

You are viewing this thread in readonly mode.

[Midterm] Past Exams - 2021 #467

E **Eric Che** STAFF 1,201
9 months ago in **Exam - Midterm** VIEWS



You can find the past exams here: <https://cs61c.org/sp24/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](#) are also available!

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, SU22-MT-Q3, FA23-MT-Q1**

Anonymous Squirrel 8mth #467ccc ✓ Resolved
SP21-MT-Q7

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

Why is it 8 and not 5? I assumed chars are 1 byte and char pointers are 4 bytes so the total would be 1 + 4 = 5 bytes in size



M **Myrah Shah** STAFF 8mth #467cce
[#467cd](#)

Anonymous Lapwing 8mth #467ccb ✓ Resolved
Fall 2021, Q3: Union

Can someone explain all these thoroughly, I understand some part of it, not as a whole. And why 16-4 in this case for sizeof(cons)?

Q3 IC a Scheme

Consider the following C code:

```
union ExtraStuff {
    char a[5];
    uint16_t b;
    int c;
    double d;
};

typedef struct ConsCell {
    void *car;
    void *cdr;
    union ExtraStuff extra;
} cons;
```

Consider the following function: `cons *map(cons *c, (void *)(*f)(void *))`;
map takes a pointer to a cons struct c and a function pointer f.

If the cons struct pointer is NULL, map returns NULL. Otherwise, it does the following:

1. Allocate a new cons struct. ret is a pointer to this new struct.
2. Set the contents of the extra union in ret to be all zeros.
3. Set the car field in ret to the result of calling f on the car pointer in c.
4. Set cdr field in ret to the result of calling map recursively on the cdr pointer in c.

loc

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

    if ( c == NULL ) return NULL ;

    ret = malloc(sizeof(cons));

    ret -> extra .d = 0;

    ret -> car = f(c->car) ;

    ret -> cdr = map(c->cdr, f) ;

    return ret;
}
```

Fall 2021 Question 3

Q3 IC a Scheme

Consider the following C code:

```
union ExtraStuff {
    char a[5];
    uint16_t b; → 16 bytes
    int c;
    double d;
};

typedef struct ConsCell {
    void *car; → 16-4
    void *cdr;
    union ExtraStuff extra;
} cons;
```

Consider the following function: `cons *map(cons *c, (void (*)(void *));`

`map` takes a pointer to a `cons` struct `c` and a function pointer `f`.

If the `cons` struct pointer is `NULL`, `map` returns `NULL`. Otherwise, it does the following:

1. Allocate a new `cons` struct. `ret` is a pointer to this new struct.
2. Set the contents of the `extra` union in `ret` to be all zeros.
3. Set the `car` field in `ret` to the result of calling `f` on the `car` pointer in `c`.
4. Set `cdr` field in `ret` to the result of calling `map` recursively on the `cdr` pointer in `c`.

On a 32-bit architecture, what is `sizeof(cons)`?

16 - 4 for `*car`, 4 for `*cdr`, and 8 for `extra` (as the biggest field in `extra` is a `double`, the union `extra` is of size 8 bytes).



Anonymous Tiger 8mth #467cca

✓ Resolved

Sp21, MT-Q 8.1

What is the minus 2 exponent mantissa bits for?

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.

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



Anonymous Dinosaur 8mth #467cbe

✓ Resolved

FA21-MT-Q4.6

With 9 bits, the largest number that you can represent is $2^9 - 1$ right? and then if we add 1 to that it should just be 2^9 ?

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

Solution: $2^9 + 1$

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

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

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

Grading: Half credit was awarded for $2^8 + 1$.

♡ ...



A

Andy Chen STAFF 8mth #467cbf

You are correct that the largest representable number with 9 bits is $2^9 - 1$, though this question is asking something a bit different. We are looking for the smallest positive number representable by a 16-bit unsigned integer that our floating point system cannot represent. To give an example of the two systems, the number 64 (2^6) can be represented by both systems.

16-bit unsigned: 0000 0000 0100 0000 --> 2^6

16-bit Floating Point: 0 1000101 00000000 --> $2^{(69-63)} * 1.00...00_2 = 2^6$

A 16-bit unsigned integer can represent every integer from 0 to $2^{16} - 1$. So, we're really looking for the smallest integer that our floating point system can't represent. In our floating point system, once the exponent term is 2^9 , the numbers start going up by 2's. We can represent $2^9 + 2$, $2^9 + 4$, and so on (but not $2^9 + 1$, which is the first of the unrepresentable integers). Here's an illustrative example where we find the next biggest representable floating point number starting from 2^9 :

0 1001000 00000000 --> $2^{(72-63)} * 1.00...00_2 = 2^9$

0 1001000 00000001 --> $2^{(72-63)} * 1.00...01_2 = 2^9 * (2^0 + 2^{-8}) = 2^9 + 2^1$

♡ ...



Anonymous Dinosaur 8mth #467cad

✓ Resolved

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.

FA21-MT-Q2

i don't understand when each part of memory is used...

i understand local variables are stored on stack and anything that is dynamically allocated memory is gonna be stored on the heap. but i'm not too sure when code and static is used?

♡ ...

 M **Myrah Shah** STAFF 8mth #467cae

Can you take a look at the [Memory Management section of this discussion](#) and follow up on which part is confusing you? This section breaks down the different segments and what's stored in them.

♡ ...

 **Anonymous Dinosaur** 8mth #467cbc

perfect thank you! :)

♡ ...

 **Anonymous Buffalo** 8mth #467cdb

i don't see how `*str1` will be a part of static. it is declared inside this function and thus shouldn't it be a local var and on the stack?

♡ ...

 **Anonymous Mink** 8mth #467bff ✓ Resolved

Fa21 Mt1Q3.1

How do we know that we are supposed to set `d` to 0? How do we know that it isn't `ret->extra.c?`

Q3 I C a Scheme

(20 points)

Consider the following C code:

```
union ExtraStuff {
    char a[5];
    uint16_t b;
    int c;
    double d;
};
typedef struct ConsCell {
    void *car;
    void *cdr;
    union ExtraStuff extra;
} cons;
```

Consider the following function: `cons *map(cons *c, (void (*)(void *));`

`map` takes a pointer to a `cons` struct `c` and a function pointer `f`.

If the `cons` struct pointer is `NULL`, `map` returns `NULL`. Otherwise, it does the following:

1. Allocate a new `cons` struct. `ret` is a pointer to this new struct.
2. Set the contents of the `extra` union in `ret` to be all zeros.
3. Set the `car` field in `ret` to the result of calling `f` on the `car` pointer in `c`.
4. Set `cdr` field in `ret` to the result of calling `map` recursively on the `cdr` pointer in `c`.

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

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

    if ( _____ ) return _____;

    ret = malloc(_____);

    _____extra_____ = 0;

    _____car = _____;

    _____cdr = _____;

    return ret;
}
```

♡ ...

