RISC-V
Conditional Operations (e.g., Branch)
(Useful for implementing If/Else/Switch)

Sparsh Mittal

Instructions: beq, bne, blt, bge

- Branch to a labeled instruction if a condition is true
  - Otherwise, continue sequentially
- `beq rs1, rs2, L1`
  - if (rs1 == rs2) branch to instruction labeled L1
- `bne rs1, rs2, L1`
  - if (rs1 != rs2) branch to instruction labeled L1
- `blt rs1, rs2, L1`
  - if (rs1 < rs2) branch to instruction labeled L1
- `bge rs1, rs2, L1`
  - if (rs1 >= rs2) branch to instruction labeled L1

beq	bne	blt	bge
==	!=	<	>=

Assume: a and b are mapped to a0 and a1, respectively.

C code	RISC-V code
if(a==b) goto L1;	beq a0, a1, L1
if(a<b) goto L1;	blt a0, a1, L1

Understanding labels

- Text labels are used as branch and unconditional jump targets
- What is the difference between these two codes?

```
li a0, 15
li a1, -16
blt a0, a1, new
beq x0, x0, exit
```

```
END:     li a0, 15
HELLO:   li a1, -16
WHY:     blt a0, a1, new
START:   beq x0, x0, exit
```

- Answer: they are same.
- Labels have no impact on the execution of a program. A label can have any name, including misleading names, grammatically incorrect names, etc. Labels are not functions.

Understanding Labels

- You may observe that “j exit” is not recognized by Ripes.
- Reason: When you write an instruction like "j exit", the stored instruction in the memory will have the address of the instruction following the label 'exit'.
- If there is no instruction immediately after the exit label, the instruction 'j exit' will not work because the jump points to an address that does not exist for the program.

Source code

```
.globl _main
_main:
    addi a1, x0, 0
    j exit
exit:
```

Source code

```
.globl _main
_main:
    addi a1, x0, 0
    j exit
exit:
    addi a1, x0, 0
```

Left: “j exit” is not recognized.

Right: after writing an instruction after the exit label, now the “j exit” is recognized.

Branching Instructions (BLT vs. BLTU)

```
        li a0, 15               # a0 <- 0x0000000F
        li a1, -16              # a1 <- 0xFFFFFFF0
        blt a0, a1, new   # a0 > a1, no branching
        beq x0, x0, exit  # unconditional branch
new:
        add a0,a0, a1     # not executed
exit:   nop
```

```
        li a0, 15         # a0 <- 0x0000000F
        li a1, -16        # a1 <- 0xFFFFFFF0
        bltu a0, a1, new  # a0 < a1, branch taken
        beq x0, x0, exit  # unconditional branch
new:
        add a0,a0, a1     # a0 <- 0xFFFFFFFF
exit:   nop
```

Branching Instructions (BGE vs. BGEU)

```
        li a0, 15              # a0 <- 0x0000000F
        li a1, -16             # a1 <- 0xFFFFFFF0
        bge a0, a1, new        # a0 > a1, branch taken
        beq x0, x0, exit       # unconditional branch
new:
        add a0,a0, a1          # a0 <- 0xFFFFFFFF
exit:
```

```
        li a0, 15              # a0 <- 0x0000000F
        li a1, -16             # a1 <- 0xFFFFFFF0
        bgeu a0, a1, new       # a0 < a1, no branching
        beq x0, x0, exit       # unconditional branch
new:
        add a0,a0, a1          # not executed
exit:
```

Realizing unconditional jump from conditional branches

- Consider C code

```
if(a==a){
b=c+1;
}
```

- Here, if condition is always true.
- Similarly, in RISC-V, we can write following to always jump to label L1
- `beq x0, x0, L1`
- This is equivalent to each of the following instructions
  - jal x0, L1
  - j L1

Solved Examples

"if condition" with equality

- C code:

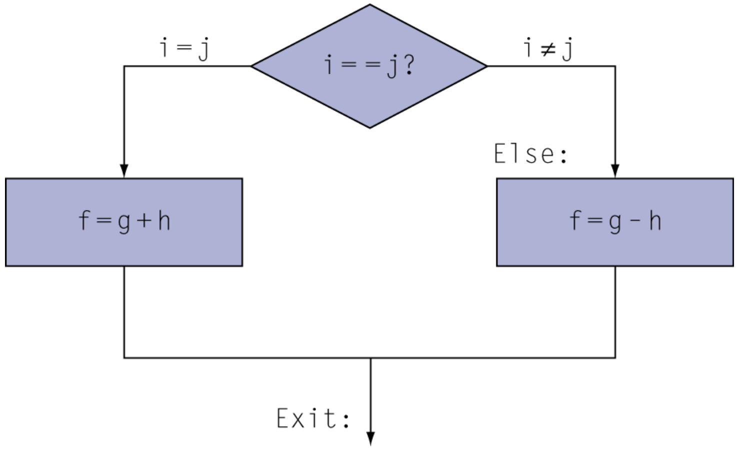
```
if (i==j) f = g+h;
else f = g-h;
```

