

RSIC-V: Immediate and Control Flow



Z. Jerry Shi

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 add ?X
 sub ?X

ADDI: Addition with an immediate

addi **rd, rs1, imm** # **rd = rs1 + imm**

rd: a register, used as the destination
rs1: a register, used as a source

Inst: [31 ... 21 0] 32-bit

- ADDI:

- Three operands, but the second source operand is an immediate
- The 1st operand is the destination.
- The immediate must be in the range [-2048, 2047]?
 - Examples: 0, 1, 2, -20, 1010.

11 11
12 bits Signed bin

No subtract immediate instruction. **Why?**

Design Principle 3: Make the common case fast

- Small constants are common

Exercises

- How do we do the following operations?

s1, s2, ... are registers

s1 += 4 → *addi s1, s1, 4*

s2 -= 1 → *addi s2, s2, -1*

s3 = 10 → *addi s3, zero, 10*

s4 = t0

s5 = -s5 → *sub s5, zero, s5*

Question

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

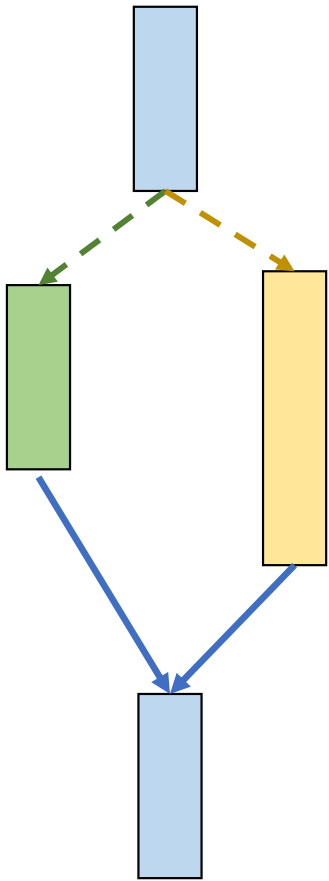
The execution of instructions is normally sequential.

How does a processor support selection and repetition?

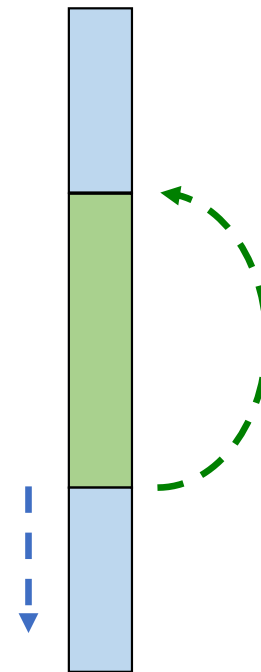
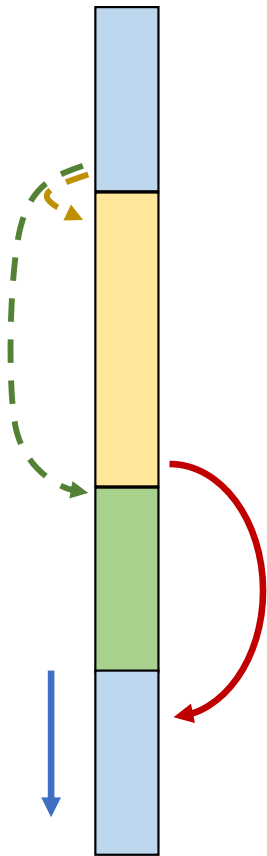


Selection and repetition

- How do we support them?



Selection



Repetition

Branches

$t_0 = 0$
 $t_1 = \neq 0$

- A branch compares **two registers**

$< =$

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

branch equal

beq rs1, rs2, **L1** *offset* # if (rs1 == rs2) goto L1

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

branch not equal

blt rs1, rs2, L3 # if (rs1 < rs2) goto L3

less than
bge rs1, rs2, L4 # if (rs1 >= rs2) goto L4

great equal

example of a label. **Label itself is not an instruction!**

L1: ADD x1, x2, x3

Branches compare **two registers!** Not with an immediate

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
```

Skip: # this is a label

Label is not an instruction

It is the location/address of an instruction


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
```

```
add s0, s1, s2
```

```
# 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

Pseudocode

```
① if (i != j) goto Else
② f = g + h
③ goto EndIf
Else:
④ f = g - h
EndIf:
```

RISC-V code

