Digital Logic Structures
Basic Logic Gates

- **NOT**
  - ![NOT Gate](image)

- **OR**
  - ![OR Gate](image)

- **NOR**
  - ![NOR Gate](image)

- **AND**
  - ![AND Gate](image)

- **NAND**
  - ![NAND Gate](image)

- **XOR**
  - ![XOR Gate](image)
More Than Two Inputs?

- AND and OR gates can take any number of inputs
  - AND gives 1 if all inputs are 1
  - OR gives 1 if any input is 1
- NAND?? NOR??
  - Not associative!
AND NAND NAND
NAND NOR
\textbf{INV}

\textbf{AND}

\textbf{OR}

\[
\bar{A} \bar{B} = (A + B)
\]

\[
\bar{A} \bar{B} = (A + B)
\]

\[
\begin{array}{cccc}
A & B & C & OUT \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]
One-Bit Full Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_in</th>
<th>C_out</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Four-Bit Full Adder

Ripple-carry adder

Look ahead adder
More Logic Structure

• As we start to build more complex structures we need ways to control parts of them
  – To select signals
  – To activate certain outputs
Signal Selection

odd

Mult
Two-Way Multiplexer
Two-Way Multiplexer

2-way multiplexer: the output is equal to one of the two inputs, based on a selector.

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Four-Way Multiplexer

- \( n \)-bit selector and \( 2^n \) inputs, one output
  - output equals one of the inputs, depending on selector
- "Four-to-one mux"
Two-to-Four Decoder

- $n$ inputs, $2^n$ outputs
  - exactly one output is 1 for each possible input pattern
- Generates a walking-ones pattern
- Uses:
  - Convert memory or register address to a control line
  - Convert an opcode to one of $n$ control lines
  - We will get to this in the LC-3 material
Building functions from logic gates

- **Combinational Logic Circuit**
  - Output depends only on the current inputs
  - Stateless (memoryless)

- **Sequential Logic Circuit**
  - Output depends on the sequence of inputs (past and present)
  - Stores information (state) from past inputs
Combinational vs. Sequential
Two types of “combination” locks

**Combinational**
Success depends only on the values, not the order in which they are set.

**Sequential**
Success depends on the sequence of values (e.g., R-13, L-22, R-3).
Combinational vs. Sequential

- Combinational circuit
  - Always gives the same output for a given set of inputs
  - Example: Adder always generates sum and carry, regardless of previous inputs

- Sequential circuit
  - Remembers previous input
  - Output depends on state and input
Synchronization of Sequential Circuits

- These are real devices and require time to compute.
- If we want proper results we need a way to ensure consistent timing.
- One way of doing this is a clock:
  - Repeating signal at certain frequency
  - When you buy a computer this is the number in gigahertz
- All our actions take place in relation to this clock.
D-Flip-Flop (the one for Lab)

- Basic Memory Device
- Stores the value of $D$ when conditions are met and outputs it on $Q$
- Otherwise $Q$ holds the last value of $D$

- D-flip-flop is edge-triggered (changes only on the edge of the clock)
- This can be both edges or a single type (up or down)
D-Flip Flop: Timing Diagram

- **Ck**
- **D**
- **Q**

Diagram showing the timing and waveforms for a D-Flip Flop.
D-Flip-Flop with Write Enable

- Same idea as a Flip-Flop but adds another input
- Instead of changing on clock edges. You can only change on a clock edge when WE is high
D-Flip Flop: Timing Diagram (up)

- **Ck**: Clock signal
- **WE**: Write Enable
- **D**: Data input
- **Q**: Output signal

Graphs show the timing relationships between these signals over time.
Register

- A register stores a multi-bit value
- Common WE which latches the n-bit value
Memory

Now that we know how to store bits, we can build a memory – a logical $k \times m$ array of stored bits.

