1 RISC-V Instruction Formats

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 needed. Each of the fields above is filled in with binary 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.

| 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[31:12] | rd | opcode | U-type |

2 Addressing in RISC-V

2.1 What is range of 32-bit instructions that can be reached from the current PC using a branch instruction?

2.2 What is the range of 32-bit instructions that can be reached from the current PC using a jump instruction?

2.3 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!).

1. 0x002CFF00: loop: add t1, t2, t0  |________|________|________|________|________|__0x33__|
2. 0x002CFF04: jal ra, foo  |__________________________|_________________|__0x6F__|
3. 0x002CFF08: bne t1, zero, loop  |________|________|________|________|________|__0x63__|
4. ...
5. 0x002CFF2C: foo: jr ra  ra=___________________
3 RISC-V with Arrays and Lists

Comment what each code block does. Each block runs in isolation. Assume that there is an array, int arr[6] = {3, 1, 4, 1, 5, 9}, which starts at memory address 0xBFFFFF00, and a linked list struct (as defined below), struct ll* lst, whose first element is located at address 0xABCD0000. Let s0 contain arr’s address 0xBFFFFF00, and let s1 contain lst’s address 0xABCD0000. You may assume integers and pointers are 4 bytes and that structs are tightly packed. Assume that lst’s last node’s next is a NULL pointer to memory address 0x00000000.

struct ll {
    int val;
    struct ll* next;
}

3.1 lw t0, 0(s0)
lw t1, 8(s0)
add t2, t0, t1
sw t2, 4(s0)

3.2 loop: beq s1, x0, end
    lw t0, 0(s1)
    addi t0, t0, 1
    sw t0, 0(s1)
    lw s1, 4(s1)
    jal x0, loop
end:

3.3 add t0, x0, x0
loop: slti t1, t0, 6
   beq t1, x0, end
   slli t2, t0, 2
   add t3, s0, t2
   lw t4, 0(t3)
   sub t4, x0, t4
   sw t4, 0(t3)
   addi t0, t0, 1
   jal x0, loop
end:
4 RISC-V Calling Conventions

4.1 How do we pass arguments into functions?

4.2 How are values returned by functions?

4.3 What is sp and how should it be used in the context of RISC-V functions?

4.4 Which values need to be saved by the caller, before jumping to a function using jal?

4.5 Which values need to be restored by the callee, before using jalr to return from a function?
5 Writing RISC-V Functions

5.1 Write a function sumSquare in RISC-V that, when given an integer \( n \), returns the summation below. If \( n \) is not positive, then the function returns 0.

\[
n^2 + (n - 1)^2 + (n - 2)^2 + \ldots + 1^2
\]

For this problem, you are given a RISC-V function called square that takes in a single integer and returns its square. Implement sumSquare using square as a subroutine. Be sure to follow RISC-V caller/callee convention. (Hints: for sumSquare, in what register can we expect the parameter \( n \)? What registers should hold square’s parameter and return value? In what register should we place the return value of sumSquare? What needs to go in sumSquare’s prologue and epilogue?)
6 More Translating between C and RISC-V

translate between the RISC-V code to C. You may want to use the RISC-V Green Card on the next page as a reference. What is this RISC-V function computing? Assume no stack or memory-related issues, and assume no negative inputs.

<table>
<thead>
<tr>
<th>C</th>
<th>RISC-V</th>
</tr>
</thead>
<tbody>
<tr>
<td>Func: addi t0 x0 1</td>
<td></td>
</tr>
<tr>
<td>Loop: and t1 a1 a1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>beq t1 x0 Done</td>
</tr>
<tr>
<td></td>
<td>mul t0 t0 a0</td>
</tr>
<tr>
<td></td>
<td>addi a1 a1 -1</td>
</tr>
<tr>
<td></td>
<td>jal x0 Loop</td>
</tr>
<tr>
<td>Done: add a0 t0 x0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>jr ra</td>
</tr>
</tbody>
</table>