CSM 61C Spring 2020 — RISC-V Instruction Formats

Exam Question Compilation

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

Questions:

- Summer 2019 Midterm 2 Q2
- Fall 2019 Final Q2c
- Spring 2018 Midterm 1 Q5
- Fall 2018 Midterm Q4
- Fall 2019 Final Q10h
- Summer 2018 Midterm 1 Q5
- Spring 2018 Midterm 2 Q10f-h
- Fall 2019 Final Q4

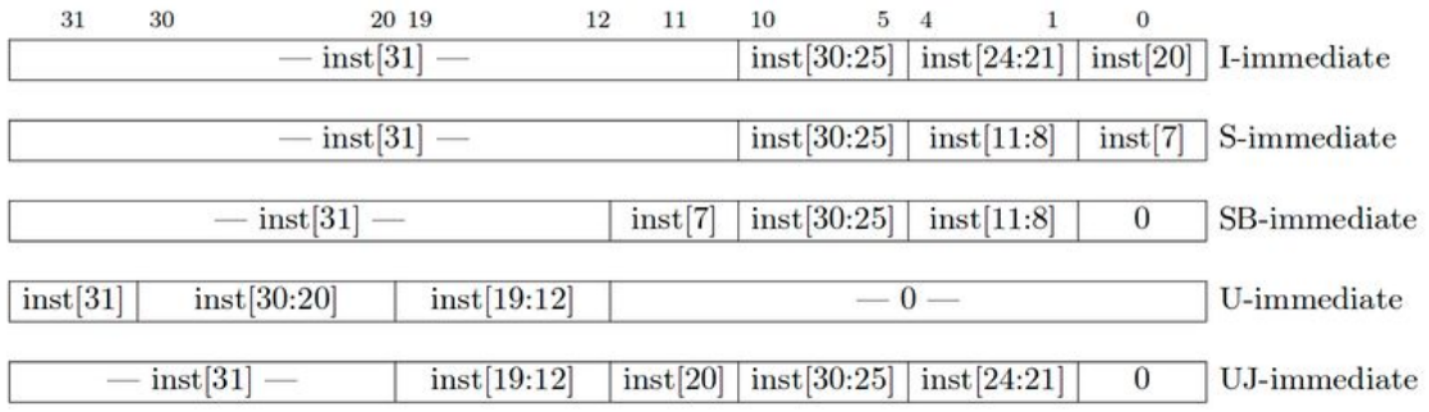
Summer 2019 Midterm 2

Question 2: ReCALL This Information (or have it written down I guess) - 16 pts

Consider the following assembly code in a file foo.s:

```
      .text
1.             mv s1 a0
2.             addi s2 s2 4
3.    Start:   beq s1 x0 End
4.             lw a0 0(s1)
5.             jal ra printf
6.             add s1 s2 s1
7.             lw s1 0(s1)
8.             jal x0 Start
9.    End:     jalr x0, ra, 0
```

Recall that immediate values are generated from instructions with the following table:



31	30 … 20	19 … 12	11	10 … 5	4 … 1	0	
— inst[31] —				inst[30:25]	inst[24:21]	inst[20]	I-immediate
— inst[31] —				inst[30:25]	inst[11:8]	inst[7]	S-immediate
— inst[31] —			inst[7]	inst[30:25]	inst[11:8]	0	SB-immediate
inst[31]	inst[30:20]	inst[19:12]	— 0 —				U-immediate
— inst[31] —		inst[19:12]	inst[20]	inst[30:25]	inst[24:21]	0	UJ-immediate

We will refer to the number produced after this process is completed as the “immediate value.”

1. Fill in all fields (or write Does Not Apply) for the machine code generated for **beq s1 x0 End** (line 3).

Immediate value: ______________________________ funct3: _________________

opcode:______________________________ funct7: _________________________

rs1: _________________ rs2: ___________________ rd: _____________________

Given the hex representation, which line number in the above program does it correspond to?

2. 0x0004A483

Line: ____________________________________

3. 0xFEDFF06F

Line: ____________________________________

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

Spring 2018 Midterm 1

Problem 5 *RISC-U ISA* (15 points)

Here are the standard 32-bit RISC-V instruction formats taught in lecture for your reference:

31	30 25	24 21	20	19 15	14 12	11 8	7	6 0	
funct7		rs2		rs1	funct3	rd		opcode	R-type
imm[11:0]				rs1	funct3	rd		opcode	I-type
imm[11:5]		rs2		rs1	funct3	imm[4:0]		opcode	S-type
imm[12]	imm[10:5]	rs2		rs1	funct3	imm[4:1]	imm[11]	opcode	SB-type
imm[31:12]						rd		opcode	U-type
imm[20]	imm[10:1]		imm[11]	imm[19:12]		rd		opcode	UJ-type

Considering the standard 32-bit RISC-V instruction formats, convert `lw t5, 17(t6)` to machine code:

(a) 0x ______________________________

Prof. Wawrzynek decides to design a new ISA for his ternary neural network accelerator. He only needs to perform 7 different operations with his ISA: XOR, ADD, LD, SW, LUI, ADDI, and BLT. He decides that each instruction should be 17 bits wide, as he likes the number 17. There are no funct7 or funct3 fields in this new ISA.

