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

note @908 292 views

Actions

[Past Midterms] 2020

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

Summer 2020 midterm 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: **SP20-MT1-Q1**, or **FA20-MT2-Q3**

FA20-MT1 Video Walk Through Links

Bit Manipulations Walkthrough

Slip Walkthrough

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 @908_f1

Michelle Li 7 months ago

[SU20-MT1-Q2]

iv. (0.75 pt) `sentinel.nxt->data`

- ○ Stack
- ○ Heap
- ● Static
- ○ Code

2. Doubly Linked Trouble!

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

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

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

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

For this question, part vi, the answer to sentinel.nxt -> data is static. I wasn't sure why this was static instead of stack? More generally, I'm a bit confused at the difference between char *r vs. char s[]. If someone could help that would be very much appreciated, thanks :)

helpful! | 0

Adelson Chua 7 months ago

Let me know if @176 helps.

good comment | 1

Michelle Li 7 months ago

that was very useful thank you :)

helpful! | 0

Reply to this followup discussion

Resolved Unresolved @908_f2

Anonymous Poet 7 months ago

[SU20-MT1-Q2]

(b) (3.0 pt) How many bytes of memory are allocated but not `free()`d by this program, if any? (assuming we have not called `free_list`) (Leave your answers as an integer. Do not include the units, we are telling you it's bytes after all!)

63

Each node will be `sizeof(node) == 12` bytes. We have allocated 4 nodes so 48 bytes.

We also made a `calloc` of 15 bytes (Since the compiler knows the length of the `s` array since it is stored on the stack which is 13 characters plus a null terminator so 14 bytes long). This means that we will leak 63 bytes.

2. Doubly Linked Trouble!

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

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

```
} node;

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

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

Why is the calloc 15 bytes? The second paragraph of the explanation only seems to account for 14 bytes?

helpful! | 0

Peyrin Kao 7 months ago

We allocated space for `sizeof(s) + 1` bytes. `sizeof(s)` is 14 bytes as explained, so adding 1 gives us 15.

good comment | 1

Anonymous Gear 7 months ago

SP20-MT1-4 is extremely similar to this problem. However, I'm confused why it's 62 for SP20, but 63 for SU20.

helpful! | 0

Peyrin Kao 7 months ago

SP20 uses `malloc(strlen(r) + 1)`, and `strlen(r)` evaluates to 13 (it doesn't count the null byte).

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f3**

Anonymous Poet 7 months ago

[SU20-MT1-Q2]

(c) Select all which is true for the following statements.

i. (1.0 pt) Which of the following interpretations allows for multiple different bit sequences to map to the same underlying value?

- ■ Sign and Magnitude
- ■ One's Complement
- ☐ Two's Complement
- ■ Floating Point
- ☐ Biased (for at least 1 choice of bias)
- ☐ Unsigned
- ☐ None of the other options

Is Floating Point a correct answer to this question because of NaN's?

helpful! | 0

Adelson Chua 7 months ago
Can be. Also zero!
good comment | 1

Anonymous Beaker 7 months ago
Why are the first 2 options correct?
helpful! | 0

Jero Wang 7 months ago
The first two are correct because in those representations, there are two representations for the value 0.
good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f4

Anonymous Atom 7 months ago
[SP-20-MT1-Q3]
Why does u[0] share the same byte as i[0]?
helpful! | 0

Anonymous Atom 7 months ago
Never mind I realize now the problem used a union and not a struct, however, I am still confused by why c is -1.
helpful! | 1

Anonymous Atom 7 months ago
Specifically I don't get how at the end of C we get ff ff ff 81
I would think negating would make almost all of the bits 0?
helpful! | 1

Peyrin Kao 7 months ago
The union is 4 bytes long in total. Each element in the union is 4 bytes long and they all share the same memory (which is why `u[0]` and `i[0]` are the same byte).

In question A and B, we put the byte -2 = `0xFE` in the lowest byte of the union. The upper 3 bytes are still `0x00`.

In question C, we negate `0x00 0x00 0x00 0xFE` to get `0xFF 0xFF 0xFF 0x02` (flip the bits and add one). In binary, this is `0b1111 1111 1111 1111 1111 1111 0000 0010`. Right-shifting by one gives us `0b1111 1111 1111 1111 1111 1111 1000 0001 = 0xFF 0xFF 0xFF 0x81`.
good comment | 1

Anonymous Atom 7 months ago
Thanks so much!

helpful! | 0

Anonymous Atom 7 months ago

I am a bit confused, in the first step when we say calloc(1, sizeof(union Fun)) doesn't that populate all the bytes with 1 initially?

helpful! | 0

Anonymous Atom 7 months ago

nvm

helpful! | 0

Anonymous Calc 2 7 months ago

For int8_t, do we use sign/magnitude or 2's complement?

helpful! | 0

Peyrin Kao 7 months ago

`int8_t` is a signed integer type so you would interpret the bits using two's complement.

good comment | 1

Anonymous Gear 2 7 months ago

can I ask how we get 255? for a

helpful! | 0

Peyrin Kao 7 months ago

`fun->i[0] = -1` sets the first byte of the union to 0xFF since `i` is a signed integer, and 0xFF in hex is 255 in decimal when interpreted as an unsigned number.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f5

Anonymous Helix 7 months ago

(c) Bloomin Onion

A very clever datastructure for efficiently and probabilistically storing a set is called a "bloom filter". It has two functions: `check` and `insert`. The basic idea for checking is that you hash what you are looking for multiple times. Each hash tells you a particular bit you need to set or check. So for checking you see if the bit is set. You repeat this for multiple iteration, with the hash including the iteration count (so each hash is different). If not all bits are set then the element does not exist in the bloom filter. If all bits are set then the element PROBABLY exists in the bloom filter. Similarly, for setting an element as present in a bloom filter you just set all those bits to 1.

We want to make a bloom filter design that is flexible and portable. So we define the following structure.

```
struct BloomFilter {
   uint32_t size; /* Size is # of bits, NOT BYTES, in the bloom filter */
   uint16_t itercount;
   uint64_t (*)(void *data, uint16_t iter) hash;
   uint8_t *data;
};
```

i. **(2.0 pt)** On a 32b architecture that requires word alignment for 32b integers and pointers, what is `sizeof(struct BloomFilter)` ?

16

ii. And now we have the insert function... For this we need to set the appropriate bit for each iteration.

