# RSIC-V: Immediate and Control Flow

Caiwen Ding
Department of Computer Science and Engineering
University of Connecticut

CSE3666: Introduction to Computer Architecture

## Outline

- Immediate operands
- Control flows
  - If-Else
  - Loops: while, for, …

Reading: Sections 2.3 and 2.7. Skip memory operand in 2.3

References: Reference card in textbook

## Question

We have learned registers in RISC-V and add/sub instructions

- How do we count?

```
t0 = t0 + 1
```

addi t0, t0, 1

add t2, t0, t1

zero

## Immediate Operands

- Constant data can be specified in some instructions

```
addi s3, s3, 4          # increment s3 by 4
```

- No subtract immediate instruction
  - Just use a negative constant

```
addi s2, s1, -1         # s2 = s1 - 1
```

- The immediate must be in the range [-2048, 2047]
- **Design Principle 3**: Make the common case fast
  - Small constants are common

## The Constant Zero

- Register 0 (zero) is the constant 0
  - Cannot be overwritten
  - If you need 0, it is already in x0. No need to use another register

zero

- Useful for common operations
  - Move value between registers

```
add t2, s1, zero
```

✓

3

  - Load an immediate into a register

```
addi t2, x0, 100
addi t2, x0, -20
```

## Question

- How do we calculate the absolute value of an integer? If?
- How do we calculate the sum of integers from 0 to 99? for-loop?

The execution of instructions is normally sequential.

How does a processor support selection and repetition?

## Branches

- Conditional branches
  - If a condition is true, go to the instruction indicated by the label
  - Otherwise, continue sequentially

branch Equal → Source Register

```
beq rs1, rs2, L1 # if (rs1 == rs2) goto L1
bne rs1, rs2, L2 # if (rs1 != rs2) goto L2
```

branch Not Equal

less than

```
blt rs1, rs2, L3 # if (rs1 <  rs2) goto L3
bge rs1, rs2, L4 # if (rs1 >= rs2) goto L4
```

great Equal

```
# example of a label
L1:     ADD  x1, x2, x3
```

Branches compare two registers! Not with an immdiate

## If Statements

```
if (i == j)
{
        f = g + h;
}
```

| Variable | Register |
|---|---|
| f | s0 |
| g | s1 |
| h | s2 |
| i | s3 |
| j | s4 |

Pseudocode

```
        if (i != j) goto Skip
        f = g + h   (if i == j)
Skip:  # this is a label
```

## If Statements

Pseudocode

```
    if (i != j) goto Skip
    f = g + h
Skip:
```

| Variable | Register |
|---|---|
| f | s0 |
| g | s1 |
| h | s2 |
| i | s3 |
| j | s4 |

RISC-V code

```
    bne  s3, s4, Skip     s3!=s4
    add  s0, s1, s2  ( s3==s4)
    # more instructions if necessary
Skip:
```

## If-else Statements

```
if (i == j)
        f = g + h;
else
        f = g - h;
```

| Variable | Register |
|---|---|
| f | s0 |
| g | s1 |
| h | s2 |
| i | s3 |
| j | s4 |

## If-else Statements - 2

```
if (i == j)
         f = g + h;
else
         f = g - h;
```

| Variable | Register |
|---|---|
| f | s0 |
| g | s1 |
| h | s2 |
| i | s3 |
| j | s4 |

Pseudocode

```
         if (i != j) goto Else
         f = g + h
         goto EndIf   ← Do not forget to skip the else branch
Else:
         f = g - h
EndIf:
```

## If-else Statements - 3

| Variable | Register |
|---|---|
| f | s0 |
| g | s1 |
| h | s2 |
| i | s3 |
| j | s4 |

Pseudocode

```
        if (i != j) goto Else
        f = g + h
        goto EndIf
Else:
        f = g - h
                sub
EndIf:
```

RISC-V code

```
        bne   s3, s4, Else (s3!=s4)
        add   s0, s1, s2 ← (s3==s4)
        beq   x0, x0, EndIf        # don't forget
  Else: sub   s0, s1, s2
  EndIF: …
```

## While Loop

```
      while (cond) {
            Statements
      }
```

```
# Method 1
Loop:
      if (! cond) goto Exit
      Statements
      goto Loop
Exit:
```

```
# Method 2
      goto Test
Loop:
      Statements
Test:
      if (cond) goto Loop
```

