

# [Midterm] Past Exams - 2021 #489



**Jero Wang** STAFF  
Last year in **Exam - Midterm**

1,422  
VIEWS



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

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

## Semester-Exam-Question Number

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



**Anonymous Otter** 12mth #489dcb Unresolved

### FA21-Final-Q2.4

I am really confused on how to approach this problem with FP. Could a staff walk through it please?

♡ 1 ...



**Anonymous Otter** 12mth #489dca Unresolved

### SP21-MT-Q7b:

Could one of the staff please give me a walk through on how to approach this problem. I couldn't even understand the description of what was being asked. Furthermore, the solutions to this problem make no semantical sense to me. Please help!!

♡ ...



**Anonymous Barracuda** 1y #489dbd ✓ Resolved

### FA21-MT-Q3.1

I don't understand why **Set the contents of the extra union in ret to be all zeros** means that **ret->extra.d = 0** cuz it just set the value of double d. What about a, b, and c?

♡ ...



**Erik Yang** STAFF 1y #489dbe

a union is different than a struct, in that the size of the union is equal to the largest field (in bytes). Since a `double` is the largest field in that struct, you only need to set that to be zero.

♡ ...



**Anonymous Hawk** 1y #489dae ✓ Resolved

FA21-Final-Q6.3, can anyone explain what it means that "Y never distinguishes between 1 and 0" thus "Y never affects the result of our output". I think Y does affect the output. When Y is 0, the

answer is undetermined and when Y is 1 the answer is determined. Also how to deal with this problem regarding the X?

**Solution:** Staff solution:  $W \wedge Z$

Other answers may be possible (Notably, this question can yield a score well below par).

In this case, we note that there's never a time when  $Y$  distinguishes between 1 and 0; as such,  $Y$  never affects the result of our output, and we can look at the reduced truth table containing only  $W$  and  $Z$ .

♡ ...



Anonymous Giraffe 1y #489dad

✓ Resolved

SP21-Final-Q6a

How is the sizeof(struct foo) 12? and How does swapping b and b increase the size of the structure?

## 6. C Structures

### (a) (6.0 points) The Structure of Structures

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

Consider the following structure definition and code:

```
struct foo {
    char a;
    uint16_t b;
    char *c;
    struct foo *d;
}
```

What is `sizeof(struct foo)` (Answer as an integer, with no units)?

12

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

True

- True  
 False

♡ ...



Sam Xu STAFF 1y #489daf

`char a` takes one byte, and `uint16_t b` takes two bytes. However, `uint16_t` must have byte-aligned to 2 bytes, so after put one-byte `char a`, the computer will put one byte as padding. Then, `char a + uint16_t b` takes four bytes. Then both pointers `c` and `d` take 8 bytes, the total is 12 bytes.

♡ 1 ...



Anonymous Hawk 1y #489cfe

✓ Resolved

SP21-MT-Q2C(ii), I understand what the solution says, but I also remember that the linker is the step where absolute addressing is assured, while the assembler still uses relative addressing. Then how can the assembler produce an object file that specifies the exact location of the components in the object file?

ii. (0.5 pt) Object files are distinctly segmented into components and will tell the linker exactly how many bytes and where each component is located.

- True  
 False

True; the object file header tells the linker the size and location of the following components of the object file: text segment, data segment, relocation table, symbol table, and debugging information.

♡ ...

↳ S Sam Xu STAFF 1y #489dbf

The assembler will not tell the absolute address of the functions. Instead, it uses symbolic labels and relative offsets to represent memory addresses in a way that's independent of the final memory layout. The Linker uses these information and figure out the absolute address.

♡ ...

Anonymous Hawk 1y #489cee ✓ Resolved

SP21-Final-Q1C, what does "generate parse trees" mean and why the compile? I didn't remember it was discussed in lecture...

C. Generates parse trees.

- Compiler  
 Assembler  
 Linker  
 Loader

♡ ...

↳ C Catherine Van Keuren STAFF 1y #489dbc

Parse trees were not discussed in lecture, thus out of scope for this semester.

♡ 1 ...

Anonymous Armadillo 1y #489ced ✓ Resolved

Sp 21 Final Q6.2

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

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

Don't really understand why we are left shifting by 4 and loading 4(t0), probably have something to do with size of the struct which I think is 8 since it contains two pointers.

♡ ...

↳ Anonymous Armadillo 1y #489cff

Update: looked at walkthrough video and the solution in the video is literally different from the answer key. So uh idk whats going on anymore.



Anonymous Mole 1y #489cdf

✓ Resolved

Fall 2021 MT 1, Q2.1

**Q2** *Now, Where Did I Put Those Strings?*

(10 points)

Consider the following code:

```

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
}

```

The char \*strcpy(char \*dest, char \*src) copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. The strings may not overlap, and the destination string dest must be large enough to receive the copy.

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

- code
  static
  heap
  stack

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

- code
  static
  heap
  stack

I don't understand why the answer to 2.1 is static? Both str1 and str2 were declared inside the foo function; therefore, shouldn't both be stored in the stack?



E Eddy Byun STAFF 1y #489cea  
#489aed



Anonymous Mole 1y #489ceb

Followup:

Q2.8 (1 point) Printing the string.

- printf("%s\n", str1);
  printf("%s\n", str3);
- printf("%s\n", str2);
  None of the above

**Solution:** printf("%s\n", str1); and printf("%s\n", str2); and printf("%s\n", str3);

Recall that printf("%s\n", str1); dereferences the str1 pointer and prints out the string stored at that address (up until the null terminator).

Since str1, str2, and str3 all point to a valid null-terminated string, it is safe to call printf on all three of them.

**Grading:** Each answer choice was graded independently, just like in the previous subparts.

How can we print str2? If str2 is allocated in stack memory, I though we couldn't reference it outside of the foo function?



Justin Yokota STAFF 1y #489cef

We can't reference it after returning from foo; if you think about RISC-V, we can see that the sp moved back up, and any future function call could overwrite that stack data. In this case, we're making the printf within foo still, so the stack data is still valid.



Anonymous Cobra 1y #489cde

✓ Resolved

### SP21-MT-Q4

If we have a register `a0` that has a string, how do we access its bytes? If `a0` was a pointer to the string, we could do manipulations like `0(a0)` and so forth, but if `a0` itself has the string, what do we do? The solution here for example has the line `1bu t0 0(a0)`, but if `a0` has the string itself and not the pointer to the string, how does that work?



Anonymous Lapwing 1y #489cdb

✓ Resolved

### FA21-MT-Q4.3

I believe floating point numbers have both +0 and -0. Wouldn't this make 509 the correct answer? (assuming that +/- 0 are 2 different numbers from question 4.2)

$2^9 - 2 (+/- \text{infinity}) - 1$  (negative zero, since positive 0 is included in unsigned int) = 509



Justin Yokota STAFF 1y #489cfa

Where did you get the  $2^9$  from?



Anonymous Lapwing 1y #489cfb

$2^8$  (8 bits of significand) \* 2 (1 bit for sign) =  $2^9$



Justin Yokota STAFF 1y #489cfc

↩ Replying to Anonymous Lapwing

Why does that tell you the number of values that the 16-bit integer can represent, but the floating point number can't?



Anonymous Lapwing 1y #489cfd

↩ Replying to Justin Yokota

$2^9$  is the number of values that the floating point allocates for NaNs and +/- infinity. The answer in the solutions is  $2^9 - 2$  (for +/- infinity). However, floating point numbers also allocate one value for -0, which I think would make the answer 509.




Justin Yokota STAFF 1y #489daa

↩ Replying to Anonymous Lapwing

Why would that subtract? Is the binary for the value representing -0 not a valid number as a 16-bit integer?



 **Anonymous Lapwing** 1y #489dab  
↩ Replying to Justin Yokota


the problem before (which this problem builds on) states that we can treat  $-/+ 0$  as distinct values. So I thought that unsigned 16-bit integers cannot represent  $-0$ , so we would have to subtract 1 from the total number of values that unsigned ints can represent but floats cannot.

♡ ...

**J** **Justin Yokota** STAFF 1y #489dac  
↩ Replying to Anonymous Lapwing

Yeah, but there are a lot of numbers that unsigned 16-bit integers can't represent, that our floating point can. 1.5, for instance. If we're subtracting  $-0$ , why aren't we subtracting 1.5? Also there are a lot of numbers the unsigned integer can represent that the float can't represent, like 65535. Why aren't we adding that?

