RISC-V Architecture and Programming

Davide di Trocchio

April 11, 2021

1 RISC-V Programming

1.1 Registers

1.1.1 Basic registers

Register rd, t1, t2 **t[0] ← t[1], t[2]**
Basic form of registers. May vary slightly.

add t0, t1, t2 **t[0] ← t[1] + t[2]**
Adds t1 and t2 to t0.

addi t0, t0, integer **t[0] ← t[0] + integer**
Adds an integer to a rd (t0).

sub t0, t1, t2 **t[0] ← t[1] - t[2]**
Subtracts t1 and t2 to t0.

1.1.2 Variables, Special Registers

Registers in forms of "zero" and such.

zero
Shorthand for zero value. Cannot be used in mov instruction. Check why later ex. add t2, t1, zero

1.2 Branching

Branch and jump on labels on different conditions. It follows a "format B" of bits, not like other registers we discussed earlier. Its format has rs1, rs2 and f3 like other popular registers, but it has two "c" parts of 5 and 7 bits each which act as counters to add up to the program counter. However, this counter can be interpreted by a label by the RARS compiler. :

beq t1, t2, c(label)
"Branch If Equal", if t1 == t2, jump to label.

bne t1, t2, c(label)
"Branch If Not Equal", if t1 != t2, jump to label.

1.3 Loading and Saving

Loading and saving words in memory.

lui t0, c
"Load Upper Immediate" loads in the upmost part of register c and puts 0 in all the rest. Results in 0xc0000. Loads 20 bits before.

ori t1, t2, c

"OR Immediate" Puts in t1 the OR between t2 and c. It's a bit for bit OR and puts the result in the right hand side of the register. Loads 12 bits. Can be useful to put an ori and a lui to create a full custom bit.

lw t0, c(t1) **t0 ← M[t1 + c]**

"Load Word" loads a word from the memory. It loads an address saved in memory t1, with offset c.

sw t0, c(t1) **M[t1 + c] ← t0**

"Save Word" saves a word in memory at offset t1+c. Uses s-type format, which consists of offset, rs1, f3 and an opcode.

1.4 Registers construction

First seven bits are used for the **opcode**.
The **opcode** tells the basic operation of the instruction.
The **rd** (Register destination) gets the result of the operand. In this case, it is t0.
The **funct3** selects a specific variant of the current operation.
The **funct7** Still to define. May occupy a bigger space to create immediate instructions. May contain jump instructions or general branching.
The **rs1** (Register source) is the first operand of the two registers.
The **rs2** (Register source 2) is same as before. They both take the same number of bits
This generates a very large binary string which is generally provided. Coded inside a map using a word (31, 0 bits).

1.5 Putting it all together.

1. Write a RARS program which takes in all numbers from 1 to 10 and stores them. Something like: t0 ← 1+2+3+4+5+6+7+8+9+10.

My solution:

```
addi t3, t3, 11
add t2, zero, zero
loop:
    add t1, t1, t2
    addi t2, t2, 1
    bne t2, t3, loop
```

All of this outputs 0x0000002d, which is 45, being the n-th triangular sum of 9.