Bits, Bytes, and Integers

Kai Shen

Bits, Bytes, and Integers

- Representing information as bits
- Bit-level manipulations
- Integers
  - Representation: unsigned and signed
  - Conversion, casting

Byte-Oriented Memory Organization

- Programs Refer to Virtual Addresses
- Conceptually very large array of bytes
- Actually implemented with hierarchy of different memory types
- System provides address space private to particular “process”
  - Program being executed
  - Program can clobber its own data, but not that of others
- Compiler + Run-Time System Control Allocation
  - Where different program objects should be stored
  - All allocation within single virtual address space

Encoding Byte Values

- Byte = 8 bits
- Binary 00000000_2 to 11111111_2
- Decimal: 0_10 to 255_10
- Hexadecimal 00_16 to FF_16
  - Base 16 number representation
  - Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
  - Write FA1D37B in C as
    - 0xFA1D37B
    - 0xfa1d37b

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>A</td>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>E</td>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>F</td>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>
Machine Words

- Machine Has "Word Size"
  - Nominal size of integer-valued data, more importantly, basic unit of internal storage (registers) and computation
    - Including addresses
  - Some machines use 32 bits (4 bytes) addresses
    - Limits addresses to 4GB
    - Too small for memory-intensive applications
  - High-end systems use 64 bits (8 bytes) addresses
    - Potential address space = \(1.8 \times 10^{19}\) bytes
    - x86-64 machines support 48-bit addresses: 256 Terabytes
  - Machines support multiple data formats
    - Fractions or multiples of word size
    - Always one or multiples of bytes

Data Representations

<table>
<thead>
<tr>
<th>C Data Type</th>
<th>Typical 32-bit</th>
<th>Intel IA32</th>
<th>x86-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>long double</td>
<td>8</td>
<td>10/12</td>
<td>10/16</td>
</tr>
<tr>
<td>pointer</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Word-Oriented Memory Organization

- Addresses Specify Byte Locations
  - Address of first byte in word
  - Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)

Byte Ordering

- How should bytes within a multi-byte word be ordered in memory?
- Big Endian (Sun, PPC Mac, Internet)
  - Most significant byte has lowest address
- Little Endian (Intel x86)
  - Least significant byte has lowest address
- Example
  - Variable x has 4-byte representation 0x01234567
  - Address given by &x is 0x100

Big Endian

<table>
<thead>
<tr>
<th>0x100</th>
<th>0x101</th>
<th>0x102</th>
<th>0x103</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>23</td>
<td>45</td>
<td>67</td>
</tr>
</tbody>
</table>

Little Endian

<table>
<thead>
<tr>
<th>0x100</th>
<th>0x101</th>
<th>0x102</th>
<th>0x103</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>45</td>
<td>23</td>
<td>01</td>
</tr>
</tbody>
</table>
Reading Byte-Reversed Listings

- Disassembly
  - Text representation of binary machine code
  - Generated by program that reads the machine code
- Example Fragment

  **Address** | **Instruction Code** | **Assembly Rendition**
  --- | --- | ---
  8048365: | 5b | pop %ebx
  8048366: | 81 c3 ab 12 00 00 | add $0x12ab,%ebx
  804836c: | 83 bb 28 00 00 00 00 | cmpl $0x0,0x28(%ebx)

- Deciphering Numbers
  - Value: 0x12ab
    - Pad to 32 bits: 0x000012ab
    - Split into bytes: 00 00 12 ab
    - Reverse: ab 12 00 00

Representing Integers

- **Decimal**: 15213
  - **Binary**: 0011 1011 0110 1101
  - **Hex**: 3B 6D

  int A = 15213;

- **Decimal**: -15213
  - **Binary**: 0011 1011 0110 1101 (reversed bits)
  - **Hex**: 3B 6D

  int B = -15213;