(b) What is the minimum number of bits required for the opcode field?

____

(c) Suppose Prof. Wawrzynek decides to make the opcode field 6 bits. If we would like to support instructions with 3 register fields, what is the maximum number of registers we could address?

____

(d) Given that the opcode field is 6 bits wide and each register field is 2 bits wide in the 17 bit instruction, answer the following questions:

(i) Using the assumptions stated in the description of part (d), how many bits are left for the immediate field for the instruction BLT (Assume it takes opcode, rs1, rs2, and imm as inputs)?

____

(ii) Let `n` be your answer in part (i). Suppose that BLT's branch immediate is in units of instructions (i.e. an immediate of value 1 means branching 1 instruction away). What is the maximum number of **bits** a BLT instruction can jump **forward** from the current PC using these assumptions? Write your answer in terms of `n`.

____

(iii) Using the assumptions stated in the description of part (d), what is the most negative immediate that could be used in the ADDI instruction (Assume it takes opcode, rs1, rd, and imm as inputs)?

____

(iv) For LUI, we need opcode, rd, and imm as inputs. Using the assumptions stated in the description of part (d), how many bits can we use for the immediate value?

____

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(__________) ⇒ _______________
```

Fall 2019 Final

Q10) Parallelism and Potpourri (30 pts = 6 + 4 + 3 + 3 + 3 + 3 + 4 + 4)

h) You are designing a 64-bit ISA for a simplified CPU with 3 bit-fields: `immediate | register | opcode`. You reserve enough of the rightmost bits to handle 1,500 opcodes, and enough of the leftmost bits to encode unsigned numbers up to 500 trillion. What's the greatest number of registers can you have?

SHOW YOUR WORK

SID: ________________

Question 5: RISC-V Instruction Formats (12 pts)

You are given the following RISC-V code:

```
Loop:       andi  t2 t1 1
            srli  t3 t1 1
            bltu  t1 a0 Loop
            jalr s0 s1 MAX_POS_IMM
            ...
```

1) What is the value of the **byte offset** that would be stored in the immediate field of the `bltu` instruction?

____

2) What is the binary encoding of the `bltu` instruction? Feel free to use the following space for scratch work—it will not be graded. Put your final answer in hexadecimal.

31 0

0x______________________

As a curious 61C student, you question why there are so many possible opcode, but only 47 instructions. Thus, you propose a revision to the standard 32-bit RISC-V instruction formats where **each instruction has a unique opcode (which still is 7 bits)**. You believe this justifies taking out the funct3 field from the R, I, S, and SB instructions, allowing you to allocate bits to other instruction fields **except the opcode field**.

1) What is the largest number of registers that can now be supported in hardware?

____

2) With the new register sizes, how far can a `jal` instruction jump to **(in halfwords)**?

`jal` jump range: [____________, ____________]

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?

s0 = ______________ s1 = ______________ PC = ______________

Spring 2018 Midterm 2
Questions 10f-h

(f) Complete the following RISC-V procedure `jal_address_fixing` that handles address relocation for all `jal` instructions. It first calls `find_next_jal` to find a `jal` instruction that does not yet have its offset filled in (the immediate bits are all zeroes), calculates the jump offset, and fills the immediate field of the `jal` instruction.

- Fill in **one** instruction for each of the 5 blanks.
- You can assume `jal_address_fixing` has the ability to modify text segment instruction memory.
- The function `find_next_jal` returns two values: the first is the address of a `jal` instruction stored in `a0`; the second is the address of the target instruction this `jal` instruction is jumping to stored in `a1`. If there are no more `jal` instructions to fill in offsets for, it returns 0 and 0.

Spring 2018 Midterm 2
Questions 10f-h

```
jal_address_fixing:
     jal ra, find_next_jal
     beq a0, x0, DONE

     _______________________     # set a1 as the jump offset
IMM_20:          # sets imm[20]

     _______________________
     slli a5, a5, 31      # a5 has imm[20]
IMM_19_12:     # sets imm[19:12]
     li a3, 0xFF000
     and a3, a1, a3
     or a5, a5, a3      # now a5 has imm[20] and imm[19:12]
IMM_10_1:     # sets imm[10:1]
     li a3, 0x7FE
     and a3, a1, a3
     slli a3, a3, 20
     or a5, a5, a3      # now a5 has imm[20], imm[10:1], and imm[19:12]
IMM_11:     # sets imm[11]
     li a3, 0x800
     and a3, a1, a3
     slli a3, a3, 9
     or a5, a5, a3      # now a5 has imm[20], imm[10:1], imm[11], and imm[19:12])
UPDATE:     # inserts immediate into jal instruction

     _______________________          # load the current jal instruction

     _______________________          # insert the immediate

     _______________________          # save the updated instruction
     j jal_address_fixing          # jump back to fix the next one
DONE:
     ...
```

(g) The above code works for a `jal` target address that is 2^{16} bytes smaller than the `jal` instruction address.

O True   O False

(h) The above code works for a `jal` target address that is 2^{24} bytes larger than the `jal` instruction address.

O True   O False

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.