Lecture 2: Number System

Today's Topics

- Review binary and hexadecimal number representation
- **Convert** directly from one base to another base
- Review addition and subtraction in binary representation
- Determine overflow in unsigned and signed binary addition and subtraction.
### Why do we need other bases

- **Human: decimal number system**
  - Radix-10 or base-10
  - Base-10 means that a digit can have one of ten possible values, 0 through 9.

- **Computer: binary number system**
  - Radix-2 or base-2
  - Each digit can have one of two values, 0 or 1

- **Compromise: hexadecimal**
  - Long strings of 1s and 0s are cumbersome to use
  - Represent binary numbers using hexadecimal.
  - Radix-16 or base-16
  - This is only a convenience for humans not computers.

- All of these number systems are positional

### Unsigned Decimal

- Numbers are represented using the digits 0, 1, 2, …, 9.
- Multi-digit numbers are interpreted as in the following example

\[
793_{10} = 7 \times 10^2 + 9 \times 10^1 + 3 \times 10^0
\]

- We can get a general form of this
  - \( A \times (\text{radix})^2 + B \times (\text{radix})^1 + C \times (\text{radix})^0 \)

OK, now I see why we say that number systems are positional.
Unsigned Binary

- Numbers are represented using the digits 0 and 1.
- Multi-digit numbers are interpreted as in the following example.
  - $10111_2$
    - $= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
    - $= 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1$
- Bit: Each digit is called a bit (Binary Digit) in binary.
- Important! You must write all bits including leading 0s, when we say $n$-bit binary.
  - Ex: 000101112 (8-bit binary)

Unsigned Hexadecimal

- Numbers are represented using the digits 0, 1, 2, ..., 9, A, B, C, D, E, F where the letters represent values: A=10, B=11, C=12, D=13, E=14, and F=15.
- Multi-digit numbers are interpreted as in the following example.
  - $76CA_{16}$
    - $= 7 \times 16^3 + 6 \times 16^2 + C(=12) \times 16^1 + A(=10) \times 16^0$
    - $= 7 \times 4096 + 6 \times 256 + 12 \times 16 + 10$
    - $= 30,410_{10}$

The same rule is applied to here!
Notes on Bases

• Subscript is mandatory at least for a while.
   We use all three number bases.
   When a number is written out of context, you should include the correct subscript.

• Pronunciation
   Binary and hexadecimal numbers are spoken by naming the digits followed by “binary” or “hexadecimal.”
    • e.g., $1000_{16}$ is pronounced “one zero zero zero hexadecimal.”
    • c.f., “one-thousand hexadecimal” refers the hexadecimal number corresponding to $1000_{10}$ (so, $3E8_{16}$)

Ranges of Unsigned Number Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Lowest</th>
<th>Highest</th>
<th>Number of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bit binary</td>
<td>$0000_2$</td>
<td>$1111_2$</td>
<td>$16_{10}$</td>
</tr>
<tr>
<td>(1-digit hex)</td>
<td>$0_{10}$</td>
<td>$15_{10}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0_{16}$</td>
<td>$F_{16}$</td>
<td></td>
</tr>
<tr>
<td>8-bit binary</td>
<td>$0000 0000_2$</td>
<td>$1111 1111_2$</td>
<td>$256_{10}$</td>
</tr>
<tr>
<td>(1 byte)</td>
<td>$0_{10}$</td>
<td>$255_{10}$</td>
<td></td>
</tr>
<tr>
<td>(2-digit hex)</td>
<td>$0_{16}$</td>
<td>$FF_{16}$</td>
<td></td>
</tr>
<tr>
<td>16-bit binary</td>
<td>$0000 0000 0000_2$</td>
<td>$1111 1111 1111_2$</td>
<td>$65536_{10}$</td>
</tr>
<tr>
<td>(2 bytes)</td>
<td>$0_{10}$</td>
<td>$65535_{10}$</td>
<td></td>
</tr>
<tr>
<td>(1-digit hex)</td>
<td>$0_{16}$</td>
<td>$FFFF_{16}$</td>
<td></td>
</tr>
<tr>
<td>n-bit binary</td>
<td>$0_{10}$</td>
<td>$2^{n-1}_{10}$</td>
<td>$2^n$</td>
</tr>
</tbody>
</table>
2’s Complement Binary Numbers

Negative Number Representation

- Most microprocessors use 2’s complement numbers to represent number systems with positive and negative values.
- Hardware performs addition and subtraction on binary values the same way whether they are unsigned or 2’s complement systems.
- In signed systems, MSB(Most Significant Bit) has a weight of \(-2^{(n-1)}\).