♡ ...

 **Anonymous Mole** 1y #489cda ✓ Resolved  
Spring 2021:

## 7. C Programming

(a) Consider the following structure definition. Assume we are using a 32-bit machine.

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

And the following C code


```
void bar(struct foo *f){
    int i;
    ....
    for(i = 0; i < 5, ++i){
        baz(f[i].b);
    }
    ....
}
```


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
```

why are we doing `sll a0 s6 3`? Shouldn't it be `slli a0 s6 1`? Shouldn't we also do `lw a0 0(a0)` instead of `lw a0 4(a0)`?

♡ ...

 **S** **Su-Ann Ho** STAFF 1y #489cdd  
[#489bcb](#)  
♡ ...

 **Anonymous Chinchilla** 1y #489cce ✓ Resolved  
**FA21-MT-Q5.3**

what's the common procedure or intuition for PC-relative addresses? does it always start from the address of the function? what exactly is the PC?

♡ ...

S **Su-Ann Ho** STAFF 1y #489cdc

The intuition for using the PC and PC-relative addressing is that your code—the actual RISC-V instructions themselves—are stored in memory as 32-bit (4-byte) information. As mentioned in lecture slides, "the PC (program counter) is a register internal to the processor that holds the byte address of next instruction to be executed". This means that the Program Counter is a register responsible for pointing to the **memory address of the next instruction**. If we just want the instruction on the next line of our code to execute (meaning we are not jumping or branching), the Program Counter simply increments by 4 bytes, since instructions are stored consecutively.

When you do PC-relative addressing, you are computing the distance between the instruction you are currently on, and the instruction you want to move to next, in terms of bytes in memory. For example, to jump to a label that is 4 lines of instruction down instead of the next instruction, you would want to add 16-bytes to the current address instead of 4-bytes. Therefore, PC would contain (PC + 16) instead of (PC + 4) when we reach the jump instruction.

♡ ...

Anonymous Chinchilla 1y #489cec

that makes a lot of sense, thank you! whenever we have a j or b branch with a label, do we always have to follow the procedure with offsetting like in this question? is the general procedure just finding the function call line's offset to the label?

♡ ...

Anonymous Hawk 1y #489caf ✓ Resolved

It seems that SP21-MT doesn't have a Q10A, [Spring 2021 Midterm Q10A](#)

♡ ...

J **Jero Wang** STAFF 1y #489cbc

It should be the final, sorry about that! Should be fixed on the website now.

♡ ...

Anonymous Hawk 1y #489cbd

Same for Q10B [Spring 2021 Midterm Q10B](#)

♡ ...

J **Jero Wang** STAFF 1y #489cbf

↩ Replying to Anonymous Hawk

Fixed as well, thanks for the catch!

♡ 1 ...

Anonymous Pony 1y #489bfc ✓ Resolved

**FA21 MT Q3** - few clarifications/questions about this one:

- why can't i do `*c == NULL` in the 1st blank?
- where do we get "extra.d" from in the 5th blank? the question asked us to "set the contents of the extra union in ret to be all zeros." i thought we have to somehow set `char a[5]`, `uint16_t b`, `int c`, and `double d` to be all zeros...

- for the 7th blank, is it valid to do: "(\*f)(c->car)" instead of "f(c->car)"? will we ever need to dereference a given function to call it?

**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;
}
```

- for **Q3.2** - when did we learn about what UNIONS are? the solution key said that the sizeof(union) is always size of largest element. where'd this come from?

♡ ...



**Jero Wang** STAFF 1y #489cae

- If `c` is `NULL`, then you're dereferencing a null pointer and would get a segfault.
- Members in unions share memory, only one of the members "exist" at any given time. Since `d` is the largest member, by setting it to 0, you set all of the bits in the union to 0.
- That's fine. C will automatically dereference for you, though it won't hurt to dereference manually.
- Unions were covered in homework 2.8 (though you probably needed to Google/look up some stuff to learn more about it).

♡ ...



**Anonymous Pony** 1y #489beb

✓ Resolved

**FA21 MT Q5.1** - why can't you do "mv t0 Password" on line 6 instead of using "la", since line 7 uses "mv"?



**Solution:**

```
1: verifypassword:
2:     addi sp, sp, -24 # Make space for a 20-byte buffer
3:     sw ra 20(sp)
4:     mv a0, sp
5:     jal ra Get20chars
6:     la t0 Password
7:     mv t1 sp
8: Loop:
9:     lb t2 0(t0)
10:    lb t3 0(t1)
11:    bne t2, t3, Fail
12:    beq t2, x0, Pass
13:    addi t0 t0 1
14:    addi t1 t1 1
15:    j Loop
16: Pass:
17:    li a0, 1
18:    j End
19: Fail:
20:    li a0, 0
21: End:
22:    lw ra, 20(sp)
23:    addi sp, sp, 24
24:    jr ra
```



Andrew Liu STAFF 1y #489bfb

#489bba



Anonymous Mole 1y #489bdd

✓ Resolved

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

0b1111111

I am unsure how they got 64 with the biased notation.... I though we couldn't represent 64 in this representation?



Kushal Kodnad 1y #489bef **ENDORSED**

there are 7 bits, so the max value that can be represented (before bias) is  $2^7-1 = 127$ . we want:  $\text{exp} + \text{bias} = 64$

$\Rightarrow \text{exp} = 64 - \text{bias} = 64 - (-63) = 127$

127 in binary is 0b1111111 (7 bits)



Anonymous Pony 1y #489bdb

✓ Resolved

SP21 Q7A

1. how do we get the size of (struct foo)?

2. this was my understanding, but i think i'm wrong. "char a" is a character (1 byte), but "char \*b" is a pointer to a string sequence, which has 4 bytes? honestly not really sure how to interpret the "struct foo" definition here.

3. why is sizeof(f) equal to 4? isn't f a pointer to a "struct foo" object?

## 7. C Programming

(a) Consider the following structure definition. Assume we are using a 32-bit machine.

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

And the following C code

```
void bar(struct foo *f){
    int i;
    ....
    for(i = 0; i < 5, ++i){
        baz(f[i].b);
    }
    ....
}
```

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

8

ii. (2.0 pt) What is `sizeof(f)`?

4

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
```

♡ ...

J Jero Wang STAFF 1y #489cac

In a 32-bit system, `char` s are 1 byte, and `char *` s (or all pointers) are 4 bytes. To fulfill the alignment requirements, each member must have an offset (within the struct) that's a multiple of its size. Therefore, `char a` is at offset 0, and `char *b` is at offset 4 (offsets 1-3 are padding). `char *b` would occupy bytes 4 through 7, so the struct takes up 8 bytes.

Your interpretation is correct! I think you might just be missing the padding.

Yes, `f` is a pointer to `struct foo`. However, pointers are just really memory addresses, so it doesn't matter what type it points to, it always has the same size. For example, `char *` and `int64_t *` would be both 4 bytes in a 32-bit system.

♡ 1 ...

Anonymous Pony 1y #489cad

gotcha! those all make sense, thanks!!

♡ ...



Anonymous Chinchilla 1y #489bce

✓ Resolved

### FA21-MT-Q2

can someone clarify what goes in static data? i thought \*str was a local variable because it's defined in a function. i understand that \*str is an address since it's a pointer, but i'm not sure what it means for it to be in static data or what's defined as static data



Jero Wang STAFF 1y #489cab

String literals and variables defined outside of functions are in static memory.

\*str 's value is wherever str points to, which could be any segment of memory. If str points to a string literal, then \*str 's value would be stored in static.



Cecilia Aiko 1y #489bcb

✓ Resolved

### SP21-Midterm-Q7a

Can anyone explain why we need to use the first line (sll) and why we use lw a0 4(a0) instead of 0(a0) ? Thank you!

- 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
```



Anonymous Weasel 1y #489bcf ENDORSED

I'm not sure if I'm right but from what I understand

size of foo = 8

f is a pointer to foo

so at f , we have foo1

at f+8, we have foo2

we are looking for foo[i]

which is  $f + i * 8$

to multiply by 8, we can sll by 3 (since  $2^3 = 8$ )

adding this to s5, we get f[i]

now we are at f[i] and we want f[i].b

char a occupies first 4 bytes of foo, next 4 are of char\*b

since we want b, we go to 4(a0)

♡ 1 ...



