

You are viewing this thread in readonly mode.

[Midterm] Past Exams - 2022 #883



Jero Wang ADMIN
2 years ago in Exam - Midterm

1,829
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**



Anonymous Herring 2y #883ebf ✓ Resolved

SP22-MT-Q3

line8: would Vector and *malloc be the same as the solution?

Solution:

```
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 }
```



Erik Yang TA 2y #883ecb

no, because the function wants us to return a vector pointer, so you have to retain the Vector* from the malloc call



Anonymous Herring 2y #883ebc ✓ Resolved

SP22-MT-Q3.2

```

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

```

for sp22 3.2 why are we storing s registers? Aren't we the caller for malloc?

- is it because we are also a callee from wherever its calling from and are using s registers?

which means we could also store other registers thats not s?

♡ ...

 E Erik Yang TA 2y #883ebd

if you use any sort of s registers, since they are callee saved registers, you have to store them on stack before you can start using them

♡ ...

  Anonymous Herring 2y #883ebe

what does s2 store after malloc? is it a pointer to the memory? from what I see in the code, it assumes that a0 after malloc returns is the self vector. Why is that?

♡ ...

E Erik Yang TA 2y #883eca

↩ Replying to Anonymous Herring

yup it stores the memory that was malloced in a0. Usually, a0 is the output for functions. So the code moves that output to s2.

♡ ...

 Anonymous Salmon 2y #883ead

✓ Resolved

SP22-MT-Q2.5

Q2.5 (3 points) What is the smallest positive number that can be represented by this system?

Express your answer as an odd integer multiplied by a power of 2.

Solution:

Smallest significand = 0b0001

Smallest exponent = 0b000 = $0 + (-3) + 1 = -2$

$0.0001 \times 2^{-2} = 1 \times 2^{-6}$

Why is the smallest exponent $0 + (-3) + 1$ instead of just $0 + (-3)$?

♡ ...

 N Nikhil Kandkur TA 2y #883eae

We are working with denormalized numbers here so we have to add 1 to the exponent.

♡ ...

 Anonymous Echidna 2y #883dff

✓ Resolved

Sp22-mt-q4.2

For line 12, why can't you use mv instead of lw? Or does either work?

♡ ...

 E Erik Yang TA 2y #883eaa

You're trying to load one element of the array into a0 while mv moves the entire register into a0 so it wouldn't be the same

♡ ...

 Anonymous Echidna 2y #883eac

Got it, thanks! And just to clarify; say the first element of a struct is a char, we would do load byte instead, right?

♡ ...

E Erik Yang TA 2y #883eaf

↩ Replying to Anonymous Echidna

Yes!

♡ 1 ...

 Anonymous Dogfish 2y #883dfc

✓ Resolved

SP22-mt-Q4.2

In line 13 and 16, what is ra here?

♡ ...

 **Erik Yang** TA 2y #883dfd

Ra is the ra address, it tells the function where to return to after it is done executing

♡ ...

 **Anonymous Sardine** 2y #883dfa ✓ Resolved

su22-mt-q4

For 4.1, it feels like we need to return the sum of all the even numbers by storing that value in a0. Since t0 is the running sum, it seems like we therefore need to move the value of t0 into a0. Maybe a line like `li a0 t0`. However, the solution directly returns without storing the value of t0 into a0, which should mean that it is returning a pointer to the end of the array, rather than the value of the sum of all the even numbers. Why does the solution feel no need to move the running sum into a0 if the expected output is the sum of even values in the array?

```
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
```

♡ ...

 **Jero Wang** ADMIN 2y #883dfb

[#883ade](#)

♡ ...

 **Anonymous Goldfinch** 2y #883ded ✓ Resolved

su22-mt-q2.3: why is it invalid if somethings are not initialized? could it potentially have bad behavior?

♡ ...

 **Erik Yang** TA 2y #883dee

Yeah since not all the pointers are initialized in the next array, that means we will have undefined behavior if we ever try to do something with some of those pointers inside next that haven't been initialized

♡ ...

 **Anonymous Red panda** 2y #883ddb ✓ Resolved

su22-mt-q4

for 4.1, why don't we do the line `addi a0 a0 4` in the `add_even_numbers` block as well because don't we want to move the pointer regardless if it's an even or odd number?

♡ ...



Jero Wang ADMIN 2y #883dec

It is in the `add_even_numbers` function, just part of the loop within the function. The line you mentioned gets executed regardless if the number is odd or even (if even, it adds, then goes to the line you mentioned, if odd, it branches to `pass`, which is the line that you mentioned).

♡ ...



Vishnu Suresh 2y #883dce

✓ Resolved

SU22-MT-Q3.6

Could someone explain how we got $2^5 - 2^{-6}$ for this. I did not understand the step size logic used in the solution.

♡ ...



Eric Kusnanto TA 2y #883dde

$2^5 - 2^{-6}$ refers to a float with exponent bits = 4 and a mantissa of all 1s (1.111...) This can be calculated based on the power of two that the LSB is at, also known as step size. Similar to how you can calculate an unsigned int `0b01111` as `0b10000 - 0b00001` ($2^4 - 2^0$), we can extend this to calculate our float as $2^5 - \text{step size}$ instead of adding 10 powers of two together. Step size can be found for normalized floats as $2^{(\text{exp} + \text{bias})} / 2^n$, where $n = \#$ of mantissa bits

♡ ...



Anonymous Salmon 2y #883dbb

✓ Resolved

SU22-MT-Q4.2

How do we get the answer `0xFDDFF` for the immediate bit? I know someone asked a similar question but it doesn't address my specific concern where I'm just having trouble translating to two's complement. Here is my work:

To represent -36 in binary, we want to find 36 in binary, flip its bits, then add 1.

$36 = 32 + 4 = 2^5 + 2^2$, so we can represent it as: `0b 0000 0000 0000 0010 0100`, so -36 is represented in 2's complement as `0b 1111 1111 1111 1101 1100`. Now I'm trying to grab the bits corresponding to 20, 10:1, 11, 19:12 and then stack them according to how immediates are stored in the instruction.

I get the 20th bit is 1, bits 19:12 are `111 1111 1`, bit 11 is 1, and bits 10:1 are `11 1101 1100` by just taking these directly from -36 in 2's complement.

Stacking these together, I get

`0b 1 | 11 1101 1100 | 1 | 111 1111 1` which is `imm[20] | imm[10:1] | imm[11] | imm[19:12]`.

Translating this to hexadecimal, I get `0b 1111 1011 1001 1111 1111` which is `0x FD9FF` which is pretty close to the answer of `0xFDDFF`, but I'm not sure where I went wrong

♡ ...



