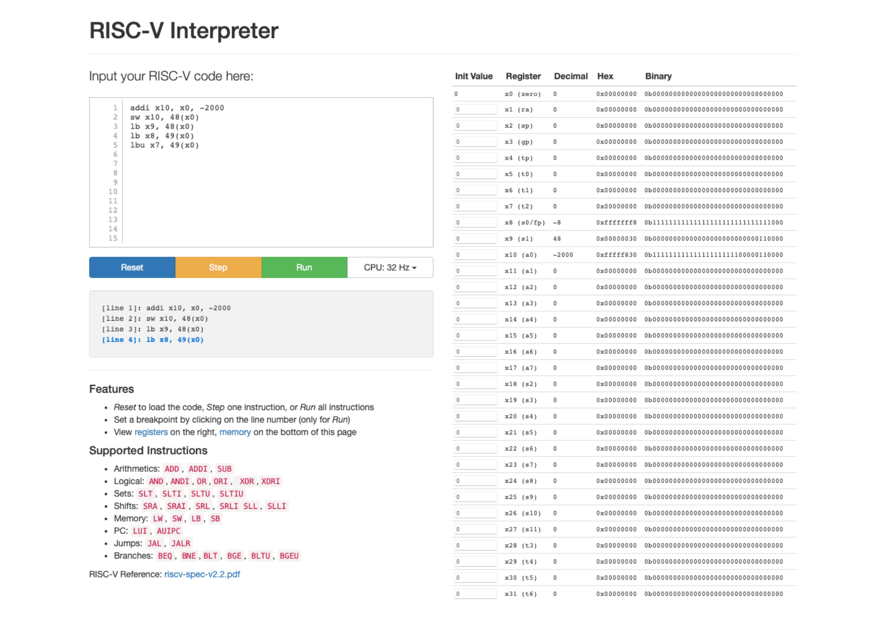
CSC 411

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

Prof. Marco Alvarez, University of Rhode Island



RISC-V Interpreter

Input your RISC-V code here:

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

```
[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

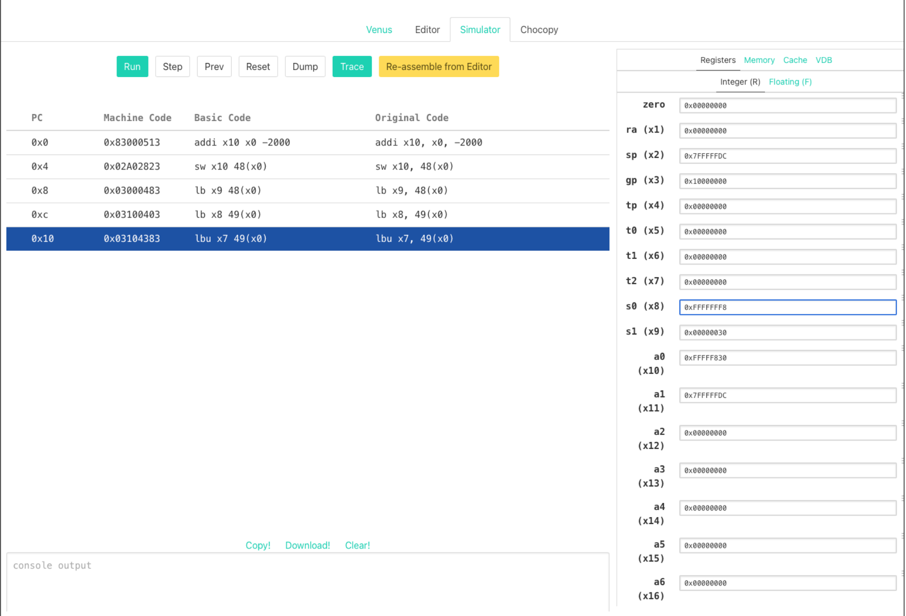
Supported Instructions

- 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

Init Value	Register	Decimal	Hex	Binary
0	x0 (zero)	0	0x00000000	0b00000000000000000000000000000000
0	x1 (ra)	0	0x00000000	0b00000000000000000000000000000000
0	x2 (sp)	0	0x00000000	0b00000000000000000000000000000000
0	x3 (gp)	0	0x00000000	0b00000000000000000000000000000000
0	x4 (tp)	0	0x00000000	0b00000000000000000000000000000000
0	x5 (t0)	0	0x00000000	0b00000000000000000000000000000000
0	x6 (t1)	0	0x00000000	0b00000000000000000000000000000000
0	x7 (t2)	0	0x00000000	0b00000000000000000000000000000000
0	x8 (s0/fp)	-8	0xfffffff8	0b11111111111111111111111111111000
0	x9 (s1)	48	0x00000030	0b00000000000000000000000000110000
0	x10 (a0)	-2000	0xfffff830	0b11111111111111111111100000110000
0	x11 (a1)	0	0x00000000	0b00000000000000000000000000000000
0	x12 (a2)	0	0x00000000	0b00000000000000000000000000000000
0	x13 (a3)	0	0x00000000	0b00000000000000000000000000000000
0	x14 (a4)	0	0x00000000	0b00000000000000000000000000000000
0	x15 (a5)	0	0x00000000	0b00000000000000000000000000000000
0	x16 (a6)	0	0x00000000	0b00000000000000000000000000000000
0	x17 (a7)	0	0x00000000	0b00000000000000000000000000000000
0	x18 (s2)	0	0x00000000	0b00000000000000000000000000000000
0	x19 (s3)	0	0x00000000	0b00000000000000000000000000000000
0	x20 (s4)	0	0x00000000	0b00000000000000000000000000000000
0	x21 (s5)	0	0x00000000	0b00000000000000000000000000000000
0	x22 (s6)	0	0x00000000	0b00000000000000000000000000000000
0	x23 (s7)	0	0x00000000	0b00000000000000000000000000000000
0	x24 (s8)	0	0x00000000	0b00000000000000000000000000000000
0	x25 (s9)	0	0x00000000	0b00000000000000000000000000000000
0	x26 (s10)	0	0x00000000	0b00000000000000000000000000000000
0	x27 (s11)	0	0x00000000	0b00000000000000000000000000000000
0	x28 (t3)	0	0x00000000	0b00000000000000000000000000000000
0	x29 (t4)	0	0x00000000	0b00000000000000000000000000000000
0	x30 (t5)	0	0x00000000	0b00000000000000000000000000000000
0	x31 (t6)	0	0x00000000	0b00000000000000000000000000000000

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



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)

zero	0x00000000
ra (x1)	0x00000000
sp (x2)	0x7FFFFFDC
gp (x3)	0x10000000
tp (x4)	0x00000000
t0 (x5)	0x00000000
t1 (x6)	0x00000000
t2 (x7)	0x00000000
s0 (x8)	0xFFFFFFF8
s1 (x9)	0x00000030
a0 (x10)	0xFFFFF830
a1 (x11)	0x7FFFFFDC
a2 (x12)	0x00000000
a3 (x13)	0x00000000
a4 (x14)	0x00000000
a5 (x15)	0x00000000
a6 (x16)	0x00000000

https://venus.cs61c.org/

Practice

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

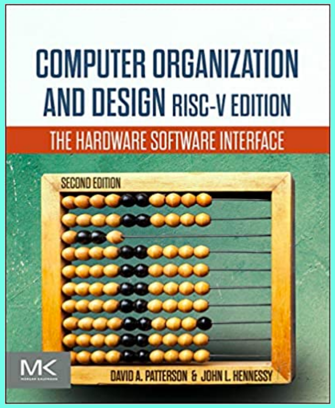
```
addi x10, x0, -2000
sw x10, 48(x0)
lb x9, 48(x0)

