CSM 61C Spring 2020

RISC-V Programming

Exam Question Compilation

This document is a PDF version of old exam questions by topic, ordered from least difficulty to greatest difficulty.

Questions:

- Fall 2018 Quest Q4
- Fall 2019 Final Q2b
- Fall 2017 Final Q4
- Spring 2018 Midterm 1 Q6
- Fall 2019 Midterm Q4
- Summer 2018 Midterm 1 Q3
- Fall 2018 Midterm Q4
- Summer 2019 Midterm 1 Q5
- Summer 2018 Final Q3
- Spring 2019 Final Q3
- Fall 2019 Final Q4
- Fall 2018 Final Q3

Fall 2018 Quest

Q4a) Which RISC-V snippet could be the compilation of the C code: **x15 = 20 - x5**? (Select ALL that apply) *Assume the C variables **x5** and **x15** map directly to the registers of the same name.*

□ sub x5, 20, x15	□ sub x15, 20, x5	□ addi x15, x0, 20 sub x15, x15, x5	□ addi x15, x5, -20 sub x15, x0, x15

Q4b) Say we have an **int array A[99]** starting at address **0x00010000**, and register **x5** contains **&A[0]**. Assuming **sizeof(int) == 4**, what value is in register **x10** after **lw x10, 8(x5)** ? (Select ALL that apply)

□ A[2]	□ A[8]	□ &A[2]	□ &A[8]	□ 0x00010008	□ 0x00080000

Fall 2019 Final

Q2) Open to Interpretation (11 pts = 2 + 3 + 4 + 2)

Let’s consider the hexadecimal value **`0xFF000003`**. How is this data interpreted, if we treat this number as…

c) a RISC-V instruction? If there’s an immediate, write it in decimal. __________________________________	**SHOW YOUR WORK**

Q4: This question might be a RISC

Consider the following RISC code. The function **read_input** will prompt the user to provide a 32-bit integer and stores it in a0. As a reminder, the **ecall** instruction will call an OS function (determined by the ecall number stored in a0), with the value stored in a1 as the function's argument. **ecall numbers are as follows: 1 = print integer, 4 = print string, 10 = exit.**

```
.data
Boom:  .asciiz "Ayy, man's not dumb." # strlen(this string) == 20
Skraa: .asciiz "The ting goes skkkraaa." # strlen(this string) == 23

.text
MAGIC:         # prologue
               la s0, Risc-tery
               la s1, Boom
               addi s2, x0, 0x61C
Get:           jal read_input  # provide either 0 or 1 (USER_IN_1)
               beq a0, x0, Default
Risc-tery:     jal read_input  # provide any integer (USER_IN_2)
               beq a0, x0, QuickMaths # Q2
               addi t0, x0, 9
               slli t0, t0, 2
               add s0, s0, t0
               lw t1, 0(s0)
               slli a0, a0, 20   # shift user input by 20
               add t1, t1, a0
               sw t1, 0(s0)
QuickMaths:    addi a1, s1, 0
               addi a0, x0, 4
               ecall
               j Done
Default:       addi a0, x0, 1
               add a1, s2, x0
               ecall
Done:          # epilogue
               jalr ra
```

1. Consider the function MAGIC. The prologue and epilogue for this function are missing. Which registers should be saved/restored on the stack? Select all that apply.

2. Assume the assembler has been run. What machine code is the line commented Q2 `(beq a0, x0, QuickMaths)` converted to? Please write your answer in the table provided on your answer sheet.

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
0b____________	0b___________	0b__________	000	0b_________	1100011

3. Assume you call `MAGIC`, providing the input 0 (to the call to `read_input` on line 10, commented `USER_IN_1`). What does the program print? Write your answer in the blank provided.