Erik Yang TA 2y #883dbc

remember bits are zero indexed, so bit 20 is actually is at `0b 1111 1111 1011 1001 1111 1111`

♡ ...



Anonymous Salmon 2y #883dbd

Isn't the thing you wrote 24 bits instead of 20? Shouldn't the immediate just be 20 bits?

♡ ...

E Erik Yang TA 2y #883dbe

↩ Replying to Anonymous Salmon

well -9 is gonna be a 32 bit integer, but you only use bits 1-20 for the imm in the risc v instruction

♡ ...

Anonymous Salmon 2y #883dcb

↩ Replying to Erik Yang

Gotcha! And in general if I'm given an immediate like `0xFDDFF` for a j-type instruction, I also have to do this process in reverse to recover `imm[20] | imm[10:1] | imm[11] | imm[19:12]`, stack it in the correct order of `imm[20] | imm[19:12] | imm[11] | imm[10:1]` and then convert that back to decimal?

♡ ...

E Erik Yang TA 2y #883dcc

↩ Replying to Anonymous Salmon

if you're given `0xFDDFF`, then you reverse the stacking basically so the first bit of the imm goes into the 20th position, etc., so yeah you're right. And then make sure you know if it's gonna be sign extended or not to 32 bits

♡ 1 ...

Anonymous Salmon 2y #883dcd

↩ Replying to Erik Yang

How do we know if it's gonna be sign extended or not? Or do you know which lecture I could look at to figure this out?

And when I recover the immediate I guess I just have to look at the sign or zero-extended 32 bit integer and convert that back to decimal?

Also what happens to the bit at position 0? Does sign/zero extending apply to that as well, or is it just for bits 21-31?

♡ ...

E Erik Yang TA 2y #883dcf

↩ Replying to Anonymous Salmon

hmm i tried looking for it in lecture. i know i-types are always sign extended, but i'm actually not sure for the others. Sign extension only applies to bits 21-31. You could ask in OH tom!

♡ ...

Anonymous Trout 2y #883dad

✓ Resolved

SP22-MT-Q3.4

What does the solution mean by "#define statements are effectively find-and-replaces?" Does that mean we do not evaluate the operand before a function call in C?

♡ ...

E Erik Yang TA 2y #883dae

yeah, this means that $0 + 1$ will not be evaluated since we are replacing that with a in the define statement; so that's why it will be $0 + 1 * 10 / 4$ and not $1 * 10 / 4$.

♡ ...

Anonymous Trout 2y #883daf

Does this rule apply to all C code? For example, suppose h is a one-argument function, would $f(h(x), h(y))$ be $h(x) + h(y)/4$, with $h(x)$ and $h(y)$ not evaluated until we run the function?

♡ ...

E Erik Yang TA 2y #883dba

↩ Replying to Anonymous Trout

If f is a defined statement then yeah

♡ ...

Anonymous Cheetah 2y #883dbf

↩ Replying to Erik Yang

Why does the b get evaluated to 10, but the $0 + 1$ equation does not? How do we know what we should and shouldn't evaluate with these define replacements?

♡ ...

E Erik Yang TA 2y #883dca

↩ Replying to Anonymous Cheetah

so what define does is it finds what the a and b arguments are for f , and then replaces them with what they are. So it will see " $0 + 1$ " and replace that for a , and then it sees " 10 " and replaces that with b . This gives you $0 + 1 * 10 / 4$

♡ 1 ...

Anonymous Salmon 2y #883cfc

✓ Resolved

SU22-MT-Q4.1

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
```

Why does `andi t2, t1, 1`, then `beq t2, x0, pass` check if $t1$ is odd? I think that what we should care about/look at is the LSB of $t1$ and check if that's 0 (even) or 1 (odd), but I feel like `andi t2, t1, 1` will just return $t1$ back which isn't necessarily the same as $x0$ if $t1$ is odd

♡ ...

E Erik Yang TA 2y #883cfd
#883bb
♡ ...

Anonymous Salmon 2y #883cfe
I think my question still applies even if they have a bne , I just don't really understand what the andi t2 t1 1 does because I feel like this just gives us t1 back, which in general will not be 0
♡ ...

E Erik Yang TA 2y #883cff
↩ Replying to Anonymous Salmon
x & 1 produces a value that is either 1 or 0 , depending on the least significant bit of x : if the last bit is 1 , the result of x & 1 is 1 ; otherwise, it is 0 . This is a bitwise AND operation
♡ ...

Anonymous Salmon 2y #883daa
↩ Replying to Erik Yang
If we had x represented as 101 (5 in decimal), then wouldn't x & 1 give 010 which is 2?
♡ ...

E Erik Yang TA 2y #883dab
↩ Replying to Erik Yang
If you think about it like 0b110001 being t1, then andi t2 t1 1 would be 0b110001 & 0b000001 which means all the ands with zero will be come zero and the lsb being 1 will be 1
♡ 1 ...

Anonymous Salmon 2y #883dac
↩ Replying to Erik Yang
Ohhhhh, that makes a lot more sense! Thanks
♡ ...

Anonymous Hare 2y #883cde ✓ Resolved
This is more of a conceptual question, but why do we not care about the hold time when finding the minimum clock period?
♡ ...

R Rosalie Fang ADMIN 2y #883ced
hold time is the time after rising edge of the clock where input has to stay constant, which happens at the same time as the clk-to-q time - no output wire is dependent on the hold time, as long as the rule is upheld.
♡ 1 ...

Anonymous Quetzal 2y #883cdc ✓ Resolved
SU22-MT-Q4.2
I don't understand how we go from -36 in decimal to "0xFDDFF06F". Are we not suppose to use 2's complement to calculate the hexadecimal representation?

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`

Optionally, for partial credit, write the offset in bytes as a decimal number in the box below.

Solution: `-36`

The line of code labeled `loop` is 9 instructions before the `j` loop instruction.



Gina Choi 2y #883cdd

we don't just do `-36`! we also have to include the entire instruction in hex as well



Anonymous Quetzal 2y #883cdf

Thank you!



Anonymous Porcupine 2y #883ccf

✓ Resolved

SP22-MT-Q3

May I ask when are the characters interpreted as boolean logic and when are they interpreted as performing math computations? Thanks!



Rosalie Fang ADMIN 2y #883cee

Depending on the context. If you have code, like in this question, follow the C convention (and will be `&`, or will be `|`, not will be `~`, and xor will be `^`). Only when you have a Boolean logic question you can use `(+ , !, *)` as Boolean logic.



Anonymous Salmon 2y #883ebb

So just to clarify, are the `&`, `|`, `~`, `^` just applied to logical values like `a & b = true` iff `a = true`, `b = true` or do we read it as `a & b` is a bitwise operation that `&`s both `a` and `b`?



Anonymous Goose 2y #883cce

✓ Resolved

SP22-MT-Q2.5

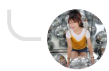
Q2.6 (3 points) Translate the following RISC-V instruction into its corresponding hexadecimal value.