**Anonymous Gorilla** 1y #489dbb

Isn't f a pointer to one foo ? Since sizeof(foo) = 8 , when adding 8 wouldn't we get an error ?

♡ ...



**Kushal Kodnad** 1y #489bda **ENDORSED**

for the first part of your question, i think it's because from part (i), we know that sizeof struct foo is 8. so since i is an integer representing the index of the foo that we're looking for, doing "sll a0 s6 3" basically multiplies s6 by  $2^3 = 8$ , so it's basically a byte shift.

then for the 2nd part, i think 0(a0) gives you "char a" while 4(a0) gives you "char \*b" from the struct definition.

hope this helps! if i missed anything, please lmk!

♡ 1 ...



**Anonymous Mallard** 1y #489bbe **Resolved**

Q2.5 (1 point) Returning the string.

`return str1;`

`return str3;`

`return str2;`

None of the above

**Solution:** `return str3;`

`str1` is a pointer to static memory, which doesn't change throughout program execution, so `return str1;` is safe.

`str2` is a pointer to the stack. When the function returns, the string on the stack is erased, which causes `return str2;` to have undefined behavior.

`str3` is a pointer to the heap. Heap memory stays allocated until the programmer calls `free`. Since this function never calls `free`, the string on the heap will stay allocated, so `return str3;` is safe.

**Grading:** Each answer choice was graded independently. 1/3 of a point for correctly selecting `str1`, 1/3 of a point for correctly *not* selecting `str2`, and 1/3 of a point for correctly selecting `str3`. Selecting "None of the above" is worth 1/3 points (for correctly not selecting `str2`).

In FA21-MT-Q2.5, Does returning a "string" literally mean any string? I didn't choose these because I thought `return str1` or `return str3` would return the address, not "Hello World" string itself. So are `return str1` and `return str3` both answers because the address itself is also a "string"?

♡ ...



**Jero Wang** STAFF 1y #489caa

Strings are really just `char *s`, which...are pretty much just addresses if you peel away all of the abstractions. I think the prompt here might be a bit confusing, but it's trying to ask which of those lines would cause defined behavior, rather than what actually returns a string (though both of the correct answers do return strings).

♡ ...



**Anonymous Weasel** 1y #489bbc **Resolved**

SP22-MT-Q2

what are interpreters?

2. (5.0 points) CALL

(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.
- The code produced is always more efficient and higher performance than that produced by interpreters.
- There is only one compiler per language.
- Compilers are always more difficult to write than interpreters.
- The easiest step of CALL is compilation; the harder parts are assembling, linking, and loading.

TBD

2B . What are translators?

♡ ...



Jero Wang STAFF 1y #489bff

Please post in the 2022 thread!

♡ ...



Anonymous Manatee 1y #489bbb

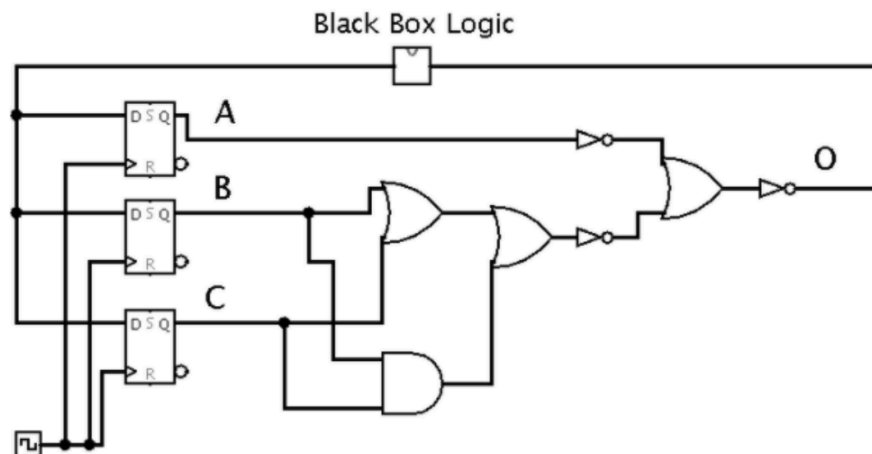
✓ Resolved

SP21-MT-Q3

3. SDS

For the following question, do NOT include units in your answer!

In the following circuit, the registers have a clk-to-q delay of 6ns and setup times of 5ns. NOT gates have a delay of 3ns, AND and OR gates have a delay of 7ns, and the "Black Box" logic component has a delay of 9ns.



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 =  $6 + 2 \cdot 3 + 7 + 9 = 28\text{ns}$


what is O and why is the shortest path from A to O? isn't the shortest combinational path basically just A back to A? the answer does make sense, I'm just confused on what O is specifically.

♡ ...

 **Anonymous Pony** 1y #489bdc  
adding onto this question, why does the path from A to O go through the Black Box Logic?  
♡ ...

 **Anonymous Manatee** 1y #489bdf  
my best guess was that the shortest path is from A --> A and to get there you would go past O and through the Black Box Logic then back to A. i'm just confused as to what O is  
♡ ...

 **Jero Wang** STAFF 1y #489bfe  
O refers to the wire connecting the output of the NOT gate (that it's somewhat next to) to the input of the black box. The shortest path always starts at a register and ends at a register. In this case, it starts with the clk-to-q of the top register, then goes through the NOT/OR/NOT gates, then through the black box, then back to any of the three registers.  
♡ 1 ...


 **Ali Khani** 1y #489bae ✓ Resolved  
SP21-Final-Q6A: For part (i), in the `struct foo` we have the following contents:


```
char a;  
uint16_t b;  
char *c;  
struct foo *d;
```

a should be 1 byte since it's a char,  
b is 2 unsigned bytes,  
and \*c and \*d should be 4 bytes each since they're pointers.

I don't understand how the answer isn't a total of 11 bytes for the `sizeof(struct foo)` — in particular, I don't get where the last 1 byte comes in from. What's the right approach to this question?

♡ ...

 **Jero Wang** STAFF 1y #489bfd  
Alignment causes the extra byte to be added. In general, when considering field alignment, the main rule is that each member's offset must be a multiple of its size. In this case, `char a` has a size of 1, and its offset is 0, so that works. Then, `uint16_t b` has size 2, so it must be at an offset that's a multiple of 2. Therefore, we need to insert a byte of padding (at offset 1), and `uint16_t b` would start at offset 2.  
♡ 1 ...

 **Anonymous Barracuda** 1y #489bad ✓ Resolved  
FA21-Final-Q6.2: im getting to  $\sim y + (y * \sim z)$  but the sol has  $\sim (y \& z)$

can someone explain how I would get to that solution from the expression I have? also is my answer a valid answer?

♡ ...

 **Jero Wang** STAFF 1y #489bfa  
You can simplify this as follows:

$$\begin{aligned} & \sim y + (y * \sim z) \\ & = (\sim y + y) * (\sim y | \sim z) \text{ (distributive property)} \end{aligned}$$

$$\begin{aligned}
 &= 1 * (\sim y \mid \sim z) && \text{(complements)} \\
 &= \sim y \mid \sim z && \text{(identity)} \\
 &= \sim(y * z) && \text{(DeMorgan's)}
 \end{aligned}$$

Your solution would receive partial credit, as it uses 4 operators where the staff solution uses 2 operators.

♡ ...



Anonymous Crow 1y #489bac

✓ Resolved

Exam generated for cs61c@berkeley.edu

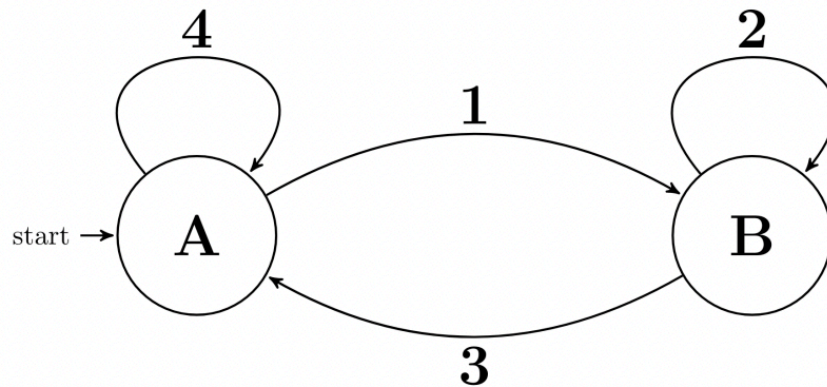
12

### 3. FSM

We want to construct a finite-state machine that determines if adding together two binary unsigned numbers causes an overflow. The machine consumes two bit strings of equal length, starting from their least significant bits. After consuming each pair of bits from the two inputs, the machine outputs 1 if addition of the two strings (as seen so far) would cause an overflow, or a 0 otherwise.

