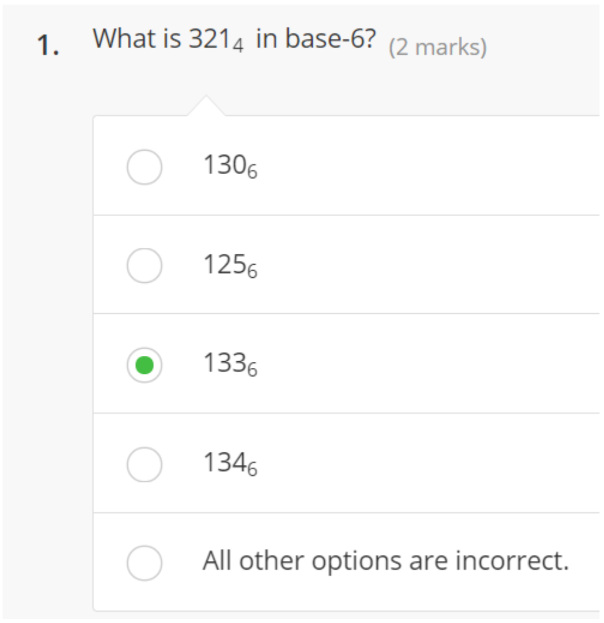
CS2100 AY2021 Semester 1

Midterm Assessment Sample Answer and unnecessary statistics

1152! That's the number of potential variants of this midterm, it is safe to say that the chance of any two of you having the same set of questions is very very low! Although I am proud of that as a mechanism to protect and strengthen the integrity of the test, the number of variants pose interesting problems for a "sample answer" like this document. In the end, I decided to share the answer for only one variant per question. As the variant differs only in the "parameter", you should be able to deduce the answer for your own variant if needed. I will also indicate the number of variants per question.



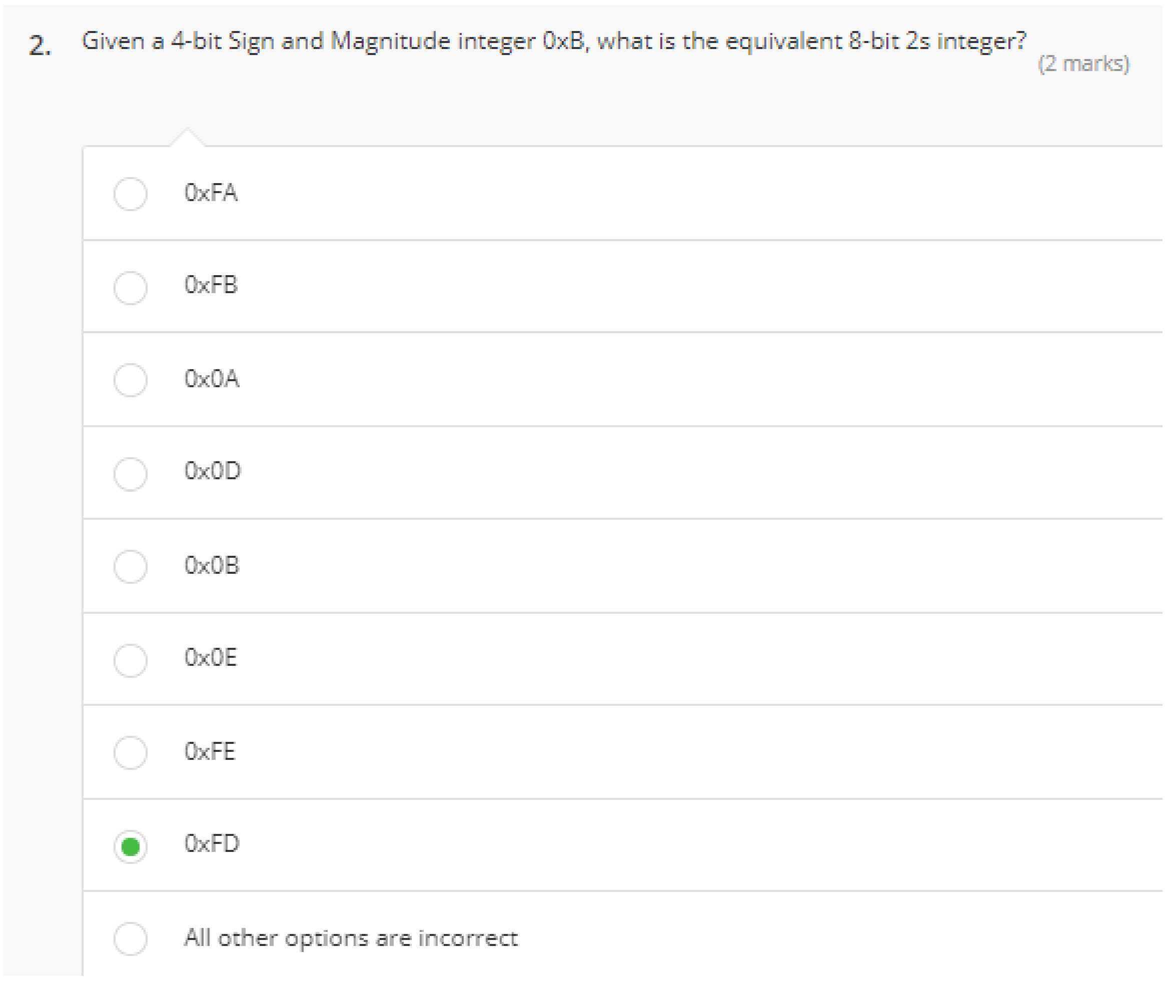
1. What is 321_4 in base-6? (2 marks)

- ○ 130_6
- ○ 125_6
- ● 133_6
- ○ 134_6
- ○ All other options are incorrect.

[Easy] Convert to base-10, then to base-6.

