

# CSC 411

## Computer Organization (Spring 2024) Lecture 13: RISC-V conditionals and loops

Prof. Marco Alvarez, University of Rhode Island

Venus Editor Simulator Chocopy

Run Step Prev Reset Dump Trace Re-assemble from Editor

| PC   | Machine Code | Basic Code        | Original Code       |
|------|--------------|-------------------|---------------------|
| 0x0  | 0x83000513   | addi x10 x0 -2000 | addi x10, x0, -2000 |
| 0x4  | 0x02A02823   | sw x10 48(x0)     | sw x10, 48(x0)      |
| 0x8  | 0x03000483   | lb x9 48(x0)      | lb x9, 48(x0)       |
| 0xc  | 0x03100403   | lb x8 49(x0)      | lb x8, 49(x0)       |
| 0x10 | 0x03104383   | lbu x7 49(x0)     | lbu x7, 49(x0)      |

Registers Memory Cache VD8

Integer (R) Floating (F)

- zero 0x00000000
- ra (x1) 0x00000000
- sp (x2) 0x7FFFFFFD
- gp (x3) 0x00000000
- tp (x4) 0x00000000
- t0 (x5) 0x00000000
- t1 (x6) 0x00000000
- t2 (x7) 0x00000000
- s0 (x8) 0xFFFFFFF8
- s1 (x9) 0x00000030
- a0 (x10) 0xFFFFF830
- a1 (x11) 0x7FFFFFFD
- a2 (x12) 0x00000000
- a3 (x13) 0x00000000
- a4 (x14) 0x00000000
- a5 (x15) 0x00000000
- a6 (x16) 0x00000000

Copy! Download! Clear!

console output

<https://venus.cs61c.org/>

### RISC-V Interpreter

Input your RISC-V code here:

```

1 addi x10 x0 -2000
2 sw x10 48(x0)
3 lb x9, 48(x0)
4 lb x8, 49(x0)
5 lbu x7, 49(x0)
6
7
8
9
10
11
12
13
14
15

```

Reset Step Run CPU: 32 Hz ▾

```

(line 1): addi x10, x0, -2000
(line 2): sw x10, 48(x0)
(line 3): lb x9, 48(x0)
(line 4): lb x8, 49(x0)

```

Features

- Reset to load the code, Step one instruction, or Run all instructions
- Set a breakpoint by clicking on the line number (only for Run)
- View registers on the right, memory on the bottom of this page

Supported Instructions

- Arithmetics: ADD, ADDI, SUB
- Logical: AND, ANDI, OR, ORI, XOR, XORI
- Sets: SLT, SLTI, SLTU, SLTIU
- Shifts: SRA, SRAI, SRL, SRLI, SLL, SLLI
- Memory: LW, SW, LB, SB
- PC: LUI, AUIPC
- Jumps: JAL, JALR
- Branches: BEQ, BNE, BLT, BGE, BLTU, BGEU

RISC-V Reference: [riscv-spec-v2.2.pdf](https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/)

<https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/>

## Practice

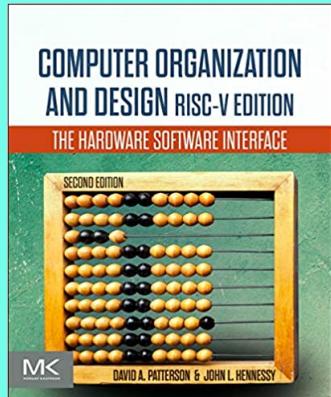
- What are the values in registers x9, x8, and x7?

