ICS 233
Computer Architecture & Assembly Language

MIPS PROCESSOR INSTRUCTION SET

ICS 233
Computer Architecture & Assembly Language

Lecture 4

Lecture Outline

- ❑ **MIPS Processor Overview**
- ❑ **MIPS Instruction Formats**
- ❑ **MIPS Addressing Modes**

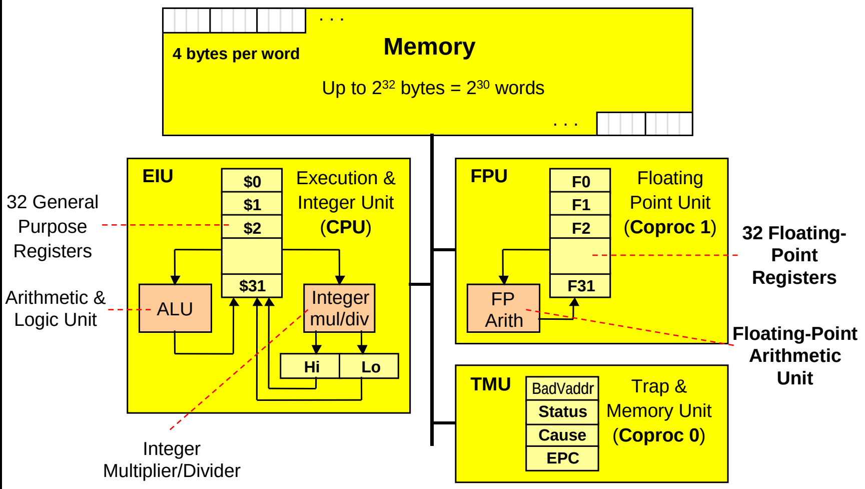
The MIPS CPU

- ➢ MIPS CPU originated from **research project at Stanford,** most successful and flexible CPU design of the 1990s
- ➢ MIPS CPUs were found in SGI graphics workstations, Windows CE handhelds, CISCO routers, and Nintendo 64 video game consoles
- ➢ MIPS CPUs follow the RISC (Reduced Instruction Set Computer) design principle:
  - – limited repertoire of machine instructions
  - – limited arithmetical complexity supported
  - – extensive supply of CPU registers (reduce memory accesses)

MIPS Processor Architecture

- MIPS processor consists of an **integer processing unit (CPU) and a collection of coprocessors** that perform ancillary tasks or operate on other types of data such as floating-point numbers.
- **Coprocessor 0** handles exceptions, interrupts, and the virtual memory system
- **Coprocessor 1** is the floating-point unit.
- **MIPS is a load-store architecture**, which means that only load and store instructions access memory.
- Computation instructions (like arithmetic & logical) operate only on values in registers.

Overview of the MIPS Processor



. . .
4 bytes per word
Memory
Up to 2^{32} bytes = 2^{30} words
. . .
EIU
$0
$1
$2
$31
Execution & Integer Unit (CPU)
32 General Purpose Registers
Arithmetic & Logic Unit
ALU
Integer mul/div
Hi
Lo
Integer Multiplier/Divider
FPU
F0
F1
F2
F31
Floating Point Unit (Coproc 1)
FP Arith
32 Floating-Point Registers
Floating-Point Arithmetic Unit
TMU
BadVaddr
Status
Cause
EPC
Trap & Memory Unit (Coproc 0)

MIPS CPU Registers

- MIPS CPU contains **32 general-purpose registers that are numbered $0. . . $31**
- **Register $0** always contains the hardwired value 0.
- **Registers also have symbolic names reflecting their conventional use (Most of these conventions concern Procedure call and return)**
- Register **$at ($1)** reserved for the **assembler**, Registers **$k0 ($26) and $k1($27)** are reserved for the **operating system**. These registers should not be used by user programs or compilers
- Registers **$a0 to $a3 ($4 to $7)** are used to pass the first four arguments to functions (remaining arguments are passed on the stack).
- Registers **$v0 ($2) and $v1($3)** are used to return values from functions

