Data Representation in Memory

CSCI 2400 / ECE 3217: Computer Architecture

Instructor:
David Ferry

Slides adapted from Bryant & O’Hallaron’s slides via Jason Fritts
Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- Representing Pointers
Byte-Oriented Memory Organization

- Modern processors: Byte-Addressable Memory
  - Conceptually a very large array of bytes
  - Each byte has a unique address
  - Processor address space determines address range:
    - 32-bit address space has $2^{32}$ unique addresses: 4GB max
      - 0x00000000 to 0xffffffff (in decimal: 0 to 4,294,967,295)
    - 64-bit address space has $2^{64}$ unique addresses: $\sim 1.8 \times 10^{19}$ bytes max
      - 0x0000000000000000 to 0xffffffffffffffff
      - Enough to give everyone on Earth about 2 Gb
  - Address space size is not the same as processor size!
    - E.g.: The original Nintendo was an 8-bit processor with a 16-bit address space
Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- Representing Pointers
Why Use Bits & Binary?

- Digital transistors operate in high and low voltage ranges
- Voltage Range dictates Binary Value on wire
  - high voltage range (e.g. 2.8V to 3.3V) is a logic 1
  - low voltage range (e.g. 0.0V to 0.5V) is a logic 0
  - voltages in between are indefinite values
- Ternary or quaternary systems have practicality problems
Bits & Bytes

- Computers use bits:
  - a “bit” is a base-2 digit
  - \{L, H\} => \{0, 1\}

- Single bit offers limited range, so grouped in bytes
  - 1 byte = 8 bits
  - a single datum may use multiple bytes

- Data representation 101:
  - Given \(N\) bits, can represent \(2^N\) unique values
    - Letters of the alphabet?
    - Colors?
Encoding Byte Values

- **Processors generally use multiples of Bytes**
  - common sizes: 1, 2, 4, 8, or 16 bytes
  - Intel data names:
    - Byte 1 byte (8 bits) $2^8 = 256$
    - Word 2 bytes (16 bits) $2^{16} = 65,536$
    - Double word 4 bytes (32 bits) $2^{32} = 4,294,967,295$
    - Quad word 8 bytes (64 bits) $2^{64} = 18,446,744,073,709,551,616$

*Unfortunately, these names are not standard so we’ll often use C data names instead (but these vary in size too... /sigh)*
# C Data Types

<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 byte</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 (addr)</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

*32-bit* vs *64-bit* key differences
Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- Representing Pointers
Encoding Byte Values

- **1 Byte = 8 bits**
  - Binary: $00000000_2$ to $11111111_2$

- **A byte value can be interpreted in many ways!**
  - depends upon how it’s used

- **For example, consider byte with:** $01010101_2$
  - as ASCII text: ‘U’
  - as integer: $85_{10}$
  - as IA32 instruction: `pushl %ebp`
  - the 86\textsuperscript{th} byte of memory in a computer
  - a medium gray pixel in a gray-scale image
  - could be interpreted MANY other ways...
Binary is Hard to Represent!

- Problem with binary – Cumbersome to use
  - e.g. approx. how big is: $1010011101010001011101011_2$?
  - Would be nice if the representation was closer to decimal: $21,930,731$

- Let’s define a larger base so that $R^1 = 2^x$
  - for equivalence, $R$ and $x$ must be integers – then 1 digit in $R$ equals $x$ bits
  - equivalence allows direct conversion between representations
  - two options closest to decimal:
    - octal: $8^1 = 2^3$ (base eight)
    - hexadecimal: $16^1 = 2^4$ (base sixteen)
Representing Binary Efficiently

- Octal or Hexadecimal?
  - Binary: \(1010011101010001011101011_2\)
  - Octal: \(123521353_8\)
  - Hexadecimal number: \(14EA2EB_{16}\)
  - Decimal: \(21930731\)

- Octal and Hex are closer in size to decimal, BUT...

- How many base-\(R\) digits per byte?
  - Octal: \(8/3 = 2.67\) octal digits per byte -- BAD
  - Hex: \(8/4 = 2\) hex digits per byte -- GOOD

**Hexadecimal wins:** 1 hex digit \(\Leftrightarrow 4\) bits
Expressing Byte Values

Juliet:
"What's in a name? That which we call a rose
By any other name would smell as sweet."