```
ori t6 s0 -12
```

```
Solution: FF446793
opcode = 0b001 0011
funct3 = 0b110
rd = t6 = x31 = 0b11111
rs1 = s0 = x8 = 0b01000
imm = -12 = 0b1111 1111 0100
0b111111110100 01000 110 11111 0010011
0b1111 1111 0100 0100 0110 1111 1001 0011
0xFF44 6F93
```

is there a typo here? the solution says 0xFF446793 but the final line says 0xFF446F93

♡ ...



Gina Choi 2y #883cdb

yeah i asked apparently it's a typo

♡ 1 ...



Anonymous Peafowl 2y #883ccd

✓ Resolved

SU22-MT-Q2.5

Could anyone explain it in detail, especially when the realloc does change the pointer?

♡ 1 ...



Rosalie Fang ADMIN 2y #883cfa

When you call realloc and it changes the pointer, the memory address at the old node is freed. We should not be able to access node->next[i] in the second for loop.

♡ ...



Anonymous Peafowl 2y #883ddf

Do you mean that we cannot access a freed memory address?

♡ ...



Jero Wang ADMIN 2y #883deb

↩ Replying to Anonymous Peafowl

Yes, you cannot access a free'd memory address.

♡ ...



Anonymous Marten 2y #883ccc

✓ Resolved

SP22-MT-4.2

Q4.2 (20 points) Translate the `transform` function to RISC-V. The function takes inputs `self` in `a0` and `f` in `a1`, and returns output in `a0`. You may assume that `Vector` is as defined in the C code. You may also assume that you have access to `malloc`, and that `malloc` and `f` each receive their argument in `a0`, and return their result in `a0`. Your solution MUST fit within the lines provided.

```

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

```

For this question do we assume that Vector is just a pointer to two consecutive ints in memory, which is why we only malloc 8 bytes?

♡ ...

 R **Rosalie Fang** ADMIN 2y #883cef

```

typedef struct Vector {
    int x;
    int y;
} Vector;

```

This is from the same question. Since this is a struct, x and y should be consecutive in memory.

♡ ...



Anonymous Salmon 2y #883ccb

✓ Resolved

SU22-MT-Q3.3

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).

In general, given that we have m mantissa bits and an interval $[a, b]$ with $a = 2^n$, $b = 2^{n+1}$, is the step size between numbers just $2^{-m} \times 2^n$? But the number of representable numbers should be the length of the interval/step size, which will always give just 2^m representable numbers in any interval $[a, b]$ as described above?

$$\frac{(2^{n+1} - 2^n)}{2^{-m+n}} = \frac{2^n}{2^{-m+n}} = 2^m$$

♡ 1 ...



R Rosalie Fang ADMIN 2y #883cec

Yes. another way you can think about it is that all the numbers between $[a, b]$ with $a = 2^n$, $b = 2^{n+1}$ will have the same sign and exponent value, and the mantissa bits have all possible combinations, giving you 2^m .

♡ ...



Anonymous Salmon 2y #883cca

✓ Resolved

SU22-MT-Q2.2

Q2.2 (8 points) `insert` takes in a trie root node (`node`), and a pointer to a string (`word`). It should insert the word into the trie.

```
1 void insert(AlphaTrieNode* node, char* word) {
2   for (int i = 0; i < strlen(word); i++) {
3     int char_to_ascii = (int) word[i] - 97;
4     if (node->next[char_to_ascii] == NULL) {
5       node->next[char_to_ascii] = calloc(1, sizeof(AlphaTrieNode));
6     }
7     node = node->next[char_to_ascii];
8   }
9   node->last = true;
10 }
```

Why should we be callocing 1 AlphaTrieNode instead of 26? I would think it should be 26 because you want 1 node per letter, not just 1

♡ ...

 R **Rosalie Fang** ADMIN 2y #883ceb

This code allocates for `node->next[char_to_ascii]` if the letter's node has not been allocated yet. This means that if a letter is not used in this word, that node is not allocated. This code dynamically allocates AlphaTrieNodes instead of allocating all of them at once.

♡ ...

 **Anonymous Kudu** 2y #883cbd ✓ Resolved


SU22-MT-Q4.3 -- Why do we need to save the `s` registers in the caller (`add_even_numbers`)? I thought the saving of `s` registers are handled by the callee (`is_prime`). I always get confused by this.

♡ ...

 E **Erik Yang** TA 2y #883cbf

If you ever use `s` registers, you need to save them in that function. If you call another function, you assume that that function saves the `s` registers it uses as well

♡ 1 ...

 **Vishnu Suresh** 2y #883cbc ✓ Resolved

Sp22-MT-Q4.2 Do `jal malloc` and `jal ra malloc` do the same thing. Also, in the same vein, do `jalr ra, s1, 0` and `jalr s1, 0` do the same thing?

♡ 1 ...

 E **Erik Yang** TA 2y #883cbe

Yes it jsut takes what `ra` is currently.

♡ ...

 **Anonymous Gorilla** 2y #883caf ✓ Resolved

sp22-MT-Q2.3: For the exponent why is $3 - (-3)$ and then in Q2.4 the exponent is $6 + (-3)$. Why was there a subtraction in the first one and addition in the second one. Then shouldn't the first one be $3 + (-3)$?

Also for Q2.5: For finding the smallest exponent why did is there a $+1$ at the end. $0 + (-3) + 1$

♡ ...

 **D Darwin Zhang** 2y #883cbb 

In 2.3, we found our exponent of 1100 ($=1.100 * 2^3$) is 3, then we subtract the bias (-3) to get the exponent of 6 ($=0b110$) to represent in the IEEE-754 system.

Conversely, in 2.4, when we have the largest exponent of 6 ($=0b110$) in the IEEE-754 format, we want to add the bias (-3) to get the actual exponent of the largest number that we want to represent, which 3 in this case.

In 2.5, we want to add one to the smallest exponent bit because the exponent bits are all zero, so we want to add one to the smallest exponent. $(-1)^{\text{sign}} * 0b0.SSSS * 2^{(0bXXX + (-3) + 1)}$, also known as a denormalized number.


 ...

 **Anonymous Lyrebird** 2y #883ddc

How do we know when to add bias and when to subtract bias?

 ...

D Darwin Zhang 2y #883dea

 Replying to Anonymous Lyrebird

Add bias to exponent when converting from IEEE-754 (0bXXXX...) to binary representation: $0bXXXX... + (\text{bias})$

Subtract bias from exponent when converting binary ($1.... * 2^X$) to IEEE-754: $X - (\text{bias})$

Keep in mind that in the given question, bias is a negative number.

 ...


 **Anonymous Lyrebird** 2y #883eba

 Replying to Darwin Zhang

Thank you!

 ...



Stella Kaval 2y #883cad 

SU22 MT1 Q3.3: I'm not quite sure how to calculate how many numbers in the range [16, 64) can be represented by floating point. What is the strategy for using step size for these types of problems?

 1 ...

 **R Rosalie Fang** ADMIN 2y #883cea

The easiest way is probably to convert 16 to floating point, then convert 64 to floating point. The number of bit combinations between those 2 binary representations will be the number of representable values because the numbers are in sequential order.

 ...



Anonymous Spider 2y #883bfd 

SP22-MT-Q3

Is the content of Q3.3 to Q3.5 in scope? If so, where did we learn about this behavior of define statements (a reference to the section in lecture would be nice, as I don't recall where we learned about it!)

 ...

 **J Jero Wang** ADMIN 2y #883bfe

Yes, lecture 3 slide 13

♡ ...



Anonymous Spider 2y #883cae

Where did we learn specifically that `#define` is effectively a find and replace? I would like to view that section just to see if there are any other unique properties of `#define` I should know about.

♡ ...



Jero Wang ADMIN 2y #883cba

↩ Replying to Anonymous Spider

There's an example on slide 14 of the same lecture.

♡ 1 ...



Anonymous Kouprey 2y #883bfc

✓ Resolved

SU22-MT-Q4.2

