CS 61C Spring 2024

Yan, Yokota Midterm

PRINT Your Name: _____

PRINT Your Student ID: _____

You have 110 minutes. There are 8 questions of varying credit (100 points total).

Question:	1	2	3	4	5	6	7	8	Total
Points:	14	7.5	7	20	23.5	12	16	0	100

For questions with **circular bubbles**, you may select only one choice.

- ○ Unselected option (completely unfilled)
- ● Only one selected option (completely filled)
- ◍ Don't do this (it will be graded as incorrect)

For questions with **square checkboxes**, you may select one or more choices.

- ☐ You can select
- ■ multiple squares
- ■ (completely filled)

Anything you write outside the answer boxes or you ~~cross out~~ will not be graded. If you write multiple answers, your answer is ambiguous, or the bubble/checkbox is not entirely filled in, we will grade the worst interpretation. For coding questions with blanks, you may write at most one statement per blank and you may not use more blanks than provided.

If an answer requires hex input, you must only use capitalized letters (`0xDEADBEEF` instead of `0xdeadbeef`). For hex and binary, please include prefixes in your answers unless otherwise specified, and do not truncate any leading 0's. For all other bases, do not add any prefixes or suffixes.

Write the statement below in the same handwriting you will use on the rest of the exam.

> I have neither given nor received help on this exam (or quiz), and have rejected any attempt to cheat; if these answers are not my own work, I may be deducted up to 0x0123 4567 89AB CDEF points.

_____

_____

_____

_____

_____

SIGN your name: _____

Q1 *Potpourri* **(14 points)**

For Q1.1-Q1.2, convert the 8-bit binary `0b1100 1111` to decimal, treating it as...

Q1.1 (1 point) ...an **unsigned** integer.

Q1.2 (1 point) ...a **two's complement** integer.

Q1.3 (1 point) Convert the 12-bit number 4727_8 to hexadecimal.

0x

Q1.4 (1.5 points) Suppose there are 615 students enrolled in CS61C. What is the minimum number of bits needed to uniquely identify each student? Express your answer in decimal form.

bits

Q1.5 (1.5 points) An IEEE-754 double-precision floating point number can represent every integer that a 32-bit two's complement number can.

○ True ○ False

For Q1.6-Q1.7, consider a **16-bit** floating point format that follows the IEEE-754 standard, with 1 sign bit, 5 exponent bits (with a bias of -15) and 10 significand bits.

Q1.6 (1.5 points) Convert 6.25 into hexadecimal in this floating point format. If it cannot be represented, write "None".

0x

Q1.7 (1.5 points) What is the smallest positive value this format can support? Express your answer as 2^n where n is an integer.

Q1.8 (1.5 points) All labels in an assembly file can be referenced from other assembly files.

○ True ○ False

Q1.9 (1.5 points) The first step of CALL will take in a C file and turn it into an object file.

○ True ○ False

Q1.10 (2 points) Convert the following RISC-V machine code into its corresponding instruction. If there is an immediate value, express it in decimal form. Provide the appropriate register names, not numbers, where necessary (i.e.: `s5` instead of `x21`).

`0x0655 0F63`

Q2 *C-narios* **(7.5 points)**

Each of the following scenarios represents a bug in a program. For each of the scenarios, please indicate whether the bug is caused by an arithmetic overflow, precision loss, or another reason.

If you choose "Arithmetic Overflow" or "Precision Loss", please indicate the exact C type of the variable(s) that are involved from the following list of types:
`int8_t, uint8_t, int32_t, uint32_t, float, double`.

If you choose "Other", describe the likely bug **using at most 5 words**; no complete sentences needed.

Q2.1 (1.5 points) Completing level 255 in a game resets you to level 0.

◯ Arithmetic Overflow ◯ Precision Loss ◯ Other

Q2.2 (1.5 points) A rectangular platform that is designed to move in a sinusoid up-and-down pattern in a game slowly drifts upwards over the course of several days.

◯ Arithmetic Overflow ◯ Precision Loss ◯ Other

Q2.3 (1.5 points) A game's score counter behaves unexpectedly when the score exceeds $\approx 10^{308}$.