```
① bne    is3, js4, Else
② add    fs0, s1s1, s2s2
③ beq    x0, x0, EndIf
Else: ④ fs0 = sub    s0, s1, s2
EndIf: ...
```

don't forget

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

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

Pseudocode

i = 0

sum = 0

loop:

if (!(i < 100)) goto exit

sum += i

i += 1

goto loop

exit:

Variable	Register
i	s0
sum	s1
end	s2

Method 1 RISC-V code

① addi s0, x0, 0

② addi s1, x0, 0

③ addi s2, x0, 100

④ bge s0, s2, exit

⑤ add s1, s1, s0

⑥ addi s0, s0, 1

⑦ beq x0, x0, loop

exit:

404

How many instructions are executed?

While – Method 2

Pseudocode

i = 0

sum = 0

goto test

loop: sum += i

i += 1

test: if (i < 100) goto loop

Method 2 RISC-V code

① addi s0, x0, 0

② addi s1, x0, 0

③ addi s2, x0, 100

④ beq x0, x0, test

loop: ⑥ add s1, s1, s0

⑦ addi s0, s0, 1

test: ⑤ blt s0, s2, loop

Variable	Register
i	s0
sum	s1
end	s2

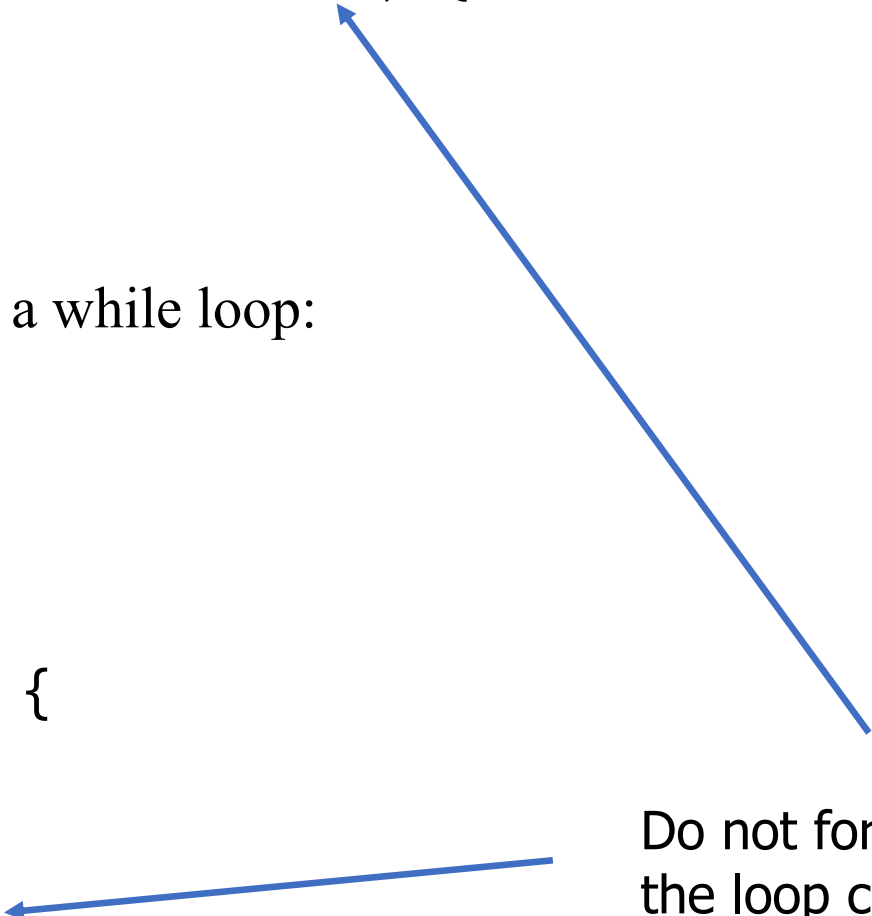
304?
305? ✓
How many instructions are executed?

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 `<=`?

< ↔ ≥
> ↔ ≤
= ≠

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

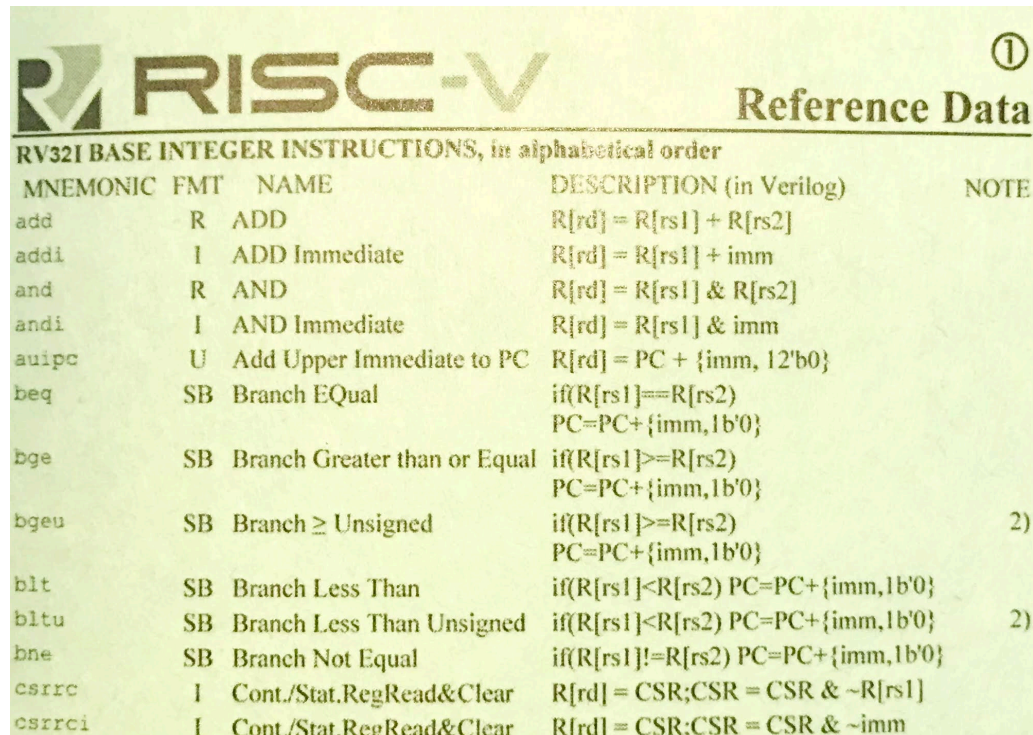
if `s1 <= s2` goto L1

`s1 > s2?`

`beq`
`bne`
`blt`
`bge`

What instructions do I have to learn?

- We won't discuss every instruction in detail in lecture
- Please 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



The image shows a green reference card for RISC-V RV32I base integer instructions. It features the RISC-V logo at the top left and the title 'RV32I BASE INTEGER INSTRUCTIONS, in alphabetical order' in the center. To the right of the title is a circled '1' and the text 'Reference Data'. Below the title is a table with five columns: MNEMONIC, FMT, NAME, DESCRIPTION (in Verilog), and NOTE. The table lists 20 instructions, including add, addi, and, andi, auipc, beq, bge, bgeu, blt, bltu, bne, csrrc, and csrrci. The descriptions are in Verilog syntax, and some instructions have a '2)' in the NOTE column.

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	$if(R[rs1] \geq R[rs2])$ $PC = PC + \{imm, 1b'0\}$	
