

! This class has been made inactive. No posts will be allowed until an instructor reactivates the class.

note @909

427 views

Actions

[Past Midterms] 2021

You can find the past exams here: <https://cs61c.org/sp22/resources/exams/>

Spring 2021 final walkthrough

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: **SP21-MT1-Q1**, or **SU21-MT2-Q3**

exam

exam/midterm

good note | 0

Updated 5 months ago by Jerry Xu and 2 others

followup discussions, for lingering questions and comments

Resolved Unresolved @909_f1



Anonymous Beaker 7 months ago

For SP21-MT2-Q1b, the solution says that bias notation is the same as unsigned notation but with an added bias. What about bias for two's complement? Would the range be the fourth choice or does it end up being the same?

(b) (1.0 pt) What is the range of numbers that can be represented in biased notation where N is the number of bits in the binary representation and B is the bias (B is a negative number)?

- $[-2(n-1) + B, 2(n-1) - 1 + B]$
- $[B, 2(n) - 1 + B]$
- $[-2(n-1) - B, 2(n-1) - 1 - B]$
- $[-2(n-1) - 1 + B, 2(n-1) - 1 + B]$
- $[-B, 2(n) - 1 - B]$
- $[-2(n-1) - 1 - B, 2(n-1) - 1 - B]$
- $[-B, 2(n-1) - 1 - B]$
- $[B, 2(n-1) - 1 + B]$

Bias notation is the same as unsigned notation but with an added bias. The range of values that can be represented by an unsigned number is $[0, 2(n) - 1]$. To determine the range of values that can be represented in biased notation, we just add the bias to the unsigned range giving us $[0 + B, 2(n) - 1 + B]$.

helpful! | 0



Peyrin Kao 7 months ago

If I'm understanding your question correctly, there's no such thing as bias with two's complement. By definition, bias is unsigned notation but with a bias added to every number.

[good comment](#) | 1



Anonymous Beaker 7 months ago

I think I confused representing a two's complement number in bias notation with the definition of bias notation, but I get it now, thank you!

[helpful!](#) | 0



Anonymous Gear 7 months ago

Why is it $2(n) - 1$ as opposed to $2^n - 1$?

[helpful!](#) | 0



Adelson Chua 7 months ago

They probably intended to interpret $2(n)$ as 2^n . They just can't write exponents properly for some reason, haha.

Resolved Unresolved [@909_f2](#)



Elizabeth (Vi) Nguyen 7 months ago

FA21-MT Was there no SDS/Timing section in fa 21 mt?

Reply to this followup discussion

[helpful!](#) | 0



Jero Wang 7 months ago

Yep. The midterm only covered materials up to the CALL lecture last semester due to timing.

[good comment](#) | 0

Reply to this followup discussion

Resolved Unresolved [@909_f3](#)



Anonymous Scale 7 months ago

For SP21-MT-Q7B, can someone explain the bloom filter code, especially when `data[n/8]` is used? We are having trouble understanding what the bloom filter is supposed to do.

[helpful!](#) | 0



Peyrin Kao 7 months ago

Honestly, I think this question was kind of poorly worded, and we tried our best to make this semester's exam clearer in terms of what you need to solve.

That said, the line you're asking about is described in the comment: "Now we set the n'th BIT in the bloom filter to 1". C indexes in bytes, and 1 byte = 8 bits, so `data[n/8]` is getting the byte where the n'th bit is located.

[good comment](#) | 2

Reply to this followup discussion

Resolved Unresolved [@909_f4](#)



Anonymous Mouse 7 months ago

Can someone explain this part to 2.9 in Fall 2021. Why is the system storing the original string in big endian when they said that the code was running on little-endian?

Q2.9 (2 points) If this code was run on a little-endian system, what would `((uint32_t*) str1)[2]` evaluate to? Express your answer in hexadecimal, with the necessary prefix. Note that `uint32_t` refers to an unsigned 32-bit integer.

Solution: `0x00646C72`

This question requires you to think about endianness and how memory would be interpreted if casted to a different type.

First, note that `str1` contains the address of the bytes `Hello World`. These bytes are stored contiguously in memory; `H` is at the lowest address, and `d` is at the highest address. A null byte is stored immediately after `d` in memory.

helpful! | 0



Anonymous Gear 7 months ago

Where is the fall 2021 exam? I don't see it on the website

helpful! | 0



Peyrin Kao 7 months ago

The fall 2021 exam is linked on the website (it's listed as midterm 2 because we only had one midterm).

I don't see where the question mentions a big-endian system. Endianness is only relevant for storing the bytes within a word (e.g. addresses, integers). Strings are always stored with the first character at the lowest address and the last character at the highest address, regardless of endianness.

good comment | 3



Anonymous Poet 3 7 months ago

Once we change this to be a 32b unsigned int, endianness actually affects the way the value is interpreted right? Also, does endianness go beyond just word sizes? In HW 2 we had an unsigned 64b integer type which we understood to be influencing the entire 8 bytes as a whole and not just on a word-basis?

helpful! | 0



Anonymous Poet 3 7 months ago

e.g. storing `0xDEADBEEF00112233` would be in memory as:

`0x00: 0xEFBEADDE`

`0x04: 0x33221100`

helpful! | 0



Anonymous Poet 3 7 months ago

Further, are half-words stored differently depending on endianness? Sorry for the multi-post as I can't edit.

helpful! | 0



Peyrin Kao 7 months ago

Yeah, casting from a character array to an integer introduces endianness into the order in which the bytes need to be read.

I believe endianness always works on words, and the size of a word depends on your system. In a 64-bit system, all 8 bytes would be reversed within a word, and in a 16-bit system, both bytes would be reversed within a word.

good comment | 0



Justin Yokota 7 months ago

Endianness is not affected by the word size of the system; rather, the critical factor is the fact that endianness affects both how we store data in bytes, and how we read those resulting bytes.

When working with endianness, it is important to specify the size of the objects we are splitting memory into; for example, a char array is dealing with byte-sized objects, while an int array deals with 4-byte objects. I've adopted the convention of adding a new 0x prefix at the beginning of every block.

All data is ultimately stored in byte array form (the underlying system generally stores bytes as a single atomic object, though the math still works the same if we consider going down to the bit level).

Storing 0xDEADBEEF00112233 yields:

```
0x00: 0x33
0x01: 0x22
0x02: 0x11
0x03: 0x00
0x04: 0xEF
0x05: 0xBE
0x06: 0xAD
0x07: 0xDE
```

Which if you interpret as a sequence of ints, would yield

```
0x00: 0x00112233
0x04: 0xDEADBEEF
```

The easiest way to avoid mistakes with this is to always transform down to the byte array first, before regrouping bytes according to the new object size.

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f5



Anonymous Helix 7 months ago

Want to clarify that -- A pointer initialized in a function is stored on the stack whereas a pointer initialized outside of the function is part of the static data. Will a pointer ever be stored on the heap?

helpful! | 0



Peyrin Kao 7 months ago

```
int i = 5;
int** ptr = malloc(sizeof(int*));
*ptr = &i;
```

run code snippet

Visit Manage Class to disable runnable code snippets

This puts a pointer to `i` on the heap.

good comment | 0

Reply to this followup discussion

Resolved Unresolved

@909_f6



Anonymous Comp 7 months ago

In SP21 MT2 on Question 4a, instead of doing

```
slli t1 t0 8
add t1 t1 t0
sh t1 0(a1)
sb t0 2(a1)
```

Would a viable alternative be to do the below?

sb t0 0(a1)

sb t0 1(a1)

sb t0 2(a1)

helpful! | 0



Adelson Chua 7 months ago

No.

sb only gets the least significant byte within t0.

sh will get the least significant half-word (2 bytes) within t0.

You can't get the 2nd least significant byte from t0 if you just do consecutive sb instructions.

good comment | 0



Anonymous Comp 7 months ago

I'm still confused

So the full code loads an unsigned byte into t0. Assuming the byte is 1111 1111, the t0 register would be 0000 0000 0000 0000 0000 0000 1111 1111.

After `slli t1 t0 8`, the t1 register would be 0000 0000 0000 0000 1111 1111 0000 0000.

After `add t1 t1 t0`, the t1 register would be 0000 0000 0000 0000 1111 1111 1111 1111.

In this case, wouldn't `sh t1` just be putting 1111 1111 1111 1111 into 0(a0), which would be same thing if I just did `sb t0 0(a0)` and `sb t0 1(a1)`?

helpful! | 0



Adelson Chua 7 months ago

Ah, okay. Now that you gave an example and the full context of the problem, it makes it more apparent.

Then yes, seems like it is correct.

good comment | 1



Yiteng Zhou 7 months ago

yeah, I think you can just sb that character twice, I even make a simple loop for it.

helpful! | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f7



Anonymous Atom 7 months ago

Any chance we get solutions to the manually graded questions for the PM on PrairieLearn (SU21, I believe)?

helpful! | 2



Adelson Chua 7 months ago

Solutions can now be seen for all questions after three attempts.

good comment | 1



Anonymous Atom 7 months ago

Awesome, thanks!

helpful! | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f8



Anonymous Calc 7 months ago

For FA-21-MT1-Q2.9, if it says Hell is bytes 0-4, shouldn't there be 5 bytes?

The `[2]` syntax says to dereference the pointer and look for the second element (zero-indexed) in the array. Each element in the array is 4 bytes long because of the cast. Thus the 0th element is bytes 0-4 (Hell), the 1st element is bytes 5-8 (o Wo), and the 2nd element is bytes 9-12 (rld and the null byte).

helpful! | 0



Adelson Chua 7 months ago

Looks like a typo. Should be bytes 0-3, 4-7, 8-11. The answer is still the same though since the bytes are correctly partitioned into 4.

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f9



Anonymous Calc 7 months ago

For FA-21-MT1-Q4.3, how is $1.11111111 * 2^{15}$ equivalent to $2^{16} - 2^7$? Also, I'm confused by this part of the solution: "The significand has 8 bits plus the implicit 1 (e.g. 1.1111 1111), so to represent a 16-bit integer, we would need an exponent of 15 to create 1 1111 1111 0000 000.""

helpful! | 0



Anonymous Calc 7 months ago

^This is for 4.4 actually

helpful! | 0



Anonymous Poet 7 months ago

Note that 1.11111111 represents $2^0 + 2^{-1} + \dots + 2^{-8} = \frac{1-2^{-9}}{1-2^{-1}} = 2 - 2^{-8}$. Then, $2^{15}(2 - 2^{-8}) = 2^{16} - 2^7$.

~ An instructor (Adelson Chua) thinks this is a good comment ~

helpful! | 2



Adelson Chua 7 months ago

1.1111 1111 is the largest mantissa that you can set (we are looking for the largest integer). Then we pad 0s at the right through the exponent, resulting in 1111 1111 1000 0000, that's 16 bits. We can only pad 0s at the right since those cannot be represented in the mantissa anymore.

good comment | 0



Anonymous Helix 5 7 months ago

how do we get 2^{15} here? the exponent is all 0s if I am not mistaken? if the exponent is all 0s, this is a denorm?

helpful! | 0



Peyrin Kao 7 months ago

@909_f55

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f10



Anonymous Beaker 2 7 months ago

For Fa21 Mt1 Q4.5, why isn't it 2^{63} ? I thought the exponent would be 2^{63} and the mantissa is

0.00000001 = 2^{-8} , giving 2^{-71} .

helpful! | 0



Adelson Chua 7 months ago

The smallest denorm is $0.0000\ 0001 \times 2^{-62}$. That's equivalent to 1×2^{-70} .

Check the pattern for denorm exponents given the exponent bias in the lecture.

good comment | 1

Reply to this followup discussion



Resolved



Unresolved

@909_f11



Anonymous Scale 2 7 months ago

Hi, are SP21-MT1-Q5 and SP21-MT1-Q6 in scope?

helpful! | 0



Adelson Chua 7 months ago

No.