```
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:

```
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
    srli t2, t1, 1
    slli t2, t2, 1
    bne t1, t2, pass
    add t0, t0, t2
pass:
    addi a1, a1, -1
    addi a0, a0, 4
    j loop
end:
    mv a0, t0
```

For the first code solution, it should be 8 lines (-32 bytes offset). In other words, -36 bytes offset is referring to the alternative code solution, right?

♡ ...

Jero Wang ADMIN 2y #883bff

No, it says

Assume that every line in the above code is filled with exactly one instruction

So it would always be -36.

♡ ...

Anonymous Kouprey 2y #883cab

I see, so we are not counting labels including "skip" and "loop" since they are not stored in memory, right?

♡ ...

Jero Wang ADMIN 2y #883cac

← Replying to Anonymous Kouprey

Yes, those are labels and do not take up space in memory.

♡ 1 ...

Anonymous Toad 2y #883bfa

✓ Resolved

For SU22 Q2.4 :

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.

Why the we encounter the same problem as the `malloc` case, where not all pointers in the new block of memory have been allocated? Didn't we get access to more memory and set all of it null pointers? Or is it because in the `malloc` case we overwrote our pointers by setting them to null?

What does it mean by "not all the pointers in this new block of memory have been allocated"?

♡ ...

J **Jero Wang** ADMIN 2y #883caa

Not sure what the solution itself was referring to here, but I believe intended solution when we wrote this question was that at this point, `node` (the old pointer before `realloc`) has already been free'd, therefore accessing `node->next[j]` is an invalid memory access.

♡ ...

Anonymous Elephant 2y #883beb ✓ Resolved

SP22-MT2-Q5, for 5.2, is "output of A -> Adder -> Input of A" also the longest path?

♡ ...

E **Erik Yang** TA 2y #883bef

i think you mean 4.2? then yes, output of A works as well!

♡ ...

Anonymous Whale 2y #883bea ✓ Resolved

For SP 22 Midterm, 4.2. Why don't we consider the hold time as well. The hold time represents the amount of time the input needs to be stable after the positive edge of the clock, so it's still important. Is it cause the `clk-to-q` is already greater than the hold time?

Q4.2 (4 points) Assume that the circuit has the following delays:

Register clk-to-q time	3ns
Register setup time	2ns
Register hold time	1ns
Adder propagation delay	4ns

Wires are assumed to have no propagation delay. What is the minimum clock period needed for this circuit to have the same behavior as in Q5.1?

Solution: 9 ns

The longest path between sequential logic blocks (blocks that depend on the clock; in this case, just the registers) is the path from the output of Register B, through the adder gate, and into the input of Register A.

How long does it take for a signal to travel through this longest path? From the positive edge of the clock, we have to wait 3 ns (`clk-to-q` time) for Register B's input to appear at its output. Then, we have to wait 4 ns (adder delay) for the signal to travel through the adder. Finally, when the signal arrives at the input Register A, we have to wait 2 ns (setup time) before the next positive edge of the clock. In total, our shortest clock period is $3 + 4 + 2 = 9$ ns.

♡ ...

E **Erik Yang** TA 2y #883bec

min clock period = critical path = setup time + clk to q + longest combinatorial path

♡ ...

Anonymous Whale 2y #883bed

could you explain how we came up with that equation/why is that the case? I know it was in the hw guide but I didn't really understand it

♡ ...

E Erik Yang TA 2y #883bee

↳ Replying to Anonymous Whale

yeah so your min clock period has to be \geq the longest time it takes for an input to travel to output bc if it was any shorter, then that path would not be valid. So we want to ideally find the longest possible path there is. To do that, you need 3 things: clk to q tells you how long input is translated to output in a register, you need the path with longest logic delays, and you need the setup time because you need to know how long it takes for that output to be stable in the "second" register. All together, this determines how long it is necessary for a clock period to at least be

♡ ...

Anonymous Elephant 2y #883bde ✓ Resolved

SP22-MT2-Q1, for 1.2, does it matter for different bits of systems (i.e., 32-bit vs 64-bit system)?

♡ ...

N Nikhil Kandkur TA 2y #883bdf

The size of each data item (except for chars) does matter when it comes to different bit system, but the alignment scheme remains the same.

♡ 1 ...

Anonymous Salamander 2y #883bdb ✓ Resolved

for SU22 4.2, I'm confused on how -36 translates to 0xFDDFF06F in hex. I got something different and I'm having trouble understanding how I'm supposed to end up with the correct answer.

♡ ...

E Eric Kusnanto TA 2y #883bdc

Just to clarify, we're translating `j loop` with -36 as the byte offset. Note that in J-type instructions, we have an implicit zero that isn't stored in the instruction format.

♡ ...

Anonymous Salamander 2y #883bdd

oh I get it now. I didn't know we were translating the entire instruction and not just the offset. thank you

♡ 1 ...

