[Midterm] Past Exams - 2020 and Older #466

E Eric Che STAFF
9 months ago in Exam – Midterm

707
VIEWS

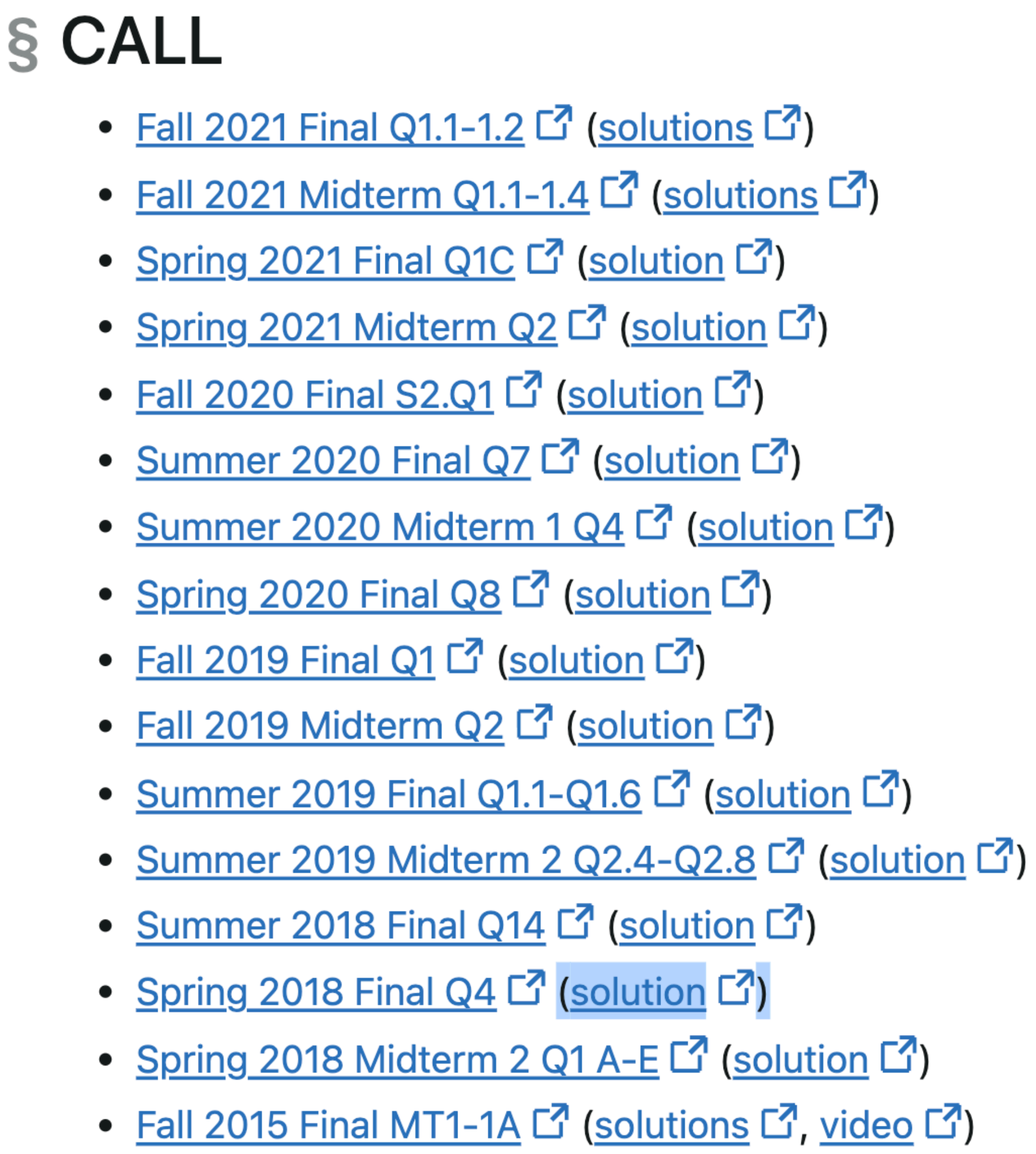
You can find the past exams here: https://cs61c.org/sp24/resources/exams/. Please check the linked past Piazza/Ed Q&A PDFs first before asking here. Many of the questions are already answered in those! Video walkthroughs are also available!

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**

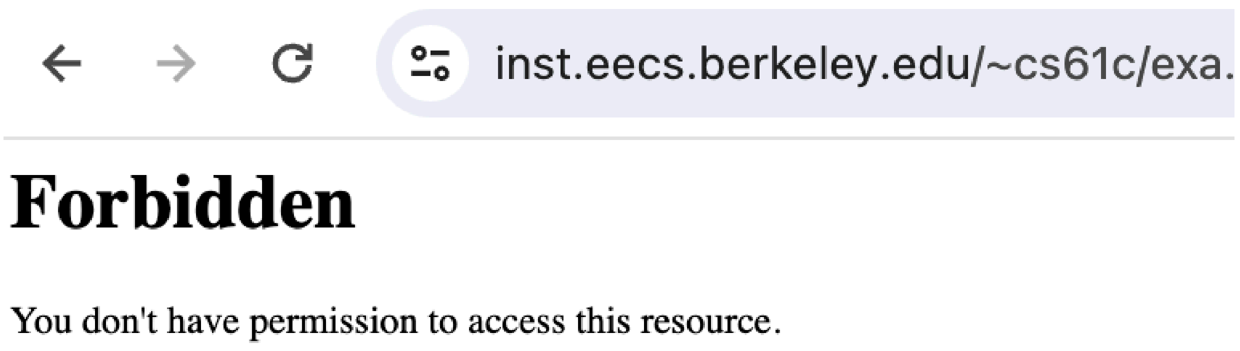
Anonymous Snake 8mth #466fd ✓ Resolved



§ CALL

- Fall 2021 Final Q1.1-1.2 (solutions)
- Fall 2021 Midterm Q1.1-1.4 (solutions)
- Spring 2021 Final Q1C (solution)
- Spring 2021 Midterm Q2 (solution)
- Fall 2020 Final S2.Q1 (solution)
- Summer 2020 Final Q7 (solution)
- Summer 2020 Midterm 1 Q4 (solution)
- Spring 2020 Final Q8 (solution)
- Fall 2019 Final Q1 (solution)
- Fall 2019 Midterm Q2 (solution)
- Summer 2019 Final Q1.1-Q1.6 (solution)
- Summer 2019 Midterm 2 Q2.4-Q2.8 (solution)
- Summer 2018 Final Q14 (solution)
- Spring 2018 Final Q4 (solution)
- Spring 2018 Midterm 2 Q1 A-E (solution)
- Fall 2015 Final MT1-1A (solutions, video)

The link to solution for sp28 final seems to be broken. Is there a way to access it?



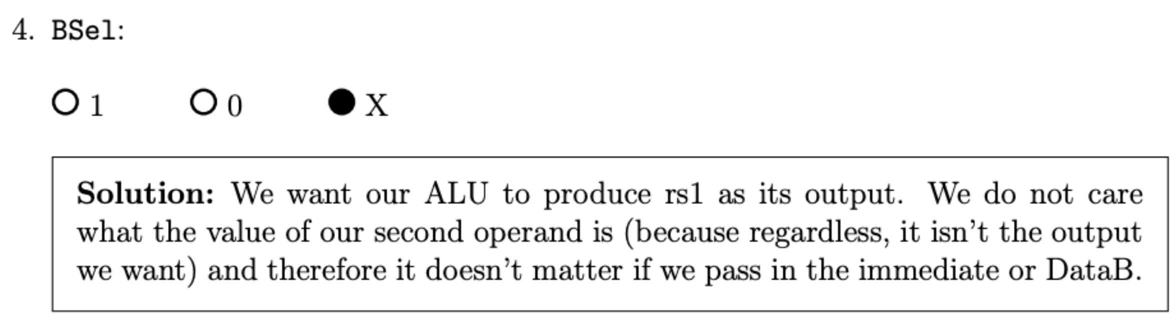
inst.eecs.berkeley.edu/~cs61c/exa

Forbidden

You don't have permission to access this resource.