MIPS CPU Registers

- Registers **$t0 to $t9 ($8 to $15, $24, $25)** are **caller-saved registers** that are used to hold temporay quantities that need not be preserved across calls
- Register **$s0 to $s7 ($16 to $23)** are **callee-saved registers** that hold long-lived values that should be preserved across calls.
- Register **$gp ($28)** is a **global pointer** that points to the middle of a 64K block of memory in the static data segment.
- Register **$sp ($29)** is the **stack pointer**, which points to the last location (stack top) on the stack.
- Register **$fp ($30)** is the **frame pointer**
- Registers **$ra ($31)** is the **return address register** used to hold the return address from procedure call.

MIPS Registers

Register Name	Register Number	Usage
$zero	$0	Constant 0
$at	$1	Reserved for assembler
$v0	$2	Expression evaluation & result of a function
$v1	$3	Expression evaluation & result of a function
$a0	$4	Argument 1
$a1	$5	Argument 2
$a2	$6	Argument 3
$a3	$7	Argument 4
$t0	$8	Temporary (not preserved across call)
$t1	$9	Temporary (not preserved across call)
$t2	$10	Temporary (not preserved across call)
$t3	$11	Temporary (not preserved across call)
$t4	$12	Temporary (not preserved across call)
$t5	$13	Temporary (not preserved across call)
$t6	$14	Temporary (not preserved across call)
$t7	$15	Temporary (not preserved across call)

MIPS Registers

Register Name	Register Number	Usage
$s0	$16	Saved temporary (preserved across call)
$s1	$17	Saved temporary (preserved across call)
$s2	$18	Saved temporary (preserved across call)
$s3	$19	Saved temporary (preserved across call)
$s4	$20	Saved temporary (preserved across call)
$s5	$21	Saved temporary (preserved across call)
$s6	$22	Saved temporary (preserved across call)
$s7	$23	Saved temporary (preserved across call)
$t8	$24	Temporary (not preserved across call)
$t9	$25	Temporary (not preserved across call)
$k0	$26	Reserved for OS Kernel
$k1	$27	Reserved for OS Kernel
$gp	$28	Pointer to global area
$sp	$29	Stack pointer
$fp	$30	Frame pointer
$ra	$31	Return address (used by function call)

MIPS Register Conventions

- Assembler can refer to registers by name or by number
  - It is easier for you to remember registers by name
  - Assembler converts register name to its corresponding number

Name	Register	Usage
`$zero`	`$0`	Always 0 (forced by hardware)
`$at`	`$1`	Reserved for assembler use
`$v0 – $v1`	`$2 – $3`	Result values of a function
`$a0 – $a3`	`$4 – $7`	Arguments of a function
`$t0 – $t7`	`$8 – $15`	Temporary Values
`$s0 – $s7`	`$16 – $23`	Saved registers (preserved across call)
`$t8 – $t9`	`$24 – $25`	More temporaries
`$k0 – $k1`	`$26 – $27`	Reserved for OS kernel
`$gp`	`$28`	Global pointer (points to global data)
`$sp`	`$29`	Stack pointer (points to top of stack)
`$fp`	`$30`	Frame pointer (points to stack frame)
`$ra`	`$31`	Return address (used by jal for function call)

MIPS Instruction Formats

- **All instructions are of fixed length, 32 bits long**
- **Three types of instruction formats :**