4. Say that you call `MAGIC` again. What input values to the calls to `read_input` will print "Ayy, man's not dumb"? Remember each call to `read_input` takes in a single input value. If you select option D for part (b), then please write the exact value that the input should be.

   a) USER_IN_1:
      A. 0 B. 1 C. Not Possible

   b) USER_IN_2:
      A. Any integer B. Any nonzero integer C. 0 D. Exact value _____________

5. Assume we can both read and write to **any** valid memory address. Please specify the input values to `read_input` such that calling `MAGIC` prints out "The ting goes skkkraaa."

   a) USER_IN_1:
      A. 0 B. 1 C. Not Possible

   b) USER_IN_2:
      A. Any integer B. Any nonzero integer C. 0 D. Exact value _____________

Problem 6 *RISC-V to C Magic* (15 points)

Assume we have two arrays `input` and `result`. They are initialized as follows:

```
int *input = malloc(8*sizeof(int));
int *result = calloc(8, sizeof(int));
for (int i = 0; i < 8; i++) {
     input[i] = i;
}
```

You are given the following RISC-V code. Assume register `a0` holds the address of `input` and register `a2` holds the address of `result` when `MAGIC` is called by `main`.

```
main:
     ...
     # Start Calling MAGIC
     addi a1, x0, 8
     jal ra, MAGIC      # a0 holds input, a2 holds result
     # Checkpoint:  finished calling MAGIC
     ...
exit:
     addi a0, x0, 10
     add a1, x0, x0
     ecall      # Terminate ecall
MAGIC:
     # TODO: prologue.  What registers need to be stored onto the stack?
     mv s0, x0
     mv t0, x0
loop:
     beq t0, a1, done
     lw t1, 0(a0)
     add s0, s0, t1
     slli t2, t0, 2
     add t2, t2, a2
     sw s0, 0(t2)
     addi t0, t0, 1
     addi a0, a0, 4
     jal x0, loop
done:
     mv a0, s0
     # TODO: epilogue.  What registers need to be restored?
     jr ra
```

(a) Consider the function `MAGIC`. The prologue and epilogue for this function are missing. Which registers should be saved/restored in `MAGIC`'s prologue/epilogue? Select all that apply.

○ t0

○ t1

○ t2

○ s0

○ a0

○ a1

○ a2

○ ra

○ x0

(b) Assume you have the prologue and epilogue correctly coded. You set a breakpoint at "Checkpoint: finish calling `MAGIC`" and call `main`. What does `result` contain when your program pauses at the breakpoint? Please write the 8 numbers starting at `result` in the blanks below.

______ ______ ______ ______ ______ ______ ______ ______

(c) Translate `MAGIC` into C code. You may or may not need all of the lines provided below.

```
 // sizeof(int) == 4
int MAGIC(__________ a, __________ b, __________ c) {

     ____________________________________________________________

     ____________________________________________________________

     ____________________________________________________________

     ____________________________________________________________

     ____________________________________________________________

     ____________________________________________________________

}
```

Q4) !noitseuq V-CSIR taerg a s'ereH **(20 pts = 12 + 4 + 4)**

a) Below you will find the standard definition for a linked-list node. The recursive C code below reverses a linked list *with at least one node*. (For the initial call, the head of the list would be the first parameter, and the second parameter would be `NULL`) Your project partner translated this to nice RISC-V 32-bit code which honors the RISC-V calling conventions. Unfortunately, you spilled boba on it rendering it much of unreadable, and now you need to reconstruct it. Our solution used every line, but if you need more lines, just write them to the right of the line they're supposed to go after and put semicolons between them (like you would do in the C language). ***Don't waste time trying to understand the algorithm*** *for* `reverse`*, just compile it line-by-line.*

```
struct node_struct {
    int32_t value;
    struct node_struct *next;
}
typedef struct node_struct Node;
```

```
Node *reverse(Node *node, Node *prev) { // Requires: node != NULL
    Node *second = node->next;
    node->next  = prev;
    if (second == NULL) { return node; }
    return reverse(second, node);  }
```

```
reverse:

    lw t0, _______________________   ### Node *second = node->next;

    ______________________________   ### node->next = prev

    beq x0, t0, returnnode           ### if (second == NULL) { return node; }

    ______________________________

    ______________________________

    addi sp, sp, -4

    ______________________________

    jal ra reverse                   ### return reverse(second, node);

    ______________________________

    ______________________________

returnnode:

    ______________________________
```

Now assume all blanks above contain a single instruction (no more, no less).

b) The address of `reverse` is `0x12345678`.