**Address Space:** number of locations (usually a power of 2)

**Addressability:** number of bits per location (e.g., byte-addressable)
1 KB  1000 bytes

1 KB  1024 bytes
Memory
State Machine

The basic type of sequential circuit

- Combines combinational logic with storage
- "Remembers" state, and changes output (and state) based on inputs and current state
Representing Multi-bit Values

- Number bits from right (0) to left (n-1)
  - just a convention -- could be left to right, but must be consistent

- Use brackets to denote range:
  \( D[l:r] \) denotes bit \( l \) to bit \( r \), from left to right

\[
A = \begin{array}{cccccccccccccccc}
15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{array}
\]

\[
A[14:9] = 101001
\]

\[
A[2:0] = 101
\]

May also see \( A<14:9> \), especially in hardware block diagrams.
July 4th Tuesday
\[ x = 4 \]
\[ y = 3 \]
\[ z = x + y \]
1) Flip-flop
RS-Lo+ch
Arithmetic and Logical Operations: An ALU
(with a little grab bag)
Character Representation

Memory location for a character usually contains 8 bits:

- 00000000 to 11111111 (binary)
- 0x00 to 0xff (hexadecimal)

Which characters?
- A, B, C, ..., Z, a, b, c, ..., z, 0, 1, 2, ..., 9
- Punctuation (,,:{...}
- Special (\n \O ...)

Which bit patterns for which characters?
- Want a standard!!!
- Want a standard to help sort strings of characters.
Character Representation

- **ASCII** (American Standard Code for Information Interchange)
- Defines what character is represented by each sequence of bits.
- Examples:
  - 0100 0001 is 0x41 (hex) or 65 (decimal). It represents “A”.
  - 0100 0010 is 0x42 (hex) or 66 (decimal). It represents “B”.
- Different bit patterns are used for each different character that needs to be represented.
ASCII Properties

ASCII has some nice properties.

• If the bit patterns are compared, (pretending they represent integers), then

   "A" < "B"
   65  <  66

• This is good because it helps with sorting things into alphabetical order.

• But....:

   • ‘a’ (61 hex) is different than ‘A’ (41 hex)
   • ‘8’ (38 hex) is different than the integer 8
   • ‘0’ is 30 (hex) or 48 (decimal)
   • ‘9’ is 39 (hex) or 57 (decimal)
# 7-bit ASCII Table