For example, if the machine consumes the two bit strings 0111 and 0100, the sequence of output values will be 0, 0, 1, 0.

A diagram for this finite-state machine is shown below. We have given the states generic placeholder names. Transition labels use the notation  $x,y/o$ , where  $x$  is a bit read from the first input string,  $y$  is a bit read from the second input string, and  $o$  is the output. To simplify, you are allowed to assign multiple labels to the same arrow in the diagram – each label corresponds to a different transition with the same start and end states.



State Machine

- (a) i. (1.0 pt) Which of the following labels (each corresponding to one transition) should be assigned to **Arrow 1** in the diagram?
- 1,1/0
  - 0,0/0
  - 0,1/0
  - 1,0/0
  - 1,0/1
  - 0,1/1
  - 0,0/1
  - 1,1/1

can someone explain how to approach this question?

♡ ...



Jero Wang STAFF 1y #489bee

First, consider when the sum of two strings would overflow. It only overflows if both inputs are 1 OR if one of the inputs is 1 and the "carry" was previously 1. We need to save one value



between iterations, which is if we're carrying a value or not. We can use the current state to represent this: let A be if there is no carry from previous calculations, and B if there is a carry (since we start at A, and we start off without a carry).

Then, to look at each of the transitions:

- Arrow 1 goes from A to B, which is going from no carry to carry. When would this happen? It should only happen if both inputs are 1 (e.g.  $1 + 1 = 0$  with a carry of 1). In this case, since we have a carry of 1, it would overflow (say we have 1-bit binary numbers,  $0b1 + 0b1 = 0b0$ , overflows)
  - We can extrapolate from this that whenever there is a carry after the current computation, the output of the FSM should be 1, since there is an overflow.
- Arrow 2 is going from B to B, which is going from carry to carry. The cases that would result in this would be
  - 1,1 ( $1 + 1 + \text{carried } 1 = 1$  with carry of 1)
  - 1,0 ( $1 + 0 + \text{carried } 1 = 0$  with carry of 1)
  - 0,1 ( $0 + 1 + \text{carried } 1 = 0$  with carry of 1)
  - All three cases have a carry of 1, therefore, the output of the FSM would be all 1
- Arrow 3 goes from a carry to no carry, which means that the only case is both inputs are 0 (if either input is 1, you have to add it to the previously carried value of 1, which gives you another value to carry). As you don't have a carry after the current computation, you aren't overflowing, and the output is 0.
- Finally, Arrow 4 would have the rest of the possible transitions, going from no carry to no carry, and output would be 0 since you're not carrying after the current computation.

♡ ...



Anonymous Pony 1y #489afd

✓ Resolved

Sp21 MT Q5 - is this in scope?

♡ ...



Isita Talukdar 1y #489bab

👑 ENDORSED

I don't think so, Single\_Cycle-Datapath is not in scope and neither is Pipelining

♡ 1 ...



Anonymous Kangaroo 1y #489afc

✓ Resolved

FA21-MIDTERM-Q5:

I had a different solution that I wanted to run by staff.

In my solution, everything except for lines 12, and 15 are different.

In line 12, I instead do: **li t4 '\0'**

In line 15, I instead do: **bne t3 t4 Loop**

Would this be an acceptable solution?

My reasoning behind this solution is the following:

There can only be 3 cases: either the input is same size as password, the input is longer than password or the input is shorter than password.

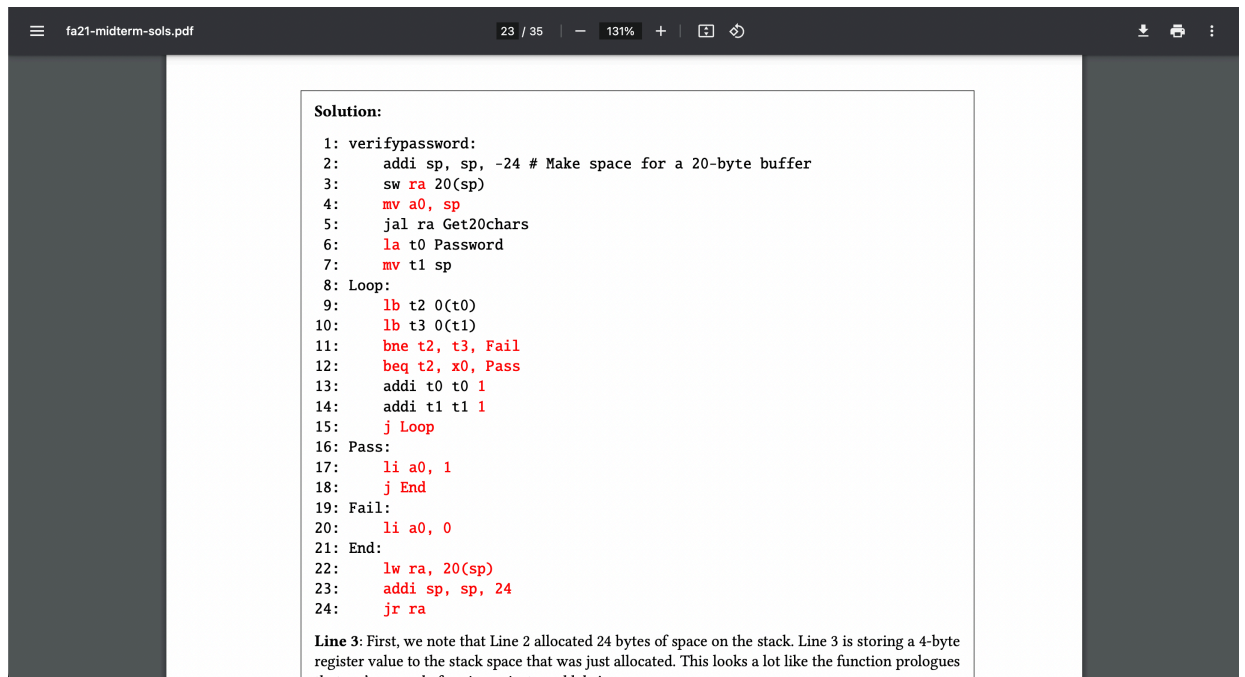


1) If same size, once we're at the last iteration, and we compare t3 to t4, we'll realize that they are equal (both are null terminator), so we will continue to the pass Branch (this is correct as we have correctly finished checking the input against the password).

2) If input is longer than password, well it doesn't matter because we'll reach the case where t2 = null terminator, t3 = some character AND hence line 11 will become true and we'll jump to the Fail branch which is correct.

3) If input is shorter than password, well we'll reach the case where t2 = some character, t3 = null terminator AND hence line 11 will also become true and we'll jump to the Fail branch which is correct.

In all cases, this logic still holds, would this be an accepted solution?



The screenshot shows a PDF document titled 'fa21-midterm-sols.pdf' with page number 23 / 35 and a zoom level of 131%. The document contains the following assembly code:

```
Solution:
1: verifypassword:
2:   addi sp, sp, -24 # Make space for a 20-byte buffer
3:   sw ra 20(sp)
4:   mv a0, sp
5:   jal ra Get20chars
6:   la t0 Password
7:   mv t1 sp
8: Loop:
9:   lb t2 0(t0)
10:  lb t3 0(t1)
11:  bne t2, t3, Fail
12:  beq t2, x0, Pass
13:  addi t0 t0 1
14:  addi t1 t1 1
15:  j Loop
16: Pass:
17:  li a0, 1
18:  j End
19: Fail:
20:  li a0, 0
21: End:
22:  lw ra, 20(sp)
23:  addi sp, sp, 24
24:  jr ra
```

Line 3: First, we note that Line 2 allocated 24 bytes of space on the stack. Line 3 is storing a 4-byte register value to the stack space that was just allocated. This looks a lot like the function prologues that we've seen before to reserve and label

♡ ...