◯ Arithmetic Overflow ◯ Precision Loss ◯ Other

Q2.4 (1.5 points) When a program outputs a string, it infrequently prints seemingly random, corrupted characters after the expected string.

◯ Arithmetic Overflow ◯ Precision Loss ◯ Other

Q2.5 (1.5 points) After leaving a program open for several days, the program and all other programs running concurrently on your computer start to slow down.

◯ Arithmetic Overflow ◯ Precision Loss ◯ Other

Q3 `void *cf0` (7 points)

C's standard library has a built-in `qsort` function that implements the quicksort sorting algorithm. Here is an excerpt from its `man` pages.

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));

DESCRIPTION
   The qsort() function sorts an array with nmemb elements of size size.
   The base argument points to the start of the array.

   The contents of the array are sorted in ascending order according to
   a comparison function pointed to by compar, which is called with two
   arguments that point to the objects being compared.

   The comparison function must return an integer less than, equal to,
   or greater than zero if the first argument is considered to be
   respectively less than, equal to, or greater than the second. If two
   members compare as equal, their order in the sorted array is undefined.
```

Implement the `sort_matrices` function, which sorts a `list` of `matrix` structs of length `list_len` by their `size` in ascending order using the `qsort` function. You'll need to implement your own comparison function `compare_matrices` to do this. Assume appropriate C standard libraries are already imported.

```
 1 typedef struct {
 2   int **data;
 3   size_t size;
 4 } matrix;
 5
 6 int compare_matrices (const void *p, const void *q) {

 7   return _____Q3.1_____;
 8 }
 9
10 void sort_matrices (matrix *list, size_t list_len) {

11   qsort(_____Q3.2_____, _____Q3.3_____,

12         _____Q3.4_____, _____Q3.5_____);
13 }
```

Q4 *FCVT.S.WU* **(20 points)**

Implement `magnitude`, a RISC-V function, as follows:

- Input `a0`: a **nonzero** unsigned integer
- Returns in `a0`: the **index** of the most significant bit that is 1 in the binary representation of `a0`. The **least** significant bit is at index 0.

For example, `magnitude` of 2 (`0b10`) returns 1, and `magnitude` of 727 (`0b10 1101 0111`) returns 9.

Ensure that your implementation follows calling convention.

```
1 magnitude:
2     li t0 0
3     li t1 1
4 loop:
5     beq t1 a0 end

6     _____________________________________
                        Q4.1

7     _____________________________________
                        Q4.2
8     j loop
9 end:

10    _____________________________________
                        Q4.3
11    jr ra
```

Implement `convert`, a RISC-V function, as follows:

- Input `a0`: a **nonzero** unsigned integer
- Returns in `a0`: the IEEE-754 single-precision floating point representation of `a0`, **rounded down** if there is no exact representation.

For example, the integer 2 (`0x00000002`) should be converted to its corresponding floating point representation `0x40000000`. The integer 268435471 (`0x1000000F`) has no exact floating point representation; instead, the representation `0x4d800000` (rounded towards zero) should be returned.

You may assume that `magnitude` is implemented correctly and behaves as specified in the first part, regardless of your implementation above. You may not assume any specific implementation of `magnitude`, and you may not modify any `s` registers except for `s4`.

```
1  convert:
2      # prologue omitted

3      mv s4 a0
4      jal ra magnitude

5      addi t1 x0 32                          # set significand

6      _____________________________
                    Q4.4

7      _____________________________
                    Q4.5

8      _____________________________
                    Q4.6

9      addi a0 a0 _____________________       # set exponent
                      Q4.7

10     slli a0 a0 _____________________
                      Q4.8

11     _____________________________
                    Q4.9

12     # epilogue omitted
13     jr ra
```

Q4.10 (2.5 points) Which registers need to be saved in the prologue and restored in the epilogue for `convert` to satisfy calling convention?

☐ `s4` ☐ `a0` ○ None

☐ `ra` ☐ `t1`

Q5 *Pets* **(23.5 points)**

The `Pets` struct is defined as follows:

```
typedef struct {
  uint32_t count; // The number of pets in this struct
  char **names;   // An ordered list of each pet's name
} Pets;
```

The function `void add_pet(Pets *p, char *name)` is defined as follows:

- `Pets *p`: A valid pointer to a `Pets` struct.
- `char *name`: The pet's name as a properly null-terminated string.

You may assume the following:

- For any `Pets` struct, `count` is initialized to 0 and `names` is initialized to `NULL`.
- Dynamic memory allocation will never fail.
- All relevant standard libraries have been imported.

Below is an example of the behavior of `add_pet`:

```
 1 int num_cats = 3;
 2 int main () {
 3   Pets dogs = {0, NULL};
 4   char dog[] = "Harold";
 5   add_pet(&dogs, dog);
 6   dog[0] = 'D';
 7   dog[1] = 'a';
 8   dog[2] = 'v';
 9   dog[3] = 'e';
10   dog[4] = '\0';
11   add_pet(&dogs, dog);
12   printf("%d\n", dogs.count); // output is 2
13   printf("%s\n", (dogs.names)[0]); // output is Harold
14   printf("%s\n", (dogs.names)[1]); // output is Dave
15   return 0;
16 }
```

Useful C stdlib function prototypes:

```
void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t num_elements, size_t size);
void *realloc(void *ptr, size_t size);

size_t strlen(char *s);
char *strcpy(char *dest, char *src);
```

Implement `add_pet` to match the described behavior above. Note: `realloc(NULL, n)` is equivalent to `malloc(n)`.

```
1 void add_pet(Pets *p, char *name) {
2   int name_len = strlen(_____Q5.1_____);
3   p->names = _____Q5.2_____;
4   char *name_copy = _____Q5.3_____;
5   strcpy(_____Q5.4_____, _____Q5.5_____);
6   *(_____Q5.6_____) = _____Q5.7_____;
7   p->count = p->count + 1;
8 }
```

For each of the following symbols from the example, choose which section of memory it would live in.

Q5.8 (1.5 points) `num_cats` (defined on line 1)

○ Code ○ Static ○ Stack ○ Heap

Q5.9 (1.5 points) `add_pet`

○ Code ○ Static ○ Stack ○ Heap

Q5.10 (1.5 points) `dogs.names`

○ Code ○ Static ○ Stack ○ Heap

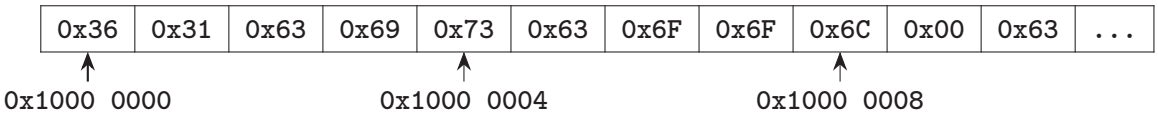
Q5.11 (1.5 points) `(dogs.names)[1]`

○ Code ○ Static ○ Stack ○ Heap

For Q5.12-Q5.14, assume we have a **big-endian** system, and the code below has been run.