Anonymous Pigeon 2y #883bcd ✓ Resolved

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.

SP22-MT-Q1.1

Does the integer 0xDEADBEEF stay 0xDEADBEEF solely because it's an integer?

♡ ...

E Eric Kusnanto TA 2y #883bda

No, endianness is simply how each byte is stored in memory within the 4-byte block. The example they provide is that when we want to store `int x = 0x00000001`, we don't write `int x = 0x01000000`.

♡ ...

 **Anonymous Salmon** 2y #883dfe

I thought we would want to write it as `0xDEADBEEF` where the `D` at the start is at the largest memory address and the `F` at the end is at the smallest memory address, so if we want to write it in little endian we would want to write it as `0xFEEDBAED`. Where does my thinking go wrong?

♡ ...

 **Anonymous Mink** 2y #883bcc ✓ Resolved

Can someone explain **exactly** how the solution for Summer 2022 4.2 was calculated? The part about the byte offset from `j` loop?

Here's what I calculate:

```
add_even_numbers:
addi t0, x0, 0
loop:
beq a1 x0 end # 1 instruction
lw t1 0(a0) # 1 instruction
andi t2 t1 1 # 1 instruction
beq t2 x0 pass # 1 instruction
add t0 t0 t1 # 1 instruction
pass:
addi a0 a0 4 # 1 instruction
addi a1 a1 -1 # 1 instruction
j loop # 7 instructions away so offset = -28
end:
ret
```

OR

```
add_even_numbers:
addi t0, x0, 0
loop:
beq a1, x0, end # 1 instruction
lw t1 0(a0) # 1 instruction
srli t2, t1, 1 # 1 instruction
slli t2, t2, 1 # 1 instruction
bne t1, t2, pass # 1 instruction
add t0, t0, t2 # 1 instruction
pass:
addi a1, a1, -1 # 1 instruction
addi a0, a0, 4 # 1 instruction
j loop # 8 instructions away so offset = -32
end:
mv a0, t0
```

♡ ...

 **Yile Hu** TUTOR 2y #883bce

The solution assumes you use all lines in 4.1

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
```

♡ ...



Anonymous Hare 2y #883bbf

✓ Resolved

SU22-MT-Q2.1

For 2.2 and 2.1, can we do modular division by 97 instead of subtracting by it? Also, why are we casting it to an int?

♡ ...



E Eric Kusnanto TA 2y #883bca

Sure. We cast to an int from a char so that we can properly index into the array of Trie nodes. IIRC, we gave partial credit for not explicitly casting.

♡ ...



Anonymous Sparrow 2y #883def

So, if we have a pointer to a character array and we cast it with (int), it will automatically convert characters into their ASCII integer number?

♡ ...



Anonymous Human 2y #883bbd

✓ Resolved

SU22-MT-Q4.2

Do labels get included when calculating offset? Or do we straight up just count the number of instructions between the jump call and the label its jumping to.

♡ ...



S Sam Xu TUTOR 2y #883bbe

Labels are skipped

♡ ...

M

Melanie McKune 2y #883bad ✓ Resolved

Q2.3 (3 points) Convert -12 to its floating point representation under this floating point system. Express your answer in binary, including the relevant prefix.

Solution:

$$12 = 0b1100 = 0b1.1000 \times 2^3$$

$$\text{Significand} = 0b1000$$

$$\text{Exponent} = 3 - (-3) = 6 = 0b110$$

$$\text{Sign bit} = 1 \text{ (negative)}$$

$$0b11101000 \rightarrow 0b11101000$$

For Spring 22 MT 1, why is there an extra 0 on the line $12 = 0b1100 = 0b1.1000 \times 2^3$? (the zero added from shifting the decimal over to only have one 1 before decimal point.)

♥ ...

L

E Erik Yang TA 2y #883bae

It's jsut to show that there's 4 significant bits

♥ ...

M

Melanie McKune 2y #883bac ✓ Resolved

For Spring 22 MT 1, question 1.1, why is 0xDEADBEEF stored as 0xDEADBEEF in little endian?

♥ ...

L

S Sam Xu TUTOR 2y #883bbc

In little endian, the lower byte will be stored in lower address. Therefore, 0xDEADBEEF will be stored as 0xDEADBEEF. In big endian, the lower byte will be in higher address. In big endian, 0xDEADBEEF will be stored as 0xEF BE AD DE.

♥ ...

Anonymous Kouprey 2y #883baa ✓ Resolved

SU22-MT-Q2.5

Q2.5 (4 points)

```
ASCIITrieNode* convert(AlphaTrieNode* node) {
    if (node == NULL) {
        return NULL;
    }
    ASCIIITrieNode* 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.

For the first scenario where "realloc does not change the pointer to node," the node pointer is not freed, right, so this could be another potential problem?

♡ ...



Jero Wang ADMIN 2y #883bab

The handling of free'ing the old node is dependent on what `realloc` does, and if `realloc` doesn't move the block of memory to a different address, you don't need to free the old pointer, since it is the same as the new pointer.

♡ ...



Anonymous Cheetah 2y #883afe

✓ Resolved

Q2.6 (3 points) Translate the following RISC-V instruction into its corresponding hexadecimal value.

```
ori t6 s0 -12
```

```
Solution: FF446793
opcode = 0b001 0011
funct3 = 0b110
rd = t6 = x31 = 0b11111
rs1 = s0 = x8 = 0b01000
imm = -12 = 0b1111 1111 0100
0b111111110100 01000 110 11111 0010011
0b1111 1111 0100 0100 0110 1111 1001 0011
0xFF44 6F93
```

This is SP22-Midterm-Q2.6

Is '7' in the solution a typo? Shouldn't it be F?

♡ ...



Jero Wang ADMIN 2y #883aff

Yeah, should be F, thanks for the catch.

♡ 1 ...



Anonymous Porcupine 2y #883afb ✓ Resolved

Su22-MT-Q4.2: how is the immediate calculated? i thought there are 11 lines before j loop, how are there 9? i got the offset to be -44.

♡ ...



Sam Xu TUTOR 2y #883afd

Lines with label are skipped.

♡ ...



Anonymous Red panda 2y #883aed ✓ Resolved

Su22-MT-Q 5.3

how did they get 1 at the end?

Solution: $\sim(\sim(A + D) * \sim(C + \sim B)) + B$
 $(A + D) + (C + \sim B) + B$
 $A + D + C + (B + \sim B)$
1

♡ ...



Erik Yang TA 2y #883aee

$B + \sim B = 1$ and anything + 1 equals 1

♡ ...