Anonymous Deer 8mth #466fb ✓ Resolved

Sp18-mt2-q3


4. BSel:

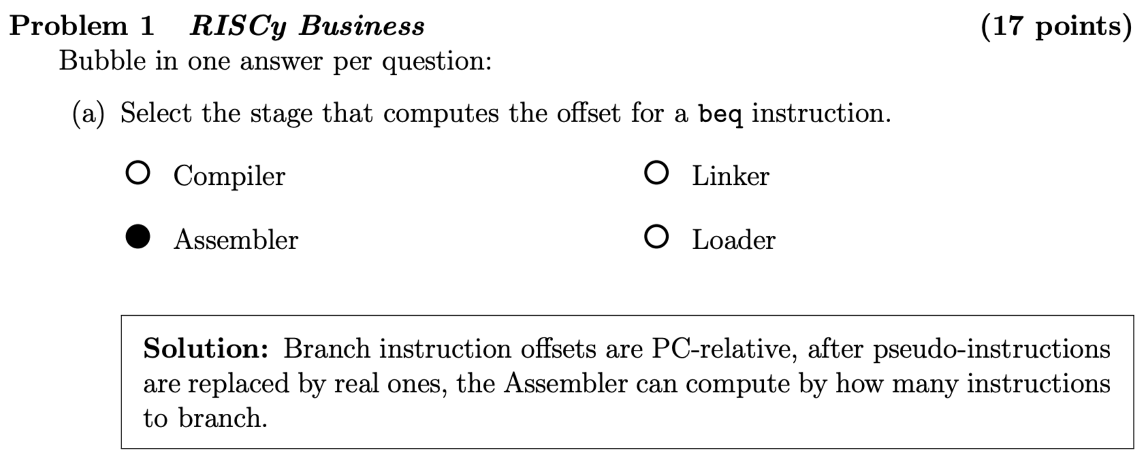
1 0 [✓ X]

Solution: We want our ALU to produce rs1 as its output. We do not care what the value of our second operand is (because regardless, it isn't the output we want) and therefore it doesn't matter if we pass in the immediate or DataB.

They said that ALUSel would be ADD. So shouldn't we add the imm (which is always 0 based on question) to rd[rs1]?

Anonymous Deer 8mth #466ef ✓ Resolved

Sp18-mt2


Problem 1 ***RISCy Business*** **(17 points)**

Bubble in one answer per question:

(a) Select the stage that computes the offset for a `beq` instruction.

Compiler [✓ Assembler] Linker Loader

Solution: Branch instruction offsets are PC-relative, after pseudo-instructions are replaced by real ones, the Assembler can compute by how many instructions to branch.

Could this also be linker? Can beq not jump to another function? If jal can do it why not beq

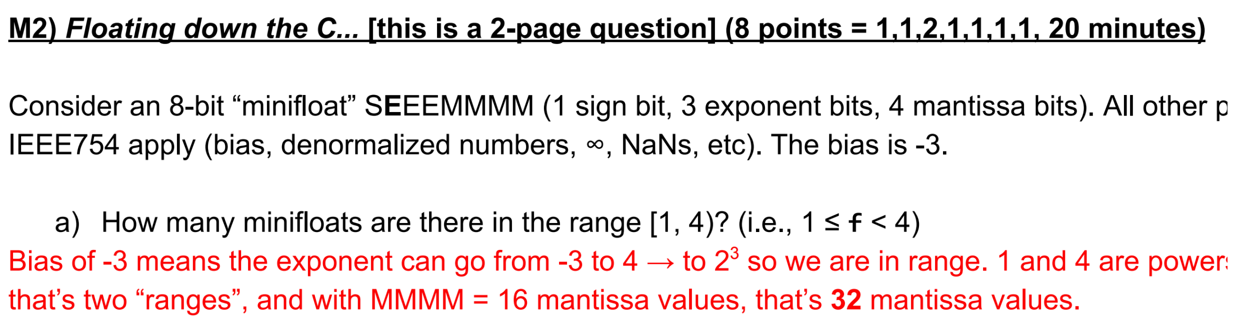
Justin Yokota STAFF 8mth #466fc

beq has a much smaller jump distance, since it only stores 12 bits of immediate instead of 20. As such, branches generally are not allowed to jump "outside" your current file; instead, you can do a branch, followed by a jump.

1

Anonymous Deer 8mth #466ee ✓ Resolved

Fa18-Final-Q2


M2) *Floating down the C...* [this is a 2-page question] (8 points = 1,1,2,1,1,1,1, 20 minutes)

Consider an 8-bit "minifloat" SEEEMMMM (1 sign bit, 3 exponent bits, 4 mantissa bits). All other p

IEEE754 apply (bias, denormalized numbers, ∞, NaNs, etc). The bias is -3.

a) How many minifloats are there in the range [1, 4)? (i.e., $1 \le f < 4$)

Bias of -3 means the exponent can go from -3 to 4 → to 2^3 so we are in range. 1 and 4 are power

that's two "ranges", and with MMMM = 16 mantissa values, that's **32** mantissa values.

How did they calculate this :(((((

A Andy Chen STAFF 8mth #466fa

It's similar to #466db!

1 in FP: 0 011 0000 --> $2^{(3-3)} * 1.0000_2 = 2^0$

4 in FP: 0 101 0000 --> $2^{(5-3)} * 1.0000_2 = 2^2$

Since we are excluding 4, there are 2 possible exponent values within the range:

Exponent = 0b011 (corresponding to 2^0)

Exponent = 0b100 (corresponding to 2^1)

Then, a floating point number within the range **must** look like one of the following:

0 011 MMMM

0 100 MMMM

The 4 mantissa bits (MMMM) can be anything, so there are 2^4 = 16 distinct numbers for each of the 2 exponent values, adding up to 32 (16 + 16) total distinct minifloats.

Anonymous Alligator 8mth #466eb ✓ Resolved

SP21-MT 5c -> is 5c out of scope since it relates to pipelining which isn't in scope (right?)

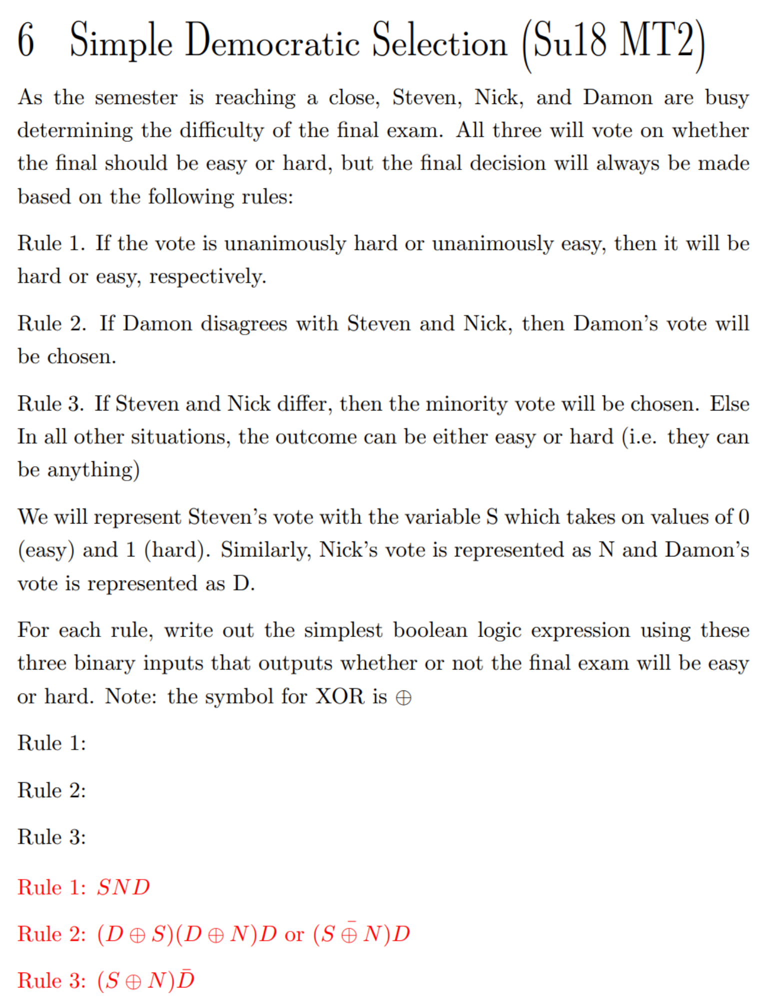
N Nikhil Kandkur STAFF 8mth #466ed

Yup pipelining is not in scope!