J Jero Wang STAFF 1y #489bed

Your logic seems correct, I think this would be a valid alternate solution.

♡ ...

Anonymous Grasshopper 1y #489afb

✓ Resolved

Fa21-Midterm-Q5.1/5.3

For 5.1, How can we use labels to store strings? An example would be helpful.

For 5.3, how do we know that jal is an I type? The reference card says it's a J type, which is confusing

♡ ...

J Jero Wang STAFF 1y #489bec

5.1: You can use assembler directives (such as this from [project 2](#)).

5.3: Yes, that's a typo, it should be J type, sorry.

♡ ...

Anonymous Grasshopper 1y #489aee

✓ Resolved

Fa21-Midterm-Q3.1

Shouldn't we be writing ret->car instead of just car?

♡ 2 ...

 **Anonymous Stork** 1y #489aef

I had this same question, how can you access car if it is a variable of the struct?

♡ ...

 **E Eddy Byun** STAFF 1y #489afa

Yea, sorry for the confusion! It's a typo #489ce

♡ 2 ...

 **Anonymous Stork** 1y #489aed ✓ Resolved

FA21-Midterm-Q2.1: Why is \*str1 located in static memory? I thought since it is a variable defined inside a function it is a local variable and all local variables are stored on the stack. How do I know when variables inside functions are static memory?

## Q2 *Now, Where Did I Put Those Strings?*

(10 points)

Consider the following code:

```
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
}
```

The char \*strcpy(char \*dest, char \*src) copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. The strings may not overlap, and the destination string dest must be large enough to receive the copy.

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


code

static

heap

stack

♡ 4 ...

 **Isita Talukdar** 1y #489aff 🔗 ENDORSED

str1 is a preallocated variable since it's a char\* string literal. It's immutable, unlike str2[] which is a mutable char array stored in the stack

♡ ...

 **Anonymous Mallard** 1y #489bbd

I thought char \*string and char string[] were nearly the same thing, as shown in Lec 4 Slide 28. But why is the first stored in static and the second stored in a stack?

♡ ...

 **Isita Talukdar** 1y #489bea

↩ Replying to Anonymous Mallard

they're similar but again, different.

char \*string = "potato" is an immutable string literal, just like in Python and other languages, you cannot change individual letters in a string

but char string[] makes it a mutable array of characters that you can change

because of that, it makes sense that the first is stored in static read only segment, as we cannot change it

♡ ...

 **Anonymous Gorilla** 1y #489ccf

↩ Replying to Isita Talukdar

would `char *string` be a pointer to an immutable string literal or the string literal itself or first char of the string literal ?

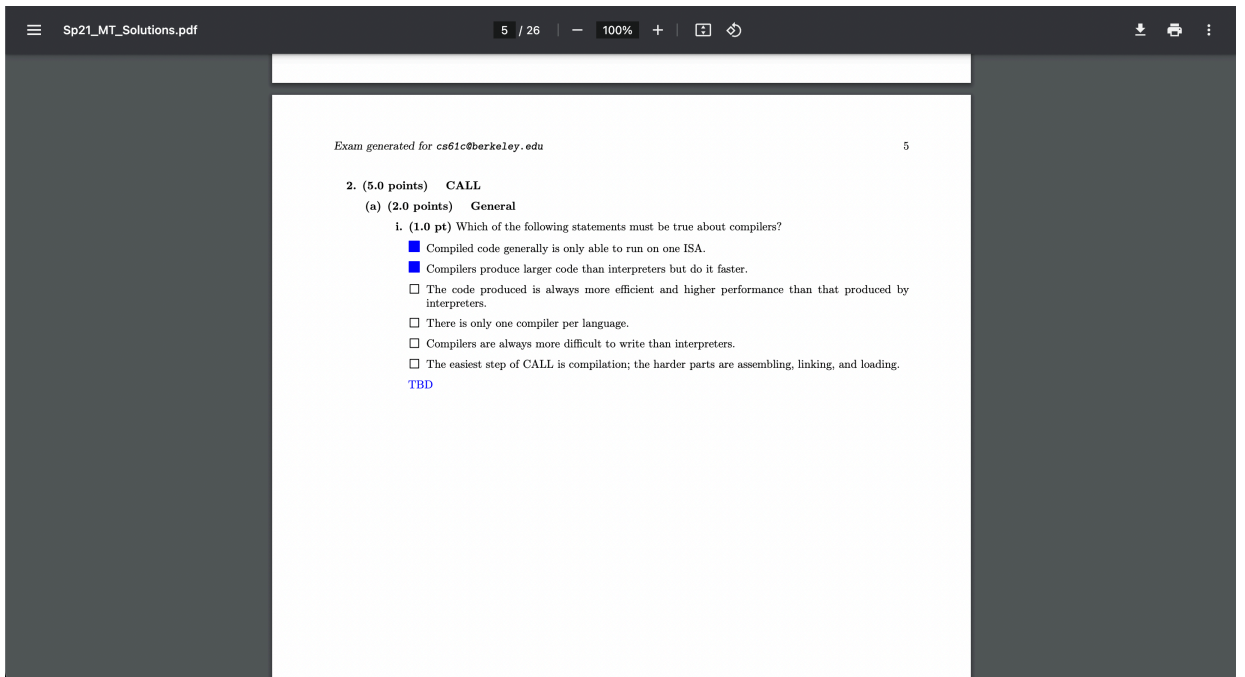
♡ ...

 **Anonymous Kangaroo** 1y #489aec



✓ Resolved

SP22-MT-Q2A

Can anyone explain why this is the solution and why the other options are wrong?



♡ 2 ...

 **Isita Talukdar** 1y #489baa  ENDORSED

for the first uncolored option, technically in the slides it says "almost always" not always when it talks about that. I'm not sure why, but maybe it's because technically you can interpret in RISC-V and I imagine that's more/equally efficient than compiling C++ code to RISC-V?


for the second, that's untrue because there are lots of different lower level languages like x86(?) so there's certainly more than one compressor

I'm not sure exactly why for the third one, it just seems a little too much of an absolute statement, I hope a TA answers this one XD

The last one is from the slides, Compiler is the hardest and most complex out of the steps in CALL. I think it's because going from a high level language to a very specific and limited assembly language takes a lot of thought (you have to replace simple instructions with so many lines). Eddy explained this also in thread [#489add](#)

im no TA but at least that's what I thought

♡ ...

 **Andrew Liu** STAFF 1y #489baf

Great answer from Isita! Let me fill in some more details.

C) Always is a super strong term, and efficient and higher performance aren't always guaranteed because of the languages. Some high performance interpreted languages take advantage of things like lazy computation and other optimizations to be quite efficient.

D) Isita is spot on! Even for C, there are lots of compilers, like `clang`, `gcc`, and many many more: [https://en.wikipedia.org/wiki/List\\_of\\_compilers#C\\_compilers](https://en.wikipedia.org/wiki/List_of_compilers#C_compilers)

E) Once again, always is a super strong term here. You could imagine a super simple compiler for a language that only has say, 1 instruction (it would be a pretty useless or obtuse language, but as a thought experiment it's a good model).

F) Just as Isita said.

♡ 1 ...

 **Anonymous Red deer** 1y #489adf ✓ Resolved

FA-Midterm-Q3, Are unions in C in scope?

♡ ...

 **Eddy Byun** STAFF 1y #489aea

Yes, unions are in scope

♡ ...

 **Anonymous Giraffe** 1y #489acf ✓ Resolved

**FA19-Midterm-Q4b**

Why do we move the PC forward by 8 instructions? I thought we only moved it till we reached the line with the label so that would be 7 instructions. And why do we multiply by 4? I saw earlier calculations where immediate is just the value of the offset, not the number of bytes we move forward by.

Now assume all blanks above contain a single instruction (no more, no less).

b) The address of `reverse` is `0x12345678`.

What is the hex value for the machine code of `beq x0, t0, returnnode`? **`0x02500063`**

c) The user adds a library and this time the address of `reverse` is `0x76543210`.

What is the hex value for the machine code of `beq x0, t0, returnnode`? **`0x02500063`**

**Part B**

`beq x0, t0, returnnode` moves the PC (program counter) forward by 8 instructions, each of which is 1 word or 4 bytes. This means the immediate =  $8 * 4 = 32$ .

