

Lab Sheet 5 for CS F342 Computer Architecture Semester 1 – 2023-2024

Goals for the Lab: To understand and use (1) the branch instructions and jump instructions, and (2) logical and shift operations.

MIPS does not have if statements, nor does it have for or while statements. To create conditional processing and loops it is necessary to build the control structure with branch statements. Normally, the execution of a statement is followed by the execution of the next statement in the code. A branch instruction causes execution to jump to a different statement. Some branches are conditional -- they only branch if a specified condition is met. Others are unconditional -- they always cause a branch to occur.

I) Branch Instructions: branch if a specified condition is met. All branch instructions use the I-format.

| Name | Format | Meaning | Example |
|---|-------------------|-----------------------------|-----------------------|
| Branch If Equal | beq Rs, Rt, label | if Rs == Rt branch to label | beq \$t1, \$t2, next |
| Branch If Not Equal | bne Rs, Rt, label | if Rs != Rt branch to label | bne \$t1, \$t2, else |
| Branch If Greater Than | bgt Rs, Rt, label | if Rs > Rt branch to label | bgt \$t1, \$t2, top |
| Branch If Less Than | blt Rs, Rt, label | if Rs < Rt branch to label | blt \$t1, \$t2, two |
| Branch If Greater Than or Equal | bge Rs, Rt, label | if Rs >= Rt branch to label | bge \$t1, \$t2, done |
| Branch If Less Than or Equal | ble Rs, Rt, label | if Rs <= Rt branch to label | ble \$t1, \$t2, three |
| Branch If Equal To Zero | beqz Rs, label | if Rs == 0 branch to label | beqz \$t1, after |
| Branch If Not Equal To Zero | bnez Rs, label | if Rs != 0 branch to label | bnez \$t1, second |
| Branch If Greater Than Zero | bgtz Rs, label | if Rs > 0 branch to label | bgtz \$t1, high |
| Branch If Less Than Zero | bltz Rs, label | if Rs < 0 branch to label | bltz \$t1, low |
| Branch If Greater Than or Equal To Zero | bgez Rs, label | if Rs >= 0 branch to label | bgez \$t1, somehi |
| Branch If Less Than or Equal To Zero | blez Rs, label | if Rs <= 0 branch to label | blez \$t1, someho |
| Branch | b label | branch to label | b loop |

The if-else construct :

```

if (var1 == var2) {
    ....          /* block of code #1 */
}
else {
    ....          /* block of code #2 */
}

```

Let's assume that the values of variables *var1* and *var2* are in registers **\$t0** and **\$t1**. Then, this piece of C code would be translated as:

```

        bne $t0, $t1, Else      # go to Else if $t0 != $t1
        ....                   # code for block #1
        beq $0, $0, Exit        # go to Exit (skip code for block #2)
Else:
        ....                   # code for block #2
Exit:
        ....                   # exit the if-else

```

Instructions like, *SLT* -- Set on less than (signed), *SLTI* -- Set on less than immediate (signed), and *SLTU* -- Set on less than (unsigned) uses R-format.

| Instruction | Example | Meaning | Comments |
|----------------------------|--------------------|---------------------------------|---|
| set on less than | slt \$1, \$2, \$3 | if(\$2<\$3)\$1=1; else \$1=0 | Test if less than. If true, set \$1 to 1. Otherwise, set \$1 to 0. |
| set on less than immediate | slti \$1, \$2, 100 | if(\$2<100)\$1=1; else \$1=0 | Test if less than. If true, set \$1 to 1. Otherwise, set \$1 to 0. |

The sltu instruction is used with unsigned integers: sltu d,s,t # if (\$s < \$t) d = 1 else d=0.

Exercise 1: Write MIPS code to find the minimum and maximum of two integer numbers.

