MIPS Assembly Programming
MIPS

- In MIPS microprocessors ...the memory (RAM) access is allowed only with **Load** and **Store** instructions.

- MIPS microprocessors (MIPS R2000 and ARM) have a **«Load/Store»** architecture.
Accessing the RAM

- **Load instructions**: Read data from RAM and copy it to a register. \((\text{lw} \ $t0, \text{Memory-Address})\)

  ![Diagram of load instruction]

- **Store instructions**: Write data from a register to RAM. \((\text{sw} \ $t0, \text{Memory-Address})\)

  ![Diagram of store instruction]
RAM and Registers

• In our machine, we have only 32 registers
• For large data structures (Arrays, Images, …)
• The 32 registers are not enough
• We need more storage…
• Must use the System Memory (RAM)
  • Memory (RAM) is large
  • RAM to Registers and back … it is very slow with respect to the speed of the registers [only] use.
• Commonly used variables are kept in registers.
Load instructions

<table>
<thead>
<tr>
<th>Op</th>
<th>Operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>la</td>
<td>des, addr</td>
<td>Load the address of a label.</td>
</tr>
<tr>
<td>lb(u)</td>
<td>des, addr</td>
<td>Load the byte at addr into des.</td>
</tr>
<tr>
<td>lh(u)</td>
<td>des, addr</td>
<td>Load the halfword at addr into des.</td>
</tr>
<tr>
<td>li</td>
<td>des, const</td>
<td>Load the constant const into des.</td>
</tr>
<tr>
<td>lui</td>
<td>des, const</td>
<td>Load the constant const into the upper halfword of des, and set the lower halfword of des to 0.</td>
</tr>
<tr>
<td>lw</td>
<td>des, addr</td>
<td>Load the word at addr into des.</td>
</tr>
<tr>
<td>lwl</td>
<td>des, addr</td>
<td></td>
</tr>
<tr>
<td>lwr</td>
<td>des, addr</td>
<td></td>
</tr>
<tr>
<td>ulh(u)</td>
<td>des, addr</td>
<td>Load the halfword starting at the (possibly unaligned) address addr into des.</td>
</tr>
<tr>
<td>ulw</td>
<td>des, addr</td>
<td>Load the word starting at the (possibly unaligned) address addr into des.</td>
</tr>
</tbody>
</table>
# Store Instructions

<table>
<thead>
<tr>
<th>Op</th>
<th>Operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sb</td>
<td>src1, addr</td>
<td>Store the lower byte of register src1 to addr.</td>
</tr>
<tr>
<td>sh</td>
<td>src1, addr</td>
<td>Store the lower halfword of register src1 to addr.</td>
</tr>
<tr>
<td>sw</td>
<td>src1, addr</td>
<td>Store the word in register src1 to addr.</td>
</tr>
<tr>
<td>swl</td>
<td>src1, addr</td>
<td>Store the upper halfword in src to the (possibly unaligned) address addr.</td>
</tr>
<tr>
<td>swr</td>
<td>src1, addr</td>
<td>Store the lower halfword in src to the (possibly unaligned) address addr.</td>
</tr>
<tr>
<td>ush</td>
<td>src1, addr</td>
<td>Store the lower halfword in src to the (possibly unaligned) address addr.</td>
</tr>
<tr>
<td>usw</td>
<td>src1, addr</td>
<td>Store the word in src to the (possibly unaligned) address addr.</td>
</tr>
</tbody>
</table>
MIPS addressing
MIPS addressing and modes

• Addressing; General methods to access the data in the CPU or the RAM:
  – Register:  \texttt{add \$s0, \$t2, \$t3}
  – Immediate: \texttt{addi \$s4, \$t5, 34}
  – Indexed (based)

• MIPS uses indexed (based) addressing to access (Load \texttt{lw} … Store\texttt{sw}) the RAM.
lw ... sw

lw $rd, offset($rs)

sw $rs, offset($rd)
MIPS memory access instructions

- **WORD** *(Address in memory must be word-aligned)*
  - `lw`; Loads a word from a location in memory to a register
  - `sw`; Store a word from a register to a location in memory

- **BYTE** *(Address not aligned-Only one byte is loaded from memory)*
  - `lb`; Loads a byte from a location in memory to a register. Sign extends this result in the register.
  - `sb`; Store the least significant byte of a register to a location in memory.
Memory word-alignment