Anonymous Scorpion 8mth #466ea ✓ Resolved

SU18-MT-Q6.1

Can Rule 2 be D(NOT S)(NOT N)? Since for possibilities D = 1, S, N = 0 and D = 0, S, N = 1 the resulting value is 1 and 0 respectively.



6 Simple Democratic Selection (Su18 MT2)

As the semester is reaching a close, Steven, Nick, and Damon are busy determining the difficulty of the final exam. All three will vote on whether the final should be easy or hard, but the final decision will always be made based on the following rules:

Rule 1. If the vote is unanimously hard or unanimously easy, then it will be hard or easy, respectively.

Rule 2. If Damon disagrees with Steven and Nick, then Damon's vote will be chosen.

Rule 3. If Steven and Nick differ, then the minority vote will be chosen. Else In all other situations, the outcome can be either easy or hard (i.e. they can be anything)

We will represent Steven's vote with the variable S which takes on values of 0 (easy) and 1 (hard). Similarly, Nick's vote is represented as N and Damon's vote is represented as D.

For each rule, write out the simplest boolean logic expression using these three binary inputs that outputs whether or not the final exam will be easy or hard. Note: the symbol for XOR is $\oplus$

Rule 1:

Rule 2:

Rule 3:

Rule 1: SND

Rule 2: $(D \oplus S)(D \oplus N)D$ or $(S \overline{\oplus} N)D$

Rule 3: $(S \oplus N)\bar{D}$

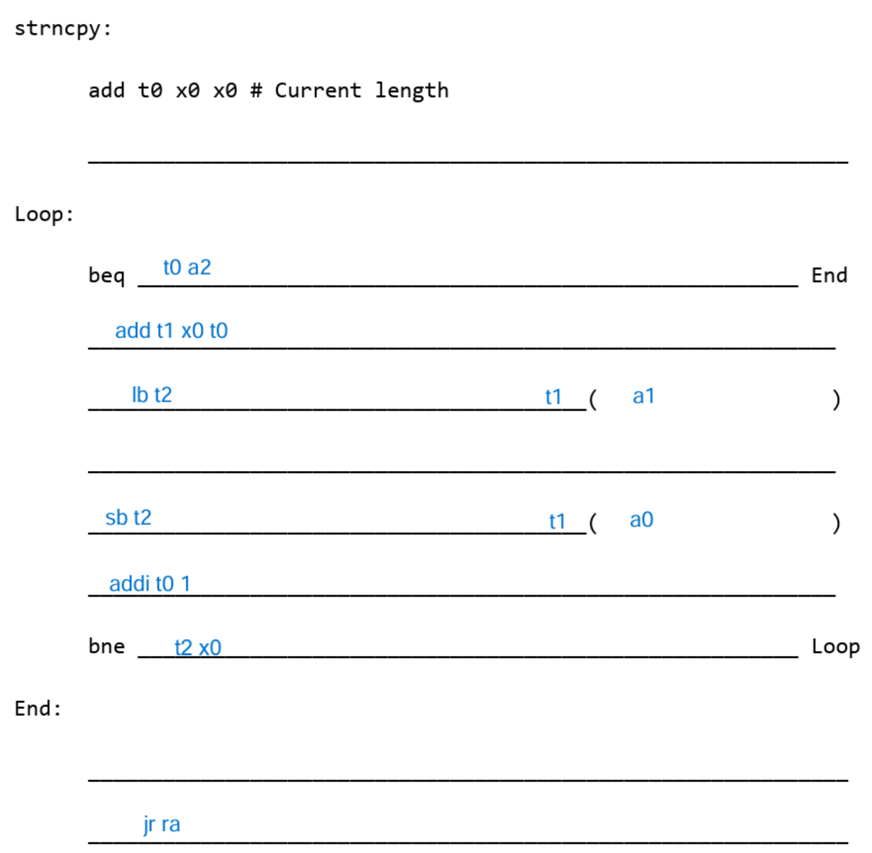
Justin Yokota STAFF 8mth #466fe

It depends on what the expected behavior of the question is for other values (e.g. if it's intended to be "don't care"s, but if you just need those values, I think it's valid.

Anonymous Sand Dollar 8mth #466df ✓ Resolved

Su19_MT1_Q5

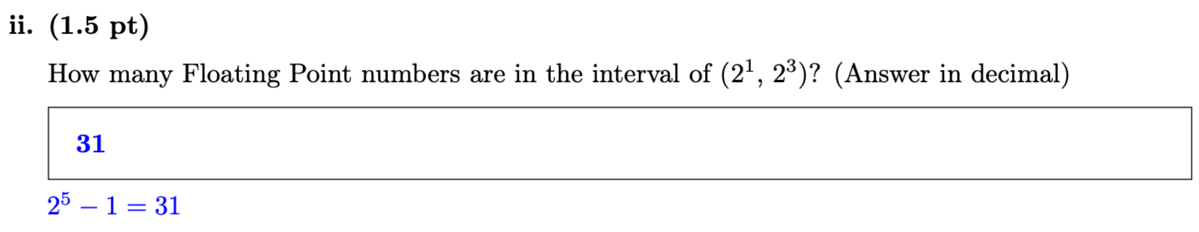
I was wondering if this solution was valid, like can I use a temporary register as the immediate for lb and sb instructions? Or does the immediate have to be a decimal?



```
strncpy:
        add t0 x0 x0 # Current length
        ____________________________________________
Loop:
        beq  t0 a2 _________________________________ End
        add t1 x0 t0 _______________________________
        lb t2 ___________________________ t1 ( a1          )
        ____________________________________________
        sb t2 ___________________________ t1 ( a0          )
        addi t0 1 __________________________________
        bne  t2 x0 _________________________________ Loop
End:
        ____________________________________________
        jr ra ______________________________________
```

Anonymous Deer 9mth #466ce ✓ Resolved

SU20-MT1-Q6



ii. (1.5 pt)

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

31

$2^5 - 1 = 31$

How did they get this?