[4 variants, differs in the number]

[Average: 1.8, Median: 2.0]



2. Given a 4-bit Sign and Magnitude integer 0xB, what is the equivalent 8-bit 2s integer? (2 marks)

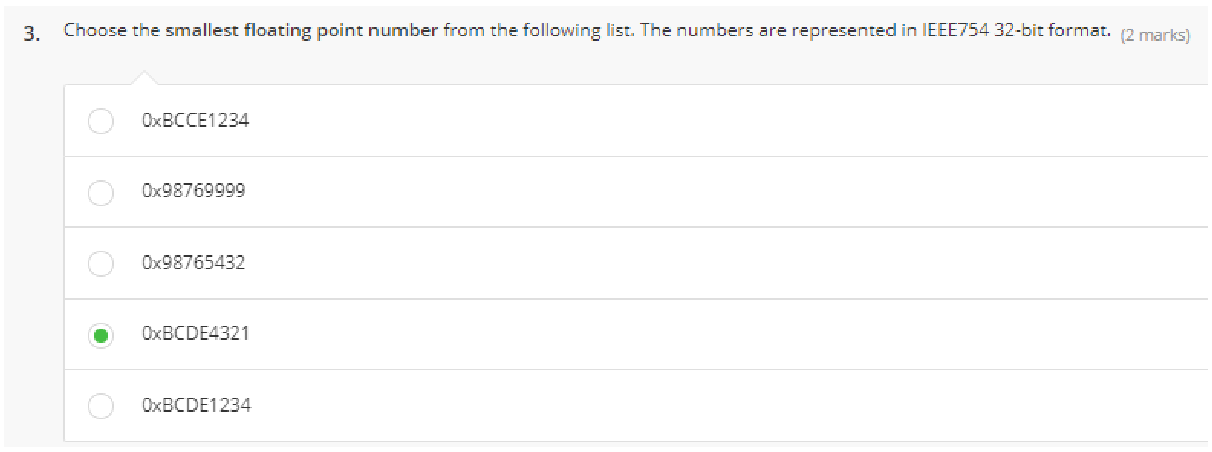
- ○ 0xFA
- ○ 0xFB
- ○ 0x0A
- ○ 0x0D
- ○ 0x0B
- ○ 0x0E
- ○ 0xFE
- ● 0xFD
- ○ All other options are incorrect

[Easy] The S&M number is NEGATIVE, so the 2s integer needs to do the “flip and add 1” operation after padding to 8-bit.

0xB = 1011_2 = -3_{10} = $-0000\ 0011_2$ = $1111\ 1101_2$ = 0xFD

[4 variants, differs in the S&M number given (all negative)]

[Average: 1.5, Median: 2]



3. Choose the **smallest floating point number** from the following list. The numbers are represented in IEEE754 32-bit format. (2 marks)

- 0xBCCE1234
- 0x98769999
- 0x98765432
- [✓ 0xBCDE4321]
- 0xBCDE1234

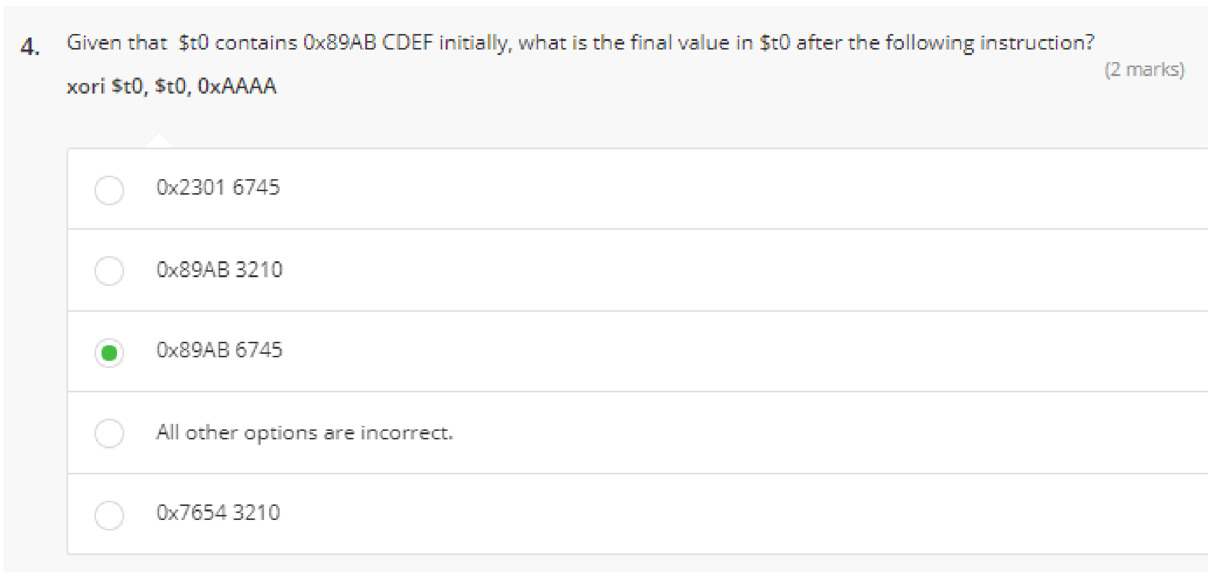
[Medium] Need to make two observations:

a. All number are negative ➔ whichever has the largest exponent is smallest. E.g. $-1.0 * 2^6 < -1.0 * 2^7$
b. Larger exponent ➔ The hexadecimal representation is just larger
c. Larger mantissa ➔ The hexadecimal representation is larger (if the exponent is the same)

So, instead of translating all of the numbers into floating point, you can just take the hexadecimal and take the largest (as interpreted as integer).

[3 variants, essentially the same set of number but change the first hexadecimal while ensuring it is negative.]

[Average: 1.0, Median: 2.0]



4. Given that $t0 contains 0x89AB CDEF initially, what is the final value in $t0 after the following instruction? (2 marks)

xori $t0, $t0, 0xAAAA

- 0x2301 6745
- 0x89AB 3210
- [✓ 0x89AB 6745]
- All other options are incorrect.
- 0x7654 3210

[Easy] Need to know the bit-wise immediate instruction takes in only **lower 16-bit** bit mask, i.e. the higher 16-bit are **zeroes**. So, xori can only modify the **lower-16 bit** of the $t0.

Xori **flips the bit** only if the bit mask is a 1.

