MIPS I-Type Memory Instructions

- I-Type also encode memory access
  - Store: \( M[R[rs] + \text{Immediate}] = R[rt] \)
  - Load: \( R[rt] = M[R[rs] + \text{Immediate}] \)
- MIPS has load/stores for byte, half word, and word
- Sub-word loads can also be signed or unsigned
  - Signed loads sign-extend the value to fill a 32 bit register.
  - Unsigned zero-extend the value.
- “immediate” is a 16 bit offset
  - Useful for accessing structure components
  - It is signed.

Examples

- \( \text{lw } \$t0, 4(\$t1) \)
  - \( \text{opcode} = 0x23 \)
- \( \text{sb } \$t0, -17(\$t1) \)
  - \( \text{opcode} = 0x28 \)
MIPS J-Type Instructions

- J-Type encodes the jump instructions
- Plain Jump
  - JumpAddress = \{PC+4[31:28],Address,2'b0\}
  - Address replaces *most* of the PC
  - PC = JumpAddress
- Jump and Link
  - R[ra] = PC + 8; PC = JumpAddress;
- J-Type also encodes misc instructions
  - syscall, interrupt return, and break (more later)

Examples
- j $t0
- PC = R[8]
- opcode = 0x2
- jal $t0
- R[31] = PC + 8
- PC = R[8]
Consider a hypothetical ISA that supports 16 instructions and 16 registers (0-15), and 16 bit instructions. It will have just one instruction format. How many register operands can the instruction format encode?

<table>
<thead>
<tr>
<th>Selection</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>&lt;= 1</td>
</tr>
<tr>
<td>B</td>
<td>&lt;= 2</td>
</tr>
<tr>
<td>C</td>
<td>&lt;= 3</td>
</tr>
<tr>
<td>D</td>
<td>&lt;= 4</td>
</tr>
<tr>
<td>E</td>
<td>None of the above</td>
</tr>
</tbody>
</table>
How many of these instruction formats would work for an ISA with 36 instructions and 32 registers?

<table>
<thead>
<tr>
<th>Selection</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
</tr>
</tbody>
</table>

I. 6 4 4 4
II. 5 5 5 5
III. 6 5 5
IV. 7 5 5
V. 6 6 6 6

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>Bits</td>
<td>Bits</td>
<td>Bits</td>
</tr>
<tr>
<td>Opcode</td>
<td>Register</td>
<td>Register</td>
<td>Register</td>
</tr>
</tbody>
</table>

Selection operands:

- A: 1
- B: 2
- C: 3
- D: 4
- E: 5
Convert this MIPS machine instruction to assembly:

```
0010 0001 0001 0000 0000 0000 0010 0010
```

<table>
<thead>
<tr>
<th>Selection</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>addi r16, r8, #34</td>
</tr>
<tr>
<td>B</td>
<td>addi r8, r16, #34</td>
</tr>
<tr>
<td>C</td>
<td>sub r8, r16, r31</td>
</tr>
<tr>
<td>D</td>
<td>sub r31, r8, r16</td>
</tr>
<tr>
<td>E</td>
<td>None of the above</td>
</tr>
</tbody>
</table>
Executing a MIPS program

- All instructions have
  - <= 1 arithmetic op
  - <= 1 memory access
  - <= 2 register reads
  - <= 1 register write
  - <= 1 branch

- All instructions go through all the steps

- As a result
  - Implementing MIPS is (sort of) easy!
  - The resulting HW is (relatively) simple!