MIPS requires that all words start at byte addresses and are multiples of 4 bytes (4 x 8 = 32-bits)

```assembly
.align     # directive the next datum on a 2^n byte boundary.
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12348000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x12348004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x12348008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x1234800C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x12340010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Indexed (Based) addressing

• The address operand specifies an **Immediate (signed constant or Offset)** and a **rs (register source)** that holds the based-address:

```markdown
lw $t2, 4($t0)
```

• The actual memory location from where the operand is retrieved for an instruction is the **Effective Address (EA)**.

• The Effective Address (EA) is determined by **adding** the offset to the Register (based-address)

```markdown
EA ← Mem{ $(register) + sign-ext_{32}( offset ) }
```

```markdown
$t2 ← Mem{ $t0 + 00000000000000000000000000000100 }
```

MIPS uses **INDEXED ADDRESSING**

http://logos.cs.uic.edu/366/notes/
lw $t2, 4($t0)

<table>
<thead>
<tr>
<th>op code</th>
<th>rs ($t0)</th>
<th>rt ($t2)</th>
<th>Address/Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>100011</td>
<td>01000</td>
<td>01010</td>
<td>0000 0000 0000 0100</td>
</tr>
</tbody>
</table>

rs is the first source register
rt is the second source register
lw $t2, 4($t0)
Example

loads a word from a location in memory to a register

lw
### Word addressable memory

- Each 32-bit data word has a unique address

<table>
<thead>
<tr>
<th>Word Address</th>
<th>Data</th>
</tr>
</thead>
</table>
| 0000000000   | A B C D E F 7 8 | Word 0
| 0000000001   | F 2 F 1 A C 0 7 | Word 1
| 0000000002   | 0 1 E E 2 8 4 2 | Word 2
| 0000000003   | 4 0 F 3 0 7 8 8 | Word 3
Reading Word from Memory (**lw**)

- Memory read is called **load**
- **Mnemonic:** **load** word (**lw**)
- **Format:**
  
  \[ \text{lw} \; \$t2, \; 1(\$t0) \]

- Effective Address calculation:
  - add **base-address** ($\$t0$) to the **offset** (1)
  - Effective-address: \$t2 \leftarrow \text{Mem}[\$t0+1]

- **Result:**
  - $\$t2$ holds the value at effective-address ($\$t0+1$)

Any register may be used as base address.
Example: \texttt{lw}

- **Read** a word of data at memory address 0x00000001 into register: $t2$
  - Effective Address:
    \[ $t2 \leftarrow \text{Mem}[t0+1] = 0x00000001 \]
  - $t2$ holds the value: 0xF2F1AC07 after load

**Assembly code**

\[ \texttt{lw } t2, 1(t0) \quad \#\text{read memory word 1 into } t2 \]

\[
\begin{array}{|c|c|}
\hline
\text{Reg} & \text{value} \\
\hline
\texttt{t0} & \quad \\
\texttt{...} & \quad \\
\texttt{t2} & 0xF2F1AC07 \\
\hline
\end{array}
\]

1. Go to the memory address \([1+t0]\)
2. Take the data and put them in the register ($t2$).
Writing (storing) word to RAM

Store a word from a register to a location in memory

SW
Indexed (Based) Addressing; Store

Store a word from a register to a location in memory.

1. Go to the memory address \[4+\text{t0}\]
2. Put the data of the register ($t2$), to the memory address \[4+\text{t0}\]

\[
\text{sw } \text{t2}, \ 4 (\text{t0})
\]

\[\text{t2 }\rightarrow \text{Mem}[\text{t0}+4]\]
Example: \textbf{sw}

- Memory write is called \textit{store}
- \textbf{Mnemonic:} \textit{store} \textit{word} (\textit{sw})
- Format:

\[
\text{sw} \ \$s0, \ 3(\$t0)
\]

- Effective Address (EA): \$s0 \rightarrow \text{Mem}[\$t0+3].
Writing Word-Addressable Memory

- Example: Write (store) the value in \$s0 into Memory Address: 0x00000003
  - Effective-Address: \( \text{Mem}[\$t0+3] \Rightarrow 0x00000000+3 = 0x00000003 \)
