CS 61C
Spring 2020

Great Ideas in Computer Architecture

Final Exam

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `cs61c@berkeley.edu`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

◯ You must choose either this option

◯ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

☐ You could select this choice.

☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

Please complete and submit these questions before the exam starts.

(a) What is your full name?

Solutions

(b) What is your student ID number?

dQw4w9WgXcQ (This is a YouTube video)

(c) Some of the questions may use images to describe a problem. If the image is too small, you can click and drag the image to a new tab to see the full image. You can also right click the image and download it or copy the address of it to view it better. You can try that with the image below. You can also click the star by the question if you would like to go back to it (it will show up on the side bar). In addition you are able see a check mark for questions you have fully entered in the sidebar. Questions will auto submit about 5 seconds after you click off of them though we still recommend you click the save button.

Good luck, and don't F@#)(it up

1. **Bitz are Bitz!**

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

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

i. (0.5 pt) A[0]

3

ii. (0.5 pt) A[1]

0

iii. (0.5 pt) A[2]

0

iv. (0.5 pt) A[3]

250 (0xFA in decimal)

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

$$-1.0000000000000000000000011 * 2^{58}$$

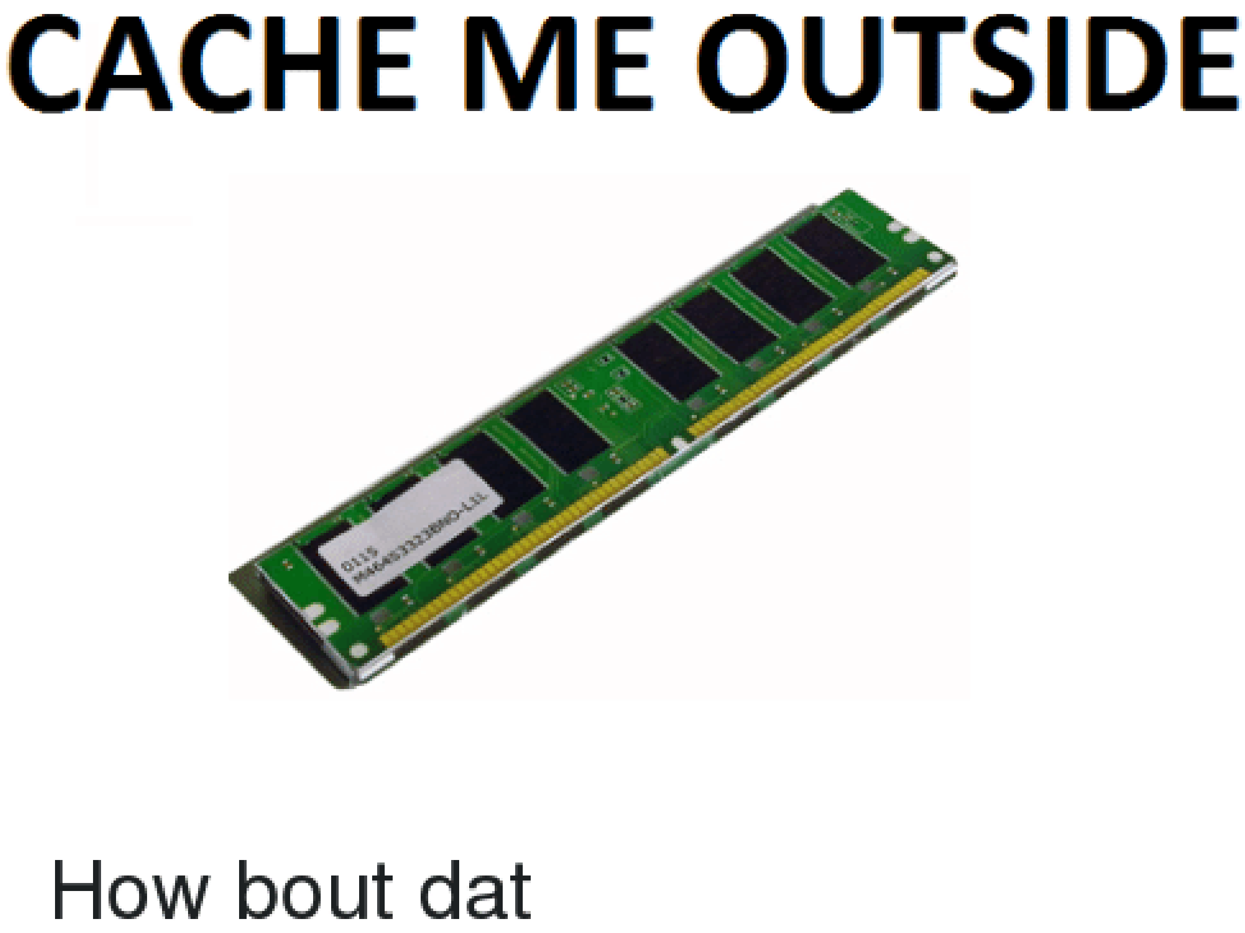
(c) **(2.0 pt)** a RISC-V instruction? If there's an immediate, write it in decimal. If it is an invalid instruction, write INVALID INSTRUCTION (in all caps).