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f12



Anonymous Mouse 2 7 months ago

SU21-MT-Q6 (PraireLearn)

I am a bit confused about blanks b1-b4 in part B. I understand that to maintain calling convention, we would need to save the t0 on the stack as atoi uses a t0 anyway. However, as there is no assignment of a0 to t0 further down in the main branch, do we instead save the a0 register and not care/deal with the t0 register at all? Thanks!

helpful! | 0



Peyrin Kao 7 months ago

Can you post a screenshot of the relevant question?

good comment | 0



Anonymous Mouse 2 7 months ago

[Screen_Shot_2022-02-28_at_17.37.11.png](#)

I have blacked out my answers as well, so anybody scrolling through does not get the answers accidentally.

helpful! | 0



Anonymous Mouse 2 7 months ago

Oh nvm, it didn't get posted as a picture on piazza anyway (my bad).

helpful! | 0



Peyrin Kao 7 months ago

I think my answers here would be

B1: `addi sp sp -4` (make space on the stack for t0)
B2: `sw t0 0(sp)` (store t0's value on the stack, because the next call to `atoi` might mess it up)
B3: `lw a0 0(sp)` (restore t0's value back into a0 to pass the first argument into `pow`)
B4: `addi sp sp 4` (delete space off the stack)
[good comment](#) | 0



Anonymous Mouse 2 7 months ago

I had said the same thing, but then how would `power` use the value of `a0` (624), as we overload it with the return of the `power(arg1)` and then also do not transfer to back to `a0` before calling `pow`?
Would we just use it in `pow`? Wouldn't that be a calling convention error?

[helpful!](#) | 0



Anonymous Mouse 2 7 months ago

*t0 instead of it in the second question.

[helpful!](#) | 0



Peyrin Kao 7 months ago

If I'm understanding your question, according to calling convention, the `pow` function is allowed to modify all the a registers, so it's going to modify the `a0` register to put the return value there.

[good comment](#) | 0



Anonymous Mouse 2 7 months ago

I'm still a bit confused about how `pow` uses the value of `a0`. From what I understand, `a0` (the value we have gotten as '624' and transferred into 624) is overridden when we call

```
la a0 arg2
```

The question does save the value of `a0` in `t0`, but it does not load the value of `t0` back into `a0` before calling `pow` again, so wouldn't `a0` and `a1` be:

`a0 = 128`

`a1 = 128`

when we call `pow`?

[helpful!](#) | 0



Peyrin Kao 7 months ago

The integer 624 is saved on the stack in B2 (before `la a0 arg2` overwrites `a0`), then loaded back into `a0` in B3.

[good comment](#) | 1

Reply to this followup discussion

Resolved Unresolved [@909_f13](#)



Anonymous Gear 2 7 months ago

Sp 21 Final 6a, could someone please explain the intuition behind the `sizeof` and T/F question please?

Thanks!

helpful! | 0



Peyrin Kao 7 months ago

sizeof is 12 because:

- a (1 byte)
- Padding (1 byte so that b starts at an even address)
- b (2 bytes)
- c (4 bytes)
- d (4 bytes)

If you swap b and c :

- a (1 byte)
- Padding (3 bytes so that c starts at a multiple of 4)
- c (4 bytes)
- b (1 byte)
- Padding (3 bytes so that d starts at a multiple of 4)
- d (4 bytes)

This is 16 bytes > 12 bytes.

good comment | 0



Anonymous Gear 2 7 months ago

Why do we not pad both a and b in the first example by 2 bytes each?

helpful! | 0



Monis Mohiuddin 7 months ago

Also having trouble understanding when and when not to pad by the max amount or not. It seem like we pad bytes if the following item has more bytes, but then we don't pad b before swapping.

helpful! | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f14



Anonymous Helix 2 7 months ago

SP21-MT2-Q7

(b) A very cool data-structure is called a "Bloom Filter" which implements a probabilistic set (a test to see if a given element is present or not, but there is a chance of an error where it says something is in the set when it isn't).

To set an element in a Bloom filter we repeatedly hash the element. This hash function takes both the input and an index number and returns a 64 bit value that both "looks random" (take 161 for details on how we can do that). Simply adding 1 to the index number means the hash for the next index is apparently random and unrelated from the previous hash.

So for a fixed number of iterations we hash the element and for each one set the corresponding bit in a giant bitfield to 1 to add an element to the set and check that all such bits are set to see if an element is in the set.

```
struct BloomFilter {
    uint64_t (*hashFunc) (void *, uint32_t);
    uint16_t iters;
```

```

uint32_t size; /* The size in BITS of the bloom filter */
uint8_t * data; /* Size of uint8_t is always 1 */
};

```

Fill in the following code:

```

/* Be sure to initialize the data */
struct BloomFilter *bfalloc(uint64_t (*hashFunc) (void *, uint32_t), int size, int iters) {
    struct BloomFilter *ret = malloc(sizeof(struct BloomFilter));
    ret->hashFunc = hashFunc;
    ret->size = size;
    ret->iters = iters;
    ret->data = _____; /* Fill in A */
}

void bfset(struct BloomFilter *bf; void *element){
    uint32_t i = 0;
    uint32_t n;
    for(i = 0; i < b->iters; ++i){
        /* n = hash function called on the element modulo the # of BITS in the bloom filter */
        n = _____; /* Fill in B */
        /* Now we set the n'th BIT in the bloom filter to 1 */
        bf->data[n/8] = (_____) | bf->data[n/8]; /* Fill in C */
    }
}

```

- i. (3.0 pt) What should be line A? You don't need a sizeof() because uint8_t is always defined as one byte.

`calloc(size / 8 + 1)`

- ii. (3.0 pt) What should be line B?

`bf->hashFunc(element, i) % bf->size`

For ii., would `*(bf->hashFunc)(element, i) % bf-> size` also be a correct way to call the hash function on the element mod the number of bits in the bloom filter?

helpful! | 0



Peyrin Kao 7 months ago

See [this Stack Overflow post](#) - seems like dereferencing the function pointer still gives you the same function pointer.

good comment | 0



Anonymous Calc 2 7 months ago

why do we +1 for b.i.? size is the total number of bits in the data array, so shouldn't it just be size/8 bytes?

helpful! | 0



Peyrin Kao 7 months ago

If `size` isn't a multiple of 8, you'd want to round up to make one extra byte of space for the last few bits. (Remember that division in C rounds down.)

good comment | 0



Anonymous Calc 2 7 months ago

Why do we divide by 8 for part i?

helpful! | 0



Peyrin Kao 7 months ago

@909_f3

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f15



Anonymous Comp 2 7 months ago

[su21-mt-q3]

Just to clarify, here *str3 means dereferencing str3, not pointer str3?

<code>*str3</code>	heap	✓ 100%
<code>str3</code>	stack	✓ 100%

helpful! | 1



Adelson Chua 7 months ago

Yes correct.

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f16



Anonymous Gear 2 7 months ago

Fa 2021 Final 4.4:

The solutions says: Shortest clock period = clock-to-Q delay + largest combinatorial delay + hold time = 6+16+3 =

25 ps, but the hold time is 15 as calculated in the last question. So, is it supposed to be Shortest clock period = clock-to-Q delay + largest combinatorial delay + setup constraint (6), or is the hold time calculated differently from this than the last part?

helpful! | 0



Adelson Chua 7 months ago

Yeah that's a typo in the explanation.

You are correct, Shortest clock period = clock-to-Q delay + largest combinatorial delay + setup

constraint (6)

[good comment](#) | 0

Reply to this followup discussion

Resolved Unresolved @909_f17 



Anonymous Gear 2 7 months ago

SP 21 Final 3: can someone please explain the solution to this question? I am lost..

[helpful!](#) | 0



Adelson Chua 7 months ago

That's FSM, this is not covered in the lecture nor is part of the exam coverage.

[good comment](#) | 0

Reply to this followup discussion

Resolved Unresolved @909_f18 



Anonymous Comp 2 7 months ago

What's the solution for [su21-mt-q7] (SDS)? It says "manually graded" on prairie learn

[helpful!](#) | 0



Adelson Chua 7 months ago

It has been discussed here. @910_f1

[good comment](#) | 0

Reply to this followup discussion

Resolved Unresolved @909_f19 



Anonymous Atom 2 7 months ago

[SP21-MT-Q2B] I'm confused on question 2b. I looked over the lecture slides, but I'm still not understanding what a translator is. Is it the same thing as the compiler? Or is it a category that encompasses interpreters and compilers?

[helpful!](#) | 1



Anonymous Calc 2 7 months ago

Same here, I thought that translators only compiled code.

[helpful!](#) | 0



Peyrin Kao 7 months ago

Translators are a more general category than compilers. A translator is any program that takes code in one language as input and outputs the same code in a different language. For example, you could hypothetically design a translator that converts C code to Python (interpreted) code. See [lecture 9, slide 19](#).

[good comment](#) | 0

Reply to this followup discussion

Resolved Unresolved @909_f20



Anonymous Calc 2 7 months ago
SP21-MT2-Q7

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

8

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

shouldn't `sizeof(struct foo) = 5`? because 1 for the character and 4 for the pointer?

helpful! | 1



Anonymous Calc 2 7 months ago

<https://stackoverflow.com/questions/40679801/difference-between-sizeofchar-and-sizeofchar>

helpful! | 0



Justin Yokota 7 months ago

Note that the C compiler forces primitives to be stored at aligned addresses. Since C structs can't be rearranged (we can't reorder the components of a struct), the compiler instead decides to add padding (bytes that don't correspond to used data). In this case, it would assign one byte for a, 3 bytes padding, and 4 bytes for b, for a total size of 8 bytes.

good comment | 0



Anonymous Mouse 2 7 months ago

You need to align the data in the struct. For example, char a will take only 1 byte, but char *b needs to take 4 bytes and align its position, so its starting position is divisible by its own size (so it will start at position 4 relative to the start of the struct). Thus, the memory structure is

a - 1 byte

3-byte padding

b - 4 bytes

and hence 8 bytes.

helpful! | 0



Anonymous Calc 2 7 months ago

Okay, so anything thing that isn't divisible by 4 must be padded to get to 4 bytes?

helpful! | 0



Adelson Chua 7 months ago

Yes

good comment | 0



Anonymous Gear 4 7 months ago

If we have

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

is sizeof(struct foo) 5 bytes now?

helpful! | 0



Peyrin Kao 7 months ago

Yeah, I think this would be 5 bytes. (I checked [this Stack Overflow post](#) for reference.)

good comment | 0



Yiteng Zhou 7 months ago

yeah, but from here: <https://stackoverflow.com/questions/4306186/structure-padding-and-packing> the struct foo{ char *b; char a;} seems still to have sizeof 8, even if a is at the end.

helpful! | 0

Reply to this followup discussion

Resolved Unresolved @909_f21



Anonymous Calc 7 months ago

SP21-MT-Q1Ciii

How do we get this binary number?

iii. (1.0 pt) 64 in bias notation (with an added bias of -63)

0b1111111

helpful! | 0



Anonymous Calc 7 months ago

Sorry I meant this one

vi. (1.0 pt) -25 in bias notation (with an added bias of -63)

0b0100110

helpful! | 0



Adelson Chua 7 months ago

@106

unbiased encoding - bias = biased encoding.

$-25 - (-63) = 38$ (0100110)

good comment | 0

Reply to this followup discussion

Resolved Unresolved

@909_f22



Anonymous Poet 2 7 months ago

SP21-MT2-Q7 (a) iii

I'm really confused about what the solution code is doing, especially the shift in the first line. Why do we shift the index by 3?