  - To the above address load the word: 0x40F30788

**Assembly code**

\( \text{sw} \ \$s0, \ 3(\$t0) \)  \# write the value in \$s0 to memory word 3

\( \$t0 = 0x00000000 \) (base address)

<table>
<thead>
<tr>
<th>Register File</th>
<th>Word Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg</td>
<td>value</td>
<td></td>
</tr>
<tr>
<td>$t0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s0</td>
<td>0x40F30788</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{c|c|c}
\hline
\text{Word Address} & \text{Data} & \text{Word} \\
\hline
00000000 & 4 0 F 3 0 7 8 8 & 3 \\
00000001 & 0 1 E E 2 8 4 2 & 2 \\
00000002 & F 2 F 1 A C 0 7 & 1 \\
00000003 & A B C D E F 7 8 & 0 \\
\hline
\end{array}
\]
Load immediate, Load RAM address

`lw $t0, x`  # Load contents of RAM location x into $t0

$x$: (Memory Address, where a value is stored)

`sw $t0, x`  # Store contents of x into $t0."
Example-1
Example-1; \textit{lw-sw}

\begin{verbatim}
# load-store example-1

.text
.globl main
main:
    lw $t0, x  # Load contents of RAM location \texttt{x} into register \texttt{$t0}
    addi $t0, $t0, 3
    sw $t0, x
    lw $t1, x
    li $v0, 10
    syscall

.data
x: .word 9  # RAM location \texttt{x}

\end{verbatim}

\texttt{$t0 = ?}$
\texttt{$t1 = ?$}
Example-1; \texttt{lw-sw}

\begin{verbatim}
# load-store example-1

.text
.globl main

main:
  lw  $t0, x  # Load contents of RAM location x into register $t0
  addi $t0, $t0, 3
  sw  $t0, x
  lw  $t1, x
  li   $v0, 10
  syscall

.data

x:    .word 9
\end{verbatim}
Example-2
Example-2; \texttt{lw-sw}

\texttt{.text}
\texttt{.globl main}
\texttt{main:}
\texttt{lw \ $t0, x}
\texttt{addi \ $t0, \ $t0, 3}
\texttt{sw \ $t0, x}
\texttt{lw \ $t1, x}
\texttt{lw \ $t0, x}
\texttt{addi \ $t2, \ $t1, 3}
\texttt{li \ $v0, 10}
\texttt{syscall}

\texttt{.data}
\texttt{x: .word 9}

$\texttt{t0} = ?$
$\texttt{t1} = ?$
$\texttt{t2} = ?$
Example-2; lw-sw

# load-store example-2

.text
.globl main

main:
    lw     $t0, x
    addi   $t0, $t0, 3
    sw     $t0, x
    lw     $t1, x
    addi   $t2, $t1, 3
    li $v0, 10
    syscall

.data

x: .word 9

$t0 = 12
$t1 = 12
$t2 = 15
.word/.byte/.space

(Word):
\texttt{x: .word 1, 2, 3, ...}

(Bytes):
\texttt{array: .byte 'x', 'y', ...}

(Array-Space):
\texttt{array: .space x}
.word/.byte/.space

(Word):
x: .word 1, 2, 3, ...

(Bytes):
array: .byte 'x','y', ...

array: .space 12

# Allocate 12 consecutive bytes, with storage uninitialzed
# Create a 12-element character array
# Equivalent to a 3-element integer array (3 x 4 = 12)
Based or Indexed addressing

- **lw** $t2, 4($t0) # $t2 ← Mem[$t0 + 4]
  - load word at RAM address ($t0+4) into register $t2
  - $t0 contains the base address
  - "4" gives offset from address in register $t0

- **sw** $t2, 4($t0) # $t2 → Mem[$t0 + 4]
  - store word in register $t2 into RAM at address ($t0 + 4)
  - $t0 contains the base address
  - negative offsets are fine

- Note: based addressing is especially useful for:
  - arrays; access elements as offset from base address
  - stacks; easy to access elements at offset from stack pointer or frame pointer

http://logos.cs.uic.edu/366/notes/
Offset

Based address; the offset is: 0.
`lw    $t1, 0($t0)`

For characters the offset is: 1.
`lw    $t1, 1($t0)`

For integers the offset is: 4.
`lw    $t1, 4($t0)`

1-Byte/character
4-Bytes (1-Word)
Example-3
# load-store WORD example-3

.text
.globl main
main:
la $s0, 0xFFFF0010  # Load FFFF0010 address to $s0
li $t0, 123
sw $t0, 0($s0)     # Store to memory location FFFF0010
lw $t1, 0($s0)      # Load from memory location FFFF0010 to reg. $t1
li $t2, 5
add $t3, $t2, $t1
li $v0, 10
syscall

$t0 = ?
$t1 = ?
$t2 = ?
$t3 = ?
Result