 **Nikhil Kandkur** STAFF 8mth #467cab

The purpose of the line is to zero out the entire union, and since `ExtraStuff` is a union, we can do this by setting `d`, the largest element of the union, to zero to zero out the union.

♡ ...

 **Anonymous Mink** 8mth #467caf

Is union in scope for the exam?

♡ ...

 **Nikhil Kandkur** STAFF 8mth #467cba

↩ Replying to Anonymous Mink

Since it is part of the C lectures, yes it is in scope.

♡ ...



Anonymous Mink 8mth #467bfd

✓ Resolved

SP21 MT1Q8biib

How do I calculate this?

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

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

0xA6900000

B. (2.0 pt) 0xC07C0000

-7.75



Nikhil Kandkur STAFF 8mth #467cac

If you are referring to part B, what you can do is write out the mantissa first, and then depending on your exponent (after subtracting the bias), you move the "floating point" to the right that many digits. Please refer to discussion 2 for more information on this.



Anonymous Flamingo 8mth #467ccd

If you were referring to part a, I did this.

$-2^{-100} \cdot 0.625$
 sign bit $\ominus 0.625 \cdot 2^{100}$
 ↓ to binary
 $0.101 \cdot 2^{-100}$
 $1.01 \cdot 2^{-101}$
 $x - 255 = -101$
 $x = 154$
 exponent: 010011010
 sign: 1
 mantissa 0100...

1 sign bit 9 exp 22 mantissa
 bias = 255
 because \checkmark Range [1, 510]
 exp

exp mid $2^9 - 2$
 \downarrow 11111110
 $510/2 = \text{bias}$
 bias = -255

$2+4+8+16$

total 101001101010...
 A B C D 0000
 2 hex = 1 byte
 ↳ 8 bits
 0x A6900000
 1 hex = $2^4 = 4$ bits
 2 hex = $2^{16} = 16$ bits



Anonymous Goose 8mth #467bea

✓ Resolved

Spring 2021, Q2 are we suppose to know about interpreters, also what does the first statement mean?

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



Lisa Yan STAFF 8mth #467bee

1) You do not need to know interpreters this semester

2) ISA is an Instruction Set Architecture, such as RISC-V. The ISA generally specifies assembly instructions and their corresponding machine codes.



Anonymous Ram 9mth #467baf

✓ Resolved

SP21-Midterm-Q7: Why is the offset for this question 4? Would it not be 0?

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
```



Lisa Yan STAFF 8mth #467bcd

Because of 32-b word alignment, `sizeof(*f)=8` (part (i)), and `f[i].b` (the second field of `f[i]`) is at the address `&f[i] + 4`. The first two instructions compute the value `&f[i]` in `a0`, and then we add 4 to get the proper address.



Anonymous Goose 9mth #467bad

✓ Resolved

Spring 2021, is a question like this on pipelining in scope for us?

(c) (1.5 points) **Pipelining**

Assume we've added pipeline registers to create a 6-stage pipeline with our updated datapath, as seen below (Figure 3). This pipelined datapath acts similarly to our standard RISC-V 5-stage pipeline with the additional change of the memory section being separated into two sections. Assume no forwarding logic has been implemented, no branch prediction, and one register operation per cycle.

For the given code, what hazards might exist and due to which lines? Assume all registers have been initialised and that all labels are defined and that all branches are taken. **HINT: having a table open will be useful for scratch work.**

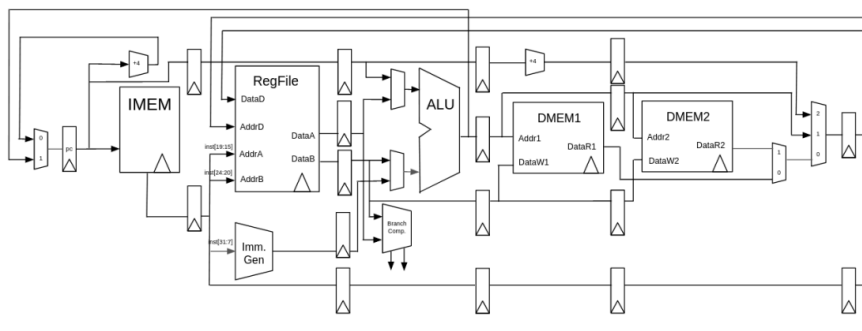


Figure 3