Lisa Yan STAFF 8mth #466db

8-bit floating point: 1 sign bit, 3 exponent bits (bias -3), and 4 significand bits

```
2^1: +1.0000 x 2^(4-3)  --> 0 100 0000
2^3: +1.0000 x 2^(7-3)  --> 0 111 0000
values within range:        0 10Y XXXX
```

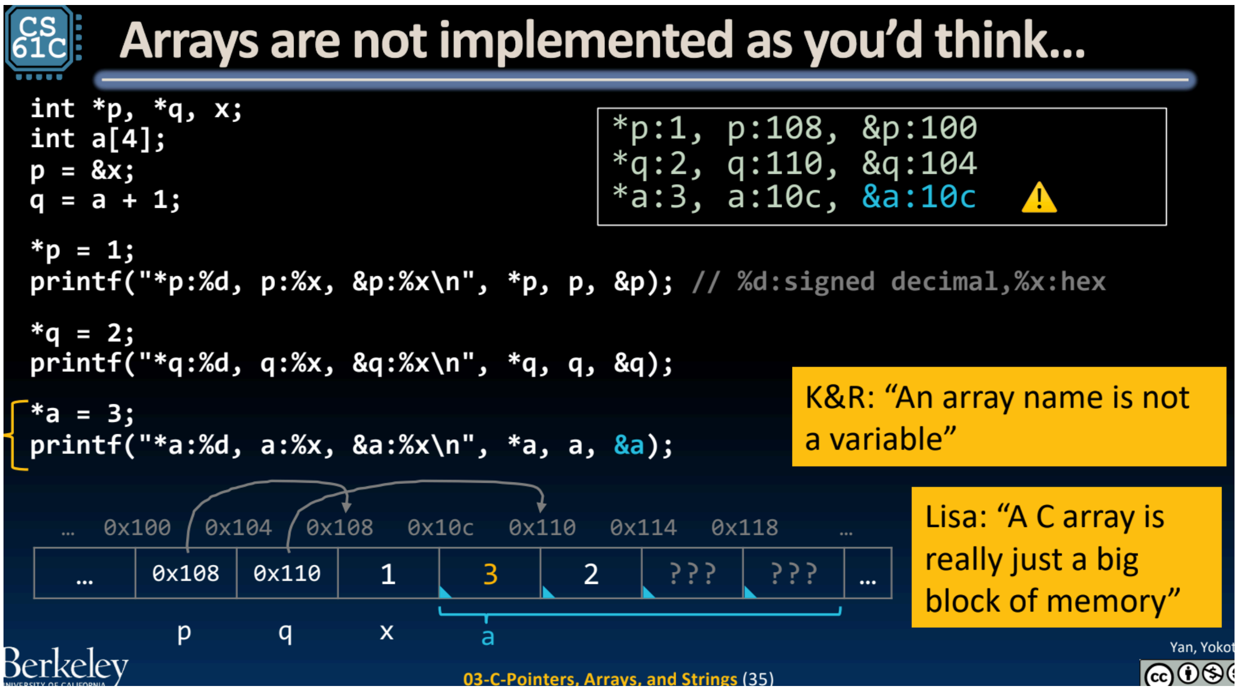
Assuming Y, X's are all 0's or 1s, then there are 2^1 * 2^4 - 1 values in this range (*excluding* the ranges), where we discount the lower bound, i.e., `Y XXXX=0 0000` .

Anonymous Red deer 9mth #466cd ✓ Resolved

SP20-Final-Q1d

Why is the printed result 0xFA instead of GARBAGE?

From what I understand, we are assigning 0xFA000003 to pointer c, so c=0xFA000003 but we didn't know its address?



Arrays are not implemented as you'd think...

```
int *p, *q, x;
int a[4];
p = &x;
q = a + 1;

*p = 1;
printf("*p:%d, p:%x, &p:%x\n", *p, p, &p); // %d:signed decimal,%x:hex

*q = 2;
printf("*q:%d, q:%x, &q:%x\n", *q, q, &q);

*a = 3;
printf("*a:%d, a:%x, &a:%x\n", *a, a, &a);
```

```
*p:1, p:108, &p:100
*q:2, q:110, &q:104
*a:3, a:10c, &a:10c ⚠
```

K&R: "An array name is not a variable"

Lisa: "A C array is really just a big block of memory"

… 0x100 0x104 0x108 0x10c 0x110 0x114 0x118 …
… 0x108 0x110 1 3 2 ??? ??? …
p q x a

Berkeley UNIVERSITY OF CALIFORNIA

03-C-Pointers, Arrays, and Strings (35)

Yan, Yokot

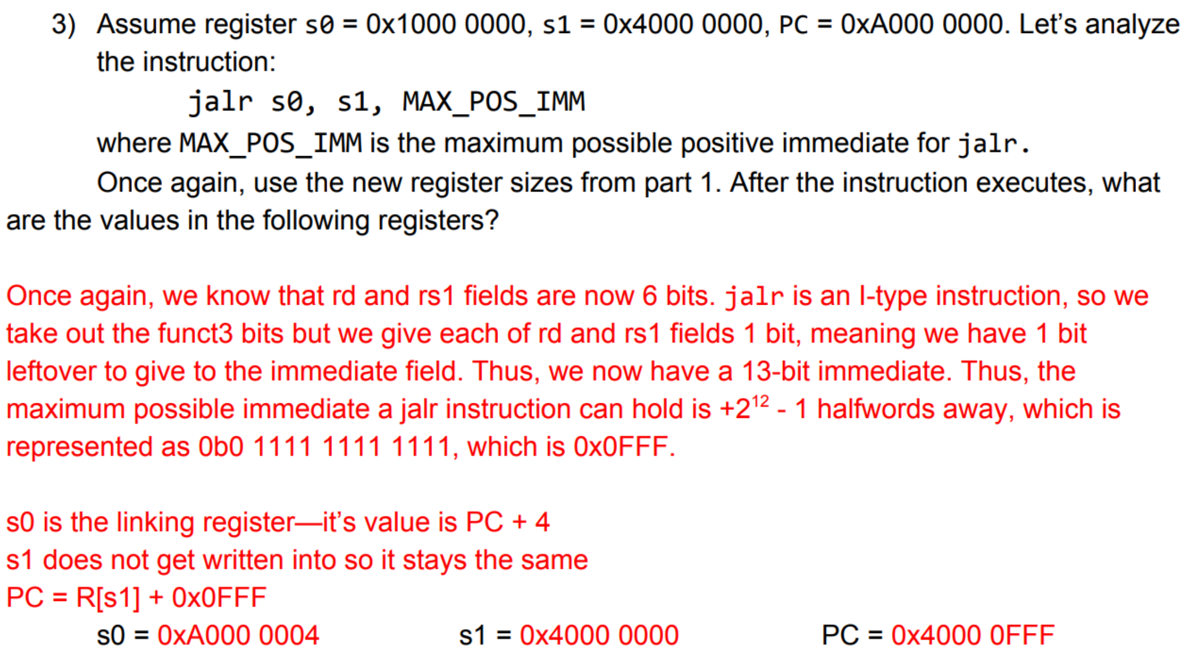
Lisa Yan STAFF 8mth #466dc

Addressed in the rewritten solutions; note that we get the address with `&` but then we dereference with array element access `[]`:
https://inst.eecs.berkeley.edu/%7Ecs61c/exams/pdfs/sp20-final-sols-rewritten.pdf

Anonymous Red deer 9mth #466cc ✓ Resolved

SU18-MT1-Q5.3:

For this problem, why do we consider half word instead of a word?



3) Assume register `s0` = 0x1000 0000, `s1` = 0x4000 0000, `PC` = 0xA000 0000. Let's analyze the instruction:

```
jalr s0, s1, MAX_POS_IMM
```

where MAX_POS_IMM is the maximum possible positive immediate for `jalr`.

Once again, use the new register sizes from part 1. After the instruction executes, what are the values in the following registers?

Once again, we know that rd and rs1 fields are now 6 bits. `jalr` is an I-type instruction, so we take out the funct3 bits but we give each of rd and rs1 fields 1 bit, meaning we have 1 bit leftover to give to the immediate field. Thus, we now have a 13-bit immediate. Thus, the maximum possible immediate a jalr instruction can hold is $+2^{12} - 1$ halfwords away, which is represented as 0b0 1111 1111 1111, which is 0x0FFF.

s0 is the linking register—it's value is PC + 4
s1 does not get written into so it stays the same
PC = R[s1] + 0x0FFF

s0 = 0xA000 0004 s1 = 0x4000 0000 PC = 0x4000 0FFF

Lisa Yan STAFF 8mth #466dd

jumps (`beq/etc.` or `jal/jalr`) use offsets that represent halfwords. We tried to avoid this terminology in this semester, but effectively the question computes the `jalr` offset for

`MAX_POS_IMM = 0b0 1111 1111 1111`. This corresponds to jumping `0b01 1111 1111 1110` bytes, which is equivalent to 2^12 - 1 halfwords.