```
void insert(struct BloomFilter *b, void *element){
    uint64_t bitnum; /* which bit we need to set */
    int i;
    for(i = 0; i < (CODE INPUT 1); ++i){
       bitnum = (CODE INPUT 2);
       b->data[bitnum >> 3] = (CODE INPUT 3);
    }
}
```

A. (1.0 pt) (CODE INPUT 1):

`b->itercount`

B. (3.0 pt) (CODE INPUT 2):

`b->hash(element, (uint16_t) i) % b->size`

C. (3.0 pt) (CODE INPUT 3):

`b->data[bitnum >> 3] | (1 << (bitnum & 0x7))` **(or equivalent)**

Could someone please explain the last line of code? Thanks!

helpful! | 0

Peyrin Kao 7 months ago

`data` is an array of bytes (8 bits each), and we want to set a specific bit given by the index `bitnum`. `bitnum >> 3` gives all but the lowest 3 bits of the index, which are used to choose what byte we want to set. `bitnum & 0x7` gives us only the lowest 3 bits of the index, which are used to choose which bit within the byte we want to set.

`1 << (bitnum & 0x7)` puts a 1 in the bit that we want to set. Then we OR that with the existing byte to flip that bit to 1.

good comment | 0

Anonymous Calc 7 months ago

midterm is this?

helpful! | 0

Peyrin Kao 7 months ago

SP20-Final-5C I think

good comment | 0

Anonymous Beaker 7 months ago

I'm still confused about the last line. Why do we need the lowest 3 bits (why 3 instead of other numbers like 2, 4, 0?) and how they're used to choose byte?

helpful! | 0

Peyrin Kao 7 months ago

Within one byte, there are 8 bits, so the bottom 3 bits of `bitnum` represents which bit we're looking at. For example, if you wanted to reference bit 22, this is the 6th bit in the second byte (zero-indexed) of the value. 22 = 0b10110. The top two bits 0b10 tell us to look in the second byte, and the bottom three bits 0b110 tell us to look at the 6th bit of this byte.

good comment | 0

Anonymous Scale 2 7 months ago

Hi, for code input 3 is `b->data[bitnum >> 3] | (1 << (bitnum % 8))` an equivalent expression to the answer?

I don't really understand the answer on the answer key so here's my thought process:

bitnum >> 3 gives the index for the uint8_t array since every element is 8 bytes. So if we take the bit number and divide by 8 that should give us the index.

bitnum % 8 will give you, for any byte, the bit # it is. So for bit 22, then bit >> 3 gives us 2 for the index and 6 for the bit number (bitnum % 8 == 6). Then we can do `1 << (bitnum % 8)` to get 00100000 <-- 6th bit will be turned on when OR'ed with what's already in that byte.

Is this a valid alternative to the solution

helpful! | 0

Peyrin Kao 7 months ago

Yeah, `x % 8` is equivalent to `x & 7`. The bitwise AND by 7 = 0b111 zeroes out all but the bottom 3 bits, which is equivalent to taking the number mod 8.

good comment | 1

Reply to this followup discussion

Resolved Unresolved @908_f6

Anonymous Mouse 7 months ago

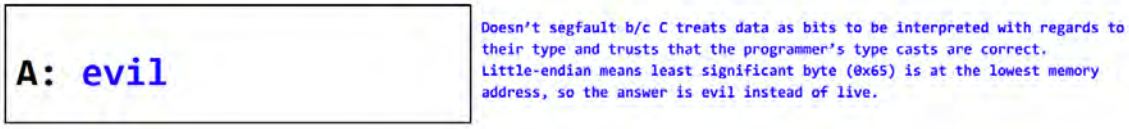
SP20-MT1-Q1c

Why is the value printed out "evil"? In terms of the bytes, I thought this would print (left to right) 65 76 69 6C, and the ASCII for these values does not correspond to the letters e,v,i, and l

The following code is executed on a 32-bit little-endian system.

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

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



helpful! | 0

Anonymous Mouse 7 months ago

Whoops that's in hex, have to convert to decimal *then* translate to ASCII!

helpful! | 0

Anonymous Scale 7 months ago

How do we know in this case the pointer increments by 8 bytes?

helpful! | 0

Caroline Liu 7 months ago

@Anon Scale do you mean increments by 8 bits? If that's the case, it's because the pointer is of type `char`, and `char` is 8 bits/1 byte. When we have an explicitly typed pointer, pointer arithmetic will increment by the size of the type, here, 1 byte.

If you mean 8 bytes, I'm not sure what you're referring to haha, so could you point me to where it says that?

Also I'm not entirely sure what you mean by the pointer incrementing by 8 [somethings] here? It doesn't seem like we're doing pointer arithmetic, from what I see.

good comment | 1

Anonymous Scale 7 months ago

I meant the former, this makes a lot more sense thank you.

helpful! | 0

Anonymous Atom 7 months ago

Are we expected to know how to convert between numbers and ascii?

helpful! | 1

Adelson Chua 7 months ago

ASCII table is included with the reference card.

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f7**

Anonymous Beaker 7 months ago

FA20-Quest-Q1

Where could we find solutions to Q1 in Fall 20 quest?

https://inst.eecs.berkeley.edu/~cs61c/sp21/pdfs/exams/Fa20_Quest_Solutions.pdf#page=1 only contains Q2-Q3.

helpful! | 0

Peyrin Kao 7 months ago

@757

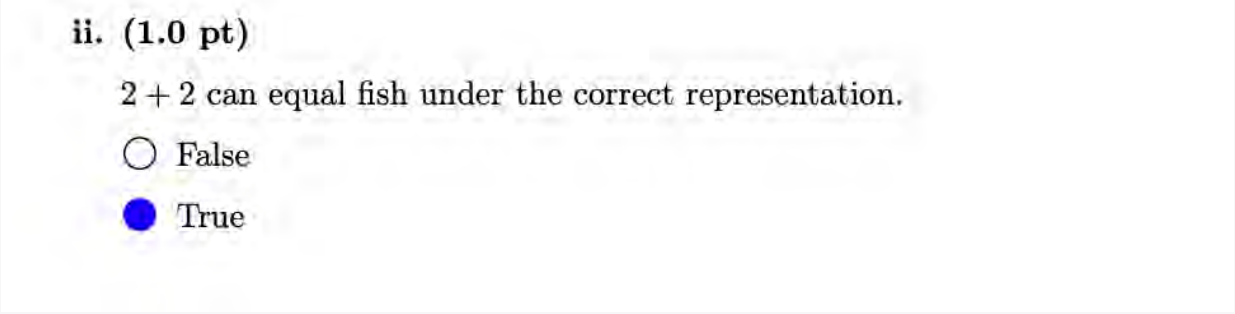
good comment | 1

Reply to this followup discussion

Resolved Unresolved **@908_f8**

Anonymous Beaker 7 months ago

[SU20-MT1-Q5(b)ii]What does it mean 'can equal fish' and how should we approach this question?



helpful! | 0

Jero Wang 7 months ago

Normally, in decimal representation, we know that `2 + 2 = 4`. We can come up with an arbitrary representation (let's call it "fish representation") where the word "fish" represents the decimal value of 4. So, the expression `2 + 2` can evaluate to and equals `"fish"`.

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f9**

Anonymous Comp 7 months ago

[SU20-MT1-Q5(e)(v)

I am not sure how to get the solution 0b10111011. What I did was first add the bias to 60

60 - 127 = -67 then convert -67 to binary using 2's complement. Then my solution is 0b10111101

helpful! | 1

Adelson Chua 7 months ago

@106

unbiased encoding - bias = biased encoding

You are given the unbiased encoding 0x3C = 60.

60 -(-127) = 187 (10111011)

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f10

Anonymous Gear 7 months ago

SU20-MT1-Q5C

For part i, I just want to make sure that the underlying value they are referring is 0 because each one of these representations can represent to two 0's.

For part ii, why is bias a correct answer?

(c) Select all which is true for the following statements.

i. (1.0 pt) Which of the following interpretations allows for multiple different bit sequences to map to the same underlying value?

- [x] Sign and Magnitude
- [x] One's Complement
- [ ] Two's Complement
- [x] Floating Point
- [ ] Biased (for at least 1 choice of bias)
- [ ] Unsigned
- [ ] None of the other options

ii. (1.0 pt) Which of the following interpretations allows us to deduce the sign just by looking at the most significant bit? (Ignore 0)

- [x] Two's Complement
- [x] One's Complement
- [x] Floating Point
- [x] Biased (for at least 1 choice of bias)
- [ ] None of the other options
- [x] Sign and Magnitude

helpful! | 0

Adelson Chua 7 months ago

part i) @908_f3

part ii) @106 You could set the bias such that the 0 in the original number line maps to the biased encoding of 01111.. (number of bits depending on the biased encoding number line). Then you can say that if MSB is 1, it is positive, if MSB is 0 it is negative (or 0).

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f11

Anonymous Scale 7 months ago

(b) Find the length of a null-terminated string in bytes. The function should accept a pointer to a null-terminated string and return an integer. Your solution must be recursive!

```
strlen:
    __<CODE INPUT 1>__
    beq t0, zero, basecase
```

```
__<CODE INPUT 2>__
__<CODE INPUT 3>__
__<CODE INPUT 4>__
jal strlen
__<CODE INPUT 5>__
__<CODE INPUT 6>__
__<CODE INPUT 7>__
ret
basecase:
__<CODE INPUT 8>__
ret
```

Fill in the following:

i. (0.75 pt) <CODE INPUT 1>

lb t0, 0(a0)

For i, why do we lb t0, 0a(a0), instead of lw?

(d) i. (1.0 pt)

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

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

0xABCDDBBC

di) Isn't this 0x100 + 0xABCDE000 = 0xABCDE100 + 0xABC = 0xABCDEBBC? Where is the discrepancy coming from?

(d) (4.0 pt) Translate the instruction at address 0x1C into machine code (in hex).

0x014000EF

	Imm[20|10:1|11|19:12]	rd	opcode
0b	0 0000001010 0 00000000	00001	1101111

if square's address is at 0x30 = 0000 0000 0000 0011 0000, how is 10:1 not 00 0011 0000?

helpful! | 0

Adelson Chua 7 months ago

First question: because we are counting the bytes, so we load 1 byte at a time. Also the null terminator is also byte 0x00, so this is also something that can only be checked through lb.

Second question: 0xABC will be sign extended to 0xFFFFFABC. Do the binary math, the answer is consistent with the solution.

Third question: I can't verify this, can you point me on what exam this is?

good comment | 1

Anonymous Scale 7 months ago

Summer 2020

helpful! | 0

Adelson Chua 7 months ago

Ah, jalr jumps to PC+offset. You should calculate the offset from the current instruction. square is 5 instructions away. 5*4bytes = 20. Then since we don't include bit 0, 10:1 is 0...001010

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f12

Anonymous Poet 2 7 months ago

viii. (0.75 pt) <CODE INPUT 8>