So:

$t0, original lower 16-bit 0xCDEF:

1	1	0	0	1	1	0	1	1	1	1	0	1	1	1	1

XOR with bit mask:

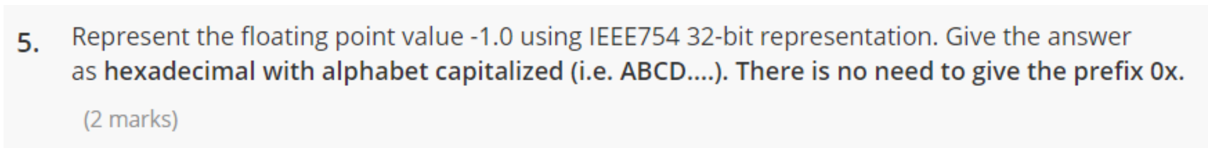
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Result, only lower 16-bit changed:

0	1	1	0	0	1	1	1	0	1	0	0	0	1	0	1

[2 variants, the original value is different, but same bit mask]

[Average:1.6, Median: 2]



5. Represent the floating point value -1.0 using IEEE754 32-bit representation. Give the answer as **hexadecimal with alphabet capitalized (i.e. ABCD....). There is no need to give the prefix 0x.**

(2 marks)

[Easy] Need to remember to bias the exponent. Number in normalized format = -1.0×2^0

Sign bit = 1

Exponent = 0 + 127 = 127 = $0111\ 1111_2$

Mantissa = 000000……

Answer = 0xBF80 0000

[2 Variants, one positive, one negative]

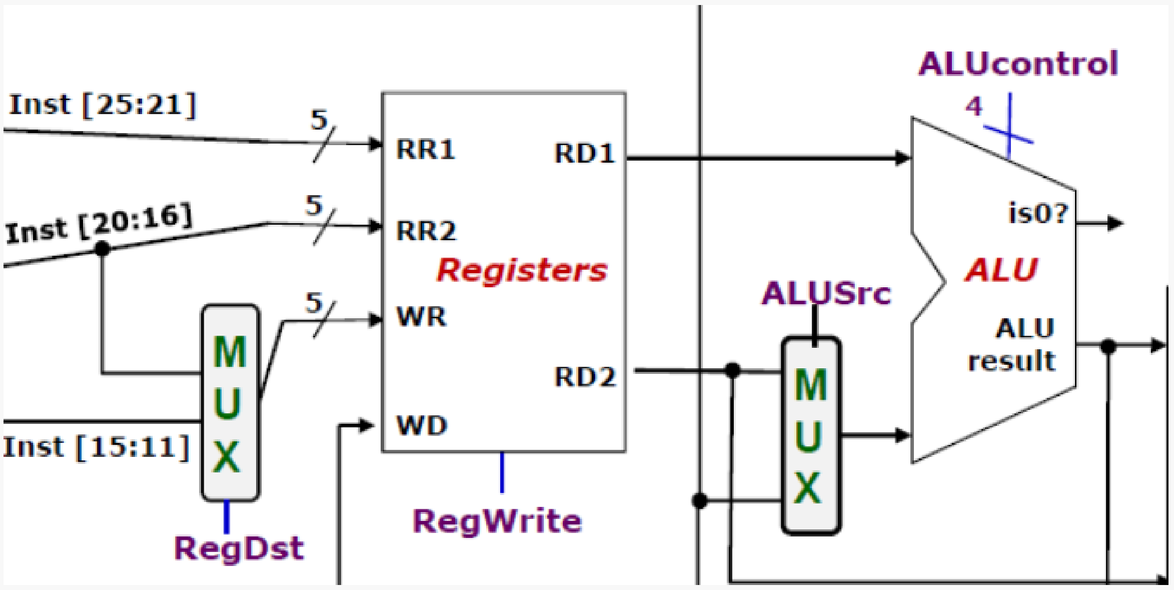
[Marking note: if you get 2 out of 3 components, e.g. Sign bit and Mantissa bit correct, you get 1 mark. No mark for a single component only.]

[Average:1.5, Median: 2]

Suppose we have a mysterious instruction M with the following characteristic:

1. M is an I-format instruction.
2. M perform subtraction of rs and immediate value in the ALU unit.
3. M does not change any register content at the end of execution

You are also given the relevant part of MIPS processor:



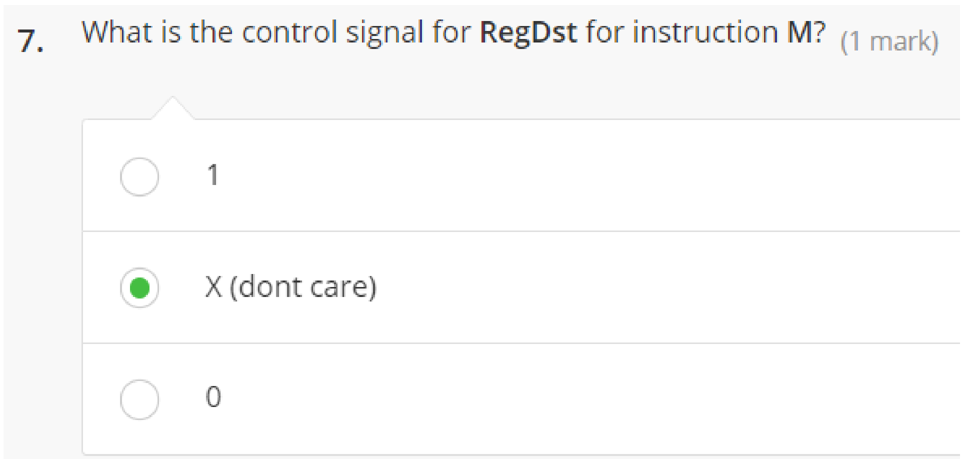
6. What is the control signal for **ALUSrc** for instruction M? (1 mark)

- 0
- X (dont care)
- [✓ 1]

[Easy] From the given info (1) and (2), we know the lower path is used.

[Variant: Q6 and 7 may be swapped randomly.]