- **Common ways of expressing a byte**
  - Binary: 00000000₂ to 11111111₂
  - Decimal: 0₁₀ to 255₁₀
  - Hexadecimal: 00₁₆ to FF₁₆
    - Base-16 number representation
    - Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
    - in C/C++ programming languages, D3₁₆ written as either
      - 0xD3
      - 0xd3
# Decimal vs Binary vs Hexadecimal

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>10</td>
</tr>
<tr>
<td>17</td>
<td>10001</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>10010</td>
<td>12</td>
</tr>
</tbody>
</table>
Convert Between Binary and Hex

**Convert Hexadecimal to Binary**
- Simply replace each hex digit with its equivalent 4-bit binary sequence
- Example:

```
6 D 1 9 F 3 C_{16}
```

```
0110 1101 0001 1001 1111 0011 1100
```

```
1 6 4 6 B A C 5 3_{16}
```

**Convert Binary to Hexadecimal**
- Starting from the radix point, replace each sequence of 4 bits with the equivalent hexadecimal digit
- Example:

```
1011001000110110110110001010011
```

```
1 6 4 6 B A C 5 3_{16}
```
Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- Representing Pointers
Unsigned Integers – Binary

- Computers store Unsigned Integer numbers in Binary (base-2)
  - Binary numbers use place valuation notation, just like decimal
  - Decimal value of \( n \)-bit unsigned binary number:

\[
value_{10} = \sum_{i=0}^{n-1} a_i \cdot 2^i
\]

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^7</td>
<td>2^6</td>
<td>2^5</td>
<td>2^4</td>
<td>2^3</td>
<td>2^2</td>
<td>2^1</td>
<td>2^0</td>
</tr>
</tbody>
</table>

\[
value_{10} = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0
\]

\[
= 2^6 + 2^5 + 2^4 + 2^2 + 2^0
\]

\[
= 64 + 32 + 16 + 4 + 1 \quad = 117_{10}
\]
Unsigned Integers – Base-R

**Convert Base-R to Decimal**

- Place value notation can similarly determine decimal value of any base, \( R \)
- Decimal value of \( n \)-digit base \( r \) number:

\[
\text{value}_{10} = \sum_{i=0}^{n-1} a_i \times r^i
\]

- Example: \( 317_{8} = ?_{10} \)

\[
\text{value}_{10} = 3 \times 8^2 + 1 \times 8^1 + 7 \times 8^0
\]
\[
= 3 \times 64 + 1 \times 8 + 7 \times 1
\]
\[
= 192 + 8 + 7 = 207_{10}
\]
Unsigned Integers – Hexadecimal

- Commonly used for converting hexadecimal numbers
  - Hexadecimal number is an “equivalent” representation to binary, so often need to determine decimal value of a hex number
  - Decimal value for \( n \)-digit hexadecimal (base 16) number:

\[
value_{10} = \sum_{i=0}^{n-1} a_i \times 16^i
\]

- Example: \(9E4_{16} = ?_{10}\)

\[
value_{10} = 9 \times 16^2 + 14 \times 16^1 + 4 \times 16^0
\]
\[
= 9 \times 256 + 14 \times 16 + 4 \times 1
\]
\[
= 2304 + 224 + 4 = 2532_{10}
\]
Unsigned Integers – Convert Decimal to Base-R

- Also need to convert decimal numbers to desired base
- Algorithm for converting unsigned Decimal to Base-R
  a) Assign decimal number to \( NUM \)
  b) Divide \( NUM \) by \( R \)
     - Save remainder \( REM \) as next least significant digit
     - Assign quotient \( Q \) as new \( NUM \)
  c) Repeat step b) until quotient \( Q \) is zero
- Example: \( 83_{10} = ?_7 \)

<table>
<thead>
<tr>
<th>NUM</th>
<th>R</th>
<th>Q</th>
<th>REM</th>
</tr>
</thead>
<tbody>
<tr>
<td>83</td>
<td>7</td>
<td>11</td>
<td>r6</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>1</td>
<td>r4</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>0</td>
<td>r1</td>
</tr>
</tbody>
</table>

\[ = 146_7 \]

least significant digit
most significant digit
## Unsigned Integers – Convert Decimal to Binary

### Example with Unsigned Binary:

$$52_{10} = ?_2$$

<table>
<thead>
<tr>
<th>NUM</th>
<th>R</th>
<th>Q</th>
<th>REM</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>2</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$$= 110100_2$$
Unsigned Integers – Convert Decimal to Hexadecimal

Example with Unsigned Hexadecimal: \(437_{10} = ?_{16}\)