```
addi a0, zero, 0
```

For problem 3b, on Summer 2020 exam, how does the variable 'zero' work in this context. I'm assuming that we check if the current byte (char) is equal to zero, but at the end we addi a0, zero, 0 (which I thought would just place 0 in a0, when we want to put in the length of the string. How am I thinking about this incorrectly?

helpful! | 0

Adelson Chua 7 months ago

That's correct. It sets a0 = 0.

'zero' is x0. Not sure why they are using the MIPS (another processor, predecessor of RISC-V) naming scheme...

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f13

Anonymous Atom 2 7 months ago

SU20 Final Q7c & 7d

7. CALL

Suppose we have compiled some C code using the Hilfinger-Approved(TM) CS61Compiler, which will compile, assemble, and link the files `max.c` and `jie.c`,among others, to create a wonderful executable. After the code has been assembled to RISC-V we have the following labels across all files: `sean`, `jenny`, `stephan`, `philspel`, `poggers`, `crossroads`, and `segfault`. Assume no two files define the same label, though each file interacts with every label, either via reference or definition.

Note: **segment** refers to a directive in any assembly file, e.g. `.data` or `.text`

The CS 61Compiler begins to fill out the relocation table on the first pass of assembling `max.s`, which defines or references all of the labels above. This is its relocation table after the first pass:

label	address
`sean`	????
`stephan`	????
`jenny`	????
`segfault`	????
`philspel`	????

(c) (2.0 pt) After the second pass by the assembler, we see that `philspel` is no longer in the relocation table.

Which of the following is true about `philspel`? Select all that apply.

- [x] `philspel` is in the `.text` segment of `max.s`
- [ ] None of the other options
- [x] The address for `philspel` was resolved.
- [ ] `philspel` is in the `.text` segment of `jie.s`
- [ ] `philspel` is an external reference.

What does it mean and why philspel is in the .text segment of max.s? Isn't max.s another file?

(d) (2.0 pt) After assembling `jie.s` to `jie.o` we have the following symbol table for `jie.o`. In linking `max.o` and `jie.o` we get `dan.out`. Which of the following could be true about 'sean' and 'jenny' after linking? Select all that apply.

---

label	address
`sean`	0x061c
`jenny`	0x1620

- [x] They are in the same segment.
- [x] `sean` and `jenny` will have the same byte difference after linking as it did in `jie.o`.
- [ ] They are in different files.
- [x] `sean` and `jenny` are in different sections of `jie.s`.
- [ ] None of the other options

For 7d, what does it mean by different sections? How could we know they are in the same segment but different sections?

helpful! | 1

Anonymous Atom 2 7 months ago

Could anyone help with this?

helpful! | 0

Peyrin Kao 7 months ago

I'm checking in Slack...I'm honestly not totally sure what "sections" is referring to here.

good comment | 1

Anonymous Atom 2 7 months ago

Any follow-up on this?

helpful! | 0

Caroline Liu 7 months ago

Segments are referring to information that we use at runtime, AKA the static and data segments. Sections are things that're used during linking, and in this case, I think it's just generally referring to being literally located in different sections (the traditional definition of the word) of the same file.

This question is also asking what *could* be true; here, we could theoretically have `sean` and `jenny` be located in `static` or `code` separate from each other, or both be data labels or both be code labels.

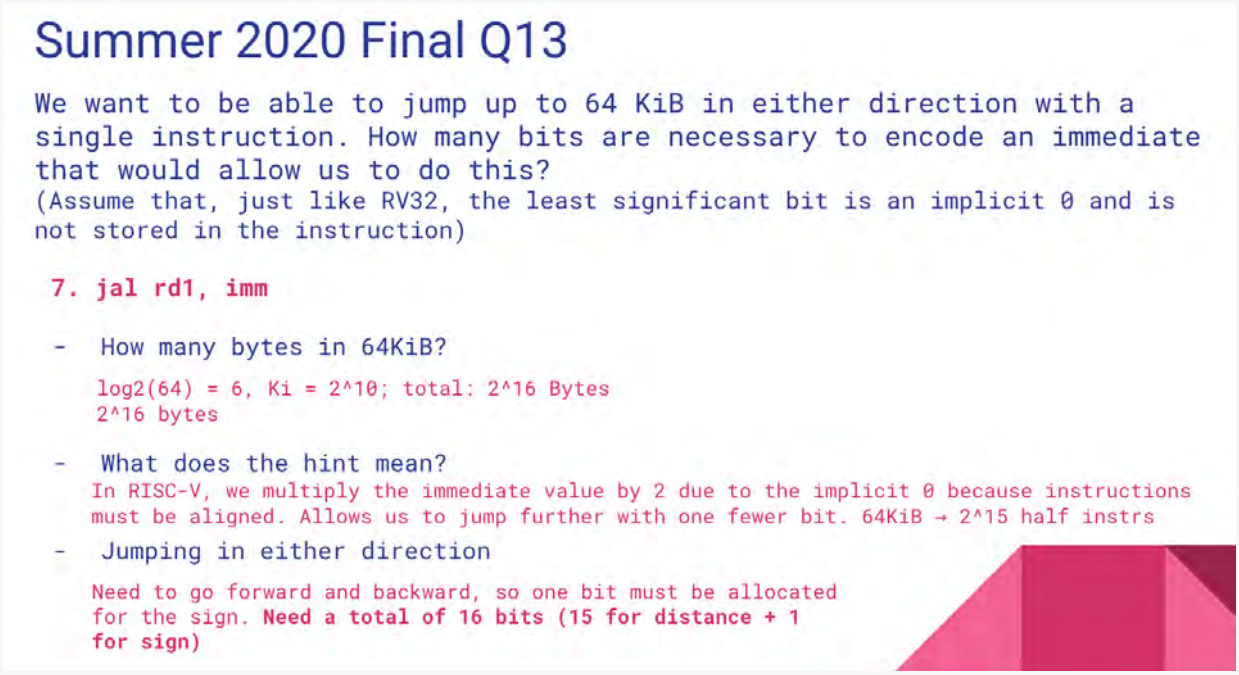
good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f14**

Anonymous Atom 2 7 months ago

SU20 Final Q13



Why do we jump by half instruction? I remember in lecture slides we use 32-bit instruction (which is 4 bytes)

helpful! | 0

Adelson Chua 7 months ago

RISC-V actually has support for 16-bit instructions, however, they are beyond the scope of 61C.

good comment | 0

Anonymous Atom 2 7 months ago

Sorry I'm getting confused, so why does this question use half instruction? In real exams, do we assume we would use 32-bit instruction or 16-bit half instruction?

helpful! | 0

Adelson Chua 7 months ago

Does it use half instructions? It just explained the reason for the implicit 0, which makes an immediate value divisible by 2.

good comment | 0

Anonymous Atom 2 7 months ago

But isn't that if we need to jump 2^16 bytes, which is equivalent to 2^14 32-bit instructions (since each instruction is 4 bytes), and the final result would be 2^(14+1) = 2^15? Could you please tell me where I'm going wrong?

helpful! | 0

Adelson Chua 7 months ago

I don't know what your question is actually.

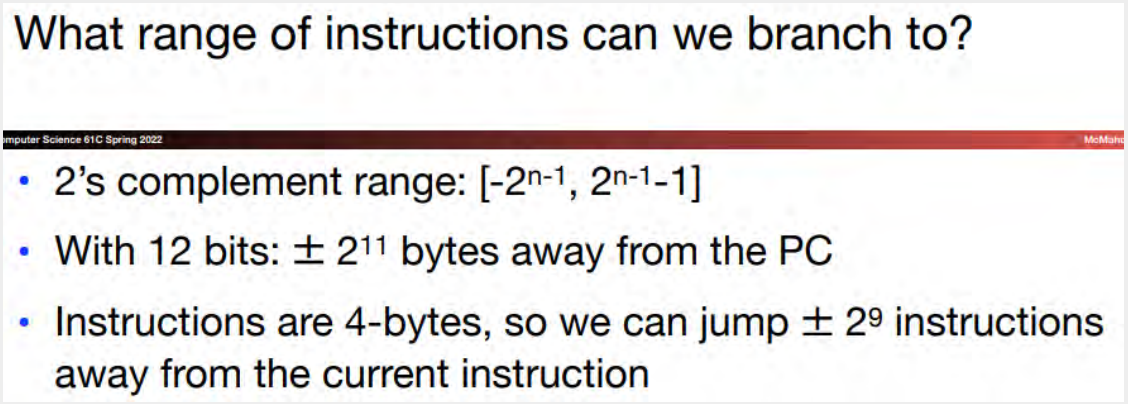
The question asks how many bits do we need to jump +-64kB = +-2^16. Add 1 bit for the sign, 17 bits. Minus 1 because we don't include bit 0, 16 bits.

The explanation regarding half-instructions might be confusing you, forget about it. Just know that in RISC-V, J and B type instructions omit bit 0. That's it.

good comment | 1

Anonymous Atom 2 7 months ago

Thank you for the explanation! I still have a question regarding the 2's complement range covered in class. Why do we not need to add an extra bit to account for this?



helpful! | 0

Adelson Chua 7 months ago

We don't have space anymore for the instruction encoding.

We cannot just randomly add bits to the immediate if we want, we are limited by the 32-bit instruction encoding. For branches, +-2^9 instructions is the best we can do.

good comment | 0

Anonymous Atom 2 7 months ago

so in the branch instruction, the bits are limited, but in this question, why don't we need to account for the two's complement range (i.e. add one more bit)?

helpful! | 0

Justin Yokota 7 months ago

The above slide is the first (naive) approach to branch immediates. Later slides on this lecture discuss the inclusion of the implicit zero, yielding a final jump range of +- 2^10 instructions.

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f15**

Anonymous Gear 7 months ago

FA20-FINAL-Q12Biii

Could someone explain how we reached the solution for part iii of 12B of the final?

helpful! | 0

Adelson Chua 7 months ago

Are you sure you are referencing the right exam? I don't see Q12...

good comment | 0

Anonymous Gear 7 months ago

This question: https://inst.eecs.berkeley.edu/~cs61c/sp21/pdfs/exams/Su20_Final_Solutions.pdf#page=41

helpful! | 0

Adelson Chua 7 months ago

You see, that's Summer, not Fall.

Also, why are you checking this? This is I/O, which is beyond the scope of the midterm...

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f16**

Anonymous Helix 7 months ago

Sp20 Midterm Q3

Q3) *Unions* (8 pts = 4 * 2)

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

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

Write what each print statement will print out in the corresponding box. Assume that this system is little-endian and that right shifts on signed integers are arithmetic.

a)

255