II) **Jump Instructions:** an instruction which unconditionally branches to a labeled instruction. We use jump instructions.

| Instruction | Meaning |
|-------------|--|
| j label | Unconditionally jump to the instruction at the label. |
| jal label | Jump and Link; Typically used in function calls. |
| jr Rsrc | Unconditionally jump to the instruction whose address is in register Rsrc. |

The C *while* and MIPS

The C *while* expression closely resembles the *if* expression. It has a predicate and something that happens continuously as long as the expression returns true.

```

predicate:  slt $t0, $s1, $s2      # while ($s1 < $s2)
            beq $t0, $zero, endwhile # {
consequent: addi $s1, $s1, 1      #   $s1++;
            j predicate           #   }
endwhile:    #
```

| | |
|---------------------------------|---|
| predicate: slt \$t0, \$s1, \$s2 | } Predicate (\$t0 = 1 if true) |
| beq \$t0, \$zero, endwhile | } Branch Statement to exit loop |
| consequent: addi \$s1, \$s1, 1 | } Consequent (skipped if not true) |
| j predicate | } Loop Back Statement |
| endwhile: | |

The C *for* and MIPS

The *for* expression is like the *while* expression except it has two additional components to it. Not only does it have the *consequent* body which is evaluated continuously as long as the predicate returns a true value, it has *initialization* and *next* statements built into it. It would look as follows:

```

initialize: add $s1, $zero, $zero      # for ($s1 = 0,
addi $s2, $zero, 10                   #   $s2 = 10;
predicate:  slt $t0, $s1, $s2          #   $s1 < $s2; $s1++)
            beq $t0, $zero, endfor     # {
consequent: addi $s1, $s1, 0           #   $s1 += 0;
            addi $s1, $s1, 1           #   /* $s1 ++; */
            j predicate                #   }
endfor:    #
```

Notice how its form doesn't really change that much from the *while* loop. It's really just a construct in C to make code more compressed and readable. The form in MIPS looks like:

| | |
|--------------------------------------|---|
| initialize: add \$s1, \$zero, \$zero | } Initialization Statements |
| addi \$s2, \$zero, 10 | |
| predicate: slt \$t0, \$s1, \$s2 | } Predicate (\$t0 = 1 if true) |
| beq \$t0, \$zero, endfor | } Branch Statement to exit loop |
| consequent: addi \$s1, \$s1, 0 | } Consequent (skipped if not true) |
| addi \$s1, \$s1, 1 | } Next Statements |
| j predicate | } Loop Back Statement |
| endfor: | |

Exercise 2 : Write MIPS code to find the sum of first n natural numbers.

III) Floating point branch instructions :

Conditional jumps are performed in two stages :

1. Comparison of FP values sets a code in a special register
2. Branch instructions jump depending on the value of the code.

InstructionOperation

c.eq.d \$f0,\$f12 set floating point coprocessor flag true if equal double

c.eq.s \$f0,\$f12 set floating point coprocessor flag true if equal float

c.ge.d \$f0,\$f12 set floating point coprocessor flag true if greater or equal

double c.ge.s \$f0,\$f12 set floating point coprocessor flag true if greater or

equal float c.gt.d \$f0,\$f12 set floating point coprocessor flag true if greater

than double c.gt.s \$f0,\$f12 set floating point coprocessor flag true if greater

than float c.le.d \$f0,\$f12 set floating point coprocessor flag true if less or

equal double c.le.s \$f0,\$f12 set floating point coprocessor flag true if less or

equal float c.lt.d \$f0,\$f12 set floating point coprocessor flag true if less than

double c.lt.s \$f0,\$f12 set floating point coprocessor flag true if less than float

c.ne.d \$f0,\$f12 set floating point coprocessor flag true if not equal double

c.ne.s \$f0,\$f12 set floating point coprocessor flag true if not equal float

The floating point compare instructions set or clear the floating point coprocessor flag. The bc1t (flag true) and bc1f (flag false) instructions can then be used for branching conditioned on the comparison result. [Note : Some of the above instructions are not available in QtSpim]

| Instruction | Example | Meaning |
|-------------|---------------|--|
| bc1t | bc1t label | Branch to label if condition flag 0 is true |
| bc1t | bc1t 1, label | Branch to label if condition flag 1 is true |
| bc1f | bc1f label | Branch to label if condition flag 0 is false |
| bc1f | bc1f 4, label | Branch to label if condition flag 4 is false |

