Lecture 3: Instruction Set Architecture

CSE-2204: Computer Architecture and Organization

Logical Operations

1. Shift Left Logical:

Sll $t2, $s0, 4 **# $t2 = $s0 << 4**

Decimal Representation:

op	rs	rt	rd	shamt	funct
0	0	16	10	4	0

Shamt = Shift amount

1. Shift Right Logical:

Srl $t2, $s0, 4 **# $t2 = $s0 >> 4**

Shift Left Logical

Shifting left by i bits gives the same result as multiplying by 2^i .

Example:

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001_2 = 9_{10}$

After executing shift left logical by 4, the new value is

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001\ 0000_2 = 144_{10}$

That is, $9 * 2^4 = 9 * 16 = 144$

Logical Operations

Name	Format	Op	Rs	Rt	Rd	Shamt	Funct	Example
and	R	0	18	19	17	0	36	and $s1, $s2, $s3
Or	R	0	18	19	17	0	37	or $s1, $s2, $s3
NOR	R	0	18	19	17	0	39	nor $s1, $s2, $s3
andi	I	12	18	17	100			andi $s1, $s2, 100
Ori	I	13	18	17	100			ori $s1, $s2, 100

Note:

In keeping with the two operand format, the designer of MIPS decided to include the instruction NOR instead of NOT.

A NOR 0 = NOT (A OR 0) = NOT (A)

Instruction for Making Decisions

Conditional Branch:

- **beq register1, register2, L1 [Branch if equal]**

  Go to the statement labeled L1 if the value of register1 equals the value in register2.

2. **bne register1, register2, L1 [Branch if not equal]**

   Go to the statement labeled L1 if the value of register1 does not equal the value in register2.

Unconditional Branch:

1. Jump *Label* – jump to the target address.

2. Jump register – Jump to the address specified in the register ($ra) to return to the point of calling.

3. Jump and Link– Save the return address in $ra and jump to the starting address of a procedure.

Example

If (i == j) f= g + h; else f= g – h;

f-----→ $s0, g ------→ $s1, h------→ $s2, i-------→ $s3, j-------→ $s4

MIPS Instructions:

1. bne $s3, $s4, ELSE
2. add $s0, $s1, $s2
3. j Exit
4. ELSE: sub $s0, $s1, $s2
5. Exit:

Example

While (save [i] == k) i+=1;

i→ $s3, k→ $s5, base of *save* array→$s6

MIPS Instructions:

```
1.    Loop: sll $t1, $s3, 2        # $t1 = $s3 * 4  [calculating i]
2.    add $t1, $t1, $s6            # $t1 = address of save [i]
3.    lw $t0, 0($t1)               # $t0 = save [i]
4.    bne $t0, $s5, Exit           # go to Exit if save [i] ≠ k
5.    add $s3, $s3, 1              # i = i + 1
6.    j Loop                       # go to Loop
7.    Exit:
```

Conditional Branch Instruction

1. **Set on less than:**

   slt $t0, $s3, $s4

   $t0 is set to 1 if the value in register $s3 is less than the value in register $s4.

2. **Set on less than immediate:**

   slti $t0, $s2, 10

   $t0 is set to 1 if the value in register $s2 is less than 10.

Decision Making Instructions

Name	Format	Op	rs	rt	rd	shamt	funct	Example
beq	I	4	17	18	100			beq $s1, $s2, 100
bne	I	5	17	18	100			bne $s1, $s2, 100
slt	R	0	18	19	17	0	42	slt $s1, $s2, $s3
j	J	2	2500					j 10000
Jr	R	0	31	0	0	0	8	jr $ra
Jal	J	3	2500					jal 10000

Supporting Procedures in Computer Hardware

In executing a procedure, the program must follow these six steps:

1. Place parameters in a place where the procedure can access them.
2. Transfer control to the procedure
3. Acquire the storage procedure needed by the procedure.
4. Perform the desired task.
5. Place the result value in a place where the calling program can access them.
6. Return control to the point of origin.

Registers used in procedure calling:

1. $a0--$a3: Four argument registers in which to pass parameters.
2. $v0--$v1: Two value registers in which to return values.
3. $ra (31): One return address register to return to the point of origin.

Supporting Procedures in Computer Hardware

Execution Sequence Using MIPS instruction:

1. The calling program puts the parameter values in $a0--$a3.
2. Use **jal X** to jump to the procedure. [X is the name of the called procedure]
3. Procedure X performs the calculations.
4. Place the result in $v0--$v1.
5. Return control to the calling program using **jr $ra**.

Need for a Stack in Procedure Calling

- ✓ Any registers needed by the caller must be restored to the values that they contained before the procedure was invoked.
- ✓ The convention is to store the registers used by the caller into a stack and restores them from the stack when the caller need them.
- ✓ The stack pointer ($sp) is used to store the address of the most recently allocated address in the stack.
- ✓ The stack grows from higher addresses to the lower addresses.

Example

Procedure:

```
int calculation ( int g, int h, int i, int j )
{
        int f;
        f= (g+h)-(i+j);
        return f;
}
```

g, h, i, j → $a0---$a3, f→ $s0, g+h → $t0, i+ j → $t1

MIPS Instructions:

1. addi $sp, $sp, -12
2. sw $t1, 8($sp)
3. sw $t0, 4($sp)
4. sw $s0, 0($sp)

Example

MIPS Instructions:

```
5. add $t0, $a0, $a1          #$t0 = g + h
6. add $t1, $a2, $a3           #$t1 = i + j
7.sub $s0, $t0, $t1           #$s0 = (g+h) – (i+j)
8.add $v0, $s0, $zero         #return f
9.lw $s0, 0($sp)              #restore the register
10. lw $t0, 4($sp)
11. lw $t1, 8($sp)
12. addi $sp, $sp, 12         #adjust stack pointer
13. jr $ra                    #jump back to the calling program.
```

Saving Registers across the Procedure Call

- ✓ $t0--$t9: 10 temporary registers are not preserved by the callee.
- ✓ $s0--$s7: 8 registers are preserved on the procedure call.

NOTE:

This convention will eliminate two store and two load instruction in the previous example.

Chapter 2 of Text Book.