Computer Organization

605.204

Module Three

Part Four

Language for the People

Module Three

- Part Four
- In this presentation, we are going to talk about :
-
- A second set of MIPS assembly language instructions

Previously

- Previously we talked about:
- A Language for the People - Part one
- Now: A Language for the People - Part two

At this point:

- MIPS
  - Loading words but addressing bytes
  - Arithmetic operands in registers only

Instruction	Meaning
`add $s1, $s2, $s3`	$s1 = $s2 + $s3
`sub $s1, $s2, $s3`	$s1 = $s2 – $s3
`lw $s1, 100($s2)`	$s1 = Memory[$s2+100]
`sw $s1, 100($s2)`	Memory[$s2+100] = $s1

Assembly Control Instructions

- Decision making instructions
  - alter the control flow,
  - select the "next" instruction to be executed
- **Branch**
- MIPS conditional branch instructions:

```
bne $t0, $t1, Label
beq $t0, $t1, Label
```

- Example: if (i==j) h = i + j;

```
        bne $s0, $s1, Label
        add $s3, $s0, $s1
Label:      ....
```

Assembly Control Instruction

- **Jump**

- MIPS unconditional branch instructions:

```
j  label
```

- Example:

```
if (i!=j)                    beq $s4, $s5, Lab1
   h=i+j;                    add $s3, $s4, $s5
else                         j Lab2
   h=i-j;          Lab1:     sub $s3, $s4, $s5
                   Lab2:     ...
```

A New Assembly Instruction

- We have: beq, bne, what about Branch-if-less-than?
- New instruction:

```
slt $at, $s1, $s2
```

if $s1 < $s2 then
$at = 1
else
$at = 0

- Can use this instruction to build: **"`blt $s1, $s2, Label`"**

  Can now build general control structures

- The assembler needs a register to do this, **$at**

  There is a policy of use convention for registers

Assembly Instructions for Constants

- Small constants are used quite frequently (50% of operands)

  for example: AB = A + 5;
  CQ = C - 18;

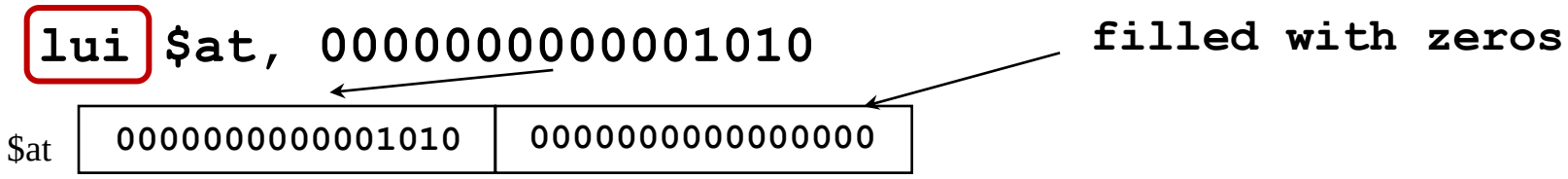
- Solutions ? ?
  - put 'typical constants' in memory and load them, or …
  - create hard-wired registers (like $zero) for constants, or ...

- MIPS Instructions:

```
addi $29, $29, 4
slti $8,  $18, 10
andi $29, $29, 6
ori  $29, $29, 4
```

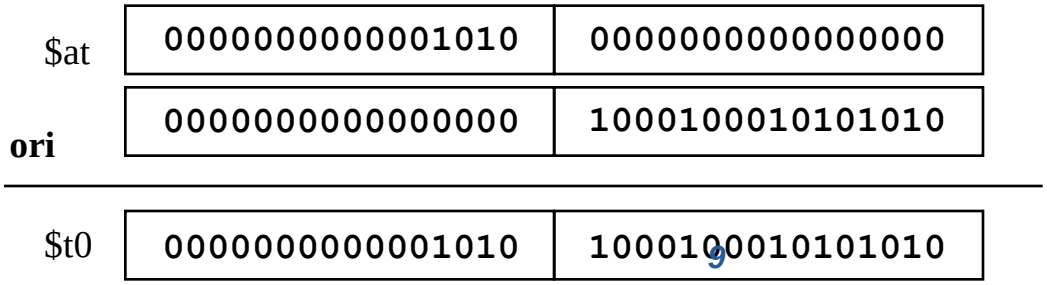
Large Assembly Constants

- We'd like to be able to load a big value constant into a register
- Must use two instructions, new "load upper immediate" instruction



- Then must get the lower order bits correct,

```
ori $t0, $at, 1000100010101010
```