```
};

int main() {
      union Fun *fun =
        calloc(1, sizeof(union
Fun));

      fun->i[0] = -1;

      //Question a
      printf("%u\n", fun->u[0]);

      fun->u[0] *= 2;

      //Question b
      printf("%d\n", fun->i[0]);

      fun->t *= -1;
      fun->t >>= 1;

      //Question c
      printf("%d\n", fun->i[1]);

      fun->s[0] = '\0';
```

Because we are asking for the unsigned representation of the first byte in the union, we will get the value 255 instead of -1.

b)

-2

Multiplying u[0] by -2 multiplies i[0] by -2 as well (since they share the same bytes). Thus, the value printed is -2.

c)

-1

We negate all the bytes in the union and right shift it by one. This gives us 0xffffff81. Because of little-endianness, we want the second to last element– which is 0xff. This is equivalent to -1 in 8-bit signed decimal.

d)

Hi, could anyone kind of walk through this problem? Esp part c, thanks!

helpful! | 0

Peyrin Kao 7 months ago

@908_f4 maybe? Feel free to follow up if you have more specific questions.

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f17**

Anonymous Atom 2 7 months ago

SU20 Q7 (a)(ii)

6. Don't Float Away!

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

Quick reminder about intervals: (and) are exclusive while [and] are inclusive.

(a) i. (1.5 pt)

What's the gap (aka absolute value of the difference) between the **smallest positive** non-zero denorm and **smallest positive** non-zero norm? (Answer in decimal)

0.234375

$1.0000_2 * 2^{1-3} - 0.0001_2 * 2^{0-3+1} = \frac{1}{4} - \frac{1}{64} = \frac{15}{64} = 0.234375$

ii. (1.5 pt)

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

31

$2^5 - 1 = 31$

Could someone explain the(ii) question? how come the term 2^5 - 1?

helpful! | 0

Anonymous Atom 2 7 months ago
Sorry it should be Q6

helpful! | 0

Adelson Chua 7 months ago
You have 4 mantissa bits, That has 2^4 = 16 possible combinations. The exponent changes twice going from 2^1 to 2^3, for every change in exponent you get a fresh set of mantissa combinations. so 16*2 = 32.

Subtract 1 so that you don't include 2^1 itself since the range is described within ().

So 32-1 = 31.

good comment | 2

Reply to this followup discussion

Resolved Unresolved **@908_f18**

Anonymous Scale 7 months ago

b) You receive the data "0b1110xxxxxxxxxxx". What is the **decimal value** of the **smallest number** the sender could have sent (i.e. it is less than all of the other possibilities)? You must provide the decimal form, ***do not leave as a power of 2***.

-8188

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

Wouldn't the smallest be when the mantissa is just 0, because 1 < 1.11111... ? Also how is 1.1111.... = 2-2^10?

helpful! | 0

Adelson Chua 7 months ago
I believe this question is trying to get the most negative number, which is the smallest. It is not talking about the smallest in absolute value.

good comment | 0

Anonymous Scale 7 months ago
Thank you. Also how is 1.1111.... = 2-2^10?

helpful! | 0

Peyrin Kao 7 months ago
0b10.0000 0000 00 = 2

0b0.1111 1111 11 = 0b10.0000 0000 00 - 0b0.0000 0000 01 = $2 - 2^{-10}$

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f19

Anonymous Helix 2 7 months ago

Q:
li t0, 0xABCDEFAD
sw t0, 0(s0)
lb t0, 0(s0)

Write down the value of t0 in hex. Assume big-endianness. Reminder: include the prefix in your answer!
Answer: 0xFFFFFFAB

I do not really understand why it is AB at the back, I understand the most significant bit A should be at the back due to big endianness but I am confused why the B is there.

helpful! | 0

Adelson Chua 7 months ago

What do you mean by 'back'?

The byte that was loaded was 0xAB (remember 1 byte = 2 hex digits). Then the 0xFFF.. are the result of sign extension.

good comment | 1

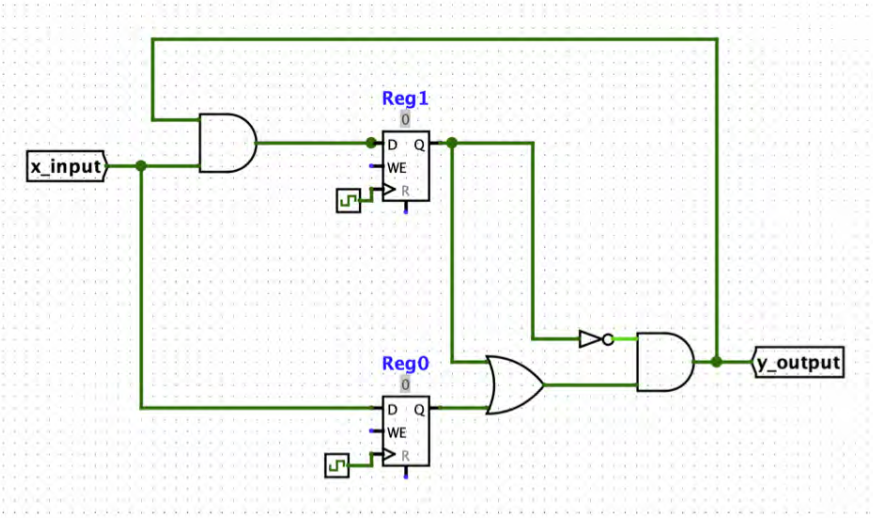
Reply to this followup discussion

Resolved Unresolved @908_f20

Anonymous Mouse 2 7 months ago

4. SDS, Logic

We will be analyzing the following circuit:



Circuit

Given the following information:

- **AND** gates have a propagation delay of 9ns
- **OR** gates have a propagation delay of 14ns
- **NOT** gates have a propagation delay of 5ns
- **x_input** switches value(i.e. 1 to 0, 0 to 1) 30 ns after the rising edge of the clk
- **y_output** is directly attached to a register
- **Setup** time is 3ns
- **Clk-to-q** delay time: 4ns

(a) (2.0 pt) What is the max hold time in ns?

18

Shortest CL: NOT -> AND = 5 + 9 = 14ns clk-to-q + shortest CL = 4ns+14ns = 18ns

(b) (2.0 pt) What is the minimum clock period in ns?

42

Critical path = clk-to-q + longest CL + setup = 30ns for x_input to change (includes clk-to-q) + 9 AND + 3 setup = 42 ns

SU20-Final-Q4.Why is the shortest CL 14? Shouldn't it be NOT + AND + AND? Do we not take into account the AND at the top right corner, but then we cannot reach a register?

helpful! | 0

Anonymous Mouse 2 7 months ago

Also why is the minimum clock period 42? clk-to-q is 4, longest cl is 14 + 9 + 9 and setup is 3. Why do we need to take into account the 30ns? And why is the longest CL just 9?

helpful! | 0

Adelson Chua 7 months ago

The shortest CL goes from the output of Reg 1 going to the y_output. The problem says that y_output is connected to a register.

Trace all the paths going to the input of Reg 1. As the problem stated, x_input is like a clk-to-q from the input side. Calculate all of the delays. You'll see that the 30ns "clk-to-q" of x_input dominates all the rest of the delays. The longest path starts from x_input to AND to input of Reg 1.

good comment | 0

Anonymous Beaker 7 months ago

Is the x_input also a state element? What counts as state elenents?

helpful! | 0

Adelson Chua 7 months ago

It's not technically a state element. However, the behavior, as defined in the problem, makes it seem like x_input is also synchronized with the clock. Thus, it's delay can be thought of as a clk-to-q delay when calculating path delays.

good comment | 0

Anonymous Beaker 7 months ago

Sorry, I'm still confused about why we include x_input when counting the longest combinational logic delay. Since we only consider paths between 2 state elements, then shouldn't we only

consider potential paths register 1 and register 0?

helpful! | 0

Adelson Chua 7 months ago

I know. But again, since x_input was defined by the problem being **synchronous to the clock**, we see it as a "register".

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f21

Anonymous Gear 7 months ago

For SU20-MT1-2A, why is v stack, but the next two heap?

v. (0.75 pt) sentinel.prv->prv->data

- ○ Static
- ○ Heap
- ● Stack
- ○ Code

vi. (0.75 pt) sentinel.prv->data

- ○ Static
- ● Heap
- ○ Stack
- ○ Code

vii. (0.75 pt) sentinel.prv->prv

- ○ Code
- ● Heap

○ Stack

○ Static

helpful! | 0

Peyrin Kao 7 months ago

