lecture 8

MIPS assembly language 1

- what is an assembly language?
- addressing and Memory
- instruction formats (R, I, J)

February 3, 2016
http://www.asciitable.com/

**ASCII: 8 bit (one byte) code**

In fact, it only uses 7 of the 8 bits.

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
<td>NUL</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td> </td>
<td>96</td>
<td>60</td>
<td>140</td>
<td>&gt;</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>SOH</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td>&lt;A</td>
<td>97</td>
<td>61</td>
<td>141</td>
<td>&lt;A</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>STX</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td>&lt;B</td>
<td>98</td>
<td>62</td>
<td>142</td>
<td>&lt;B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>ETX</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td>&lt;C</td>
<td>99</td>
<td>63</td>
<td>143</td>
<td>&lt;C</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>EOT</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td>&lt;D</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td>&lt;D</td>
<td>100</td>
<td>64</td>
<td>144</td>
<td>&lt;D</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td>&lt;E</td>
<td>101</td>
<td>65</td>
<td>145</td>
<td>&lt;E</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td>&lt;F</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td>&lt;F</td>
<td>102</td>
<td>66</td>
<td>146</td>
<td>&lt;F</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td>&lt;G</td>
<td>71</td>
<td>47</td>
<td>107</td>
<td>&lt;G</td>
<td>103</td>
<td>67</td>
<td>147</td>
<td>&lt;G</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td>(</td>
<td>104</td>
<td>68</td>
<td>150</td>
<td>(</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>TAB</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td>)</td>
<td>105</td>
<td>69</td>
<td>151</td>
<td>)</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>012</td>
<td>LF</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td>*</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td>*</td>
<td>106</td>
<td>6A</td>
<td>152</td>
<td>*</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>013</td>
<td>VT</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td>+</td>
<td>107</td>
<td>6B</td>
<td>153</td>
<td>+</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>014</td>
<td>FF</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td>#</td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td>#</td>
<td>108</td>
<td>6C</td>
<td>154</td>
<td>#</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>015</td>
<td>CR</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td>-</td>
<td>109</td>
<td>6D</td>
<td>155</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>016</td>
<td>SO</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td>.</td>
<td>110</td>
<td>6E</td>
<td>156</td>
<td>.</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>017</td>
<td>SI</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td>/</td>
<td>111</td>
<td>6F</td>
<td>157</td>
<td>/</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>020</td>
<td>DLE</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>120</td>
<td>0</td>
<td>112</td>
<td>70</td>
<td>160</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>021</td>
<td>DC1</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>121</td>
<td>1</td>
<td>113</td>
<td>71</td>
<td>161</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>022</td>
<td>DC2</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>122</td>
<td>2</td>
<td>114</td>
<td>72</td>
<td>162</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>023</td>
<td>DC3</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>123</td>
<td>3</td>
<td>115</td>
<td>73</td>
<td>163</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>024</td>
<td>DC4</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>124</td>
<td>4</td>
<td>116</td>
<td>74</td>
<td>164</td>
<td>4</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>025</td>
<td>NAK</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>125</td>
<td>5</td>
<td>117</td>
<td>75</td>
<td>165</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>026</td>
<td>SYN</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>126</td>
<td>6</td>
<td>118</td>
<td>76</td>
<td>166</td>
<td>6</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>027</td>
<td>ETB</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>127</td>
<td>7</td>
<td>119</td>
<td>77</td>
<td>167</td>
<td>7</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>030</td>
<td>CAN</td>
<td>56</td>
<td>38</td>
<td>070</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>130</td>
<td>8</td>
<td>120</td>
<td>78</td>
<td>170</td>
<td>8</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>031</td>
<td>EM</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>131</td>
<td>9</td>
<td>121</td>
<td>79</td>
<td>171</td>
<td>9</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>032</td>
<td>SUB</td>
<td>58</td>
<td>3A</td>
<td>072</td>
<td>#</td>
<td>90</td>
<td>5A</td>
<td>132</td>
<td>#</td>
<td>122</td>
<td>7A</td>
<td>172</td>
<td>#</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>033</td>
<td>ESC</td>
<td>59</td>
<td>3B</td>
<td>073</td>
<td>$</td>
<td>91</td>
<td>5B</td>
<td>133</td>
<td>$</td>
<td>123</td>
<td>7B</td>
<td>173</td>
<td>$</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>034</td>
<td>FS</td>
<td>60</td>
<td>3C</td>
<td>074</td>
<td>%</td>
<td>92</td>
<td>5C</td>
<td>134</td>
<td>%</td>
<td>124</td>
<td>7C</td>
<td>174</td>
<td>%</td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>035</td>
<td>GS</td>
<td>61</td>
<td>3D</td>
<td>075</td>
<td>&amp;</td>
<td>93</td>
<td>5D</td>
<td>135</td>
<td>&amp;</td>
<td>125</td>
<td>7D</td>
<td>175</td>
<td>&amp;</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>036</td>
<td>RS</td>
<td>62</td>
<td>3E</td>
<td>076</td>
<td>'</td>
<td>94</td>
<td>5E</td>
<td>136</td>
<td>'</td>
<td>126</td>
<td>7E</td>
<td>176</td>
<td>'</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>037</td>
<td>US</td>
<td>63</td>
<td>3F</td>
<td>077</td>
<td>(</td>
<td>95</td>
<td>5F</td>
<td>137</td>
<td>(</td>
<td>127</td>
<td>7F</td>
<td>177</td>
<td>(</td>
</tr>
</tbody>
</table>
Unicode is a 16 bit code which includes characters for most written natural languages. [Correction: It is more complicated than that.] https://en.wikipedia.org/wiki/Unicode
- written and read by humans
- not executable (ASCII)
- not machine specific