```
1 bne x0, t0, next
2 addi t1, t0, 1
3 lb s0, 0(t1)
4 shw s0, 4(t0)
```

i. (0.5 pt) Lines 1/2



Jedidiah Tsang STAFF 9mth #467bbc

nope!



Anonymous Raccoon 9mth #467baa

✓ Resolved

FA23-MT-Q4

I'm particularly confused on this part, how do we know we need an exponent of 15?

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

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

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

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

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

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

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



Anonymous Crane 9mth #467bac

+also curious how we get 2^7 in this problem?



Anonymous Goose 9mth #467bae

There's 7 bits that are 0 and 16 bits total. So we have $2^{16}-2^7$



Anonymous Tiger 9mth #467aff

✓ Resolved

Q4.6-MT-FA21

Im confused why we include the implicit 1. We cannot control whether that is a 1 or 0 so how can we use it to make more numbers available to us.

I currently understand that because we have 8 bits of mantissa, and that we can push the decimal past those 8 bits, we must be able to accurately represent the first 2^8 numbers.



Lisa Yan STAFF 8mth #467bcf

Great insight—it's precisely because we can't control the implicit 1 that it must be part of every (normalized) floating point representation.

Assuming $X=0/1$: $1.xxxxxxxx \times 2^e$ is the normalized form, to some exponent e . The smallest possible representable number with the same exponent is 1.00000000×2^e , and so therefore the next smallest number that we *cannot* represent is 1.00000001×2^e . To find the unsigned **integer** that matches this form, $e=9$ and therefore the number is $2^9 + 1$.

As a sanity check, we could consider denormalized form, which doesn't have the implicit 1, but this option is not viable because any denormalized number in this provided floating rep is a non-integer.



Anonymous Raccoon 9mth #467afe

✓ Resolved

FA21-MT-Q2.9

Why is this '\0'dlr correct in little endian? wouldn't r be the most significant bit and therefore stored at the highest memory address meaning r is first?

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

Casting `str1` to `(uint32_t*)` doesn't change the fact that `str1` is still a pointer to the bytes `Hello World`, but those bytes are now being read as an array of `uint32_t` (32-bit = 4-byte unsigned integers), instead of an array of `chars`.

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

Finally, we need to interpret these four bytes (`rld` and the null byte) as a `uint32_t`. We can use the ASCII table to look up the bytes being stored in memory to represent these characters. From lowest to highest memory address, the bytes are `0x72` (`r`), `0x6C` (`l`), `0x64` (`d`), and `0x00` (null byte).

Because the system is little-endian, the most-significant byte is stored at the highest memory address. In other words, we should read the integer starting from the highest address and ending at the lowest address. This gives us `0x00646C72`.

♡ ...



Lisa Yan STAFF 8mth #467bda

Character arrays are stored in-order, from lowest address to highest address. Endianness impacts how **words** are loaded from/stored to memory, byte-by-byte.

In this case, the character array `Hello World\0` was stored in byte-order, because it was considered a string (and therefore effectively written byte-by-byte). However, because of the cast, loading in `((uint32_t *) str1)[2]` reads in a full 4B word, assuming little endian, and hence `\0` is considered the most-significant byte.



♡ ...



Anonymous Raccoon 9mth #467afd

✓ Resolved

SP21-MT-Q3 Is a valid path (ie. something we track when finding minimum path vs maximum path) always from one register to another (including itself) or from one input to a register?

♡ ...



Lisa Yan STAFF 8mth #467bdc

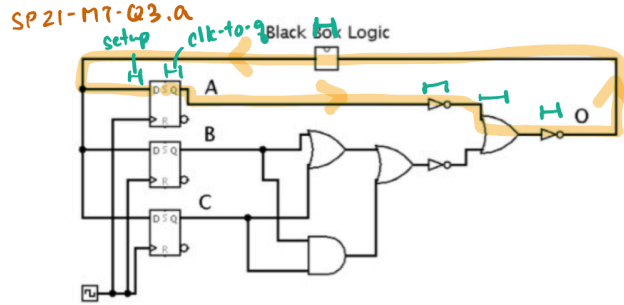
Good question—I took the liberty of updating your question label, but let me know if I misunderstood. A path is defined as how you track a signal between two clocked circuit elements (possibly the same one). More precisely you can define it as the time it takes, starting from a rising clock edge, for a signal to be stable at the input of a clocked element.

My strategy: If the two clocked elements are registers:

- start with `clk-to-q` (the time it takes for register output to be stable)
- add all combinational logic block delays until we hit a register (i.e., the second clock element)
- end with setup time (the time required for register input to be stable)

Here's the diagram of SP21-MT-Q3.a.

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.



♡ ...



Anonymous Raccoon 9mth #467aed

✓ Resolved

FA23-MT-Q3 Why do we not need to dereference f when we call it since f is a pointer to a function? Or is it dereferenced when its passed in? For instance here we would want to call f (so f(c->car) but when we pass it into map don't we want the pointer to f?)

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

    if ( c == NULL ) return NULL;

    ret = malloc(sizeof(cons));

    ret->extra.d = 0;

    ret->car = f(c->car);

    ret->cdr = map(c->cdr, f);

    return ret;
}
```

♡ ...



Lisa Yan STAFF 8mth #467bdd

Good question; this is a C quirk, meaning that *f and &f are both syntactic sugar for f . The only time we need * is in the declaration of a function pointer type, as shown in the parameter definition (void *) (*f) (void *) . In all other places where the variable f is assigned, called, or used, we can drop * (and we should, for stylistic clarity). More here:

[#179aa](#)

♡ ...



Anonymous Goose 9mth #467aeb

✓ Resolved

FA21. Q2.1

I don't understand why str1 is located in static memory - wouldn't it just be on the stack since it's a local variable inside foo? How can we tell what is located in static memory vs what is not? Also I

thought `*str1` is dereferencing the pointer `str1` but it seems like this question is referring to the address stored in `str1`. what exactly does that mean?

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.

♡ ...

↳ V **Vinay Agrawal** STAFF 9mth #467aef

When a string is directly assigned to a pointer, it is stored as a read-only block in static memory (typically `chr* str = "Something";`). The address of this string however, which is the value of `str1`, is stored on the stack. This is specifically a behavior of strings in C. If a string is directly assigned to an array, then the array is stored on the stack and you are free to modify it as `str2[1] = 'a';`

♡ 2 ...

↳ **Anonymous Tiger** 9mth #467afa

how would the answer change if we were dealing with ints instead of chars?

♡ ...

↳ V **Vinay Agrawal** STAFF 9mth #467afb

↩ Replying to Anonymous Tiger

If we tried doing `int* ptr_num = 5`, then this would cast 5 into a pointer, which is not very useful and potentially dangerous because we don't really know what part of memory that is. `ptr_num` would be stored on the stack but `*ptr_num` is uncertain. For defined behavior, we should `malloc(sizeof(int))` when creating an integer pointer. If we tried using `int num_arr[] = {1, 2, 3, 4, 5}` then this would be stored on the stack.

♡ ...

↳ **Anonymous Tiger** 9mth #467afc

↩ Replying to Vinay Agrawal

So is it just a special case for chars that allows us to store the string into static memory? Or could we create a scenario where we store an int into static memory from within a

function? Right now, it seems the only possible way to store an int into static memory would be to define it in the global frame.