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

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

helpful! | 0



Adelson Chua 7 months ago

`sizeof(struct foo)` is 8. So when you have an array of structs, the next element of that array is at +8, which is equivalent to shifting the index by 3 to the left.

good comment | 0



Anonymous Scale 3 7 months ago

so

line 1: moves the address 8 bytes to the left to get to address of the next element.

line2: ??

line3: why are we loading the word from 4 bytes added to the address of `a0`? why not `0(a0)`

helpful! | 0



Peyrin Kao 7 months ago

Line 1: Multiply `s6 (i)` by 8 to get the number of bytes we want to move the address by

Line 2: Increment the address of `f` to get to the address of element `i`

Line 3: Load the `b` field from element `i` (that field is 4 bytes after the start of the struct)

good comment | 0

Binary Representation 11000000011111000000000000000000
Hexadecimal Representation 0xc07c0000

helpful! | 0



Peyrin Kao 7 months ago

The exam says "For the following parts, use a floating point standard with 1 sign bit, 9 exponent bits, and 22 mantissa bits." but it seems like your paper work (and the online tool) are assuming the standard breakdown of 8 exponent bits and 23 mantissa bits.

good comment | 1

Reply to this followup discussion

Resolved Unresolved @909_f24



Anonymous Comp 7 months ago

SP21-MT-Q8 b) i.

How would we go about solving this question? My thinking was that if we knew the bias, then we can represent 80 with IEEE 754, then get the difference between it and the previous number that we can represent with IEEE-754. But we don't know the bias, so how would we proceed?

helpful! | 1



Anonymous Comp 7 months ago

I think I found the answer on the slides

$$\text{Bias} = -(2^{8-1} - 1) = -127$$

Except Replacing the 8 with number of exponent bits, so in this case it would be 9.

helpful! | 0



Peyrin Kao 7 months ago

Yeah, this probably should have been explicitly stated, but in general, floating points use a default bias depending on the number of bits in the exponent, given by the formula you posted there.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f25



Anonymous Calc 7 months ago

SP21-FINAL-Q6B

For part v, could someone explain the purpose of the last lw function: lw t0 t0(4).

For part vi, why did we store the data into a0?

For part vii, why does jalr only have two registers to the right: jalr ra t0? I'm confused how this calls cmp.

Thanks!

helpful! | 0



Adelson Chua 7 months ago

v) s1 is the structure. Look at the structure definition, the ****next** pointer is at +4 bytes (since the first variable ***data** is a 4-byte pointer).

vi) a0 is the argument for the function call

vii) if jalr does not have an immediate, default is 0. lw t0 12(sp) loads the address of the cmp function into t0, jalr ra t0 jumps to the address pointed to by t0 (which is cmp)

good comment | 0



Mallika Parulekar 7 months ago

For part v) I'm confused about what next[level] means - from the first two lines it seems like we are multiplying level by 16 and then adding it to s1-> next - could you elaborate why we do that?

helpful! | 0



Mallika Parulekar 7 months ago

Also is **lw a0 sp(0)** equivalent to **lw a0 0(sp)** (the second being notation I feel like we've seen more in class)?

helpful! | 0



Adelson Chua 7 months ago

I can't answer your next[level] question yet. But I can answer your 2nd question, yes it's the same. I don't know why they writing it like that.

good comment | 1



Mallika Parulekar 7 months ago

thanks! yeah I'm actually confused about what level means even? if i were to assume level means a certain number of nodes to skip we would need to solve this recursively right?

helpful! | 0



Adelson Chua 7 months ago

Well the ****next** variable within the struct can be thought of as an array of pointers yes? You are just accessing a specific pointer by doing next[level]. 'level' is the index of the array.

good comment | 0



Adelson Chua 7 months ago

@909_f31

good comment | 0



Anonymous Comp 5 7 months ago

I don't understand the use of jalr. I thought it's used in callee functions. What exactly does t0 represent in jalr ra t0? I know that it's the source register, but the source register from which caller?

Reply to this followup discussion

Resolved Unresolved @909_f26



Anonymous Beaker 3 7 months ago

For the prairielearn practice problem,

For this entire section, you are not allowed to use the C programming language as a calculator, an actual calculator, a floating point simulator, nor any other computational aid.

Consider a 13-bit floating-point number with the following components (1 sign bit, 6 exponent bits, 6 mantissa bits); that is,

SEEEEEEMMMMMM

All other properties of IEEE-754 apply (bias, denormalized numbers, ∞ , NaNs, etc). The bias is the usual value of $-(2^{E-1}-1)$, which here would be $-(2^5-1) = -31$.

PART A (3 pts.)

What is the bit representation (in hex) of the floating-point number -6.5? **Do NOT include the 0x prefix when writing your answer.** Answers given in binary will not be given partial credit.

0x A1A0

? x 0%

e.g. -6.5 (1 sign bit, 6 exp bits, 6 mantissa bits)
 sign: 1 bias = $-(2^{6-1}-1) = -31$
 (6 → 0b0110, 5 → 0b1)
 110.1 → 1.101
 Exponent: 2
 bias rep: $2 + 31 = 33 \rightarrow 0b010001$
 mantissa: 101
 1; 010001; 101000

Since there are only 13 bits, how can we convert it to hex? Also, the answer is 0x1868 but I'm very confused about how it got there

helpful! | 0



Caroline Liu 7 months ago

For your first question, if we don't have a number of bits that can evenly split until 4s (for hex), we want to 0-pad (not 0-extend)! What this means is we just add some ghost 0s until we can split into groups of 4. This doesn't change our fundamental value because we're not really counting those 0s as real values, they just help with our conversion. This is used when we need to divide bits evenly but we don't actually have those extra bit positions in hardware. You'd want to start blocking off groups of 4 bits starting from the least significant bit, and 0-pad **only** on the most significant bit's side.

For your second question, your exponent representation is one bit off! 33 in binary is `0b100001`, not `0b010001`! Thus, we have

`0b1 100001 101000` → `0b0001 1000 0110 1000` → `0x1868`.

Hope that makes sense!

good comment | 1

Reply to this followup discussion

Resolved Unresolved @909_f27 ↻



Anonymous Scale 3 7 months ago

for fa21 5.1, why do we do `addi t0, t0, 1` instead of adding 4? I thought strings were stored as `char[]` and moving up the array requires us to add 4?

helpful! | 0



Anonymous Scale 3 7 months ago

Lines 13-14: Once we finish checking a pair of characters, we need to increment the addresses to the next character. The next time the loop runs, the load instructions will now load the next character in the string. Recall that characters are 1 byte, and RISC-V indexes memory by bytes, so we should increment the addresses in `t0` and `t1` by 1. Accepted solutions: 1

lol oops

helpful! | 0

Reply to this followup discussion

Resolved Unresolved @909_f28 ↻



Anonymous Scale 3 7 months ago

for fa21 5.3, how do we know we are jumping backwards in memory and not forwards? how can we tell?

helpful! | 0



Adelson Chua 7 months ago

The question already said it.

Assume that `verifypassword` is located at `0x1000` and `Get20chars` is at `0xf00`. Your target is on a lower memory address, so you jump back.

good comment | 1

Reply to this followup discussion

Resolved Unresolved @909_f29



Anonymous Mouse 3 7 months ago

FA21-MT-2.9

Since it's a little endian system, why isn't `str[2] = "Hell"` and `str[0] = "rld\0"`?

helpful! | 0



Adelson Chua 7 months ago

When "Hello World\0" was stored in memory, they are stored byte per byte. Each letter corresponding to a byte: H is at address 0, e at address 1, and so on.

When type-casted to `uint32`, the group of 4 bytes are treated as a singular unit. "Hell" at address 0, "o Wo" at address 4, "rld\0" at address 8.

You can think of this as an array of ints starting at address 0. The [2] element would be "rld\0".

Little endianness will only matter on how you read the word, which is done in reverse.

good comment | 0



Adam Chmielewski 7 months ago

But since it's little endian wouldn't "Hello World\0" being stored byte by byte be stored as "\0dlroW... H"?

helpful! | 0



Peyrin Kao 7 months ago

Strings are not affected by endianness.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f30



Anonymous Gear 3 7 months ago

I'm a little confused as to why we have to `calloc` for that specific memory size for SP21-MT-7.b.i. Can anyone explain? Thanks.

i. (3.0 pt) What should be line A? You don't need a `sizeof()` because `uint8_t` is always defined as one byte.

```
calloc(size / 8 + 1)
```

helpful! | 0



i Adelson Chua 7 months ago

The code has a comment:

```
/* Be sure to initialize the data */
```

good comment | 0



Anonymous Gear 3 7 months ago

But why do we +1?

helpful! | 0



i Adelson Chua 7 months ago

I think this compensates for the fact that you are doing `size/8`. If `size` is not divisible by 8, you'll just allocate the floored version of the result. `+1` ensures that there is still space for the rest of the data (modulo 8)

good comment | 0



Ranelle Gomez ✓ 7 months ago

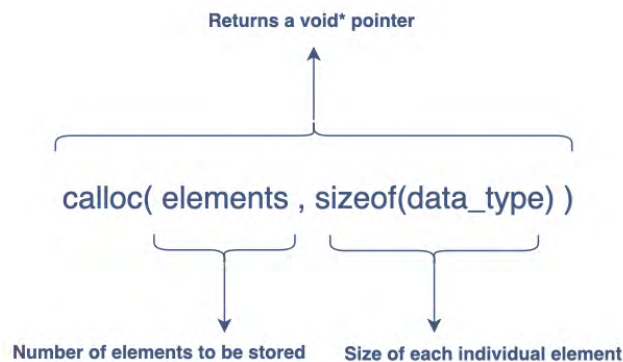
Doesn't `calloc` take two arguments? I believe the 2nd one isn't optional.

What is Calloc in C?



Edpresso Team

The **`calloc()`** function in C is used to allocate a specified amount of memory and then initialize it to *zero*. The function returns a *void pointer* to this memory location, which can then be cast to the desired type. The function takes in two parameters that collectively specify the amount of memory to be allocated.



Code

Take a look at the code below. Note how `(int*)` is used to convert the void pointer to an `int` pointer.

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main() {
5
6     int* a = (int*) calloc(5, sizeof(int));
7
8     return 0;
9 }
```

helpful! | 0



Ranelle Gomez ✓ 7 months ago

I know we don't need sizeOf here, so in this case, shouldn't we pass in 1 into the 2nd argument?

helpful! | 0



Adelson Chua 7 months ago

Yeah that must've been a typo since calloc needs 2 arguments. And yes, passing 1 as the second argument should work.

Resolved Unresolved

@909_f31



Anonymous Helix 3 7 months ago

for SP21-Final-6.c.v, why are shifting 4 bits, multiplying by 16, to find the offset? I was under the impression that the struct was 8 bytes hence we'd need to calculate the appropriate offset?

Reply to this followup discussion

v. load s1->next[level] into t0

