C335
Computer Structures

MIPS Instructions (Part #5)

Dr. David R. Surma

Department of Computer and Information Sciences

Adapted from Morgan Kaufmann, Dr. L. Zhang and others
## Review: MIPS Instructions, so far

<table>
<thead>
<tr>
<th>Category</th>
<th>Instr</th>
<th>OpCd</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arithmetic</strong></td>
<td>add</td>
<td>0 &amp; 20</td>
<td>add $s1, $s2, $s3</td>
<td>$s1 = $s2 + $s3</td>
</tr>
<tr>
<td>(R &amp; I format)</td>
<td>subtract</td>
<td>0 &amp; 22</td>
<td>sub $s1, $s2, $s3</td>
<td>$s1 = $s2 - $s3</td>
</tr>
<tr>
<td></td>
<td>add immediate</td>
<td>8</td>
<td>addi $s1, $s2, 4</td>
<td>$s1 = $s2 + 4</td>
</tr>
<tr>
<td><strong>Data transfer</strong></td>
<td>load word</td>
<td>23</td>
<td>lw $s1, 100($s2)</td>
<td>$s1 = Memory($s2+100)</td>
</tr>
<tr>
<td>(I format)</td>
<td>store word</td>
<td>2b</td>
<td>sw $s1, 100($s2)</td>
<td>Memory($s2+100) = $s1</td>
</tr>
<tr>
<td><strong>Cond. branch</strong></td>
<td>br on equal</td>
<td>4</td>
<td>beq $s1, $s2, L</td>
<td>if ($s1==$s2) go to L</td>
</tr>
<tr>
<td>(I format)</td>
<td>br on not equal</td>
<td>5</td>
<td>bne $s1, $s2, L</td>
<td>if ($s1 !=$s2) go to L</td>
</tr>
<tr>
<td></td>
<td>set on less than immediate</td>
<td>a</td>
<td>slti $s1, $s2, 100</td>
<td>if ($s2&lt;100) $s1=1; else $s1=0</td>
</tr>
<tr>
<td>(R format)</td>
<td>set on less than</td>
<td>0 &amp; 2a</td>
<td>slt $s1, $s2, $s3</td>
<td>if ($s2&lt;$s3) $s1=1; else $s1=0</td>
</tr>
<tr>
<td><strong>Uncond. jump</strong></td>
<td>jump</td>
<td>2</td>
<td>j 2500</td>
<td>go to 10000</td>
</tr>
<tr>
<td></td>
<td>jump register</td>
<td>0 &amp; 08</td>
<td>jr $t1</td>
<td>go to $t1</td>
</tr>
</tbody>
</table>
MIPS Organization

Processor

Register File
- src1 addr
- src2 addr
- dst addr
- write data
- src1 data
- src2 data
- 32 registers ($zero - $ra)

32 bits

Memory
- read/write addr
- read data
- write data
- 2^{30} words
- 1…1100
- 0…1100
- 0…1000
- 0…0100
- 0…0000
- 32 bits
- byte address (big Endian)

ALU
- PC
- Add
- br offset
- Exec
- Decode
- Fetch
- PC = PC + 4
- 32
- 4
- 32
- 32
- 32
- 32
- 32
- 32
- 32
- 32
- 32
- 32
Review: Bits can represent anything!!

- Characters
  - 26 letters ⇒ 5 bits ($2^5 = 32$)
  - upper/lower case + punctuation ⇒ 7 bits (in 8) (“ASCII”)
  - standard code to cover all the world’s languages ⇒ 8,16,32 bits (“Unicode”)
    www.unicode.com

- Logical values
  - 0 ⇒ False, 1 ⇒ True

- colors

- locations / addresses / commands
Byte Addresses

- Since bytes (8 bits) are so useful, most ISAs support addressing individual bytes in memory.

- **Endianness**: refer to how bytes of a data word are ordered within memory.
  - *Jonathan Swift’s Gulliver’s Travels*: the Little-Endians broke their eggs on the little end of the egg and the Big-Endians broke their eggs on the big end.

- **Little Endian**: increasing numeric significance with increasing memory addresses
  - Intel 80x86, x86-64, DEC Vax, DEC Alpha, Z80, 8051

- **Big Endian**: on the contrary to Little Endian
  - IBM 360/370, Motorola 68k, SPARC (before v9)

- **Bi-Endian**: can switch between the two endianness
  - MIPS, Power PC, SPARC (v9), ARM (after v3), IA-64
Byte Addresses

- For example: to store 0x0A0B0C0D in memory:
  - **If Little Endian:**
    - Base Address+0: 0D
    - Base Address+1: 0C
    - Base Address+2: 0B
    - Base Address+3: 0A
  - **If Big Endian:**
    - Base Address+0: 0A
    - Base Address+1: 0B
    - Base Address+2: 0C
    - Base Address+3: 0D
Addressing Objects: Endianess and Alignment

- **Big Endian:** leftmost byte is the address of word
- **Little Endian:** rightmost byte is the address of word

Alignment restriction: requires that objects fall on address that is multiple of their size.
Big & Little Endian - Summary

The address used for a group of bytes is the *smallest* address of the four byte addresses.