**addi x10, x0, -2000**  
**sw x10, 48(x0)**  
**lb x9, 48(x0)**  
  
**lb x8, 49(x0)**  
**lbu x7, 49(x0)**

## Disclaimer

Some figures and slides are adapted from:

Computer Organization and Design (Patterson and Hennessy)  
The Hardware/Software Interface



## Larger constants

- **12-bit I-immediate** values are sufficient for most cases (range goes from -2048 to 2047)
- Using larger constants
  - **lui** (load upper immediate) is used to build 32-bit constants, placing the 20-bit U-immediate value in the top 20 bits of the destination register **rd**, filling in the lowest 12 bits with zero
  - a subsequent operation can be used to fill in the lowest 12 bits with another value

**lui rd, imm**

```
addi x1, x0, 2046      0x000007fe 0b000000000000000000000000000011
addi x2, x0, 2047      0x000007ff 0b000000000000000000000000000011
addi x3, x0, 2048      -2048     0xfffff800 0b111111111111111111111111111100
addi x4, x0, 2049      -2047     0xfffff801 0b111111111111111111111111111100
lui x5, 200000          819200000 0x30d40000 0b001100001101010000000000
addi x6, x5, 2000      819202000 0x30d407d0 0b001100001101010000000011
```

## So far ...

- Addition / subtraction

**add rd, rs1, rs2**  
**sub rd, rs1, rs2**

- Add immediate

**addi rd, rs1, imm**

- Load / store

**lw rd, imm(rs1)**  
**sw rs2, imm(rs1)**

## Logical operations

- Instructions for bitwise manipulation

| Logical operations     | C operators | Java operators | RISC-V instructions |
|------------------------|-------------|----------------|---------------------|
| Shift left             | <<          | <<             | sll, slli           |
| Shift right            | >>          | >>>            | srl, srli           |
| Shift right arithmetic | >>          | >>             | sra, srai           |
| Bit-by-bit AND         | &           | &              | and, andi           |
| Bit-by-bit OR          |             |                | or, ori             |
| Bit-by-bit XOR         | ^           | ^              | xor, xori           |
| Bit-by-bit NOT         | ~           | ~              | xori                |

# Conditionals and loops

## Branches

### Conditional branch

- change control to a labeled instruction if condition is **true**
- beq, bne, blt, bge, bltu, bgeu**
- example — branch to L1 if rs1 equals to rs2:

```
beq rs1, rs2, L1
```

### Unconditional branch

- change control unconditionally
- j** (jump) — note this is a **pseudo-instruction**, not an instruction
- example — branch to L1 unconditionally

```
j L1
```

## Labels

- A label is an identifier to a particular line of code
  - does not count as code, it is just a point of reference
- Labels are defined with unique names
  - similar to variable names
- RISC-V has instructions that can “jump” to specific labels
  - conditional and unconditional branches

```
main:  
# ... instructions  
bne x22, x23, label1  
add x19, x20, x21  
beq x0, x0, label2  
  
label1:  
sub x19, x20, x21  
  
label2:  
addi x19, x19, 1
```

## Branch instructions

|                         |                                      |                  |                            |  |
|-------------------------|--------------------------------------|------------------|----------------------------|--|
| Conditional<br>branch   | Branch if equal                      | beq x5, x6, 100  | if (x5 == x6) go to PC+100 | PC-relative branch if registers equal                      |
|                         | Branch if not equal                  | bne x5, x6, 100  | if (x5 != x6) go to PC+100 | PC-relative branch if registers not equal                  |
|                         | Branch if less than                  | blt x5, x6, 100  | if (x5 < x6) go to PC+100  | PC-relative branch if registers less                       |
|                         | Branch if greater or equal           | bge x5, x6, 100  | if (x5 >= x6) go to PC+100 | PC-relative branch if registers greater or equal           |
|                         | Branch if less, unsigned             | bltu x5, x6, 100 | if (x5 < x6) go to PC+100  | PC-relative branch if registers less, unsigned             |
|                         | Branch if greater or equal, unsigned | bgeu x5, x6, 100 | if (x5 >= x6) go to PC+100 | PC-relative branch if registers greater or equal, unsigned |
|                         | Jump and link                        | jal x1, 100      | x1 = PC+4; go to PC+100    | PC-relative procedure call                                 |
| Unconditional<br>branch | Jump and link register               | jalr x1, 100(x5) | x1 = PC+4; go to x5+100    | Procedure return; indirect call                            |

