Memory organization

Byte ordering within words

- Little-endian: word address is LSB
- Big-endian: word address is MSB

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

word address A

MIPS addressing modes

MIPS addressing modes
Addressing modes are the ways of specifying a location of an operand, a location in memory, or the address of an instruction for a control transfer.

- register addressing
- immediate addressing
- base addressing
- PC relative addressing
- indirect addressing
- pseudodirect addressing

Memory organization and addressing

- memory is viewed as a single-dimensional array of bytes individually addressable. 32 bit words are aligned to 4 byte boundaries (instructions).

- 2^32 bytes, with addresses from 0 to 2^32 - 1
- 2^30 words, with addresses 0, 4, 8, ... 2^32 - 4
**Immediate Addressing:** the operand is embedded inside the encoded instruction

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate value</th>
</tr>
</thead>
</table>

16-bit signed integers range from __________ to __________
16-bit unsigned integers range from __________ to __________

Example: addi $t0, $t1, 77

---

**Register addressing:** specify the register number

Example: add $t0, $t1, $t2

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
</table>

5 bits \(\leftrightarrow\) 32 registers

---

**Immediate addressing:**

<table>
<thead>
<tr>
<th>CPU REG</th>
<th>MEMORY</th>
</tr>
</thead>
</table>

---

**Register addressing:**

<table>
<thead>
<tr>
<th>CPU REG</th>
<th>MEMORY</th>
</tr>
</thead>
</table>
MIPS addressing modes

**PC-relative addressing:** the value in the immediate field (a two's complement 16 bit number) is interpreted as an offset in number of instructions with respect to the address of the next instruction (PC + 4).

\[
\text{ADDR} = \text{offset} \ll 2 + \text{PC} + 4
\]

---------------

**Base (or displacement) addressing:** the address is the sum of the immediate (16 bits, two's complement) and the value in a register (rs)

\[
\text{ADDR} = \text{byte offset} + (\text{rs})
\]

---------------

Example: lw $s0, 12($s1)

---

**Example:**

```plaintext
.. 
addi $s0, $s0, 1
beq $s0, $s1, label
addi $s0, $s0, 1
addi $s0, $s0, 1
label: addi $s0, $s0, 1
addi $s0, $s0, 1
...
```

The binary coding of the line `beq $s0, $s1, label` is actually `0x12110002`, i.e. 2 instructions after the next one (PC+4).

NOTE on SPIM; SPIM uses the current PC and not the next
**MIPS addressing modes**

**Indirect addressing:**

The address is the value in a register. Also called "register indirect".

In MIPS, it is used in the `jump register` instructions (jr, jalr)

Example: `jr $ra`

AND it is a special case of base addressing (offset = 0)

Example: `lw $s0, 0($s1)`
PowerPC additional addressing modes

Indexed addressing:

The address is the sum of two registers

Example (p. 175):

The MIPS code:

```
add $t0, $a0, $a3  # $a0 has base of an array, $a3 is an index
lw $t1, 0($t0)    # $t1 gets ($a0+$a3)
```

- can be replaced by the single PowerPC instruction
- does not require additional hardware

MIPS addressing modes

Pseudo-direct addressing:

```
Instruction
```

Note that:
Addressing mode ≠ instruction type (format)

Example:

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>TYPE</th>
<th>ADDRESSING MODE(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>addi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
One more addressing mode

One more addressing mode (not in PowerPC)

Memory indirect addressing: a register points to a memory location that points to the operand.

Example, PDP 11 assembly language:  
`move $s3, 0($s3)`

Update addressing: base addressing with automatic base register increment.

Example (p. 176):

The MIPS code:

```
1w  $t0, 4($s3)  # $t0 gets ($s3+4)
addi $s3, $s3, 4  # $s3 points to the next word
```

can be replaced by the single PowerPC instruction

```
lwu $t0, 4($s3)  # $t0 gets ($s3+4), $s3 = $s3+4
```

• does not require new hardware, but
• does require to write two registers at the same time

---

Homework

Recommended exercises

• Ex. 3.12 to 3.17, 3.19, 3.20