```
slli t1 t1 4 add t0 t0 t1 lw t0 t0(4)
```

helpful! | 0



Peyrin Kao 7 months ago

Pretty sure this is a typo, and that 4 should be a 3. Sorry about that!

good comment | 0



Anonymous Helix 3 7 months ago

gotcha, thank you!

helpful! | 0

Reply to this followup discussion

Resolved Unresolved

@909_f32



Anonymous Beaker 3 7 months ago

For SU21 PM1.4, why do we want to allocate memory to the tree and the subtrees using calloc? Can we

use malloc as well?

helpful! | 0



Adelson Chua 7 months ago

The problem states that trees are initialized to NULL. calloc allocates memory and initializes them to 0, satisfying the requirement.

good comment | 1

Reply to this followup discussion



Resolved



Unresolved

@909_f33



Anonymous Comp 3 7 months ago

8. FLOATING POINT

For the following floating point questions, please use $^$ as the power operator in your answer. Do NOT put parentheses around the exponent. For example, 2^{-12} .

- (a) (3.0 pt) We define floating-point standard A to have 1 sign bit, 10 exponent bits, and 21 mantissa bits and floating-point standard B to have 1 sign bit, 18 exponent bits, and 45 mantissa bits. All other rules of IEEE 754 apply to standard A and B. How many more non-zero positive values can standard B represent compared to standard A? Please format your answer as additions and subtractions of 2's powers.

$$2^{53} - 2^{45} - 2^{31} + 2^{21}$$

- (b) For the following parts, use a floating point standard with 1 sign bit, 9 exponent bits, and 22 mantissa bits.

- i. In discussion 3, we defined the step size of x to be the distance between x and the smallest value larger than x that can be completely represented. Now consider all floating-point numbers in the range $[2^{-120} + 2^{-110}, 80]$.

- A. (2.0 pt) What is the largest step size?

$$2^{-16}$$

Can someone explain how we got the answer for a and b, I know for a we are taking denorm into consideration, but not sure why we need to subtract them. Also, for part b, how do we get 2^{-16}

helpful! | 0



Anonymous Comp 3 7 months ago

This is SP21-MT1-8a,b

helpful! | 0



Peyrin Kao 7 months ago

In standard B, there are $45+18=63$ bits that can be varied to create different positive numbers (the sign bit has to be 0). When the exponent is all 1s and the significand is non-zero, we have $2^{45} - 1$ NaNs that we have to subtract. The same logic works for standard A.

Step sizes increase as the magnitude of the number increases, so we just need to check the step size around the top part of the range. $80 = 0b1010000 = 0b1.010000 \times 2^6$. The next largest number we can represent is $0b1.010000\ 0000\ 0000\ 0000\ 0001 \times 2^6$ which is $0b0.000000\ 0000\ 0000\ 0000\ 0001 \times 2^6 = 2^{-16}$ larger than 80.

good comment | 0



Anonymous Beaker 4 7 months ago

Do, we also subtract an extra 1 from both sides for the case of all zeroes in mantissa and exponents?

[helpful!](#) | 0



Anonymous Gear 2 7 months ago

Hello,

I am still having some trouble understanding the solution to a. Could someone please explain how each term came to be in the equation?

Thanks!

[helpful!](#) | 0



Anonymous Scale 4 7 months ago

Should we look for the next largest element that is smaller than 80 instead since it has to be in the given range? (though the answer will be same)

[helpful!](#) | 0



Anonymous Beaker 5 7 months ago

Part a uses denorms to calculate the solution.

For standard B, there's exponent + mantissa possible combinations of positive numbers ($2^{45} + 2^{18}$).

However, there are NaNs we need to take into account. NaNs happen when the exponent is all 1s and the mantissa is nonzero. Since there are 45 mantissa bits, when the exponent is all 1s, then there are 2^{45} possible combinations for NaNs. But we still need to subtract 1 from the number of NaNs. When the mantissa is all 0s, the floating point value is infinity, so we need to subtract this possible value from 2^{45} .

Same concept applies to standard A.

[helpful!](#) | 0



Peyrin Kao 7 months ago

"Do, we also subtract an extra 1 from both sides for the case of all zeroes in mantissa and exponents?"

Yeah, I left this out of my work because both sides would have one zero, and they would cancel out.

"Could someone please explain how each term came to be in the equation?"

2^{63} comes from varying all 45+18 bits of the exponent and mantissa. 2^{45} comes from counting all the possible NaNs (the 18 exponent bits are all 1s, and the 45 mantissa bits can be varied).

Same for the other standard.

"Should we look for the next largest element that is smaller than 80 instead since it has to be in the given range?"

Sure, that would work too. The answer ends up being the same in this case.

[good comment](#) | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f34



Sachin Chhabria 7 months ago

SP21-MT2-Q8

Part a.

I tried counting the positive values by counting the regular positive numbers, infinity, positive NaNs, and positive denormalized numbers for both A and B and then finding the difference. I got the wrong answer. I tried counting them without the NaNs (not included in Screenshot) but I still got the wrong answer.

Standard B positive values:

$$\begin{array}{l} \text{Regular Pos: } (2^{18}-2) \cdot 2^{45} \\ \text{Infinity: } 1 \\ \text{NaN: } 2^{45}-1 \\ \text{Denorm: } 2^{45}-1 \end{array} \quad \left. \vphantom{\begin{array}{l} \text{Regular Pos: } (2^{18}-2) \cdot 2^{45} \\ \text{Infinity: } 1 \\ \text{NaN: } 2^{45}-1 \\ \text{Denorm: } 2^{45}-1 \end{array}} \right\} |B| = 2^{18+45} - 2^{1+45} + 1 + 2^{45}-1 + 2^{45}-1 \\ = 2^{63} - 2^{46} + 2^{45} + 2^{45} - 1 \\ = 2^{63} - 2^{46} + 2 \cdot 2^{45} - 1 \\ = 2^{63} - 1$$

Standard A positive values:

$$\begin{array}{l} \text{Regular Pos: } (2^{10}-2) 2^{21} \\ \text{Infinity: } 1 \\ \text{NaN: } 2^{21}-1 \\ \text{Denorm: } 2^{21}-1 \end{array} \quad \left. \vphantom{\begin{array}{l} \text{Regular Pos: } (2^{10}-2) 2^{21} \\ \text{Infinity: } 1 \\ \text{NaN: } 2^{21}-1 \\ \text{Denorm: } 2^{21}-1 \end{array}} \right\} |A| = 2^{10+21} - 2^{1+21} + 1 + 2^{21}-1 + 2^{21}-1 \\ \vdots \\ = 2^{31} - 1$$

$$\Rightarrow |B| - |A| = 2^{63} - 1 - 2^{31} + 1 = 2^{63} - 2^{31}$$

I would appreciate it if someone could tell me where I went wrong or what the correct way of doing this question is. Thank you!

[helpful!](#) | 0



Anonymous Calc 3 7 months ago

I think you have to consider the positive denorm values as well.

[helpful!](#) | 0



Anonymous Calc 3 7 months ago

nvm

helpful! | 0



Peyrin Kao 7 months ago

I think you're double-counting denorms. It might be easier to bundle them in with the regular numbers and get 2^{18+45} possible numbers, then just subtract the NaNs.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f35



Anonymous Comp 3 7 months ago

SP21-MT2-Q7 (a) iii, how do we know baz use a0 as argument register?

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

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

helpful! | 0



Adelson Chua 7 months ago

Following RISC-V calling convention.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f36



Anonymous Atom 3 7 months ago

So this doesn't matter how many bits we use for the FP numbers?

Q4.2 (1 point) Which representation has more representable numbers? Count $+0$, -0 , $+\infty$, and $-\infty$ as 4 different representable numbers.

- The floating point number
- The unsigned 16-bit integer
- Both systems can represent the same number of values

Solution: There are a total of 2^{16} bit patterns in either system, since 16 bits store 2^{16} possible values. This means we only need to think about which bit patterns as numbers and which bit patterns are not numbers.

In the integer system, every bit pattern represents a different number.

In the floating point system, some bit patterns represent NaNs, which are not numbers ("NaN" stands for "Not a Number").

Since the floating point system has some bit patterns that aren't numbers, and the integer system has no bit patterns that aren't numbers, the integer system can represent more numbers.

helpful! | 0



Justin Yokota 7 months ago

The floating point number in this question was explicitly defined above to use 16 bits.

good comment | 1



Anonymous Gear 2 7 months ago

Hi,

What are some examples of bit patterns that are NaNs for floats?

helpful! | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f37



Anonymous Comp 3 7 months ago

(a) (2.0 points) General

i. (1.0 pt) Which of the following statements must be true about compilers?

- Compiled code generally is only able to run on one **ISA**.
- Compilers produce larger code than interpreters but do it faster.

SP21-MT1-Q2a

Confused about why compiled code can generally only run on one ISA. What is an instruction set architecture?

helpful! | 0



Adelson Chua 7 months ago

Set of instructions like RISC-V.

When you have compiled code from high-level to assembly, the assembly is already written following the RISC-V instructions (as an example). Thus, the compiled code can only run in a processor that supports the RISC-V instructions.

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f38



Anonymous Calc 3 7 months ago

SP21-MT2-Q7 (a) iii

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

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

1. Is the "sll" in the first line a typo for "slli"?
2. Why do we offset by 4 in the third line?

helpful! | 0



Peyrin Kao 7 months ago

Yeah, that should say slli. The offset of 4 comes from the fact that b is the second field in the struct.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f39



Anonymous Poet 3 7 months ago

Fa21 2.2

```
char *foo() {
    char *str1 = "Hello World";
    char str2[] = "Hello World";
    char *str3 = malloc(sizeof(char) * X);
    strcpy(str3, "Hello World");
    // INSERT CODE FROM PARTS 5-7
}
```

Q2.1 (1 point) Where is `*str1` located in memory?

- code static heap stack

Solution: Static

This question is asking about the location of `*str1`, the address stored in `str1`.

The code assigns the `str1` pointer to a hard-coded string "Hello World". C will put this hard-coded string in static memory.

Grading: 1 point for selecting static.

Q2.2 (1 point) Where is `*str2` located in memory?

- code static heap stack

Solution: Stack

This question is asking about the location of `*str2`, the address stored in `str2`.

`str2` is a character array, and it is declared inside the `foo` function, so it is a local variable. Local variables are stored in stack memory.

Grading: 1 point for selecting stack.

Why is `*str2` on the stack if it's pointing to the same static string "Hello World"? Shouldn't this be in static memory? Dereferencing `str2` (i.e. `*str2`) gives the first character 'H', which is located in the same position in memory as `*str1` I thought.

helpful! | 0



Adelson Chua 7 months ago

Does this help? @176

good comment | 1



Anonymous Poet 3 7 months ago

as a follow-up, would this mean there's 20 bytes allocated on the stack in this function call: 4 bytes for `char* str1`, 12 bytes for `str2[]`, 4 bytes for `char* str3`? Or would it be 24 bytes since `str2` is still a pointer so it would be 4 bytes for the pointer of `str2` and 12 bytes for the string it points to ("Hello World")?

helpful! | 0



Justin Yokota 7 months ago

There will be 20 bytes allocated on the stack for local variables; `str2` is only the 12 byte array.

In practice, there will be more data stored on the stack, depending on the underlying assembly language and enabled security features. In RISC-V, for example, we would also use the stack to store saved registers, while in x86, we would also store function arguments, `eip`, and more. In order to make your code harder to hack, the stack may also contain random values that are checked to ensure that the stack wasn't corrupted.

good comment | 0

Reply to this followup discussion

Resolved Unresolved

@909_f40



Anonymous Beaker 4 7 months ago

```
.data
arg:  .string "123123"

.text
# Calls atoi(arg)
main: addi sp sp -4
      sw ra 0(sp)
      la a0 arg
      jal ra atoi
      lw ra 0(sp)
      addi sp sp 4
      jr ra

atoi: li t5 '0'      # loads the ASCII value for the character '0' into t5
      neg t5 t5     # negates the ASCII value of '0'
      # HINT: we are doing this so we can subtract the ASCII value of
      # '0' from something later
```

```

    addi t0 x0 0 # sum
    addi t1 x0 0 # index

Loop:  add t2 a0 t1
       ____BLANK_A1____
       ____BLANK_A2____
       ____BLANK_A3____
       addi t1 t1 1
       li t3 10
       mul t0 t0 t3
       ____BLANK_A4____
       j loop

done:  addi a0 t0 0
       ret

```

For clarification, this is summer 2021's RISC-V midterm question.

I'm unsure of what we should do with t5 and why t5 exists to be honest, even after what the hint shows :o. Why do we need t5? and why negate it.

Also, RISC-V is little endian. So when we do, lb t2 0(arg) lets just say arg is a register that holds that string. Since its little endian would t2 now hold the number 3, starting from the back? or would it hold the number 1.

Thank you!

[helpful!](#) | 0



Peyrin Kao 7 months ago

I'd have to see more of the question to know what this code is supposed to do, but my guess is you're trying to do some math with ASCII values. For example, uppercase A-Z is 65 to 90 in ASCII, and lowercase a-z in ASCII is 97-122, so you can add 32 to any uppercase letter to convert it to lowercase.

Strings aren't affected by endianness - they're always stored such that the first character is at the lowest address in memory, and the last character is stored at the highest address in memory.

[good comment](#) | 1

Reply to this followup discussion

Resolved Unresolved @909_f41



Anonymous Comp 3 7 months ago

Q3.1 (18 points) Complete the following code by filling in the blanks. This code should compile without errors or warnings. Each blank is worth 2 points.