Representing Strings

- **Strings in C**
  - Represented by array of characters
  - Each character encoded in ASCII
    - Standard 7-bit encoding of characters
    - Character "0" has code 0x30
      - Digit / has code 0x30+n
    - String should be null-terminated
      - Final character = 0
  - **Compatibility**
    - Byte ordering not an issue

char S[6] = "18243";

Bits, Bytes, and Integers

- Representing information as bits
- Bit-level manipulations
- Integers
  - Representation: unsigned and signed
  - Conversion, casting

Linux/Alpha | Sun
--- | ---
31 | 31
30 | 30
21 | 21
20 | 20
30 | 30
31 | 31
32 | 32
33 | 33
01 | 01
### Boolean Algebra
- **Algebraic representation of logic**
  - Encode “True” as 1 and “False” as 0
- **And**
  \[ A \& B = 1 \text{ when both } A=1 \text{ and } B=1 \]
- **Or**
  \[ A | B = 1 \text{ when either } A=1 \text{ or } B=1 \]
- **Not**
  \[ \neg A = 1 \text{ when } A=0 \]
- **Exclusive-Or (Xor)**
  \[ A \oplus B = 1 \text{ when either } A=1 \text{ or } B=1, \text{ but not both} \]

### Boolean Algebra on Bit Vectors
- **Operate on Bit Vectors**
- **Operations applied bitwise**

<table>
<thead>
<tr>
<th>01101001</th>
<th>01101001</th>
<th>01101001</th>
</tr>
</thead>
<tbody>
<tr>
<td>01010101</td>
<td>01010101</td>
<td>01010101</td>
</tr>
</tbody>
</table>

### Representing & Manipulating Sets
- **Representation**
  - Width w bit vector represents subsets of \{0, ..., w-1\}
  - \(a_j = 1\) if \(j \in A\)
    - 01101001 \(\{0, 3, 5, 6\}\)
    - 76543210
    - 01010101 \(\{0, 2, 4, 6\}\)
    - 76543210
- **Operations**
  - **Intersection**
    - 01000001 \(\{0, 6\}\)
  - **Union**
    - 01111101 \(\{0, 2, 3, 4, 5, 6\}\)
  - **Symmetric difference**
    - 00111100 \(\{2, 3, 4, 5\}\)
  - **Complement**
    - 10101010 \(\{1, 3, 5, 7\}\)

### Bit-Level Operations in C
- **Operations & | ~ ^ Available in C**
- **Apply to any basic data type**
  - long, int, short, char, unsigned
- **View arguments as bit vectors**
- **Arguments applied bit-wise**
- **Examples**
  - \(~0x41 = 0xBE\)
  - \(~01000001_2 = 10111110_2\)
  - \(~0x00 = 0xFF\)
  - \(~00000000_2 = 11111111_2\)
  - 0x69 & 0x55 = 0x41
  - 0x69 | 0x55 = 0x7D
  - 0x69 | 0x55 = 0x7D
  - 01101001 | 01010101 = 01111101
  - 01000001 | 01111101 = 00111100
  - 10101010 | 10101010 = 10101010
Contrast: Logic Operations in C

- Contrast to Logical Operators
  - &&, ||, !
  - View 0 as “False”
  - Anything nonzero as “True”
  - Always return 0 or 1
  - Early termination

- Examples (char data type)
  - \(0x41 \& 0x00 = 0x00\)
  - \(0x00 \& 0x41 = 0x01\)
  - \(0x00 \| 0x55 = 0x01\)
  - \(0x55 \| 0x41 = 0x01\)
  - \(p \& *p\) (avoids null pointer access)

Shift Operations

- Left Shift: \(x << y\)
  - Shift bit-vector \(x\) left \(y\) positions
  - Throw away extra bits on left
  - Fill with 0’s on right

- Right Shift: \(x >> y\)
  - Shift bit-vector \(x\) right \(y\) positions
  - Throw away extra bits on right
  - Fill with 0’s on right