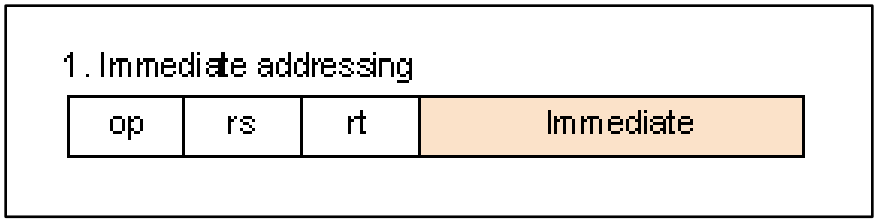


Assembly Addressing modes

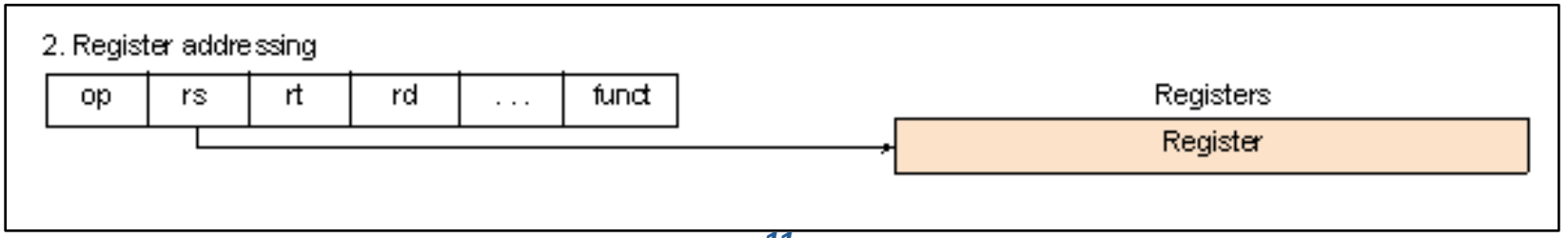
- Immediate
- Register
- Base Relative
- PC Relative
- Pseudo Direct

Addressing Modes

- Immediate
  - ADDI

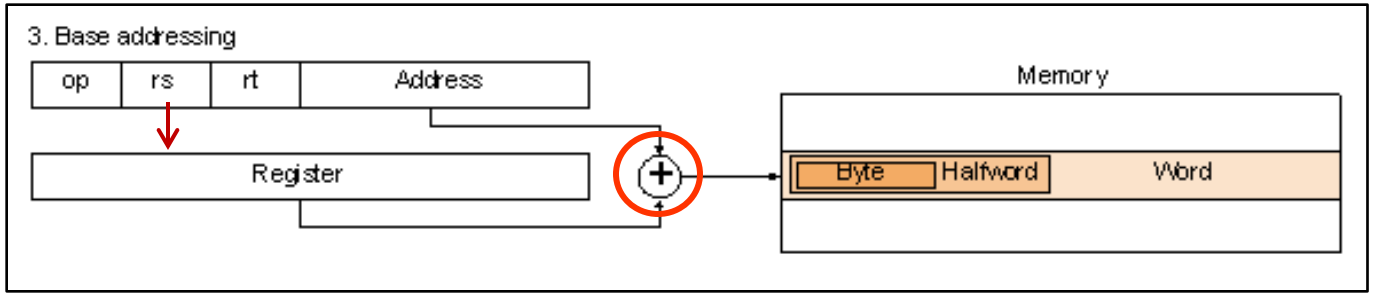


- Register
  - MULT

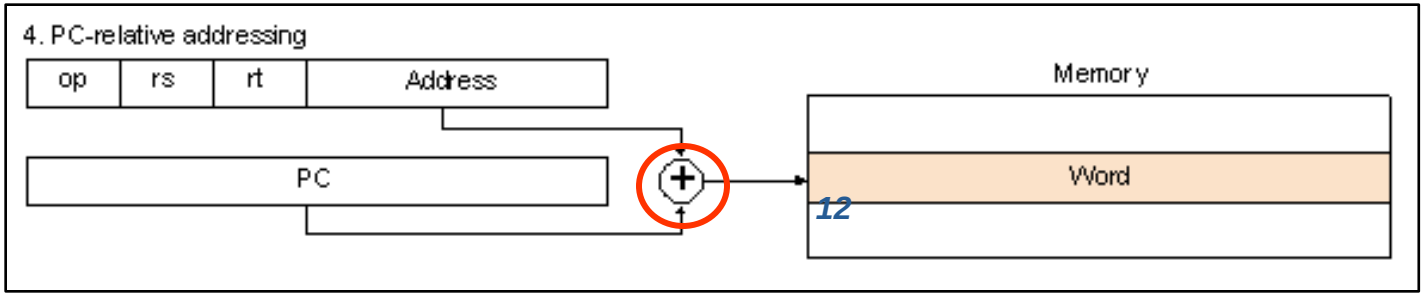


Addressing Modes

- Base Relative
  - Memory access

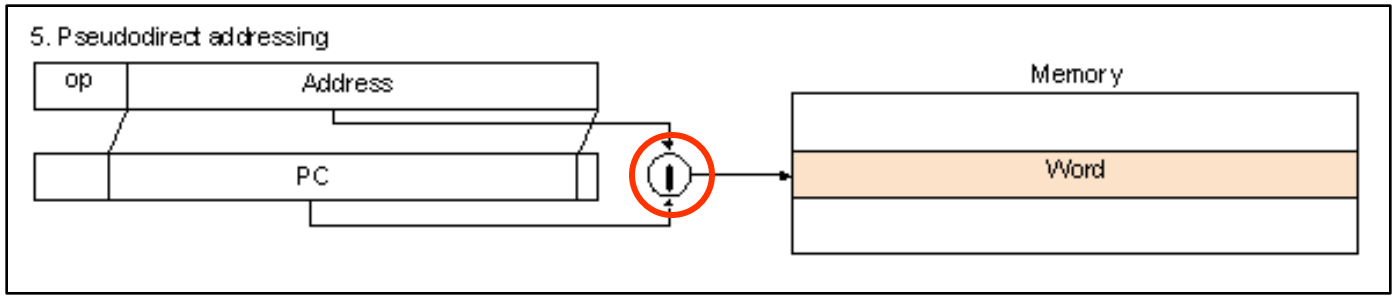


- PC Relative
  - Branch



Addressing Modes

- Pseudo Direct
  - Jump



concatenate

MIPS Assembly 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
Uncondi-tional 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

Review

- Design Principles:
  - **Simplicity favors regularity**
  - **Smaller is faster**
  - **Good design demands compromise**
  - **Make the common case fast**

Summary

- More of the MIPS Assembly Language

Next: The MIPS Machine Language