MIPS Instruction Set

Dr. Prasenjit Chanak

Department of Computer Science and Engineering
Indian Institute of Technology (BHU), Varanasi
UP, 221005

MIPS ISA features - addressing

Purpose

Operand sources

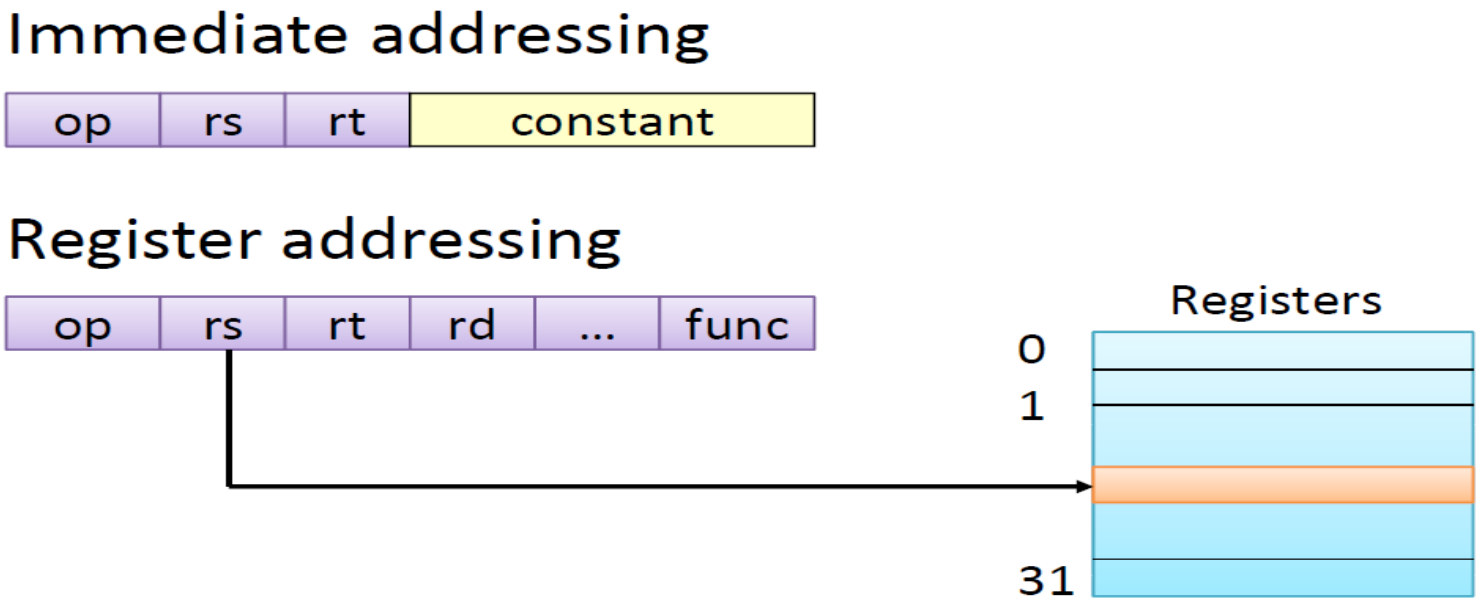
Result Destinations

Jump targets

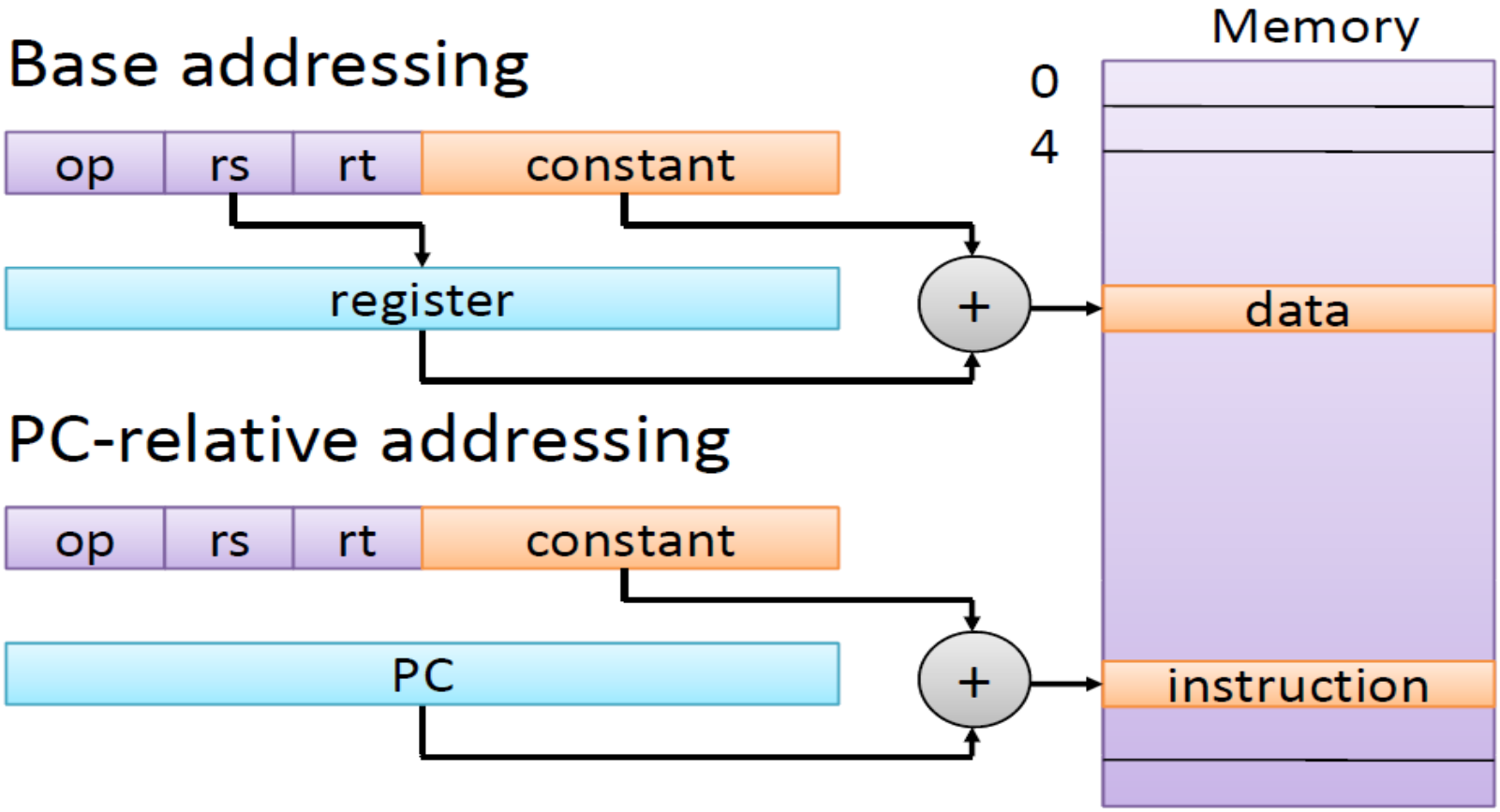
Addressing modes

- Immediate
- Register
- Base/index
- PC relative
- (pseudo) Direct
- Register indirect