<table>
<thead>
<tr>
<th>Bin</th>
<th>Signed</th>
<th>Unsigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0000 0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0000 0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0111 1110</td>
<td>126</td>
<td>126</td>
</tr>
<tr>
<td>0111 1111</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>1000 0000</td>
<td>-128</td>
<td>128</td>
</tr>
<tr>
<td>1000 0001</td>
<td>-127</td>
<td>129</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1111 1110</td>
<td>-2</td>
<td>254</td>
</tr>
<tr>
<td>1111 1111</td>
<td>-1</td>
<td>255</td>
</tr>
</tbody>
</table>
2’s Complement Binary Numbers

- We will use ‘2C’ subscript to indicate a 2’s complement number.

- Examples
  - Convert 100110102C in decimal
    \[= -2^{7} \times 1 + 2^{6} + 2^{5} + 2^{4} + 2^{1} = -102_{10}\]
  - Convert 110112C in decimal
    \[= -2^{5} \times 1 + 2^{4} + 2^{1} + 2^{0} = -5_{10}\]
  - Convert 010112C in decimal
    \[= -2^{4} \times 0 + 2^{3} + 2^{1} + 2^{0} = 11_{10}\]

A Group of Bits are A Group of Bits.

- To microprocessors, a group of bits are simply a group of bits.

- Humans interpret the group as an unsigned, signed values or also as just a group of bits.
Ranges of Signed Number Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Lowest</th>
<th>Highest</th>
<th>Number of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bit binary</td>
<td>1000₂ –8₁₀</td>
<td>0111₂ 7₁₀</td>
<td>16₁₀</td>
</tr>
<tr>
<td>8-bit binary (1 byte)</td>
<td>1000 0000₂ –128₁₀</td>
<td>0111 1111₂ 127₁₀</td>
<td>256₁₀</td>
</tr>
<tr>
<td>16-bit binary (2 bytes)</td>
<td>1000 0000 0000₂ –32768₁₀</td>
<td>0111 1111 1111₂ 32767₁₀</td>
<td>65536₁₀</td>
</tr>
<tr>
<td>n-bit binary</td>
<td>–2ⁿ⁻¹₁₀</td>
<td>2ⁿ⁻¹⁻¹₁₀</td>
<td>2ⁿ</td>
</tr>
</tbody>
</table>

Sign Bit

- The leftmost bit (MSB) is a sign bit.
- We can tell the number is negative or positive by simply inspecting the leftmost bit.
- If MSB is 1, the number is negative. Otherwise, positive.
- Why?
  - The leftmost column has a negative weight, and the magnitude of that weight is larger than the weights of all the positive columns added altogether, any number with a 1 in the leftmost column will be negative.
Negating a 2’s Complement Number

- Negate a number:
  - Generate a number with the same magnitude but with the opposite sign.
  - Ex: $25 \leftrightarrow -25$

- Two steps in binary systems
  - 1. Perform the 1’s complement (flip all the bits)
  - Ex: Negate $00101001_2$ ($41_{10}$)
    - 1. Flip all the bits: $11010110_2$
    - 2. Add 1: $11010110 + 1 \rightarrow 11010111_2$ ($-41_{10}$)

Converting Decimal to Binary

- Ex: Convert $53_{10}$ to 8-bit unsigned binary.

Don’t forget to add two zeros before the numbers to make 8-bit binary!
Converting Decimal to Binary

- Ex: Convert $172_{10}$ to 2-digit hexadecimal.

$$
\begin{array}{c}
16 \mid 172 \\
\underline{10} \ 12 (= C) \uparrow \\
= A \\
\hline
\text{AC}_{16}
\end{array}
$$

Converting a Negative Value

- Converting a negative value
  - 1. convert the magnitude to correct number of bits
  - 2. negate the result.

- Ex: $-127_{10}$ to 8-bit signed binary

$$
\begin{array}{c}
-127_{10} \\
2 \mid 127 \uparrow \\
2 \mid 63 \ 1 \downarrow \\
2 \mid 31 \ 1 \downarrow \\
2 \mid 15 \ 1 \downarrow \\
2 \mid 7 \ 1 \downarrow \\
2 \mid 3 \ 1 \downarrow \\
2 \mid 1 \ 1 \downarrow \\
\hline
\text{1111 1111 (negate)}
\end{array}
$$
Binary to Hexadecimal

• This conversion is the reason that hexadecimal is used.

• We can group 4 bits since four bits can represent 16 (\(=2^4\)) different values.
  • Examples:
    - \(1001\ 0101\ 1110\ _2 = 9\ 5\ E\ _{16}\)
    - \(0110\ 1010\ 1011\ _2 = 6\ A\ B\ _{16}\)

• If a binary number is not multiple of 4 bits, padding the number with zeros regardless of the sign of the number.
  • Examples:
    - \(1\ 0101\ 1110\ _{2C} = 0001\ 0101\ 1110\ _2 = 1\ 5\ E\ _{16}\)
    - \(1\ 1011\ _{2C} = 0001\ 1011\ _2 = 1\ B\ _{16}\)

Hexadecimal to Binary

• Hexadecimal is not interpreted as signed or unsigned.

