Multiply/Divide

`mult rs, rt`  # start signed multiply
`multu rs, rt`  # start unsigned multiply
    # lo, bottom 32 bits
`div rs, rt`  # start signed divide
`divu rs, rt`  # start unsigned divide
    # lo = quotient, hi = remainder
`mfhi rd`  # rd = hi
`mflo rd`  # rd = lo

`mthi rs`  # hi = rs
`mtlo rs`  # lo = rs

Special registers used to handle 64-bit result.
Arithmetic Instructions

C Code

```c
int a, b, c, d;
a = b - (2*c + 7);
c = (a < 0) ? 1 : 0;
```

Assembly

```
# a in reg 16, b in reg 17, c in reg 18
addu $8, $18, $18  # temp = 2*c
addiu $8, $8, 7      # temp = temp + 7
subu $16, $17, $8  # a = b - temp = b - (2*c+7)
slt  $18, $16, $0   # c = (a < 0)
```
Logical Operations

**AND:** both bits must be 1 (C operator \&)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**OR:** either bit is 1 (C operator |)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Logical Operations

**eXclusive OR:** bits must be different (C operator ^)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**NOR:** OR followed by NOT

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Instruction</td>
<td>Operands</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>rd, rs, rt</td>
<td># rd = rs &amp; rt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>andi</td>
<td>rt, rs, imm</td>
<td># rt = rs &amp; imm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nor</td>
<td>rd, rs, rt</td>
<td># rd = ~(rs</td>
<td>rt)</td>
<td></td>
</tr>
<tr>
<td>lui</td>
<td>rt, imm</td>
<td># rt = imm &lt;&lt; 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>rd, rs, rt</td>
<td># rd = rs</td>
<td>rt</td>
<td></td>
</tr>
<tr>
<td>ori</td>
<td>rt, rs, imm</td>
<td># rt = rs</td>
<td>imm</td>
<td></td>
</tr>
<tr>
<td>sll</td>
<td>rd, rt, shamt</td>
<td># rd = rt &lt;&lt; shamt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sllv</td>
<td>rd, rt, rs</td>
<td># rd = rt &lt;&lt; (rs&amp;0x1f)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sra</td>
<td>rd, rt, shamt</td>
<td># rd = rt &gt;&gt;s shamt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>srav</td>
<td>rd, rt, rs</td>
<td># rd = rt &gt;&gt;s (rs&amp;0x1f)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>srl</td>
<td>rd, rt, shamt</td>
<td># rd = rt &gt;&gt; shamt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>srlv</td>
<td>rd, rt, rs</td>
<td># rd = rt &gt;&gt; (rs&amp;0x1f)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xor</td>
<td>rd, rs, rt</td>
<td># rd = rs ^ rt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xori</td>
<td>rt, rs, imm</td>
<td># rt = rs ^ imm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How Do I ...

**Load a 16-bit constant?**
```
addiu $8, $0, value
```

**Load a 32-bit constant?**
```
lui $8, (value >> 16)
ori $8, $8, (value & 0xffff)
```

**Why no subiu?**
```
addiu $8, $9, (-value)
```

**Move from one register to another?**
```
or $8, $9, $0
```
How Do I ...

Negate a register?
    subu $8, $0, $8

Complement a register?
    nor $8, $8, $0

Check for equality?
    xor $8, $16, $17
    sltiu $8, $8, 1
Calculate absolute value of a register?

sra $9,$8,31
xor $8,$9,$8
andi $9,$9,1
addu $8,$9,$8
Control Flow

Instructions that modify the pc. They specify:

- **New pc value**
  - pc-relative address
  - absolute address

- **When is pc modified?**
  - unconditional, equality between registers, etc.

- **Save current pc?**
  - Used for function calls
  - More later
Control Flow: Branches

Branch instructions:

- **beq**  \(\text{rs}, \text{rt}, \text{imm}\)  # if (\(\text{rs}==\text{rt}\)) goto L
- **bgez**  \(\text{rs}, \text{imm}\)  # if (\(\text{rs}\geq 0\)) goto L
- **bgezal**  \(\text{rs}, \text{imm}\)  # link; if (\(\text{rs}\geq 0\)) goto L
- **bgtz**  \(\text{rs}, \text{imm}\)  # if (\(\text{rs}>0\)) goto L
- **blez**  \(\text{rs}, \text{imm}\)  # if (\(\text{rs}\leq 0\)) goto L
- **bltz**  \(\text{rs}, \text{imm}\)  # if (\(\text{rs}<0\)) goto L
- **bltzal**  \(\text{rs}, \text{imm}\)  # link; if (\(\text{rs}<0\)) goto L
- **bne**  \(\text{rs}, \text{rt}, \text{imm}\)  # if (\(\text{rs}\neq \text{rt}\)) goto L

**pc-calculation:**  \(L = pc + 4 + (\text{s_ext(imm)} \ll 2)\)

**link:**  \$31=pc+8\
Control Flow: Jumps

Jump instructions:
- j   tgt   # goto target
- jal tgt   # link; goto target
- jalr rs, rd # rd=pc+8; goto rs
- jr   rs   # goto rs

pc-calculation:

\[
\text{target} = ((pc + 4) \& 0xf0000000) | (tgt << 2)
\]

<table>
<thead>
<tr>
<th>op</th>
<th>tgt</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>26 bits</td>
</tr>
</tbody>
</table>

Jumps: absolute; Branches: pc-relative
How Do I...