In binary, we represent 32 as `0b100000`. However, since we know branch/jump immediates are always even numbers, we don't store bit 0.

♡ ...

 **Eddy Byun** STAFF 1y #489ada

Can you repost this question in this megathread: [#490?](#)

♡ ...

 **Anonymous Giraffe** 1y #489adb

yes, sorry about that.



Anonymous Grasshopper 1y #489acd

✓ Resolved

### FA21-Midterm-Q1

Can someone explain why the compiler is more complex than the assembler. Doesn't the assembler stage involve more steps such as making the relocation table and symbol table, whereas the compiler simply needs to convert C code into RISC-V?

Q1.2 (1.25 points) True or False: The assembler is the step with the highest computational complexity among CALL.

True

False

**Solution:** False. The compiler is more complex than the assembler.

**Grading:** 1.25 points for False.



E Eddy Byun STAFF 1y #489add

The compiler has to do a lot of things. One thing it does is optimize your code. For example, if there is some piece of code that is not needed, the compiler might not compile that section. The compiler may also reorder your code. The compiler also does error checking. From lab or project, you may have seen an error when you assign something of one type to something of another type, and this job is up to the compiler. All of these jobs are up to the compiler so converting C code to assembly is takes up a lot of computation and complexity.

♥ 1 ...



Anonymous Grasshopper 1y #489aeb

Thank you, this makes a lot of sense!

♥ ...



Anonymous Spider 1y #489bde

Would we be expected to know this for the exam? Looking at the slides, the Assembler is described with much more detail and it seems kind of arbitrary to compare it with another part of the CALL process in terms of computational complexity given what we were taught.

♥ ...



Jero Wang STAFF 1y #489ccb

↩ Replying to Anonymous Spider

[#616](#)

We'll reply in this post once we have an official answer.

♥ ...



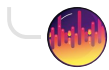
Theo Putterman 1y #489acb

✓ Resolved

### FA21-MT-Q5.1 Line 6

Why is mv not a valid option here? It says that password is a pointer to a string in static memory, so wouldn't password be a 4-byte integer (assuming 32 bit system), that we can just move into t0?

♥ 1 ...



Andrew Liu STAFF 1y #489bba

to the following labels defined externally

password is a label, not a pointer directly.



Anonymous Grasshopper 1y #489aca

✓ Resolved

SP21-MT-Q7-a.iii

For the coding portion, is it wrong to use jalr or jal ra instead of just jal?



E Eddy Byun STAFF 1y #489adc

jal baz and jal ra baz are the same. You can't use jalr in this case because jalr is going to take the value inside of rs1, add the offset, and then jump to that instruction. We don't know what instruction baz is at.



Anonymous Crow 1y #489bbf

for this question why do we do sll a0 s6 3?



J Jero Wang STAFF 1y #489cca

↩ Replying to Anonymous Crow

It should be slli, but the purpose of that instruction is to get the offset in bytes from the index (we multiply by 8/shift left by 3 because each struct is 8 bytes).



Anonymous Crow 1y #489ccd

↩ Replying to Jero Wang

but s6 contains i and i is an int. so shouldn't we do 4 bytes?





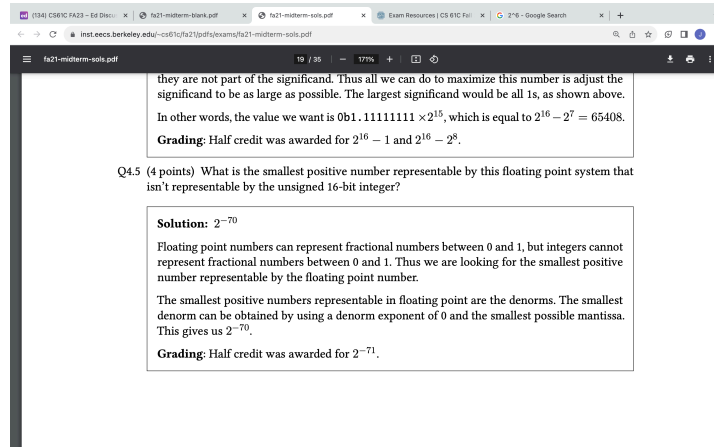
Anonymous Bison 1y #489aae

✓ Resolved

FA21-MT-Q4.5

I understand that we are looking for a denorm (exponent of zero, significand of nonzero). Why isn't the answer

$$2^{-8} \cdot 2^{-63} = 2^{-71}$$



♡ 2 ...



E Eddy Byun STAFF 1y #489abd

Recall that for denorm this is the following equation:

$$\pm 0.\text{Mantissa}_2 * 2^{\text{bias} + 1}$$

$$\text{In this case, we would have } 0.00000001 * 2^{-63 + 1} = 1 * 2^{-8} * 2^{-62} = 2^{-70}$$

♡ ...

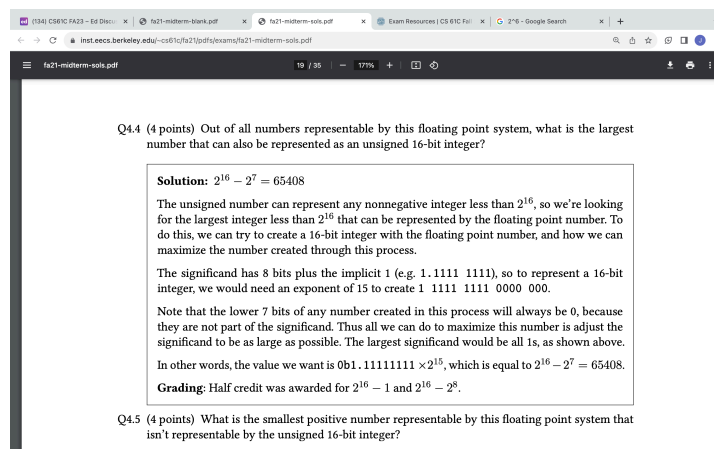


Anonymous Bison 1y #489aad

✓ Resolved

FA21-MT-Q4.4

I'm not really understanding anything after the first paragraph. Can someone explain why we need an exponent of 15 to represent 16 bits?



♡ ...



E Eddy Byun STAFF 1y #489abe

We want to create a 16-bit integer. Currently we have 1.1111 1111. To create a 16-bit integer, we need to multiply this by  $2^{15}$  to get 0b 1111 1111 1000 0000

♡ ...



Anonymous Finch 1y #489aab

✓ Resolved

SP21-MT-Q4a.

Could someone please explain the following code to me?