- **Big Endian**: the “big end” goes into the smallest byte
- **Little Endian**: the “small end” goes into the smallest byte

The remaining three bytes are filled in sequence.

The MIPS processor chip can be set up in hardware to use *either* byte ordering. A computer system designer makes whatever choice best fits the rest of the components in the computer system.
Loading and Storing Bytes

- MIPS provides special instructions to move bytes
  
  \[ \text{lb} \quad \text{sb} \]
  
  \[ \begin{array}{c|c|c|c}
    \text{op} & \text{rs} & \text{rt} & \text{16 bit number} \\
  \end{array} \]

- What 8 bits get loaded and stored?
  
  - load byte places the byte from memory in the rightmost 8 bits of the destination register
    - what happens to the other bits in the register?
  
  - store byte takes the byte from the rightmost 8 bits of a register and writes it to the byte in memory
    - leaving the other bytes in the memory word unchanged

\[ \text{lb} \quad \text{sb} \]
Example of Loading and Storing Bytes

Given following code sequence and memory state what is the state of the memory after executing the code? {Assume bigEndian}

```
add $s3, $zero, $zero
lb $t0, 1($s3)
sb $t0, 6($s3)
```

What value is left in $t0?

$t0 = 0xFFFFFFFF90$

What word is changed in Memory and to what?

mem(4) = 0xFFFF90FF

What if the machine was littleEndian?

$t0 = 0x00000012$

mem(4) = 0xFF12FFFF
Loading and Storing Bytes

- What do with other 24 bits in the 32 bit register?
  - \( lb \): sign extends to fill upper 24 bits

\[
\begin{array}{cccccccccccc}
xxxx & xxxx & xxxx & xxxx & xxxx & xxxx & xxxx & xxxx & xzzz & zzzz \\
\end{array}
\]

- ...is copied to “sign-extend”

- Normally don’t want to sign extend chars

- MIPS instruction that doesn’t sign extend when loading bytes:
  - load byte _unsigned_: \( lbu \)
Loading and Storing Half Words

- MIPS also provides special instructions to move half words

```
lh $t0, 2($s3)  #load half word from memory
sh $t0, 6($s3)  #store half word to memory
```

- What 16 bits get loaded and stored?
  - load half word places the half word from memory in the rightmost 16 bits of the destination register
    - what happens to the other bits in the register?
  - store half word takes the half word from the rightmost 16 bits of the register and writes it to the half word in memory
    - leaving the other half word in the memory word unchanged
MIPS also provides instructions to shift words

- `sll` shift left logical
- `srl` shift right logical
Shift Operations

- Need operations to **pack** and **unpack** 8-bit characters into 32-bit words

- Shifts move all the bits in a word left or right

  \[
  \text{sll } \$t2, \$s0, 8 \quad \# \$t2 = \$s0 \ll 8 \text{ bits} \\
  \text{srl } \$t2, \$s0, 8 \quad \# \$t2 = \$s0 \gg 8 \text{ bits}
  \]

- Such shifts are called **logical** because they fill with zeros

  - Notice that a 5-bit shamt field is enough to shift a 32-bit value \(2^5 - 1\) or 31 bit positions
More Shift Operations

- An arithmetic shift (\texttt{sra}) maintains the arithmetic correctness of the shifted value (i.e., a number shifted right one bit should be $\frac{1}{2}$ of its original value; a number shifted left should be 2 times its original value)
  - \texttt{sra} uses the most significant bit (sign bit) as the bit shifted in
  - \texttt{sll} works for arithmetic left shifts for 2’s compl. (so there is no need for a \texttt{sla})

\[
sra \; \$t2, \; \$s0, \; 8 \quad \# \; \$t2 = \$s0 >> 8 \text{ bits}
\]

<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>0</td>
<td>16</td>
<td>10</td>
<td>8</td>
<td>0x03</td>
<td>R format</td>
</tr>
</tbody>
</table>
Compiling Another While Loop

Compile the assembly code for the C `while` loop where \( i \) is in \( \$s3 \), \( k \) is in \( \$s5 \), and the base address of the array \( \text{save} \) is in \( \$s6 \)

```assembly
while (save[i] == k)
    i += 1;

Loop:
    sll $t1, $s3, 2
    add $t1, $t1, $s6
    lw $t0, 0($t1)
    bne $t0, $s5, Exit
    addi $s3, $s3, 1
    j Loop

Exit: ...
```
Logical Operations

- There are a number of **bit-wise** logical operations in the MIPS ISA.

