

EN164: Design of Computing Systems

Lecture 10: Processor / ISA 3

Professor Sherief Reda

<http://scale.engin.brown.edu>

Electrical Sciences and Computer Engineering
School of Engineering
Brown University
Spring 2011



[material from Patterson & Hennessy, 4th ed and Harris 1st ed]

Summary of last lecture

```
addi $s0, $0, 4
```

```
addi $s1, $0, 12
```

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

```
lw $t1, 4($s0)
```

```
lw $t2, -4($s1)
```

```
sub $t3, $t1, $t2
```

```
sw $t3, 4($0)
```

Logical operations

- Instructions for bitwise manipulation

Operation	C	Java	MIPS
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bitwise AND	&	&	and, andi
Bitwise OR			or, ori
Bitwise NOT	~	~	nor

- Useful for extracting and inserting groups of bits in a word

Shift operations

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- shamt: how many positions to shift
- Shift left logical
 - Shift left and fill with 0 bits
 - sll by i bits multiplies by 2^i
- Shift right logical
 - Shift right and fill with 0 bits
 - srl by i bits divides by 2^i (unsigned only)

AND operations

- Useful to mask bits in a word
 - Select some bits, clear others to 0
- and \$t0, \$t1, \$t2

\$t2	0000 0000 0000 0000 0000 1101 1100 0000
\$t1	0000 0000 0000 0000 0011 1100 0000 0000
\$t0	0000 0000 0000 0000 0000 1100 0000 0000

OR operations

- Useful to include bits in a word
 - Set some bits to 1, leave others unchanged
- or \$t0, \$t1, \$t2

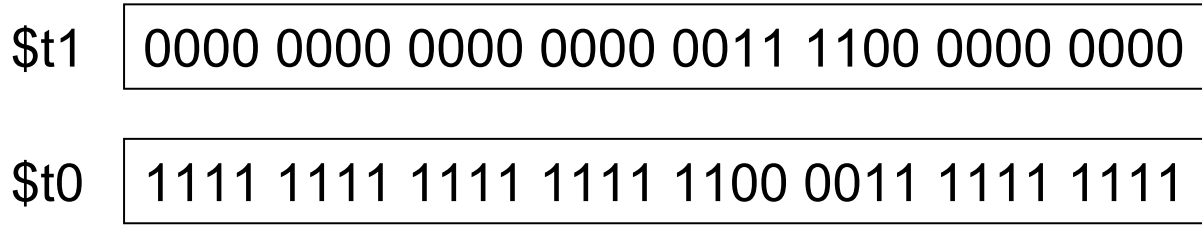
\$t2	0000 0000 0000 0000 0000 1101 1100 0000
\$t1	0000 0000 0000 0000 0011 1100 0000 0000
\$t0	0000 0000 0000 0000 0011 1101 1100 0000

NOT operations

- Useful to invert bits in a word
 - Change 0 to 1, and 1 to 0
- MIPS has NOR 3-operand instruction
 - $a \text{ NOR } b == \text{NOT} (a \text{ OR } b)$

```
nor $t0, $t1, $zero
```

Register 0: always read as zero



Conditional operators

- Branch to a labeled instruction if a condition is true
 - Otherwise, continue sequentially
- `beq rs, rt, L1`
 - if (`rs == rt`) branch to instruction labeled L1;
- `bne rs, rt, L1`
 - if (`rs != rt`) branch to instruction labeled L1;
- `j L1`
 - unconditional jump to instruction labeled L1

Compiling if statement

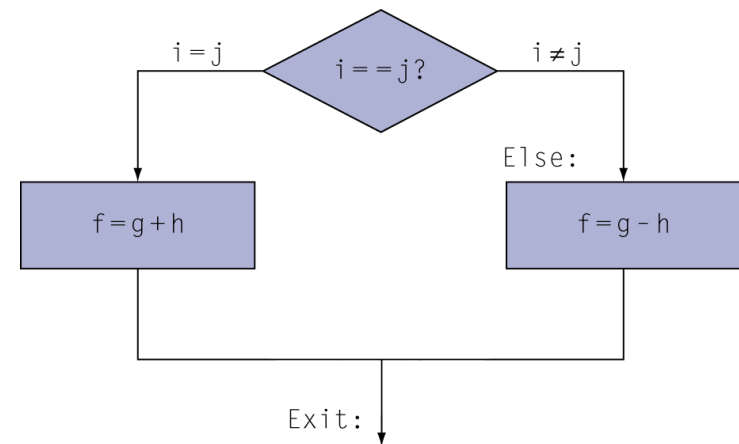
- C code:

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

- f, g, ... in \$s0, \$s1, ...

- Compiled MIPS code:

```
        bne $s3, $s4, Else  
        add $s0, $s1, $s2  
        j   Exit  
Else:   sub $s0, $s1, $s2  
Exit:   ...
```



Compiling loop statements

- C code:

```
while (save[i] == k) i += 1;
```

- i in \$s3, k in \$s5, address of save in \$s6

- Compiled MIPS code:

```
Loop:  sll    $t1, $s3, 2
       add   $t1, $t1, $s6
       lw    $t0, 0($t1)
       bne  $t0, $s5, Exit
       addi $s3, $s3, 1
       j    Loop
Exit:  ...
```

More conditional operations

- Set result to 1 if a condition is true
 - Otherwise, set to 0
- `slt rd, rs, rt`
 - if ($rs < rt$) $rd = 1$; else $rd = 0$;
- `slti rt, rs, constant`
 - if ($rs < \text{constant}$) $rt = 1$; else $rt = 0$;
- Use in combination with `beq`, `bne`

```
    slt $t0, $s1, $s2    # if ($s1 < $s2)
    bne $t0, $zero, L    # branch to L
```

Branch instruction design

- Why not `blt`, `bge`, etc?
- Hardware for `<`, `≥`, ... slower than `=`, `≠`
 - Combining with branch involves more work per instruction, requiring a slower clock
 - All instructions penalized!
- `beq` and `bne` are the common case
- This is a good design compromise

Summary of instruction formats

R-Type



I-Type



J-Type