<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>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Chr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>000</td>
<td>NUL (null)</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td>a#32: Space</td>
<td>64</td>
<td>40</td>
<td>100</td>
<td>a#64: 8</td>
<td>96</td>
<td>60</td>
<td>140</td>
<td>a#96:</td>
<td>'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>SOH (start of heading)</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td>a#33: !</td>
<td>65</td>
<td>41</td>
<td>101</td>
<td>a#65: A</td>
<td>97</td>
<td>61</td>
<td>141</td>
<td>a#97: a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>002</td>
<td>STX (start of text)</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td>a#34: &quot;</td>
<td>66</td>
<td>42</td>
<td>102</td>
<td>a#66: B</td>
<td>98</td>
<td>62</td>
<td>142</td>
<td>a#98: b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>003</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td>a#35: #</td>
<td>67</td>
<td>43</td>
<td>103</td>
<td>a#67: C</td>
<td>99</td>
<td>63</td>
<td>143</td>
<td>a#99: c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>004</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td>a#36: $</td>
<td>68</td>
<td>44</td>
<td>104</td>
<td>a#68: D</td>
<td>100</td>
<td>64</td>
<td>144</td>
<td>a#100: d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>005</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td>a#37: %</td>
<td>69</td>
<td>45</td>
<td>105</td>
<td>a#69: E</td>
<td>101</td>
<td>65</td>
<td>145</td>
<td>a#101: e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>006</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td>a#38: &amp;</td>
<td>70</td>
<td>46</td>
<td>106</td>
<td>a#70: F</td>
<td>102</td>
<td>66</td>
<td>146</td>
<td>a#102: f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>007</td>
<td>BEL (bell)</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td>a#39: '</td>
<td>71</td>
<td>47</td>
<td>107</td>
<td>a#71: G</td>
<td>103</td>
<td>67</td>
<td>147</td>
<td>a#103: g</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>010</td>
<td>BS (backspace)</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td>a#40: (</td>
<td>72</td>
<td>48</td>
<td>110</td>
<td>a#72: H</td>
<td>104</td>
<td>68</td>
<td>150</td>
<td>a#104: h</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>011</td>
<td>TAB (horizontal tab)</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td>a#41: )</td>
<td>73</td>
<td>49</td>
<td>111</td>
<td>a#73: I</td>
<td>105</td>
<td>69</td>
<td>151</td>
<td>a#105: i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>012</td>
<td>LF (NL line feed, new line)</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td>a#42: *</td>
<td>74</td>
<td>4A</td>
<td>112</td>
<td>a#74: J</td>
<td>106</td>
<td>6A</td>
<td>152</td>
<td>a#106: j</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>013</td>
<td>VT (vertical tab)</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td>a#43: +</td>
<td>75</td>
<td>4B</td>
<td>113</td>
<td>a#75: K</td>
<td>107</td>
<td>6B</td>
<td>153</td>
<td>a#107: k</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>014</td>
<td>FF (NF form feed, new page)</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td>a#44: ,</td>
<td>76</td>
<td>4C</td>
<td>114</td>
<td>a#76: L</td>
<td>108</td>
<td>6C</td>
<td>154</td>
<td>a#108: l</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>015</td>
<td>CR (carriage return)</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td>a#45: -</td>
<td>77</td>
<td>4D</td>
<td>115</td>
<td>a#77: M</td>
<td>109</td>
<td>6D</td>
<td>155</td>
<td>a#109: m</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>016</td>
<td>SO (shift out)</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td>a#46: .</td>
<td>78</td>
<td>4E</td>
<td>116</td>
<td>a#78: N</td>
<td>110</td>
<td>6E</td>
<td>156</td>
<td>a#110: n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>017</td>
<td>SI (shift in)</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td>a#47: /</td>
<td>79</td>
<td>4F</td>
<td>117</td>
<td>a#79: O</td>
<td>111</td>
<td>6F</td>
<td>157</td>
<td>a#111: o</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>020</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td>a#48: 0</td>
<td>80</td>
<td>50</td>
<td>120</td>
<td>a#80: P</td>
<td>112</td>
<td>70</td>
<td>160</td>
<td>a#112: p</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>021</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td>a#49: 1</td>
<td>81</td>
<td>51</td>
<td>121</td>
<td>a#81: Q</td>
<td>113</td>
<td>71</td>
<td>161</td>
<td>a#113: q</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>022</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td>a#50: 2</td>
<td>82</td>
<td>52</td>
<td>122</td>
<td>a#82: R</td>
<td>114</td>
<td>72</td>
<td>162</td>
<td>a#114: r</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>023</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td>a#51: 3</td>
<td>83</td>
<td>53</td>
<td>123</td>
<td>a#83: S</td>
<td>115</td>
<td>73</td>
<td>163</td>
<td>a#115: s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>024</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td>a#52: 4</td>
<td>84</td>
<td>54</td>
<td>124</td>
<td>a#84: T</td>
<td>116</td>
<td>74</td>
<td>164</td>
<td>a#116: t</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>025</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td>a#53: 5</td>
<td>85</td>
<td>55</td>
<td>125</td>
<td>a#85: U</td>
<td>117</td>
<td>75</td>
<td>165</td>
<td>a#117: u</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td>a#54: 6</td>
<td>86</td>
<td>56</td>
<td>126</td>
<td>a#86: V</td>
<td>118</td>
<td>76</td>
<td>166</td>
<td>a#118: v</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>027</td>
<td>ETB (end of block)</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td>a#55: 7</td>
<td>87</td>
<td>57</td>
<td>127</td>
<td>a#87: W</td>
<td>119</td>
<td>77</td>
<td>167</td>
<td>a#119: w</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>030</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>38</td>
<td>070</td>
<td>a#56: 8</td>
<td>88</td>
<td>58</td>
<td>130</td>
<td>a#88: X</td>
<td>120</td>
<td>78</td>
<td>170</td>
<td>a#120: x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>031</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td>a#57: 9</td>
<td>89</td>
<td>59</td>
<td>131</td>
<td>a#89: Y</td>
<td>121</td>
<td>79</td>
<td>171</td>
<td>a#121: y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>032</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>3A</td>
<td>072</td>
<td>a#58: :</td>
<td>90</td>
<td>5A</td>
<td>132</td>
<td>a#90: Z</td>
<td>122</td>
<td>7A</td>
<td>172</td>
<td>a#122: z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>033</td>
<td>ESC (escape)</td>
<td>59</td>
<td>3B</td>
<td>073</td>
<td>a#59: ;</td>
<td>91</td>
<td>5B</td>
<td>133</td>
<td>a#91: ]</td>
<td>123</td>
<td>7B</td>
<td>173</td>
<td>a#123: {</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>034</td>
<td>FS (file separator)</td>
<td>60</td>
<td>3C</td>
<td>074</td>
<td>a#60: &lt;</td>
<td>92</td>
<td>5C</td>
<td>134</td>
<td>a#92: \</td>
<td>124</td>
<td>7C</td>
<td>174</td>
<td>a#124:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>035</td>
<td>GS (group separator)</td>
<td>61</td>
<td>3D</td>
<td>075</td>
<td>a#61: =</td>
<td>93</td>
<td>5D</td>
<td>135</td>
<td>a#93:</td>
<td></td>
<td>125</td>
<td>7D</td>
<td>175</td>
<td>a#125: )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>036</td>
<td>RS (record separator)</td>
<td>62</td>
<td>3E</td>
<td>076</td>
<td>a#62: &gt;</td>
<td>94</td>
<td>5E</td>
<td>136</td>
<td>a#94: ^</td>
<td>126</td>
<td>7E</td>
<td>176</td>
<td>a#126: _</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>037</td>
<td>US (unit separator)</td>
<td>63</td>
<td>3F</td>
<td>077</td>
<td>a#63: ?</td>
<td>95</td>
<td>5F</td>
<td>137</td>
<td>a#95: `</td>
<td>127</td>
<td>7F</td>
<td>177</td>
<td>a#127: DEL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
10