## Question

Implement the following loop with RISC-V instructions.

```
sum = 0;

i = 0;

while (i < 100) {

        sum += i;

        i += 1;

}
```

| Variable | Register |
|---|---|
| `i` | `s0` |
| `sum` | `s1` |
| `end` | `s2` |

## While – Method 1

i 0 1 2 --- 99 100

```
# Pseudocode

        i = 0
        sum = 0
loop:
        if (!(i < 100)) goto exit
        sum += i
        i += 1
        goto loop
exit:
```

```
# Method 1 RISC-V code

        addi    s0, x0, 0       s0=0
        addi    s1, x0, 0       s1=0
        addi    s2, x0, 100     s2=100
loop:   bge     s0, s2, exit    101
        add     s1, s1, s0      s1=s1+s0
        addi    s0, s0, 1       s0=s0+1
        beq     x0, x0, loop
exit:
```

3

4×100

zero zero

| Variable | Register |
|---|---|
| i | s0 |
| sum | s1 |
| end | s2 |

4×100+3+1

404

How many instructions are executed?

## While – Method 2

i = 0, 1, 2, ---, 99, 100

```
# Pseudocode

        i = 0
        sum = 0

        goto test
loop:   sum += i
        i += 1
test:   if (i < 100) goto loop
```

```
# Method 2 RISC-V code

        addi   s0, x0, 0
        addi   s1, x0, 0
        addi   s2, x0, 100
        beq    x0, x0, test
loop:   add    s1, s1, s0
        addi   s0, s0, 1
test:   blt    s0, s2, loop
```

4

100x3

4+100x3+1=305

| Variable | Register |
|---|---|
| i | s0 |
| sum | s1 |
| end | s2 |

How many instructions are executed?

## For Loop

```
sum = 0;
for (i = 0; i < 100; i += 1) {
        sum += i;
}
```

Convert a for loop to a while loop:

```
sum = 0;
i = 0;
while (i < 100) {
        sum += i;
        i += 1;
}
```

Do not forget to increment the loop control variable.

## Design question

- Can `beq, bne, blt, bge` compare values in all the ways software developers need? Do we need to add more instructions for ">" and "<="?

$<$ $\geq$

$>$ $\leq$

$=$ $\neq$

~~bgt~~ s0 s1, 4

blt s1, s0, 4

~~ble~~ s0, s1, L1

bge ~~s4~~ s0, L2

How do we do the following in RISC-V program?

```
if s1 <= s2 goto L1
```

## Pseudoinstructions

- Pseudoinstructions are fake instructions
  - Do not add instructions for the operations that can be done
- Purpose: Make it easier for programmers to write assembly code
  - Assembler converts pseudoinstructions to real instructions