♡ ...

V **Vinay Agrawal** STAFF 9mth #467bba

↩ Replying to Anonymous Tiger

Yes strings are special, but this can also be done for integers with a little more effort. ~~To store an integer in static memory, we would have to either use a preprocessor directive, such as `#define NUM_LOOPS 3`, or use the const qualifier when defining a read-only integer within a function, such as `const uint32_t NUM_LOOPS = 3`.~~ EDIT: Using the `#define` preprocessor directive does not store values in static memory. Preprocessor directives are used by compilers to replace the text of the program with the specific value.

♡ ...

 **Anonymous Tiger** 9mth #467bbd

↩ Replying to Vinay Agrawal

I am currently looking at HW 2.5, q1.6 and the solution seems to disagree with `#define` being put into static memory. Is there a difference between the hw question and what you are suggesting?

♡ ...

V **Vinay Agrawal** STAFF 9mth #467bbe

↩ Replying to Anonymous Tiger

Good catch! The HW is right. When using `#define`, all it does is string replacement by changing the text of the program with the preprocessor directive value. I will make an update on my previous reply.

♡ ...



Dan Nguyen 9mth #467acb

✓ Resolved

FA21-MT-Q5

What is stdin, how do we use/interpret it, and do we have to know what it is in the exam?

♡ ...



Justin Yokota STAFF 9mth #467acc

stdin is a special FILE* that connects to the terminal for input. Much as how stdout gets used whenever you print stuff, stdin gets used whenever you request user input. It's the equivalent of Python's `input()` function.

♡ ...



Anonymous Dolphin 9mth #467abf

✓ Resolved

FA21-mt-q4.5

Q4.5 (4 points) What is the smallest positive number representable by this floating point system that isn't representable by the unsigned 16-bit integer?

Solution: 2^{-70}

Floating point numbers can represent fractional numbers between 0 and 1, but integers cannot represent fractional numbers between 0 and 1. Thus we are looking for the smallest positive number representable by the floating point number.

The smallest positive numbers representable in floating point are the denorms. The smallest denorm can be obtained by using a denorm exponent of 0 and the smallest possible mantissa. This gives us 2^{-70} .

How did they get -70 from 7 bit exponent 😞

♡ ...



Justin Yokota STAFF 9mth #467acd

This is a combination of the exponent AND number of mantissa bits. We effectively find the smallest positive representable number.

♡ ...



Anonymous Raccoon 9mth #467bab

Why would this be -70 and not -71?

♡ ...



Anonymous Dolphin 9mth #467bbf

Yea shouldn't it be $2^{-63} * 2^{-8} = 2^{-71}$?

♡ ...



Justin Yokota STAFF 9mth #467bca

↩ Replying to Anonymous Dolphin

Note that denorms use an exponent one higher than normal!

♡ ...



Anonymous Dolphin 9mth #467bcb

↩ Replying to Justin Yokota

facs. Justin why aren't you sleeping

♡ 1 ...



Anonymous Swan 9mth #467abe

✓ Resolved

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

Solution: $2^9 + 1$

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

FA21-MT-Q4.6

since a significand of 9 bits is not representable, then 1.000000001 is not representable. if this significand multiplied by any power of 2 is still not representable then if we want the smallest shouldn't we choose an exponent of 0? i'm a little confused why we choose exponent of 9 to get 1000000001 as the smallest not representable if $1.000000001 * 2^0$ is also not representable?

♡ ...



Justin Yokota STAFF 9mth #467ace

If we picked an exponent of 0, we'd get a number that's not an integer. This would therefore not be representable by the uint16.

♡ ...



Anonymous Dolphin 9mth #467abd

✓ Resolved

FA21-MT-q3

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

map takes in a cons* and f. But c->cdr is a void*. How does map take this in?

♡ ...



Justin Yokota STAFF 9mth #467acf

void*s get implicitly typecast to any other pointer type.

♡ ...



Anonymous Dolphin 9mth #467abc

✓ Resolved

FA21-MT-Q3

Blank 5 sets the contents of the **extra** union to be all zeros. This was probably the hardest blank in this question! The key observation is that the largest element in the union is **double d**, which is 8 bytes = 64 bits. (**char a[5]** is 5 bytes = 40 bits, **uint16_t b** is 2 bytes = 16 bits, and **int c** is 4 bytes = 32 bits.) Thus, if we set the largest element in the union to 0, all the other union elements using that same memory will also be set to 0.

Accepted solutions:

- **d**

I thought structs allocates enough memory for all variables in the struct?!?!? Why would double d overwrite the other variables?

♡ ...




Justin Yokota STAFF 9mth #467ada

structs do. This question uses a union, not a struct.

♡ 1 ...

 **Anonymous Mink** 8mth #467bfe
Are unions in scope?
♡ ...

 **Anonymous Dolphin** 9mth #467abb ✓ Resolved
FA21-MT-Q2

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`).


Conceptually, when we return `str2`, will *only* the first item in the array stay saved? Since when we return `str1` for example we return the pointer safely, so from the same logic returning `str2` will return only the first item safely right? the rest will get erased?

♡ ...

 **Justin Yokota** STAFF 9mth #467adb

No; none of the items stay saved, since all the data is on the stack in a segment that will get overwritten.

♡ ...

 **Anonymous Dolphin** 9mth #467aba ✓ Resolved

FA21-MT-Q2. Can someone explain the answers below? Why are they treated differently? Where can I find more information about this?

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.

♡ ...



Justin Yokota STAFF 9mth #467adc

This is mostly a quirk of how C treats its variables; `char*`s are pointers which could point anywhere (and there's a preference for putting things in the data segment when possible), and `char[]`s are arrays, which get treated as a sequence of local variables (and therefore gets placed on the stack).

♡ ...



Anonymous Tiger 9mth #467aee

Would the same hold if we were dealing with other types? For instance,

```
void foo() {
    int x = 5;
    int *p = &x;
}
```

Would `x` and `*p` be in static and `p` be in the stack?

My thinking is that `x` represents an `int` which could be placed into the data segment. Therefore, dereferencing `p` is equivalent to finding what `x` is. And because `p` is a pointer declared inside of a function, it is placed in the stack.