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

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

250 (0xFA in decimal)

2. Cache Me Outsize



Given the following looping workload over an array where `N` is a large power of 2. The cache starts out empty, and the `process()` function doesn't introduce any significant cache pressure (so you can discount any hits or misses in the `process()` function)

```
uint32_t arr[N];

for (int j=0; j < 30; j++) {
    for (int i=0; i < N; i += 1) {
        process(arr[i]);
    }
}
```

Express the following answers as a function of `N`. **If you have a fraction, please fully simplify it.** If you believe that an answer is of the form `42 * N`, DO NOT include the multiply. You should format that answer like `42N` (Also in that exact order). Failure to do so or not capitalizing N will result is no points! If you have a fraction answer of the form `1 / 42 * N`, format it like `(1/42)N`. Note if it is NOT a fraction, you MUST not include the parentheses.

(a) Suppose we have a LRU fully associative cache of size 4N B and a block size of 4B:

i. (2.0 pt) Number of hits:

29N

ii. (2.0 pt) Number of misses:

N

iii. (2.0 pt) What type of locality is this cache taking advantage of (select all that apply)

- ☐ Quasi-balistic
- [✓ Temporal]
- ☐ None
- ☐ Spatial

iv. (2.0 pt) Does your answer change if the cache is 2 way set-associative? (Note: The cache size is still the same)

- ○ Yes
- [✓ No]

(b) Suppose we have a LRU fully associative cache of size 2N B and a block size of 4B:

i. (2.0 pt) Number of hits:

0

ii. (2.0 pt) Number of misses:

30N

iii. (2.0 pt) What type of locality is this cache taking advantage of (select all that apply)

- [✓ None]
- ☐ Temporal
- ☐ Quasi-balistic
- ☐ Spatial

iv. (2.0 pt) Does your answer change if the cache is 2 way set-associative? (Note: The cache size is still the same)

- [✓ No]
- ○ Yes

(c) Suppose we have a LRU fully associative cache of size 2N B with a block size of of 8B:

i. **(2.0 pt)** Number of hits:

15N

ii. **(2.0 pt)** Number of misses:

15N

iii. **(2.0 pt)** What type of locality is this cache taking advantage of (select all that apply)

☐ Temporal

☐ None

[✓ Spatial]

☐ Quasi-balistic

iv. **(2.0 pt)** Does your answer change if the cache is 2 way set-associative? (Note: The cache size is still the same)

○ Yes

[✓ No]

3. Nick's Parallelism Compute (uh huh) Setup

Nick has several resources at his disposal. The first is a 12 core AMD Ryzen processor running at over 3 GHz for MIMD computation, the second is a massive SIMD computational engine in the form of a high-end graphics card with 10 teraflops of floating point computation (he got it for compute... Uh hu.), and the final is a large map/reduce cluster on campus he has access to. If a problem requires reading as many memory locations as compute operations, mark it as "memory bound" because the speedups from parallelism are going to be minor because it will be limited by the memory subsystem.