Write RISC-V code for following mapping

- f → x19   g → x20
- h → x21   i → x22, j → x23

- Compiled RISC-V code:

```
       bne x22, x23, Else
       add x19, x20, x21
       beq x0,x0,Exit // unconditional
Else:  sub x19, x20, x21
Exit:  …
```



“if condition” with greater than

- Let variables f, g, h, i, and j be assigned to registers x5, x6, x7, x28, and x29, respectively. Write RISC-V code for this C code.

```
if(i > j)
   f = g-h
```

Solution: RISC-V has blt, beq, bne and bge instructions. We first express our C-code in a way that can be directly translated into RISC-V codes. This can be done in (at least) three ways:

“if condition” with greater than

```
if(i > j)
  f = g-h
```

Rewritten code 1

C code

```
if(i<j) goto EXIT;
if(i==j) goto EXIT;
f=g-h
EXIT:
```

RISC-V

```
blt x28, x29, EXIT
beq x28, x29, EXIT
sub x5, x6, x7
EXIT: NOP
```

Rewritten code 2

```
if (j>=i) goto EXIT;
f=g-h
EXIT:
```

```
bge x29, x28, EXIT
sub x5, x6, x7
EXIT: NOP
```

Rewritten code 3

```
if (j<i)
f=g-h
```

```
blt x29, x28, L1
j EXIT
L1: sub x5, x6, x7
EXIT: NOP
```

Converting Ternary operator

- Convert h = (i>j)? 3:5;

h → x7 i → x28 j → x29

```
The C code is:
if(j<i) {h=3;}
else  {h=5;}
```

```
Rewritten C code is:

if(j<i) goto TRUEPART;
h=5;
goto EXIT;
TRUEPART: h=3;
EXIT:
```

```
RISC-V code is:

blt x29, x28, TRUEPART
li x7, 5            #False part of code
j EXIT
TRUEPART: li x7, 3   #True part of code
EXIT: NOP
```

What is wrong with this code?

- `blt x29, x28, TRUEPART`
- `TRUEPART: li x7, 3    #True part of code`
- `li x7, 5              #False part of code`

- Answer: The line “`li x7, 3`” will be executed, whether or not x29<x28 condition is true or false.
- This is not what the programmer wanted.

Solved Example

- Convert below code to RISC-V. x/y/z are a0/a1/a2.

if (x==y and x>z)

z=50;

Code 1

```
    bne a0, a1, exit
    ble a0, a2, exit
    li a2, 50
exit:
    nop
```

Code2

```
    beq a0, a1, equal
    j exit
equal:
    blt a2, a0, load
    j exit
load:
    li a2, 50
    j exit
exit:
    nop
```

Solved Example

- Convert below code to RISC-V. x is in a0.

```
switch(x)
        {case 2:
                y= 5;
                break;
        case 3:
                y=7;
                break;
        case 6:
                y=9;
                break;
        default:
                y=100;
                break;
}
```

```
        li a2, 2
        li a3, 3
        li a4, 6
        beq a0, a2, L1  #case 2
        beq a0, a3, L2  #case 3
        beq a0, a4, L3  # case 6
        j default
L1:
        li a1, 5    #y=5
        j exit      #break
L2:
        li a1, 7
        j exit
L3:
        li a1, 9
        j exit
default:
        li a1, 100
exit:
        nop
```

Convert to RISC-V

- Convert k -= i>1? 5: 3;

This is equal to
if(i>1) k= k-5;
else k = k-3;

```
li x10, 1                  # load '1' for comparison
    blt x10, x22, cond1            # check condition i>1 and branch on True
    beq x0, x0, cond2      # unconditional branch taken on previous being False
cond1:
    addi x24, x24, -5      # subtract '5' from K
    beq x0, x0, exit       # exit
cond2:
    addi x24, x24, -3      # subtract '3' from k
    beq x0, x0, exit       # exit
exit:   NOP
```

Example: Checking if number is prime

- A prime is a natural number greater than 1 that has no positive divisors other than 1 and itself.
- 2 is prime, but 6 is not prime.

