CSC 411

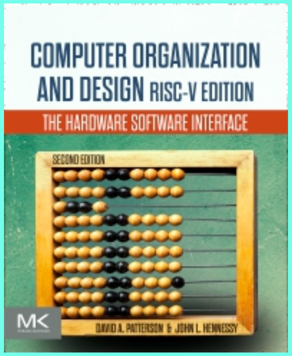
Computer Organization (Spring 2022)
Lecture 8: Conditionals and loops

Prof. Marco Alvarez, University of Rhode Island

Disclaimer

Some of the following slides are adapted from:

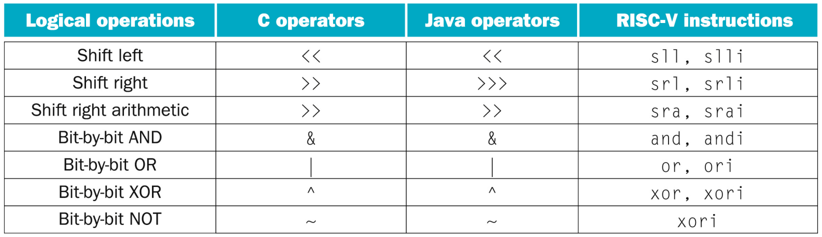
- Computer Organization and Design (Patterson and Hennessy)
  - The Hardware/Software Interface


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

More instructions

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`

Conditional operations

- Jump/branch to a labeled instruction if a condition is `true`
  - otherwise, continue sequentially

```
// if equal, jump to label L1
beq <rs1>, <rs2>, L1

// if not equal, jump to label L1
bne <rs1>, <rs2>, L1
```

Example (if)

```
// 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
```

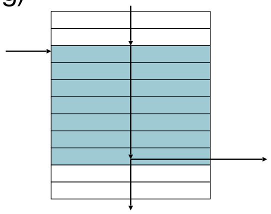
Example (loop)

```
// 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
```

Basic blocks

- A basic block is a sequence of instructions with
  - no embedded branches (except at end)
  - no branch targets (except at beginning)



- Optimization
  - compilers can identify basic blocks and optimize code
  - advanced processors can accelerate execution of basic blocks

More conditional operations

- Branch to instruction if condition is `true`

```
// branch to Label if rs1 less than rs2
blt <rs1>, <rs2>, Label
// branch to Label if rs1 greater or equal than rs2
bge <rs1>, <rs2>, Label


// example
    // ... instructions
    bge  x23, x22, Exit
    addi x22, x22, 1
Exit:
```

Signed vs unsigned

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

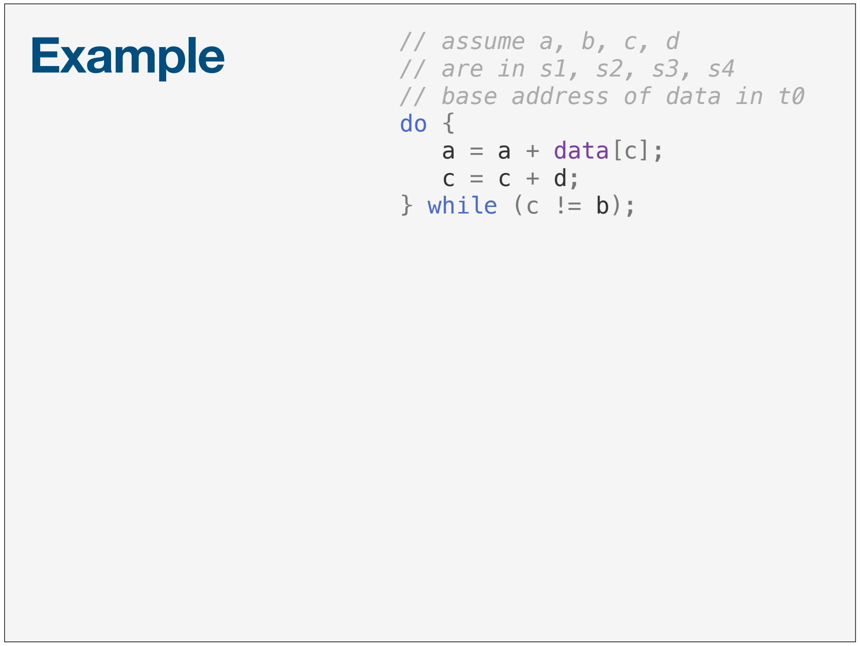
```
// assume x22 stores 0xFFFFFFFF
// assume x23 stores 0x00000001
// which instruction branches?
blt  x22, x23, Label
bltu x22, x23, Label
```

Example

```
// assume t0 holds the value 0x00101000
// what is the value of t2?
//
        addi t2, zero, 10
        blt zero, t0, Else
        beq zero, zero, Done
Else:   addi t2, t2, 2
Done:   addi t2, t2, 1
```