➢ **R-type or R-format**

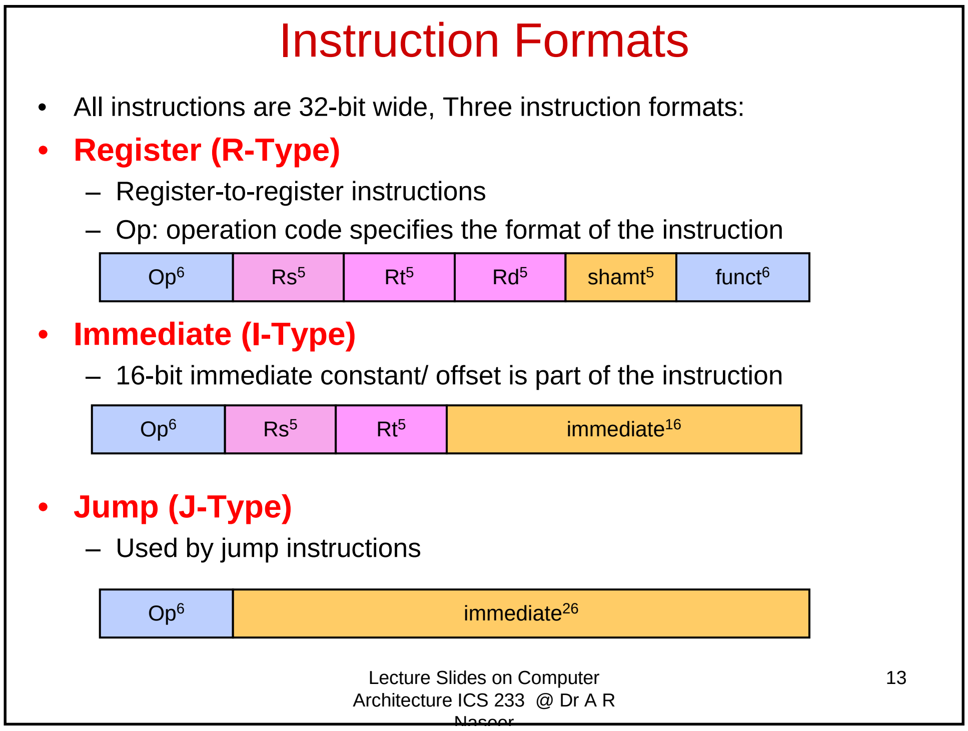
(for Registers)

➢ **I-type or I-format**

(for data transfer)

➢ **J-type or J-format**

(for jump instructions)



Instruction Formats

- All instructions are 32-bit wide, Three instruction formats:
- **Register (R-Type)**
  - Register-to-register instructions
  - Op: operation code specifies the format of the instruction