♡ ...



Anonymous Raccoon 9mth #467aaf ✓ Resolved

FA23-MT-Q3 why are we allowed to ignore the space (bytes) the other parts of the extra struct use and just use the largest part of the struct to represent the total memory?

♡ ...



Justin Yokota STAFF 9mth #467add

That's not a struct, but a union. Unions have different behavior from structs.

♡ ...



Anonymous Raccoon 9mth #467aae ✓ Resolved

FA23-MT-Q3 Why do we not have to cast the void pointers to cons pointers when we call f or map?

♡ ...



Justin Yokota STAFF 9mth #467ade

Void pointers implicitly get typecast to other pointer types.

♡ ...



Anonymous Raccoon 9mth #467aec

So it is always unnecessary to explicitly cast a void pointer?

♡ ...



Lisa Yan STAFF 8mth #467bdb

↩ Replying to Anonymous Raccoon

This is a stylistic decision. We wouldn't take off credit if you chose to explicitly cast. See this discussion: [#114db](#)

♡ ...



Anonymous Weasel 9mth #467aad ✓ Resolved

Spring21 Q5 datapath pipelining is not in scope correct?

♡ ...



Eddy Byun STAFF 9mth #467aca

Pipelining is not in scope. For this particular problem, it looks like parts a and b are in scope because they cover the single cycle datapath

♡ 2 ...



Anonymous Weasel 9mth #467aac ✓ Resolved

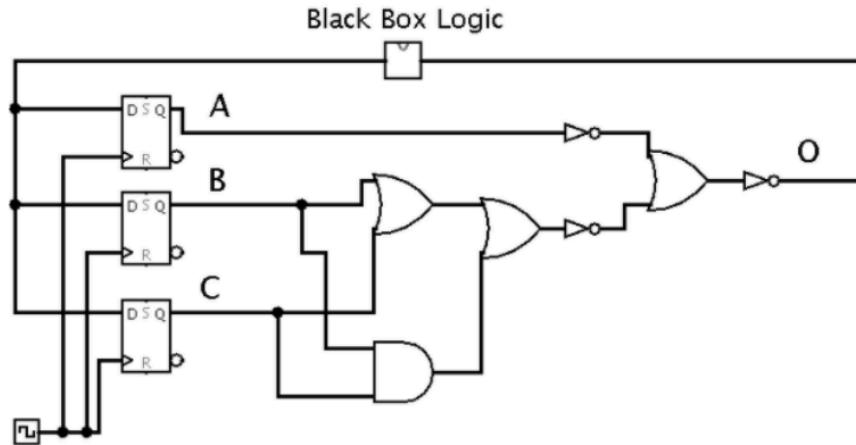
Spring 21, 3(a)

can someone reminds me how to calculate the maximum allowable hold time of the registers, you can use this problem as an example and reference

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

♡ ...

Justin Yokota STAFF 9mth #467adf

The maximum hold time is the same as the first time a register input changes in response to the clock tick. In this case, this first happens at time 28 ns, at the input to A.

♡ ...

Anonymous Turtle 9mth #467bcc

Why do we include the "Black Box" delay in it?

♡ ...

Anonymous Flamingo 8mth #467cbd

↩ Replying to Anonymous Turtle

Because in order for the cycle to be complete (output of one register to input of another or Q side to D side), the blackbox has to be passed through. In other words, it's part of the logic

♡ ...

Anonymous Koala 9mth #467aab ✓ Resolved

[Fa21 Q3] I looked at the solutions of extra.d is 0 but I still do not get why that is the case.

♡ ...

Justin Yokota STAFF 9mth #467aea

extra.d is the double, which overlaps with all other elements of the union. Setting the double to 0 also sets all components of the union to 0.

♡ ...

Anonymous Dunlin 9mth #467aaa ✓ Resolved

8. FLOATING POINT

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

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

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

SP21-MT-Q8(a)

How to get this answer?

♡ ...

 **Lisa Yan** STAFF 8mth #467bde

Define X, Y as 0/1 bit values, but strings X...X and Y...Y not all-0s. Also define y_32, y_64 as remainders from the total.

32b		s = 0 values	64b	non-zero positive value?
1		0 0...0 0.....0	1	F (+0)
2^21 - 1		0 0...0 X.....X	2^45 - 1	T (+denorm)
1		0 1...1 0.....0	1	T (+inf)
2^21 - 1		0 1...1 X.....X	2^45 - 1	F (NaN)
y_32		0 Y...Y X.....X	y_64	T (+norm)
-----			-----	
2^31			2^63	

Therefore the difference in rows 2, 3, and 5 is:

$$(2^{63} - 1 - (2^{45} - 1)) - (2^{31} - 1 - (2^{21} - 1)) = 2^{63} - 2^{45} - 2^{31} + 2^{21}$$

♡ ...

 **Anonymous Dunlin** 8mth #467bdf

omg thank you so much Lisa this is so clear 😭

♡ 1 ...

 **Anonymous Kouprey** 8mth #467ccf

if we only count positive numbers for 32 bit number,

why is it $2^{32} - 1 - (2^{21} - 1)$ # total number can represent - positive zero - total number of positive NaN

but not $2^{31} - 1 - (2^{21} - 1)$ # total positive number can represent - positive zero - total number of positive NaN

♡ ...

 **Lisa Yan** STAFF 8mth #467cda

↩ Replying to Anonymous Kouprey

That's a typo on my side; solutions are correct. Fixed the above latex math equation, thanks!

♡ ...

 **Anonymous Mantis** 9mth #467fb

✓ Resolved

SP21-MT-Q5(b)(i)

i. (4.0 pt) What is the minimum amount of additional control logic and hardware may be needed to implement this in the datapath for it to be functional? Select all that apply.

- Path from MEM1 output to MEM2 input.
- Path from ALU to MEM1 input.
- Add additional state element.
- Path from control logic to additional mux.
- Path from ALU to MEM2 input.
- Path from control logic to MEM1.
- Path from MEM2 out to WB mux.
- Path from control logic to MEM2.
- Path from MEM2 to control logic.
- Path from MEM1 to control logic.
- Add additional mux.
- Path from control logic to additional state element.
- Path from MEM1 to WB mux.