bgeu	SB	Branch \geq Unsigned	$if(R[rs1] \geq R[rs2])$ $PC = PC + \{imm, 1b'0\}$	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	$if(R[rs1] \neq R[rs2])$ $PC = PC + \{imm, 1b'0\}$	
csrrc	I	Cont./Stat.RegRead&Clear	$R[rd] = CSR; CSR = CSR \& \sim R[rs1]$	
csrrci	I	Cont./Stat.RegRead&Clear	$R[rd] = CSR; CSR = CSR \& \sim imm$	

Pseudoinstructions

- Pseudoinstructions are fake instructions. Their purpose is to make it easier for programmers to write assembly code
 - Assembler converts pseudoinstructions to real instructions
 - Avoid adding new instructions if the operations can be done with existing instructions

Pseudoinstruction		Real Instructions
nop		addi x0, x0, 0
mv	rd, rs	addi rd, rs, 0
neg	rd, rs	sub rd, x0, rs
li	rd, imm	addi rd, x0, imm
		Or more than one instruction *

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

Policy on pseudoinstructions

- **Do not write pseudoinstructions in your code**
 - We do not write a lot of assembly code in this course
- We will see pseudoinstructions in some examples
 - We need to recognize them and be able to **convert them to real instructions**
 - For example, the example in RARS uses LI

Example: display the value stored in \$t0 on the console

```
li    a7, 1           # service 1 is
```

commonly seen pseudoinstructions

nop

li

la **# we will learn la soon**

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

- Are the following instructions set the same value in x1?

`ADDI x1, x0, 65`

`ADDI x1, x0, 0x41`

A. Yes

B. No

Question

- What is the value in s5 after the execution of the following instructions?

```
addi    s1, x0, 1
addi    s2, x0, 1
add     s3, s1, s2
add     s4, s2, s3
add     s5, s3, s4
```

Exercises

Set s1 to 0.

Set s0 to 0x30.

Move the value in s1 to s2.

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:

Write a loop

Write RISC-V instructions to implement the following loop.

s1, s2, and s3 are registers.

```
while (s1 + s2 < s3)
    s3 -= 1;
```

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

Exercises answers

- How do we do the following operations

`s1 += 4`

`s2 -= 1`

`s3 = 10`

`s4 = t0`

`s5 = -s5`

`addi s1, s1, 4`

`addi s2, s2, -1`

`addi s3, x0, 10`

`addi s4, t0, 0`

`sub s5, x0, s5`