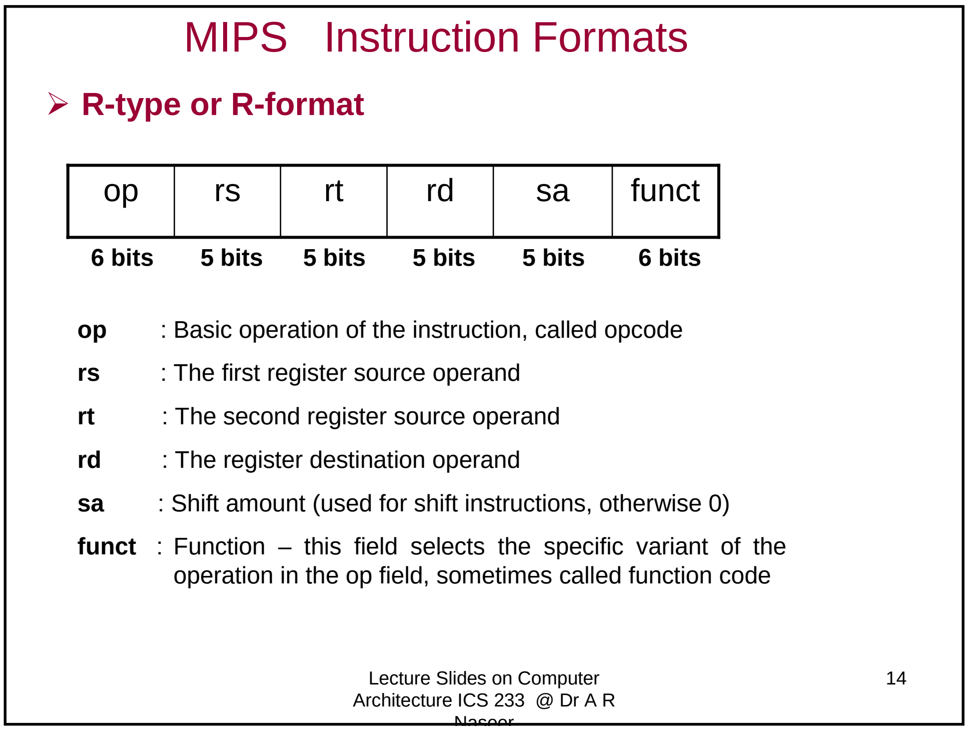
Op^6	Rs^5	Rt^5	Rd^5	$shamt^5$	$funct^6$

- **Immediate (I-Type)**
  - 16-bit immediate constant/ offset is part of the instruction

Op^6	Rs^5	Rt^5	$immediate^{16}$

- **Jump (J-Type)**
  - Used by jump instructions

Op^6	$immediate^{26}$



MIPS Instruction Formats

➢ **R-type or R-format**

op	rs	rt	rd	sa	funct
6 bits	**5 bits**	**5 bits**	**5 bits**	**5 bits**	**6 bits**

op	: Basic operation of the instruction, called opcode
rs	: The first register source operand
rt	: The second register source operand
rd	: The register destination operand
sa	: Shift amount (used for shift instructions, otherwise 0)
funct	: Function – this field selects the specific variant of the operation in the op field, sometimes called function code

MIPS Instruction Formats

➢ **I-type or I-format**

op	rs	rt	address
6 bits	**5 bits**	**5 bits**	**16 bits**

op : Basic operation of the instruction, called opcode

rs : source register

rt : destination register which receives the result

Address : 16-bit address or immediate constant

MIPS Instruction Formats

➢ **J-type or J-format**

op	address
6 bits	**26 bits**

op : Basic operation of the instruction, called opcode

Address : 26-bit address

MIPS Addressing Modes

❑ **There are five addressing modes :**

- **Register addressing**

  where the operand is a register

- **Base or displacement addressing**

  where the operand is at the memory location whose address is the sum of a register and a constant in the instruction

- **Immediate Addressing**

  where the operand is a constant within the instruction itself

- **PC-relative addressing**

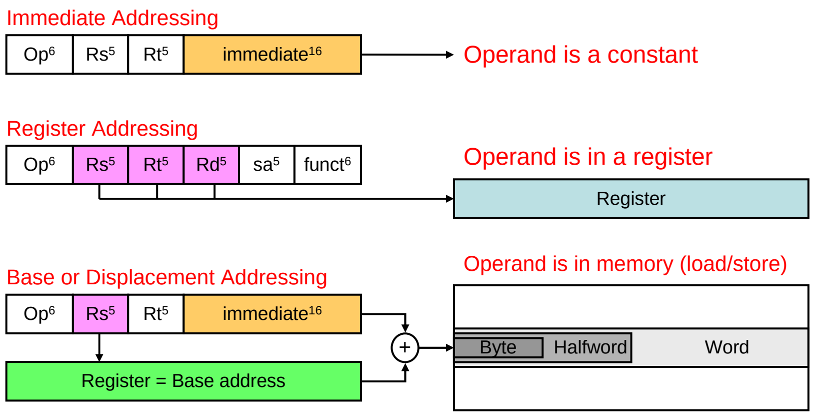
  where the address is the sum of the PC and a constant in the instruction