Given our 32-bit size instructions in this course, we would never realistically jump halfwords, but some compressed RISC-V architectures support this.

Anonymous Wolverine 9mth #466cb ✓ Resolved

SU20-Final-Q7.d --> How come we can understand the following answers and specifically why they end up in different sections of jie.e file but in same segment?

7. CALL

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

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

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

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

(a) (2.0 pt) `sean`, `stephan`, `jenny`, `segfault`, and `philspel` all show up in the relocation table after the first pass through. Which of the following must be true? Select all that apply.

- [✓] They are referenced before they are defined.
- ☐ They belong in the `.text` segment.
- ☐ They are external references.
- ☐ None of the other options
- ☐ They are referenced before `poggers` and `crossroads`.

(b) (2.0 pt) After the first pass through, `poggers` and `crossroads` don't show up in the relocation table. What does this imply about the two function labels? Select all that apply.

- ☐ They are `.globals`.
- ☐ None of the other options
- ☐ They are both referenced before they are defined.
- [✓] After the assembler is finished, they are in the same segment.

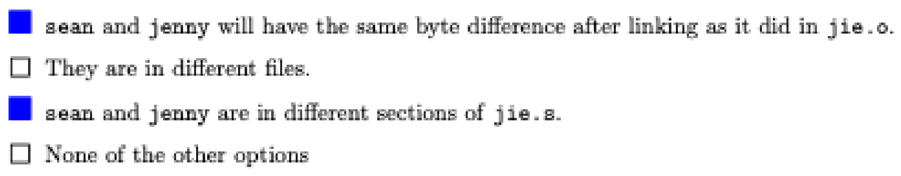
(c) (2.0 pt) After the second pass by the assembler, we see that `philspel` is no longer in the relocation table. Which of the following is true about `philspel`? Select all that apply.

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

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

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

- [✓] They are in the same segment.



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

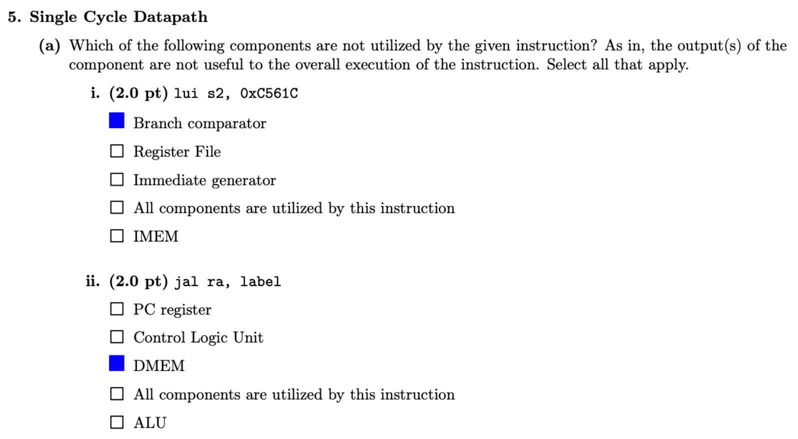
☐ They are in different files.

■ sean and jenny are in different sections of `jie.s`.

☐ None of the other options

Anonymous Red deer 9mth #466ca ✓ Resolved

SU20-Final-Q5a:



5. Single Cycle Datapath

(a) Which of the following components are not utilized by the given instruction? As in, the output(s) of the component are not useful to the overall execution of the instruction. Select all that apply.

i. **(2.0 pt)** `lui s2, 0xC561C`

■ Branch comparator

☐ Register File

☐ Immediate generator

☐ All components are utilized by this instruction

☐ IMEM

ii. **(2.0 pt)** `jal ra, label`

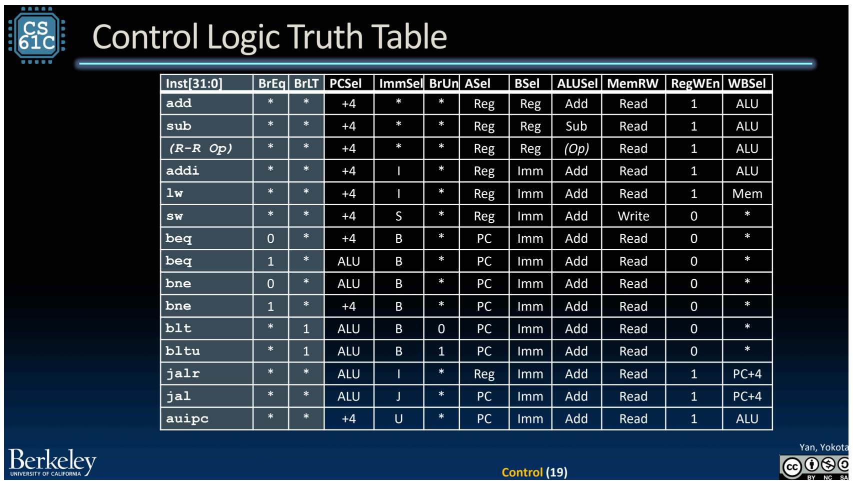
☐ PC register

☐ Control Logic Unit

■ DMEM

☐ All components are utilized by this instruction

☐ ALU



Control Logic Truth Table

Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
`add`	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
`sub`	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	*(Op)*	Read	1	ALU
`addi`	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
`lw`	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
`sw`	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
`beq`	0	*	+4	B	*	PC	Imm	Add	Read	0	*
`beq`	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
`bne`	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
`bne`	1	*	+4	B	*	PC	Imm	Add	Read	0	*
`blt`	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
`bltu`	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
`jalr`	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
`jal`	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
`auipc`	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

Yan, Yokota

Control (19)

For (ii), isn't the MemRW signal passed in to the DMEM?

My guess for the reasoning is that even though DMEM perform the READ operation, the result is not used at the end? But if this is the case, couldn't we also argue that DMEM is being used since a READ operation is performed?

Lisa Yan STAFF 8mth #466de

As per the question prompt: "As in, the output(s) of the component are not useful to the overall execution of the instruction." Every instruction will necessarily perform a memory access (Write if `sw`, and Read in every other case), because signals will flow through the datapath. But for `jalr` we don't care about what is read from memory.

Anonymous Red deer 9mth #466bb ✓ Resolved

SP18 MT1 Problem 3:

For (aii), why is the address of song1->title evaluate to static address?

For (aiv), why is &song2 evaluate to stack address?

Is there a rule of deciding the type of address a value evaluate to?

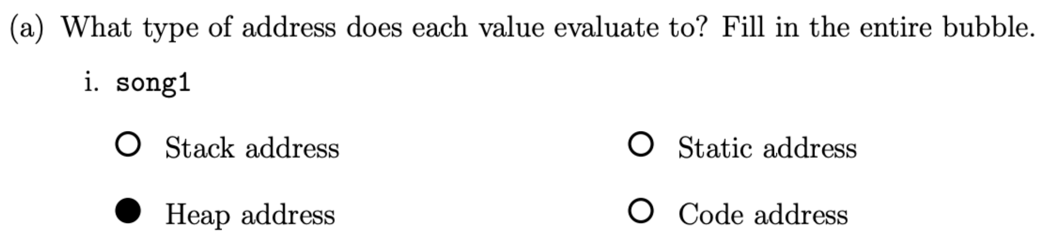
Andrew Liu STAFF 9mth #466bc

(a) ii. `song1->title` points to a string literal, which is on static.

(a) iv. `song2` is a local variable, so its address is on the stack.

The lecture slides for C memory layout describe what is found in each segment.