In order of the options given above: For read data output; We don't need any sort of connection there; the logic is done through control; For read data output; Signal for when we don't find the data we need in the first portion of main memory; set to 1 when we don't find it and have to subsequently search MEM2; upon a miss there, we'd fetch from disk (but that doesn't matter at this point in this class); Regardless of what happens in MEM2, there will be no more logic undertaken so we don't need a path in that direction; For Mem1RW; For Mem2RW; For the absolute address with offset; For the absolute address with offset; we don't want this passed into MEM2 as the output of MEM1 because the path no longer has the value we want which is the address; For selecting output value from DMEM1 or DMEM2 to feed into the 0th input of the writeback mux; Selector to determine output of memory output selector; We do not want additional memory states given otherwise that would change our critical paths and it would not accomplish the output selector; Selector not needed if state element is unnecessary.

What is the purpose of the additional mux here? Does it help select between MEM1 and MEM2?

♡ ...



E Erik Yang STAFF 8mth #467bfa

we need an additional mux to see if the data address was found in MEM1; that *if* condition means that we need some sort of selector to either go straight to writeback or ALU

♡ ...



Anonymous Swan 9mth #467fa

✓ Resolved

FA21-MT-Q4.4

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

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

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

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

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

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

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

Is it basically saying that we want to find the largest u16int in floating point form so we max out the significand and we want to find an exponent such that the significand of 1.11111111 will give

us a total of 16 bits which is an exponent of 15? How do we know that max out the significand is the right method and will produce an integer instead of a FP decimal value?

♡ ...

J Justin Yokota STAFF 9mth #467ff

Well, ideally. However, the largest uint16 isn't necessarily representable. In this case, we know that 2^{16} is representable by the floating point system, and is barely larger than all representable uint16s. So we can find the next largest floating point number and confirm separately that the number is representable in uint16.

♡ ...

Anonymous Finch 9mth #467ef ✓ Resolved

SP21-MT-Q8(b)(ii)(A)

The answer key says the exponent for this value $-2 * 2^{(-100)} * 0.625$ is equal to 77. (I converted their answer from hex to decimal). How did they arrive at this number? if the bias is 127, wouldn't this give us an exponent of -50, not -100 as desired?

♡ ...

L Lisa Yan STAFF 8mth #467bed

The bias is 255 because we have 9 bits of exponent. Conversion then follows:

```
-1 x 0.625 x 2(-100) # DECIMAL POINT
-1 x 0.101 x 2(-100) # BINARY POINT
-1 x 1.01 x 2(-101)
exp = -101 + 255 = 154 010011010
```

```
num = 1 010011010 010...0
      1010 0110 1001 0..0 ... 0...0
0x   A   6   9   0   ... 0
```

Given the confusion, after Spring 2021 we started providing the bias explicitly on exams.

♡ 1 ...

Anonymous Swan 9mth #467ee ✓ Resolved

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

FA21-MT-Q2.9 are arrays stored differently than other values? since if this string is contiguous, why wouldn't 'H' be considered the most significant and thus stored at highest address for little endian?

♡ ...

J Justin Yokota STAFF 9mth #467fe

"Most significant" and "Least significant" apply only to individual items; the byte "H" is both the most and least significant byte of the zeroth element of `str1`. Arrays are always stored in contiguous blocks, with the 0th block being at the lowest address.


♡ ...

Anonymous Swan 9mth #467ed ✓ Resolved


FA21-MT-Q2.1

what is the difference between *str1 and *str2? if they are both pointers to character arrays (since a literal string is an array of chars), why is str1 stored as static and not considered a local variable while str2 is?

♡ ...

 **M** [Myrah Shah](#) STAFF 9mth #467bbb
[#467aef](#)

♡ ...

 [Anonymous Spider](#) 9mth #467eb ✓ Resolved
[sp21 final] Can somebody please explain the rules?

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

♡ ...


 **L** [Luca Poulos](#) 8mth #467bce

When creating structs, the fields with the most bytes determine the alignment to memory. In this case, it is 4 bytes.

In this example, consider the struct address as 0x1000. Then `a` is at 0x1000, `b` is at 0x1001, but we cannot insert `c` into 0x1003 (it will not be aligned with 4-byte memory address). So instead, we add one byte of padding and store `c` at 0x1004. `d` is stored at 0x1008 for a total of 12 bytes. We know `d` is 4 bytes because we're on 32-bit architecture with 4 bytes pointers.

Now if we swap the order of `b` and `c`, then we would need 1 + 3 padding for `a`, 4 bytes for `c`, and 2 + 2 padding for `b`, and 4 bytes for `d`. Totaling 16 bytes instead of 12.

♡ 1 ...

 [Anonymous Toad](#) 9mth #467ea ✓ Resolved
SP21-MT-Q8ab

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?

$$2^{-16}$$

I'm pretty lost on how we got these answers. Any guidance?



Lisa Yan STAFF 8mth #467beb

8(a): #467aaa

8(b)(i): Check out Su22 2021 Past Exam Ed thread (link), search for Anonymous Partridge/#635a



Dan Nguyen 9mth #467cb

✓ Resolved

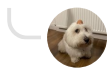
[FA21 MT1 Q5.2]

If not explicitly specified, do we assume we use 2 complement instead of unsigned numbers for representing negative numbers in machine-language representation?



Justin Yokota STAFF 9mth #467cc


RISC-V specifically mandates that its numbers are interpreted as 2's complement signed numbers. C also has a similar specification as of January 2024, and it's been a de facto standard for much longer.



Dan Nguyen 9mth #467ec


thank you so much mr.yakulta



 **Anonymous Toad** 9mth #467ca ✓ Resolved
[SP21 MT1 Q7aii]


Why is the size of the struct not the size of its elements ($1 + 4 = 5$). Is this because we word-align (and the closest next word would be 8?)

♡ ...

 **Andrew Liu** STAFF 9mth #467cd


The struct must be padded to be a multiple of the element with the largest align. In this case, the pointer has `alignof 4`, so we pad to the nearest multiple of 4. In this class, you can assume that the alignment of a type is the same as its size.