## Pseudo-instructions

- RISC-V uses pseudo-instructions alongside real instructions
  - provide shorthand syntax for common operations
  - pseudo-instructions are interpreted by the assembler and translated into actual RISC-V instructions
  - some examples:

| pseudo-instruction | equivalent RISC-V instruction   |
|--------------------|---|
| <b>not</b> rd, rs1 | <b>xori</b> rd, rs1, -1   |
| <b>mv</b> rd, rs1  | <b>addi</b> rd, rs1, 0  |
| <b>nop</b>         | <b>addi</b> x0, x0, 0   |
| <b>li</b> rd, imm  | translates into one ( <b>addi</b> ) or two ( <b>lui</b> , <b>addi</b> ) real instructions depending on the magnitude of the immediate value |

## Practice

```
// assume f, g, h, i, j are in  
// x19, x20, ...  
if (i == j) {  
    f = g + h;  
} else {  
    f = g - h;  
}
```

### main:

```
# ... instructions  
bne x22, x23, label1  
add x19, x20, x21  
beq x0, x0, label2  
label1:  
    sub x19, x20, x21  
label2:  
    # ... instructions
```

## Practice

# what is the value of x2?

```
addi x1, x0, 1  
slli x1, x1, 5  
addi x2, x0, 10  
blt x1, x2, if  
beq x0, x0, done  
if:  
    addi x1, x1, 1  
done:  
    addi x2, x2, 1
```

## Signed vs unsigned

- Signed comparison
  - **blt**, **bge**
- Unsigned comparison
  - **bltu**, **bgeu**
- Example

```
# assume x22 stores 0xFFFFFFFF  
# assume x23 stores 0x00000001  
# which instruction branches?  
blt x22, x23, Label  
bltu x22, x23, Label
```

# Loops

- Conditional branches are key to writing loops in RISC-V
  - multiple ways of writing a loop

```
# assume x1 holds the value 10 and x2  
# is zero, what is the value of x2?  
loop:  
    bge x0, x1, done  
    addi x1, x1, -1  
    addi x2, x2, 2  
    beq x0, x0, loop  
done:
```

# Practice

```
// assume i in x22, k in x24  
// base address of save in x25  
while (save[i] == k) {  
    i += 1;  
}  
  
main:  
    # ... instructions  
label3:  
    slli x10, x22, 2  
    add x10, x10, x25  
    lw x9, 0(x10)  
    bne x9, x24, label4  
    addi x22, x22, 1  
    beq x0, x0, label3  
label4:  
    # ... instructions
```

# Practice

```
// assume a, b, c, d  
// are in x1, x2, x3, x4  
// base address of data in x5  
do {  
    a = a + data[c];  
    c = c + d;  
} while (c != b);
```

# Practice

```
# translate the following loop to C  
# assume that the C-level integer `i` is  
# in register x1, x4 holds the C-level  
# integer `result`, and x2 holds the  
# base address of the integer `MemArray`  
    addi x1, x0, 0  
    addi x3, x0, 100  
loop:  
    lw x5, 0(x2)  
    add x4, x4, x5  
    addi x2, x2, 4  
    addi x1, x1, 1  
    blt x1, x3, loop
```

# Practice

```
// assume a, b, c, v are
// in x1, x2, x3, x5
switch (v) {
    case 0:
        a = b + c;
        break;
    case 1:
        a = b - c;
        break;
    case 2:
        a = b * c;
        break;
}
```

## Summary of instructions

A few standard operations do not appear in the table:

- **not** (exists as a pseudo-instruction)
- **multiplication/ division/mod** (exist as an extension to the base RISC-V)
- **floating point arithmetic** (exist as an extension)

| RISC-V operands              |  |  |                                     |
|------------------------------|--|--|-------------------------------------|
| Name                         | Example  | Comments   |                                     |
| 32 registers                 | x0 - x31   | Fast locations for data. In RISC-V, data must be in registers to perform arithmetic. Register x0 always equals 0.  |                                     |
| 2 <sup>30</sup> memory words | Memory[0], Memory[4], ...<br>Memory[4,294,967,292] | Accessed only by data transfer instructions. RISC-V uses byte addresses, so sequential word accesses differ by 4. Memory holds data structures, arrays, and spilled registers. |                                     |
| RISC-V assembly language     |  |  |                                     |
| Category                     | Instruction  | Example  | Meaning                             |
| Arithmetic                   | Add  | add x5, x6, x7   | x5 = x6 + x7                        |
|                              | Subtract   | sub x5, x6, x7   | x5 = x6 - x7                        |
|                              | Add immediate                                      | addi x5, x6, 20  | x5 = x6 + 20                        |
|                              | Load word  | lw x5, 40(x6)  | x5 = Memory[x6 + 40]                |
|                              | Load word, unsigned                                | lwu x5, 40(x6)   | x5 = Memory[x6 + 40]                |
|                              | Store word   | sw x5, 40(x6)  | Memory[x6 + 40] = x5                |
|                              | Load halfword                                      | lh x5, 40(x6)  | x5 = Memory[x6 + 40]                |
|                              | Load halfword, unsigned                            | lhu x5, 40(x6)   | x5 = Memory[x6 + 40]                |
|                              | Store halfword                                     | sh x5, 40(x6)  | Memory[x6 + 40] = x5                |
|                              | Load byte  | lb x5, 40(x6)  | x5 = Memory[x6 + 40]                |
| Data transfer                | Load byte, unsigned                                | lbu x5, 40(x6)   | x5 = Memory[x6 + 40]                |
|                              | Store byte   | sb x5, 40(x6)  | Memory[x6 + 40] = x5                |
|                              | Load reserved                                      | lr,d x5, (x6)  | x5 = Memory[x6]                     |
|                              | Store conditional                                  | sc,d x7, x5, (x6)  | Memory[x6] = x5; x7 = 0/1           |
|                              | Load upper immediate                               | lui x5, 0x123456   | x5 = 0x12345600                     |
|                              | And  | and x5, x6, x7   | x5 = x6 & x7                        |
|                              | Inclusive or                                       | or x5, x6, x7  | x5 = x6   x7                        |
|                              | Exclusive or                                       | xor x5, x6, x9   | x5 = x6 ^ x9                        |
|                              | And immediate                                      | andi x5, x6, 20  | x5 = x6 & 20                        |
|                              | Inclusive or immediate                             | ori x5, x6, 20   | x5 = x6   20                        |
| Logical                      | Exclusive or immediate                             | xori x5, x6, 20  | x5 = x6 ^ 20                        |
|                              | Shift left logical                                 | sll x5, x6, x7   | x5 = x6 << x7                       |
|                              | Shift right logical                                | srl x5, x6, x7   | x5 = x6 >> x7                       |
|                              | Shift right arithmetic                             | sra x5, x6, x7   | x5 = x6 >> x7                       |
|                              | Shift left logical immediate                       | slli x5, x6, 3   | x5 = x6 << 3                        |
|                              | Shift right logical immediate                      | srai x5, x6, 3   | x5 = x6 >> 3                        |
|                              | Shift right arithmetic immediate                   | srai x5, x6, 3   | x5 = x6 >> 3                        |
|                              | Shift left by register                             |  | Shift left by register              |
|                              | Shift right by register                            |  | Shift right by register             |
|                              | Arithmetic shift right by register                 |  | Arithmetic shift right by register  |
| Shift                        | Shift left by immediate                            |  | Shift left by immediate             |
|                              | Shift right by immediate                           |  | Shift right by immediate            |
|                              | Arithmetic shift right by immediate                |  | Arithmetic shift right by immediate |