C++ Pseudo-Code to check if number stored in a1 is prime

```
int t1;
int t0=2;
do
{
   t1 = a1 % t0;   //find the remainder
   if (t1 ==0)
      goto notPrime;
   t0 = t0+1;
}while(t0!= a1);
  return 1; //the number is prime
notPrime:
   return 0; //the number is not prime.
```

Code to check if number stored in a1 is prime.

```
# input in a1 , return value in a0
.main :
        li t0, 2                          # starting divisor
.loop :
        rem t1 , a1 , t0                  # find the remainder (t1)
        beq t1 , zero , .notPrime
        addi t0 , t0 , 1                  # increment the divisor
        bne t0 , a1 , .loop               # loop back
.isPrime:
        li a0, 1                          # number is prime
.notPrime :
        li a0, 0
```

Instruction	Type	Example	Meaning
Branch equal	**SB**	`beq rs1, rs2, imm12`	`if (R[rs1] == R[rs2])` `pc = pc + SignExt(imm12 << 1)`
Branch not equal	**SB**	`bne rs1, rs2, imm12`	`if (R[rs1] != R[rs2])` `pc = pc + SignExt(imm12 << 1)`
Branch greater than or equal	**SB**	`bge rs1, rs2, imm12`	`if (R[rs1] >= R[rs2])` `pc = pc + SignExt(imm12 << 1)`
Branch greater than or equal unsigned	**SB**	`bgeu rs1, rs2, imm12`	`if (R[rs1]` $>=_u$ `R[rs2])` `pc = pc + SignExt(imm12 << 1)`
Branch less than	**SB**	`blt rs1, rs2, imm12`	`if (R[rs1] < R[rs2])` `pc = pc + SignExt(imm12 << 1)`
Branch less than unsigned	**SB**	`bltu rs1, rs2, imm12`	`if (R[rs1]` $<_u$ `R[rs2])` `pc = pc + SignExt(imm12 << 1)`
Jump and link	**UJ**	`jal rd, imm20`	`R[rd] = PC + 4` `PC = PC + SignExt(imm20 << 1)`
Jump and link register	**I**	`jalr rd, imm12(rs1)`	`R[rd] = PC + 4` `PC = (R[rs1] + SignExt(imm12)) & (~1)`

<table>
<tr><th>Pseudo-instruction</th><th>Base instruction(s)</th><th>Meaning</th></tr>
<tr><td>li rd, imm</td><td>addi rd, x0, imm</td><td>Load immediate</td></tr>
<tr><td>la rd, symbol</td><td>auipc rd, D[31:12]+D[11]
addi rd, rd, D[11:0]</td><td>Load absolute address where D = symbol – pc</td></tr>
<tr><td>mv rd, rs</td><td>addi rd, rs, 0</td><td>Copy register</td></tr>
<tr><td>not rd, rs</td><td>xori rd, rs, -1</td><td>One’s complement</td></tr>
<tr><td>neg rd, rs</td><td>sub rd, x0, rs</td><td>Two’s complement</td></tr>
<tr><td>bgt{u} rs, rt, offset</td><td>blt{u} rt, rs, offset</td><td>Branch if > (u: unsigned)</td></tr>
<tr><td>ble{u} rs, rt, offset</td><td>bge{u} rt, rs, offset</td><td>Branch if ≥ (u: unsigned)</td></tr>
<tr><td>b{eq|ne}z rs, offset</td><td>b{eq|ne} rs, x0, offset</td><td>Branch if { = | ≠ }</td></tr>
<tr><td>b{ge|lt}z rs, offset</td><td>b{ge|lt} rs, x0, offset</td><td>Branch if { ≥ | < }</td></tr>
<tr><td>b{le|gt}z rs, offset</td><td>b{ge|lt} x0, rs, offset</td><td>Branch if { ≤ | > }</td></tr>
<tr><td>j offset</td><td>jal x0, offset</td><td>Unconditional jump</td></tr>
<tr><td>call offset</td><td>jal ra, offset</td><td>Call subroutine (near)</td></tr>
<tr><td>ret</td><td>jalr x0, 0(ra)</td><td>Return from subroutine</td></tr>
<tr><td>nop</td><td>addi x0, x0, 0</td><td>No operation</td></tr>
</table>