  - **and**  
    
    \[
    \text{and } \quad \text{or } \quad \text{nor} \quad \text{andi} \quad \text{ori} \\
    \]

    \[
    \begin{align*}
    \text{and} & \quad \text{or} & \quad \text{nor} & \quad \text{andi} & \quad \text{ori} \\
    \text{op} & = & \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

    \[
    \begin{align*}
    \text{and} & \quad \text{or} & \quad \text{nor} & \quad \text{andi} & \quad \text{ori} \\
    \text{op} & = & \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

  - **or**  
    
    \[
    \begin{align*}
    \text{or} & \quad \text{nor} & \quad \text{andi} & \quad \text{ori} \\
    \text{op} & = & \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

  - **nor**  
    
    \[
    \begin{align*}
    \text{nor} & \quad \text{andi} & \quad \text{ori} \\
    \text{op} & = & \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

  - **andi**  
    
    \[
    \begin{align*}
    \text{andi} & \quad \text{ori} \\
    \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

  - **ori**  
    
    \[
    \begin{align*}
    \text{ori} & \quad \text{andi} \\
    \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

  - **andi**  
    
    \[
    \begin{align*}
    \text{andi} & \quad \text{ori} \\
    \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

  - **ori**  
    
    \[
    \begin{align*}
    \text{ori} & \quad \text{andi} \\
    \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

  - **andi**  
    
    \[
    \begin{align*}
    \text{andi} & \quad \text{ori} \\
    \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

  - **ori**  
    
    \[
    \begin{align*}
    \text{ori} & \quad \text{andi} \\
    \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

  - **andi**  
    
    \[
    \begin{align*}
    \text{andi} & \quad \text{ori} \\
    \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]

  - **ori**  
    
    \[
    \begin{align*}
    \text{ori} & \quad \text{andi} \\
    \text{op} & = & \text{op} & = \\
    \text{rs} & = 9 & \quad \text{rt} & = 10 & \quad \text{rd} & = 8 \\
    \text{shamt} & = & \quad \text{funct} & = & \text{funct} & = \\
    \text{op} & = 0 & \quad \text{op} & = & \text{op} & = \\
    \text{funct} & = 0 \times 24 & \quad \text{funct} & = & \text{funct} & = 
    \end{align*}
    \]
Logic Operations

- Logic operations operate on individual bits of the operand.

\[
\begin{align*}
&t2 = 0\ldots0 0000 1101 0000 \\
&t1 = 0\ldots0 0011 1100 0000 \\
\text{and } &t0, t1, t2 \quad t0 = 0\ldots0 0000 1100 0000 \\
\text{or } &t0, t1, t2 \quad t0 = 0\ldots0 0011 1101 0000 \\
\text{nor } &t0, t1, t2 \quad t0 = 1\ldots1 1100 0010 1111
\end{align*}
\]

- How about “not”?
  - No “not” Instruction in MIPS! Why?
  - A NOR 0 = NOT (A OR 0) = NOT (A)
How About Larger Constants?

- We'd also like to be able to load a 32-bit constant into a register, for example: \(0xaaaaabbbb\)

- Must use two instructions, new "load upper immediate" instruction
  
  \[
  \text{lui } \$t0, \ 0xaaaa
  \]