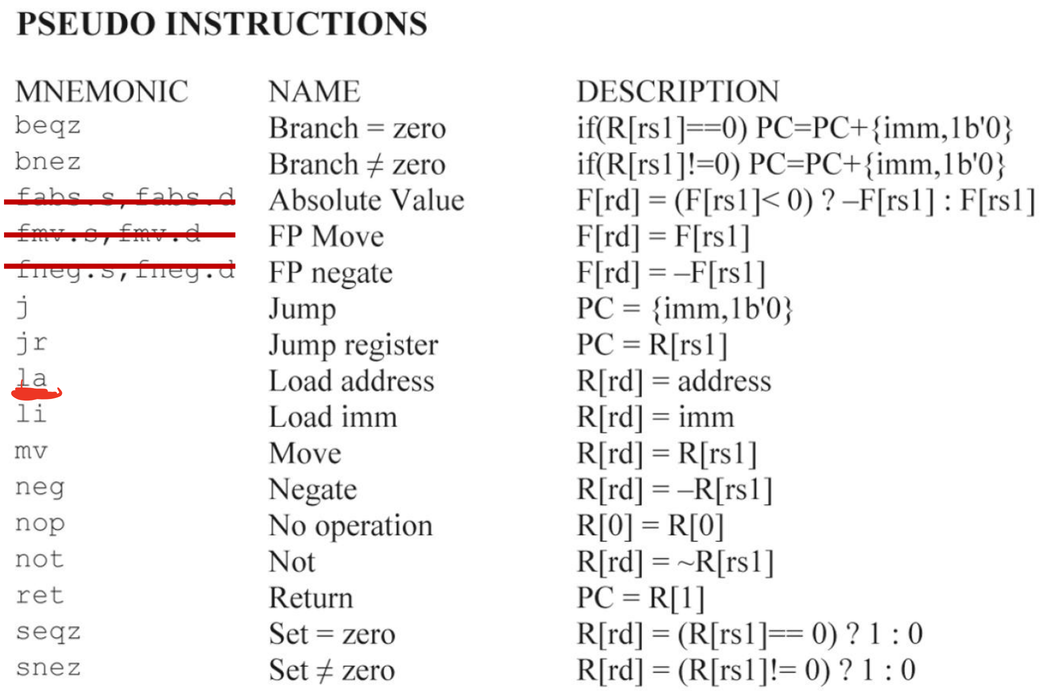
Destination Register

| Pseudoinstruction | Real Instructions |
|---|---|
| `nop` no operation | `addi  x0, x0, 0` |
| `mv` move `rd, rs` | `addi  rd, rs, 0` |
| `neg` negate `rd, rs` | `sub   rd, x0, rs` (-rs) |
| `li    rd, immd` load Immd | `addi  rd, x0, immd` <br> `Or more than one instruction` * |

* Depending on immd. If immd is small, use only one instruction.

## Policy on pseudoinstructions

- We can use the pseudoinstructions listed on the (green) card
- We also need to recognize pseudoinstructions and be able to convert them to real instructions



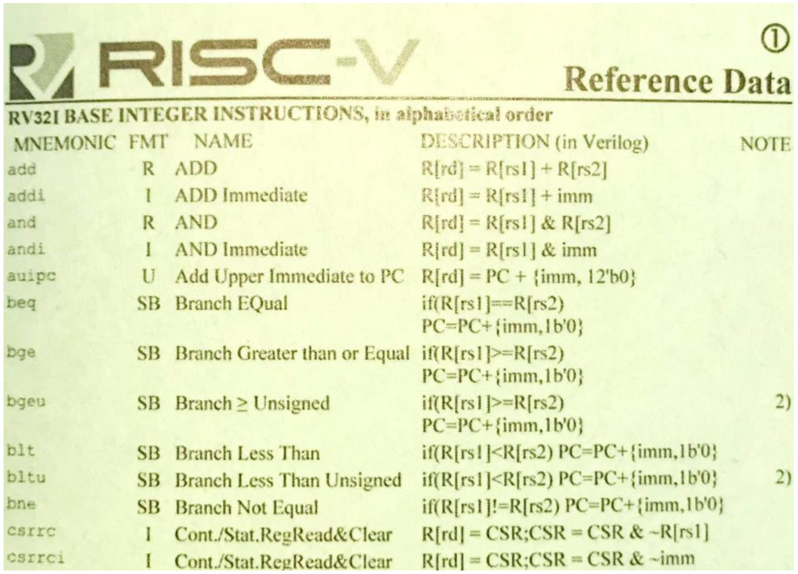
**PSEUDO INSTRUCTIONS**

| MNEMONIC | NAME | DESCRIPTION |
|---|---|---|
| `beqz` | Branch = zero | if(R[rs1]==0) PC=PC+{imm,1b'0} |
| `bnez` | Branch ≠ zero | if(R[rs1]!=0) PC=PC+{imm,1b'0} |
| ~~`fabs.s,fabs.d`~~ | Absolute Value | F[rd] = (F[rs1]< 0) ? –F[rs1] : F[rs1] |
| ~~`fmv.s,fmv.d`~~ | FP Move | F[rd] = F[rs1] |
| ~~`fneg.s,fneg.d`~~ | FP negate | F[rd] = –F[rs1] |
| `j` | Jump | PC = {imm,1b'0} |
| `jr` | Jump register | PC = R[rs1] |
| `la` | Load address | R[rd] = address |
| `li` | Load imm | R[rd] = imm |
| `mv` | Move | R[rd] = R[rs1] |
| `neg` | Negate | R[rd] = –R[rs1] |
| `nop` | No operation | R[0] = R[0] |
| `not` | Not | R[rd] = ~R[rs1] |
| `ret` | Return | PC = R[1] |
| `seqz` | Set = zero | R[rd] = (R[rs1]== 0) ? 1 : 0 |
| `snez` | Set ≠ zero | R[rd] = (R[rs1]!= 0) ? 1 : 0 |