Anonymous Whale 2y #883aeb ✓ Resolved

For SU 22 4.3, why is option F wrong. Since our function still depends on a, shouldn't we preserve its value by storing it on the stack. In general, is it bad calling convention to store both s and a registers on the stack? Is F wrong since it says "at the beginning of the function".

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

Solution: (A): Calling convention doesn't need you to keep track of how many times a function is called. The functionality of this program also doesn't require you to keep track of how many times the `is_prime` function is called.

♡ ...



E Erik Yang TA 2y #883aec

Yeah you only need to store a regs before you call a function and want to use those a regs again

♡ ...



Anonymous Whale 2y #883bbb

Is there ever a case where we could store `t` regs on the stack?

♡ ...



Anonymous Eagle 2y #883aea

✓ Resolved

SU22-MT-Q4

Q4.1

Could you explain the code in the two possible solutions that enables us to check if a number is even or odd? I was struggling to write this part of the code.

Q4.2

I got an offset of -44. When calculating offset, do we only consider the lines with instructions and ignore the lines with labels?

Thanks!



Sam Xu TUTOR 2y #883afc

4.1.A If a binary number is odd, its last bit must be 1. Otherwise, all even number has 0 as last bit. We can check if a number is odd by checking last bit. Solution A check last bit of $t1$ by using `andi t2, t1, 1`. That is because $0 \text{ AND } b$ (b can be 1 or 0) will be 0, $1 \text{ AND } b$ is b . If $t1$'s last bit is 0, $t1 \text{ AND } 0b0000...001$ is equal to $0b000...00$. if $t1$'s last bit is 1, the result is $0b00...01$.

4.1.B Solution B uses `srl` $t2, t1, 1$ and `sll` $t2, t2, 1$. First, it left shift $t1$ by 1, which basically takes the last bit off from $t1$. Then we right shift it by 1, which will put a 0 at the left end of $t1$. As a result, these two instructions replace the last bit of $t1$ by 0, regardless its original last bit. If $t1$ is even, these instructions will not change its value, because it has a 0 as last bit. If $t1$ is odd, these instructions change its value by changing last bit from 1 to 0.

4.2 Yes, we just consider lines with instructions



Anonymous Hummingbird 2y #883cda

For 4.1 solution A, if $t1$'s last bit is 0, then $t1$ must be even. But $t2 = t1 \& 1 = 0$, which will lead to the branch to pass by `beq t2, x0, pass`. Shouldn't this be the other way around and use `bne t2, x0, pass` instead?



Sam Xu TUTOR 2y #883cfb

↩ Replying to Anonymous Hummingbird

If $t2=0$, it means $t1$ is even and we should pass it, without put it into the sum



Anonymous Seahorse 2y #883ddd

↩ Replying to Sam Xu

But the question asks to return "the sum of all even numbers in the array". So shouldn't $t1$ be added to the sum (when $t2=0$), not passed?



Anonymous Eel 2y #883adf

✓ Resolved

SP22-MT-Q4.2

Question 1: for line 10, why we cannot use "**jal malloc**" as we usually did before?

Question 2: For line 13, why we are using `jarl` rather than `jar`?

Question 3: I don't understand how we represent a Struct in RISC-V, specifically in this question, the Vector. It seems like we are treating Vector as an array, and we use offset to get `vector.x` and `vector.y`, but why can we do that?

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 ...



Sam Xu TUTOR 2y #883aef

1. Jal label is a pseudo instruction and is same as jal ra label in the project. Since this pseudo instruction is not on reference sheet, it is better to use jal ra label

2. jalr rd rs1 imm will jump to the instruction at $rs1+imm$, but jal rd label will just jump to label. Transformer function takes a function f as an argument. Remember that arguments are stored in register, so f in this function is in $a1$ register. In transformer function, when we want to use f function, we need to jump to it. Jalr can jump to a register's value, but jal can only jump to label.

3. In RISC-V, yes, we are treating struct like "array". When we initialize a struct in RISC-V, we basically just have a reference pointing to a memory block that will store the struct. When we want to read/change the value in a struct, we load/store value at its reference. Each variables' address in a struct are calculated by using struct's address. For instance, the second variable of a struct is usually at struct's address + 4, etc.

♡ 1 ...



Anonymous Echidna 2y #883aca

✓ Resolved

SU22-MT-Q1.1:

Is the question basically saying that endian-ness is something taken care of in the "background" (i.e. it's just how they are stored in memory or seen by the registers), which is why it doesn't really matter when writing C code?

♡ ...

J Jero Wang ADMIN 2y #883adb

Yeah, endianness is about how data is stored in memory, the actual data being stored is the same so that's why it doesn't matter when you're writing C code.

♡ 2 ...

Anonymous Echidna 2y #883abe ✓ Resolved

SP22-MT-Q4.1

Shouldn't the function *f* be dereferenced before being applied here? Because *f* is a *pointer to a function* if I am understanding correctly.

♡ 1 ...

J Jero Wang ADMIN 2y #883adc

That step is optional for functions, C dereferences it automatically.

<https://stackoverflow.com/a/7519008>

♡ 1 ...

Anonymous Eel 2y #883abd ✓ Resolved

SU22-MT-Q4.2

Hi, I don't know how to deal with ""j label". On the reference card, the only J type is "jal", can anyone explain how to get this answer?

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

♡ ...

Y Yile Hu TUTOR 2y #883abf

`j label` is a pseudo instruction for `jal x0, label`. See #883bc for next steps.

♡ 1 ...

V Vasti Orbach 2y #883baf

Since the offset is -36. Do we use two's complement or sign magnitude to convert the decimal to binary.

♡ ...

Y Yile Hu TUTOR 2y #883bba

↩ Replying to Vasti Orbach

two's complement

♡ ...

Anonymous Eel 2y #883abc ✓ Resolved

SU22-Q4.1

After the addi instruction, I thought if a number is odd (say 5), then addi will output 1. But why the solution is doing "beq t2 x0 pass"? This instruction means, if the number is even, we then skip the number. But I think we want to skip odd numbers. If there any problems in the solution?

Also, as mentioned by another student, the solution didn't mv a0 to t0. Then how we know the a0 contains the final answer?