```

cons *map(cons *c, (void *) (*f) (void *)) {
    cons *ret;

    if ( _____ ) return _____;

    ret = malloc(_____);

    _____extra_____ = 0;

```



```
_____ car = _____ ;  
  
_____ cdr = _____ ;  
return ret;  
}
```

Solution:

```
cons *map(cons *c, (void *) (*f) (void *)) {  
    cons *ret;  
    if ( c == NULL ) return NULL;  
    ret = malloc(sizeof(cons));  
    ret->extra.d = 0;  
    car = f(c->car);  
    ret->cdr = map(c->cdr, f);  
    return ret;  
}
```

FA21-MT1-Q3, why is the blank before car empty?

helpful! | 0



Peyrin Kao 7 months ago

Sorry, that's a typo. It should say `ret->car = f(c->car)` .

good comment | 0

Reply to this followup discussion

Resolved Unresolved

@909_f42



Anonymous Calc 3 7 months ago

SP21-MT1-Q8b

b.
bias $n = -(2^{q-1} - 1) = -255$
Convert 80 to floating point:

$$80 = 0b1010000 = 1.01 \times 2^6$$

$$\text{Exponent} = 6 + 255 = 261 = 0b1000000101$$

80 in floating point

$$0b0\ 1000000101\ 010\dots 000$$

Next step up, add 1 to mantissa

$$0b0\ 1000000101\ 010\dots 001$$

Convert

$$\underbrace{1.010\dots 001}_{22\ \text{bits}} \times 2^6 = (1 + 2^{-2} + 2^{-22})2^6 = 2^6 + 2^4 + 2^{-16}$$

Convert 80 to powers of 2

$$1.01 \times 2^6 = (1 + 2^{-2})2^6 = 2^6 + 2^4$$

step size

$$\cancel{2^6 + 2^4} + 2^{-16} - (\cancel{2^6 + 2^4}) = \boxed{2^{-16}}$$

Is this a valid way of solving this problem? I can't find an example of this type of problem anywhere.

helpful! | 0



Peyrin Kao 7 months ago

@909_f33

good comment | 0

Reply to this followup discussion

Resolved Unresolved

@909_f43



Anonymous Scale 3 7 months ago

can someone please explain fa21-q7.b.i and iii?

i. what is calloc doing and how do we know we need to divide by a byte? what does adding 1 do?

iii. why are we moding by 8? and not / or something?

helpful! | 0



Peyrin Kao 7 months ago

@909_f3

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f44



YoonWha Won 7 months ago

In 1(b), in the biased notation, we add the bias, but when accounting for bias in IEEE floating point, we subtract them? So I think of these ways of dealing with biases separately?

helpful! | 0



YoonWha Won 7 months ago

Also, if we are adding the bias to go from unsigned to bias, why 64 in bias notation with an added bias of -63 0b1111111 in (c)(iii)? In the case of (c)(iii), doesn't that mean we subtract the bias instead of adding it? Why are they contradicting to each other?

helpful! | 0



YoonWha Won 7 months ago

+) On a side note, I already read the related post on biased notation that was uploaded like last month, but still couldn't get it.

helpful! | 0



Adelson Chua 7 months ago

@106

@907_f6

good comment | 0



YoonWha Won 7 months ago

Yeah, after reading these piazza replies, I'm still confused. So does B being negative has anything to do with the range being (b)? Or is the range same with positive B? Moreover, how come we know we are going from biased to unbiased encoding in this question since we're adding the bias?

helpful! | 0



Rosalie Fang 7 months ago

Bias is almost always negative, and you always add the bias. In IEEE floating point, it still should be $2^{(\text{expo} + \text{bias})} * 1.\text{significand}$, with a negative bias (because you want to account for negative numbers and since the binary form is translated in an unsigned way, you have to add a

negative number to get a negative number as your final result). In c(iii), 64 in the bias notation means $x + \text{bias} = 64$, where x is the unsigned decimal form of your binary number. Therefore, x is 127 in this case $\Rightarrow 0b11111111$.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f45



YoonWha Won 7 months ago

For 2(b), how come translators produce both interpreted and compiled code? I know that any language can be translated/interpreted, but aren't these processes themselves separated things?

helpful! | 0



Peyrin Kao 7 months ago

I think this is asking about the output of the translator. I could build a translator from C to Python, where the output (Python) is code that needs to be interpreted. I could also build a translator from Python to C, where the output (C) is code that needs to be compiled.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f46



Monis Mohiuddin 7 months ago

For SP21-MT1-Q4, I had a question about the triple string method. I'm trying to understand why each char takes up 8 bits (2 bytes) of memory in `a1`, when a char is only 4 bits (1 byte). If I'm understanding the solution correctly, we repeat the char 3 times over bits 23-0 in `a1`, but I feel like it should only be from bits 11-0, or 4 bits per char. I wrote out my understanding of the question.

`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
```

Handwritten annotations:

- Copy `a1` (ret string) into `t2` (points to `mv t2 a1`)
- Put val at start of `a0` into `t0` (points to `lbu t0 0(a0)`)
- if `t0` is 0 (null term) end func (points to `beq t0 x0 End`)
- shifts `t0` over 8 bits & stores in `t1` (points to `slli t1 t0 8`)
- adds `t0` into `t1` (those last 8 bits in `t1` become `t0`, essentially `t0` twice) (points to `add t1 t1 t0`)
- store bits 15-0 into bits 15-0 of `a1` (points to `sh t1 0(a1)`)
- store bits 7-0 into bits 23-16 of `a1` (points to `sb t0 2(a1)`)
- go to next letter in parameter string (points to `addi a0 a0 1`)
- move forward 3 letters in ret string (points to `addi a1 a1 3`)
- 2nd time (bracketed around `sh` and `sb`)
- 3rd time (bracketed around `sb`)
- Null terminator at end? (points to `sb x0 0(a1)`)
- put `t2` back into `a0` to return (points to `mv a0 t2`)

Diagram: (Ex. 0...0 01000101 → 0 0 0100010 0...0)
32 8 7 6 32 17 16 8 7 0

helpful! | 1



Justin Yokota 7 months ago



Note that 1 byte is 8 bits, not 4. A char is 1 byte, since we can't store many characters with only 4 bits. (4 bits is known as a nybble)

[good comment](#) | 1



Monis Mohiuddin 7 months ago
damn major brain fart. thanks!

[helpful!](#) | 0



Anonymous Scale 3 7 months ago
can someone pls explain the slli by 8? i'm not sure why it's 8. chars are 1 byte so shouldn't it move over by 3 bytes or something?

[helpful!](#) | 0



Peyrin Kao 7 months ago
Consider the character 'A' (0x41 in ASCII). The lbu instruction loads the value 0x0000 0041 into t0. Shifting left by 8 bits gives you 0x0000 4100. Then adding this to t0 gives you 0x0000 4141, which doubles the character 'A'.

[good comment](#) | 2



Anonymous Beaker 3 7 months ago
I'm a bit confused about the second to last line, why don't we do "mv a0, a1"? Doesn't a1 point to the buffer that contain the string with every char tripled and t2 point to the original buffer (without char tripled)?

[helpful!](#) | 0



Peyrin Kao 7 months ago
a1 gets modified in the line `addi a1, a1, 3` so it no longer points to the start of the buffer. t2 points to the start of the modified buffer where the tripled string is stored (note `mv t2, a1` at the start, and that t2 never gets modified in the function).

[good comment](#) | 0



Anonymous Beaker 3 7 months ago
ahh that makes more sense now, thanks!

[helpful!](#) | 0



Anonymous Gear 7 months ago
I am still a bit confused about the doubling. I get that 0x0000 0041 becomes doubled when the lines s
llli t1, t0, 8 and
add t1, t1, t0

but I am confused as to why we would want to do this? When we triple the letters shouldn't we want

0b0000 0041 0b0000 0041 0b0000 0041? Because each character is 1 byte? Why would we want to duplicate the value within the one byte?

I am also then confused about the sh and sb instructions that follow

helpful! | 0



Anonymous Gear 7 months ago

Ohhhh you wrote it in hex!

helpful! | 0



Anonymous Gear 7 months ago

Does this question assume a little endian system? why does lbu line give 0x 0000 0041 for example instead of 0x41000000

helpful! | 0



Anonymous Gear 7 months ago

Sorry for the additional follow up!! finally get the staff solution, but I was wondering if my solution would be valid as well?

```
string-triple:
    mv $2, $1 #copies a1 into t2
    Loop:
        lbu $0, 0($a0) #loads unaligned byte to t0
        breq $0, $0 End-branch if $0 is 0
        sllr $1, $0, $1 #left shift t0 pointer by 8 bits (1 byte)
        addi $0, $0, 3 #0 = $0 = 3 incrementing the pointer by 1*3 bytes (memory is byte addressable!)
        addi $1, $1, -1 #s1 = s1 - 1
        addi $0, $0, 1 #a0 = a0 + 1
        j loop
    done:
        lw $0, 0($p)
        lw $1, 4($p)
        lw $3, 8($p)
        addi $p, $p, 12
        mv $0, $1
        jr $a
```

helpful! | 0



Anonymous Gear 6 7 months ago

^ Second this Anonymous Gear question! Is it valid just to store byte for offset 0, 1, and 2 instead of slli and add?

helpful! | 0



Peyrin Kao 7 months ago

The question asks students to use Venus, so I'd guess that it's assuming a little-endian 32-bit system.

The only way to be sure of your solution is to try it out in Venus. There should be alternate solutions that involve three store byte instructions, but I don't think the solution in that screenshot fully works because it doesn't actually call store byte more than once.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f47



Anonymous Scale 4 7 months ago

- i. (3.0 pt) What should be line A? You don't need a sizeof() because unit8_t is always defined as one byte.

```
calloc(size / 8 + 1)
```

For SP21-MT2-Q7Bi, I'm confused about where the + 1 came from. I know that size/8 gives us the size in bytes but why do we need the +1?

helpful! | 0



Adelson Chua 7 months ago

@909_f30

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f48



Anonymous Calc 7 months ago

SP21-MT1-Q8CiiA,

Hi! I'm having trouble solving this question! Could I get someones help?

- ii. Conversions: convert the following floating-point representation into their decimal values or vice versa. If the conversion is impossible, write N/A. Please specify infinities as +inf or -inf, not a number as NaN, hex numbers as 0xdddddddd where each d is in [0, f]. If your answer is a decimal number, DO NOT round it.

A. (2.0 pt) $-2^{-100} \cdot 0.625$

```
0xA6900000
```

helpful! | 0



Peyrin Kao 7 months ago

First note that $0.625 = 5/8 = 1/2 + 1/8$.

Convert the number to scientific notation:

$$-0.625 \times 2^{-100} = -0b0.101 \times 2^{-100} = -0b1.01 \times 2^{-101}.$$

Exponent = -101 . Accounting for bias gives $-101 + 255 = 154$. In binary, that's 0b0 1001 1010.