<table>
<thead>
<tr>
<th>NUM</th>
<th>R</th>
<th>Q</th>
<th>REM</th>
</tr>
</thead>
<tbody>
<tr>
<td>437</td>
<td>/ 16 → 27</td>
<td>r 5</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>/ 16 → 1</td>
<td>r 11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>/ 16 → 0</td>
<td>r 1</td>
<td></td>
</tr>
</tbody>
</table>

\[= 1B5_{16}\]
Unsigned Integers – Ranges

- **Range of Unsigned binary numbers based on number of bits**
  - Given representation with \( n \) bits, min value is always sequence
    - \( 0....0000 = 0 \)
  - Given representation with \( n \) bits, max value is always sequence
    - \( 1....1111 = 2^n - 1 \)
  - So, ranges are:
    - unsigned char: \( 0 \rightarrow 255 \quad (2^8 - 1) \)
    - unsigned short: \( 0 \rightarrow 65,535 \quad (2^{16} - 1) \)
    - unsigned int: \( 0 \rightarrow 4,294,967,295 \quad (2^{32} - 1) \)

\[
\begin{array}{cccccc}
1 & 1 & \ldots & 1 & 1 & 1 \\
2^{n-1} & 2^{n-2} & 2^3 & 2^2 & 2^1 & 2^0 \\
\end{array}
\]

\[
= \sum_{i=0}^{n-1} 2^i = 2^n - 1
\]
Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- Representing Pointers
Signed Integers – Binary

- Signed Binary Integers converts half of range as negative
- Signed representation identical, except for most significant bit
  - For signed binary, most significant bit indicates sign
    - 0 for nonnegative
    - 1 for negative
  - Must know number of bits for signed representation

Unsigned Integer representation:

<table>
<thead>
<tr>
<th>2^7</th>
<th>2^6</th>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
</table>

Signed Integer representation:

<table>
<thead>
<tr>
<th>-2^7</th>
<th>2^6</th>
<th>2^5</th>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
</table>

Place value of most significant bit is negative for signed binary
Signed Integers – Binary

- Decimal value of \( n \)-bit signed binary number:

\[
value_{10} = -a_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} a_i \times 2^i
\]

- Positive (in-range) numbers have same representation:

**Unsigned Integer representation:**

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
\end{array}
\]

\[= 105_{10}\]

**Signed Integer representation:**

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
-2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
\end{array}
\]

\[= 105_{10}\]
Signed Integers – Binary

- Only when most significant bit set does value change
- Difference between unsigned and signed integer values is $2^N$

**Unsigned Integer representation:**

<table>
<thead>
<tr>
<th>$2^7$</th>
<th>$2^6$</th>
<th>$2^5$</th>
<th>$2^4$</th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$= 105 + 128_{10}$
$= 233_{10}$

**Signed Integer representation:**

<table>
<thead>
<tr>
<th>$-2^7$</th>
<th>$2^6$</th>
<th>$2^5$</th>
<th>$2^4$</th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

$= 105 - 128_{10}$
$= -23_{10}$
Quick Check:

For an 8-bit representation:
- What bit pattern has the minimum value?
- What bit pattern has the maximum value?
- What bit pattern represents 0?
- What bit pattern represents -1?
Signed Integers – Ranges

Range of Signed binary numbers:

- Given representation with $n$ bits, min value is always sequence
  - $\text{100...0000} = -2^{n-1}$
- Given representation with $n$ bits, max value is always sequence
  - $\text{011...1111} = 2^{n-1} - 1$
- So, ranges are:

<table>
<thead>
<tr>
<th>C data type</th>
<th># bits</th>
<th>Unsigned range</th>
<th>Signed range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>8</td>
<td>$0 \rightarrow 255$</td>
<td>$-128 \rightarrow 127$</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>$0 \rightarrow 65,535$</td>
<td>$-32,768 \rightarrow 32,767$</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>$0 \rightarrow 4,294,967,295$</td>
<td>$-2,147,483,648 \rightarrow 2,147,483,647$</td>
</tr>
</tbody>
</table>
Signed Integers – Convert to/from Decimal

- Convert Signed Binary Integer to Decimal
  - Easy – just use place value notation
    - two examples given on last two slides

- Convert Decimal to Signed Binary Integer
  - MUST know number of bits in signed representation
  - Algorithm:
    a) Convert magnitude (abs val) of decimal number to unsigned binary
    b) Decimal number originally negative?
      - If positive, conversion is done
      - If negative, perform negation on answer from part a)
        » zero extend answer from a) to N bits (size of signed repr)
        » negate: flip bits and add 1
Signed Integers – Convert Decimal to Base-R