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
```

♡ ...

 **Erik Yang** TA 2y #883acc
#883bf

♡ ...

 **Anonymous Eel** 2y #883add

Thanks! But I am still confused about my question 2. Should we include "mv a0 to" in our answer?

♡ ...

 **Jero Wang** ADMIN 2y #883ade

↩ Replying to Anonymous Eel

I think we're missing a line in our solutions, the "End" label should have mv a0 t0 and ret.

♡2 ...

 **Anonymous Antelope** 2y #883aae

✓ Resolved

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

SP22-MT-Q4.1

A couple of content questions. Why does transform need a "*" next to it? Why is the return value in the function header Vector and not Vector*? Lastly, the "*" on the input args are put there so we

are able to modify them, correct?



E Erik Yang TA 2y #883aaf

i think you're referring to *? You need that because the return value should be a Vector pointer. newVector is a vector pointer so that's correct. The * on the input args means the parameters passed in are pointers.



Anonymous Whale 2y #883fe

✓ Resolved

for SU 22 1:

It says 8 bit two's complement numbers can be represented up to $2^8 - 1$, isn't that 255, not 127? What is the actual range for two's complement and sign-magnitude numbers then?

Q1 Potpourri

(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.



E Erik Yang TA 2y #883aba

the range for 2's complement is $[-2^{(b-1)}, 2^{(b-1)} - 1]$

range for signed is $[-2^{(b-1)} - 1, 2^{(b-1)} - 1]$. I believe the 8 is a typo



Anonymous Red panda 2y #883ee

✓ Resolved

sp22-MT-Q2.4

why don't we make the exponent 0b111 instead of 0b110 since we're trying to maximize the number?

♡ ...



Anonymous Porcupine 2y #883ef

also, why is the form $0b1.1111 * 2^3$? Shouldn't it be $0b0.1111 * 2^3$, to make the sign bit 0?

♡ ...



Jero Wang ADMIN 2y #883fc

0b111 is NaN/infinity, so not a number.

1.1111 is referring to the implicit 1 and the mantissa, the sign bit is separate and is positive (0) in this case.

♡ ...



Anonymous Porcupine 2y #883aab

↩ Replying to Jero Wang

i'm still a little confused, where did the extra 1 come from, bc aren't there just 4 1's from the mantissa? so I don't get why there are 5 1's

♡ ...



Erik Yang TA 2y #883aac

↩ Replying to Anonymous Porcupine

normal numbers have an implicit 1, so you need to make the number have a leading 1, followed by the mantissa bits

♡ 1 ...



Anonymous Panther 2y #883ed

✓ Resolved

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.

Does `realloc` set garbage to the new space?

When will `realloc` change the pointer to original node and when will not?

♡ ...

 **Jero Wang** ADMIN 2y #883fb

Realloc might have garbage in the newly allocated space, and it may or may not change the address, and there's no way of predicting this.

♡ ...

 **Anonymous Whale** 2y #883fd

why is it unpredictable?

♡ ...

 **Anonymous Porcupine** 2y #883ec ✓ Resolved

SP22-MT-Q1.8: what is the "implicit 0th index bit of 0"? Why is it included/ is it indicated on the references sheet?

Q1.8 (1.5 points) TRUE or FALSE: Branch instructions can represent a larger immediate value than I-type instructions.

TRUE

FALSE

Solution: True; Branch instructions encode 12 bits worth of immediate, but we include an implicit 0th index bit of 0, bringing up the immediate to be 13 bits. I-type instructions encode only 12 bits, without any implicit bits.

♡ ...



Jero Wang ADMIN 2y #883fa

There's an implicit 0 for branch immediates because the offset for branches will always be a multiple of 2. This is indicated on the reference sheet by the fact that the branch instruction does not encode the 0th bit of its immediate in the instruction itself.

♡ ...



Anonymous Salmon 2y #883eab

Do we have any other implicit bit values? I see J and B instructions omit the 0th bit but use the 1st bit, so can we assume both of these types have an implicit 0? Are there any other types with implicit values?

♡ ...



Anonymous Whale 2y #883df

✓ Resolved

For SU 22 2.3

Isn't this also wrong because we just only freed TrieNode struct? Don't we also need to free all the pointers in the next[] array?


```
-----  
} ASCIITrieNode;
```

We would like to write a function that converts a trie of `AlphaTrieNodes` to a trie of `ASCIITrieNodes`. The function should also free all `AlphaTrieNodes` in the process. You may assume that all `AlphaTrieNodes` are properly initialized.

Below, we have 3 implementations of this conversion function. For each implementation, determine whether or not it is a valid implementation. If the implementation is not valid, please provide a brief explanation (10 words or fewer).

Q2.3 (4 points)

```
ASCIITrieNode* convert(AlphaTrieNode* node) {  
    if (node == NULL) {  
        return NULL;  
    }  
    ASCIITrieNode* new_node = malloc(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 doesn't work because `malloc` allocates memory for the `ASCIITrieNode`, but does not initialize that memory. In the for loop, we only fill in 26 pointers in the `next` array of 256 pointers. The rest of the pointers are uninitialized (garbage), but they should be `NULL` since there are no nodes for the corresponding characters yet.

♡ ...



E Erik Yang TA 2y #883ea

no bc these are all recursive calls, so eventually you will free all the nodes

♡ ...



Anonymous Human 2y #883cf

✓ Resolved

SU22-MT-Q2.4

Q2.4 (4 points)

```
ASCIITrieNode* convert(AlphaTrieNode* node) {
    if (node == NULL) {
        return NULL;
    }
    ASCIIITrieNode* 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`.

I thought `calloc` zeroes out the memory. Is this the same as `NULL`?


♡ ...

 E Erik Yang TA 2y #883da
ptrs to null, ints to 0

♡ ...


 Anonymous Gazelle 2y #883acd
If you `calloc` a struct, does all of the element within the struct get initialized? If so, what happens to the 'last' element, which is a boolean?

♡ ...


 Anonymous Human 2y #883ce ✓ Resolved
SU22-MT-Q1.6

Would the correct answer be the loader?

♡ ...


 E Erik Yang TA 2y #883db
loader initializes machine registers so yes

♡ ...

 Anonymous Human 2y #883cd ✓ Resolved
SU22-MT-Q4.1

What is the difference between `ret`, `jr ra`, `mv a0 t0` when exiting a function in risc-v? If we don't do `mv a0 t0` in the first solution, how do we have the sum stored in register `a0` for the output?

♡ ...

 E Erik Yang TA 2y #883dc
`ret` is a psuedo instr for `jalr`, `jr ra` jumps to the return address, `mv a0 t0` moves `t0` to the return value

♡ ...



Anonymous Human 2y #883dd

how does ret move the correct output into a0?