Anonymous Red deer 8mth #466cf



(a) What type of address does each value evaluate to? Fill in the entire bubble.

i. `song1`

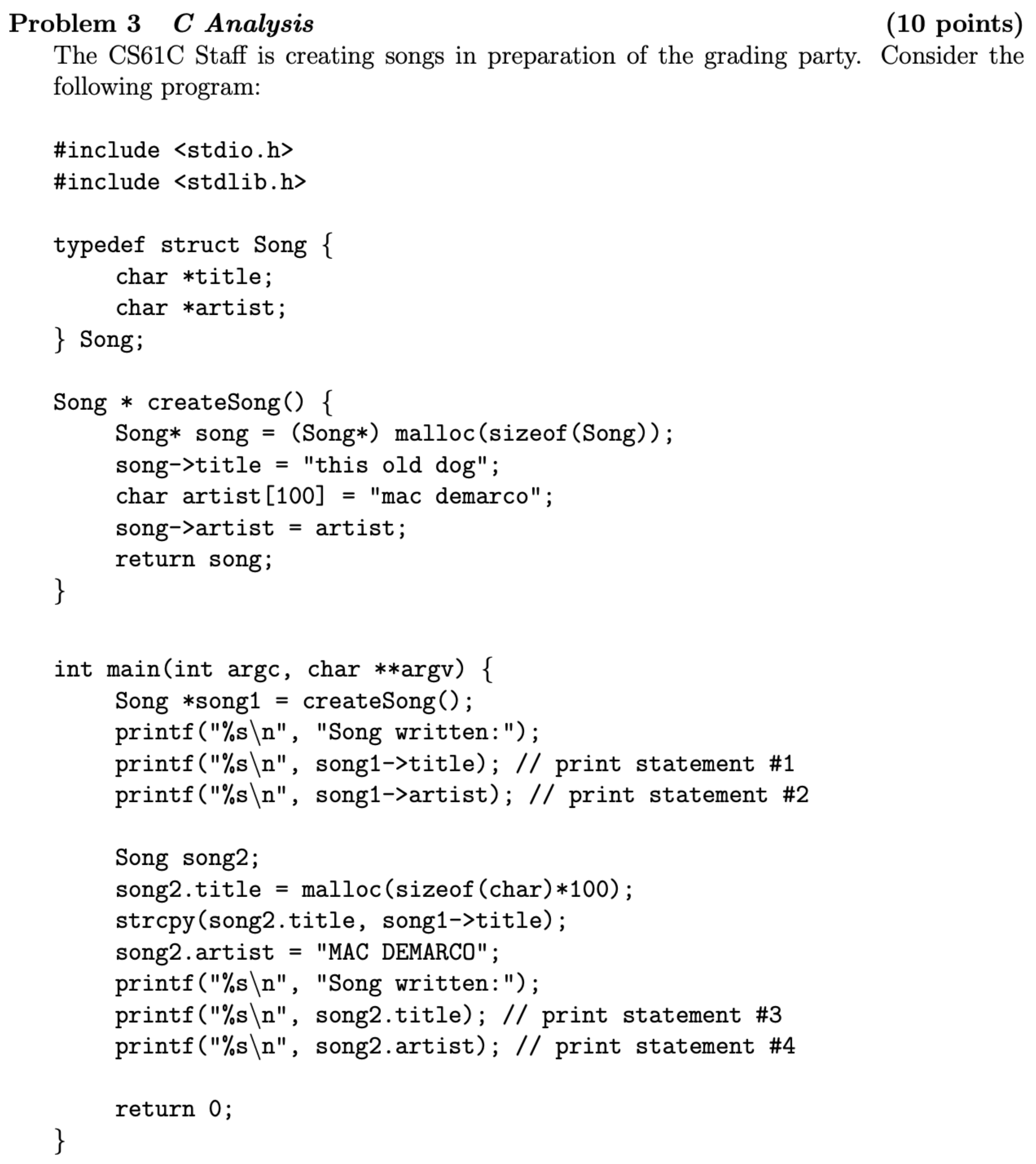
○ Stack address ○ Static address

[✓ Heap address] ○ Code address

In this case, why is `song1` on the heap?

Isn't it located on stack but points to memory on heap (Like in this link)?

(so `song1` is on stack and `*song1` is on heap?)



Problem 3 ***C Analysis*** **(10 points)**

The CS61C Staff is creating songs in preparation of the grading party. Consider the following program:

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

typedef struct Song {
     char *title;
     char *artist;
} Song;

Song * createSong() {
     Song* song = (Song*) malloc(sizeof(Song));
     song->title = "this old dog";
     char artist[100] = "mac demarco";
     song->artist = artist;
     return song;
}

int main(int argc, char **argv) {
     Song *song1 = createSong();
     printf("%s\n", "Song written:");
     printf("%s\n", song1->title); // print statement #1
     printf("%s\n", song1->artist); // print statement #2

     Song song2;
     song2.title = malloc(sizeof(char)*100);
     strcpy(song2.title, song1->title);
     song2.artist = "MAC DEMARCO";
     printf("%s\n", "Song written:");
     printf("%s\n", song2.title); // print statement #3
     printf("%s\n", song2.artist); // print statement #4

     return 0;
}
```

M

Myrah Shah STAFF 8mth #466da

Replying to Anonymous Red deer

There is a difference in asking which segment a value is located in, and

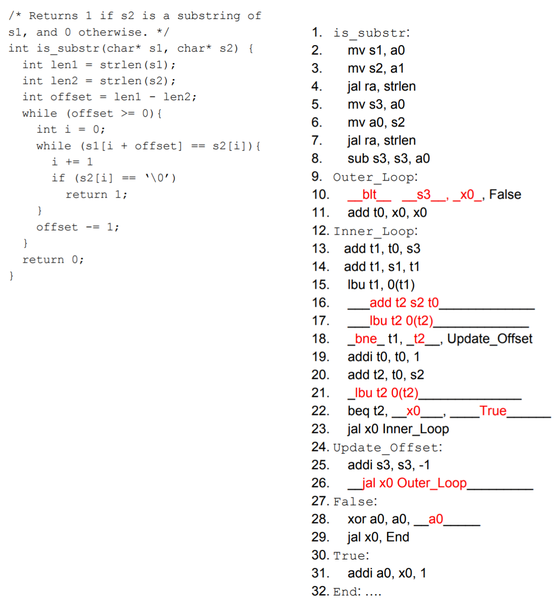
> What type of address does each value evaluate to?

In the latter, we want to think about the value stored in `song1`, which here would be a pointer to the song we created. This song is stored on the heap, since we called `malloc` for it, so the value **stored** in song1 is a heap address (a pointer to something on the heap).

Anonymous Red deer 9mth #466af ✓ Resolved

FA 17- Midterm 1 Q4:

On line 26, does jal change the x0 register?



```
/* Returns 1 if s2 is a substring of
s1, and 0 otherwise. */
int is_substr(char* s1, char* s2) {
  int len1 = strlen(s1);
  int len2 = strlen(s2);
  int offset = len1 - len2;
  while (offset >= 0){
    int i = 0;
    while (s1[i + offset] == s2[i]){
      i += 1
      if (s2[i] == '\0')
        return 1;
    }
    offset -= 1;
  }
  return 0;
}
```