[Average: 0.8, Median: 1.0]

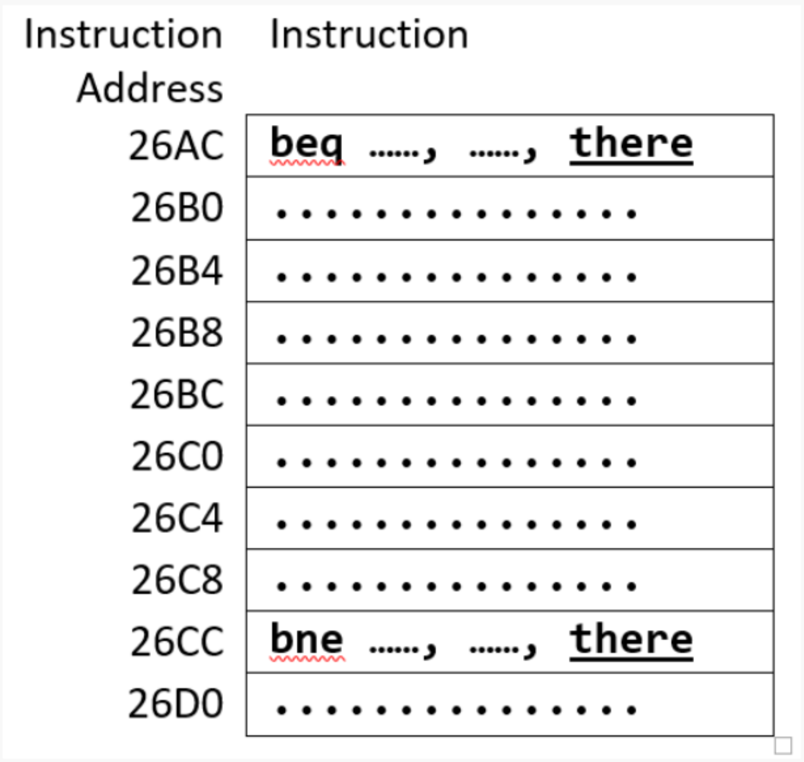

7. What is the control signal for **RegDst** for instruction **M**? (1 mark)

- ○ 1
- [✓ X (dont care)]
- ○ 0

[Easy] From the given info (3), we know the register is not written, i.e. doesn't matter what is the destination register number.

[Average: 0.9, Median: 1.0]

8. Observe the following instructions with memory address:



Instruction Address	Instruction
26AC	beq ......, ..., there
26B0	....
26B4	....
26B8	....
26BC	....
26C0	....
26C4	....
26C8	....
26CC	bne ......, ..., there
26D0	....

Suppose the beq (the first branch) has an immediate value of +5, what is the immediate value for the bne (the second branch)? All options are in decimal.

(1 mark)

- +4
- +3
- -5
- +5
- -4
- All other options are incorrect.
- [✓ -3]

[Easy] Just remember to count from PC+4 (for both beq and bne).

[3 Variants, the immediate value of beq is different]

[Average: 0.9, Median: 1.0]

9. Mr.Mehta managed to place the 32-bit encoding of an ADD instruction, 0x0291 9820 in register $t1 (i.e. the content of register $t1 is 0x0291 9820).

He wants to change $t1 to store the encoding of instruction "slt $26, $20, $17" instead. Give **one MIPS instruction to manipulate register $t1 to get what Mr.Mehta wants.** Below is the relevant information from the MIPS reference sheet.

add	R	R[rd] = R[rs] + R[rt]	$0 / 20_{hex}$
slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	$0 / 2a_{hex}$

(3 marks)

[Medium] This question combine encoding and MIPS instruction into one.

First observe that the difference in the **add** and **slt** instruction encodings lies in the lower 16-bit only:

Add (original): 0x0291 9820

Slt : 0x0291 D02A

If we focus only on the lower-16 bits:

0x9820	1001 1000 0010 0000
0xD02A	1101 0000 0010 1010

The bits different from the original add encoding is highlighted in red.

There are then two options:

a. Use XORI to flip the bits that are different. OR
b. Use ADDI to add the difference (i.e. treat them as number).

In this variant, the answers:

a. xori $t1, $t1, 0x480A OR
b. addi $t1, $t1, 0x380A

[2 variants, the $t1 is slightly different]

[Marking note: If you use the **right instruction (i.e. addi or xori)**, then partial mark will be given if you get the immediate value slightly wrong. I use a -1 penalty per every bit difference. E.g. if you use 0x580A for xori, there is a 1-bit difference = 2 marks, if you use 0x780A ➔ 2-bit difference = 1 mark; if you use 0x781A ➔ 3 bit difference = 0 mark].

[Average: 1.4, Median: 1]

Questions in this section are based on the following context.

The MIPS source code is placed on the left. Note that initial setup for the registers $s0 to $s2.

The data memory is shown on the right. Note that the data memory segment shown is **incomplete**. Also, note that we show four bytes per row.

```
$s0 = 0x2100 (base address of R[])
$s1 = 0x54325432 (user input)
$s2 = 0 initially (result)
loop:
      beq  $s1, $zero, end
      andi $t0, $s1, 15
      add  $t1, $s0, $t0
      lb   $t2, 0($t1)
      add  $s2, $s2, $t2
      srl  $s1, $s1, 4
      j loop
end:
```

	Byte 0	**Byte 1**	**Byte 2**	**Byte 3**
0x2100	0	1	1	2
0x2104	1	2	2	3
0x2108	1	2	2	3
…				
…				

Data memory (**partially shown**).
Values shown in decimal.

10. After the **first iteration** of the code (i.e. when we reach "j" for the first time), give the updated content of register $s1 in hexadecimal. Use capital letter for ABCDEF and leave out the prefix 0x in your answer.

(1 mark)

[Easy] A simple test on shift right. Answer is 0x05432543 (leading zero can be left out).