Please select a parallelism technique which would benefit the problem the most.

(a) **(2.0 pt)** 32b floating point matrix multiply of 100M entry matrixes with a transposed matrix

- ○ Map/Reduce
- ○ Memory Bound
- ○ None (sequential)
- ○ MIMD parallelism
- [✓ SIMD parallelism]

(b) **(2.0 pt)** Find all references to himself in a downloaded corpus of every Internet post ever made (some 5 PB of data)

- [✓ Map/Reduce]
- ○ Memory Bound
- ○ MIMD parallelism
- ○ None (sequential)
- ○ SIMD parallelism

(c) **(2.0 pt)** Run a program he's written that has 40 threads that communicate through queues or channels

- ○ Memory Bound
- ○ None (sequential)
- ○ SIMD parallelism
- [✓ MIMD parallelism]
- ○ Map/Reduce

(d) **(2.0 pt)** 32b floating point matrix addition of 100M entry matrixes

- [✓ Memory Bound]
- ○ SIMD parallelism
- ○ None (sequential)
- ○ Map/Reduce
- ○ MIMD parallelism

4. Virtual Reality! I Mean Memory...

Consider a system with 2 MiB of physical memory and 4 GiB of virtual memory. Page size is 4 KiB. Recall that the single level page table is stored in physical memory and consists of PTE's, or page table entries.

(a) (3.0 pt) If we choose to store seven information bits in each PTE, how big is the page table in bytes?

2^{21}
VPN contains $log(\frac{2^{32}}{2^{12}}) = 20$ **bits**
PPN contains $log(\frac{2^{21}}{2^{12}}) = 9$ **bits**
9 bit PPN + 7 info bits = 16 bits per PTE
2^4 **bit PTE *** 2^{20} **PTE's =** 2^{24} **bit page table, or** 2^{21} **bytes**

(b) (3.0 pt) The page table starts off empty, then we make the following accesses: `0x00111999`, `0x00234567`, `0x00555FFF`. If the page table begins at address `0x20000000`, at what address can we find the PTE for the first access? (Your answer should be in hex)

`0x20000222`**.** `0x00111999` **has a VPN of** `0x00111`**. Each PTE is 2 bytes, so** `0x00111` *** 2 =** `0x00222`**.** `0x20000000` **+** `0x00222` **=** `0x20000222`**.**

(c) (2.0 pt) We have a fully associative TLB that also started empty but now contains the three entries from the accesses above. If we access `0x00556000` now, will we get a TLB hit, page hit, or page fault?

○ TLB Hit

○ Page Hit

○ None of the other answers

[✓ Page Fault]

Page Fault. `0x00556000` is a new page

5. Mover your A**

The (not turing complete) programming language Mover is defined as follows:

The program stores an 2-D grid of 8-bit integers, initialized to 0, a memory pointer, which starts at (x,y) = (0,0), and a program flow, which starts at “FORWARD”. The program recognizes only the following 8 commands:

- `>` Moves the pointer one step right (+1 to `x`)
- `<` Moves the pointer one step left (-1 to `x`)
- `^` Moves the pointer one step up (+1 to `y`)
- `v` Moves the pointer one step down (-1 to `y`)
- `+` Increments the value at the pointer by 1
- `-` Decrements the value at the pointer by 1
- `]` If the pointer is currently pointing at a 0 and the current program flow is “FORWARD”, change the program flow to “BACKWARD”. Otherwise do nothing.
- `[` If the pointer is currently pointing at a 0 and the current program flow is “BACKWARD”, change the program flow to “FORWARD”. Otherwise do nothing.