`sentinel.prv->prv` points to the second-last element in the linked list, which is `&sentinel`. This is a pointer to the struct on the stack, so if we dereference it with `sentinel.prv->prv->data`, we get a struct field on the stack.

`sentinel.prv` points to the last element in the linked list, which is `calloc(sizeof(s) + 1, sizeof(char))`. This is a pointer to the heap, so dereferencing it gives us struct fields on the heap.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f22

Anonymous Atom 2 7 months ago

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

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

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

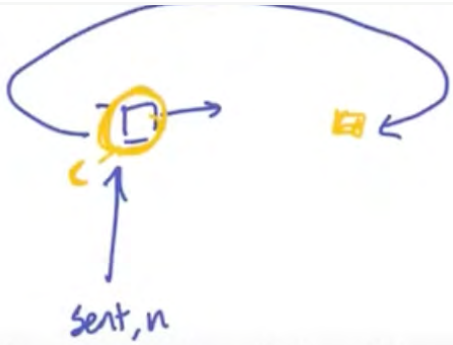
● False

○ True

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

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

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



According to the walkthrough, wouldn't free_list just skip the sentinel node? Also, why does the code only free the sentinel on the stack but not in the heap?

helpful! | 0

Anonymous Atom 2 7 months ago
BTW it's SU20 MT1 Q2c

helpful! | 0

Anonymous Atom 2 7 months ago
Could someone help me with this?

helpful! | 0

Peyrin Kao 7 months ago
~~I think it would cycle back to the sentinel node eventually, since it's a doubly-linked list.~~ The sentinel was never allocated on the heap in the code, I think.

good comment | 0

Anonymous Atom 2 7 months ago
But isn't that once it detects that "c == n," it will just end the for loop? (i.e. when it finally reaches the sentinel node, it will jump out of the for loop and does not free the sentinel node?

helpful! | 0

Peyrin Kao 7 months ago
Yeah, that's right, my mistake. Actually I think it might be trying to free the sentinel because one of the linked list items is a pointer to the sentinel: `push_back(&sentinel, &sentinel);`

good comment | 1

Anonymous Atom 2 7 months ago
Ohh that makes sense, thanks!

helpful! | 0

Reply to this followup discussion

Resolved Unresolved **@908_f23**

Anonymous Helix 2 7 months ago
auipc t0, 0xABCDE # Assume this instruction is at 0x100
addi t0, t0, 0xABC
Write down the value of t0 in hex. Reminder: include the prefix in your answer!

Ans: ABCDDBBC

I understand that the sign extension in 0xABC causes 0xABCDE to become ABCDD, but I dont see why the last 3 hex digits become BBC.

helpful! | 0

Adelson Chua 7 months ago

There's an auipc instruction before that addi instruction that added 0x100. (0x100 was the PC, and auipc adds the immediate to the PC)

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f24**

Anonymous Beaker 2 7 months ago

fa20-final-q5.b

Can someone explain the intuition behind 2048? I thought we can only change bit 20-31 which is 12 bits. but 2^12 doesn't give the right answer, 2^11 does. doesnt adding 16 change bit 20?

helpful! | 0

Anonymous Beaker 2 7 months ago

got it. it's because if we reach the last bit, bit 31, then the immediate becomes -2048 (2's complement) which is not an accurate representation of the number of times we've used the function so 2^11

helpful! | 0

Reply to this followup discussion

Resolved Unresolved **@908_f25**

Anonymous Beaker 2 7 months ago

fa20-final-q5.c

can someone explain the hexcode in binary?

helpful! | 0

Adelson Chua 7 months ago

Can you post a picture here? I can't see the question in FA20 final...

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f26**

Anonymous Beaker 2 7 months ago

su20-final-q12

how is 0x108 - 0x100 = 8 bytes? I thought it'd be a difference of 8 bits? what does this look like in binary?

helpful! | 0

Adelson Chua 7 months ago

0x108 and 0x100 are memory addresses. Memory is byte addressable, so the difference is in bytes.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f27

Eric Lu 7 months ago

SP20-MT1-Q5b

(b) Having discovered the identity, you follow it and find a large array of double precision floating point (type **double**). The clue says you want the 5th smallest element casted to an integer. True, you could just go through the array but, being a proper CS student, you decide to first sort the array using a library function and then take the 5th element. Fortunately, C has a quicksort function in the standard library:

```
void qsort ( void * base, size_t num, size_t size,
             int ( * comparator ) ( const void *, const void * ) );
```

That is, the function takes four arguments: a pointer to the array, the total number of elements, the size of each element of the array, and a comparison function. The comparison function should return negative if the first element is less than the second, 0 if they are the same, or positive if the first element is bigger. Your code should compile without warnings.

```
int comp(void *p1, void *p2){
    double a = *((double *) p1);
    double b = *((double *) p2);
    return a-b ; /* C will cast a double to an int automagically */
}
```

Could someone explain why this function for comp works correctly please?

helpful! 0

Adelson Chua 7 months ago

I'm not sure what is unclear with it.

(double *) p1 basically typecasts p1 as a pointer to a double. This is identical to saying 'hey treat p1 as an address pointing to a double'.

The preceeding * will get the value at that pointer, giving you a number of type double. Then you just subtract the two numbers.

good comment 0

Eric Lu 7 months ago

My bad, fully misread the question for the subtraction part.

helpful! 0

Reply to this followup discussion

Resolved Unresolved @908_f28

Anonymous Comp 2 7 months ago

SP20-MT1-Q5b

(c) You arrive at the room, only to find a door locked with a keycode. Spray painted on the wall, you see *"How many stairwells have a power-of-two number of steps? Print the answer **in hex**..."*

So close to your goal, you crowdsource this question to your favorite social media. Enlisting a friend taking CS 186, you end up with an array of step counts for all stairs which are all positive integers. Create a function to see the total number of stairwells with exactly a power of 2. Hint: you know ***X*** is a power of 2 if and only if ***X*** and ***X-1*** have no bits in common and ***X*** is nonzero. You do not need to use all the lines.

There are multiple valid methods to approach this question. The staff solution requires the least number

of lines, and uses the hint that x and x-1 have no bits in common for a power of 2. A power of 2 is found for any non-zero value where the logical and of x and x-1 results in a zero value (thus `stairs[i]` is checked to ensure a non-zero value, then `(stairs[i] & (stairs[i]-1))` is tested for a zero value. Another way to check if the entry is a power of two which is possible with the given lines is to test each bit within the entry and make sure only one bit has a 1 value. Finally, `printf` is called with %x, printing the number of entries in hexadecimal (given by the chart below).

