# Instruction Types

**Computation**:

- arithmetic (e.g., add)
- logical (e.g., xor)
- compare (e.g., set if not equal)

**Data transfer**:

- load
- store

**Control**

- branch
- jump

# MIPS Computation Instructions

Opcode rd, rs, rt

Opcode rd, rs, immed

- rd: destination register (modify)
- rs: source register (read-only)
- rt: source/destination register (read-only/modify)
- immed: 16-bit value (constant)

# MIPS Computation Instructions

Some examples:

```
add     $8, $9, $10      # $8 = $9+$10
addi    $t0, $t1, 20     # $t0 = $t1+20
addu    $8, $9, $10      # $8 = $9+$10
sub     $t5, $0, $t5     # $t5 = -$t5
and     $8, $9, $10      # $8 = $9&$10
slt     $8, $9, $10      # if $9<$10, $8 = 1,
                         #     else $8 = 0
slti    $8, $9, -6       # if $9<-6, $8 = 1,
                         #     else $8 = 0
```

The GPRs are used to store the result of a condition.

Alternative architecture: **condition codes**

- special 1-bit registers that store the result of specific conditions
  - whether the result is zero
  - whether the result is negative

The machine does not know if a value is signed or unsigned (the bag of bits) --- you have to specify this by using the appropriate instruction

# Instruction Encoding

ISA defines the formats for instructions

- what fields they contain
- the size of the fields
- the field values & what the values signify

Being a RISC, MIPS has few (3) instruction formats

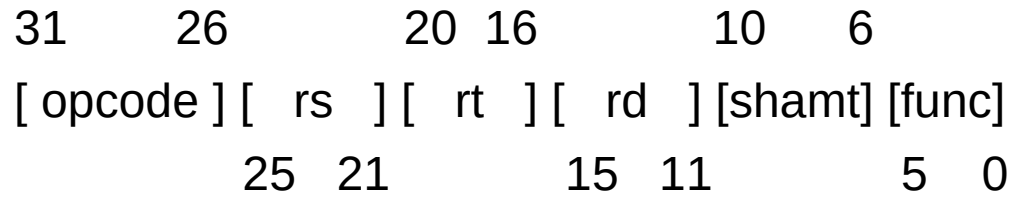
- all instructions are the same length, 32 bits
- most formats have similar fields for example, an opcode, at least one source register
- fields that are common to more than one format have the same location in the instruction for example, the opcode is always first
- fields that are common to more than one format are the same size for example, the opcode is always 6 bits

Shows us how the CPU processes instructions

- bridge between architecture & implementation

# R-type Format

For arithmetic, logical, comparative instructions with register operands

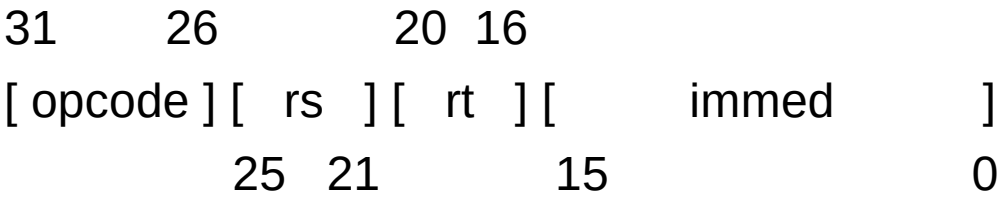



- **opcode, func** = operation
  - opcode = a computational instruction
  - func = which computation
- **rs, rt** = source operands
- **rd** = destination operand
  - **shamt** = shift distance in bits

```
add $8, $9, $10
[  0  ][  9   ][  10   ][  8   ][  X  ][  32 ]
xor $11, $12, $13
[  0  ][  12  ][  13   ][  11  ][  X  ][  38 ]
sll $10, $16, 4
[  0  ][  X   ][  16   ][  10  ][  4  ][  0  ]
```

# I-type Format

For arithmetic, logical, comparative instructions with one register operand & one constant operand




- **opcode** = operation
  - opcode = a computational instruction
- **rs** = source operand
- **rt** = destination operand
- **immed** = constant, $\pm 2^{15}$
  - sign-extended when used (replicate msb)

Using an immediate value is faster than loading the constant from memory & saves using a register

```
ori $8, $9, -256
[  13 ][  9   ][  8   ][      -256      ]
```