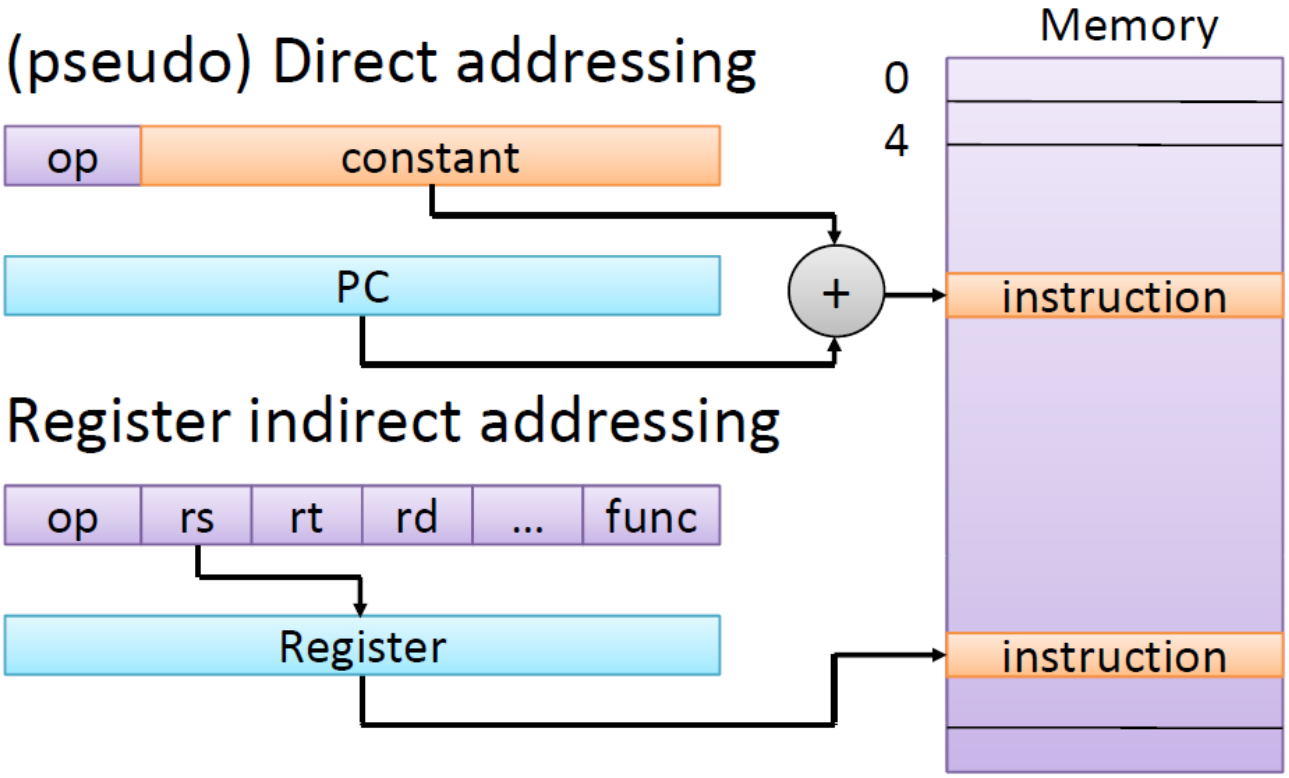
MIPS addressing modes - 1



MIPS addressing modes - 2



MIPS addressing modes - 3



RISC vs. CISC

Reduced (vs. Complex) Instruction Set Computer

- Uniformity of instructions
- Simple set of operations and addressing modes
- Register based architecture with 3 address instructions

RISC Philosophy

- 1970s John Cocke at IBM
- Majority of combinations of orthogonal addressing modes and instructions were not used
  - By most programs generated by compilers
- Difficult in many cases to write a compiler
  - To take advantage of the features provided by conventional CPUs.

RISC examples

- Virtually all new instruction sets since 1982 have been RISC
  - SUN's SPARC (S*calable* P*rocessor* ARC*hitecture*)
  - HP's PA-RISC
  - ARM (Advance RISC Machine)
  - Motorola's PowerPC (P*erformance* O*ptimization* W*ith* E*nhanced* R*ISC Performance Computing,*)
  - DEC's Alpha
  - MIPS
  - CDC 6600 (1960's)

MIPS (RISC) Design Principle

MIPS (RISC) Design Principle

- Simplicity favors regularity
  - Fixed size instructions
  - Small number of instruction formats
  - Opcode always the first 6 bits
- Smaller is faster
  - Limited instruction set
  - Limited number of registers in register file
  - Limited number of addressing modes
- Make the common case fast
  - Arithmetic operands from the register file (load-store machine)
  - Allow instructions to contain immediate operands
- Good design demands good compromises
  - Three instruction formats

Arithmetic Operations

- Add and subtract, three operands
  - Two sources and one destination

```
add a, b, c    # The sum of b and c is placed in a
```

- All arithmetic operations have this form
  - Design principle 1: Simplicity favors regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at low cost

Arithmetic Example

- C Code:

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

- Compiled “MIPS Code”:

```
add t0,g,h # temporary variable t0 contains g + h
add t1,i,j # temporary variable t1 contains i + j
sub f,t0,t1 # f gets t0 - t1, which is (g + h) - (i + j)
```

MIPS Arithmetic Instructions

- MIPS assembly language arithmetic statement

```
add $t0,$s1,$s2
sub $s0,$t0,$t1
```

- Each arithmetic instruction performs one operation
- Each specifies exactly three operands that are all contained in the data path's register file

Register Operands

- Arithmetic instructions use register operands
- MIPS has 32×32-bit register file
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32 bit data called “word”
- Design Principle 2: Smaller is faster
  - Main memory: millions of location

Register vs. Memory

- Registers are faster to access than memory
- Operating on memory data requires loads and stores
  - More instructions to be executed
- Compiler must use registers for variables as much as possible
  - Only spill to memory for less frequently used variables
  - Register optimization is important

Immediate Operands

- Constant data specified in instruction
- No subtract immediate instruction
  - Just use negative constant
- Design principle 3: Make the common case fast
  - Small constants are common
  - Immediate operand avoids a load instruction