♡ ...

 **Anonymous Toad** 9mth #467bf ✓ Resolved
[SP21 MT1]


For number 5, was only part a in scope? Also is part 7b in scope?

♡ ...


 **Andrew Liu** STAFF 9mth #467ce

Parts a and b are in scope for this semester's midterm

♡ ...

 **Anonymous Spider** 9mth #467be ✓ Resolved
[sp21 mt1 2b] is this in ? If so, where can I learn more about this ?


♡ ...

 **Andrew Liu** STAFF 9mth #467cf

CALL is in scope. Some resources I would recommend are the lectures on CALL, the various TA discussion slides on CALL, the discussion worksheets on CALL, and the homework questions on CALL.

The discussion slides can be found in the central index.

♡ ...

 **Anonymous Kangaroo** 9mth #467bd ✓ Resolved
SP21-Final-Q7b,

- B.** What changes would you need to make in order for the instruction to be able to execute correctly? Assume all modifications and additions are done on top of the existing single cycle datapath. Select all that apply
- Modify the control signals to the ALU.
 - Modify Branch Comparator logic.
 - Modify the control logic for WBSel.
 - Modify the control logic for parsing instr[31:0].
 - Modify control logic for ALU/ALUSel.
 - Add an additional comparator.
 - Add additional control signals for the writeback mux.
 - Modify ALU buses.
 - Modify the control logic for the Branch Comparator.
 - None of the above.

7. Datapath

(a) NEWINSTR

Given the standard RISC-V datapath (Figure 1), determine if the following is implementable or not without any additional functional units? Assume the instruction is not a pseudoinstruction encoding.

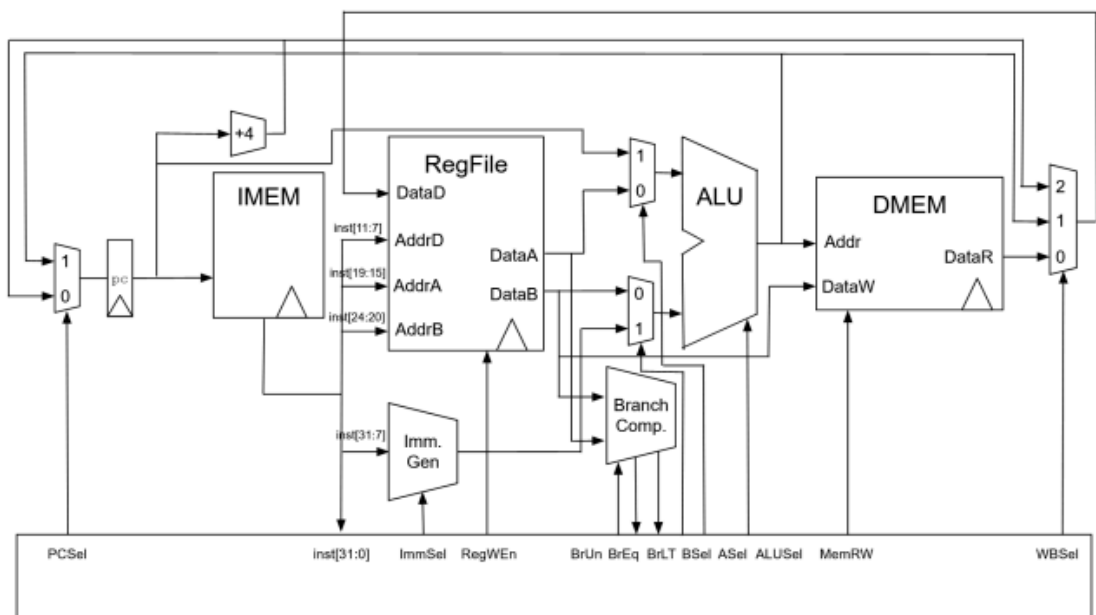


Figure 1

i. Is Null

- A.** `is_null rd, rs1`: check if an input given through `rs1` is considered NULL or not by C standard. The result is returned through `rd` as a bit.

Hi,

Could I get an explanation as to how these modifications and additions would allow us to implement the function `is_null`. I am mainly confused as to how we send the result back through `rd` as a bit after we modify the control logic for the branch comparator.

♡ ...



Andrew Liu STAFF 9mth #467da

NULL is just 0, so you can allow WBSel to send back `rs1 == 0`.



Anonymous Swan 9mth #467bc

✓ Resolved

1 sign
9 exp
22 mantissa

$$-0.625 * 2^{-100} = (1011 * 2^{-3}) * 2^{-100}$$

$$= (1011 * 2^{-103})$$

$$= 1.011 * 2^{-100}$$

$$0.625 = 0.101 = 0101 * 2^{-3}$$

$$\begin{array}{r} 1010 \\ + 1 \\ \hline -0.625 = 1011 * 2^{-3} \end{array}$$

$$\text{exp field} = -100 + (-255) = -355$$

$$355: \begin{array}{r} 0101100011 \\ 1010011100 \\ + 1 \\ \hline -355: 1010011101 \end{array}$$

mantissa: 011 0000 0000 0000 0000 0000

sign: 1

FP: 1 | 1010011101 011 0000 0000 0000 0000 0000

Sp21-MT-Q8bii.A.

for the conversion of $-2^{-100} * 0.625$, this is my work so far but i'm kind of stuck on the conversion of the exponent field. i calculated the bias to be -255 using standard bias with 9 exponent bits but when it convert using 2's complement, i get 10 bits while preserving the sign but only 9 exponent bits are allowed?



Andrew Liu STAFF 9mth #467db

You should subtract the bias from the value to get the value to be stored in binary, since binary + bias = value



Anonymous Swan 9mth #467bb

✓ Resolved

SP21-MT-Q8b)

how is the step size calculated?

(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}



Andrew Liu STAFF 9mth #467dc

Step size is the value of the LSB times 2 to the exponent. Thus, step size increases as the absolute value of the number goes up. In this case, you should calculate the step size of 80. Since 80 is $1.25 * 64 = 1.01_2 * 2^6$, the stepsize is 2^{-22} (value of LSB) $\times 2^6$ (exponent) = 2^{-16}





Anonymous Swan 9mth #467fd

So the exponent we multiply by is the exponent of the largest number in the range of values we're trying to find and the LSB is the smallest bit value in the number exponent bits we have? What is the intuition behind why this equation works? I can't seem to find the equation in lecture somewhere. Thanks!



