MIPS Addressing Modes and Memory Architecture

(Second Edition: Section 3.8
Fourth Edition: Section 2.10)
from Dr. Andrea Di Blas’ notes
Memory Organization and Addressing

- Memory may be viewed as a single-dimensional array of individually addressable bytes. 32-bit words are aligned to 4 byte boundaries.
  - $2^{32}$ bytes, with addresses from 0 to $2^{32} - 1$.
  - $2^{30}$ words with addresses 0, 4, 8, ..., $2^{32} - 4$
Byte ordering within words

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

Ex: 0000 0001 0010 0011 0100 0101 0110 0111
MIPS addressing modes

Addressing modes are the ways of specifying an operand or a memory address.

- Register addressing
- Immediate addressing
- Base addressing
- PC-relative addressing
- Indirect addressing
- Direct addressing (almost)
Register addressing

- Operands are in a register.
- Example: add $3, $4, $5
- Takes $n$ bits to address $2^n$ registers

| op | rs | rt | rd | shamt | funct |
Register addressing

- op
- rs
- rt
- rd
- shamt
- funct

ALU

registers

memory
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 bits

16 bit two’s-complement number:

\[-2^{15} - 1 = -32,769 < \text{value} < +2^{15} = +32,768\]
Immediate addressing

Example is addi or similar
Base (or Base-offset or displacement ) Addressing

- The address of the operand is the sum of the immediate and the value in a register (rs).
- 16-bit immediate is a two’s complement number
- Ex: lw $15,16($12)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>Immediate value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Base addressing

```
\begin{array}{c|c|c|c}
\text{op} & \text{rs} & \text{rt} & \text{Immediate value} \\
\hline
\end{array}
```

- \text{Immediate value} is a 16-bit value.

**Example:**
- `lw $8,128($5)`
  - Effective address = \text{base} + \text{immediate}
  - Address calculation: \text{base} = $5$, \text{immediate} = 128
  - Effective address = $5 + 128 = 133$
**PC-relative addressing:** the value in the immediate field is interpreted as an offset of the next instruction (PC+4 of current instruction)

Example: `beq $0,$3,Label`
PC-relative addressing

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

16 bits

Shifted by 2 and Sign-extended

beq $0,$5,Label
**Detail of MIPS PC-Relative**

<table>
<thead>
<tr>
<th>address</th>
<th>instruction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>40000008</td>
<td>addi $5, $5, 1</td>
<td>PC = 0x4000000C</td>
</tr>
<tr>
<td>4000000C</td>
<td>beq $0, $5, label</td>
<td>PC+4 = 0x40000010</td>
</tr>
<tr>
<td>40000010</td>
<td>addi $5, $5, 1</td>
<td>Add 4*2 = 0x00000008</td>
</tr>
<tr>
<td>40000014</td>
<td>addi $5, $5, 1</td>
<td>Eff. Add. = 0x40000018</td>
</tr>
<tr>
<td>40000018</td>
<td>label addi $5, $5, 1</td>
<td></td>
</tr>
<tr>
<td>4000001C</td>
<td>addi $5, $5, 1</td>
<td></td>
</tr>
<tr>
<td>40000020</td>
<td>etc...</td>
<td></td>
</tr>
</tbody>
</table>

Binary code to beq $0,$5, label is 0x10050002, which means 2 instructions from the next instruction.

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>Immediate value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010</td>
<td>00000</td>
<td>00101</td>
<td>00000000000000010</td>
</tr>
</tbody>
</table>
Register Direct Addressing: the value the (memory) effective address is in a register. Also called “Indirect Addressing”.

Special case of base addressing where offset is 0.

Used with the jump register instructions (jr, jalr).

Example: jr $31

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>rs</td>
<td>00000</td>
<td>00000</td>
<td>00000</td>
<td>001000</td>
</tr>
</tbody>
</table>
Register Direct

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

Registers

program counter

jr $5

memory
**Direct Addressing**: the address is “the immediate”. 32-bit address cannot be embedded in a 32-bit instruction.

**Pseudodirect addressing**: 26 bits of the address is embedded as the immediate, and is used as the instruction offset within the current 256MB (64MWord) region defined by the MS 4 bits of the PC.

Example: `j Label`

<table>
<thead>
<tr>
<th>op</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26 25</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PC:</th>
<th>0111</th>
<th>0001 ...</th>
<th>...</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>offs:</td>
<td>0101 0001 0100 0010 1111 0101 10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shift:</td>
<td>0111 0101 0001 0100 0010 1111 0101 10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR:</td>
<td>0111 0101 0001 0100 0010 1111 0101 10 00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pseudodirect addressing

```
op  offset
31 26 25 0
```

```
xxxx 00
```

j Label

program counter
Caution: Addressing mode is not Instruction type

- Addressing mode is how an address (memory or register) is determined.
- Instruction type is how the instruction is put together.
- Example: addi, beq, and lw are all I-types instructions. But
  - addi uses immediate addressing mode (and register)
  - beq uses pc-relative addressing (and register)
  - lw uses base addressing (and register)
MIPS Addressing Modes

1. REGISTER: a source or destination operand is specified as content of one of the registers $0$–$31$.

2. IMMEDIATE: a numeric value embedded in the instruction is the actual operand.

3. PC-RELATIVE: a data or instruction memory location is specified as an offset relative to the incremented PC.

4. BASE: a data or instruction memory location is specified as a signed offset from a register.

5. REGISTER-DIRECT: the value the effective address is in a register.

6. PSEUDODIRECT: the memory address is (mostly) embedded in the instruction.
PowerPC and x86 addressing modes and instructions

Indexed Addressing: The address is the sum of two registers. (note indexed addressing is different here than usually used)

MIPS code: add $10, $20, $13 ;$20 is base,$13 is index
lw $5, 0($10)

PowerPC: lw $5, $20+$13 ; $5 ← ($20+$13)

Saves instruction for incrementing array index.
No extra hardware.
# PowerPC: Indexed Addressing

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

## Diagram

- **Registers**
- **ALU**
- **Memory**
Additional PowerPC addressing mode - 2

**Update Addressing:** base addressing with automatic base register increment.

**MIPS code:**

```
lw $10, 4($13) ; $10 ← Mem[$10+4]
addi $13, $13, 4 ; $13 ← $13+4
```

**PowerPC:**

```
lwu $10, 4($13) ; $10 ← Mem[$10+4]
; and $13 ← $13+4
```

Requires that two registers be written at the same time → more hardware.
PowerPC: Update Addressing

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

16 bits

memory

eff. add.

registers

ALU

(for base and index addressing)
Additional non-RISC addressing mode

**Memory Indirect Addressing:** read effective address from memory. (Usually PC-relative addressing is used to get the effective address from memory).

RISC code:

\[
\begin{align*}
lw \ $10, 0($13) \\
lw \ $5, 0($10)
\end{align*}
\]

CISC:

\[
ldi \ $5, \text{Label} \quad ; \quad $5 \leftarrow \text{Mem[Label]}
\]

Requires two sequential data memory accesses.
CISC: Memory Indirect Addressing

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

registers \(\rightarrow\) (or from PC) \(\rightarrow\) memory

CPU:

- ALU
- eff. add.

(CMPE 110 – Spring 2011 – J. Ferguson)