```
char course[] = {'6', '1', 'c'};
uint64_t *q = (uint64_t *) course;
uint32_t *p = (uint32_t *) q;
```

`course` is located at address `0x1000 0000`. Memory starting at `0x1000 0000` is shown below:



Q5.12 (1.5 points) What is the value of `strlen(course)`?

- ○ 3
- ○ 4
- ○ 9
- ○ 10
- ○ 11
- ○ 12
- ○ None of the above
- ○ Compiler/runtime error

Q5.13 (1.5 points) What is the value of `*q`?

- ○ `0x3613 3696 3736 F6F6`
- ○ `0x3631 6369 7363 6F6F`
- ○ `0x6963 3136 6F6F 6373`
- ○ `0x6F6F 6373 6963 3136`
- ○ `0x7363 6F6F 3631 6369`
- ○ None of the above

Q5.14 (1.5 points) What is the value of `p + 1`?

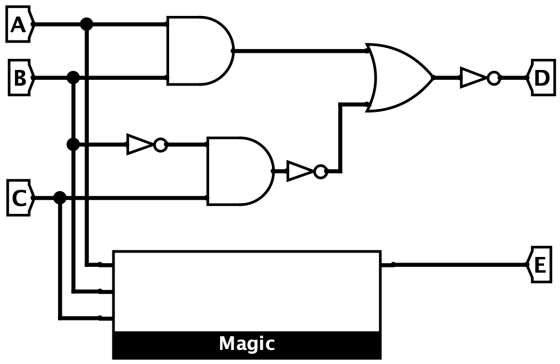
- ○ `0x3163 6973`
- ○ `0x6F6F 6373`
- ○ `0x7363 6F6F`
- ○ `0x1000 0001`
- ○ `0x1000 0004`
- ○ None of the above

This page intentionally left (mostly) blank.

The exam continues on the next page.

Q6 *Logical Logisim* **(12 points)**

Q6.1 (2 points) Given the following circuit with three inputs (A, B, and C), fill in the truth table for output D. You may ignore the Magic subcircuit and output E for Q6.1.



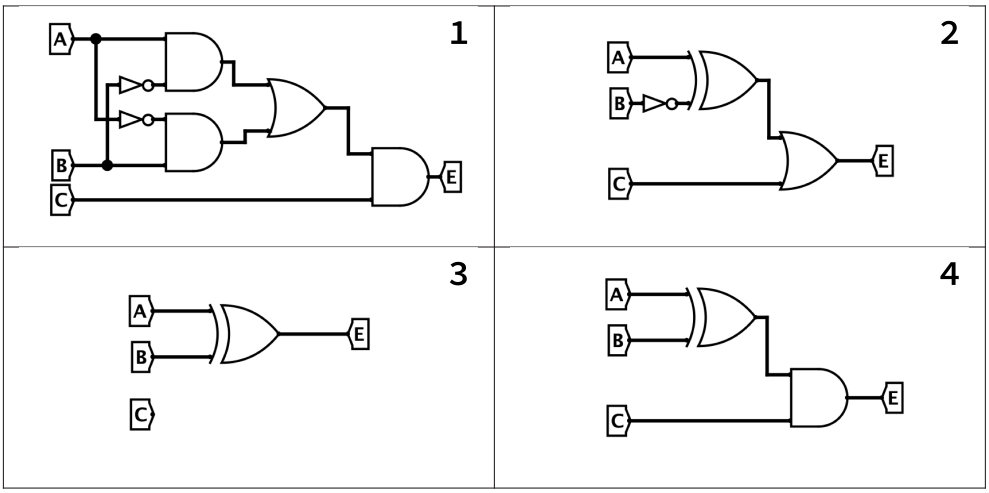
A	B	C	D	E
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0		0
1	0	1		0
1	1	0		1
1	1	1		1

Q6.2 (3 points) Referencing the truth table and circuit above, write a boolean algebra expression in terms of A, B, and C that is equivalent to the behavior of the "Magic" subcircuit (i.e output E).

For full credit, you may use at most 2 operators. You may **only** use NOT (`~`), AND (`&`), OR (`|`), and each count as one operator. We will assume standard C operator precedence, so use parentheses when uncertain.

Q6.3 (3 points) Select all circuits that are logically equivalent to the following expression. If you select "None of the above," you cannot select other options.

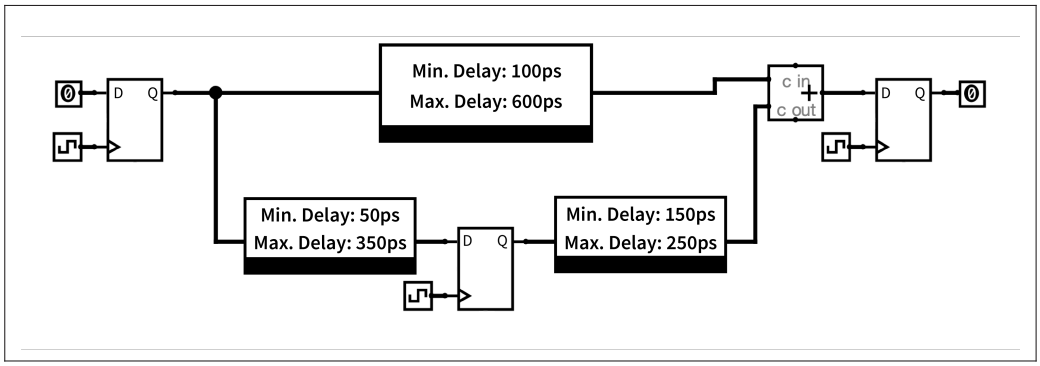
```
E = (A & ~B & C) | (~A & B & C)
```



☐ 1 ☐ 2 ☐ 3 ☐ 4

○ None of the above

Consider the following SDS circuit and combinational delays. The adder block has a delay of 50ps. Each register has a clk-to-q delay of 20ps and a setup time of 30ps. You may assume registers are triggered on the rising edge and there is only one clock signal.



Q6.4 (2 points) What is the minimum allowable clock period?

ps

Q6.5 (2 points) What is the maximum hold time for our registers in order for the circuit to have well-defined behavior?

ps

Q7 *Datapathology* **(16 points)**

For this question, refer to the RISC-V Single Cycle Datapath from the CS 61C Reference Card.

For each of the following instructions, select the proper control signals and indicate which datapath components are used. If you select "None," you cannot select other options.

`bgeu t4 t5 end` (assume branch is taken)

Q7.1 (1 point) PCSel

○ PC + 4 ○ ALU ○ Don't Care

Q7.2 (1 point) ASel

○ PC ○ RegReadData1 ○ Don't Care

Q7.3 (1 point) BSel

○ Imm ○ RegReadData2 ○ Don't Care

Q7.4 (1 point) WBSel

○ PC + 4 ○ ALU ○ Mem ○ Don't Care

Q7.5 (1 point) Datapath Components

☐ Branch Comp ☐ Imm Gen ☐ DMEM ○ None

