Warm Up Problem

Convert the number -15 into a single byte two’s complement notation. What is the corresponding interpretation of this binary encoding as an unsigned integer?

Note: In this course you are expected to use Linux (Windows is not acceptable). See the CS 241 common webpage for more information.
CS 241 Lecture 2
Characters and Machine Language
With thanks to Brad Lushman, Troy Vasiga and Kevin Lanctot
Bytes as Characters

- ASCII (American Standard Code for Information Interchange) uses 7 bits to represent characters (see next table)
- Extended ASCII uses the 8th bit (but there are lots of compatibility issues here)
- Note that 'a' is different than 0xa (the former is the decimal number 97 in ASCII and the latter is just the number 10 in decimal).
- Unicode extends the above to many different types of character sets.
- We will use ASCII throughout.
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>Null</td>
<td>32</td>
<td>20</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>\</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Start of heading</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>Start of text</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>End of text</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>End of transmit</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>Enquiry</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>Acknowledge</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>Audible bell</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>Backspace</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>Horizontal tab</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>Line feed</td>
<td>42</td>
<td>2A</td>
<td>*</td>
<td>74</td>
<td>4A</td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>Vertical tab</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>Form feed</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>Carriage return</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>Shift out</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>Shift in</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>Data link escape</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>Device control 1</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>Device control 2</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>Device control 3</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>Device control 4</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>Neg. acknowledge</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>Synchronous idle</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>End trans. block</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>Cancel</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>End of medium</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>Substitution</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>Escape</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>(</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>File separator</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>Group separator</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>)</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>Record separator</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>Unit separator</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>-</td>
<td>127</td>
<td>7F</td>
<td>□</td>
</tr>
</tbody>
</table>
Highlights

• Characters 0-31 are control characters
• Characters 48-57 are the numbers 0 to 9
• Characters 65-90 are the letters A to Z
• Characters 97-122 are the letters a to z
• Note that ’A’ and ’a’ are 32 letters away
Bit-Wise Operators

Suppose we have unsigned char a=5, b=3;. Note that this is valid as C will interpret 5 as the character corresponding to 00000101 and similar for 3 with the bit string 00000011.

• Bitwise not ∼, for example c = ∼ a; gives c = 11111010
• Bitwise and &, for example c = a&b; gives c = 00000001
• Bitwise or |, for example c = a|b; gives c = 00000111
• Bitwise exclusive or ^, for example c = a ^ b; gives c = 00000110
• Bitwise shift right or left >> and <<, for example c = a >> 2; gives c = 00000001 and c = a << 3; gives c = 00101000.
• (High level aside: bitshift ambiguity with signed characters - arithmetic left and right shift.)
• These can even be combined with the assignment operator!
Question

With all of these different ways of interpreting a byte, how do we know which one it is by looking at it?
With all of these different ways of interpreting a byte, how do we know which one it is by looking at it?

**We can’t!** We need to remember how the byte was intended to be interpreted when we store the byte in memory!
Computer Programs

What is a Computer Program?
Computer Programs

What is a Computer Program?

- Programs operate on data.
- Program are data. This is a von Neumann architecture; programs live in the same memory space as the data they operate on.
- Programs can manipulate other programs (OSes, viruses, etc.)

Recall: We cannot distinguish without extra information whether or not some data represents instructions and which does not.
Instructions

What does an instruction look like?
Instructions

What does an instruction look like?

- Many different processor specific machine languages exist.
- We will use 32-bit MIPS. (There is a 64 bit version but the underlying theory is the same)
- MIPS Machine follows on next two slides
CPU (Central Processing Unit) - brain of computer

- Registers: $0, ..., $31, hi, lo
- Control Unit: Decodes instrs and dispatches
  - PC (Program Counter)
  - IR (Instruction Register)
- Memory: MDR (Memory Data Register) and MAR (Memory Address Register)
- ALU (Arithmetic Logic Unit) - does arithmetic
CPU with Memory

- **Bus**: short for Latin *omnibus*. Data travels along the bus.

- **Memory**: many kinds. From fastest to slowest:
  - CPU/registers (fastest)
  - L1 cache
  - L2 cache
  - RAM
  - disk
  - network memory [outside your local machine] (slowest)

---

http://www2.cs.siu.edu/~cs401/Textbook/ch2.pdf

“Computer Architecture”, Zargham, Prentice Hall, Ch. 2
More on Registers

• Registers are very fast memory
• MIPS has 32 registers for general use.
• Also the registers $hi$ and $lo$ (for multiplication and division)
• Some of these registers are special:
  • $0$ is always $0$
  • $31$ is for return addresses
  • $30$ is our stack pointer
  • $29$ is our frame pointer

High Brow Comment: In most MIPS standards the roles of $29$ and $30$ above are reversed.
MIPS Instructions

MIPS Reference Sheet

Basic Instruction Formats

<table>
<thead>
<tr>
<th>Register</th>
<th>0000 00ss ssst tttt dddd d000 00ff ffff</th>
<th>R</th>
<th>s, t, d are interpreted as unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>oooo ooss ssst tttt iiiii iiiii iiiii iiiii</td>
<td>I</td>
<td>i is interpreted as two’s complement</td>
</tr>
</tbody>
</table>

Instructions

| Word                          | .word i | iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii iiiii
How MIPS Works

• Recall: Code is just data - it is stored in RAM.
• MIPS takes instructions from RAM and attempts to execute them.
• Recall Problem: MIPS doesn’t know what in memory is an instruction \textit{a priori}
• Solution:
How MIPS Works

• Recall: Code is just data - it is stored in RAM.
• MIPS takes instructions from RAM and attempts to execute them.
• Recall Problem: MIPS doesn’t know what in memory is an instruction \emph{a priori}
• Solution: Convention is to set memory address 0 in RAM to be an instruction.
How MIPS Works