Branch if register is zero?
  `beq $9,$0, L`

Unconditional branch?
  `beq $0, $0, L`

Branch if one register is smaller than another?
  `slt $8, $16,$17`
  `bne $8, $0, L`

Jump to a 32-bit address?
How Do I...

Jump to an offset > 16 bits?

```
bltzal $0, next
next:  li $8, adjoffset
      addu $8,$8,$31
      jr $8
```

Wait, what is `li`?

A *pseudo-operation*. Gets translated to either `addiu`, or `lui` and `ori`
## Summary: Instruction Formats

### R-format:

<table>
<thead>
<tr>
<th></th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

### I-format:

<table>
<thead>
<tr>
<th></th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>imm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

### J-format:

<table>
<thead>
<tr>
<th></th>
<th>op</th>
<th>tgt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>6</td>
<td>26</td>
</tr>
</tbody>
</table>
Translating Conditional Branches

C code

```c
int a, b;
if (a >= b) { a = 0; }
b += a;
```

Assembly

```assembly
# assume $16=a, $17=b

slt $8,$16,$17    # set if a < b
bne $8,$0,$skip  # if 1, then branch
li $16,0         # set a to zero
$skip: addu $17,$17,$16  # b += a
```
Translating Loops

C code

```c
int i, j;
for (i=0; i < 10; i++)
    j += i;
    j++;
```

First step toward assembly: remove structure (yikes!)
Translating Loops

**Modified C code**

```c
i = 0;
loop: if (!(i < 10)) goto finished;
j += i;
i++;
goto loop;
finished: j++;
```
Translating Loops

Minor Optimization

\[
i = 0; \\
\text{loop: } j += i; \\
\quad i++; \\
\quad \text{if (i < 10) goto loop;} \\
\quad j++;\]

Assembly \((i \text{ in } 16, j \text{ in } 17)\)

\[
\text{li } 16, 0 \quad \# \text{ set } i = 0 \\
\text{loop: } \text{addu } 17, 17, 16 \quad \# j += i \\
\quad \text{addiu } 16, 16, 1 \quad \# i++ \\
\quad \text{slt } 8, 16, 10 \quad \# \text{ compare } i < 10 \\
\quad \text{bne } 8, 0, \text{loop} \quad \# \text{ and branch} \\
\quad \text{addiu } 17, 17, 1 \quad \# j++
\]
### Characters and Strings

Characters are stored as bytes (8 bits, ASCII)

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>SPC</td>
<td>48</td>
<td>0</td>
<td>64</td>
<td>@</td>
<td>80</td>
<td>P</td>
<td>96</td>
<td>'</td>
<td>112</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>!</td>
<td>49</td>
<td>1</td>
<td>65</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
<td>a</td>
<td>113</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>&quot;</td>
<td>50</td>
<td>2</td>
<td>66</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
<td>b</td>
<td>114</td>
<td>r</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>#</td>
<td>51</td>
<td>3</td>
<td>67</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
<td>c</td>
<td>115</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>47</td>
<td>/</td>
<td>63</td>
<td>?</td>
<td>79</td>
<td>O</td>
<td>95</td>
<td>_</td>
<td>111</td>
<td>o</td>
<td>127</td>
<td>DEL</td>
<td></td>
</tr>
</tbody>
</table>
**Characters And Strings**

String: variable-length character array, terminated by NULL (byte 0)

<table>
<thead>
<tr>
<th>C</th>
<th>o</th>
<th>r</th>
<th>n</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>111</td>
<td>114</td>
<td>110</td>
<td>101</td>
<td>108</td>
<td>108</td>
<td>0</td>
</tr>
<tr>
<td>0x43</td>
<td>0x6f</td>
<td>0x72</td>
<td>0x6e</td>
<td>0x65</td>
<td>0x6c</td>
<td>0x6c</td>
<td>0x00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>addr</th>
<th>addr+1</th>
<th>addr+2</th>
<th>addr+3</th>
<th>addr+4</th>
<th>addr+5</th>
<th>addr+6</th>
<th>addr+7</th>
</tr>
</thead>
</table>

String Search

Problem: find the number of spaces in a string.

C code

/* s contains the address of the string */
count = 0;
while (*s) {
    if (*s++ == ' ') count++;
}
/* count contains the number of spaces */

First step: remove structure.
String Search

Modified C code

```c
count = 0;
start: if (!(*s)) goto done;
    if (!(*s == ' ')) goto skipinc;
    count++;
skipinc: s++;
goto start;
done: ...
```
String Search

Assembly count in $16, s in $17

```assembly
li $16,0 # count = 0
$start: lbu $8, 0($17) # temp = *s
beq $8,$0,$done # if *s == 0 goto done
li $9,32 # temp2 = ‘ ’
bne $8,$9,$skipinc # if *s != ‘ ’ # goto skipinc

addiu $16,$16,1 # count++

$skipinc: addiu $17,$17,1 # s++
beq $0,$0,$start # goto start
$done: ...
```