[Average: 0.7, Median: 1]

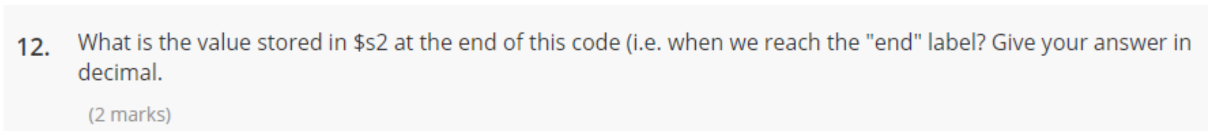
11. During the **first iteration** of the code, what is the byte value in register $t2 after the "lb $t2, 0($t1)" instruction? Give your answer in decimal.

(2 marks)

[Medium] Test your understanding on memory address. We are essentially taking the input in $s1 one hexadecimal value (i.e. 0…15) as index to R[].

In this case, we are essentially reading R[2], i.e. Byte 2 of the row 0x2100 ➔ **1**

[Average: 1.6, Median: 2]



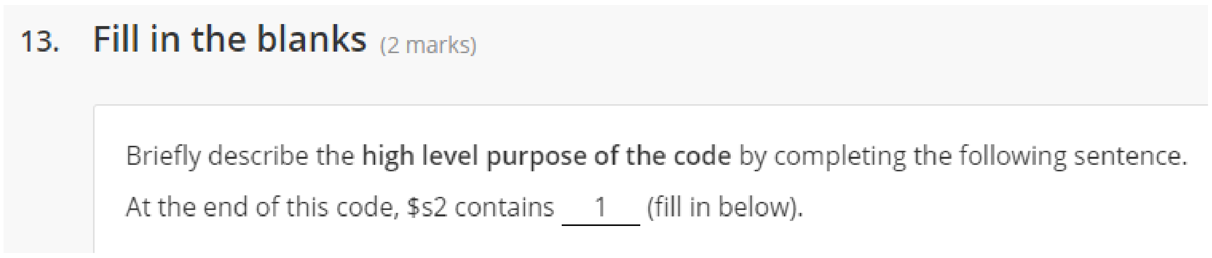
12. What is the value stored in $s2 at the end of this code (i.e. when we reach the "end" label? Give your answer in decimal.

(2 marks)

[Medium] Once you understood Q11, you can quickly calculate the answer. To reduce tediousness, I have chosen an input value that have two similar halves. So, you can calculate just one side and multiply by 2.

(R[2] + R[3] + R[4] + R[5]) x 2   = **12**

[Average: 1.1, Median: 2]



13. Fill in the blanks (2 marks)

Briefly describe the **high level purpose of the code** by completing the following sentence.

At the end of this code, $s2 contains __1__ (fill in below).

[Hard] The ability to understand code from a “zoomed out” point of view is hard to come by. So, this is more like a “badge of honor” type of question, i.e. you should feel proud if you are able to get it, but don’t feel too bad if you cant.

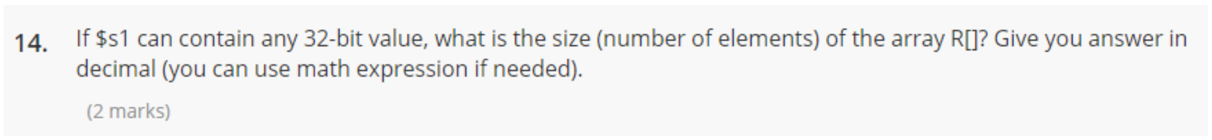
First of all, when we ask this type of question, we are not expecting a English translation of the steps (i.e. “this code take input 4 bit at a time and index into R[] and sum them”). Rather, you need to ask “why are we doing this? What good is it for?”. Like I mentioned to the class (and in previous midterm write-up), I try not to give “useless program” in assessment tasks ☺.

This code is actually an efficient way to find out the the number of ‘1’s in the user input. E.g. 0x5 ➔ 0101_2, there are two ‘1’s ➔ R[5] == 2.

0x7 ➔ 0111_2, there are three ‘1’s ➔ R[7] == 3.

By using indexing and the lookup table R[], we can quickly find out the total ‘1’s in the user input, e.g. 0x54325432 has twelve ‘1’s in total.

[Average: 0.02 ☹, Median: 0 ☹, there are only 4 students who figured this out].



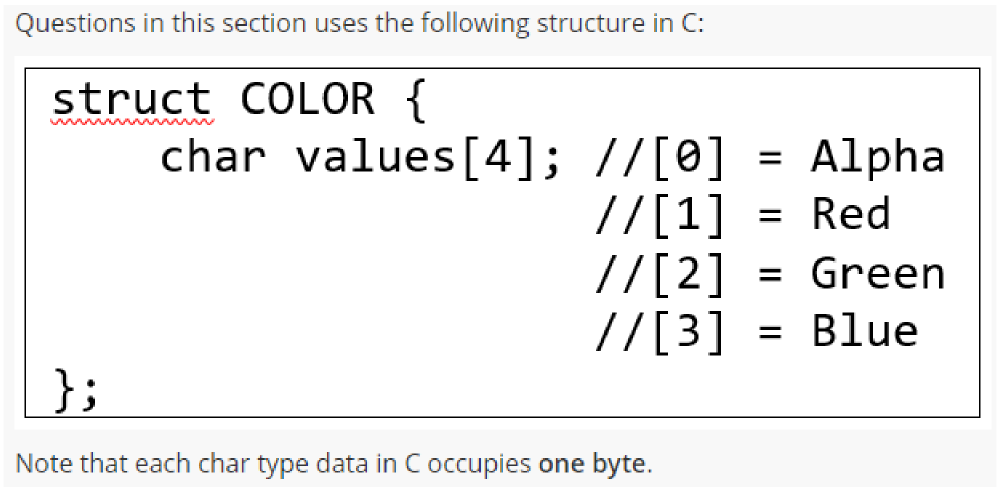
14. If $s1 can contain any 32-bit value, what is the size (number of elements) of the array R[]? Give you answer in decimal (you can use math expression if needed).

(2 marks)

[Medium] If $s1 can contain any 32-bit value, then a single 4-bit portion can contain values from 0x0 to 0xF. Since we use every 4-bit portion as index, this implies that the array need to contain at least 16 values, i.e. from R[0] to R[15].