- Then must get the lower order bits right, i.e.,
  
  \[
  \text{ori } \$t0, \ \$t0, \ 0xbbbb
  \]
<table>
<thead>
<tr>
<th>Category</th>
<th>Instr</th>
<th>OpC</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic (R &amp; I format)</td>
<td>add</td>
<td>0 &amp; 20</td>
<td>add $s1, $s2, $s3</td>
<td>$s1 = $s2 + $s3</td>
</tr>
<tr>
<td></td>
<td>subtract</td>
<td>0 &amp; 22</td>
<td>sub $s1, $s2, $s3</td>
<td>$s1 = $s2 - $s3</td>
</tr>
<tr>
<td></td>
<td>add immediate</td>
<td>8</td>
<td>addi $s1, $s2, 4</td>
<td>$s1 = $s2 + 4</td>
</tr>
<tr>
<td></td>
<td>shift left logical</td>
<td>0 &amp; 00</td>
<td>sll $s1, $s2, 4</td>
<td>$s1 = $s2 &lt;&lt; 4</td>
</tr>
<tr>
<td></td>
<td>shift right logical</td>
<td>0 &amp; 02</td>
<td>srl $s1, $s2, 4</td>
<td>$s1 = $s2 &gt;&gt; 4 (fill with zeros)</td>
</tr>
<tr>
<td></td>
<td>shift right arithmetic</td>
<td>0 &amp; 03</td>
<td>sra $s1, $s2, 4</td>
<td>$s1 = $s2 &gt;&gt; 4 (fill with sign bit)</td>
</tr>
<tr>
<td></td>
<td>and</td>
<td>0 &amp; 24</td>
<td>and $s1, $s2, $s3</td>
<td>$s1 = $s2 &amp; $s3</td>
</tr>
<tr>
<td></td>
<td>or</td>
<td>0 &amp; 25</td>
<td>or $s1, $s2, $s3</td>
<td>$s1 = $s2</td>
</tr>
<tr>
<td></td>
<td>nor</td>
<td>0 &amp; 27</td>
<td>nor $s1, $s2, $s3</td>
<td>$s1 = not ($s2</td>
</tr>
<tr>
<td></td>
<td>and immediate</td>
<td>c</td>
<td>and $s1, $s2, ff00</td>
<td>$s1 = $s2 &amp; 0xff00</td>
</tr>
<tr>
<td></td>
<td>or immediate</td>
<td>d</td>
<td>or $s1, $s2, ff00</td>
<td>$s1 = $s2</td>
</tr>
<tr>
<td></td>
<td>load upper immediate</td>
<td>f</td>
<td>lui $s1, 0xffff</td>
<td>$s1 = 0xffff0000</td>
</tr>
</tbody>
</table>
## Review: MIPS Instructions, so far

<table>
<thead>
<tr>
<th>Category</th>
<th>Instr</th>
<th>OpC</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data transfer</strong></td>
<td><strong>load word</strong></td>
<td>23</td>
<td>lw $s1, 100($s2)</td>
<td>$s1 = Memory($s2+100)</td>
</tr>
<tr>
<td><strong>store word</strong></td>
<td></td>
<td>2b</td>
<td>sw $s1, 100($s2)</td>
<td>Memory($s2+100) = $s1</td>
</tr>
<tr>
<td><strong>load byte</strong></td>
<td></td>
<td>20</td>
<td>lb $s1, 101($s2)</td>
<td>$s1 = Memory($s2+101)</td>
</tr>
<tr>
<td><strong>store byte</strong></td>
<td></td>
<td>28</td>
<td>sb $s1, 101($s2)</td>
<td>Memory($s2+101) = $s1</td>
</tr>
<tr>
<td><strong>load half</strong></td>
<td></td>
<td>21</td>
<td>lh $s1, 101($s2)</td>
<td>$s1 = Memory($s2+102)</td>
</tr>
<tr>
<td><strong>store half</strong></td>
<td></td>
<td>29</td>
<td>sh $s1, 101($s2)</td>
<td>Memory($s2+102) = $s1</td>
</tr>
<tr>
<td><strong>Cond. branch</strong></td>
<td><strong>br on equal</strong></td>
<td>4</td>
<td>beq $s1, $s2, L</td>
<td>if ($s1===$s2) go to L</td>
</tr>
<tr>
<td><strong>br on not equal</strong></td>
<td></td>
<td>5</td>
<td>bne $s1, $s2, L</td>
<td>if ($s1 !=$s2) go to L</td>
</tr>
<tr>
<td><strong>set on less than</strong></td>
<td><strong>slti</strong></td>
<td>a</td>
<td>$s1, $s2, 100</td>
<td>if ($s2&lt;100) $s1=1; else $s1=0</td>
</tr>
<tr>
<td><strong>set on less than</strong></td>
<td><strong>slt</strong></td>
<td>0 &amp; 2a</td>
<td>$s1, $s2, $s3</td>
<td>if ($s2&lt;$s3) $s1=1; else $s1=0</td>
</tr>
<tr>
<td><strong>Uncond. jump</strong></td>
<td><strong>jump</strong></td>
<td>2</td>
<td>j 2500</td>
<td>go to 10000</td>
</tr>
<tr>
<td><strong>jump register</strong></td>
<td></td>
<td>0 &amp; 08</td>
<td>jr $t1</td>
<td>go to $t1</td>
</tr>
</tbody>
</table>
Review: MIPS R3000 ISA

- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special

- 3 Instruction Formats: all 32 bits wide

<table>
<thead>
<tr>
<th>6 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>5 bits</th>
<th>6 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>funct</td>
</tr>
<tr>
<td>R format</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6 bits</th>
<th>5 bits</th>
<th>5 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
<td>rs</td>
<td>rt</td>
</tr>
<tr>
<td>I format</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
</tr>
<tr>
<td>J format</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
</tr>
<tr>
<td>26 bit jump target</td>
</tr>
</tbody>
</table>

Registers:
- R0 - R31
- PC
- HI
- LO