```

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

```



Rosalie Fang ADMIN 2y #883eb

↩ Replying to Anonymous Human

ret does not move the correct output into a0. In this example, t0 is the sum, so if we don't have mv a0 t0 at before ret, a0 will not be the running sum when the function returns.



Anonymous Human 2y #883aad

↩ Replying to Rosalie Fang

ok that's what i was confused about since this is the official solution?



Anonymous Human 2y #883cc

✓ Resolved

SU22-MT-Q5.2

Solution: $(A + B) + ((\sim A * \sim B) * \sim C)$ (direct translation of circuit)
 $(A + B) + (\sim(A + B) * \sim C)$
 $A + B + \sim C$

How do we go from step 2 to step 3?



E Eric Kusnanto TA 2y #883de

see #883e



Anonymous Whale 2y #883ca

✓ Resolved

For SU 22 2.2, where does the inserting take place. We calloc the size of 1 to node->next, but we never actually put char_to_ascii anywhere.

Q2.2 (8 points) `insert` takes in a trie root node (`node`), and a pointer to a string (`word`). It should insert the word into the trie.

```
1 void insert(AlphaTrieNode* node, char* word) {
2     for (int i = 0; i < strlen(word); i++) {
3         int char_to_ascii = (int) word[i] - 97;
4         if (node->next[char_to_ascii] == NULL) {
5             node->next[char_to_ascii] = calloc(1, sizeof(AlphaTrieNode));
6         }
7         node = node->next[char_to_ascii];
8     }
9     node->last = true;
10 }
```

♡ ...

 **Anonymous Human** 2y #883cb

I don't think we need to. If you look in the examples, we represent a character being "stored" by setting the respective `AlphaTrieNode` in the `next` array to the allocated pointer. If there is no character, the pointer at that index in the `next` array is null.

♡ ...

 **Jason Lee** 2y #883bc ✓ Resolved

SU22-MT-Q4.2

I am not sure how to convert "j loop" to hexadecimal. Any pointers on how to get started? Very confused.

♡ 1 ...

 **Yile Hu** TUTOR 2y #883be

Start by taking a look over the reference sheet on what is the underlying instruction of `j loop`. When we translate `j loop` into machine code, `loop` is translated in to an offset in byte which will be added to the `pc` register. How many instructions are between the instruction `j loop` and the `loop` label? How many bytes does this distance equal to?

♡ ...

 **Anonymous Squid** 2y #883bb ✓ Resolved

For SU22 MT Q4.1, we want to add all the even numbers in an array. The loop in the first solution looks like this:

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
```

I'm a bit confused as to why we have 'beq t2 x0 pass' instead of 'bne t2 x0 pass'. Testing a number like 1 for t1 (which is odd and we would want to skip over it), 'andi t2 t1 1' should result in t2 being set to 1 (because '1 and 1' = 1). Then when we check if t2 == 0, we find that it's false, so we don't jump to pass (this is looking at 'beq t2 x0 pass'). Then we add the current number t1 (which in this example is 1, an odd number) to the running total (t0).

I think this would be the case for all odd numbers, which makes me think that the provided solution adds all the odd numbers instead of the even ones. Is there something wrong with my thought process?

Edit: This is the full solution if this helps:

```
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
```

♡ 1 ...

 **Anonymous Guanaco** 2y #883bd

I had the same question! I also where are they setting a0 to be equal to t0 the running sum?

♡ ...

 **Erik Yang** TA 2y #883bf

You're right it should be bne - good catch!

♡ 1 ...

 **Anonymous Whale** 2y #883af ✓ Resolved

For SP22 - 2.5, why do we add 1 to -3 to make it -2. In the video guide of hw, they showed how the smallest denormal number can have an exponent of 000. Conversely, why does the max positive number have an exponent of 2^6 instead of 2^7

Q2.5 (3 points) What is the smallest positive number that can be represented by this system?

Express your answer as an odd integer multiplied by a power of 2.

Solution:

Smallest significand = 0b0001

Smallest exponent = 0b000 = $0 + (-3) + 1 = -2$

$0.0001 \times 2^{-2} = 1 \times 2^{-6}$

E Erik Yang TA 2y #883ba
It's denorm. The exponent would be $\text{exp} + \text{bias} + 1$
♡ ...

Anonymous Human 2y #883ab ✓ Resolved
SP22-MT-Q4.2

In line 9 of the solution, $\text{li a0}, 8$, how do we know to malloc out 8 bytes for the Vector?
♡ ...

Y Yile Hu TUTOR 2y #883ac
It is the size of a single Vector struct
♡ ...

Anonymous Human 2y #883ad
How do we know that a single Vector is 8 bytes? Is it cause it stores 2 ints?
♡ ...

Y Yile Hu TUTOR 2y #883ae
↩ Replying to Anonymous Human
yep
♡ ...

Anonymous Echidna 2y #883ace
↩ Replying to Yile Hu
Just a quick related follow-up here. Malloc doesn't return anything, right? So is a0 just garbage after the malloc call?
♡ ...

Y Yile Hu TUTOR 2y #883ada
↩ Replying to Anonymous Echidna
malloc returns the pointer to allocated memory
♡ 1 ...

Anonymous Scorpion 2y #883e ✓ Resolved
SU22-MT-Q5

For Q5, part 2, in the solutions, how did we get from $(A + B) + (\sim(A + B) * \sim C)$ to $A + B + \sim C$?
Thanks!
♡ ...

C Catherine Van Keuren TA 2y #883aa
It looks like they used an identity sometimes called the uniting theorem which says that :

$$\overline{AB} + A = A + B$$

This theorem can be proven through DeMorgan's as seen by this question off of the logic discussion in sp22

2.1 Use multiple iterations of De Morgan's laws to prove the identity $\bar{A} + AB = \bar{A} + B$.

$$\begin{aligned}\bar{A} + AB &= \overline{A\bar{A}B} \\ &= \overline{A(\bar{A} + B)} \\ &= \overline{A\bar{A} + AB} \\ &= \overline{AB} \\ &= \bar{A} + B\end{aligned}$$

I don't think the identity was explicitly mentioned in lecture, but it might be a useful one to know/have on your cheat sheet!

♡ ...



Anonymous Scorpion 2y #883d

✓ Resolved

SP22-MT-Q4.1

To call function pointed to by f, don't we need to dereference f? The solution directly used f. When do we need to dereference function pointers passed in as arguments? Thanks!

♡ 2 ...



Catherine Van Keuren TA 2y #883f

You don't need to dereference function pointers. Both ways, (*f)(self->x) and f(self->x), exhibit the same behavior, and both would be considered correct.

♡ ...



Anonymous Pigeon 2y #883a

✓ Resolved

SP22-MT-Q2

For Q2.4, how does the 0.1 (binary) after the 15 convert into 0.5 (decimal)?

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 × 2³ = 0b1111.1 = 15.5

♡ ...



E Eric Kusnanto TA 2y #883b

any bits to the right of the binary point in 0b1111.1 will be negative powers of 2, starting with 2⁻¹ = 0.5

♡ ...