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
```

The part I'm unsure about is everything slli to j Loop. It seems like it's multiplying the character and saving parts of that multiplied character? How I did it was to read the character, write it once, move the pointer, write, move, write, move, and then jump. Would that work? Thanks!

♡ ...



E Eddy Byun STAFF 1y #489ade

I think the way you do it works as well. Here's how this solution works: let's say you load in the character 0x44 when you do `lbu t0 0(a0)`. This solution takes the character in `t0`, which is going to be 0x00000044, shift it left by 8 (which is shifting it by 1 byte) to get 0x00004400. It then adds 0x00000044 to 0x00004400, which results in 0x00004444. The `slli` and `add` essentially stores two instances of the character inside the `t1` register. Note then we store a half word into `0(a1)`, so we store 0x4444 into `a1` and then we store a byte from `t0` (recall `t0` is 0x00000044 so we store 0x44) into `2(a1)`, which is going to store 3 instances of 0x44 at the pointer `a1` is pointing to.

♡ 1 ...

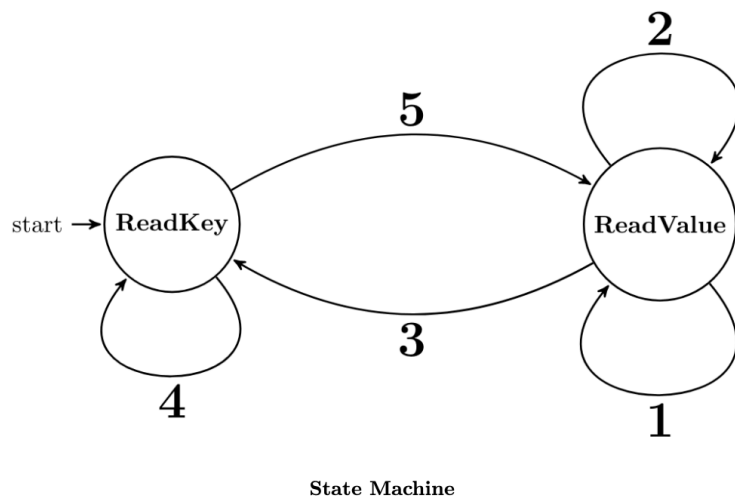


Anonymous Swallow 1y #489aaa

✓ Resolved

SP21-MT-Q6a. I've been staring at this one for a long time, I don't quite get how to approach this problem. I also don't really know how to read the FSM diagram in this case :(.





(a) (1.0 pt) Specify the input and action for **Transition 1**:

- whitespace/append
- whitespace/append,storeValue
- whitespace/consume
- whitespace/consume,storeValue

♡ 1 ...

Isita Talukdar 1y #489afe **ENDORSED**

For this question, you can think of that you only have two possible states, either you are reading a key or reading a value. Transitions 3 and 5 tell you that you need to switch between them, but 1, 4, and 2 tell us you need to be in the same state again.

Looking at Transition 1, we see that all the answer choices have to do with what happens when we get a whitespace while we are reading a key. In the question spec, we know that values have only non-whitespace characters, so we eliminate the first two options, since we don't append a whitespace.

Between the second two, you either consume the whitespace (meaning you skip the whitespace and go to read the next character), or you consume it and store the value. But storing the value should only happen if we know the value is done.

In the question, we know that key value pairs are stored on different lines, so it looks like:

key value'\n'. Only when we see a '\n' should we say the value is done and store it, so that's why it's not the last option.

In summary, transition 1 tells us, if we see a whitespace while reading the value, we should ignore it, not put it in our value, and go to the next character

♡ 2 ...

Anonymous Jellyfish 1y #489ff **Resolved**

FA21-MT-Q5.3

is calculating the distance in memory between instruction and functions in scope?

♡ ...

E Eddy Byun STAFF 1y #489abc

Yes

♡ ...

Anonymous Jellyfish 1y #489fb ✓ Resolved

FA21-MT-Q3:

Line: \_\_\_\_ car = \_\_\_\_\_

The answer key doesn't acknowledge blank 6 (the blank behind car)? Why is there a 6th blank on the question? Was this a trap blank? Typo? The correct answer would have been to leave blank 6 empty but the instructions told me to fill in the blanks so i'm kinda lost...

♡ ...

E Eddy Byun STAFF 1y #489abb

Yea, sorry for the confusion! It's a typo #489ce

♡ ...

Anonymous Snake 1y #489fa ✓ Resolved

SP21-MT-Q8 Part b.ii.A

How do we know there is an exponent bias of 255?

♡ ...

J Justin Yokota STAFF 1y #489fe

Standard bias for IEEE-754 floating point numbers is  $2^{(\text{number of exponent bits} - 1)} - 1$ . Under our current exam policy, this would have been clarified on the exam.

♡ 1 ...

Anonymous Kangaroo 1y #489de ✓ Resolved

SP21-MT-Q8

Can someone explain the step-by-step process of how we get the solutions for part a and part b?

## 8. FLOATING POINT

For the following floating point questions, please use  $\wedge$  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.

$$2^{63} - 2^{45} - 2^{31} + 2^{21}$$

- (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?

$$2^{-16}$$

♡ 3 ...

↳ | **Isita Talukdar** 1y #489aac  
also confused about it

♡ ...

↳ E **Eddy Byun** STAFF 1y #489aaf

For 8a), standard A is 32 bits long and standard B is 64 bits long, and Standard B can represent every positive number that Standard A can. Standard B can represent  $2^{63} - 2^{45} - 2$  non-zero positive numbers. The  $2^{63}$  is from the fact that there are  $2^{63}$  numbers in the floating point representation where the sign bit is 0. The  $-2^{45} - 1$  is from the fact that there are  $2^{45} - 1$  NaNs, and 1 zero. Standard A can represent  $2^{31} - 2^{21} - 2$  positive values using the same logic, except that it has less mantissa and exponent bits. We can then do this operation (# of positive values Standard B can represent) - (# of positive values Standard A can represent) =  $(2^{63} - 2^{45} - 2) - (2^{31} - 2^{21} - 2) = 2^{63} - 2^{45} - 2^{31} + 2^{21}$ , which is our final answer!

For 8b) Our largest number in this range, 80, can be represented as follows:

$1.010000\dots0 * 2^6$ . The next smallest number that can be fully represented is if we take the rightmost bit of the mantissa and change the 0 to a 1. Now, if we shift the decimal point back so that we get  $101000.00000\dots1 * 2^0$ , you should notice the 1 that we changed is in the  $2^{-16}$  position!

♡ ...

↳ **Anonymous Weasel** 1y #489bcc  
if there are  $2^{45}$  NaNs in B, then why  $2^{22}$  NaNs in A?

♡ ...

E **Eddy Byun** STAFF 1y #489bcd  
↩ Replying to Anonymous Weasel

Sorry, that was a typo on my end - it should be  $2^{45} - 1$  NaNs for B and  $2^{21} - 1$  NaNs for A

♡ ...



Anonymous Hawk 1y #489ccc

For 8b, wouldn't that be the smallest step size then?

♡ ...



Anonymous Kangaroo 1y #489dd

✓ Resolved

SP21-MT-Q3

How do I know when there is clk-to-q time and where there is setup time involved?

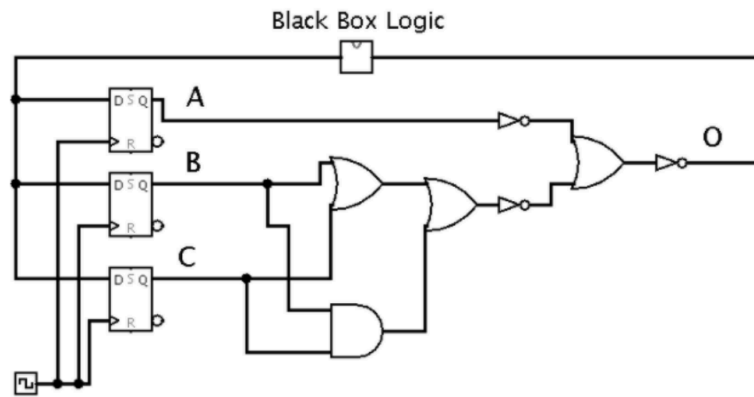
The Clk-To-Q time makes sense since for a register to change, there is clk-to-q time involved but how do I know when there is setup involved?

Why is there setup time for part b and not for part a?

### 3. SDS

For the following question, do NOT include units in your answer!

In the following circuit, the registers have a clk-to-q delay of 6ns and setup times of 5ns. NOT gates have a delay of 3ns, AND and OR gates have a delay of 7ns, and the "Black Box" logic component has a delay of 9ns.



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 =  $6 + 2*3 + 7 + 9 = 28\text{ns}$

(b) (2.5 pt) What is the minimum acceptable clock period for this circuit?

47

The period is determined by the longest path and includes: clk-to-q delay, two NOT gates, three OR gates (or two OR gates and one AND gate), the "Black Box", and the setup time. Minimum period =  $6 + 2*3 + 3*7 + 9 + 5 = 47\text{ns}$

♡ 1 ...



Max Vink 1y #489ee

+1, i don't know why the setup time is not relevant to the shortest path

♡ ...



Justin Yokota STAFF 1y #489fd

One analogy I like to think about is a photoshoot; the idea is that we need to take a bunch of photos of a group of people (once per clock trigger), and between each photo, the people move around to new places. We need to ensure that there's enough time

between photos so that everyone's in their new places, and such that the pictures don't end up blurry.

Setup time and hold time are how much time before and after the photo gets taken that we need everyone to stay still, respectively. Setup time gets added to the clock cycle, since we need to wait that long with everyone staying still before starting the next photo. However, that time has no effect on how long we need people to stay still *after* we take the photo.

♡ ...



Anonymous Kangaroo 1y #489cb

✓ Resolved

SP21-MT-Q8 Part b.ii.A

Why is the answer 0xA6900000?

1) Sign bit is 1 because the number is negative

2) My exponent bits are 0 1001 1011 = 155, because  $155 - 255$  (standard bias for this format) = -100 which is the exponent we have in the number representation.

3) My mantissa bits are 10 1000 0000 0000 0000 0000 = .625 which is exactly what we want.

Putting it all together we get: 1010 0110 1110 1000 0000 0000 0000 0000

This translates to -> 0xA6E80000.

Can someone please point out where am I going wrong?

♡ ...



E Eddy Byun STAFF 1y #489abf

[Reposting because I accidentally deleted my answer D:]

Recall that normal numbers have an implicit 1 before the mantissa.

Here's how I attempted this problem:

I first converted 0.625 to scientific binary form:

$$0.625 = 1.01\ 0000\ 0000\ 0000\ 0000\ 0000 * 2^{-1}$$

Then, simplify our expression:

$$1.01\ 0000\ 0000\ 0000\ 0000\ 0000 * 2^{-1} * 2^{-100} = 1.01\ 0000\ 0000\ 0000\ 0000\ 0000 * 2^{-101}$$

From here, we need to calculate the raw number that we convert the exponent to:  $-101 + 255 = 154$ .

154 to binary: 0b 010011010


Put it all together!

1 010011010 010000000000000000000000


1010 0110 1001 0000 0000 0000 0000

0xA6900000

♡ 1 ...


 **Anonymous Swallow** 1y #489acc  
why is there an extra 0 for the exponent. Isn't the exponent only supposed to have 8 bits?

♡ ...

 **Eddy Byun** STAFF 1y #489ace  
← Replying to Anonymous Swallow


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

♡ ...


 **Anonymous Frog** 1y #489bc ✓ Resolved  
Fa21-Midterm-Q7

Is this in scope?


♡ 1 ...

 **Eddy Byun** STAFF 1y #489ca  
There was no Q7 for the Fa21 Midterm. Did you mean Q6? If so, yes Fa21-Midterm-Q6 is in scope!

♡ ...


 **Anonymous Frog** 1y #489db  
Thanks! I did mean to say Q6 sorry about the confusion. Thanks again for the speedy response!

♡ ...


 **Anonymous Frog** 1y #489ba ✓ Resolved  
Fa21-Midterm-Q5.1

Why can we assume that the buffer is already inputted at the **sp**? Is this just how buffers work? Honestly, I'm still kind of confused what a buffer is.


♡ ...

 **Eddy Byun** STAFF 1y #489cc  
A buffer is some memory that we allocate. For this question, we allocated 20 bytes by decrementing the stack pointer (we decrement by 24 but we use 4 bytes to store `ra`). We then move the stack pointer to `a0` and then call the `Get20chars` function (line 5: `jal ra Get20chars`). The `Get20chars` function is going to fill up the buffer (aka the stack that we allocated) with 20 bytes. As a result, after making the call to `Get20chars`, we can assume that from memory address `sp` to `sp+20`, we inputted characters.

♡ ...

 **Anonymous Frog** 1y #489dc  
Ohhhh I see. We fill up the buffer after calling `Get20chars`. Thanks for clarifying!

♡ ...

 **Anonymous Anteater** 1y #489af ✓ Resolved

**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;
}