If the program flow is “FORWARD”, then the next instruction to be executed is the one after the current one; if the program flow is “BACKWARD”, then the next instruction is the one before the current instruction.

It is undefined behavior for the pointer to go outside the memory array’s bounds and likewise the behavior for integer overflow and underflow are undefined. For the C version, the program halts if it reaches the end of the program string in either direction.

The language ignores any other characters in the program, and terminates if the program counter goes past the bounds of the program.

(a) You want to write a C program that interprets this language: The inputs are a valid mover program as a null terminated string, and `memory_grid`, which points to a sequence of pointers, each of which points to a buffer that is a column of the 2D grid.

```
void runMover(char* program, int8_t** memory_grid)
{
   uint x = 0;
   uint y = 0;
   uint pc = 0;
   uint dir = 1; /* forward = true */
   uint len = strlen(program);
   while(pc>=0 && pc<len)
   {
      switch(program[pc])
      {
         //Code for each Mover command
      }
      pc += dir ? 1: -1;
   }
}
```

For the following operators, write the C code for the following Mover commands.

i. (1.0 pt)

Command: ^

Code:

```
case '^':
   <YOUR CODE HERE>
   break;
```

`y += 1;` **(or equivalent)**

ii. (2.0 pt)

Command: +

Code:

```
case '+':
   <YOUR CODE HERE>
   break;
```

`memory_grid[x][y] += 1;` **(or equivalent)**

iii. (2.0 pt)

Command:]

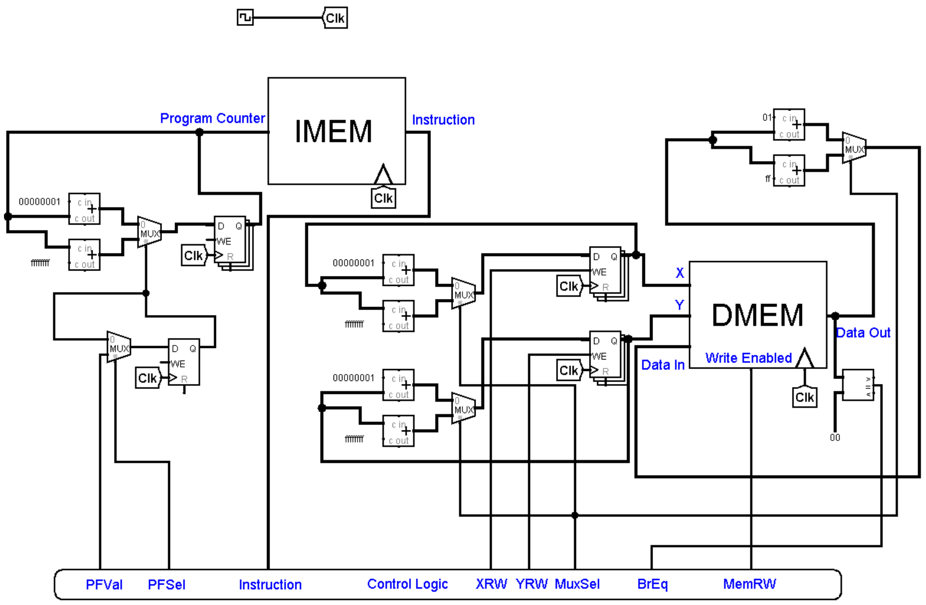
Code:

```
case ']':
   <YOUR CODE HERE>
   break;
```

iv. You now want to create a circuit that runs Mover. In order to do that, you assign each Mover command a unique 4-bit code (explanations of each command have been copied for reference):

- `(0001) >` Moves the pointer one step right (+1 to `x`)
- `(1001) <` Moves the pointer one step left (-1 to `x`)
- `(0101) ^` Moves the pointer one step up (+1 to `y`)
- `(1101) v` Moves the pointer one step down (-1 to `y`)
- `(0011) +` Increments the value at the pointer by 1
- `(1011) -` Decrements the value at the pointer by 1
- `(0111) ]` If the pointer is currently pointing at a 0 and the current program flow is "FORWARD", change the program flow to "BACKWARD". Otherwise do nothing.
- `(1111) [` If the pointer is currently pointing at a 0 and the current program flow is "BACKWARD", change the program flow to "FORWARD". Otherwise do nothing.