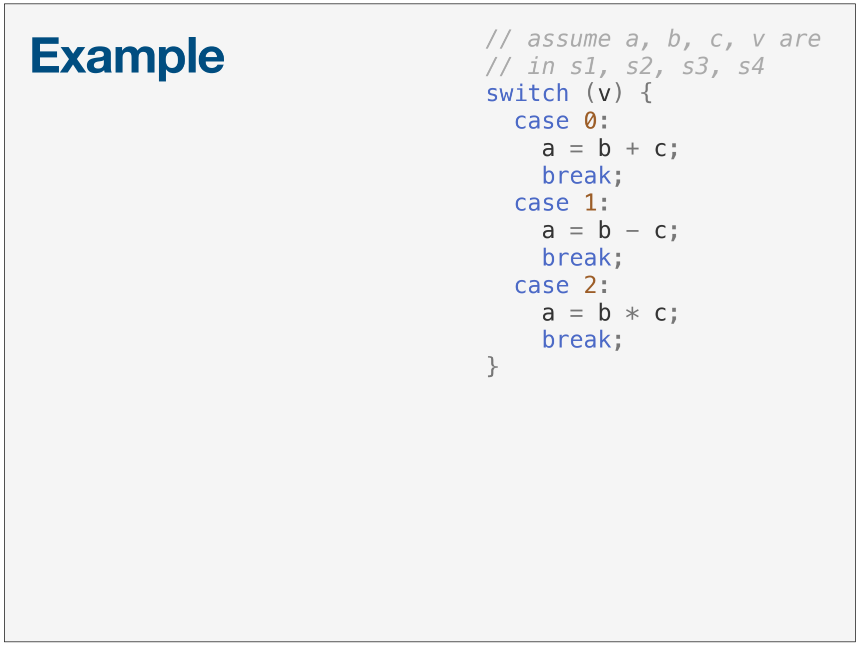
Example

```
// assume t1 holds the value 10 and s2
// is zero, what is the value of s2?
Loop:
        bge zero, t1, Done
        addi t1, t1, -1
        addi s2, s2, 2
        beq zero, zero, Loop
Done:
```



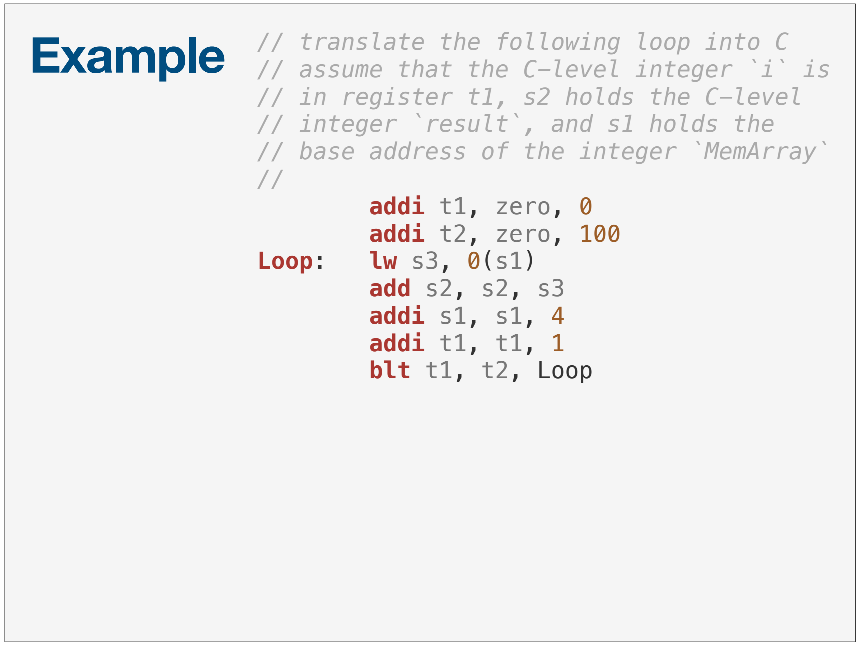
Example

```
// assume a, b, c, d
// are in s1, s2, s3, s4
// base address of data in t0
do {
   a = a + data[c];
   c = c + d;
} while (c != b);
```



Example

```
// assume a, b, c, v are
// in s1, s2, s3, s4
switch (v) {
  case 0:
    a = b + c;
    break;
  case 1:
    a = b − c;
    break;
  case 2:
    a = b ∗ c;
    break;
}
```



Example

```
// translate the following loop into C
// assume that the C−level integer `i` is
// in register t1, s2 holds the C−level
// integer `result`, and s1 holds the
// base address of the integer `MemArray`
//
        addi t1, zero, 0
        addi t2, zero, 100
Loop:   lw s3, 0(s1)
        add s2, s2, s3
        addi s1, s1, 4
        addi t1, t1, 1
        blt t1, t2, Loop
```