```
void pows_of_2(unsigned int* stairs, int len) {
    int matching_entries = 0;
    _____________________________________________;
    _____________________________________________;
    for (int i = 0; i < len; i++) {
        _____________________________________________;
        if (stairs[i] && (stairs[i] & (stairs[i]-1)) == 0) {
            matching_entries += 1;
        }
```

In the last line of solution:

Would (~(stairs[i] ^ (stairs[i] - 1) == 0) also be an acceptable solution? The inside parenthesis stairs[i] XOR stairs[i-1] will only be 1 if all of the bits are not in common, and the outside ~ checks if ~1111111 = 0 , does actually equal 0.

helpful! | 0

Anonymous Comp 2 7 months ago

5c***

helpful! | 0

Anonymous Calc 2 7 months ago

Or can I just do "if (stairs[i] ^ (stairs[i]-1) { ---}" ?

Since this will output 1 when two different bits so will 1 is true so will execute the if statement.

helpful! | 0

Peyrin Kao 7 months ago

Both of those approaches look right to me. (No guarantees though - if you want to be absolutely sure, I'd try actually running this code in C.)

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f29**

Anonymous Calc 2 7 months ago

Can I write (*(info + i))[3] as an alternative solution?

```
for (int i = 0; i < len; i++) {
    info_to_print[i] = (char) info[i]>>24 /* Others possible as well */;
}
```

helpful! | 0

Peyrin Kao 7 months ago

Not sure what question this is, but `(*(info + i))[3]` performs two dereference operations, and

`info[i]>>24` performs one dereference operation, so they're not equivalent.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f30

Anonymous Gear 2 7 months ago

For Sum20 MT1 question 1a, the reason we have 32 bits for next pointer is because we state in the question right?

helpful! | 0

Peyrin Kao 7 months ago

If you're asking why the pointer is 32 bits, it's because in a 32-bit system, pointers/addresses are always 32 bits. If you're asking why a `next` pointer exists, it's to support the linked list structure we're building.

good comment | 0

Anonymous Gear 2 7 months ago

if we have 64 bit system, and rest of condition is unchanged, then the total bytes is 12 by 8+4?

helpful! | 0

Peyrin Kao 7 months ago

That sounds right to me. The elements in the union don't depend on the size of a word.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f31

Anonymous Calc 2 7 months ago

Why do we put the address of comp function not just comp ?

Also, comps needs to output 1 if the first element is bigger but wouldn't a - b output a number bigger than 1 if a is a lot greater than b?

```
int comp(void *p1, void *p2){
    double a = *((double *) p1);
    double b = *((double *) p2);
    return a-b ; /* C will cast a double to an int automagically */
}
void clue2(double* info2, int len) {
    qsort(info2, len, sizeof(double), &comp);
    printf("%i\n", (int) info2[4] );
}
```

helpful! | 0

Peyrin Kao 7 months ago
What exam is this?
good comment | 0

Anonymous Calc 2 7 months ago
SP20 mt1 Q5.b
helpful! | 0

Peyrin Kao 7 months ago
The comparison function only needs to return a positive number if a is greater than b, according to the question.

I think `&comp` and `comp` are equivalent here - both would be treated as a function pointer.
good comment | 1

Reply to this followup discussion

Resolved Unresolved @908_f32

Anonymous Mouse 2 7 months ago

Let's consider the hexadecimal value 0xFA000003. How is this data interpreted, if we treat this number as...

(a) an array A of unsigned, 8-bit numbers? Please write each number in decimal, assume the machine is little endian. If the value is unknown, write GARBAGE (in all caps).

i. (0.5 pt) A[0]

3

ii. (0.5 pt) A[1]

0

iii. (0.5 pt) A[2]

0

iv. (0.5 pt) A[3]

250 (0xFA in decimal)

(b) (2.0 pt) a IEEE-754-style floating point number, but which uses only 7 bits for the exponent with a bias of 64 (where we subtract the bias)? Write out as binary scientific notation, so e.g, an answer that looks like this:. -1.0100100 * 2^15

$-1.00000000000000000000011 * 2^{58}$

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