You have finished the general structure of the circuit, and just need to finish writing the control logic. The memories in question are asynchronous read but synchronous write (thus the CLK). During startup the registers are set to 0. Note that the direction register now holds zero (not one) when the program is going forward. We just run forever and it is simply undefined behavior to either overflow or underflow the PC so we aren't worrying about having to check for any of that:



Singe Cycle Mover

Let x3, x2, x1, x0 be the bits of an instruction with x0 being the least significant bit, and let BrEq be the value of BrEq. Write the most simplified logical equations for each output of Control Logic.

Please use C syntax when writing out your formulas.

A. (2.0 pt) YRW

`~x1 & x2` **(or equivalent)**

B. (2.0 pt) XRW

`~(x1 | x2)` **(or equivalent)**

C. (2.0 pt) PFVal

`~x3` **(or the equivalent** `!x3`**)**

D. (2.0 pt) PFSel

`x1 & x2 & BrEq` **(or equivalent)**

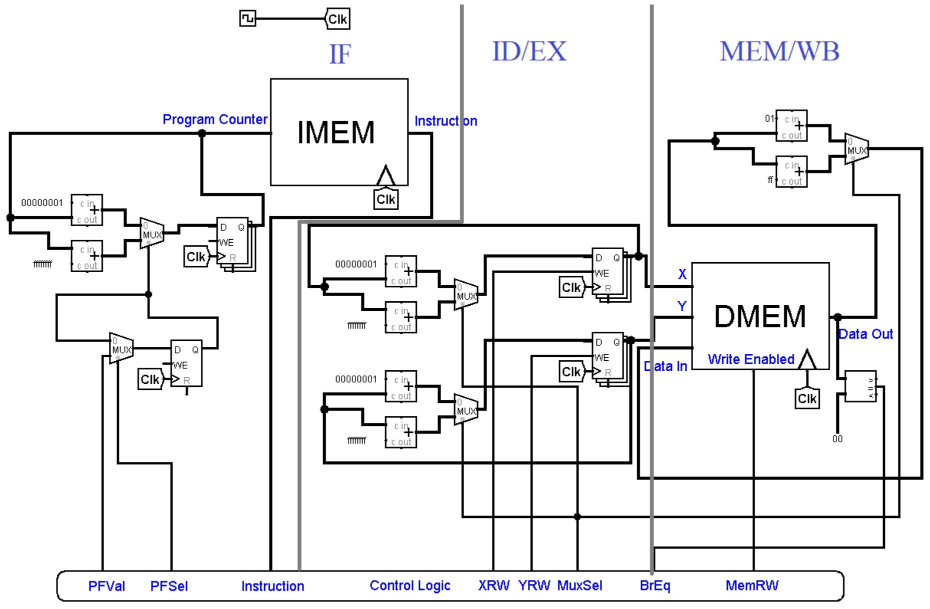
E. (2.0 pt) MemRW

`x1 & ~x2` **(or equivalent)**

F. (2.0 pt) MuxSel

`x3`

v. After finishing your circuit, you find that it's a bit slow. To speed things up, you decide to pipeline the above circuit by dividing the circuit into IF, ID/EX, and MEM/WB stages.



Pipelined Mover

A. (3.0 pt) What hazards can occur? Assume that the single cycle datapath worked as intended (Select all that apply)

■ Control Hazard

☐ Structural Hazard

☐ Data Hazard

☐ None of the Other Choices

Control Hazard: Yes. If our instruction is `[` or `]`, then we need to wait until after MEM to decide whether to reverse flow. If we pipeline, we would need to stall for `[` or `]` instructions.