↓

Escape character

"Hello World\n"

H e l l o l o w 1 0 1 0 1 d w 0
Logical Operations

Operate on raw bits with $1 = \text{true}$ and $0 = \text{false}$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>AND</th>
<th>OR</th>
<th>NAND</th>
<th>NOR</th>
<th>XOR</th>
<th>XNOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The symbols are C bitwise operators
"bit-wise" logical operations are done in parallel for corresponding bits

Example:

\[
\begin{align*}
X &= 0011 \\
Y &= 1010
\end{align*}
\]

\[
X \, \text{AND} \, Y = 0010
\]

So how do an OR? How about an XOR?
Masking

- Want to look only at **certain** bits of a **binary** word
- Use a mask to remove the uninteresting bits
- It is a bitwise AND operation with the MASK
- Example:

  mask: 100000000

<table>
<thead>
<tr>
<th>Value</th>
<th>1001 1101</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASK</td>
<td>0000 1111</td>
</tr>
<tr>
<td>Result</td>
<td>0000 1101</td>
</tr>
</tbody>
</table>
Overflow

- Remember when we threw away that bit while doing 2’s complement addition.
- That bit is thrown away safely and does not change the result.
- Overflow can also cause invalid results.
Overflow in Addition

Unsigned: When there is a carry out of the MSB

\[
\begin{align*}
1000 & \quad (8) \\
+1001 & \quad (9) \\
\underline{+} & \\
0001 & \quad \times (1)
\end{align*}
\]
Overflow in Addition

**2’s complement:** When the signs of the addends are the same, but the sign of the result is different.

\[
\begin{align*}
0011 & \quad (3) \\
+ & \quad 0110 \\
\hline
1001 & \quad (-7)
\end{align*}
\]

Adding 2 numbers of opposite signs never overflows.

Why?
Overflow in Subtraction

\[ 3 - 4 \]

Unsigned: if result would be negative

2’s complement: never do subtraction, so use the addition rule on the addition operation done.
Sign Extension

How to change a number with a smaller number of bits into the same number (same representation) with a larger number of bits?

This must be done frequently by arithmetic units.
Sign Extension - Unsigned

Copy the original integer into the LSBs, and put 0’s elsewhere.

Thus for 5 bits to 8 bits:

```
xxxxx -> 000xxxxxx
```
Sign Extension – 2SC

1. Copy the original $n-1$ bits into the LSBs
2. Take the MSB of the original and copy it elsewhere

Thus for 6 bits to 8 bits:

- $\text{sxxxxx} \rightarrow \text{ssssxxxx}$
Sign Extension – 2SC

• In 2’s complement, the MSB (sign bit) is the $-2^{n-1}$ place.
• It says “subtract $2^{n-1}$ from $b_{n-2}...b_0$”.
• Sign extending one bit
  • Adds a $-2^n$ place
  • Changes the old sign bit to a $+2^{n-1}$ place
  • $-2^n + 2^{n-1} = -2^{n-1}$, so the number stays the same
\[ -3 \]

\[ \begin{array}{c}
0011 \\
1100 \\
+ 1 \\
\hline \\
1101 \\
\end{array} \]

\[ 4 - 6i + 1 \]

\[ -8 + 4 + 0 + 1 \]

\[ 16 + 8 + 4 + 0 + 1 \]

\[ -32 + 16 - 8 + 4 + 0 + 1 \]
Unsigned Binary Multiplication

The **multiplicand** is multiplied by the **multiplier** to produce the **product**, the sum of **partial products**

\[ \text{multiplicand} \times \text{multiplier} = \text{product} \]

```
0011  (+3)  
\times 0110  (+6)  
\hline
0000
00110
001100
0000000
\hline
0010010  (+18)
```