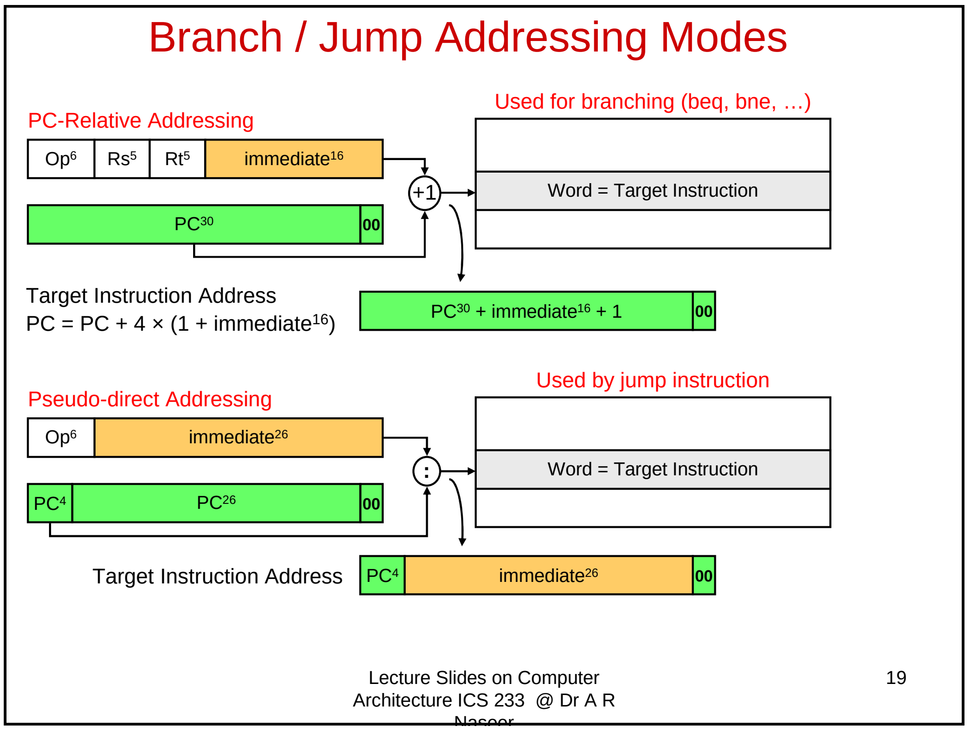
- **Pseudodirect addressing**

  where the jump address is the 26 bits of the instruction concatenated with the upper bits of the PC.

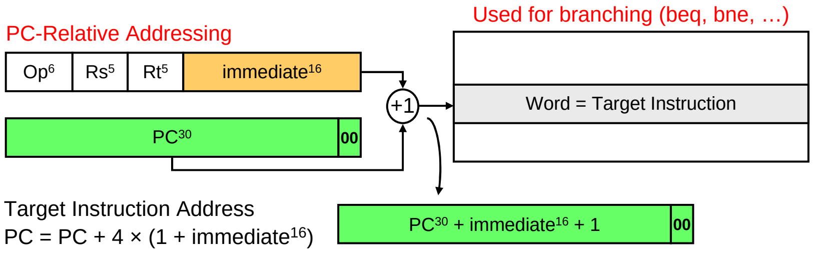
Addressing Modes

- Where are the operands?
- How memory addresses are computed?

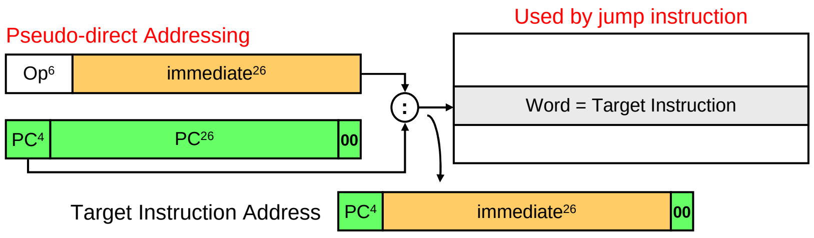




Branch / Jump Addressing Modes



Target Instruction Address
PC = PC + 4 × (1 + immediate^{16})



Target Instruction Address