• Converting hexadecimal to binary
  • Examples
    - \(B\ E\ F\ A\ _{16} = 1011\ 1110\ 1111\ 1010\ _2\)
    - \(7\ 3\ F\ C\ _{16} = 0111\ 0011\ 1111\ 1100\ _2\)

• We can specify a binary system with any number of bits.
  • Examples
    - \(0\ 7\ B\ _{16}\) to 9-bit signed = \(0\ 0111\ 1011\ _{2C}\)
    - \(1\ F\ _{16}\) to 5-bit unsigned = \(1\ 1111\ _2\)
Binary Arithmetic & Overflow

• Overflow occurs when two numbers are added or subtracted and the correct result is a number that is outside of the range of allowable numbers.
  ▪ Example:
    • 254 + 10 = 264 (<255); overflow in unsigned 8-bit.
    • \(-100 - 30 = -130(<-128);\) overflow in signed 8-bit.

Overflow detection

• For unsigned:
  ▪ It is simple. A carry occurs, so does overflow!
  ▪ A carry (or borrow) out of the most significant column indicates that overflow occurred.

• For signed:
  ▪ A carry does not mean overflow.
  ▪ Ex: in 4-bit binary system
    • \(-2 + 3 = 1 (1110 + 0011 = 0001 \text{ with carry } = 1 \text{ (carry ignored)})\)
    • \(-4 - 3 = -7 (1100 + 1101 = 1001 \text{ with carry } = -7 \text{ (carry ignored)})\)
    • \(6 + 3 = 9 \text{ (overflow), } 0110 + 0011 = 1001 (= -7), \text{ incorrect.}\)
    • \(-7 - 3 = -10 \text{ (underflow), } 1001 + 1101 = 0110 (= 6), \text{ incorrect.}\)
Binary Arithmetic & Overflow

Overflow detection

- For signed:
  - It is hard to detect overflow(underflow).
  - Addition:
    - Adding same sign numbers and the result with different sign → overflow.
    - No overflow in case if the two numbers have different sign.
  - Subtraction:
    - Minuend – subtrahend = difference
    - If sign(difference) == sign(minuend) → no overflow
    - If sign(difference) == sign(subtrahend) → overflow.

Examples

- For signed: examples
  - Addition:
    - 01101011 + 01011010 = 11000101.
    - Unsigned (no overflow), signed (overflow, because the sign of the result is different from numbers being added)
  - Subtraction:
    - 01101011 – 11011011 = 10010010.
    - Unsigned(overflow), signed(overflow, because the sign of the result is same as that of the subtrahend)
Extending Binary Numbers

- The binary numbers must have the same number of bits when performing arithmetic operations.
- It is necessary to extend the shorter number so that it has the same number of bits as the longer number.
- For unsigned:
  - Always extend by adding zeros.
- For signed:
  - Always extend by repeating sign bit.

Examples

Extend the binary numbers below to 16 bits.
- 0110 1111₂ → 0000 0000 0110 1111₂
- 1 0010 1101₂ → 0000 0001 0010 1101₂
- 0 1110₂ → 0000 0000 0000 1110₂
- 1001 1001₂ → 1111 1111 1001 1001₂
Truncating Binary Numbers

- It is **not possible** to truncate binary numbers if it yields a shorter number that **does not represent the same value** as the original number.

- **Unsigned:**
  - All bits discarded must be 0s.

- **Signed:**
  - All bits discarded must be same as the new sign bit of the shorter number.

Examples

- Truncate 16-bit values to 8 bits
  - \text{0000 0000 1011 0111}_2 \rightarrow \text{1011 0111}_2
  - \text{1111 1111 1011 0111}_2 \rightarrow \text{not possible}
  - \text{0000 0000 1011 0111}_2C \rightarrow \text{not possible}
  - \text{0000 0000 0011 0111}_2C \rightarrow \text{0011 0111}_2C
  - \text{1111 1110 1011 0111}_2C \rightarrow \text{not possible}
  - \text{1111 1111 1011 0111}_2C \rightarrow \text{1011 0111}_2C
Questions?

Wrap-up
What we’ve learned

• Binary and hexadecimal number representation
• Convert directly from one base to another base
• Addition and subtraction in binary representation
• Determine overflow in unsigned and signed binary addition and subtraction
What to Come

- Lab sessions start from Tuesday.
- Introduction to HCS12