Mantissa = 0b01 0000 0000 0000 0000 0000

Sign bit = 0b1 (negative)

Combining: 0b1010 0110 1001 0000 0000 0000 0000 0000 = 0xA6900000

good comment | 1



Anonymous Gear 7 months ago
Why is the bias 255 rater than 127?

helpful! | 0



Anonymous Gear 7 months ago
Oh it's because 9 exponent bits

helpful! | 0



Anonymous Helix 6 7 months ago
Why does the exponent have 9 bits here?

helpful! | 0



Adelson Chua 7 months ago
It was given in the problem.

good comment | 1

Reply to this followup discussion

Resolved Unresolved @909_f49



Anonymous Mouse 4 7 months ago
SP21-MT1-Q3(a) I am confused about the definition of a path when considering for critical path and max hold time. Are we considering any path between two state machines (registers)? If so, isn't the shortest path from A to O from register A - not gate - or gate - not gate - then register O? Why are we including Black Box Logic in the path as well?

helpful! | 0

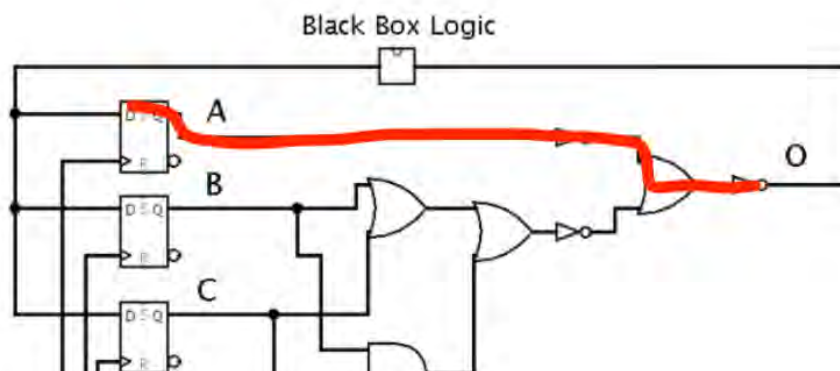


Adelson Chua 7 months ago
What is register O? There's no register O in the problem...
The black box logic is purely combinational.

good comment | 0



Anonymous Mouse 4 7 months ago
Sorry I mean the output O here? I thought a valid path could be from A - not gate - or gate - not gate - then O? Why do we also include the black box logic?





Circuit

(a) (2.5 pt) What is the maximum allowable hold time of the registers?

28

The shortest path through the circuit to a register clearly follows the path from A to O and includes: clk-to-q delay, two NOT gates, one OR gate, and the "Black Box." Maximum hold time = $6 + 2*3 + 7 + 9 = 28\text{ns}$

Because If we look at SP21-Final-Q4(i), we are considering all possible paths between any two state machines (input/output, or registers). For instance, input A -> not -> register could be a valid path.

helpful! | 0



Adelson Chua 7 months ago

You are checking if the shortest path meets the hold time requirement. The hold time requirement is a register requirement (i.e. you check if the input at the register changes within the hold time window). Therefore, the path has to end at the input of the register.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f50



Anonymous Helix 4 7 months ago

Is Fa21 question 3 in scope? Will we have to know scheme?

helpful! | 0



Adelson Chua 7 months ago

What exam specifically?

good comment | 0



Anonymous Helix 4 7 months ago

Midterm

helpful! | 0



Adelson Chua 7 months ago

It looks like a typical C programming question to me, so yes that's included.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f51



Anonymous Comp 4 7 months ago

whats the diff between lb and lw, and how do we know when to apply which?

helpful! | 0



Anonymous Comp 4 7 months ago

like for fa21 - mt - q 5.1, in line #9, the answer is lb but why isnt lw also valid?

helpful! | 0



Adelson Chua 7 months ago

lb means **load byte**. lw is **load word**. The verifypassword loop checks for every byte (since 1 char is 1 byte).

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f52



Anonymous Atom 4 7 months ago

Just a general question about RISC-V. When RISC-V is run, does it go line by line to even access things in loop labels that were not called on by the function label. For example, in question 5 on Fall 2021, we don't call End after Fail, so does it just automatically go to End?

helpful! | 0



Peyrin Kao 7 months ago

The program always increments PC by 4 and executes the next instruction, unless a jump or branch instruction changes the PC to point somewhere else.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f53



Anonymous Calc 4 7 months ago

SP21-MT1-Q5.1

When do we use la and mv? I'm confused on the differences between the two.

helpful! | 0



Adelson Chua 7 months ago

You might be referencing a different exam...

la is typically used to put an address of a label to a register: la t0 label

mv t0 t1 is equivalent to addi t0, t1, 0

good comment | 0



Anonymous Calc 4 7 months ago

I meant FALL 2021. Sorry

helpful! | 0

Reply to this followup discussion

Resolved Unresolved @909_f54



Anonymous Poet 4 7 months ago

Sp21-MT2-Q5

Is this question out of scope or will DATAPATH PIPELINING be in the exam?

helpful! | 0



Adelson Chua 7 months ago

Out of scope.

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f55



Anonymous Scale 5 7 months ago

Q4.4 (4 points) Out of all numbers representable by this floating point system, what is the largest number that can also be represented as an unsigned 16-bit integer?

Solution: $2^{16} - 2^7 = 65408$

The unsigned number can represent any nonnegative integer less than 2^{16} , so we're looking for the largest integer less than 2^{16} that can be represented by the floating point number. To do this, we can try to create a 16-bit integer with the floating point number, and how we can maximize the number created through this process.

The significand has 8 bits plus the implicit 1 (e.g. 1.1111 1111), so to represent a 16-bit integer, we would need an exponent of 15 to create 1 1111 1111 0000 000.

Note that the lower 7 bits of any number created in this process will always be 0, because they are not part of the significand. Thus all we can do to maximize this number is adjust the significand to be as large as possible. The largest significand would be all 1s, as shown above.

In other words, the value we want is $0b1.11111111 \times 2^{15}$, which is equal to $2^{16} - 2^7 = 65408$.

Grading: Half credit was awarded for $2^{16} - 1$ and $2^{16} - 2^8$.

I am having a hard time understanding what the solution is doing. Could someone explain it, specifically how they got 1 1111 1111 0000 000

helpful! | 0



Peyrin Kao 7 months ago

We're trying to create the largest possible 16-bit integer. We know that floating point numbers in this system will look something like $0b1.1111 1111 \times 2^n$ where you have 8 mantissa bits and n is an exponent of your choosing. To make this a 16-bit number, we want to move the decimal point to the right until there are 16 digits to the left of the decimal point, which we can do with an exponent of 15.

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f56



Anonymous Gear 7 months ago

I am very confused about this problem, specifically each of the lines.
Can someone explain to me what each line in this accomplishes?

```
Sp21_MT_Solutions.pdf 9 / 26 259% +
```

```
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)
```

helpful! | 0



Adelson Chua 7 months ago

@909_f46

Follow-up on that one if ever.

[good comment](#) | 0

Reply to this followup discussion

Resolved Unresolved @909_f57



Anonymous Beaker 3 7 months ago

SP21-MT1-Q1c(i)

Translate the following decimal numbers to their correct representation in binary using 7 that contain more or less than 7 bits will receive no credit. If the number cannot be represented, N/A (be sure to use all caps). REMEMBER to include the binary prefix!

) i. (1.0 pt) 64 in 2's complement

N/A

Is it because there's not enough space for the sign bit here?

helpful! | 0



Adelson Chua 7 months ago

Technically yes.

64 is out of range in 7-bit 2s complement.

[good comment](#) | 1

Reply to this followup discussion

Resolved Unresolved @909_f58



Anonymous Beaker 3 7 months ago

v. (0.5 pt) The symbol and relocation table can be synonymous in what they do during assembly.

- True
- False

False; the symbol table locates non-static labels defined in the present file being assembled that may eventually be used later on in the same file or other files; the relocation table locates external and static references that eventually need to be filled during linking.

Symbol Table

- List of “items” in this file that may be used by other files
- What are they?
 - Labels: function calling
 - Data: anything in the `.data` section; variables which may be accessed across files

So would the static data be in the symbol table or relocation table? Why's the lecture slide stating that the symbol table will include anything in the `.data` section? Isn't the static data be referenced in the relocation table later in the Linker stage?

helpful! | 0



Anonymous Beaker 3 7 months ago

Could anyone confirm this?

helpful! | 0



Caroline Liu 7 months ago

Answered somewhere else I think?

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f59



Anonymous Mouse 4 7 months ago

SP21-MT1-Q7(a) Why is `sizeof(struct foo)` 8 not $6 = 2+4$?

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

8

Also I don't really understand how padding and packing work for a struct type?

- Padding
 - Every field sits on memory addresses of

max(all elements) memory boundaries

● Packing

- Every field sits on memory addresses of min(all elements) memory boundaries

helpful! | 0



i **Peyrin Kao** 7 months ago

char a takes up 1 byte. Then char *b needs to start at a 4-byte boundary, so we add 3 bytes of padding and start char *b at byte 4. In total, we have 1 (char) + 3 (padding) + 4 (pointer) = 8 bytes.

By default, C pads structs to make sure that things are aligned (e.g. pointers shouldn't start at odd addresses). Packing forces every field of the struct to be adjacent, even if it causes things to start at unaligned addresses.

good comment | 0



Anonymous Mouse 4 7 months ago

If we look at SP21-Final-Q6(a)i, the uint16_t b only needs to start at a 2-byte boundary not 4-byte boundary. Is it because the variable b itself is taking up 2 bytes?

i. Assuming a 32-bit architecture with RISC-V alignment rules:

Consider the following structure definition and code:

```
struct foo {  
    char a; 0  
    uint16_t b; 2, 3  
    char *c; 4, 5, 6, 7  
    struct foo *d; 8, 9, 10, 11 12 13 14 15  
}
```

Handwritten annotations: "4 bytes" next to the struct definition, and memory addresses 0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 written next to the fields.

What is sizeof(struct foo) (Answer as an integer, with no uni

ii. If b and c are swapped, this increases the size of the structure:

True

False

16 bytes

helpful! | 0



i **Adelson Chua** 7 months ago

Yes, variable b is literally defined as uint16_t (unsigned 16 bit). 16 bit = 2 bytes.

This is 'packing' in action.

[good comment](#) | 0



Anonymous Mouse 4 7 months ago

Yes. "C pads structs to make sure that things are aligned (e.g. pointers shouldn't start at odd addresses)". So my question is how many bytes do we pad? SP21-MT1-Q7(a) is padding 2 bytes. SP21-Final-Q6(a)i is padding 4 bytes. It seems like we want to pad such that the starting index is a multiple of the size of the variable?

[helpful!](#) | 0



Justin Yokota 7 months ago

Yup. This happens up to at least the word size, though in practice you rarely get primitive data values that are larger than the word size.

[good comment](#) | 0



Anonymous Scale 6 7 months ago

Why is the size of (struct foo) 12 and not 16?

I had that a char is 1 byte, plus padding (+1) then b is 2 bytes and the char array is 4 bytes, then the nested struct foo would add 8 bytes for a total of 16?

thanks!

[helpful!](#) | 0



Caroline Liu 7 months ago

What happens here is when we align to 32-bits (4 bytes), we don't need to have each field start at an address with a multiple of the alignment! (That's padding.)

Here, we just need to make sure the struct itself occupies a chunk of memory that is a multiple of what we need to align to. Thus, we have 1 byte from char a , 2 bytes from uint16_t , 4 bytes from char *c , and 4 bytes from struct foo *d . This gives us a packed size of 11 bytes. We need to now add one extra byte to get to the next multiple of 4, which is 12 bytes!

