MIPS Assembly Language I

Tom Kelliher, CS 220

Dec. 1, 2003

1 Administrivia

Announcements

Assignment

From Last Time

Outline

1. Arithmetic and logical instructions.
2. Instruction operands.
3. Instruction formats.

Coming Up

Conditional execution.

2 Arithmetic and Logical Instructions

- add, sub, addi, addu, subu.

- and, or, andi, ori, sll, srl.

2.1 Instruction Semantics

```
add a, b, c        # This, BTW, is a comment.
sub a, a, b
addi a, a, 100
and a, b, c
andi a, a, 32000
```

De-compile each of the following:

```
add a, b, c
add a, a, d
add a, a, e
```

De-compile further into a single HLL statement.

Compile each of the following:

```
a = b + c;
d = a - e;
f = (g + h) - (i + j);
```

Operands are **registers**.

3 Instruction Operands

Properties of registers:

1. Number of registers. 32 for MIPS, including the hardwired register. Two ways of naming: numbers, convention "nicknames".

2. Number of bits/register. 32. Word size.

   Implications: size of address space, datapath width.

3.1 Using MIPS Registers

Recall:

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

Assume `f` through `j` are in `$1` through `$5`, respectively. Compile the statement.

3.2 Memory Addressing

1. HLL have complex data structures such as arrays and structs. How are they handled?
2. Data transfer instructions: load, store. operands: memory address, register.
3. Actual MIPS instructions: `lw`, `sw`.

   Base and offset addressing: `lw $s0, 8($s1)`
4. MIPS memory is byte addressable, so word addresses differ by 4:

Word Address	Byte Address			
0	0	1	2	3
4	4	5	6	7
8	8	9	10	11
12	12	13	14	15
	msB			lsB

Compile the following:

```
g = h + A[8];
```

where `g` is in `$s1`, `h` is in `$s2`, and the base address of `A`, an array of 100 words, is in `$s3`.

Base, offset addressing.

Compile each of the following:

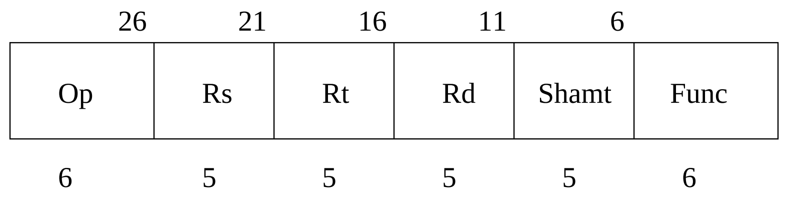
```
A[12] = h + A[8];

A[j] = h + A[i];
```

Base, offset addressing, using constant offsets, is similarly useful for accessing members of structures.

4 Instruction Formats

4.1 MIPS R-Format

Example instruction: `add $s2, $s0, $s1`



Fields:

1. Op: Opcode.
2. Rs: First *source* operand.
3. Rt: Second source operand.
4. Rd: *Destination* operand.
5. Shamt: Shift amount — ignore for now.
6. Func: Function. Further specification of the opcode.

In assembly: `Op/Func Rd, Rs, Rt`

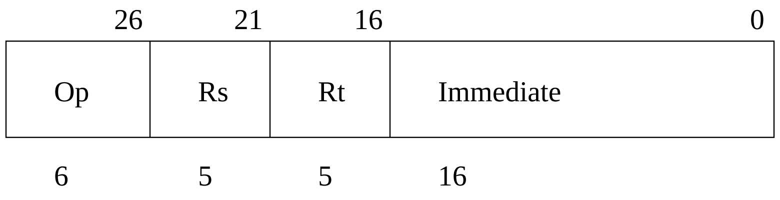
Notes:

1. Example encodings:

Assembly	**Op**	**Rs**	**Rt**	**Rd**	**Shamt**	**Func**
`add $1, $2, $3`	0	2	3	1	0	32
`sub $4, $5, $6`	0	5	6	4	0	34

4.2 MIPS I-Format

Example instruction: `lw $s0 8($s1)`



Fields:

1. Op: Opcode.
2. Rs: Source register.
3. Rt: *Destination* register.
4. Address: 16-bit signed immediate value.

   Offset range?

In assembly: `Op/Func Rt, address(Rs)`

Notes:

1. This format also used for immediate operands: `addi $1, $2, 123`.
2. Example encodings:

Assembly	**Op**	**Rs**	**Rt**	**Address**
`lw $1, 1000($2)`	35	2	1	1000
`sw $3, -12($4)`	43	4	3	-12