Information Storage

CMPU 224 – Computer Organization
Jason Waterman
What is a bit?

• All data stored in computer systems (hard drives, memory, SD cards, etc.) is stored as binary digits (bits)
• A bit represents one of two states, “on/off”, “true/false”, “1/0”
• How that bit is stored depends on the medium
  • Magnetic (hard drive, floppy disk)
  • Electronic (RAM, Registers)
  • Optical (CD / DVD / Punch Cards)

• Regardless of how it’s stored
  • A bit takes on the value of either 0 or 1
• Why use bits? The hardware for storing and performing computation on bits is simple and reliable
Bytes and Words

- A bit is not very much data, so we usually group a bunch of bits together into logical groupings.
- A **byte** is a group of 8 bits:
  - 01100110
  - 01100011
  - 00110010
  - 10101010
- How many unique bytes are there?
  - $2^8 = 256$, so a byte can represent a set of up to 256 items.
- Memory is byte addressable.
- Bytes are still too small to be the basic size of data for a computer, so we use the **word** size instead:
  - Indicates the size of pointer data (memory address size).
  - In the past the basic word size of most computers was 32 bits (4 GB address space).
  - Today newer machines have a word size of 64 bits (16 exabytes address space).
- 00101010010110101010011001101101101101101110000100110001111110100
  - Looking at a string of 64 bits is somewhat overwhelming for humans.
  - We’ll look at a way to represent words more compactly later.
Representing Numbers: Decimals

• One of the first things we’d like to do is represent numbers in binary
• But first a quick review of decimal numbers to help us understand binary numbers
  • Decimal numbers are known as a base 10 system
  • Numbers are represented by the ten symbols: 0-9
  • A decimal number has a one’s place (10^0), a ten’s place (10^1), a hundred’s place (10^2), a thousand’s place (10^3) and so on
• Example: decimal number 224

\[
\begin{array}{ccc}
10^2 & 10^1 & 10^0 \\
100 & 10 & 1 \\
2 & 2 & 4 \\
\end{array}
\]

\[2 \times 100 + 2 \times 10 + 4 \times 1 = 200 + 20 + 4 = 224\]
Binary (base 2) Number System

• Binary is base 2
  • Numbers are represented by the symbols 0 and 1
  • A binary number has a one’s place \(2^0\), a two’s place \(2^1\), a four’s place \(2^2\), an eight’s place \(2^3\), and so on
  • Used to represent unsigned integers

• Example: \(0101_2 = 5_{10}\)

\[
\begin{array}{cccc}
8 & 4 & 2 & 1 \\
2^3 & 2^2 & 2^1 & 2^0 \\
\hline
0 & 1 & 0 & 1 \\
\end{array}
\]

\[4 + 1 = 5\]
Reading Binary

• With practice, you will soon become comfortable reading binary numbers up to 255
  • All you need to do is add combinations of:
  • 1, 2, 4, 8, 16, 32, 64, and 128

• Just like decimal numbers the least significant digit (or least significant bit -- lsb) is on the right
  • In the number 123, the rightmost digit (3) is in the one’s place, etc.
  • Example: for the binary number 1010
    • There is a 1 in the 2’s place and a 1 in the 8’s place, so the value is $2 + 8 = 10$
Converting from decimal to binary

• Example: $42_{10}$
  • 32 is the largest power of two number $\leq 42$ so we know we have a one in the 32’s column and we subtract 32 from 42, leaving 10
  • 16 $> 10$, so we have a zero in the 16’s column
  • 8 $\leq 10$, so we have a one in the 8’s column and we subtract 8 from 10, leaving 2
  • 4 $> 2$, so we have a zero in the 4’s column
  • 2 $\geq 2$, so we have a one in the 2’s column and we subtract 2 from 2, leaving 0
  • 1 $> 0$, so we have a zero in the 1’s column
  • Putting it all together $42_{10} = 101010_2$
Hexadecimal (base 16) Number System