Answer is **16**.

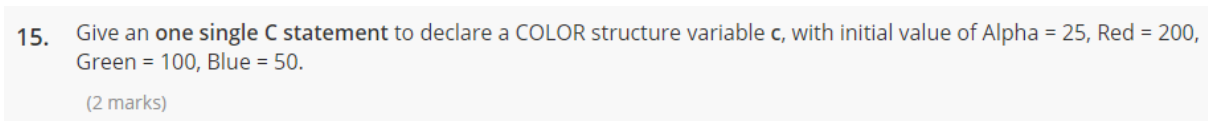
[Average: 0.7, Median: 0]



Questions in this section uses the following structure in C:

```
struct COLOR {
        char values[4]; //[0] = Alpha
                        //[1] = Red
                        //[2] = Green
                        //[3] = Blue
};
```

Note that each char type data in C occupies **one byte**.



15. Give an **one single C statement** to declare a COLOR structure variable **c**, with initial value of Alpha = 25, Red = 200, Green = 100, Blue = 50.

(2 marks)

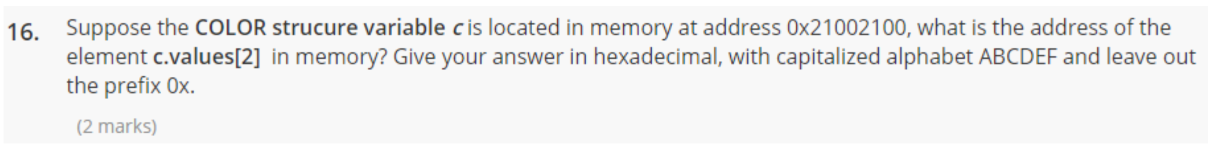
[Easy-Medium?] Just a plain C syntax question.

Answer:

struct COLOR c = {25, 200, 100, 50};

[Marking: You need to at least declare the variable “c” correctly to get partial score (1 mark)]

[Average: 1.4, Median: 2]



16. Suppose the **COLOR strucure variable** *c* is located in memory at address 0x21002100, what is the address of the element **c.values[2]** in memory? Give your answer in hexadecimal, with capitalized alphabet ABCDEF and leave out the prefix 0x.

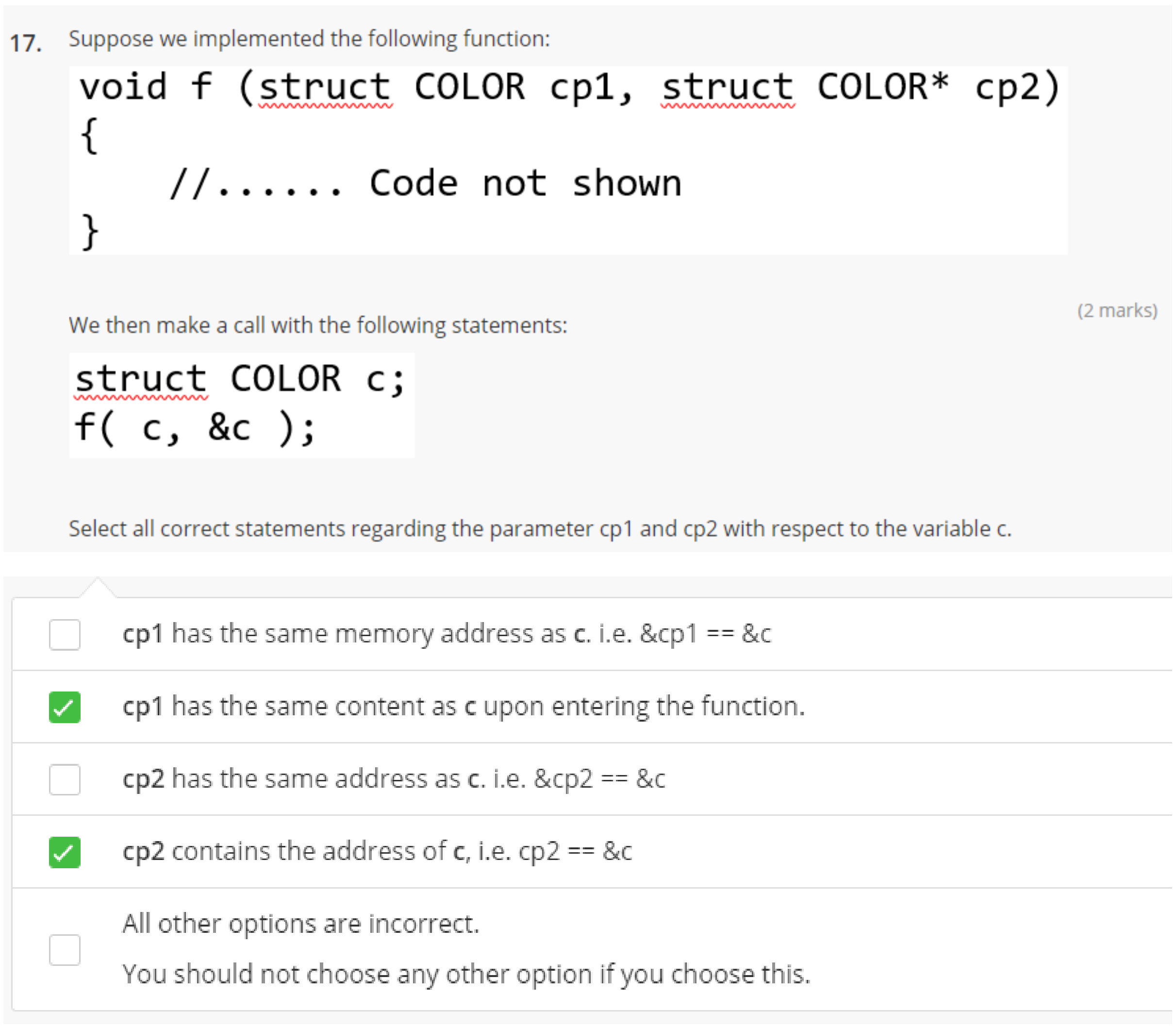
(2 marks)

[Easy] Check your understanding on memory snapshot for a structure.

