S’more Instructions*

A continuation of discussion of the MIPS ISA, instructions and instruction formats

* Cover Graham cracker with a marshmallow and chocolate
Fact Sheet on Registers/Instructions

- Machines typically 16 or 32 registers
- More registers => larger instruction format
- More registers => possibly slower clock speed
- Register 0 is the constant 0 in MIPS ISA
- Extending the instruction set ...

sub $8, $0, $8  # negate $8
add $9, $0, $5  # move $5 to $9
add $5, $0, $0  # set $8 to zero

Every computer (ISA) has a Principles of Operations manual (Prince of Ops); see appendix A.10 for MIPS
R-types Are For Basic Operations

- Different operations are specified by different codings of the op field and funct field

\[(\text{op:funct}) \quad \text{rd}, \ \text{rs}, \ \text{rt}\]

<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>add</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>sub</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>34</td>
</tr>
<tr>
<td>and</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>36</td>
</tr>
<tr>
<td>mult</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>slt</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>42</td>
</tr>
</tbody>
</table>

Example: and $7, $3, R4  # Bitwise and

\[
\begin{array}{cccccccc}
0 & 3 & 4 & 7 & 0 & 36 \\
0000000000110010000111000000100100
\end{array}
\]
R-types Continued

- Shifting instructions also have 3 operands
- **shamt** is used, but **rs** is unused

```
(op:funct)  rd,  rt,  shamt
<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>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

sll 0 0 # shift left
srl 0 0 2 # shift right
sra 0 0 3 # arith right

Example:  srl $7, $3, 12  # Logical right shift
```

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>3</th>
<th>7</th>
<th>12</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000000000001110011101100000010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decoding

- Decoding the instruction (second step in the fetch execute cycle):

```
add   $8,   $9   $10
01001 01010 01000 00000 100000
0  9  10  8  0  32
```

What Instruction is all 0’s?

```
???,  ??,  ??,  ??
00000000000000000000000000000000
```
I-type, Immediate Instructions

- When an operand is a small constant it can be stored in the instruction, saving fetching it from memory and using a general register.

\[
\text{addi } \$8, \$9 \ 100 \ # \ Add \ immediate
\]

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
<td>8</td>
<td>100</td>
</tr>
</tbody>
</table>

\[
00100001001001000000000000100100
\]

6 5 5 16

R-type instructions
Immediate Instructions

Immediate variations exist for most instructions

<table>
<thead>
<tr>
<th>op</th>
<th>rt, rs, immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>addi</td>
<td>add immediate</td>
</tr>
<tr>
<td>andi</td>
<td>and immediate</td>
</tr>
<tr>
<td>ori</td>
<td>or immediate</td>
</tr>
<tr>
<td>slti</td>
<td>set less than immediate</td>
</tr>
</tbody>
</table>

**Why is there no **subi instruction**?**

Example:  andi $7, $3, 12  # Logical and

\[
\begin{array}{cccc}
00000000000000000000000000001100 \\
\end{array}
\]

and

register 3 operand
lui, lui

The 16bit limit on immediate data is not severe

**load upper immediate (lui)** moves its immediate data to the left-most 16-bits; it zeroes the right-most 16-bits; **rs** is unused

```
lui $8, 100  # Load upper immediate
```

### I-type instructions

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0</td>
<td>8</td>
<td>100</td>
</tr>
</tbody>
</table>

```
00111100000010000000000001100100
```

Register 8