## What instructions do I have to learn?

- We won't discuss every instruction in detail → Exam ✓ ☆☆☆☆☆
- You should study the RV32I instructions listed on the green card, except for the following:
  - FENCE, FENCE.I, and instructions starting with CS
- We will cover some instructions in M and F/D extensions later

Exam? a little → Exam X



RISC-V ① Reference Data

RV32I BASE INTEGER INSTRUCTIONS, in alphabetical order

| MNEMONIC | FMT | NAME | DESCRIPTION (in Verilog) | NOTE |
|---|---|---|---|---|
| add | R | ADD | R[rd] = R[rs1] + R[rs2] | |
| addi | I | ADD Immediate | R[rd] = R[rs1] + imm | |
| and | R | AND | R[rd] = R[rs1] & R[rs2] | |
| andi | I | AND Immediate | R[rd] = R[rs1] & imm | |
| auipc | U | Add Upper Immediate to PC | R[rd] = PC + {imm, 12'b0} | |
| beq | SB | Branch EQual | if(R[rs1]==R[rs2) PC=PC+{imm,1b'0} | |
| bge | SB | Branch Greater than or Equal | if(R[rs1]>=R[rs2) PC=PC+{imm,1b'0} | |
| bgeu | SB | Branch ≥ Unsigned | if(R[rs1]>=R[rs2) PC=PC+{imm,1b'0} | 2) |
| blt | SB | Branch Less Than | if(R[rs1]<R[rs2) PC=PC+{imm,1b'0} | |
| bltu | SB | Branch Less Than Unsigned | if(R[rs1]<R[rs2) PC=PC+{imm,1b'0} | 2) |
| bne | SB | Branch Not Equal | if(R[rs1]!=R[rs2) PC=PC+{imm,1b'0} | |
| csrrc | I | Cont./Stat.RegRead&Clear | R[rd] = CSR;CSR = CSR & ~R[rs1] | |
| csrrci | I | Cont./Stat.RegRead&Clear | R[rd] = CSR;CSR = CSR & ~imm | |

## Study the remaining slides yourself

## Pitfalls

- Use large immediates
  - The assembler may report error and abort
- Forget to skip the else block
  - The else block is always executed
- Forget to update the loop control variable
  - You will have an infinite loop!
- Forget to go back to the top of the loop
  - Why is not the loop working?
- Conditions in if or while structure are inverted
  - For example, < becomes >=

https://github.com/zhijieshi/cse3666/blob/master/code-examples/pitfalls.md

## References and Resources

- RISC-V Assembly Programmer's Manual
  - https://github.com/riscv/riscv-asm-manual/blob/master/riscv-asm.md
- RARS: RISC-V Assembler and Runtime Simulator
  - https://github.com/TheThirdOne/rars
- Online simulator from Cornell University (CS3410)
  - https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/
- All you need to know about C code for CSE 3666 in one file
  - https://github.com/zhijieshi/cse3666/blob/master/code-examples/c-example.c

## Question

Calculate the absolute value of s0 and save it in s1.

## Answer

Calculate the absolute value of s0 and save it in s1.

Method 1:

```
        if s0 < 0 goto Else
        s1 = s0
        j EndIf
Else:
        s1 = - s0
EndIf:
```

Method 2:

```
        s1 = s0
        if s0 >= 0 goto EndIf
        s1 = - s0
EndIf:
```

```
#RISC-V code
        add     s1, s0, x0
        bge     s0, x0, EndIf
        sub     s1, x0, s0
EndIf:
```

## Tips for Programming with RISC-V

- Get familiar with RISC-V instructions
  - Know what you can and can NOT do with RISC-V instructions
- Write the pseudocode first
- Break down the pseudocode into small steps
  - Until you know how to do each steps
- Translate each step into RISC-V
  - Keep track use of registers
  - Know where and how the data are stored
    - Endianness, sign, etc.
    - Difference between address and data value

## Common Tasks

- Arithmetic and logic operations
  - Both operands are in register, or one is an immediate
    - Load small/large constants into a register
- Branches and loops
  - If, if-else, if-elseif-else, for, while, do-while, etc.

Coming up soon!

- Load/Store
  - Calculate the address (or use proper displacement)
  - Use proper load and store instructions
- Functions
  - Parameters
  - Return values
  - Save/restore registers and maintain stack