Outline of Lecture

- MIPS addressing modes.

MIPS Addressing Modes

- The MIPS architecture provides two more ways of accessing operands:

Constant / Immediate Operands

- ***Constants*** are used frequently in all kinds of programs (50% - 70% of arithmetic operands are constants). For example to add 4 to register $29:

```
lw $t0, AddrConstant4($0) # $t0=constant 4

add $sp, $sp, $t0         # $sp = $sp + $t0
```

- A better way is to avoid memory access, and offer new versions in which the operand can be a constant - this is the **I-type** (**immediate**) instruction format.

Example

The add instruction that has one constant operand is called `add immediate` or `addi`. To add 4 to register $29 we just write

```
addi $sp, $sp, 4 # $sp = $sp + 4
```

What is the corresponding MIPS machine code?

Answer

The MIPS machine code is as follows:

op	rs	rt	immediate
8	29	29	4

op	rs	rt	immediate
001000	11101	11101	0000 0000 0000 0100

- We can also compare the values to a constant using the instruction **set less than immediate** (**slti**):

```
slti $t0, $s2, 10  # $t0=1 if $s2 < 10
```

Principle #4: ***Always make the common case faster*****.**

Since constants are frequently used, then it is very beneficial to have an immediate addressing mode.

- The size of the constant in a MIPS immediate addressing mode is 16 bits - which covers most cases.

- In case we want to load the upper 16 bits of a register with a constant, we use the instruction `load upper immediate` (`lui`) for this purpose.

Example

What is the MIPS code to load this 32-bit constant into register $s0?

$s0 = 0000 0000 0011 1101 0000 1001 0000 0000

```
lui $s0, 61

# 61 = 0000 0000 0011 1101
```

Now the value of $s0 is:

$s0 = 0000 0000 0011 1101 0000 0000 0000 0000

The next step is to add lower 16 bits

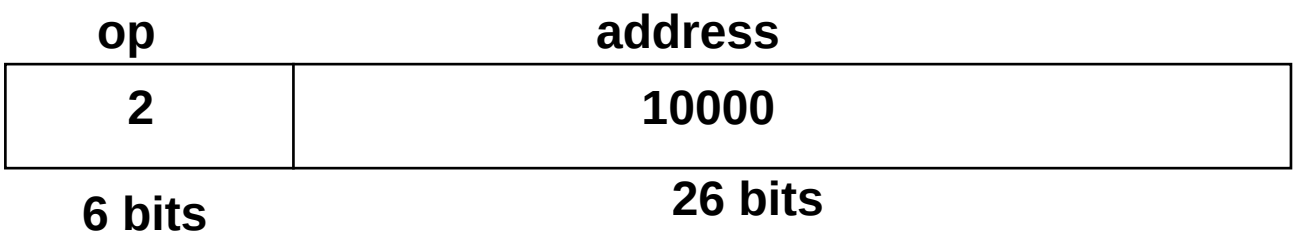
```
addi $s0, $s0, 2304

# 2304 = 0000 1001 0000 0000
```

$s0 = 0000 0000 0011 1101 0000 1001 0000 0000

Addressing in Branches and Jumps

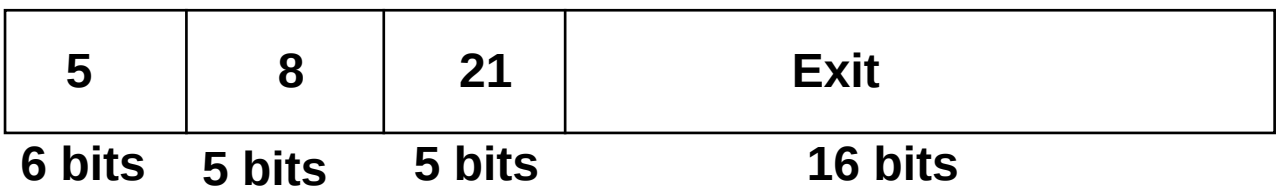
- The simplest addressing mode for MIPS is for the jump instructions. The instruction format for this addressing is called the **J-type**.

e.g., `j 10000 # goto memory location 10000`



- On the other hand, the branch instruction must specify 2 operands in addition to the branch address:

`bne $s0, $s1, Exit # goto Exit if $s0 ≠ $s1`



- If the address of the program had to fit in 16-bits, it means that the program cannot be bigger than 2^{16} memory locations (which is restrictive).

- To solve this problem, we always specify a register to be added to the branch address:

PC = register + branch address

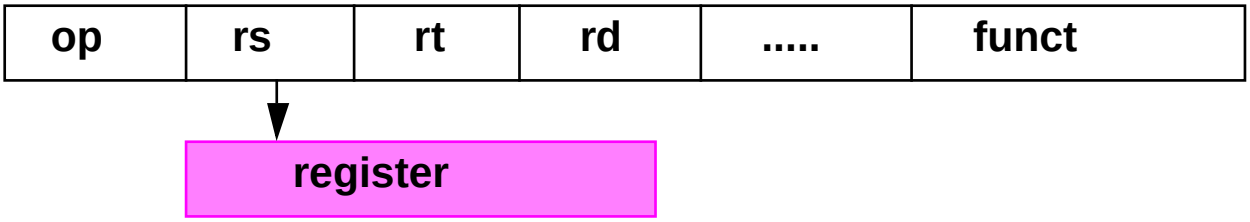
Then, the program would be allowed to be as large as 2^{32} memory locations. The register to be added is the ***program counter*** (PC).