[good comment](#) | 0

Reply to this followup discussion

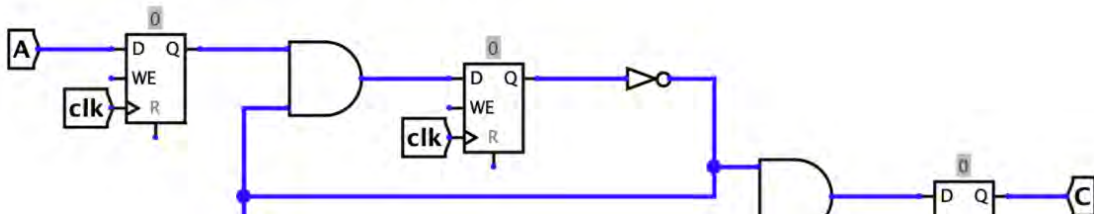
Resolved Unresolved [@909_f60](#)

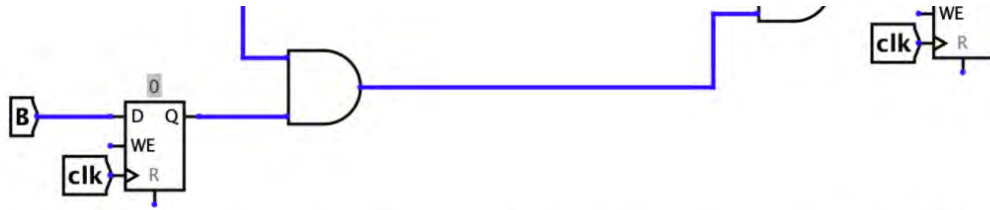


Anonymous Mouse 5 7 months ago

Correct answer

Answer the following questions about the given circuit.





Suppose that all registers have a setup time of **5 ps**, AND gates have a propagation delay of **10 ps**, and NOT gates have a propagation delay of **3 ps**. **A** and **B** are the circuit's inputs, and **c** is its output.

PART A (2 pts.)

What is the length (in ps) of the shortest combinational logic path in this circuit?

10

PART B (2 pts.)

I'm a bit confused as to why the shortest path is 10. Although I got the value for the longest combinational logic path right, I'm not too sure on how to calculate these paths. My intuition was going from the A or B labels to the C label, where I sum up the gates I came across (i.e. AND gate, XOR gate, etc). For example, I assumed that the path from B to C was the shortest because it only goes through 2 AND gates, which summed up to 20, but I know this is wrong.

helpful! | 0



Adelson Chua 7 months ago

@910_f1

@910_f4

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f61



Anonymous Calc 7 months ago

For FA21-Final-3, in the question shown below, how is verifypassword located at 0x1000, while the stack pointer is located 0xBFFF F800? I thought the stack is pointing to verifypasswords as well?

During the course of your testing, you discovered an interesting fact: the function `Get20Chars` doesn't actually work as intended! Instead of truncating at the 20th character, `Get20Chars` continues to write data until the first null terminator in its input. As before, `verifypassword` is located at 0x1000 and `Get20Chars` is located at 0x0F00. Further, assume that the stack pointer is located at 0xBFFF F800 at the start of `verifypassword`, our page size is 4 KiB, and that we are currently working on a little-endian system.

helpful! | 0



Adelson Chua 7 months ago

No, they are different segments of the memory!

You know how in the lecture they usually show you that the memory is divided into stack, heap, static data, code? `verifypassword` is in code, stack is in well... stack.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f62



Anonymous Gear 5 7 months ago

For SP21-midterm1 question 8b, how is the largest step size calculated?

helpful! | 0



Peyrin Kao 7 months ago

@909_f33

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f63



Anonymous Atom 2 7 months ago

FA21-M2-4.6

Q4.6 (4 points) What is the smallest positive number representable by the unsigned 16-bit integer that isn't representable by this floating point system?

Solution: $2^9 + 1$

Intuitively, floating point numbers can represent all smaller integers 1, 2, 3, etc. but eventually, there will be an integer that the floating point number skips over (the gaps between numbers get wider as the number gets larger). Thus we are looking for the smallest positive integer that is not representable by the floating point number.

If we make the exponent exactly equal to the number of bits in the significand, then we can use the entire significand to represent a positive integer. The significand has 8 bits, so we can set the exponent to $71-63=8$ and use the 8 bits of the significand and the implicit 1 to represent all integers up to 2^9 .

After 2^9 , the exponent must be increased to $72-63=9$. This will add a 0 to the end of the bits of the significand, which means that odd numbers are no longer representable after 2^9 . Thus the smallest positive integer that cannot be represented by the floating point number is $2^9 + 1$.

I'm not understanding why the biggest representable integer is 2^9 when the exponent is $71-63 = 8$. Wouldn't this represent all integers up to $0b11111111 = 2^8 + 2^7 + 2^6 + \dots + 2^0 = 2^9 - 1$?

helpful! | 0



Adelson Chua 7 months ago

This is very similar to the HW3 problem that we had.

Follow the discussion here for an intuitive approach. @373_f7

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f64



Anonymous Atom 5 7 months ago

clarifying question. If I had

```
Char[] c = "hello"
int main(){
}
```

I understand "hello" will be stored in static memory but what about c variable? Will it be stored in stack?

Also, char * c = "61c" .

sizeof(c) outputs 8. I am not sure why,

while char[] d = "61c", sizeof(d) will output 4(I understand this one)

helpful! | 0



Rosalie Fang 7 months ago

For your first question, the c variable will not be stored on the stack. It will be a global variable.

For the second question, char* c = "61c", c is a pointer to a char string and should be 4 bytes.

Are you testing this on your computer? It might have a 64-bit architecture which would mean that pointers are 8 bytes instead of 4.

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f65



Anonymous Calc 5 7 months ago

SU21-M1-1.4B

PART B (13 pts.)

Fill in the blanks to implement the following functions to allocate, deallocate, and insert into a binary search tree.

- An empty tree or subtree is represented by the **NULL** pointer.
- You may assume the following:
 - Any **bst** passed as function argument is a valid tree created by your implementation of **bst_new** and populated by calls to your **bst_insert**.
 - No duplicate elements are inserted into the BST.
- You must ensure that your code makes no invalid memory accesses, and that **bst_free** frees all of root's children (as well as root).

```
/* Initializes a new, empty BST. */
bst *bst_new() {
    bst *root = __BLANK_1__;
    return root;
}

/* Frees memory allocated for this tree and all sub-trees. */
void bst_free(bst *root) {
    if (root != NULL) {
        __BLANK_2__(root->left);
        __BLANK_3__(root->right);
        __BLANK_4__(root);
    }
}

/* Inserts a value into the provided tree, preserving the sorted ordering. */
void bst_insert(bst *root, uint32_t value) {
    if (value < root->value) {
        // Insert into the left sub-tree
        if (__BLANK_5__) {
            bst *subtree = __BLANK_6__;
            subtree->value = value;
            root->left = subtree;
        } else {
            __BLANK_7__;
        }
    }
}
```

```

} else {
    // Insert into the right sub-tree
    if (___BLANK_8___) {
        bst *subtree = ___BLANK_9___;
        subtree->value = value;
        root->right = subtree;
    } else {
        ___BLANK_10___;
    }
}
}
}

```

BLANK_1	NULL
BLANK_2	bst_free
BLANK_3	bst_free
BLANK_4	free
BLANK_5	root->left == NULL
BLANK_6	calloc(1, sizeof(bst))
BLANK_7	bst_insert(root->left, value)
BLANK_8	root->right == NULL
BLANK_9	calloc(1, sizeof(bst))
BLANK_10	bst_insert(root->right, value)

According to the solutions, if I call:

```

bst *tree = bst_new();
bst_insert(tree, 4);

```

Wouldn't the first if statement in bst_insert crash since root is NULL and therefore root->left is illegal?

helpful! | 0



Anonymous Calc 5 7 months ago

meant to say root->value

helpful! | 0



Adelson Chua 7 months ago

That's... actually a good point. ~~Let me confirm this. But right now I believe you are correct.~~
I'll follow up on this one.

Edit: I realized you were actually giving a situation, as the sequence of function calls you had wasn't actually on the exam.

So to answer you question, yes. You are correct. Those sequence will crash because root is NULL.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f66



Anonymous Poet 5 7 months ago

this maybe be dumb buy can I get what is bytes for long and long long?

helpful! | 0



Adelson Chua 7 months ago

<https://docs.oracle.com/cd/E19253-01/817-6223/chp-typeopexpr-2/index.html>

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f67



Anonymous Beaker 6 7 months ago

Is FA_MT1_Q6 in scope? I don't know how to approach the problem

helpful! | 0



Adelson Chua 7 months ago

FA what?

good comment | 0



Peyrin Kao 7 months ago

Assuming you mean FA2-MT-Q6, yes, this is in scope - it's related to the test writing task of Project 2A.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @909_f68



Anonymous Gear 5 7 months ago

Could someone explain how to do SP21-Final Q5(Boolean Algebra)?

helpful! | 0



Adelson Chua 7 months ago

The way to approach this is to simplify the choices through Boolean algebra and then see if you can arrive to the same expression as the question.

Although there may be times wherein your 'simplified' expression might not exactly look like the one in the question, the next approach would be to write up the truth table of your 'simplified' expressions and see if it matches the truth table of the given.

good comment | 0



Anonymous Gear 5 7 months ago

$$\begin{aligned} & F + \overline{(ABC + \bar{C})A} \\ &= \overline{A(BC + \bar{C})} \\ &= \overline{A + \overline{BC + \bar{C}}} \\ &= \overline{A} + (\overline{BC} \cdot C) \end{aligned}$$

This is what I got after simplifying choice d, so I wonder where went wrong.

helpful! | 0



Adelson Chua 7 months ago

From 3rd line:

$A' + (BC+C)'$

$A' + (BC)'C$

$A' + (B'+C)C$

$A' + B'C + C'C$

$A' + B'C$

Take care applying De Morgan's

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f69



Anonymous Beaker 3 7 months ago

v. (0.5 pt) The symbol and relocation table can be synonymous in what they do during assembly.

True

False

False; the symbol table locates non-static labels defined in the present file being assembled that may eventually be used later on in the same file or other files; the relocation table locates external and static references that eventually need to be filled during linking.

Symbol Table

Computer Science 61C Spring 2022

McMahan

- List of “items” in this file that may be used by other files
- What are they?
 - Labels: function calling
 - Data: anything in the `.data` section; variables which may be accessed across files

So would the static data be in the symbol table or relocation table? Why's the lecture slide stating that the symbol table will include anything in the `.data` section? Isn't the static data be referenced in the relocation table later in the Linker stage?

(duplicate from the previous post since there hasn't been any answers yet..)

helpful! | 0



Anonymous Beaker 3 7 months ago



So is the slide and the solution kind of contradicting...? Or is there another way to interpret it?

helpful! | 0



Anonymous Beaker 3 7 months ago

can anyone help

helpful! | 0



Caroline Liu 7 months ago

Hey! I'm writing a response, give me a minute haha.

The purpose of the symbol table for a file, say File A, is to collect all the labels defined in File A and put them in Symbol Table A, to help resolve locally defined labels in the same file. Relocation Table A is used to identify labels that haven't **yet** been resolved, which might be forward references (labels used before they're defined in the same label), external references to other files that'll be linked, or external references to library files. We put the labels in there and say "I don't know where these are defined, or if they are at all so we'll just say temporarily these are uncertain and we'll mark them as needing to be calculated".

In the second pass of the assembler, we resolve all local labels (forward references), but we're still left with the external references.

In the linker stage, we do a few things.

- We use symbol tables from other files that's not the one we're looking at to resolve external references defined in files we've written/have locally.
- We use library definitions to resolve external references from library imports.

To answer your question, static data isn't really stored in either table? The position of the static data defined with the `.data` directive in RISC-V is defined in the symbol table, usually relative to say, the beginning of the file. The data itself is not stored in the symbol table though. The relocation table doesn't include the data itself; instead it contains just references that need to be resolved/relocated, but haven't yet been.