Lisa Yan STAFF 8mth #467bec

← Replying to Anonymous Swan

L06-Slide 25 gives some reasonable intuition:

Step Size

What is the next representable number **after y**? Before y?

31	30	23	22	0		
0	1000 0001	111 0000 0000 0000 0000	0000	0000	y	
s		exponent			significand	
				+1		
0	1000 0001	111 0000 0000 0000 0000	0001		next float after y	

$y + ((.0\dots01)_{\text{two}} \times 2^{(129-127)})$
 $y + (2^{-23} \times 2^{(2)})$
 $y + 2^{-21}$
 "step size"

Because we have a fixed # of bits, we **cannot** represent all numbers in a range.
Step size is the spacing between consecutive floats with a given exponent.

- Bigger exponents → bigger step size.
- Smaller exponents → smaller step size!

Berkeley
06-Floating Point (25)



Anonymous Mink 8mth #467bfb

Where does the 64 come from?



Anonymous Swan 8mth #467cbb

So the step size can be different in between that range due to the different exponent bits so the largest step size would be to calculate the step size of $y = 80$ and $y + 1$? But since 80 is the largest possible value in the range, would this also be equal to the step size of $y = 80$ and $y - 1$?



Anonymous Dogfish 9mth #467ba

✓ Resolved

For Fa21-MT-Q5.1, can we use lw instead and increment addresses by 4?



Andrew Liu STAFF 9mth #467dd

No, since that would give a high chance of missing the null terminator and infinitely looping.



Anonymous Stork 9mth #467af

✓ Resolved

SP21-MT-Q2

"Compilers produce larger code than interpreters but do it faster"

Should we know how to compare compilers to interpreters?



Andrew Liu STAFF 9mth #467de

You should know the basics of what makes an interpreted language different than a compiled one. **In my opinion**, this question is a relatively obscure MC when comparing the two

♡ ...



Anonymous Magpie 9mth #467ae ✓ Resolved

SP21-MT-Q1

For question 1c) part vi where it asks -25 in bias notation (with an added bias of -63) how is this value in the range of numbers we can represent since I thought the new range would be [63, 190] since it is the unsigned range plus the bias and if you add the bias to -25 the new shifted value would be 38 which is not in the range of values you can represent?

♡ ...



Andrew Liu STAFF 9mth #467df

You add the bias to the bits to get the value. So, -25 is the value, and if you compute bits + bias = value, you find that bits = value - bias = $-25 - (-63) = 38$, which is representable in our bits.

♡ ...



Anonymous Swan 9mth #467f ✓ Resolved

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

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

[SP21-MT-Q7b]

for part i, does the division round down? so if size is 4 bits, then we would have $4/8+1 = 0+1 = 1$ byte to allocate? also, what is the reason behind using calloc instead of malloc here?

for part iii, i'm thinking we need the one to or with the nth bit to turn it into a one but why do we need to left shift?

♡ ...



Andrew Liu STAFF 9mth #467aa

Yes, C integer division is defined as truncating towards 0, see C11 6.5.5 p6: <http://port70.net/%7Eensz/c/c11/n1570.html#6.5.5>, and you are exactly right about the plus 1 accounting for this truncating behavior

We use `calloc` because the bits must be initialized to 0.

For part iii, we left shift it to place the 1 in the correct bit. (e.g. $1 \ll 2 = 0b0100$)

♡ ...



Anonymous Swan 9mth #467e ✓ Resolved

iv. load `s1->next` into `t0`

```
lw t0 t0(4)
```

[SP21-Final-Q6b(4)] how do we know that the next variable is in the 4 byte offset of `t0` if `t0` currently has `s1` (which I believe is the struct)?



Andrew Liu STAFF 9mth #467ab

We know this based on the struct. A struct is a package of variables guaranteed to be contiguous in memory, so we know that offset of 0 is data, and an offset of 4 is next.



Anonymous Swan 9mth #467c

✓ Resolved

Q2.1 (3.5 points) 37

Solution: Answer: 36. The adjacent floating point numbers are 36 (0b 0 10100 0010) and 38 (0b 0 10100 0011)

Q2.2 (3.5 points) 1/3 (whose binary representation is 0b0.0101 0101...)

Solution: Answer: $21 * 2^{-6} = 0.328125$. We can move the binary point to get our floating point representation $0b1.010101... * 2^{-2}$. The mantissa rounds down, so our float would be $0b1.0101 * 2^{-2}$, or $0b10101 * 2^{-6}$ or $21 * 2^{-6}$

FA21-Final-Q2.1

if the resulting adjacent representable numbers to 37 are 36 and 38, why would we choose to round down instead of up if they are both equally apart? is it related to when we convert 36 to FP representation and remove the bit to fit 4-bit significant, we get the rounded 36 whereas for 38 we would need to flip the LSB?



Justin Yokota STAFF 9mth #467d

Per the question setup, rounding always goes toward the most even number; in this case, 36 has a 0 bit as the LSB, and therefore is preferred over 38.



Anonymous Swan 9mth #467a

✓ Resolved

(a) (1.0 pt) Which of the following representation systems have two representations for the decimal number zero?

- Bias notation
- Signed-magnitude
- Floating Point
- Two's complement
- Unsigned

SP21-MT-Q1a

why does floating point have two zeros? does it come from the norm and denorm or just the different sign bit?



Andrew Liu STAFF 9mth #467b

A zero in FP is defined as 0 exp and 0 mantissa. Note that this does not constrain the sign bit, so we have +0 and -0 in FP.





This comment was deleted



E **Eddy Byun** STAFF 9mth #467ad

For 1.6, I'd first recommend converting the hex number to binary

$0x85 = 0b\ 1000\ 0101$

The problem states that we stored this number as an unsigned one-byte integer.

To convert this to decimal, we do the following addition:

$$2^0 + 2^2 + 2^7 = 1 + 4 + 128 = 133$$

♡ ...