

1 RISC-V with Arrays and Lists

Comment each snippet with what the snippet does. Assume that there is an array, `int arr[6] = {3, 1, 4, 1, 5, 9}`, which starts at memory address `0xBFFFFFF00`, and a linked list struct (as defined below), `struct ll* lst;`, whose first element is located at address `0xABCD0000`. `s0` then contains `arr`'s address, `0xBFFFFFF00`, and `s1` contains `lst`'s address, `0xABCD0000`. You may assume integers and pointers are 4 bytes and that structs are tightly packed.

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

1. `lw t0, 0(s0)`
 `lw t1, 8(s0)`
 `add t2, t0, t1`
 `sw t2, 4(s0)`

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. `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:`

2 RISC-V Instruction Formats

2.1 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.

2.2 Exercises

1. Expand `addi s0 t0 -1`
2. Expand `lw s4 5(sp)`
3. Write the format name of the following instructions:
 - (a) `jal`
 - (b) `lw`
 - (c) `beq`
 - (d) `add`
 - (e) `jalr`
 - (f) `sb`
 - (g) `lui`

3 Translating between C and RISC-V

Translate between the C and RISC-V code. You may want to use the RISC-V Green Card as a reference. We show you how the different variables map to registers – you don't have to worry about the stack or any memory-related issues.

C	RISC-V
<pre>// Nth_Fibonacci(n): // s0 -> n, s1 -> fib // t0 -> i, t1 -> j // Assume fib, i, j are already these values int fib = 1, i = 1, j = 1; if (n==0) return 0; else if (n==1) return 1; n -= 2; while (n != 0) { fib = i + j; j = i; i = fib; n--; } return fib;</pre>	

4 RISC-V Calling Conventions

1. How do we pass arguments into functions?
2. How are values returned by functions?
3. What is `sp` and how should it be used in the context of RISC-V functions?
4. Which values need to be saved before using `jal`?
5. Which values need to be restored before using `jalr` to return from a function?