- Hexadecimal (hex) is a base 16 representation
  - We use the letters A-F as the extra “digits” so we count:
    - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
    - 10 11 12 13 14 15
  - A hexadecimal number has a one’s place \( (16^0) \), a sixteen’s place \( (16^1) \), a two-hundred-fifty-six’s place \( (16^2) \), and so on
  - \( 0x54B54CDB6DC263F4 \)

- Each hexadecimal digit represents how many bits?
  - 4
- How many hexadecimal digits are in a byte?
  - 2
- How many hexadecimal digits to represent a 64-bit number?
  - 16
Hexadecimal Conversion

• Mapping to and from binary and hex is straightforward because base-16 is also a power of 2

• Binary to Hex:
  Starting from the right-hand side, group the bits in sets of four, converting each set of 4 bits to its hex digit
  Example:
  1001101010101000101
  1000 1101 0101 0100 0101
  8   D   5   4   5
  0x8D545

• Hex to Binary
  Convert each hex digit into 4 bits:
  0xF7B36
  1111 0111 1011 0011 0110
Data Sizes in C (Typical)

- The sizes of the basic data types can vary based on compiler and machine settings
- These are typical values on a x86-64 system

<table>
<thead>
<tr>
<th>C Declaration</th>
<th>Size in Bytes (64-bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>char *</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
</tbody>
</table>
Pointers in C

• Pointers in C are all about memory addresses
  • Pointers are variables that store the location of other variables in memory
  • Think of them as shortcuts to data hidden in different corners of your program's storage

• Declaration
  • You introduce pointers with an asterisk before the variable type
  • `int *ptr;`
  • Creates a pointer named `ptr` that can hold the address of an integer variable.

• Getting an Address
  • Use the ampersand `&` (address of operator) before a variable to get its address
  • `int num = 10;`
  • `int *ptr = &num;`
  • Now `ptr` has the value of memory location as `num`, where the value 10 resides

• Accessing Data
  • Use the asterisk before the pointer to peek inside the memory location it points to
  • `int value = *ptr;`
  • This assigns the value stored at the address held by `ptr` (which is 10) to the variable `value`. 
#include <stdio.h>

int main() {

    int num;  // integer declaration of variable x
    int *ptr; // int pointer declaration of variable ptr

    num = 10;

    printf("num = 10;\n");
    printf("value of num: %d\n", num);
    printf("address of num in memory: %p\n", &num);

    ptr = &num; // Address of the variable num

    printf("ptr = &num;\n");
    printf("value of ptr: %p\n", ptr);
    printf("value at the memory address pointed to by ptr: %d\n", *ptr);
}
Storing Multi-Byte Data

• Memory is byte addressable
  • Every byte of memory has an address
  • Think of memory as a large array with the address as the index in the array

• For multi-byte data
  • The address specifies starting byte location of the data in memory
  • The rest of the data is in the increasing memory addresses that follow

• Example
  • The int at address 4 contains the four bytes: 31 76 D9 5C

<table>
<thead>
<tr>
<th>ADDR 0</th>
<th>Address</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>94</td>
</tr>
<tr>
<td>1</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7F</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>EA</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDR 4</th>
<th>Address</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>D9</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>5C</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDR 8</th>
<th>Address</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>AB</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>BB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDR 12</th>
<th>Address</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>4E</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>05</td>
<td></td>
</tr>
</tbody>
</table>
Byte Ordering

• There are two conventions for the layout of multi-byte objects
  • Big Endian and Little Endian

• Example
  • 4-byte integer of \(0x01234567\) is located at memory address \(0x100\)
  • This value exists in memory locations \(0x100\), \(0x101\), \(0x102\), \(0x103\)
Byte Ordering Takeaways

• X86-64 uses **little endian**
  • When reading data from left to right in increasing memory order:
    • The bytes will be in reverse order from how the number is written

• **Bytes** are always written from msb (most significant bit) to lsb (least significant bit) in both endian conventions

• Little endian systems will always have the end byte at the smallest (littlest) address

• Big endian systems will always have the end byte at the largest (biggest) address
Representing Strings