Does that make sense?

good comment | 1



Anonymous Beaker 3 7 months ago

Thanks for your detailed explanation! But I just want to double check that do static data always contain absolute addresses? Are static data resolved during the linker stage?

helpful! | 0



Caroline Liu 7 months ago

Yes! All static data will eventually sit in the static section of memory, and references will need to be always relocated, which is resolved during the linker stage.

good comment | 1

Reply to this followup discussion

Resolved Unresolved @909_f70



Anonymous Gear 5 7 months ago

For SP21-Final 10b, I am confused on why the immediate in the solution is not sign-extended and why is it $2^{10} + 2^4$ instead of 2^4 , which is 16.

My attempt: \downarrow sign extend

ob 1111 1110 0000 1001 1101 0010 10010011

answer: ob 0100 0001 0000 1001 1101 0010 10010011

helpful! | 0



Adelson Chua 7 months ago

The immediate is 16, a positive number. It should be sign-extended with zeroes.

good comment | 0



Anonymous Gear 5 7 months ago

ok, then why is there a one at the 10th bit of the immediate? 16 is 2^4 , so I think there should only be a one at the 5th bit of the immediate.

helpful! | 0



Adelson Chua 7 months ago

Check the encoding please!

srai is an I* instruction type. The most significant bits correspond to funct7.

good comment | 0



Caroline Liu 7 months ago

srai t0 s3 16

It's an I*-type instruction, so our breakdown is funct7 | imm[4:0] | rs1 | funct3 | rd | opcode .

funct7 : 0100000

imm[4:0] → 16 = 10000

rs1 → s3 = x19 = 0b10011

rd → t0 = x5 = 0b00101

funct3 = 101

opcode = 0010011

Putting that all together we get: 0100000 10000 10011 101 00101 0010011 → 0100 0001 0000 1001 1101 0010 1001 0011 → 0x4109D293 .

~ An instructor (Adelson Chua) thinks this is a good comment ~

good comment | 1

Reply to this followup discussion

Resolved Unresolved @909_f71



Anonymous Beaker 3 7 months ago

SP21-FINAL-Q6(b)(v)

v. load sl->next[level] into t0

```
slli t1 t1 4 add t0 t0 t1 lw t0 t0(4)
```

v. load sl->next[level] into t0

irrelevant

bytes within arr

```
slli t1 t1 2  
add t0 t0 t1  
lw t0 t0(4)
```

absolute

*↑
imm*

vi. load data into a0

Why do we do slli t1 t1 4 instead of slli t1 t1 2 as that's what's said in the walkthrough video?

helpful! | 0



Adelson Chua 7 months ago

Yeah, the solution has a typo error. Follow the walkthrough video.

good comment | 1

Reply to this followup discussion

Resolved Unresolved @909_f72



Anonymous Mouse 6 7 months ago

For iii. why is the n modded here? I would've thought it was divided by 8 instead since the the number of bits is divided by the number 8 to get the number of bytes, then it is left shifted.

```
the set.

struct BloomFilter {
    uint64_t (*hashFunc) (void *, uint32_t);
    uint16_t iters;
    uint32_t size; /* The size in BITS of the bloom filter */
    uint8_t * data; /* Size of uint8_t is always 1 */
};

Fill in the following code:

/* Be sure to initialize the data */
struct BloomFilter *bfalloc(uint64_t (*hashFunc) (void *, uint32_t), int size, int iters) {
    struct BloomFilter *ret = malloc(sizeof(struct BloomFilter));
    ret->hashFunc = hashFunc;
    ret->size = size;
    ret->iters = iters;
    ret->data = _____; /* Fill in A */
}

void bfset(struct BloomFilter *bf; void *element){
    uint32_t i = 0;
    uint32_t n;
    for(i = 0; i < b->iters; ++i){
        /* n = hash function called on the element modulo the # of BITS in the bloom filter */
        n = _____; /* Fill in B */
        /* Now we set the n'th BIT in the bloom filter to 1 */
        bf->data[n/8] = (_____) | bf->data[n/8]; /* Fill in C */
    }
}
```

i. (3.0 pt) What should be line A? You don't need a sizeof() because uint8_t is always defined as one byte.

`calloc(size / 8 + 1)`

ii. (3.0 pt) What should be line B?

`bf->hashFunc(element, i) % bf->size`

iii. (3.0 pt) What should be line C?

`1 << (n % 8)`

helpful! | 0



Peyrin Kao 7 months ago

You're trying to set the n'th bit within a given byte, so by taking the bit count mod 8, you figure out

which bit in the byte you want to set. Then you shift the 1 into the correct position so that bit gets set.

[good comment](#) | 0

Reply to this followup discussion

Resolved Unresolved @909_f73



Anonymous Atom 5 7 months ago

When calculating the longest and shortest path, do we factor in the CLK to Q time ? or just the delay of the gates

[helpful!](#) | 0



Adelson Chua 7 months ago

We typically explicitly say: longest *combinational* path or shortest *combinational* path, which only considers gate delay.

When saying just path... I'm not sure, maybe ask for clarifications if ever.

[good comment](#) | 0



Peyrin Kao 7 months ago

We'll try to be explicit on exams. It depends on the context of the circuit.

[good comment](#) | 0

Reply to this followup discussion

Resolved Unresolved @909_f74



Anonymous Poet 5 7 months ago

Can I ask why SP21_MT_Q7 aii the sizeof(f) is 4?

[helpful!](#) | 0



Caroline Liu 7 months ago

f is a pointer, and in a 32-bit system, the size of a pointer is always 4 bytes!

[good comment](#) | 0

Reply to this followup discussion

Resolved Unresolved @909_f75



Yiteng Zhou 7 months ago

One conceptual problem after I see the posts above about padding:

-> Inside struct, needs padding, while the last element do not.

And if it's outside struct, like just `int a = 3; char b = '0'; int c = 50;` we can't assume the memory allocation is consecutive, but if a, b, c is consecutive allocated, is there padding or not? Or it will be specified (if appears).

I see there is a big difference between SP21_MT and FA21_MT, considering COVID, is today's midterm more like FA21_MT? (if it can be answered)

[helpful!](#) | 0



Peyrin Kao 7 months ago

I don't think we've talked about compiler padding outside of a struct, so if this is on an exam, we would clarify. (Or if I'm wrong and this was discussed in lecture, then our answer would be based off whatever you've seen in lecture.)

I can't answer questions about this semester's midterm, but note that in the SP21 midterm, students had access to Venus, but this semester, you will not have access to Venus (and the grading will account for this).

[good comment](#) | 0

Reply to this followup discussion



Resolved



Unresolved

[@909_f76](#)



Anonymous Mouse 6 7 months ago

I don't understand question v. in this question. Why is it left shifting by 16? Where is it getting 16 from? I thought it might be a typo, because I thought it was 4. I'm not sure though. Also, in this question, it only ever passes values into a0. Is this a typo? Shouldn't it pass in two arguments a0 and a1 since cmp has two arguments?

(b) Skipping Around

Consider the following code

```

struct SLN{
    void *data;
    struct SLN **next;
}

```

```

/* Only the following in the code actually matters:
   cmp(find, sl->next[level]->data)
   But if you are curious, look up "Skiplist"
*/

```

```

int SL_find(void *find, int level, SkipListNode *sl,
            (int)(*cmp)(void *, void *)){
    if(cmp(find, sl->data) == 0) return 1;
    if(level == 0 && sl->next[level] == null) return 0;
    if(sl->next[level] == null) {
        return SL_find(find, level-1, sl, cmp);
    }
    if(cmp(find, sl->next[level]->data) > 0){
        return SL_find(find, level-1, sl, cmp);
    }
    return SL_find(find, level, sl->next[level], cmp);
}

```

In translating the code `cmp(find, sl->next[level]->data)` into RISC-V we need to store arguments on the stack. So we will have `find` at `sp(0)`, `level` at `sp(4)`, `sl` at `sp(8)` and `cmp` at `sp(12)`. We're going to break up the translation into pieces

v. load `sl->next[level]` into `t0`

```
slli t1, t1, 4 add t0, t0, t1 lw t0, t0(4)
```

[helpful!](#) | 0



Caroline Liu 7 months ago

Hey! So the solution should say `slli t1, t1, 2` not 4 so yes, it would multiply by 4, not 16 (the term shifting by 4 versus 16 isn't correct because we're shifting by 2 and 4 respectively).

good comment | 0



Anonymous Mouse 6 7 months ago

Is it a typo to assign data and find both to a0 instead of the second to a1?

ALSO, in the same question v., why does it load word to 4(t0) instead of 0(t0)? Is that another typo?

Sorry, I just want to make sure.

helpful! | 0



Peyrin Kao 7 months ago

Yeah, I think data is supposed to go in a1, not a0 there.

The offset of 4 comes from the fact that `next` is the second field in the struct, so it appears 4 bytes after the start of the struct.

good comment | 0



Anonymous Mouse 6 7 months ago

But doesn't it already add 4 in part iv.? Why would it need to access `next` in the struct it reaches? Isn't `sl->next[level]` just going to the particular struct in the array?

helpful! | 0



Peyrin Kao 7 months ago

Yeah, that is a good point...I guess the 4 in part (v) should be a 0 then.

good comment | 0

Reply to this followup discussion

Resolved Unresolved

@909_f77



Anonymous Comp 2 7 months ago

Not really sure how to approach part b. I understand how to calculate the smallest number that is not representable, but not integer

PM1.2. Floating Point

For this entire section, you are not allowed to use the C programming language as a calculator, an actual calculator, a floating point simulator, nor any other computational aid.

Consider a 13-bit floating-point number with the following components (1 sign bit, 6 exponent bits, 6 mantissa bits); that is,

SEEEEEEMMMMMM

All other properties of IEEE-754 apply (bias, denormalized numbers, ∞ , NaNs, etc). The bias is the usual value of $-(2^{E-1}-1)$, which here would be $-(2^5-1) = -31$.

PART A (3 pts.)

What is the bit representation (in hex) of the floating-point number -1.25? Do NOT include the 0x prefix when writing your answer. Answers given in binary will not be given partial credit.

0x 17D0

? ✓ 100%

PART B (5 pts.)

What is the smallest non-representable positive integer under this scheme? Write your answer as a base 10 integer.

16777216

? ✗ 0%

helpful! | 0



Yiteng Zhou 7 months ago

Assume you can represent integer x , and when you add x 's mantissa by 1, the increased number $x' - x \leq 1$, then you can confidently say that you can express $x + 1$.

But when your step size is greater than 1 (such as 2), then you can't express the number between.

helpful! | 0



Peyrin Kao 7 months ago

This thread discusses a similar question from homework - @373_f7

good comment | 0

Reply to this followup discussion



Resolved



Unresolved

@909_f78



Yiteng Zhou 7 months ago

Can you clarify these two circumstances? I'm still confused about padding.

```
1. struct s1 {
    int a;
    char b;
};
```

run code snippet

Visit Manage Class to disable runnable code snippets ✕


Actually, I still think `sizeof(s1)` should be 8 not 5. If I think it's unclear, will it be clarified on exam?

```
2. struct s2 {
    another a; // sizeof(another) = 12
    int b;
    char c;
}

struct s3 {
    another a; // sizeof(another) = 12
```

```
double b;  
char c;  
}
```

run code snippet

Visit Manage Class to disable runnable code snippets 

Second, how do we compute sizeof(s2) and sizeof(s3)? I know char is kinda don't considered, but.. can you explain in detail where I should pad?

In lecture 3, there are some slide pages talking about padding, like:

```
typedef struct bar {  
    char *a; /* A pointer to a character */  
    char b[18]; /* A statically sized array  
of characters */  
} Bar;  
sizeof(Bar) == 24
```

run code snippet

Visit Manage Class to disable runnable code snippets 

helpful! | 0



 **Peyrin Kao** 7 months ago

@1059 (we'll answer there)

good comment | 0

Reply to this followup discussion

Start a new followup discussion

Compose a new followup discussion