  - Logical
  - Arithmetic

Bits, Bytes, and Integers

- Representing information as bits
- Bit-level manipulations
- Integers
  - Representation: unsigned and signed
  - Conversion, casting

Encoding Integers

Unsigned
\[ B2U(X) = \sum_{i=0}^{w-1} x_i \cdot 2^i \]

Two’s Complement
\[ B2T(X) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-1} x_i \cdot 2^i \]

<table>
<thead>
<tr>
<th>(x)</th>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>(15213)</td>
<td>15213</td>
<td>3F 8F</td>
<td>001111011 01110111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(y)</th>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-15213)</td>
<td>-15213</td>
<td>C4 93</td>
<td>11000100 10010011</td>
</tr>
</tbody>
</table>

- C short 2 bytes long
- Sign
  - For 2’s complement, most significant bit indicates sign
  - 0 for nonnegative, 1 for negative
- What is the bit representation of -1?
Encoding Example (Cont.)

\[
x = 15213: 00111011 01101101
\]
\[
y = -15213: 11000100 10010011
\]

Weight

<table>
<thead>
<tr>
<th>15213</th>
<th>-15213</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>512</td>
<td>1</td>
</tr>
<tr>
<td>1024</td>
<td>0</td>
</tr>
<tr>
<td>2048</td>
<td>1</td>
</tr>
<tr>
<td>4096</td>
<td>1</td>
</tr>
<tr>
<td>8192</td>
<td>1</td>
</tr>
<tr>
<td>16384</td>
<td>0</td>
</tr>
<tr>
<td>-32768</td>
<td>0</td>
</tr>
</tbody>
</table>

Sum 15213 -15213

Numeric Ranges

- **Unsigned Values**
  - UMin = 0
  - UMax = 2\(^w\) - 1

- **Two’s Complement Values**
  - Tmin = -2\(^{w-1}\)
  - Tmax = 2\(^{w-1}\) - 1

Values for \(W = 16\)

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMax</td>
<td>65535</td>
<td>FF FF 11111111 11111111</td>
</tr>
<tr>
<td>Tmax</td>
<td>32767</td>
<td>FF FF 11111111 11111111</td>
</tr>
<tr>
<td>Tmin</td>
<td>-32768</td>
<td>00 00 00000000 00000000</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>FF FF 11111111 11111111</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>00 00 00000000 00000000</td>
</tr>
</tbody>
</table>

Unsigned & Signed Numeric Values

<table>
<thead>
<tr>
<th>X</th>
<th>U2(X)</th>
<th>S2(X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>-8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
<td>-7</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
<td>-6</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
<td>-5</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
<td>-4</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
<td>-3</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
<td>-2</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>-1</td>
</tr>
</tbody>
</table>

- **Equivalence**
  - Same encodings for nonnegative values

- **Uniqueness**
  - Every bit pattern represents unique integer value
  - Each representable integer has unique bit encoding

Values for Different Word Sizes

<table>
<thead>
<tr>
<th>W</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMax</td>
<td>255</td>
<td>65,535</td>
<td>4,294,967,295</td>
<td>18,446,744,073,709,551,615</td>
</tr>
<tr>
<td>Tmax</td>
<td>127</td>
<td>32,767</td>
<td>2,147,483,647</td>
<td>9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>Tmin</td>
<td>-128</td>
<td>-32,768</td>
<td>-2,147,483,648</td>
<td>-9,223,372,036,854,775,808</td>
</tr>
</tbody>
</table>

- **Observations**
  - \(|Tmin| = Tmax + 1\)
    - Asymmetric range
  - UMax = 2 \^ w Tmax + 1
Disclaimer

These slides were adapted from the CMU course slides provided along with the textbook of “Computer Systems: A programmer’s Perspective” by Bryant and O’Hallaron. The slides are intended for the sole purpose of teaching the computer organization course at the University of Rochester.