Lecture 12

<2016-05-23 Mon>

Contents

1 MIPS

- ISA: instruction set architecture

  - REG

  - MEM

  - Instructions

- CISC: complex instruction set computer

  - variable length instructions

- RISC: reduced instruction set computer

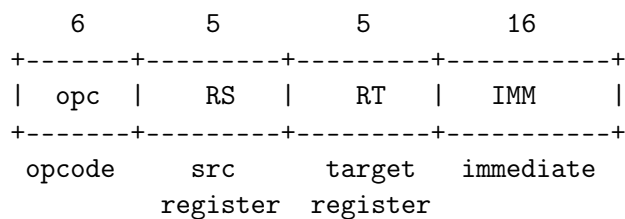|	RISC	CISC
registers	32	6, 8, 16
register class	1	some
arithmetic operands	registers	memory+registers
instructions	3-addr	2-addr
addressing modes	r; M[r+c]	several
instruction length	32 bits	variable
side effect	none	some
instruction cost	uniform	varied

1.1 MIPS: example of RISC

- all instructions are 32-bit

- following an opcode

1.2 Main Types of Instructions

- arithmetic

  - integer
  - floating point

- memory access instructions

  - load & store

- control flow

  - jump
  - conditional jump
  - call & return

```
I-Type:


     6          5          5           16
+-------+---------+---------+-----------+
|  opc  |   RS    |   RT    |  IMM      |
+-------+---------+---------+-----------+
 opcode     src      target   immediate
          register  register
```
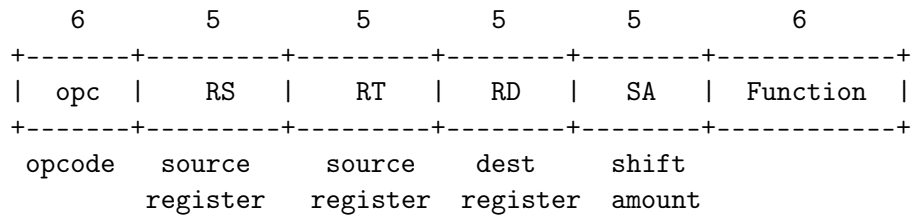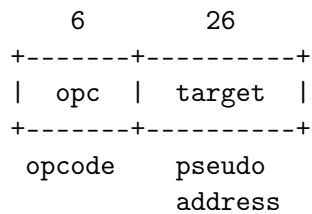
```
example:
add     $rt, $rs, immed

R-Type

     6          5          5         5         5          6
+-------+---------+---------+--------+--------+------------+
|  opc  |   RS    |   RT    |   RD   |   SA   |  Function  |
+-------+---------+---------+--------+--------+------------+
 opcode   source     source    dest     shift
          register   register  register amount

example:
add     $rd, $rs, $rt

J-Type

     6          26
+-------+----------+
|  opc  |  target  |
+-------+----------+
 opcode    pseudo
           address

example:
j       Label
```

- I-Type instructions

  - load word, store word
  - arithmetic with immediate

- R-Type instructions

  - arithmetic: add, and, or, etc.

- control type instructions

  - I-Type control: op(6) + RS(5) + RT(5) + IMM(5)
  - J-Type control: op(6) + immediate(26)

1.2.1 Arithmetic

- most instructions have 3 operand

- arithmetic operands are registers, not memory

- operand order is fixed (destination first)

- e.g. `add $s0,$s1,$s2`

```
A = B + C + D;
E = F - A;

add     $t0, $s1, $s2
add     $s0, $t0, $s3
sub     $s4, $s5, $s0
```

1.2.2 Load and Store (Data Transfer): I-Type Instructions

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

lw      $t0, 32($s3)
add     $t0, $s2, $t0
sw      $t0, 32($s3)
```

- store word operation has no destination (register) operand

instructions	explanation
`li $v0, 4`	$v0 <- 4
`la $a0, msg`	$a0 <- address of msg
`lw $t0, x`	$t0 <- x
`sw $t0, y`	y <- $t0

- `la`, `li`

    - since a label represents a fixed memory address after assembly, `la` is actually a special case of `li` (load immediate)

- `lw`, `la`

    - e.g. x at address 10, contains 2
        * `la $a0, x` : $a <- 10
        * `lw $a0, x` : $a0 <- 2

- `lw $t0 8($sp)`

1.2.3 Control

- decision making instructions

    - alter control flow
    - change the "next" instruction to be executed

```
if (i == j)
    h = i + j;

        bne     $t0, $t1, Label
        add     $s3, $s0, $s1
Label:
```

- conditional branch: I-Type Instructions

- unconditional branch: J-Type Instructions

instructions	explanation
jal proc	jump and link, start procedure proc, $ra holds address of instruction following jal
jr $ra	jump register, return from procedure call puts $ra value back into PC

- address in branches

    - I-Type:

        * specify a register and add it to address
            · use instruction address register
            · most branches are local

1.3 MIPS Miscellaneous & Examples

1.3.1 MIPS compiler conventions

1.3.2 System Calls

```
;; R-Type
add     $s1, $s2, $s3
sub     $s1, $s2, $s3
;; I-Type
lw      $s1, 100($s2)
sw      $s1, 100($s2)
bne     $s4, $s5, Label
```

Table 1: MIPS compiler conventions

name	register number	usage
$zero	0	the constant value 0
$v0-$v1	2-3	values for results and expression evaluation
$a0-$a3	4-7	arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved (by callee)
$t8-$t9	24-25	more temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Table 2: System Calls

service	code	arguments	result
print integer	1	$a0 = integer	console print
print string	4	$a0 = string addr	console print
read integer	5		$a0 = result
read string	8	$a0 = string addr, $a1 = length limit	console read
exit	10		end of program

```
beq     $s4, $s5, Label
;; J-Type
j       Label
```

1.3.3   Example

```
void swap(int v[], int k) {
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}

swap:
        multi   $2, $5, 4
        add     $2, $4, $2
        lw      $15, 0($2)
        lw      $16, 4($2)
```

```
sw      $16, 0($2)
sw      $16, 4($2)
jr      $31
```

variable	register
k	$5
v	$4
&v[k]	$2

1.3.4 Summary

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1, $s2, $s3`	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	`sub $s1, $s2, $s3`	$s1 = $s2 - $s3	Three operands; data in registers
	add immediate	`addi $s1, $s2, 100`	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	`lw  $s1, 100($s2)`	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	`sw  $s1, 100($s2)`	Memory[$s2 + 100] = $s1	Word from register to memory
	load byte	`lb  $s1, 100($s2)`	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	`sb  $s1, 100($s2)`	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	`lui $s1, 100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	`beq  $s1, $s2, 25`	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne  $s1, $s2, 25`	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt  $s1, $s2, $s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set less than immediate	`slti  $s1, $s2, 100`	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than constant
Unconditional jump	jump	`j    2500`	go to 10000	Jump to target address
	jump register	`jr   $ra`	go to $ra	For switch, procedure return
	jump and link	`jal  2500`	$ra = PC + 4; go to 10000	For procedure call