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

<table>
<thead>
<tr>
<th>Opcode</th>
<th>rd, rs, rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>rd, rs, immed</td>
</tr>
<tr>
<td></td>
<td>rd: destination register (modify)</td>
</tr>
<tr>
<td></td>
<td>rs: source register (read-only)</td>
</tr>
<tr>
<td></td>
<td>rt: source/destination register (read-only/modify)</td>
</tr>
<tr>
<td></td>
<td>immed: 16-bit value (constant)</td>
</tr>
</tbody>
</table>
MIPS Computation Instructions

Some examples:

\[
\begin{align*}
\text{add} & \quad $8, \; $9, \; $10 \quad \# $8 = $9 + $10 \\
\text{addi} & \quad $t0, \; $t1, \; 20 \quad \# $t0 = $t1 + 20 \\
\text{addu} & \quad $8, \; $9, \; $10 \quad \# $8 = $9 + $10 \\
\text{sub} & \quad $t5, \; $0, \; $t5 \quad \# $t5 = -$t5 \\
\text{and} & \quad $8, \; $9, \; $10 \quad \# $8 = $9 \& $10 \\
\text{slt} & \quad $8, \; $9, \; $10 \quad \# \text{if } $9 < $10, \; $8 = 1, \text{ else } $8 = 0 \\
\text{slti} & \quad $8, \; $9, \; -6 \quad \# \text{if } $9 < -6, \; $8 = 1, \text{ else } $8 = 0
\end{align*}
\]

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

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>func</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>20</td>
<td>16</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>

- add $8, $9, $10

| 0 | 9 | 10 | 8 | unused | 32 |

- xor $11, $12, $13

| 0 | 12 | 13 | 11 | unused | 38 |

- sll $10, $16, 4

| 0 | unused | 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, ±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 |