Data Hazard: No. Since our writeback doesn't affect `X` or `Y`, we can never have a data dependency.

None of the Other Choices: No. While data and structural hazards cannot occur, control hazards can.

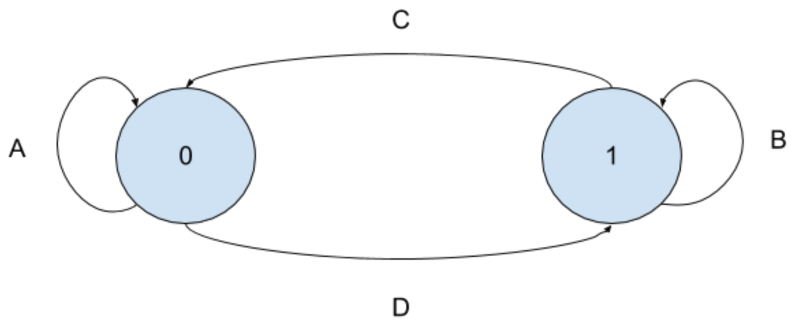
Structural Hazard: No. In the current system, register reads and writes happen during the same cycle.

B. (3.0 pt) Please leave your answer in ns and do not add the units.

If all registers (including pipeline registers) have 2ns setup, 0ns hold, and 2ns clk->q time, the memories take 4ns to do a read and have a 2ns setup time for writes, and the sequential logic takes 0ns (yes, that's a ridiculous number, we chose it to make the path simple), what is the minimum viable clock period for the resulting datapath?

8. **2ns (clk->q time) + 4ns (IMEM read) + 2ns (setup time) = 8ns**

C. Of course, **direction** tracking can be also implemented as a finite state machine with two states (1 = forward, 0 = backward) and 3 inputs: "CF" which is 1 when the current instruction is "[", "CB" which is 1 when the current instruction is "]", and "Mem" which is 1 if the current memory value is non-zero. For the four transitions on the below state transition diagram, Write the most simplified



logical equations for each edge.

D. (1.0 pt) A

`!(CF & !Mem) = !CF | Mem`

E. (1.0 pt) B

`!(CB & !Mem) = (!CB | Mem)`

F. (1.0 pt) C

`CB & !Mem`

G. (1.0 pt) D

`CF & !Mem`

(b) I Forget Where This Data Goes

For each question, select the option which best describes what an operation would evaluate to. If the operation would lead to something which is not a valid address, select "Not an Address". Consider this snippet of a C program:

```
void foo() {
    int64_t w = 4;
    int64_t* u = malloc(100 * sizeof(w));
    int64_t** v = &u;
    ...
}
```

i. (2.0 pt) What does `sizeof(*u)` evaluate to on a 32-bit system?

8 **When we dereference u, we are now asking for the size of an int64_t. This is 8 Bytes as that is the size of this type (64 bits / 8 bits = 8 Bytes)**

ii. (2.0 pt) What does `sizeof(u)` evaluate to on a 32-bit system?

4 **This is because the type of u is a pointer to an int64_t meaning we are asking for the size of the pointer.**

iii. (2.0 pt) What type of value does `v` evaluate to?

○ Heap Address

○ Static Address

[✓ Stack Address]

○ Not an Address

iv. (2.0 pt) What type of value does `*v` evaluate to?

○ Not an Address

○ Static Address

[✓ Heap Address]

○ Stack Address

v. (2.0 pt) What type of value does `(u + 1)` evaluate to?

○ Static Address

○ Not an Address

○ Stack Address

[✓ Heap Address]

vi. (2.0 pt) What type of value does `*(u + 1)` evaluate to?

- [✓ Not an Address]
- ○ Stack Address
- ○ Static Address
- ○ Heap Address

vii. (2.0 pt) What type of value does `&w` evaluate to?

