CS61C Discussion 4 – RISC-V Addressing, Inst. Formats, and CALL

1 RISC-V Instruction Formats

1.1 Overview

Instructions in RISC-V can be turned into binary numbers that the machine actually reads. There are different formats to the instructions, based on what information is need. Each of the feilds above is filled in with binary

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

that represents the information. Each of the registers takes a 5 bit number that is the numeric name of the register (i.e. zero = 0, ra = 1, s1 = 9). See your reference card to know which register corresponds to which number.
I type instructions fill the immediate into the code. These numbers are signed 12 bit numbers.

1.2 Exercises

1. Expand `addi s0 t0 -1`
   111111111111 00101 000 01000 0010011 = 0xFFF28413

2. Expand `lw s4 5(sp)`
   000000000101 00010 010 10100 0000011 = 0x00512A03

3. Write the format name of the following instructions:

   (a) `jal` UJ
   (b) `lw` I
   (c) `beq` SB
   (d) `add` R
   (e) `jalr` I
   (f) `sb` S
   (g) `lui` U

2 RISC-V Addressing

- We have several **addressing modes** to access memory (immediate not listed):

  (a) **Base displacement addressing:** Adds an immediate to a register value to create a memory address (used for lw, lb, sw, sb)

  (b) **PC-relative addressing:** Uses the PC and adds the immediate value of the instruction (multiplied by 2) to create an address (used by branch and jump instructions)

  (c) **Register Addressing:** Uses the value in a register as a memory address (jr)

1. What is range of 32-bit instructions that can be reached from the current PC using a branch instruction?
   The immediate field of the branch instruction is 12 bits. This field only references addresses that are divisible by 2, so the immediate is multiplied by 2 before being added to the PC. Thus, the branch immediate can move the reference 2-byte instructions that are within $[-2^{11}, 2^{11} - 1]$ instructions of the current PC. The instructions we use, however, are 4 bytes so they reside at addresses that are divisible by 4 not 2. Therefore, we can only reference half as many 4-byte instructions as before, and the range of 4-byte instructions is $[-2^{10}, 2^{10} - 1]$

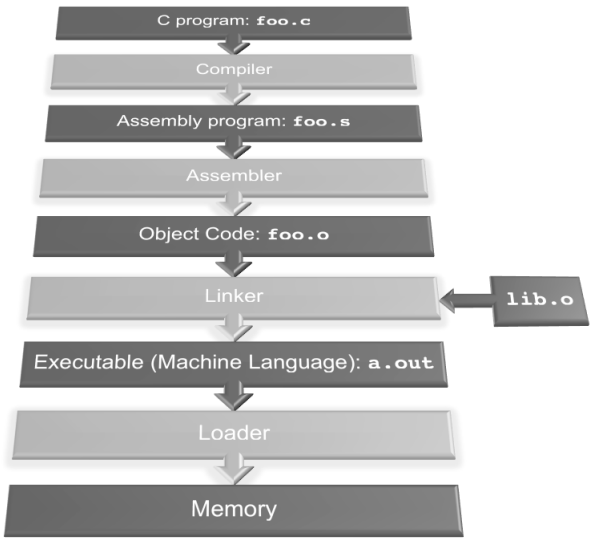
2. What is the range of 32-bit instructions that can be reached from the current PC using a jump instruction?
   The immediate field of the jump instruction is 20 bits. Similar to above, this immediate is multiplied by 2 before added to the PC to get the final address. Since the immediate is signed, the range of 2-byte instructions that can be referenced is $[-2^{19}, 2^{19} - 1]$. As we actually want the number of 4-byte instructions, we actually can reference those within $[-2^{18}, 2^{18} - 1]$ instructions of the current PC.

3. Given the following RISC-V code (and instruction addresses), fill in the blank fields for the following instructions (you'll need your RISC-V green card!).

```
0x002cff00: loop: add t1, t2, t0          |    0    |     5     |     7     |  0  |  6  |  0x33  |
0x002cff04:       jal ra, foo             |  0  |   0x14    |  0  |     0     |     1     |  0x6F  |
0x002cff08:       bne t1, zero, loop      |  1  |  0x3F  |  0  |  6  |  1  |  0xC  |  1  |  0x63  |
...
0x002cff2c: foo:  jr ra                 ra=__0x002cff08___
```

3 Compile, Assemble, Link, Load, and Go!

3.1 Overview



3.2 Exercises

a. What is the Stored Program concept and what does it enable us to do?
It is the idea that instructions are just the same as data, and we can treat them as such. This enables us to write programs that can manipulate other programs!

b. How many passes through the code does the Assembler have to make? Why?
Two, one to find all the label addresses and another to convert all instructions while resolving any forward references using the collected label addresses.

c. What are the different parts of the object files output by the Assembler?
Header: Size and position of other parts
Text: The machine code
Data: Binary representation of any data in the source file
Relocation Table: Identifies lines of code that need to be "handled" by Linker
Symbol Table: List of the files labels and data that can be referenced
Debugging Information: Additional information for debuggers

d. Which step in CALL resolves relative addressing? Absolute addressing? Assembler, Linker.

e. What does RISC stand for? How is this related to pseudoinstructions?
Reduced Instruction Set Computing. Minimal set of instructions leads to many lines of code. Pseudoinstructions are more complex instructions intended to make assembly programming easier for the coder. These are converted to TAL by the assembler.