```
lb x0, -96(x0)
```

(d) (2.0 pt) a `(uint32_t *)` variable `c` in **big-endian** format, and we call `printf("%i", (int) ((uint8_t *) &c)[0])`? If the value is unknown, write GARBAGE (in all caps).

250 (0xFA in decimal)

SP20-Final-Q1d

For part b, I'm confused about the FA, is the unit8_t* pointing to A? Also how does the & sign works in this case?

helpful! | 0

Anonymous Mouse 2 7 months ago

For part d*

helpful! | 0

Adelson Chua 7 months ago

The question is independent of array A. This is saying you declared a variable uint32_t *c = 0xFA000...3.

The printf stuff is basically typecasting *c as a uint8_t pointer instead of the original uint32_t pointer, reinterpreting it as an int, then printing it.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f33

Anonymous Helix 2 7 months ago

How many Floating Point numbers are in the interval of (2^1, 2^3)? (Answer in decimal)
In a Floating Point scheme of 3 Exponent, 4 significand, and bias of -3.
Ans: 31
2^5 -1 =31

I am not sure where the 2^5 comes from.

helpful! | 0

Adelson Chua 7 months ago

@908_f17

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f34

Anonymous Gear 2 7 months ago

For Su20 -Final, Q7d, what is byte difference meaning in the text? and why is correct?

helpful! | 0

Caroline Liu 7 months ago

`sean` and `jenny` will have the same byte difference after linking as it did in `jie.o`.

Byte difference is the number of bytes in memory the two labels are separated by. This is true when the two labels fall into the same segment because we don't shift lines of assembly around within a single segment, but we do shift segments around. Thus, two labels in different segments in the same file might not necessarily remain the same number of lines/bytes away from each other after linking but if they're in the same segment, it's possible.

The question is asking which of the options is **possible** so it's possible both are in the same segment. Hope that helps!

good comment | 0

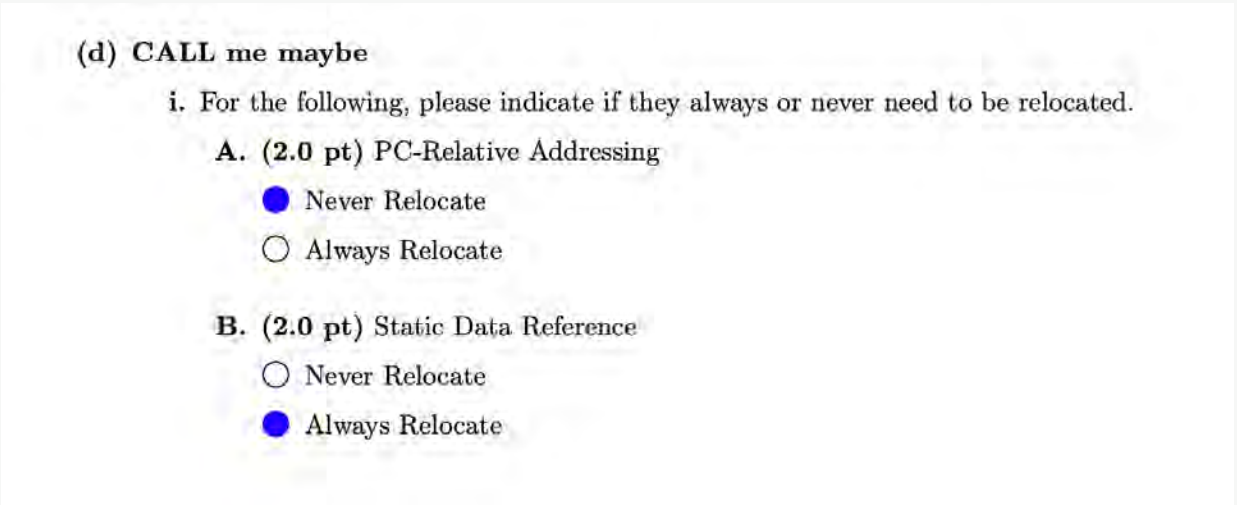
Reply to this followup discussion

Resolved Unresolved @908_f35

Anonymous Beaker 7 months ago

sp20 - final -5d(i)

Could someone summarize what is relocated and what is not? I'm pretty lost when approaching these questions.



(d) CALL me maybe

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

A. **(2.0 pt)** PC-Relative Addressing

- (●) Never Relocate
- ( ) Always Relocate

B. **(2.0 pt)** Static Data Reference

- ( ) Never Relocate
- (●) Always Relocate

helpful! | 0

Caroline Liu 7 months ago

This is pulled directly from lecture slides, but as a general rule of thumb, when you have addresses/references that eventually is going to be defined as an offset to something else anyways, that doesn't need to be relocated because well, you don't need to know definitively where it is sitting in absolute memory anyways so even if the absolute, relocated addresses is calculated, it's never needed anyways. That's why PC-relative addressing is never relocated (it depends on the PC value eventually).

Static data references though, need to be relocated because they're direct references to the static segment in memory and thus, the address to the reference needs to be calculated once all the components are "in-place" when all the files are linked. This also encompasses external references with absolute/0-relative addressing!

Hopefully that helps!

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f36

Anonymous Mouse 2 7 months ago

We have 9 instructions, so we need 4 bits to represent them.

(b) **(3.0 pt)** We want to be able to jump up to 64 KiB in either direction with a single instruction. How many bits are necessary to encode an immediate that would allow us to do this? Assume that, just like RV32, the least significant bit is an implicit 0 and is not stored in the instruction.

16

16 bits

64 KiB = 2^{16} B. Jal is the only jump instruction given, so it will determine the size of the immediate field. Recall that, in RISC-V, the immediates encoded in instructions work in units of half-instructions, so jumping up to 64 KiB in either direction is the same as saying we want to jump up to 2^15 half instructions in both directions. Since immediates are signed, we need 16 bits to represent this range of values.

SU20-Final-Q13b, what does it mean that we are not storing the least significant bit? If we want to jump in either direction up to 64Kib, then we need 16 bit to store the value, and 1 more bit to store the direction right? Why isn't it 17bit?

helpful! | 0

Peyrin Kao 7 months ago

The last bit of the immediate is not stored in jal instructions because it's always 0. Storing 16 bits gives us a 17-bit immediate.

good comment | 0

Anonymous Mouse 2 7 months ago

Why is it always 0? Than means we cannot jump with an offset of 1?

helpful! | 0

Peyrin Kao 7 months ago

Yeah, you can't jump by an odd number of bytes (it should never be necessary anyway because instructions are always located at multiples of 4). Check out the RISC-V lectures for more about this!

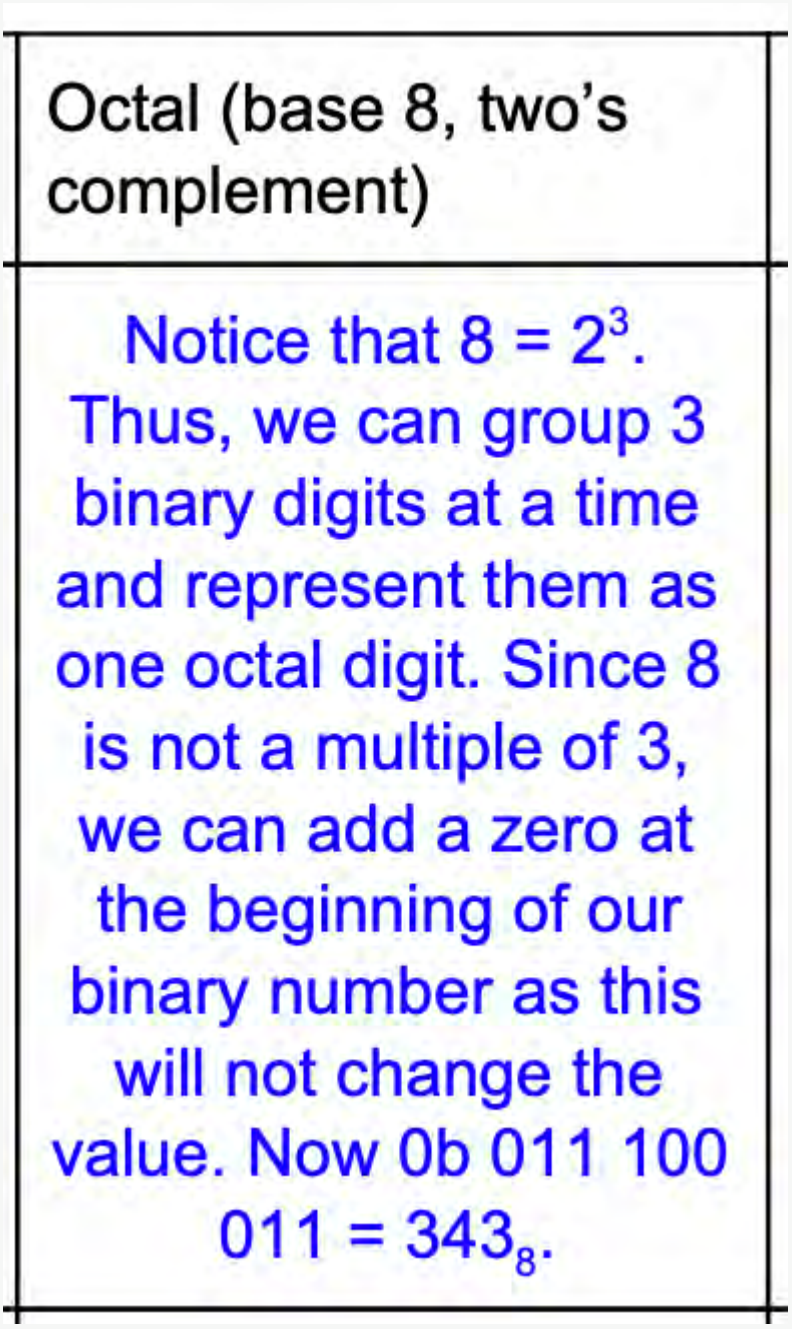
good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f37

Anonymous Poet 3 7 months ago

SP20-MT1-Q2



For this question, I'm not sure why we extend with zero, since this is in two's complement. wouldn't this change the sign to be positive?

Shouldn't we sign extend with a 1 instead? and get 743 instead

helpful! | 0

Peyrin Kao 7 months ago

That logic sounds right to me.

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f38

Anonymous Mouse 2 7 months ago

(b) Find the length of a null-terminated string in bytes. The function should accept a pointer to a null-terminated string and return an integer. Your solution must be recursive!

```
strlen:
    __<CODE INPUT 1>__
    beq t0, zero, basecase
    __<CODE INPUT 2>__
    __<CODE INPUT 3>__
    __<CODE INPUT 4>__
    jal strlen
    __<CODE INPUT 5>__
    __<CODE INPUT 6>__
    __<CODE INPUT 7>__
    ret
  basecase:
    __<CODE INPUT 8>__
    ret
```

Fill in the following:

i. **(0.75 pt)** <CODE INPUT 1>

```
lb t0, 0(a0)
```

ii. **(0.75 pt)** <CODE INPUT 2>

```
addi sp, sp, -4
```

iii. **(0.75 pt)** <CODE INPUT 3>

```
sw ra, 0(sp)
```

iv. **(0.75 pt)** <CODE INPUT 4>

```
addi a0, a0, 1
```

v. **(0.75 pt)** <CODE INPUT 5>

```
addi a0, a0, 1
```

vi. **(0.75 pt)** <CODE INPUT 6>

```
lw ra, 0(sp)
```

vii. **(0.75 pt)** <CODE INPUT 7>

```
addi sp, sp, 4
```

SU20-MT1 -Q3

Can someone explain where did we store our return value? I though we would need to store the return value to a0, but it seems like we are just keep incrementing a0 by 1 to point to the next character in our string, but at the end, we did not get the actual strlen

helpful! | 0

Peyrin Kao 7 months ago

The solution is recursive, so each call to strlen returns a number, and then 1 is added to that number. The base case returns 0 in a0.

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f39**

Anonymous Gear 7 months ago

SU20-MT1 -Q3C

How does xoring with a neg 1 work?

(c) Arithmetically negate a Two's Complement 32-bit integer without using the sub, mul or pseudo instructions.

