RISC-V Architecture and Programming

Davide di Trocchio

April 12, 2021

1 RISC-V Programming

1.1 Registers

1.1.1 Basic registers

Register rd, t1, t2 $\mathbf{t[0] \leftarrow t[1], t[2]}$

Basic form of registers. May vary slightly.

add t0, t1, t2 $\mathbf{t[0] \leftarrow t[1] + t[2]}$

Adds t1 and t2 to t0.

addi t0, t0, integer $\mathbf{t[0] \leftarrow t[0] + integer}$

Adds an integer to a rd (t0).

sub t0, t1, t2 $\mathbf{t[0] \leftarrow t[1] - t[2]}$

Subtracts t1 and t2 to t0.

and t2, t1, t0

Performs an AND operation between t1 and t0.

or t2, t1, t0

Performs an OR operation between t1 and t0.

xor t2, t1, t0

Performs a XOR operation between t1 and t0.

f7, rs2, rs1, f3, rd, opcode. They have a slightly different type of construction called **R-Type**. Ther is no NOT in the instruction set. This is because we can build a not by simply writing a xor between a register and a 0xffff type of instruction.

andi t3, t0, 4

Performs an ANDI operation between t0 and 4, puts 4 in the immediate part of t3.

ori t3, t0, 4

Performs a ORI operation between t0 and 4, puts 4 in the immediate part of t3.

xori t3, t0, 4

Performs a XORi operation between t0 and 4, puts 4 in the immediate part of t3.

immediate, rs1, f3, rd, opcode. Their construction is called **I-type**. This is because the number passed as an argument is put in the upper part of the register.

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 <u>ex.</u> add t2, t1, zero

.word integer
Create a word (variable) and assign value integer. Each number put takes 4 byte and are being put in memory at the same location. Multiple numbers can be put inside the *.word* clause. Should always start at 0x10010000.

.string "Hello"
Puts in memory a string of characters. Each character takes 1 byte plus a special character which terminates the string.

.text
Simple label where we can put piece of code afterwards. Doesn't define a variable and starts at memory location 0x00400000.

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) $\quad \mathbf{t0 \leftarrow M[t1 + c]}$
"Load Word" loads a word from the memory. It loads an address saved in memory t1, with offset c.

sw t0, c(t1) $\quad \mathbf{M[t1 + c] \leftarrow t0}$
"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 Function calls

Function calls to the operating system.

ecall

"Exception call" are calls to specific system calls. There are some special registers used for system calls, like **a7**. Every other register is used for other function calls. They would be all registers from **a0 to a6**. Two are most used.

1 is used to print an integer.

4 is used to print a string.

10 is used to exit. Every program at the end must use this system call. In c, this is appended as *return 0*. In more modern languages this is done automatically.

Also several other can be used to do other kind of stuff. But how can one use it?

An example:

```
.data
    .string "Hello world!"
.text
    lui a0, 0x10010
    addi a7, zero, 4
    ecall
    addi a7, zero, 10
    ecall
```

Ecalls are always present after adding an integer to the system call variables. (In this case we are looking at a7.)

1.5 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).

1.6 Putting it all together.

1. Write a RARS program which takes in all numbers from 1 to 10 and stores them. Something like: t0 $\leftarrow$ 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.