• There is no native type for strings in C
  • Instead, strings are represented as an array of characters (char)
  • Terminated by the null (the literal value 0) character

• Characters
  • 1 byte represented by an ASCII (American Standard Code for Information Interchange) encoding

• Arrays
  • Data of the same type ordered in contiguous memory
  • We’ll talk more about arrays later

• Example: string “Hello” at address 4
  • Note the string takes up 6 bytes of memory because of the null termination byte
### ASCII (American Standard Code for Information Interchange) Table

<table>
<thead>
<tr>
<th>dec</th>
<th>hex</th>
<th>oct</th>
<th>char</th>
<th>dec</th>
<th>hex</th>
<th>oct</th>
<th>char</th>
<th>dec</th>
<th>hex</th>
<th>oct</th>
<th>char</th>
<th>dec</th>
<th>hex</th>
<th>oct</th>
<th>char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
<td>NULL</td>
<td>0</td>
<td>040</td>
<td>000</td>
<td>space</td>
<td>64</td>
<td>000</td>
<td>100</td>
<td>@</td>
<td>96</td>
<td>000</td>
<td>140</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>041</td>
<td>SOH</td>
<td>1</td>
<td>041</td>
<td>041</td>
<td></td>
<td></td>
<td>65</td>
<td>010</td>
<td>101</td>
<td>A</td>
<td>97</td>
<td>061</td>
<td>141</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>042</td>
<td>STX</td>
<td>2</td>
<td>042</td>
<td>042</td>
<td></td>
<td></td>
<td>66</td>
<td>020</td>
<td>102</td>
<td>B</td>
<td>98</td>
<td>062</td>
<td>142</td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>043</td>
<td>ETX</td>
<td>3</td>
<td>043</td>
<td>043</td>
<td>#</td>
<td>67</td>
<td>043</td>
<td>103</td>
<td>C</td>
<td>99</td>
<td>063</td>
<td>143</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>044</td>
<td>EOT</td>
<td>4</td>
<td>044</td>
<td>044</td>
<td>$</td>
<td>68</td>
<td>044</td>
<td>104</td>
<td>D</td>
<td>100</td>
<td>064</td>
<td>144</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>045</td>
<td>ENQ</td>
<td>5</td>
<td>045</td>
<td>045</td>
<td>%</td>
<td>69</td>
<td>045</td>
<td>105</td>
<td>E</td>
<td>101</td>
<td>065</td>
<td>145</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>046</td>
<td>ACK</td>
<td>6</td>
<td>046</td>
<td>046</td>
<td>&amp;</td>
<td>70</td>
<td>046</td>
<td>106</td>
<td>F</td>
<td>102</td>
<td>066</td>
<td>146</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>047</td>
<td>BEL</td>
<td>7</td>
<td>047</td>
<td>047</td>
<td>'</td>
<td>71</td>
<td>047</td>
<td>107</td>
<td>G</td>
<td>103</td>
<td>067</td>
<td>147</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>050</td>
<td>BS</td>
<td>8</td>
<td>050</td>
<td>050</td>
<td>(</td>
<td>72</td>
<td>050</td>
<td>110</td>
<td>H</td>
<td>104</td>
<td>068</td>