```
negate:
  __<CODE INPUT 1>__
  __<CODE INPUT 2>__
  ret
```

Fill in the following:

i. (0.75 pt) <CODE INPUT 1>

```
xori a0, a0, -1
```

ii. (0.75 pt) <CODE INPUT 2>

```
addi a0, a0, 1
```

helpful! | 0

Adelson Chua 7 months ago

Do it on a piece of paper. Write any combination of bits, then xor each bit with 1 (since -1 in binary = 1111...111). It basically performs the bitwise inversion.

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f40**

Anonymous Atom 3 7 months ago

Sp20, MT1, Q2

Decimal	Binary (Two's complement)	Octal (base 8, two's complement)

-29	We first find the binary representation of 29, which is 0b00011101. We then flip all the bits and add 1, and that gives us 0b11100011	Notice that $8 = 2^3$. Thus, we can group 3 binary digits at a time and represent them as one octal digit. Since 8 is not a multiple of 3, we can add a zero at the beginning of our binary number as this will not change the value. Now 0b 011 100 011 = 343_8.

Are we not using 9, and not 8, bits for the octal representation? Is there no restriction on the number of bits for the octal representation (i.e. is it just for binary)?

helpful! | 1

Adelson Chua 7 months ago

Octal is literally base 8. Think of it like hex, but instead of grouping the binary bits by 4 (as hex is base 16), you group them by 3 (for base 8).

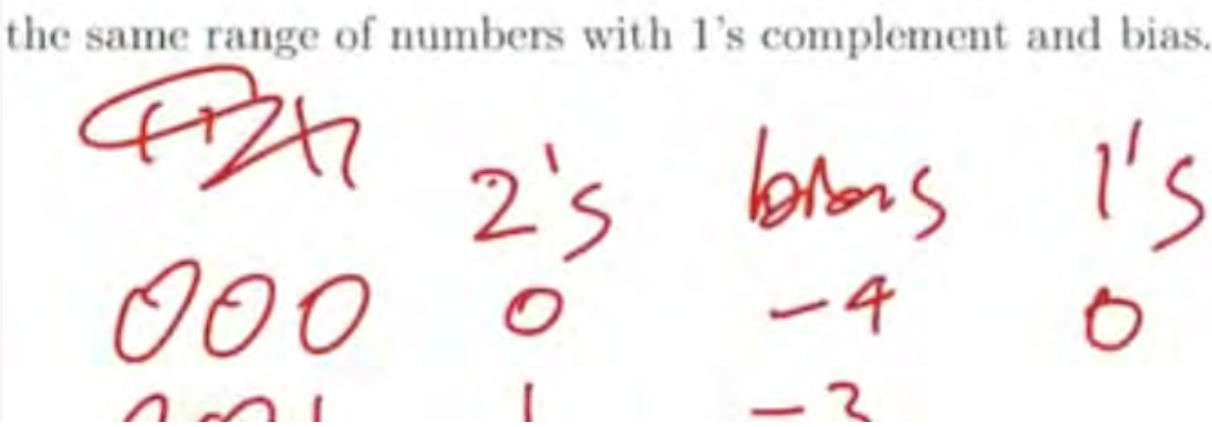
Since it is a grouping by 3 binary bits at a time, all octal representation number of bits are technically divisible by 3. There's no way around that, you can't write octal with just 2 bits. Then again, putting a 0 at the MSBs does not affect the value of the number, so that's what is important.

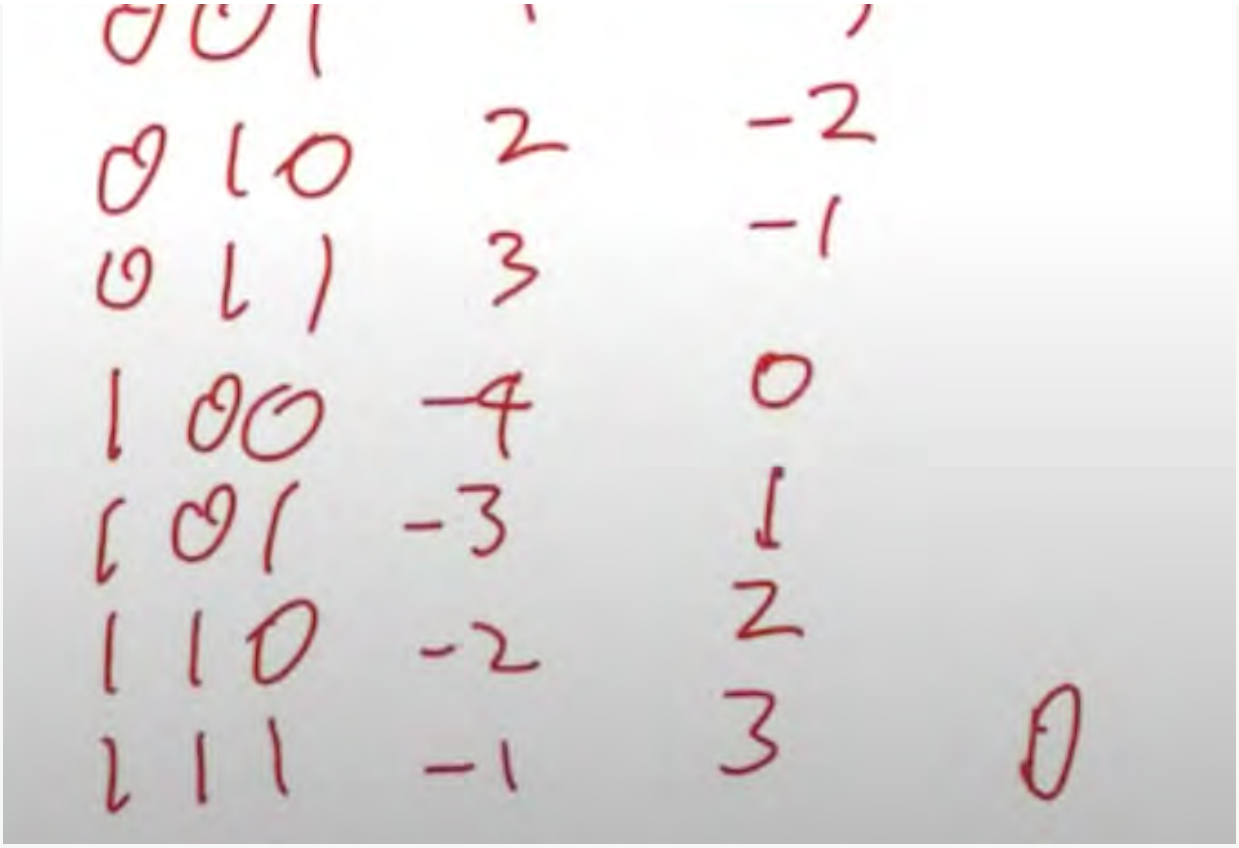
good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f41

Anonymous Helix 3 7 months ago





How is bias calculated here? Shouldn't it be -3 for 000 since equation is -(2^n-1 - 1) = -3 and then -3 - 0 = -3

helpful! | 0

Anonymous Helix 3 7 months ago

For q5bi

helpful! | 0

Adelson Chua 7 months ago

There's no real standard in setting what the bias should be, actually. The formula you cited is the standard for the floating point representation, but for any other applications, it's typically either -(2^n-1 - 1) or -(2^n-1)

Did the actual value of the bias matter for this problem?

good comment | 0

Reply to this followup discussion

Resolved Unresolved @908_f42

Anonymous Helix 3 7 months ago

(a) i. (1.5 pt)

What's the gap (aka absolute value of the difference) between the **smallest positive** non-zero denorm and **smallest positive** non-zero norm? (Answer in decimal)

0.234375

$1.0000_2 * 2^{1-3} - 0.0001_2 * 2^{0-3+1} = \frac{1}{4} - \frac{1}{64} = \frac{15}{64} = 0.234375$

How do you know what the exponents are for 2^

helpful! | 0

Adelson Chua 7 months ago

I'm guessing the bias for this problem is -3? It should either be stated somewhere there or it has been indicated that there are 3 exponent bits, since bias would then be -(2^n-1 - 1) = 3.

The smallest exponent field for a normal number is 1.

For denorm, the exponent is fixed to bias+1.

good comment | 0

Reply to this followup discussion

Resolved Unresolved **@908_f43**

Anonymous Calc 2 7 months ago

Doesn't calloc takes in 2 arguments?

```
calloc(size_t nitems, size_t size)
```

Can we put just one argument like sizeof(GenericLink)?

```
GenericLink* link = (GenericLink*) calloc(sizeof(GenericLink)); link->value.val1 = c; return link;
```

helpful! | 0

Peyrin Kao 7 months ago

That's probably a typo. You would need to add an additional argument of 1 for this code to compile.

good comment | 1

Anonymous Calc 2 7 months ago

Or can I just do malloc?

helpful! | 0

Adelson Chua 7 months ago

Depends on the requirements of the question.

Remember, calloc initializes the allocated memory, malloc does not.

good comment | 1

Peyrin Kao 7 months ago

I think you need to use calloc in this case to explicitly set all the fields in the union to 0. Would need to double-check the question to be sure, though.

good comment | 1

Reply to this followup discussion

Start a new followup discussion

Compose a new followup discussion