What is the hex value for the machine code of `beq x0, t0, returnnode`? **0x**_____________________

c) The user adds a library and this time the address of `reverse` is `0x76543210`.

What is the hex value for the machine code of `beq x0, t0, returnnode`? **0x**_____________________

SHOW YOUR WORK FOR PART (b,c) HERE

Question 3: RISC-V Coding (20 pts)

1. Fill in the following RISC-V code so that it properly follows convention. Assume that all labels not currently in the code are external functions. You may not need all the lines provided.

```
Pro:
     ___________________________
     ___________________________
     ___________________________
     ___________________________
     ___________________________
     ___________________________
     ___________________________
Body:
     mv s1 a0
     jal ra foo
     mv s2 a0
     addi a0 x0 6
Loop:
     beq a0 x0 Epi
     addi a0 a0 -1
     mv s3 a0
     jal ra foo
     addi s2 s2 a0
     mv a0 s3
     j Loop
Epi:
     ____________________________
     ____________________________
     ____________________________
     ____________________________
     ____________________________
     ____________________________
```

2.

```
foo:
           slli  t6 a0 2
           sub   sp sp t6
           mv    t4 sp
           sw    zero 0(t4)
           addi  t1 zero 1
L1:        bge   t1 a0 Next
           andi  t2 t1 1
           slli  t3 t1 2
           add   t3 t3 t4
           sw    t2 0(t3)
           addi  t1 t1 1
           j     L1
Next:      mv    t1 zero
           mv    t2 zero
           slli  a0 a0 2
L2:        bge   t1 a0 End
           add   t3 t4 t1
           lw    t3 0(t3)
           add   t2 t2 t3
           addi  t1 t1 4
           j     L2
End:       mv    a0 t2
           add   sp sp t6
           jr    ra
```

Translate the RISC-V Assembly on the left into C code to complete the function foo:

```
unsigned foo(unsigned n) {
  ____________________________________;
  unsigned total = 0;
  unsigned *ptr = ____________________;
  ptr[0] = 0;
  for (__________________________) {
    ptr[_____] = _____________________;
  }
  for (____________________________) {
    __________________ +=  ptr[_____];
  }
  return total;
}
```

Fall 2018 Midterm

Q4) *RISC-V business: I'm in a CS61C midterm & I'm being chased by Guido the killer pimp...* (14 points)

a) Write a function in RISC-V code to return *0 if the input 32-bit float = ∞*, else a non-zero value. The input and output will be stored in **a0**, as usual. *(If you use 2 lines=3pts. 3 lines=2 pts)*

```
isNotInfinity: ________________________

               ________________________

               _____  a0, ______, ______
               ret
```

(the rest of the question deals with the code on the right) Consider the following RISC-V code run on a 32-bit machine:

```
done: li a0, 1
      ret
fun:  beq a0, x0, done
      addi sp, sp, -12
      addi a0, a0, -1
      sw ra, 8(sp)
      sw a0, 4(sp)
      sw s0, 0(sp)
      jal fun
      mv s0, a0
      lw a0, 4(sp)
      jal fun
      add a0, a0, s0
      lw s0, 0(sp)
      lw ra, 8(sp)
      addi sp, sp, 12
      ret
```

b) What is the hex value of the machine code for the underlined instruction labeled **fun**? (choose ONE)