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


COMPUTER ORGANIZATION AND DESIGN RISC-V EDITION
THE HARDWARE SOFTWARE INTERFACE
SECOND EDITION
MK
DAVID A. PATTERSON & JOHN L. HENNESSY

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)
```

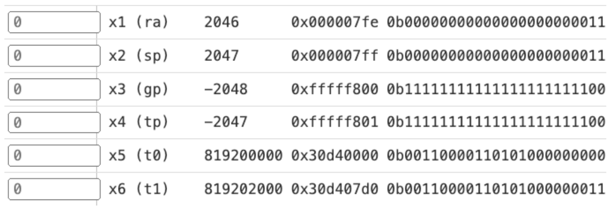
Larger constants

- Most constants are small
  - **12-bit immediate field** is sufficient
- Using larger constants (32 bits)

```
lui  rd,  constant
```

  - copies a **20-bit** constant to bits `[31:12]` of `rd` and clears remaining bits to 0

```
addi x1, x0, 2046
addi x2, x0, 2047
addi x3, x0, 2048
addi x4, x0, 2049

lui x5, 200000
addi x6, x5, 2000
```



```
0   x1 (ra)   2046      0x000007fe 0b000000000000000000000011
0   x2 (sp)   2047      0x000007ff 0b000000000000000000000011
0   x3 (gp)   -2048     0xfffff800 0b111111111111111111111100
0   x4 (tp)   -2047     0xfffff801 0b111111111111111111111100
0   x5 (t0)   819200000 0x30d40000 0b001100001101010000000000
0   x6 (t1)   819202000 0x30d407d0 0b001100001101010000000011
```

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

Branches

- A branch is a change of control flow
  - conditional branch
    - change control to a labeled instruction if a condition is `true`
    - `beq, bne, blt, bge, bltu, bgeu`

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

  - unconditional branch
    - change control unconditionally
    - `j` (jump)

Branch instructions

Conditional 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
Unconditional branch	Jump and link	`jal x1, 100`	`x1 = PC+4; go to PC+100`	PC-relative procedure call
	Jump and link register	`jalr x1, 100(x5)`	`x1 = PC+4; go to x5+100`	Procedure return; indirect call

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

```
// 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
```

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
```

Practice

Practice

Practice

Practice

Practice