Example: $-37_{10} = \text{?}_{8}$ – bit signed

A) $|-37_{10}| = \text{?}_{2}$

<table>
<thead>
<tr>
<th>NUM</th>
<th>R</th>
<th>Q</th>
<th>REM</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>/</td>
<td>2</td>
<td>→</td>
</tr>
<tr>
<td>18</td>
<td>/</td>
<td>2</td>
<td>→</td>
</tr>
<tr>
<td>9</td>
<td>/</td>
<td>2</td>
<td>→</td>
</tr>
<tr>
<td>4</td>
<td>/</td>
<td>2</td>
<td>→</td>
</tr>
<tr>
<td>2</td>
<td>/</td>
<td>2</td>
<td>→</td>
</tr>
<tr>
<td>1</td>
<td>/</td>
<td>2</td>
<td>→</td>
</tr>
</tbody>
</table>

$= 1001011_2$

least significant bit

most significant bit
Signed Integers – Convert Decimal to Base-R

Example: \(-37_{10} = ?\) 8-bit signed

- B) -37\(_{10}\) was negative, so perform *negation*
  - zero extend 100101 to 8 bits
    
    \[
    100101_{2} \rightarrow 00100101_{2}
    \]

  - negation
    - flip bits: \(00100101_{2}\)
      
      \[
      \downarrow
      \]
      
      \[
      11011010_{2}
      \]
    - add 1: \(+ \quad 1_{2}\)
      
      \[
      \downarrow
      \]
      
      \[
      11011011_{2}
      \]

Can validate answer using place value notation
Quick check:

For an 8-bit representation:

- Convert $67_{10}$ into a signed integer
Signed Integers – Convert Decimal to Base-R

Example: \(67_{10} = ? \ 8\text{-bit signed}\)

A) \(|67_{10}| = ?_2\)

\[
\begin{array}{cccc}
\text{NUM} & \text{R} & \text{Q} & \text{REM} \\
67 & / & 2 & \rightarrow & 33 & r & 1 \\
33 & / & 2 & \rightarrow & 16 & r & 1 \\
16 & / & 2 & \rightarrow & 8 & r & 0 \\
8 & / & 2 & \rightarrow & 4 & r & 0 \\
4 & / & 2 & \rightarrow & 2 & r & 0 \\
2 & / & 2 & \rightarrow & 1 & r & 0 \\
1 & / & 2 & \rightarrow & 0 & r & 1 \\
\end{array}
\]

\(= 1000011_2\)

least significant bit

most significant bit
Signed Integers – Convert Decimal to Base-R

Example: \(67_{10} = ?\) _8-bit signed_

- B) \(67_{10}\) was positive, so done

\[= 1000011_2\]

*Can validate answer using place value notation*
Quick check:

For an 8-bit representation:

- Convert $-100_{10}$ into a signed integer
Signed Integers – Convert Decimal to Base-R

Example: \(-100_{10} = ?_{8\text{-bit signed}}\)

- **A)** \(\mid -100_{10} \mid = ?_2\)

<table>
<thead>
<tr>
<th>NUM</th>
<th>R</th>
<th>Q</th>
<th>REM</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>/</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td>50</td>
<td>/</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>/</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>/</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>/</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>/</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>/</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

\[= 1100100_2\]
Example: \(-100_{10} = ? \) \(8\)-bit signed

- \(-100_{10}\) was negative, so perform \textit{negation}
  - zero extend 100101 to 8 bits

\[
\begin{array}{c}
1100100_2 \\
\rightarrow \\
01100100_2
\end{array}
\]

- \textit{negation}
  - flip bits:
    \[
    \begin{array}{c}
    01100100_2 \\
    \downarrow \\
    10011011_2
    \end{array}
    \]
  - add 1:
    \[
    \begin{array}{c}
    + \\
    1_2 \\
    \hline
    10011100_2
    \end{array}
    \]

\[= 10011100_2\]

\textit{Can validate answer using place value notation}
Signed Integers – Convert Decimal to Base-R

- Be careful of range!

- Example: \(-183_{10} = ?\) _8-bit signed_

  - A) \(|-183_{10}| = ?_{2} = 10110111_{2}\)

  - B) -183_{10} was negative, so perform _negation_
    - zero extend 10110111 to 8 bits // already done
    - negation
      - flip bits: \(10110111_{2}\)
      - add 1: \(+1_{2}\)
        \[
        \begin{array}{c}
        01001000_{2} \\
        01001001_{2} = 73_{10}
        \end{array}
        
        _not -183_{10}... WRONG!