```

Why is the third to last line not `ret->car`? Also I don't completely understand the explanation to why you don't need `*f(c->car)`, cause isn't `f` a pointer?

♡ ...

 **Anonymous Frog** 1y #489bb

^ I think it may be a typo but would like a confirmation as well

♡ ...

 **E Eddy Byun** STAFF 1y #489ce

Sorry for any confusion! I think there's a typo in the solutions. You're right - it should be `ret->car`.

You can do either `(*f)(c->car)` or `f(c->car)` are equivalent. You can't do `*f(c->car)` because this will dereference your call to `f`.

♡ ...

 **Anonymous Anteater** 1y #489df

What do you mean by dereference your call to `f`? Like shouldn't `(*f)(c->car)` dereference the pointer too?

♡ ...

 **E Eddy Byun** STAFF 1y #489aba

↩ Replying to Anonymous Anteater

`(*f)(c->car)` and `f(c->car)` are equivalent and acceptable answers. This is different from `*f(c->car)` is going to make the function call to `f` first and then we dereference the return value of `f`, which is different from `(*f)(c->car)` and `f(c->car)`.

♡ 1 ...

 **Anonymous Anteater** 1y #489ae

✓ Resolved

SP21 MT1 Q2

**Q2** Now, Where Did I Put Those Strings?

(10 points)

Consider the following code:

```
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
}
```

The char \*strcpy(char \*dest, char \*src) copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest. The strings may not overlap, and the destination string dest must be large enough to receive the copy.

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.

I dont really get the explanation for these two questions? What is the distinction that makes str1 static, and str2 stack?

♡ ...

E Erik Yang STAFF 1y #489ea

\*str1 is a string constant and is read-only. String constants are stored in the static part of memory. str2[] is a char array that is stored/allocated on the stack and is mutable

♡ ...

Anonymous Anteater 1y #489eb

so in general if something is in the form \*str1, it is static?

♡ ...


E Erik Yang STAFF 1y #489ec

↩ Replying to Anonymous Anteater

for strings defined in funcitons, yeah

♡ ...



 **Anonymous Bear** 1y #489ad ✓ Resolved

SP21-MT-Q7b(ii),

Does this mean hashFunc is a pointer to a function? If so, why don't we dereference hashFunc before calling it in 7b(ii), like this: `*(bf->hashFunc) (element, i)`

```
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 */
};
```

♡ ...

 **Eddy Byun** STAFF 1y #489cf

You don't need to dereference it in C. You could, but it would have to be `(*bf->hasFunc) (element, i)`. `*(bf->hashFunc) (element, i)` is going to dereference the result that you get from calling `hashFunc` on arguments `element` and `i`.

♡ 2 ...

 **Anonymous Otter** 1y #489ab ✓ Resolved

SP21-MT-Q8 Part b.ii.B: How does `0xC07C0000` translate to `-7.75`? I keep getting `-17.75`. Because when I write out the representation of this hex in binary it's

`0b1100 0000 1100 0111 0000 0000 0000 0000`

And group terms together gives me:

`0b1 100000011 000111000000000000000000`

where exponent is 259 (259 - bias gives us 4)

so,

$(-1)^1 * (2^4) * (0b1.000111) = -17.75$

♡ ...

 **Anonymous Kangaroo** 1y #489bf 🔒 ENDORSED


In the first step, 7 in binary is  $\rightarrow$  0111 not 1100

♡ ...

 **Eddy Byun** STAFF 1y #489cd

Yea, what Anon Kangaroo said. It looks like you switched the C and the 7.

♡ ...

 **Anonymous Lobster** 1y #489aa ✓ Resolved

FA21-MT-Q3


Is this question in scope?

♡ ...

 E **Eddy Byun** STAFF 1y #489be

Yes

♡ ...

 **Anonymous Otter** 1y #489d ✓ Resolved

SP21-MT-Q7

(a) Consider the following structure definition. Assume we are using a 32-bit machine.

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

And the following C code

```
void bar(struct foo *f){
    int i;
    ....
    for(i = 0; i < 5, ++i){
        baz(f[i].b);
    }
    ....
}
```

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

8

How is the `sizeof(struct foo) = 8`? Shouldn't it be 5? 1 byte for char a + 4 bytes for pointer b?

♡ 1 ...

 **Anonymous Lark** 1y #489e 👑 ENDORSED

I think you should leave room for padding. On the first line, the char is only 1 byte but we read 4 bytes at one time. So we need to add 3 to that.

♡ ...

 **Andrew Liu** STAFF 1y #489f

Yep! The struct is padded out to a 4 byte boundary.

♡ 1 ...

 **Anonymous Okapi** 1y #489c ✓ Resolved

SP21-MT-Q7:

I'm very very confused on how to get started on this question, is there a way i can find a walkthrough or a more detailed solution of this?

♡ ...

 E **Eddy Byun** STAFF 1y #489bd

Sure, could you specify which subpart you're stuck on? (a or b?)

♡ ...

 **Anonymous Hamster** 1y #489a ✓ Resolved

SP21-MT-Q5

Do the questions mentioned out of scope in the Q&A remain constant for this semester as well?  
Can we get a list of the out of scope ones?

♡ 1 ...



**Jero Wang** STAFF 1y #489b

Q5 is entirely out of scope for FA23 midterm (though please ask - the Q&A are for the specific semester they were asked)

♡ ...