```
auipc a0, 0x61C
```

Q7.6 (1 point) PCSel

○ PC + 4 ○ ALU ○ Don't Care

Q7.7 (1 point) ASel

○ PC ○ RegReadData1 ○ Don't Care

Q7.8 (1 point) BSel

○ Imm ○ RegReadData2 ○ Don't Care

Q7.9 (1 point) WBSel

○ PC + 4 ○ ALU ○ Mem ○ Don't Care

Q7.10 (1 point) Datapath Components

☐ Branch Comp ☐ Imm Gen ☐ DMEM ○ None

Oh no! Jedi dropped your CPU and some of your datapath components are broken. You need to figure out which types of instructions are still guaranteed to function as expected. You may ignore `ebreak` and `ecall`. If you select "None," you cannot select other options.

Example: You should only select "I-type" if all I-type instructions are guaranteed to function as expected.

Q7.11 (2 points) PCSel is always "ALU"

- ☐ R-Type
- ☐ I-Type
- ☐ S-Type
- ☐ B-Type
- ☐ U-Type
- ☐ J-Type
- ○ None

Q7.12 (2 points) ASel is always "PC"

- ☐ R-Type
- ☐ I-Type
- ☐ S-Type
- ☐ B-Type
- ☐ U-Type
- ☐ J-Type
- ○ None

Q7.13 (2 points) ImmGen always outputs `0x00000000`

- ☐ R-Type
- ☐ I-Type
- ☐ S-Type
- ☐ B-Type
- ☐ U-Type
- ☐ J-Type
- ○ None

Q8 *Coloring Book* **(0 points)**

These questions will not be assigned credit; feel free to leave them blank.

Q8.1 (0 points) What does the `FCVT.S.WU` instruction stand for?

Q8.2 (0 points) What does the `CVTSI2SS` instruction stand for?

Q8.3 (0 points) Which lecture contains a hidden animal and what was its species?

Q8.4 (0 points) If there's anything else you want us to know, or you feel like there was an ambiguity in the exam, please put it in the box below.

For ambiguities, you must qualify your answer and provide an answer for both interpretations. For example, "if the question is asking about A, then my answer is X, but if the question is asking about B, then my answer is Y". You will only receive credit if it is a genuine ambiguity and both of your answers are correct. We will only look at ambiguities if you request a regrade.