- Longhand, it looks just like decimal
- Result can require twice as many bits as the larger multiplicand (why?)
111
\times 101
\hline
1111
100000
\hline
1110000
+ \quad 1110000
\hline
1000111

6 bit

1
2
4
8
16
32
63
2’s Complement Multiplication

- If negative multiplicand, just sign-extend it.
- If negative multiplier, take 2SC of both multiplicand and multiplier (-7 x -3 = 7 x 3, and 7 x -3 = -7 x 3), also known as the additive inverse.

```
0011 (3)  x  1011 (-5)  \\
\hline
11111101
-0000000
11110100
\hline
11110001 (-15)
```

Only need 8 bits for result.
Division

Only required to know for unsigned binary

Just like you do with decimal long hand

\[ 14 \div 2 = 101110 \]
Shifts and Rotates

Logical right
- Move bits to the right, same order
- Throw away the bit that pops off the LSB
- Introduce a 0 into the MSB
  \[ 00110101 \rightarrow 00011010 \] (shift right by 1)

Logical left
- Move bits to the left, same order
- Throw away the bit that pops off the MSB
- Introduce a 0 into the LSB
  \[ 00110101 \rightarrow 11010100 \] (shift left by 2)
$16 + 2 = 18$
$8 + 1 = 9$
$4 + 0 = 4$
Arithmetic Shifts

Arithmetic right shift
- Move bits to the right, same order
- Throw away the bit that pops off the LSB
- Reproduce the original MSB into the new MSB
- Alternatively, shift the bits, and then do sign extension
  \[ \begin{align*}
    00110101 & \rightarrow 00011010 & \text{(right by 1)} \\
    1100 & \rightarrow 1111 & \text{(right by 2)}
  \end{align*} \]

Arithmetic left shift
- Move bits to the left, same order
- Throw away the bit that pops off the MSB
- Introduce a 0 into the LSB
  \[ \begin{align*}
    00110101 & \rightarrow 01101010 & \text{(left by 1)}
  \end{align*} \]
Rotations

Rotate left
• Move bits to the left, same order
• Put the bit(s) that pop off the MSB into the LSB
• No bits are thrown away or lost
  00110101 \rightarrow 01101010 \quad \text{(rotate by 1)}
  1100 \rightarrow 1001 \quad \text{(rotate by 1)}

Rotate right
• Move bits to the right, same order
• Put the bit that pops off the LSB into the MSB
• No bits are thrown away or lost
  00110101 \rightarrow 10011010 \quad \text{(rotate by 1)}
  1101 \rightarrow 01111 \quad \text{(rotate by 2)}
Building the ALU

- Now that we have the operations needed (and could design the hardware to do so) we can combine them to build our ALU.
- We need to build a system where we can select a command and have that operation performed.
The ALU

AND

ADD

NOT

I₁

I₂

Iₙ

T₁

T₂

Oₙ

I₁

I₂

Rₙ
Multiplication in hardware

\[ 3 \times 4 = 3 + 3 + 3 + 3 + 3 \]
Subtraction (4-bit)
\[ a - b \]
Symbol

N
Abstraction and Computing Systems
Computing machines are everywhere

- General purpose
  - Servers, desktops, laptops, tablets, smart phones, etc.

- Special purpose
  - Cash registers, ATMs, games, telephone switches, etc.