- ○ Not an Address
- ○ Heap Address
- ○ Static Address
- [✓ Stack Address]

(c) Bloomin Onion

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

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

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

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

16

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

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

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

`b->itercount`

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

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

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

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

iii. We also have the following function to allocate a new bloom filter

```
struct BloomFilter *alloc(
      uint64_t (*)(void *data, uint16_t iter) hash,
      uint32_t size,
      uint16_t itercount){
   struct BloomFilter *ret = malloc(64);
   /* Yes, this is way too big, but we don't want to give you
      the answer to the previous question!  */
   ret->size = size;
   ret->data = calloc(size >> 3, 1);
   ret->hash = hash;
   ret->itercount = itercount;
}
```

Complete the RISC-V translation necessary to allocate this: We will put `ret` in `s0`.

```
alloc: # Prolog
   (CODE INPUT 1)
   sw ra 0(sp)
   sw s0 4(sp)
   sw a0 8(sp)
   sw a1 12(sp)
   sw a2 16(sp)
# body
   addi (CODE INPUT 2)
   jal malloc
   mv s0 a0         # put ret in s0
   (CODE INPUT 3)   # load size into t0
   (CODE INPUT 4)   # store it
   (CODE INPUT 5)   # div size by 8 with a shift
   li a1 1
   jal calloc
   sw a0 12(s0)     # store data
   (CODE INPUT 6)   # load hash to t0
   (CODE INPUT 7)   # store it: Use the right type!
   (CODE INPUT 8)   # load itercount to t0
   (CODE INPUT 9)   # store it: Use the right type!
   mv a0 s0
# epilog
   lw ra 0(sp)
   (CODE INPUT 10)
   (CODE INPUT 11)
   jr ra
```

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

```
addi sp, sp, -20
```

B. (1.0 pt) (CODE INPUT 2):

```
a0, x0, 64
```

C. (1.0 pt) (CODE INPUT 3):

```
lw t0, 12(sp)
```

D. (1.0 pt) (CODE INPUT 4):

```
sw t0, 0(s0)
```

E. (1.0 pt) (CODE INPUT 5):

```
srli a0, t0, 3
```

F. (1.0 pt) (CODE INPUT 6):

```
lw t0, 8(sp)
```

G. (1.0 pt) (CODE INPUT 7):

```
sw t0, 8(s0)
```

H. (1.0 pt) (CODE INPUT 8):

```
lw t0, 16(sp)
```

I. (1.0 pt) (CODE INPUT 9):

```
sh t0, 4(s0)
```

J. (1.0 pt) (CODE INPUT 10):

```
lw s0, 4(sp)
```

K. (1.0 pt) (CODE INPUT 11):

```
addi sp, sp, 20
```

(d) CALL me maybe

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

A. (2.0 pt) PC-Relative Addressing

[✓ Never Relocate]

○ Always Relocate

B. (2.0 pt) Static Data Reference

○ Never Relocate

[✓ Always Relocate]

ii. Answer the following:

A. **(3.0 pt)** Select all the steps that are done during the Assembler phase of CALL

- [✓] Producing machine language
- ☐ Generating Assembly code
- ☐ Semantic Analysis
- ☐ Parsing the C code
- ☐ Outputting executable code
- ☐ Lexing the C code
- [✓] Pseudo-instruction replacement

B. **(3.0 pt)** Describe two benefits of using Dynamically Linked Libraries

There's more than two right answers to this question, but the most common ones we accepted were:

- **Saving resources (such as disk space or memory) by sharing data (they do not all need the same extra data)**
- **Easier upgrading as we only need to replace the DLL file (we do not need to recompile the code which used the DLL library)**
- **Multi-language programming: The DLL may be written in a different language as what you are using.**
- **System independence/standardized interface: A DLL may offer a standard interface for hardware of a machine which allows for an abstraction from the main program.**

C. **(2.0 pt)** What are assembler directives (explain what they are used for, don't just give examples of them)?

They give directions to the assembler, but do not produce machine instructions.

No more questions.