<td>150</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>051</td>
<td>TAB</td>
<td>9</td>
<td>051</td>
<td>051</td>
<td>)</td>
<td>73</td>
<td>051</td>
<td>111</td>
<td>I</td>
<td>105</td>
<td>069</td>
<td>151</td>
<td>i</td>
</tr>
<tr>
<td>a</td>
<td>012</td>
<td>052</td>
<td>LF</td>
<td>a</td>
<td>052</td>
<td>052</td>
<td>*</td>
<td>74</td>
<td>052</td>
<td>112</td>
<td>J</td>
<td>106</td>
<td>070</td>
<td>152</td>
<td>j</td>
</tr>
<tr>
<td>b</td>
<td>013</td>
<td>053</td>
<td>VT</td>
<td>b</td>
<td>053</td>
<td>053</td>
<td>+</td>
<td>75</td>
<td>053</td>
<td>113</td>
<td>K</td>
<td>107</td>
<td>071</td>
<td>153</td>
<td>k</td>
</tr>
<tr>
<td>c</td>
<td>014</td>
<td>054</td>
<td>FF</td>
<td>c</td>
<td>054</td>
<td>054</td>
<td>,</td>
<td>76</td>
<td>054</td>
<td>114</td>
<td>L</td>
<td>108</td>
<td>072</td>
<td>154</td>
<td>l</td>
</tr>
<tr>
<td>d</td>
<td>015</td>
<td>055</td>
<td>CR</td>
<td>d</td>
<td>055</td>
<td>055</td>
<td>-</td>
<td>77</td>
<td>055</td>
<td>115</td>
<td>M</td>
<td>109</td>
<td>073</td>
<td>155</td>
<td>m</td>
</tr>
<tr>
<td>e</td>
<td>016</td>
<td>056</td>
<td>SO</td>
<td>e</td>
<td>056</td>
<td>056</td>
<td>.</td>
<td>78</td>
<td>056</td>
<td>116</td>
<td>N</td>
<td>110</td>
<td>074</td>
<td>156</td>
<td>n</td>
</tr>
<tr>
<td>f</td>
<td>017</td>
<td>057</td>
<td>SI</td>
<td>f</td>
<td>057</td>
<td>057</td>
<td>/</td>
<td>79</td>
<td>057</td>
<td>117</td>
<td>O</td>
<td>111</td>
<td>075</td>
<td>157</td>
<td>o</td>
</tr>
<tr>
<td>g</td>
<td>020</td>
<td>060</td>
<td>DLE</td>
<td>g</td>
<td>060</td>
<td>060</td>
<td>0</td>
<td>80</td>
<td>060</td>
<td>120</td>
<td>P</td>
<td>112</td>
<td>076</td>
<td>160</td>
<td>p</td>
</tr>
<tr>
<td>h</td>
<td>021</td>
<td>061</td>
<td>DC1</td>
<td>h</td>
<td>061</td>
<td>061</td>
<td>1</td>
<td>81</td>
<td>061</td>
<td>121</td>
<td>Q</td>
<td>113</td>
<td>077</td>
<td>161</td>
<td>q</td>
</tr>
<tr>
<td>i</td>
<td>022</td>
<td>062</td>
<td>DC2</td>
<td>i</td>
<td>062</td>
<td>062</td>
<td>2</td>
<td>82</td>
<td>062</td>
<td>122</td>
<td>R</td>
<td>114</td>
<td>078</td>
<td>162</td>
<td>r</td>
</tr>
<tr>
<td>j</td>
<td>023</td>
<td>063</td>
<td>DC3</td>
<td>j</td>
<td>063</td>
<td>063</td>
<td>3</td>
<td>83</td>
<td>063</td>
<td>123</td>
<td>S</td>
<td>115</td>
<td>079</td>
<td>163</td>
<td>s</td>
</tr>
<tr>
<td>k</td>
<td>024</td>
<td>064</td>
<td>DC4</td>
<td>k</td>
<td>064</td>
<td>064</td>
<td>4</td>
<td>84</td>
<td>064</td>
<td>124</td>
<td>T</td>
<td>116</td>
<td>080</td>
<td>164</td>
<td>t</td>
</tr>
<tr>
<td>l</td>
<td>025</td>
<td>065</td>
<td>NAK</td>
<td>l</td>
<td>065</td>
<td>065</td>
<td>5</td>
<td>85</td>
<td>065</td>
<td>125</td>
<td>U</td>
<td>117</td>
<td>081</td>