-183_{10} is not in valid range for 8-bit signed
Representation of Signed Integers

- **Multiple possible ways:**
  - Sign magnitude
  - Ones’ Complement
  - Two’s Complement (what has been presented)

- **Two’s Complement greatly simplifies addition & subtraction in hardware**
  - We’ll see why when we cover operations
  - Generally the only method still used
Representation of Signed Integers

Why the name Two’s Complement?

- For a $w$-bit signed representation, we represent $-x$ as $2^w - x$
- E.g.: consider the 8-bit representation of $-37_{10}$

\[
2^8 = 256_{10}
\]
\[
2^8 - 37_{10} = 219_{10}
\]
\[
219_{10} = 11011011_2 \text{ (unsigned)}
\]
\[
-37_{10} = 11011011_2 \text{ (signed)}
\]
Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- Representing Pointers
Representing Strings

Strings in C

- Represented by array of characters
- Each character encoded in **ASCII format**
  - Standard 7-bit encoding of character set
  - Character “0” has code 0x30
- String should be null-terminated
  - Final character = 0
- ASCII characters organized such that:
  - Numeric characters sequentially increase from 0x30
    - Digit $i$ has code 0x30+$i$
  - Alphabetic characters sequentially increase in order
    - Uppercase chars ‘A’ to ‘Z’ are 0x41 to 0x5A
    - Lowercase chars ‘a’ to ‘z’ are 0x61 to 0x7A
  - Control characters, like <RET>, <TAB>, <BKSPC>, are 0x00 to 0x1A

```c
char S[6] = "18243";
```
Representing Strings

- **Limitations of ASCII**
  - 7-bit encoding limits set of characters to $2^7 = 128$
  - 8-bit extended ASCII exists, but still only $2^8 = 256$ chars
  - Unable to represent most other languages in ASCII

- **Answer: Unicode**
  - first 128 characters are ASCII
    - i.e. 2-byte Unicode for ‘4’: 0x34 -> 0x0034
    - i.e. 4-byte Unicode for ‘T’: 0x54 -> 0x00000054
  - UTF-8: 1-byte version // commonly used
  - UTF-16: 2-byte version // commonly used
    - allows $2^{16} = 65,536$ unique chars
  - UTF-32: 4-byte version
    - allows $2^{32} = \sim 4$ billion unique characters
  - Unicode used in many more recent languages, like Java and Python

### UTF-16 on Intel

```
0x31  '1'
0x00
0x38  '8'
0x00
0x32  '2'
0x00
0x34  '4'
0x00
0x33  '3'
0x00
0x00  null term
```
String Representation Links

- **ASCII**

- **Unicode**
Quick Check:

- Convert the following strings to ASCII-

  char school[4] = “SLU”;

  char name[6] = “Frank”;
Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- Representing Pointers
What is a Pointer?

Recall:

- **Memory is a contiguous array of individual bytes**
  - Consider a machine with 16-bit addresses
What is a Pointer?

Recall:

- Memory is a contiguous array of individual bytes
  - Consider a machine with 16-bit addresses and 32-bit data

```
unsigned X = 15398; //0x00003C26
```
Points to a location in memory

Suppose:

unsigned X = 15398; //0x00003C26

unsigned *ptr = &X; //0xA244

A pointer is a variable that holds the address of another variable

Different compilers and machines assign different locations to objects
Endianness

- Recall that memory is byte-addressable
  - Four bytes in a 32-bit integer, which order are they stored with?
  
  Two ways to store: **unsigned X = 15398; //0x00003C26**

- **Little Endian**
  - Least significant bits stored first in memory
  
  - | 26 | 0x0000 |
  - | 3C | 0x0001 |
  - | 00 | 0x0002 |
  - | 00 | 0x0003 |
  - | 00 | 0x0004 |

- **Big Endian**
  - Most significant bits stored first in memory
  
  - | 00 | 0x0000 |
  - | 00 | 0x0001 |
  - | 26 | 0x0002 |
  - | 3C | 0x0003 |
  - | 00 | 0x0004 |
Quick Check

- Consider the string:
  ```
  char S[6] = "HELLO";
  ```

- What is S[0]?
- What is &S[0]?
- What is S[3]?
- What is &S[3]?

<table>
<thead>
<tr>
<th>Character</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'H'</td>
<td>0xACED</td>
</tr>
<tr>
<td>'E'</td>
<td>0xACEE</td>
</tr>
<tr>
<td>'L'</td>
<td>0xACEF</td>
</tr>
<tr>
<td>'L'</td>
<td>0xACF0</td>
</tr>
<tr>
<td>'O'</td>
<td>0xACF1</td>
</tr>
<tr>
<td>null term</td>
<td>0xACF2</td>
</tr>
</tbody>
</table>