as an int to get full credit for this solution.

Anonymous Kookabura 1y #488bbb

Why wouldn't we put * in front of word in order to deference it?

1

Eddy Byun STAFF 1y #488bbd

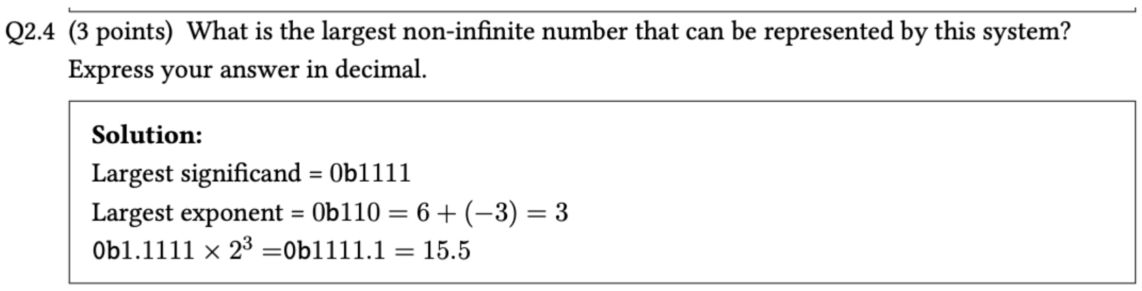
Replying to Anonymous Kookabura

word[i] and *(word+i) are the same. If we did *(word[i]), we would likely get a seg fault since we would treat the ASCII value of a character as a memory address and try accessing that memory address.

Anonymous Wren 1y #488bf ✓ Resolved

SP22-MT1-Q2.4

Why can't we represent the floating point exponent as 0b111?



Q2.4 (3 points) What is the largest non-infinite number that can be represented by this system? Express your answer in decimal.

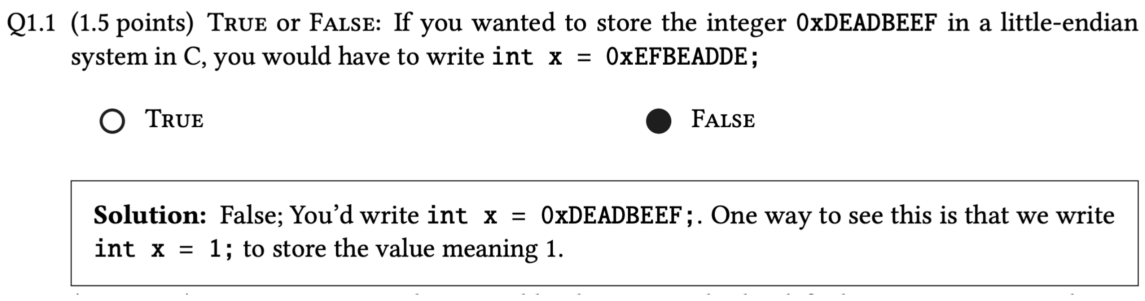
Solution:
Largest significand = 0b1111
Largest exponent = 0b110 $= 6 + (-3) = 3$
0b1.1111 $\times 2^3 =$0b1111.1 $= 15.5$

Eddy Byun STAFF 1y #488cc

If you have an exponent of all 1's, this means you have either a NAN or infinity.

Anonymous Grouse 1y #488ac ✓ Resolved

SP22-MT1-Q1.1



Q1.1 (1.5 points) TRUE or FALSE: If you wanted to store the integer `0xDEADBEEF` in a little-endian system in C, you would have to write `int x = 0xEFBEADDE;`

TRUE [✓ FALSE]

Solution: False; You'd write `int x = 0xDEADBEEF;`. One way to see this is that we write `int x = 1;` to store the value meaning 1.

I dont really understand the explanation for why it would simply be 0xDEADBEEF, cause in memory, wouldnt the least significant bytes go first so it would look something like

0xEDBEADDE? Maybe I'm reading the question wrong, but I am confused by the explanation

Justin Yokota STAFF 1y #488ad

The memory state may indeed be 0xED 0xBE 0xAD 0xDE, but that shouldn't affect how we write our code. If we want a number meaning 0xDEADBEEF, we should write 0xDEADBEEF;

we don't care how memory will actually store that value, so long as retrieving x will return 0xDEADBEEF.

Anonymous Grouse 1y #488af

thanks! So hypothetically, if the question was asking in a big-endian system instead, the answer would still be xDEADBEEF, right?