- Embedded
  - Cars, hotel doors, printers, VCRs, industrial machinery, medical equipment, etc.
Computing machines: distinguishing features

- Speed: 40 MHz - GHz
- Cost: 1¢ - $2.00
- Price/Performance
- Ease of use, software support & interface
- Scalability
- Power
- Size
Computing System

Samsung Galaxy S5

Dell computer

Refrigerator with touch screen
1st Very Important Idea

- Universal Computational Devices
  - Given enough time and memory, all computers are capable of computing exactly the same things
  - Irrespective of speed, size, or cost

- Turing’s Thesis
  - Every computation can be performed by some Turing Machine – a theoretical universal computational device
Alan Turing’s original model

(1912-1954)
A Turing Machine

Also known as a *Universal Computational Device*: a theoretical device that accepts both input data and instructions on how to operate on the data.
2nd Very Important Idea

- Problem Transformation
  - The ultimate objective is to transform a problem expressed in natural language into electrons running around a circuit
- This is computer science and computer engineering
  - A continuum that embraces software and hardware
Computer Architecture

Problems

Algorithms

Language

Instruction Set Architecture

Microarchitecture

Circuits

Devices
Computer Science

**Definition:** The study of algorithms and data structures to solve problems.

**Abstraction:** Use of level of abstraction in software design allows the programmer to focus on a critical set of problems without having to deal with irrelevant details.
Procedure or Function

int average (a, b) begin
    int avg;
    avg = (a+b)/2;
    return (avg);
end

main ()

... 
  x = 4;
  y = 2;
  k = average (x,y);
  print ("8d", k);
...

Maxwell James Dunne
Compiler: A computer program that translates code written in a high level language into an intermediate level abstract language.

Assembler: A computer program that translates code written in assembly language to the binary form that the CPU can execute.
Computer Engineering

Definition: The creative application of engineering principles and methods to the design and development of hardware and software systems.

Abstraction: Use of level of abstraction in hardware design allows the designer to focus on a critical set of problems without having to deal with irrelevant details.
Instruction Set Architecture (ISA)

Definition: Interface between a computer’s hardware and its software. Defines exactly what the computer’s instructions do, and how they are specified.

x86
x64
Central Processing Unit

The heart of computing systems

ca 1980
It took 10 of these boards to make a Central Processing Unit (CPU)

ca 2000
No wonder they called this CPU a microprocessor!
Motherboard: System
CPU: Package
SoC – System on a chip

800 PROCESSOR

Krait 400 CPU features 28Hpm process technology superior 2GHz+ performance

Adreno 330 for advanced graphics

Hexagon QDSP6 for ultra low power applications and custom programmability

Integrated LTE, 802.11ac, USB 3.0 and BT 4.0 offers broad array of high speed connectivity

CONNECTIVITY

4G LTE, WIFI, USB, BT and FM

MULTIMEDIA

Audio, Video and Gestures

CAMERA

DISPLAY/LCD

NAVIGATION

Ultra HD Capture and Playback
DTS-HD and Dolby Digital Plus audio
Expanded Gestures

55MP with dual ISP
Support for up to 2560x2048 display
Minicasts 1080p HD support
I2sat GHSS with support for three GPS constellations
CPU: Microarchitecture

Intel Core i7
CPU: Die with graphics core

Intel® Core™ M Processor Die Map
14nm 2nd Generation Tri-Gate 3-D Transistors

Intel® Core™ M Processor
Dual Core Die Shown Above
Transistor Count: 1.3 Billion
Die Size: 82mm²

Memory Controller I/O

Shared L3 Cache**
Core
Core
System Agent, Display Engine & Memory Controller

4th Gen Core Processor (Y series): 86B
** Cache is shared across both cores and processor graphics

Intel Confidential – UNDER EMBARGO UNTIL SEPTEMBER 5TH 2014 8:30AM PT
All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
Two recurring themes

1) Abstraction
   – The notion that we can concentrate on one "level" of the big picture at a time, with confidence that we can then connect effectively with the levels above and below.
   – Framing the levels of abstraction appropriately is one of the most important skills in any undertaking.
Two recurring themes

2) Hardware vs. Software

- On the other hand, abstraction does not mean being clueless about the neighboring levels.
- In particular, hardware and software are inseparably connected, especially at the level we will be studying.
What is Computer Organization?

There is a fundamentally wide gap between the intended behavior desired and the workings of the electronic devices that do the work.

Before the digital computers of today special purpose analog devices (mechanical, electrical, or electronic) were built for each desired behavior.
A general purpose computer is the bridge that links the desired behavior (application) and the basic building blocks (electronic devices).
Our computer model for now

CPU Interacts with the memory in 3 ways:
- fetches instructions
- loads the value of a variable
- stores the new value of a variable

Memory is capable of only 2 operations:
- reads – a load or a fetch
- writes – operation of storing the value of a variable
Our LC-3 Model