The first field of the structure has the same address as the structure variable. In this case, the address of c is the same as c.value[0], i.e. 0x21002100.

So, c.value[2] is 2 bytes after c.value[0] ➔ **0x21002102**

[Average: 1.4, Median: 2]



17. Suppose we implemented the following function:

```
void f (struct COLOR cp1, struct COLOR* cp2)
{
     //...... Code not shown
}
```

(2 marks)

We then make a call with the following statements:

```
struct COLOR c;
f( c, &c );
```

Select all correct statements regarding the parameter cp1 and cp2 with respect to the variable c.

- [ ] **cp1** has the same memory address as **c**. i.e. &cp1 == &c
- [✓] **cp1** has the same content as **c** upon entering the function.
- [ ] **cp2** has the same address as **c**. i.e. &cp2 == &c
- [✓] **cp2** contains the address of **c**, i.e. cp2 == &c
- [ ] All other options are incorrect.
  You should not choose any other option if you choose this.

[Medium] A quick test of pass by value (cp1) and pass by pointer (cp2).

[Average: 1.8, Median: 2]

18. Suppose we declare **an array of color variables**, e.g.

```
struct COLOR colorArray[100];
```

We want to translate the following array element access in C:

```
colorArray[idx].values[2]
```

Given **$s0 as the memory address of colorArray[], $s1 as idx**, show how do you **load the "colorArray[idx].values[2]" into register $t2.** using no more than 5 MIPS instructions.

[Medium] Test of memory offset and MIPS instructions.

The simplest solution is:

```
sll $t0, $s1, 2   # $t0 = $s1 * 4, each structure is 4 bytes
add $t1, $s0, $t0 # address of colorArray[idx]
lb  $t2, 2($t1)   # use offset to get to .value[2]
```

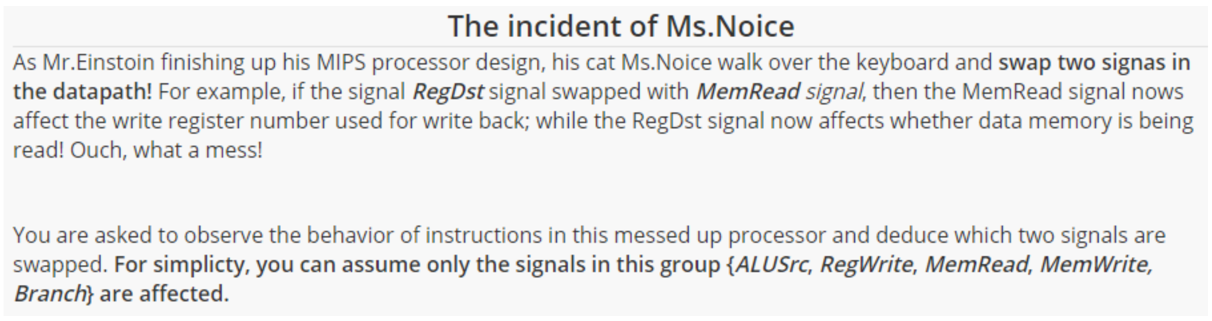
[Marking: Any > 3 instructions that **correctly solve the problem** get 3 marks. If you give the 3-line solution above, you **get 1 additional bonus mark**, i.e. 4 mark. The reason for the bonus score is that the offset of load instruction is specifically designed for accessing a structure / record. The base register points at the start of the structure, then use the offset to get to a specific field.

Mistake = penalty of 1 mark. > 2 mistakes = 0 mark.

Common mistakes:

- shift by 4 bits
- do load word instead of load byte

[Average: 1.6, Median: 1]

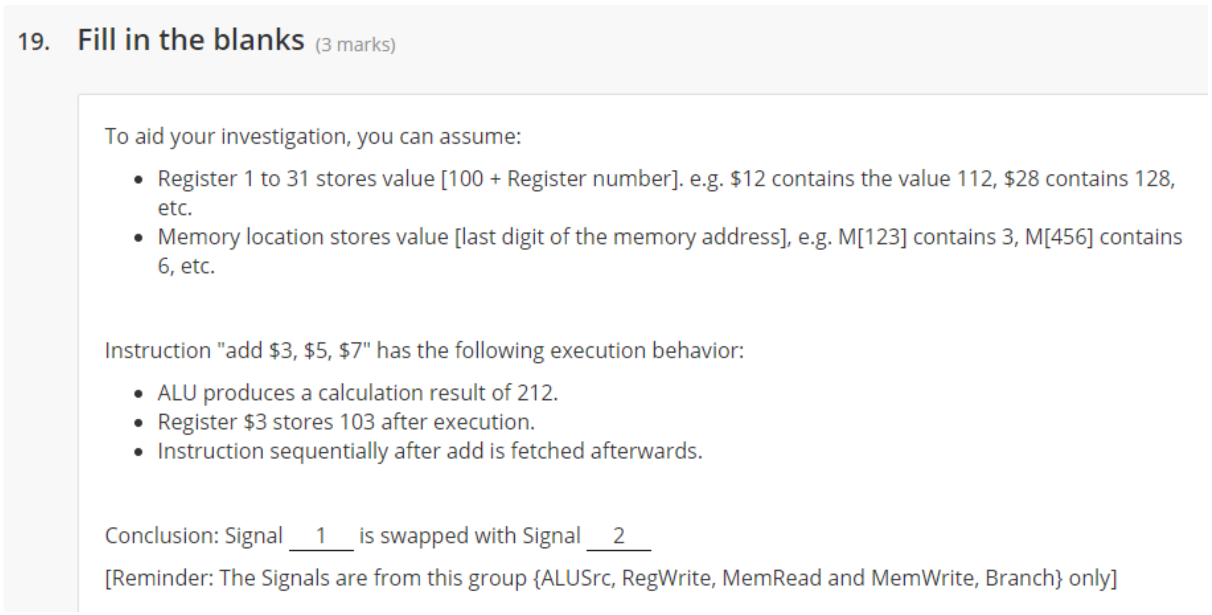


The incident of Ms.Noice

As Mr.Einstein finishing up his MIPS processor design, his cat Ms.Noice walk over the keyboard and **swap two signas in the datapath!** For example, if the signal ***RegDst*** signal swapped with ***MemRead*** *signal*, then the MemRead signal nows affect the write register number used for write back; while the RegDst signal now affects whether data memory is being read! Ouch, what a mess!