Justin Yokota STAFF 1y #488be

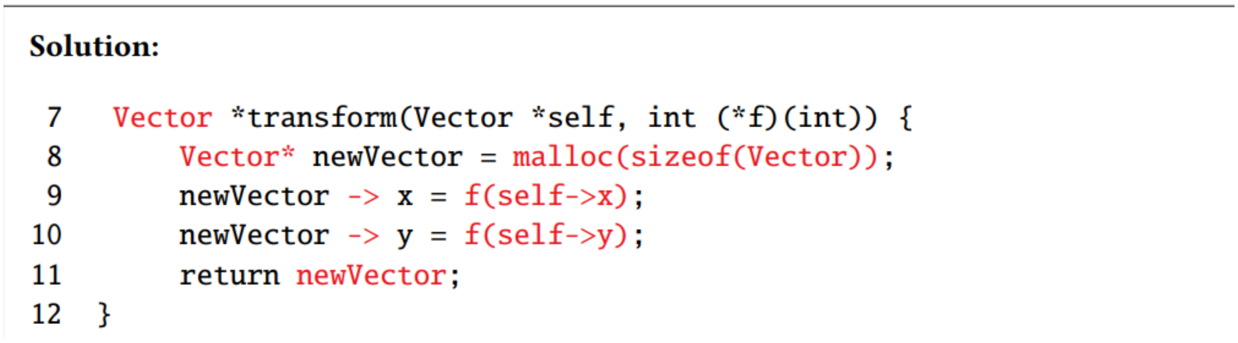
Replying to Anonymous Grouse

Yup. If we want 0xDEADBEEF, we should write 0xDEADBEEF, regardless of the endianness.

Anonymous Human 1y #488ab ✓ Resolved

SP22-MT-Q4.2

When passing functions as arguments, why don't we dereference?



Solution:

```
Vector *transform(Vector *self, int (*f)(int)) {
    Vector* newVector = malloc(sizeof(Vector));
    newVector -> x = f(self->x);
    newVector -> y = f(self->y);
    return newVector;
}
```

For example, why don't we do (*f)(self->x)?

Justin Yokota STAFF 1y #488bc

We could; the two are equivalent. In reality, this is because C is kind of bad with function pointers; the C developers tried to combine HOFs with pointer mechanics, so it causes this weird syntax.

2

Anonymous Human 1y #488bd

I see. Thank you!

Anonymous Kingfisher 1y #488aa ✓ Resolved

SP22-MT-Q2.5:

why is the +1 in the smallest exponent calculation: 0b000 = 0 + (−3) **+ 1** = −2?

1

Justin Yokota STAFF 1y #488ae

Denormalized numbers have one higher exponent than normal.

Anonymous Penguin 1y #488f ✓ Resolved

SP22-MT-Q1.1

If we were to do int x = 0xEFBEADDE on a *big endian system*, would we correctly store 0xDEADBEEF?

Justin Yokota STAFF 1y #488ba

No; you'd store 0xEFBEADDE if you did "int x = 0xEFBEADDE", regardless of the endianness of the system. The endianness will only affect if 0xEFBEADDE gets saved internally as 0xEF 0xBE 0xAD 0xDE or 0xDE 0xAD 0xBE 0xEF, but this should have no effect on our code. For example, if you said "int x = 1", do you expect that using x on a later line of code will get you 0x01000000?

Anonymous Antelope 1y #488dfb

Are you saying little endian only applies to the address of x? this question is false because x is an int that is being reversed when we know nothing about the address