○0xFE050EEA ○0xFE050EE3 ○0xFE050CE3 ○0xFE050FE3 ○0xFE050EFA ○0xFE050FEA

c) What is the one-line C disassembly of **fun** *with* recursion, and generates the same # of function calls:

```
uint32_t fun(uint32_t a0) { return ______________________________________________ }
```

d) What is the one-line C disassembly of **fun** that has *no* recursion (i.e., see if you can optimize it):

```
uint32_t fun(uint32_t a0) { return ______________________________________________ }
```

e) Show the call and the return value for the *largest possible value* returned by (d) above:

```
fun(__________) ⇒ _______________
```

Summer 2019 Midterm 1

Question 5: I GOT RISC-V ON IT. - 16 pts

Fill in the blanks to implement `strncpy` in RISC-V. You may not need all lines.

```
char* strncpy(char* destination, char* source, unsigned int n);
```

`strncpy` takes in two `char*` arguments and copies up to the first `n` characters from `source` into `destination`. **If it reaches a null terminator, then it copies that value into `destination` and stops copying in characters**. If there is no null terminator among the first `n` characters of `source`, the string placed in `destination` will not be null-terminated. You may not store anything on the stack for this problem.

`strncpy` returns a pointer to the `destination` string.

Hint: Assume the calling convention earned in lecture!

```
strncpy:

      add t0 x0 x0 # Current length

      ______________________________________________

Loop:

      beq _________________________________________ End

      ______________________________________________

      _______________________________(               )

      ______________________________________________

      _______________________________(               )

      ______________________________________________

      bne _________________________________________ Loop

End:

      ______________________________________________

      ______________________________________________
```

Question 3: Go With the Overflow (9 pts)

So far in RISC-V, we have not dealt with overflow exceptions. Implement the following using exactly the lines given such that if there is overflow done by the `add/addi` instruction, you **branch** to the `overflow` label where the exception handler is. Otherwise, if there is no overflow, you jump to `end`.

```
# Unsigned addition overflow
Q1:
           add  t0, t1, t2
           ______________________
           j end

# Signed addition overflow with
positive immediate
Q2:
           addi t0, t1, POS_IMM
           ______________________
           j end

# General signed addition
Q3:
           add  t0, t1, t2
           ______________________
           ______________________
           ______________________
           j end

overflow:  ...

end:       ...
```

Scratch space (not graded)

Hint: It is true that the sum should be less than one of the operands if and only if the other operand is negative

Suppose that the label `Q1` is at address 0x4000 0000. If the label `end` is at address 0x40**XY Z**800, what are all the possible values for **X**, **Y**, and **Z** such that `j end` can be resolved in the assembler? Formulate your answer in the form `[A - B]` where `A` and `B` are both hexadecimal digits.

X: ____________

Y: ____________

Z: ____________

Spring 2018 Final

Problem 3 *[MT1-3] RISCY* (14 points)

The function RISCY is known to take in two arguments, in `a0` and `a1`.

(a) Fill in the blanks such that the code below executes properly and evokes `ecall` to print the value in register `s1`. You may assume that `ecall` is a function that takes in two arguments `a0` and `a1`. When `a0` is 1, it prints the value in register `a1`.

```
RISCY: # Prologue
         _______________________
         _______________________
         _______________________
         _______________________
         _______________________

         addi s0, x0, 1
         add s1, x0, x0
Loop:    addi a0, a0, 4
         beq a1, s0, Ret
         lw t1, -4(a0)
         lw t2, 0(a0)
         sub t1, t1, t2
         bge t1, x0, Cont
         neg t1, t1
Cont:    blt t1, s1, next
         mv s1, t1
next:  # print value in s1 for debugging purpose.
         sw __________________
         sw __________________
         addi a0, x0, 1
         mv a1, s1
         ecall # ecall takes in a0(=1 for print) and a1(=register to print)
         lw __________________
         lw __________________
         addi s0, s0, 1
         j Loop
Ret      mv a0, s1
         # Epilogue
          _______________________
          _______________________
          _______________________
          _______________________
          _______________________

          jr ra
```