You are asked to observe the behavior of instructions in this messed up processor and deduce which two signals are swapped. **For simplicty, you can assume only the signals in this group {*ALUSrc*, *RegWrite*, *MemRead*, *MemWrite*, *Branch*} are affected.**

[Note: Standard Datapath + Signal diagram omitted here]



19. **Fill in the blanks** (3 marks)

To aid your investigation, you can assume:

- Register 1 to 31 stores value [100 + Register number]. e.g. $12 contains the value 112, $28 contains 128, etc.
- Memory location stores value [last digit of the memory address], e.g. M[123] contains 3, M[456] contains 6, etc.

Instruction "add $3, $5, $7" has the following execution behavior:

- ALU produces a calculation result of 212.
- Register $3 stores 103 after execution.
- Instruction sequentially after add is fetched afterwards.

Conclusion: Signal __1__ is swapped with Signal __2__

[Reminder: The Signals are from this group {ALUSrc, RegWrite, MemRead and MemWrite, Branch} only]

[Medium] Test out of the box thinking. ☺ It is actually simpler than you may think. For the swapped signal to have any impact ➔ they must have different value at the start ➔ once you located a missing "1" signal, you can need to find a suitable "0" signal; or vice versa.

Since Register $3 did not receive the correct execution result ➔ RegWrite signal is swapped.

Since RegWrite is a '1' for add instruction ➔ need to look for a originally '0' signal and change to 1 AND cause the right effect.

Since there is no other side effect observed ➔ the '1' must be affecting a component that doesn't cause any issue ➔ in this case, EITHER the Branch or the MemRead signal.

Note: The intended answer was Branch, but I left out an additional description "Latency was the same as the original" ☹. So, I accept BOTH answers.

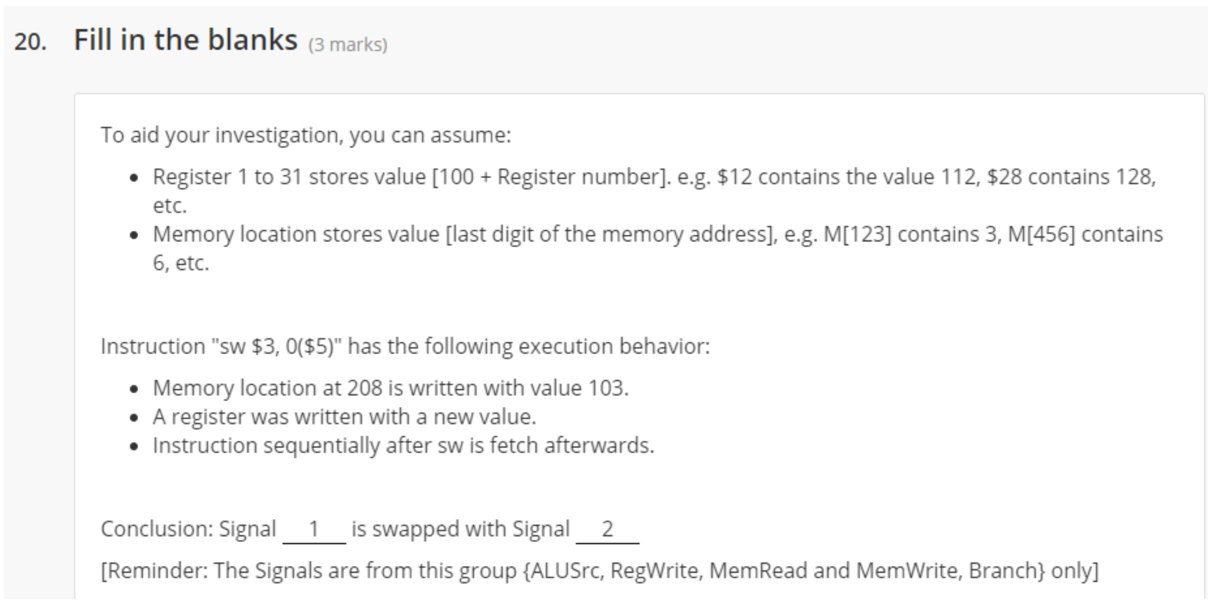
Answer:

a. RegWrite
b. Branch OR MemRead

[Variant: Your Q19 and Q20 may be swapped randomly]

[Marking: If you get only one of the signals correct ➔ 1 mark]

[Average: 1.8, Median: 3]



20. Fill in the blanks (3 marks)

To aid your investigation, you can assume:

- Register 1 to 31 stores value [100 + Register number]. e.g. $12 contains the value 112, $28 contains 128, etc.
- Memory location stores value [last digit of the memory address], e.g. M[123] contains 3, M[456] contains 6, etc.

Instruction "sw $3, 0($5)" has the following execution behavior:

- Memory location at 208 is written with value 103.
- A register was written with a new value.
- Instruction sequentially after sw is fetch afterwards.

Conclusion: Signal __1__ is swapped with Signal __2__

[Reminder: The Signals are from this group {ALUSrc, RegWrite, MemRead and MemWrite, Branch} only]

[Medium] Approach is the same as Q19.

Since memory address [208] is wrong (should be 105 instead) ➔ ALUSrc is swapped

Since ALUSrc should be '1' for sw ➔ need to look for a originally '0' signal and change to 1 AND cause the right effect.

Since sw should not write any value into register ➔ RegWrite is the other signal.

Answer:

a. RegWrite
b. ALUSrc

[Variant: Your Q19 and Q20 may be swapped randomly]

[Marking: If you get only one of the signals correct ➔ 1 mark]

[Average: 1.9, Median: 3]

Overall Statistics:

- Average: 25.43
- Median: 26
- Max: 39 (Good job!)
- Standard Deviation: 7.1