```
1.  is_substr:
2.     mv s1, a0
3.     mv s2, a1
4.     jal ra, strlen
5.     mv s3, a0
6.     mv a0, s2
7.     jal ra, strlen
8.     sub s3, s3, a0
9.  Outer_Loop:
10.    __blt__ __s3__, _x0_, False
11.    add t0, x0, x0
12. Inner_Loop:
13.    add t1, t0, s3
14.    add t1, s1, t1
15.    lbu t1, 0(t1)
16.    ___add t2 s2 t0____________
17.    ___lbu t2 0(t2)____________
18.    _bne_ t1, _t2__, Update_Offset
19.    addi t0, t0, 1
20.    add t2, t0, s2
21.    _lbu t2 0(t2)_____________
22.    beq t2, __x0___, ____True_____
23.    jal x0 Inner_Loop
24. Update_Offset:
25.    addi s3, s3, -1
26.    __jal x0 Outer_Loop_________
27. False:
28.    xor a0, a0, __a0_____
29.    jal x0, End
30. True:
31.    addi a0, x0, 1
32. End: ....
```

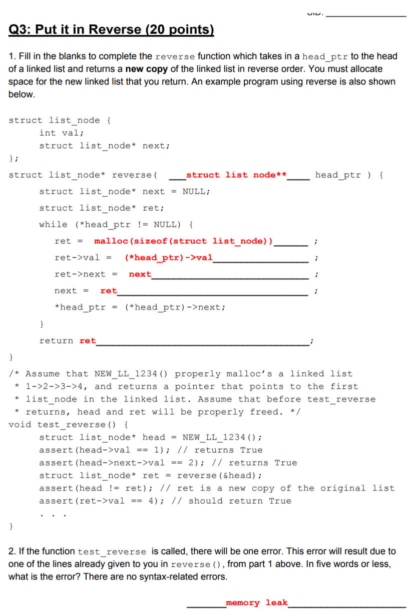
Nikhil Kandkur STAFF 9mth #466ba

No, since the x0 will always store the value 0, which means that a `jal x0 label` instruction is equivalent to a `j label` instruction since we do not want to keep track of where we jumped from.

Anonymous Red deer 9mth #466ae ✓ Resolved

FA17-Midterm 1 Q3:

For this problem, can the head_ptr instead be of type struct list_node*?



SID: ______________

Q3: Put it in Reverse (20 points)

1. Fill in the blanks to complete the `reverse` function which takes in a `head_ptr` to the head of a linked list and returns a **new copy** of the linked list in reverse order. You must allocate space for the new linked list that you return. An example program using reverse is also shown below.

```
struct list_node {
      int val;
      struct list_node* next;
};
struct list_node* reverse( ___struct list node**____ head_ptr ) {
      struct list_node* next = NULL;
      struct list_node* ret;
      while (*head_ptr != NULL) {
         ret = malloc(sizeof(struct list_node))______ ;
         ret->val = (*head_ptr)->val_________________ ;
         ret->next = next____________________________ ;
         next = ret__________________________________ ;
         *head_ptr = (*head_ptr)->next;
      }
      return ret_____________________________________;
}
/* Assume that NEW_LL_1234() properly malloc's a linked list
 * 1->2->3->4, and returns a pointer that points to the first
 * list_node in the linked list. Assume that before test_reverse
 * returns, head and ret will be properly freed. */
void test_reverse() {
      struct list_node* head = NEW_LL_1234();
      assert(head->val == 1); // returns True
      assert(head->next->val == 2); // returns True
      struct list_node* ret = reverse(&head);
      assert(head != ret); // ret is a new copy of the original list
      assert(ret->val == 4); // should return True
      . . .
}
```

2. If the function `test_reverse` is called, there will be one error. This error will result due to one of the lines already given to you in `reverse()`, from part 1 above. In five words or less, what is the error? There are no syntax-related errors.

_______memory leak____________________

Andrew Liu STAFF 9mth #466bd

No, since you must change what the head of the linked list actually is by assigning to `*headptr`, which would be a `struct list_node*`

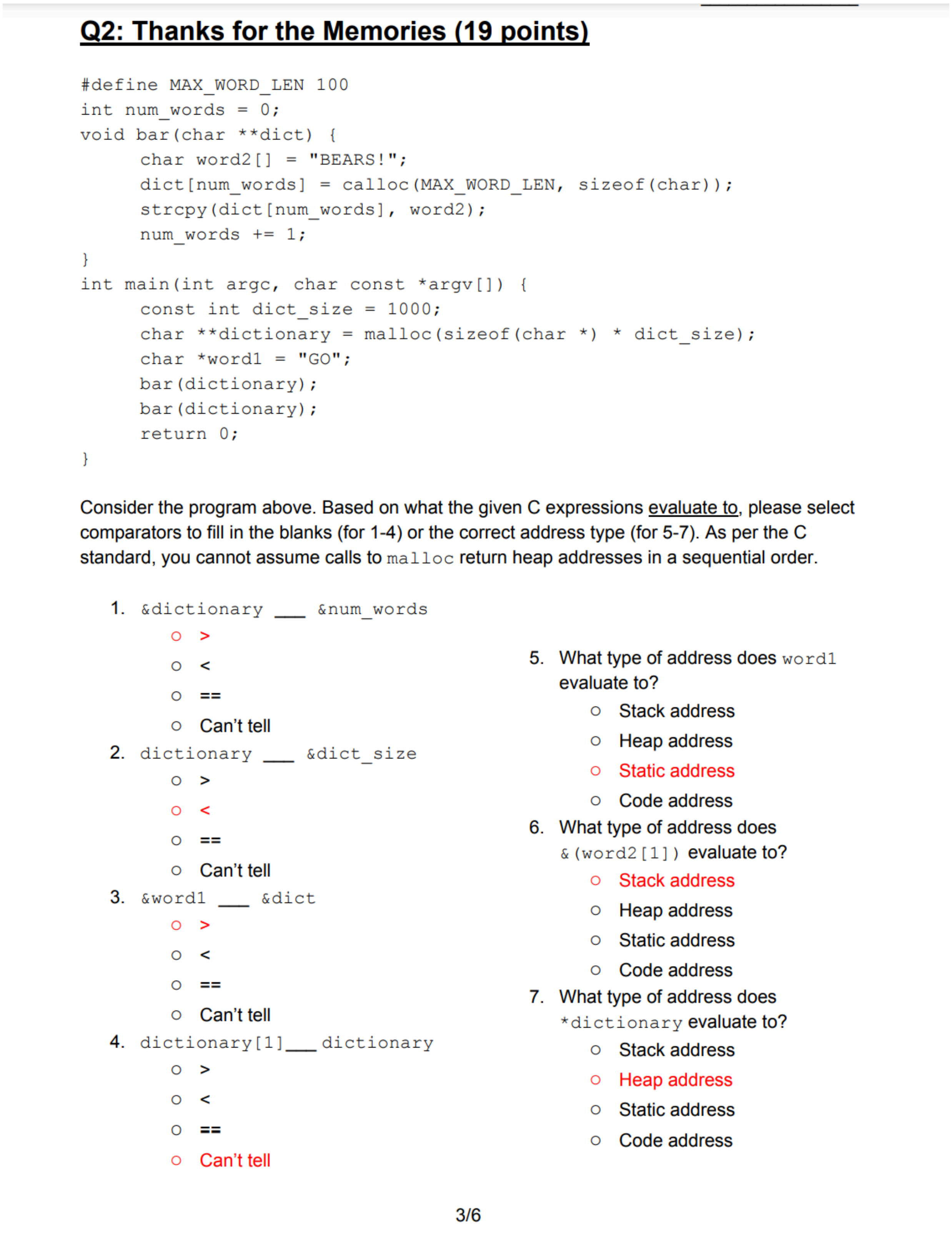
Anonymous Red deer 9mth #466ad ✓ Resolved

FA17-Midterm 1 Q2:

How can I approach problem 1-4, I am completely lost.

Also, for problem 5, why is it static address. In particular, what's the

difference between problem 5 and 6?



Q2: Thanks for the Memories (19 points)