Exercise 3 : Write MIPS code to find the minimum and maximum of two floating point numbers.

```
.data
float1 : .asciiz "Enter first float:"
float2 : .asciiz "Enter second float:"
newline : .asciiz "\n"
equal : .asciiz "both numbers are equal"
minimum : .asciiz "The minimum no. is : "
maximum : .asciiz "\nThe maximum no. is : "

.text
main:

la $a0, float1                #print string
li $v0,4
syscall

li $v0,6                      #read float
syscall

mov.s $f1,$f0                 #move float to f1

la $a0, float2
li $v0,4
syscall
#print string
```

| | |
|------------------|-------------------------|
| li \$v0,6 | #read float |
| syscall | |
| mov.s \$f2,\$f0 | #move float to f2 |
| c.lt.s \$f1,\$f2 | #if f1<f2 set flag to 1 |
| bc1f firstIsMax | |
| secondIsMax: | |
| la \$a0,minimum | #print string |
| li \$v0,4 | |
| syscall | |
| mov.s \$f12,\$f1 | #print number |
| li \$v0,2 | |
| syscall | |
| la \$a0,maximum | #print string |
| li \$v0,4 | |
| syscall | |
| mov.s \$f12,\$f2 | #print number |
| li \$v0,2 | |
| syscall | |
| j exit | |
| firstIsMax: | |
| la \$a0,minimum | |
| li \$v0,4 | |
| syscall | |
| mov.s \$f12,\$f2 | |
| li \$v0,2 | |
| syscall | |
| la \$a0,maximum | |
| li \$v0,4 | |
| syscall | |
| mov.s \$f12,\$f1 | |
| li \$v0,2 | |
| syscall | |
| exit: | |
| li \$v0,10 | |
| syscall | |

IV) Logical and Shift operations : MIPS has two types of bitwise instructions: those that operate on registers, and those that operate on a register and an immediate operand. The register instructions are:

- (1) and Rd, Rs, Rt
- (2) or Rd, Rs, Rt
- (3) nor Rd, Rs, Rt
- (4) xor Rd, Rs, Rt

The immediate instructions are:

- (1) `andi Rt, Rs, immed`
- (2) `ori Rt, Rs, immed`
- (3) `xori Rt, Rs, immed`

Example 1 : By ANDing a value with a deliberately designed constant, called a bit mask we can clear specific bits. Any bits that are 0 in the mask will be cleared in the result, and bits that are 1 in the mask will be preserved in the result.

```
value 10010100
mask 00001111
-----
      00001000
```

Shift instructions :

- Left shift: Moves bits from a lower position to a higher position.
- Right shift: Moves bits from a higher position to a lower position.

`sll $t0,$t1,5` shift left logical
`srl $t0,$t1,5` shift right logical
`sra $t0,$t1,5` shift right arithmetic
`sllv $t0,$t1,$t2` shift left logical value (register)
`srlv $t0,$t1,$t2` shift right logical value (register)
`srav $t0,$t1,$t2` shift right arithmetic value (register)

Example 1: `sll $t0,$t1,5`, suppose \$t1 is 0000 0000 0000 0000 0000 0000 0000 0000
 0000 0000 0000 0000 0000 0000 01010

The bits fill in 0s when value moved to the left by number of positions.

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0100 0000

Example 2: `sra $t0,$t1,5`, suppose \$t1 is 1011 0000 0000 0000 0000 0000 0000 0000
 0000 0000 0000 0000 0000 0000 0000 0000

Then fill in MSB value when value moved to the right by number of positions. (preserving sign).

1111 1101 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Exercise 4 : Give MIPS instruction to perform the following :

(a) Returns in \$v0 , 0 and 1 depending on the most significant bit of input stored in \$a0

[Answer : `srl $v0, $a0, 31`]

(b) Flip (Replace 0's and 1's)

[Answer : `xor $v0, $a0, -1`]

(c) To perform NOT operation, using logical operations.

Home Assignment

Exercise : Write MIPS code to find the input integer is even or odd.