(b) Convert the RISCV instruction `bge t1, x0, Cont` into machine code in **binary**. Assume `mv` and `neg` expands to one instruction. Express your answer in **binary** in the fields below.

imm[12,10:5]	rs2	rs1	func3	imm[4:1,11]	opcode

(c) Translate `RISCY` into C code. You may or may not need all of the lines provided below. You can assume you have access to a new print function `printint` which takes in one argument, an integer, and prints it out:

```
void printint(int x);
```

```
__________ RISCY(__________ a0, __________ a1) {

    ______________________________________________

    ______________________________________________

    ______________________________________________

    ______________________________________________

    ______________________________________________

    ______________________________________________

    ______________________________________________

    ______________________________________________

    ______________________________________________

    ______________________________________________

}
```

Fall 2019 Final

Q4) Felix Unger must have written this RISC-V code! (30 pts = 3*10)

```
mystery:
      la t6, loop
loop: addi x0, x0, 0        ### nop
      lw   t5, 0(t6)
      addi t5, t5, 0x80
      sw   t5, 0(t6)
      addi a0, a0, -1
      bnez a0, loop
      ret
```

You are given the code above, and told that you can read and write to any word of memory without error. The function **mystery** lives somewhere in memory, but *not* at address **0x0**. Your system has no caches.

a) At a functional level, in seven words or fewer, what does **mystery(x)** do when **x < 10**?

__________ __________ __________ __________ __________ __________ __________

b) One by one, what are the values of **a0** that **bnez** sees with **mystery(13)** at every iteration? We've done the first few for you. List no more than 13; if it sees fewer than 13, write N/A for the rest.

12, 11, ______, ______, ______, ______, ______, ______, ______, ______, ______, ______, ______

c) How many times is the **bnez** instruction seen when **mystery(33)** is called before it reaches **ret** (if it ever does)? If it's infinity, write ∞. ____________

d) Briefly (two sentences max) explain your answer for part (c) above.

Fall 2018 Final

M3) *Just one more thing…RISC-V self-modifying code!* (8 points, 20 minutes) SID________________

(this is meant to be a fairly hard problem, we recommend saving it until the end of the exam…)

Your Phase I date was too late, so you can't get into the course you want. You need to hack CalCentral's server to enroll yourself! You find the following program running on the CalCentral server:

```
.data ### Starts at 0x100, strings are packed tight (not word-aligned)
    benign: .asciiz "\dev/null"
    evil:   .asciiz "/bin/sh"

.text ### Starts at 0x0
      addi t0 x0 0x100        ### Load the address of the string "\dev/null"
      addi t2 x0 '/'          ### Load the correct character. The ASCII of '/' is 47₁₀.
      jal ra, change_reg
      sb t2 0(t0)             ### Fix the backslash "\dev/null" → "/dev/null"
      addi a0 x0 0x100
      jal ra, os
```

The subroutine `change_reg` allows a user to arbitrarily set the value of any registers they choose when the function is executed (similar to the debugger on Venus). **`os(char *a0)`** runs the command at **`a0`**. *Select as few registers as necessary, set to particular values* to **MAKE THE RISC-V CODE MODIFY ITSELF** so the **`os`** function runs "**`/bin/sh`**" to hack into the CalCentral database. **Please note: even though `change_reg` can arbitrarily change any register it STILL follows the RISC-V calling convention. You CANNOT assume that the registers are initialized to zero on the launch of the program. Also, the assembler is NOT optimized.** Hint: Think about *where* the change needs to happen, then *what* it should be.

Reg	Value to set it to (in HEX without leading zeros)
□ a0	0x
□ a1	0x
□ a2	0x
□ s0	0x
□ s1	0x
□ s2	0x
□ t0	0x
□ t1	0x
□ t2	0x
□	Not Possible