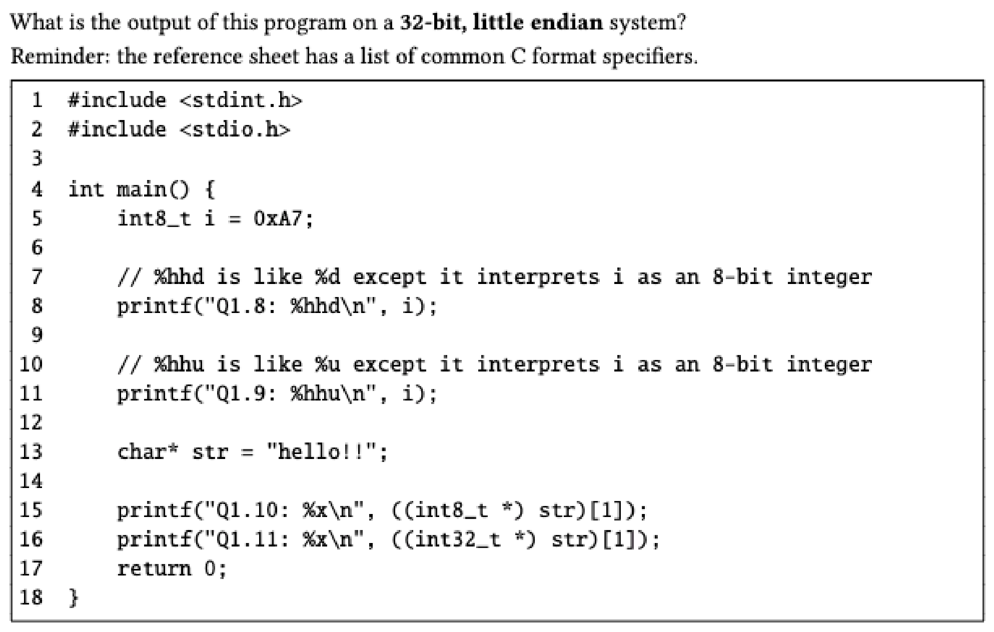
Justin Yokota STAFF 1y #488dfe

Replying to Anonymous Antelope

No; little endian applies only to how we write the value in memory, and how we read the memory back. Regardless of if I'm writing in English or Arabic, if I write down number x, I should read it back as number x.

Anonymous Antelope 1y #488dff

Replying to Justin Yokota

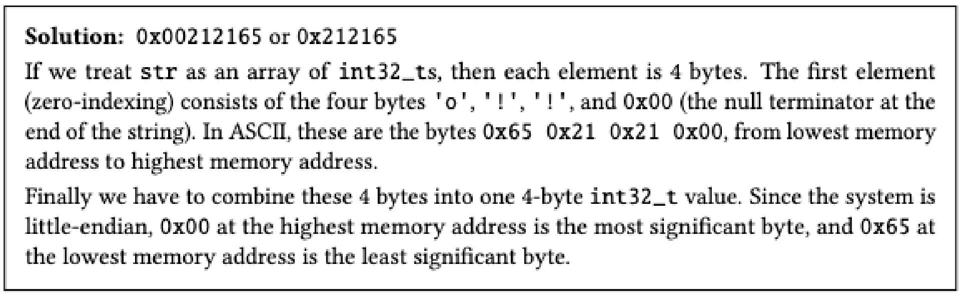


What is the output of this program on a **32-bit, little endian** system?
Reminder: the reference sheet has a list of common C format specifiers.

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

int main() {
    int8_t i = 0xA7;

    // %hhd is like %d except it interprets i as an 8-bit integer
    printf("Q1.8: %hhd\n", i);

    // %hhu is like %u except it interprets i as an 8-bit integer
    printf("Q1.9: %hhu\n", i);

    char* str = "hello!!";

    printf("Q1.10: %x\n", ((int8_t *) str)[1]);
    printf("Q1.11: %x\n", ((int32_t *) str)[1]);
    return 0;
}
```

then why is the answer to 1.11 the addresses of 0,!,!,o? this would be reading the string in backwards

Q1.11 (2 points)



Solution: 0x00212165 or 0x212165

If we treat `str` as an array of `int32_ts`, then each element is 4 bytes. The first element (zero-indexing) consists of the four bytes `'o'`, `'!'`, `'!'`, and `0x00` (the null terminator at the end of the string). In ASCII, these are the bytes `0x65 0x21 0x21 0x00`, from lowest memory address to highest memory address.

Finally we have to combine these 4 bytes into one 4-byte `int32_t` value. Since the system is little-endian, `0x00` at the highest memory address is the most significant byte, and `0x65` at the lowest memory address is the least significant byte.

J

Justin Yokota STAFF 1y #488eab

Replying to Anonymous Antelope

It's actually not: The string is read forwards first to determine which piece of data corresponds to our integer:

Bytes 4-7 of "hello!!\0" are "o!!\0". How we read this depends on the endianness of the system; in a little-endian system, the byte sequence 0x6F 0x21 0x21 0x00 is interpreted as the 4-byte value 0x0021216F. Here the difference is that I'm writing something of type X and interpreting it as a completely different type Y. If I write down a sequence of letters left-to-right, then try to interpret that sequence of letters as if it were a complete word, it definitely matters whether I'm reading in English or Arabic.

Anonymous Antelope 1y #488eac

Replying to Justin Yokota

i see, endianness only affects the way the address is read but not the value itself

Anonymous Penguin 1y #488e ✓ Resolved

SP22-MT-Q2.5

Why did we do 0-3+1=-2 to get the smallest exponent we can get? I thought the smallest exponent would just be 0-3=-3. Why do we add the 1?

J

Justin Yokota STAFF 1y #488bb

#488ae

Anonymous Lark 1y #488a ✓ Resolved

SU22-MT-Q4.1

When checking even integers and using bitwise AND with 1, we are checking whether the least significant bit is 1 right? But why do we use beq with x0 here? We want to keep the numbers with the least significant bit being 0 and the bitwise AND will result in 0 in this case.

E

Eddy Byun STAFF 1y #488d

Yea, sorry this is an error in our solution. It should be `bne` instead of `beq` . Apologies for any confusion!