- This type of addressing is called ***PC-relative addressing***. MIPS uses PC-relative for all conditional branches (like most processors).

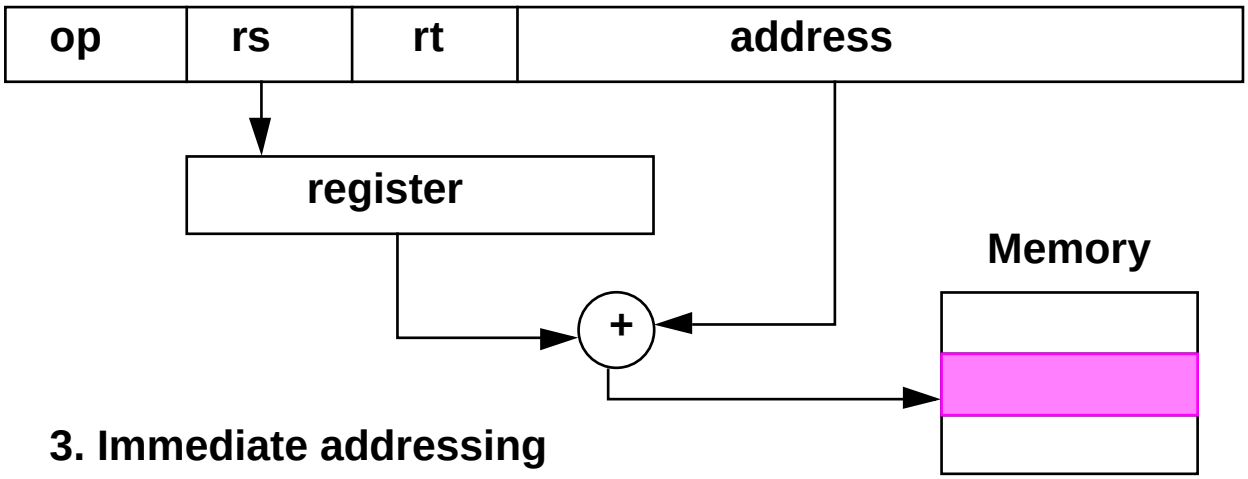
Summary

- The MIPS supports the following addressing modes:
  - ➜ `Register addressing:` The operand is a register (e.g., `add $s1, $s2, $s3`).
  - ➜ `Base or displacement addressing:` The operand is at the memory location whose address is the sum of a register and an address in the instruction (e.g., `lw $s1, 100($s2)`).
  - ➜ `Immediate addressing:` The operand is a constant within the instruction itself (e.g., `addi $s1, $s2, 100`).
  - ➜ `PC-relative addressing:` The address is the sum of the PC and a constant in the instruction (e.g., `bne $s1, $s2, 100`).

1. Register addressing

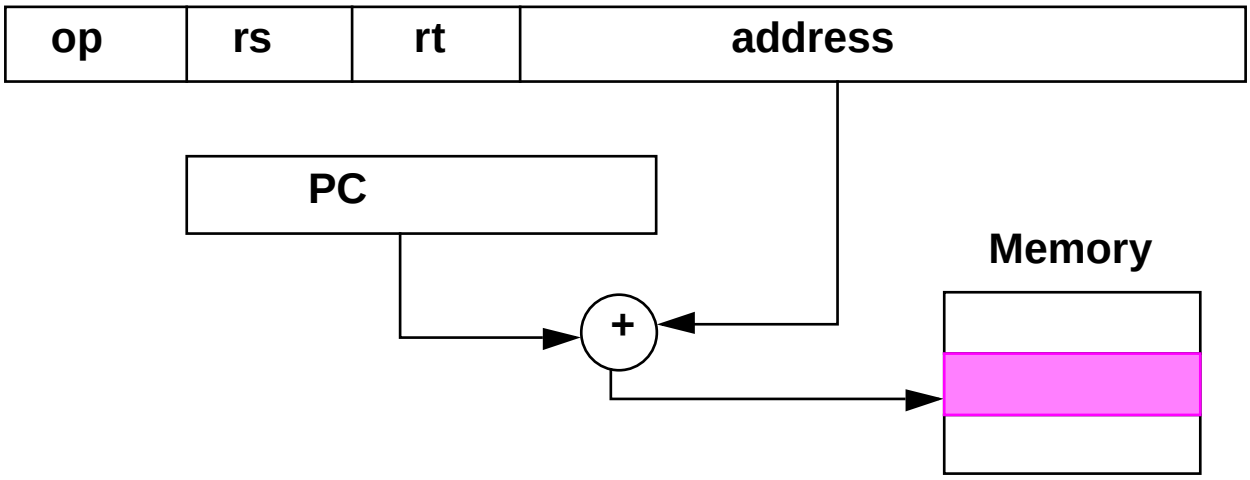


2. Base addressing



3. Immediate addressing

4. PC-relative addressing



Further Reading

Chapter 3 and Appendix. **David A. Patterson and John L. Hennessy. *Computer Organization & Design: The Hardware / Software Interface*. Morgan Kaufman Publishers, 1998.**