- executable
- machine specific

(...although what we mean by "machine" here is subtle.  
e.g. Java Virtual Machine.)
human writeable & readable machine code (in ASCII)

C, Fortran
Java

assembly language

machine code

compiler

assembler
"Reduced Instruction Set Computer" (RISC)

MIPS is not so different from today's ARM processors e.g. in your cell phone or tablet.
MARS simulator

http://courses.missouristate.edu/KenVollmar/MARS/index.htm

Edit mode

```assembly
25  # $s4 holds a character
26  # $s5 holds the address in the current array element (a pointer to string)
27  
28  # initialize variables
29  add  $s0, $zero, $zero
30  la   $a1, strings
31  add  $s3, $s1, $zero
32  la   $s5, stringRefArray
33  
34  # print prompt to enter maximum length of a string
35  la   $a0, enterMaxStringPrompt
36  li   $v0, 4
37  syscall
38  
39  # read maximum length of string buffer, read value goes into $v0
40  # la   $a0, enterMaxStringPrompt
41  li   $v0, 5
42  syscall
43  
44  # If user specifies that string is Nmax characters, then we need a buffer for
45  # reading the string that is at least Nmax+1 bytes. The reason is that the
```

```
MARS simulator

Execute mode
Addressing in MIPS

1 word = 4 bytes
= 32 bits
Each MIPS instruction is 32 bits.
Examples of MIPS instructions

```
add $16, $17, $18
```

```
# register 16 is assigned the sum of registers 17 and 18
```

Comment symbol
add $16, $17, $18
recall lecture 6
Arithmetic and logic instructions

add $16, $17, $18
sub $19, $15, $19
and $17, $17, $16
or $16, $17, $18
nor $16, $17, $20
Memory transfer instructions

# load word from Memory

lw $16, 40($17)

| copy word from base Memory address (in $17) + offset (40) to $16 | offset | Memory address (base) |
li $16, 40 ($17)
Suppose the values of x and y are assigned to registers (say $18 and $20) but the value of z is stored in Memory. Then we need to bring z into a register (say $16) in order to do the addition.

```
# x = y + z

# registers in Memory

lw $16, 40($17)
add $18, $20, $16
```
copy word from $16 \text{ to }$ Memory address specified by base (in $17$) + offset (40)
SW $16, 40 ($17)

MIPS registers

40

ALU

address

MIPS Memory

$16

$17

16

17

$2^{32}$ bytes

$2^{-1}$
If you want to store the result of the sum in a Memory address (you might need it later) then you must use a register.

In MIPS, you cannot take the result of a summation from the ALU and put it directly in Memory.

\[ z = x + y \]

Memory

\[ \text{in registers} \]

\[
\begin{align*}
\text{add} & \quad 8, \quad 16, \quad 20 \\
\text{sw} & \quad 8, \quad 40(17)
\end{align*}
\]
How does the computer keep track of which instruction is currently executing?
The Program Counter (PC) register specifies the Memory address of the instruction that is currently being executed.

The default is that the computer advances to the next instruction. But there are also branches allowed (next slide)....
Branching Instructions

if \ (a \neq b) \\
\quad f = g + h \\
else \quad f = g - h

How is conditional branching done in MIPS?
e.g. "branch equals" (beq)

\[
\text{if } (a \neq b) \\
\text{\quad } f = g + h
\]

\[
\text{beq} \\
\text{add} \\
\text{Exit 1 :} \quad \ldots \ldots
g 17, g 18, \text{ Exit 1} \\
\text{g 19, g 20, g 21}
"jump" (j)

\[ \text{if } (a \neq b) \]
\[ f = g + h \]
\[ \text{else } f = g - h \]

Exit 1:
$17, 18, \text{ Exit 1}$

$19, 20, 21$

Exit 2:

$19, 20, 21$
human writeable & readable machine code (in ASCII)
MIPS instructions (machine code)

- always 32 bits (one word)

   [Binary code representation with some unspecified bits]

   31 30 29 etc 32 10

- relatively few

  (RISC: "reduced instruction set computer")
MIPS instruction formats

R

<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>

I

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

J

<table>
<thead>
<tr>
<th>op</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>26</td>
</tr>
</tbody>
</table>
R format instructions

<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>

- **Opcode**: configurable
- **Source**: configurable
- **Dest**: configurable
- **Shift amount**: configurable
- **Function**: configurable

(Other formats)
add $16, $17, $18

<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>

All R-format instructions have 000000 opcode.
**Sub $16, $17, $18**

<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>
I format instructions ("immediate")

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

0100111 10001 10000 00000000000101000

signed offset
from address in $17

lw  $16, 40($17)

rt plays the role of a "destination" register here
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

```
011 011 10001 10000 0000000000 01 0 1000
```

```
s  w  $16, 40($17)
```
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

```
000100 10010 10001
```

offset

(number of words from current instruction)

beg, $18, $17, Exit!
J format instructions ("jump")

You might write the following instruction in a MIPS program. The assembler then will calculate what the offset is from the present instruction to the instruction that you have labelled Exit2. Note that there are now 26 bits of offset, which allows bigger jumps than the conditional branches.
Announcements

A1

- get started by learning basics of Logisim
  (construct simple circuits shown in class, e.g. left shift register)

- specification with START and DONE is the last thing you should be concerned with

Quiz 2 is Monday (lectures 3-6)