• Recall: Code is just data - it is stored in RAM.
• MIPS takes instructions from RAM and attempts to execute them.
• Recall Problem: MIPS doesn’t know what in memory is an instruction *a priori*
• Solution: Convention is to set memory address 0 in RAM to be an instruction.
• Problem: How does it know what to do next?
• Solution:
How MIPS Works

- Recall: Code is just data - it is stored in RAM.
- MIPS takes instructions from RAM and attempts to execute them.
- Recall Problem: MIPS doesn’t know what in memory is an instruction \textit{a priori}.
- Solution: Convention is to set memory address 0 in RAM to be an instruction.
- Problem: How does it know what to do next?
- Solution: Have a special register called the 	extbf{Program Counter} (or PC for short) to tell us what instruction to do next.
- Problem: How do we put our program into RAM?
- Solution:
How MIPS Works

• Recall: Code is just data - it is stored in RAM.
• MIPS takes instructions from RAM and attempts to execute them.
• Recall Problem: MIPS doesn’t know what in memory is an instruction \textit{a priori}
• Solution: Convention is to set memory address 0 in RAM to be an instruction.
• Problem: How does it know what to do next?
• Solution: Have a special register called the \textbf{Program Counter} (or PC for short) to tell us what instruction to do next.
• Problem: How do we put our program into RAM?
• Solution: A program called a \textit{loader} puts our program into memory and sets the PC to be the first address.
Fetch-Execute Cycle

Algorithm 1 Fetch-Execute Cycle

1: PC = 0
2: while true do
3: \[ \text{IR} = \text{MEM}[\text{PC}] \]
4: \[ \text{PC} \ += 4 \]
5: Decode and execute instruction in IR
6: end while

Eventually an instruction will break out of the loop. This is basically the only program a machine really runs.
Write a program in MIPS that takes in the values of registers $8$ and $9$ and stores the result in register $3$. 

```
add $d, $s, $t
```

Adds the result of registers $s$ and $t$ and store these in register $d$. 

Important! The order of $d$, $s$ and $t$ are shifted in the encoding!

Solution:

```
add $3, $8, $9
```


Write a program in MIPS that takes in the values of registers $8$ and $9$ and stores the result in register $3$.

```
add $d, $s, $t: 0000 00ss ssst tttt dddd d000 0010 0000
```

Adds the result of registers $s$ and $t$ and store these in register $d$. Important! The order of $d$, $s$ and $t$ are shifted in the encoding!
First Example

Write a program in MIPS that takes in the values of registers $8$ and $9$ and stores the result in register $3$.

```
add $d, $s, $t: 0000 00ss ssst tttt dddd d000 0010 0000
```

Adds the result of registers $s$ and $t$ and store these in register $d$. Important! The order of $d$, $s$ and $t$ are shifted in the encoding!

Solution: add $3$, $8$, $9$:

```
0000 0001 0000 1001 0001 1000 0010 0000
```
Putting Values In Registers

This works well *provided you already had values in registers*!

How to we put values in registers? In CS 241 we give you a non-standard MIPS command:

`lis $d: 0000 0000 0000 0000 dddd d000 0001 0100`

Load immediate and skip. This places the next value in RAM [an immediate] into $d and increments the program counter by 4 (it skips the next line which is usually not an instruction).

`.word i: iiii iiii iiii iiii iiii iiii iiii iiii iiii iiii`

This is an assembler directive (not a MIPS instruction). This value as a two’s complement integer is placed in the correct memory location in RAM as it occurs in the code. Can also use with hexadecimal values using `0x`i.`
Example

Write a MIPS program that adds together 11 and 13.
Example

Write a MIPS program that adds together 11 and 13.

```
is $8
.word 11
lis $9
.word 0xd
add $3, $8, $9
```

The code on the left is what we call **Assembly Code**.
The code on the right is what we call **Machine Code**.
Example

Previous example revisited.
What it looks like in RAM with memory locations

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>0000 0000 0000 0000 0100 0000 0001 0100</td>
</tr>
<tr>
<td>0x00000004</td>
<td>0000 0000 0000 0000 0000 0000 0000 1011</td>
</tr>
<tr>
<td>0x00000008</td>
<td>0000 0000 0000 0000 0000 0100 1000 0001 0100</td>
</tr>
<tr>
<td>0x0000000c</td>
<td>0000 0000 0000 0000 0000 0000 0000 0000 1101</td>
</tr>
<tr>
<td>0x00000010</td>
<td>0000 0001 0000 1001 0001 1000 0010 0000</td>
</tr>
</tbody>
</table>
Incomplete examples

The two examples above are still incomplete. Recall that the fetch-execute cycle has a while loop that we still haven’t exited yet. How to we return control back to the loader?
Incomplete examples

The two examples above are still incomplete. Recall that the fetch-execute cycle has a while loop that we still haven’t exited yet. How to we return control back to the loader?

\[ \text{jr } s: \ 0000 \ 00ss \ sss0 \ 0000 \ 0000 \ 0000 \ 0000 \ 1000 \]

Jump Register. Sets the pc to be $s$. For us, our return address will typically be in $31$ so we will typically call

\[ \text{jr } $31 \]

which is

\[ \begin{array}{cccccccccc}
0000 & 0011 & 1110 & 0000 & 0000 & 0000 & 0000 & 0000 & 1000 \\
\end{array} \]

This command returns control to the loader.
Write a MIPS program that adds together 11 and 13.
Write a MIPS program that adds together 11 and 13.

```
lis $8
.word 11
lis $9
.word 0xd
add $3, $8, $9
jr $31
```