```
#define MAX_WORD_LEN 100
int num_words = 0;
void bar(char **dict) {
      char word2[] = "BEARS!";
      dict[num_words] = calloc(MAX_WORD_LEN, sizeof(char));
      strcpy(dict[num_words], word2);
      num_words += 1;
}
int main(int argc, char const *argv[]) {
      const int dict_size = 1000;
      char **dictionary = malloc(sizeof(char *) * dict_size);
      char *word1 = "GO";
      bar(dictionary);
      bar(dictionary);
      return 0;
}
```

Consider the program above. Based on what the given C expressions evaluate to, please select comparators to fill in the blanks (for 1-4) or the correct address type (for 5-7). As per the C standard, you cannot assume calls to `malloc` return heap addresses in a sequential order.

1. `&dictionary ___ &num_words`
   - [✓ >]
   - <
   - ==
   - Can't tell
2. `dictionary ___ &dict_size`
   - >
   - [✓ <]
   - ==
   - Can't tell
3. `&word1 ___ &dict`
   - [✓ >]
   - <
   - ==
   - Can't tell
4. `dictionary[1]___ dictionary`
   - >
   - <
   - ==
   - [✓ Can't tell]
5. What type of address does `word1` evaluate to?
   - Stack address
   - Heap address
   - [✓ Static address]
   - Code address
6. What type of address does `&(word2[1])` evaluate to?
   - [✓ Stack address]
   - Heap address
   - Static address
   - Code address
7. What type of address does `*dictionary` evaluate to?
   - Stack address
   - [✓ Heap address]
   - Static address
   - Code address

3/6

Andrew Liu STAFF 9mth #466be

I would check out the lecture slides on C memory sections, as well as the problem in the C homework about memory segments or the discussion on C. There, the memory segments and their contents are listed out.

Anonymous Kookabura 8mth #466ec

For #3, isn't &word1 < &dict? word1 is stored in static data and dict is stored in the stack (as a function parameter), to the best of my knowledge.

Anonymous Buffalo 9mth #466b ✓ Resolved

SP21-Midterm-Q7

how do you know what is the size of struct if there are different data types and not all of them may take up the full 4 bytes?

Andrew Liu STAFF 9mth #466d

The answer involves stack alignment: https://en.cppreference.com/w/c/language/object

The basic idea is that each field needs to be aligned to its size (e.g. an `int32_t` must live at an address ending in `0x0, 0x4, 0x8, or 0xA`)

So, the struct looks like

```
struct foo {
    char a    (byte 0)
    padding   (byte 1)
    padding   (byte 2)
    padding   (byte 3)
    char *b   (byte 4)
    char *b   (byte 5)
    char *b   (byte 6)
    char *b   (byte 7)
}
```

Anonymous Buffalo 9mth #466e

https://www.geeksforgeeks.org/is-sizeof-for-a-struct-equal-to-the-sum-of-sizeof-of-each-member/

What about in this example where in Case 2, the int and short are next to each other and the padding is on the right of short? Can we assume that the padding is always inserted in between elements so that each element starts at a new multiple of 4?

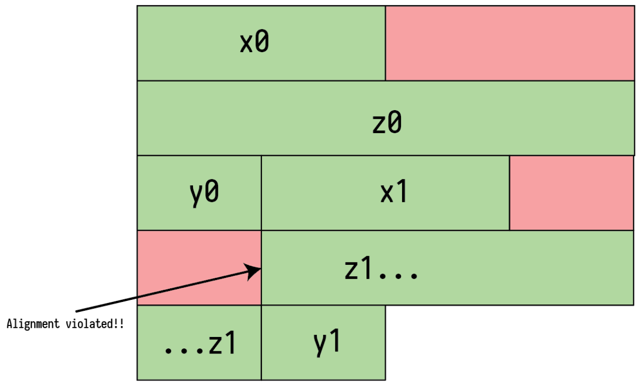
In your example, isn't char* pointer 4 bytes so is it byte 4 then 8, 12, 16 for the last four elements in the struct?

Andrew Liu STAFF 9mth #466f

Replying to Anonymous Buffalo

Padding is introduced so that *for each element in a struct, and the struct,* the alignment holds. For example, compare case 1 and case 2. Case 1 has extra padding at the end of the end of the struct. Why? Well imagine if it wasn't there. Then, we would end up with something like this if we had 2 structs next to each other:

```
struct foo {
    int32_t x;
    int64_t z;
    int16_t y;
}
```



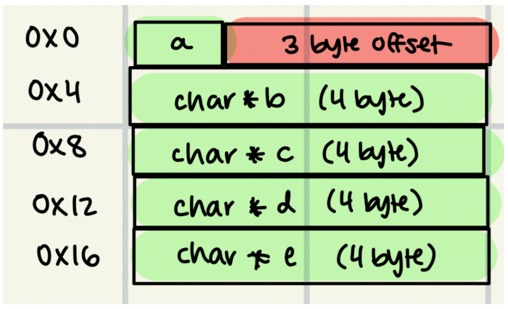
The way I think about it is that the padding is added before each member if placing it immediately causes alignment issues. For example, in case 1, placing `z` immediately after `x` would have caused `z` to be misaligned, so 4B of padding was added. After placing `z`, `y` is perfectly happy being aligned where it is, so no padding was added. And then, the struct's overall size must be padded to a multiple of the largest member (to avoid the issue in the above diagram), so we appended 6B of padding.

In case 2, we place `z`, then `x` is perfectly happy being aligned to the start of an 8B boundary, so no padding is placed before `x`. After `x`, `y` is perfectly happy being aligned to the start of a 4B boundary, so no padding is placed before `y`. To finish, the struct needs to be a multiple of `8B`, the size of a `double`, so 2B of padding was added to make the struct 16B large.

Anonymous Buffalo 9mth #466aa

Replying to Andrew Liu

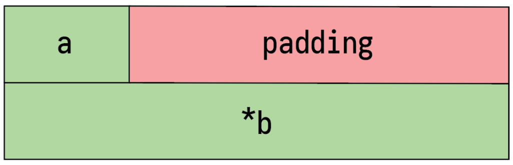
Thank you so much! So in your original example, would it look something like this



Andrew Liu STAFF 9mth #466ab

Replying to Anonymous Buffalo

ohhhhh sorry! I didn't draw a diagram and didn't realize that that was confusing — I meant that 1 byte was allocated for `a`, 3 for padding, and 4 for `b`, like this:

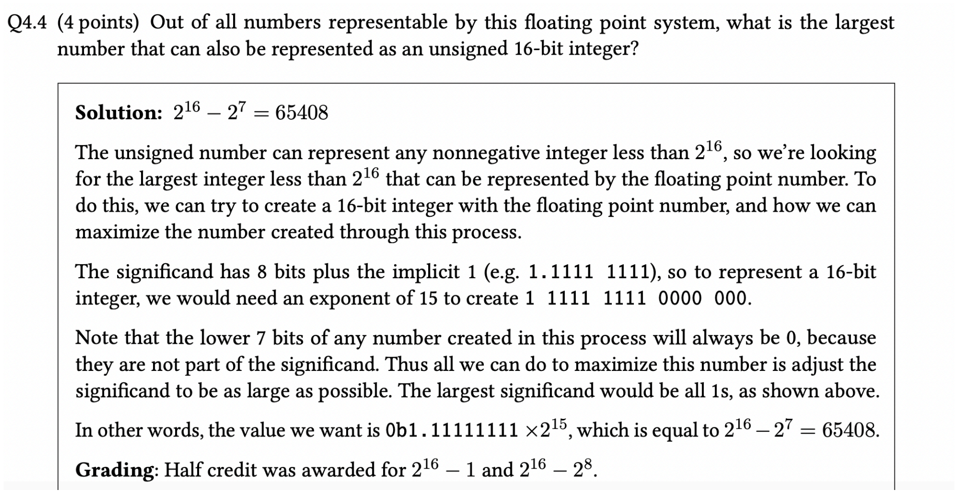


Anonymous Buffalo 9mth #466ac

Replying to Andrew Liu

oh that makes a lot of sense, thank you!

Anonymous Buffalo 9mth #466a ✓ Resolved


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 $2^{16} - 2^8$.

FA21-Midterm-Q4.4

how did they get the exponent of 15? is the strategy to first maximize the significand and then figure out what proper exponent should be?

Justin Yokota STAFF 9mth #466c

Not really significand, but the appropriate final value, yes. Another way to see this is that Exponent 16 or higher would be greater than 2^16, and therefore not representable in a 16-bit int.