<td>165</td>
<td>u</td>
</tr>
<tr>
<td>m</td>
<td>026</td>
<td>066</td>
<td>SYN</td>
<td>m</td>
<td>066</td>
<td>066</td>
<td>6</td>
<td>86</td>
<td>066</td>
<td>126</td>
<td>V</td>
<td>118</td>
<td>082</td>
<td>166</td>
<td>v</td>
</tr>
<tr>
<td>n</td>
<td>027</td>
<td>067</td>
<td>ETB</td>
<td>n</td>
<td>067</td>
<td>067</td>
<td>7</td>
<td>87</td>
<td>067</td>
<td>127</td>
<td>W</td>
<td>119</td>
<td>083</td>
<td>167</td>
<td>w</td>
</tr>
<tr>
<td>o</td>
<td>030</td>
<td>070</td>
<td>CAN</td>
<td>o</td>
<td>070</td>
<td>070</td>
<td>8</td>
<td>88</td>
<td>070</td>
<td>130</td>
<td>X</td>
<td>120</td>
<td>084</td>
<td>170</td>
<td>x</td>
</tr>
<tr>
<td>p</td>
<td>031</td>
<td>071</td>
<td>EM</td>
<td>p</td>
<td>071</td>
<td>071</td>
<td>9</td>
<td>89</td>
<td>071</td>
<td>131</td>
<td>Y</td>
<td>121</td>
<td>085</td>
<td>171</td>
<td>y</td>
</tr>
<tr>
<td>q</td>
<td>032</td>
<td>072</td>
<td>SUB</td>
<td>q</td>
<td>072</td>
<td>072</td>
<td>:</td>
<td>90</td>
<td>072</td>
<td>132</td>
<td>Z</td>
<td>122</td>
<td>086</td>
<td>172</td>
<td>z</td>
</tr>
<tr>
<td>r</td>
<td>033</td>
<td>073</td>
<td>ESC</td>
<td>r</td>
<td>073</td>
<td>073</td>
<td>;</td>
<td>91</td>
<td>073</td>
<td>133</td>
<td>(</td>
<td>123</td>
<td>087</td>
<td>173</td>
<td>(</td>
</tr>
<tr>
<td>s</td>
<td>034</td>
<td>074</td>
<td>FS</td>
<td>s</td>
<td>074</td>
<td>074</td>
<td>&lt;</td>
<td>92</td>
<td>074</td>
<td>134</td>
<td>)</td>
<td>124</td>
<td>088</td>
<td>174</td>
<td>)</td>
</tr>
<tr>
<td>t</td>
<td>035</td>
<td>075</td>
<td>GS</td>
<td>t</td>
<td>075</td>
<td>075</td>
<td>=</td>
<td>93</td>
<td>075</td>
<td>135</td>
<td></td>
<td></td>
<td>125</td>
<td>089</td>
<td>175</td>
</tr>
<tr>
<td>u</td>
<td>036</td>
<td>076</td>
<td>RS</td>
<td>u</td>
<td>076</td>
<td>076</td>
<td>&gt;</td>
<td>94</td>
<td>076</td>
<td>136</td>
<td>_</td>
<td>126</td>
<td>090</td>
<td>176</td>
<td>_</td>
</tr>
<tr>
<td>v</td>
<td>037</td>
<td>077</td>
<td>US</td>
<td>v</td>
<td>077</td>
<td>077</td>
<td>?</td>
<td>95</td>
<td>077</td>
<td>137</td>
<td>DEL</td>
<td>127</td>
<td>091</td>
<td>177</td>
<td>DEL</td>
</tr>
</tbody>
</table>
Representing Code

• Binary programs have a well-defined format
  • Linux uses the ELF binary format

• Programs loaded into memory are represented as a stream of bytes written in machine language

00000000000006ca <main>:

6ca:  55                      push %rbp
6cb:  48 89 e5                mov %rsp,%rbp
6ce:  48 83 ec 20             sub $0x20,%rsp
6d2:  89 7d ec                mov %edi,-0x14(%rbp)
6d5:  48 89 75 e0             mov %rsi,-0x20(%rbp)
6d9:  83 7d ec 02             cmpl $0x2,-0x14(%rbp)
6dd:  74 13                   je 6f2 <main+0x28>