```
Load-Store.asm

# load-store WORD example-1
.text
.globl main
main:
la $s0, 0xFFFF0010 # Load FFFF0010 address to $s0
li $t0, 123
sw $t0, 0($s0) # Store to memory location FFFF0010
lw $t1, 0($s0) # Load from memory location FFFF0010 to reg. $t1
li $t2, 5
add $t3, $t2, $t1
li $v0, 10
syscall
```

<table>
<thead>
<tr>
<th>Registers</th>
<th>Coproc 1</th>
<th>Coproc 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Number</td>
<td>Value</td>
</tr>
<tr>
<td>$zero</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$at</td>
<td>1</td>
<td>-65536</td>
</tr>
<tr>
<td>$v0</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>$v1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$a0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>$a1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$a2</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>$a3</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>$t0</td>
<td>8</td>
<td>123</td>
</tr>
<tr>
<td>$t1</td>
<td>9</td>
<td>123</td>
</tr>
<tr>
<td>$t2</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>$t3</td>
<td>11</td>
<td>128</td>
</tr>
<tr>
<td>$t4</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>$t5</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>$t6</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>$t7</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>$s0</td>
<td>16</td>
<td>-65520</td>
</tr>
<tr>
<td>$s1</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>$s2</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>$s3</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>$s4</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>$s5</td>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>$s6</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>$s7</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>$t8</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>$t9</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>$k0</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>$k1</td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>$gp</td>
<td>28</td>
<td>268468224</td>
</tr>
<tr>
<td>$sp</td>
<td>29</td>
<td>2147483648</td>
</tr>
<tr>
<td>$fp</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>$ra</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>pc</td>
<td></td>
<td>4194340</td>
</tr>
</tbody>
</table>

$t0 = 123
$t1 = 123
$t2 = 5
$t3 = 128
Example-4
Example-4

# load-store WORD example-4

.text
.globl main
main:
la $s0, 0xFFFF0010  # Load FFFF0010 address to $s0
li $t0, 15
sw $t0, 0($s0)    # Store to memory location FFFF0010
lw $t1, 0($s0)    # Load from memory location FFFF0010 to reg. $t1
li $t2, 9
sw $t2, 0($s0)    # Store to memory location FFFF0010
lw $t3, 0($s0)    # Load from memory location FFFF0010 to reg. $t3
add $t4, $t1, $t3
li $v0, 10
syscall
Example-4 (Trace)

```
.text
.globl main

main:
la $s0, 0xFFFF0010
li $t0 15
sw $t0, 0($s0)
lw $tl, 0($s0)
li $t2 9
sw $t2, 0($s0)
lw $t3, 0($s0)
add $t4, $tl, $t3
li $v0, 10
syscall
```

<table>
<thead>
<tr>
<th></th>
<th>$t0</th>
<th>$t1</th>
<th>$t2</th>
<th>$t3</th>
<th>$t4</th>
</tr>
</thead>
<tbody>
<tr>
<td>(8)</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(10)</td>
<td></td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(12)</td>
<td></td>
<td></td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(14)</td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>(16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>24</td>
</tr>
</tbody>
</table>
Example-4

```assembly
# load-store: Example-4
.text
.globl main
main:
la $s0, 0x0FFFF0010  # Load FFFFF0010 address to $s0
li $t0 15
sw $t0, 0($s0)     # Store to memory location FFFFF0010
lw $t1, 0($s0)     # Load from memory location FFFFF0010 to reg. $t1
li $t2 9
sw $t2, 0($s0)     # Store to memory location FFFFF0010
lw $t3, 0($s0)     # Load from memory location FFFFF0010 to reg. $t3
add $t4, $t1, $t3
li $v0, 10
syscall
```

$t0 = 15
$t1 = 15
$t2 = 9
$t3 = 9
$t4 = 24
Example-5

# load-store WORD example-5

.text
.globl main

main:
la  $s0, 0xFFFF0010
li  $t0, 15
sw  $t0, 0($s0)
lw  $t1, 0($s0)

li  $t2, 9
sw  $t2, 4($s0)
lw  $t3, 4($s0)
add $t4, $t1, $t3

li  $v0, 10
syscall

$t0 = 15
$t1 = 15
$t2 = 9
$t3 = 9
$t4 = 24
Next Arrays