Instruction formats

3 instruction formats: all 32 bits

- **R-type: register**
  - **arithmetic and logical**
- **I-type: immediate**
  - **use constant in instruction**
  - **arithmetic, logical, conditional branch**
- **J-type: jump**
  - **unconditional branch**

Design principle #3: "Good design demands good compromises."

- **Size of instruction vs. number of formats**

Register conventions

register conventions and mnemonics

Number	Name	Use
0	`$zero`	hardwired 0 value
1	`$at`	used by assembler (pseudo-instructions)
2-3	`$v0-1`	subroutine return value
4-7	`$a0-3`	arguments: subroutine parameter value
8-15	`$t0-7`	temp: can be used by subroutine without saving
16-23	`$s0-7`	saved: must be saved and restored by subroutine
24-25	`$t8-9`	temp
26-27	`$k0-1`	kernel: interrupt/trap handler
28	`$gp`	global pointer (static or extern variables)
29	`$sp`	stack pointer
30	`$fp`	frame pointer
31	`$ra`	return address for subroutine
	`Hi, Lo`	used in multiplication (provide 64 bits for result)

hidden registers

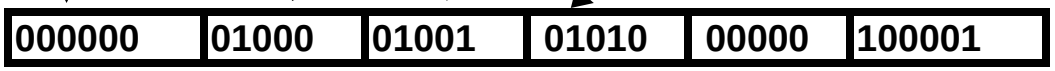
PC, the program counter, which stores the current address of the instruction being executed

IR, which stores the instruction being executed

Instruction formats: R-type, I-type

R-type: register

```
addu     $r10,$r8,$r9  # add 2 numbers
```



b_{31-26}	b_{25-21}	b_{20-16}	b_{15-11}	b_{10-6}	b_{5-0}
opcode	$rs	$rt	$rd	shamt	function

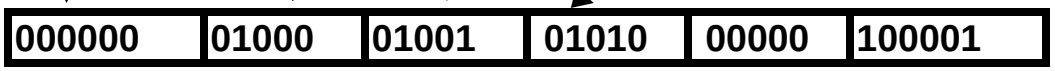
3 registers: 2 source, 1 destination

operation: opcode and function

Instruction formats: R-type, I-type

R-type: register

```
addu     $r10,$r8,$r9  # add 2 numbers
```



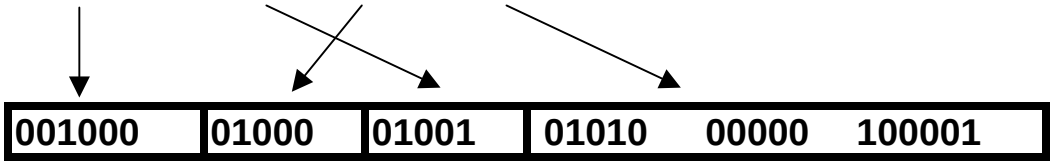
b_{31-26}	b_{25-21}	b_{20-16}	b_{15-11}	b_{10-6}	b_{5-0}
opcode	$rs	$rt	$rd	shamt	function

3 registers: 2 source, 1 destination

operation: opcode and function

I-type: immediate

```
addi     $rt, $rs, immed  # add a constant to a register
```



b_{31-26}	b_{25-21}	b_{20-16}	b_{15-0}
opcode	$rs	$rt	immediate

semantics: $R[t] = R[s] + (IR_{15})^{16} :: IR_{15-0}$

- sign-extend the immediate value to 32 bits
- add it (using signed addition) to register R[s]
- store the result in register R[t]

Instruction formats: J-type

J-type: jump

```
j       target  # jump to target address
```

b_{31-26} b_{25-0}

opcode target

semantics:

PC <- PC_{31-28} :: IR_{25-0} :: 00

update the PC by using:

- upper 4 bits of the program counter
- 26 bits of the target (lower 26 bits of instruction register)
- two 0's

(creates a 32-bit address)

Why 2 0's?

Other instruction formats: non-MIPS

Other possible formats

- older formats were designed to minimize the number of bits in an instruction
- 3-register format (MIPS)

```
addu    $r10,$r8,$r9
```

- 2-register format (CISC)

```
add2 $r1, $r2
```

semantics: `R[1] = R[1] + R[2]` (like += in C)

- same register used for source AND target
- fewer bits necessary

- 1-register format (accumulator)

```
add1 $r2
```

semantics: `Acc = Acc + R[2]`

- accumulator: special register used to hold results, implicit in instruction

- 0-register format (stack)

```
add0
```

semantics:

```
Stack[Top-4] = Stack[Top] + Stack[Top-4]
Top = Top - 4
```

- replace top of stack with sum of top 2 values
- requires push and pop operations
- must go back to memory to reuse value