Microcomputers I – CE 320

Mohammad Ghamari, Ph.D.
Electrical and Computer Engineering
Kettering University

Announcements

- Exercise 1 is uploaded.
  - It is about number systems
  - If you don’t feel comfortable with number systems, don’t miss this exercise.
- Please do the exercise. Finish the exercise in right time is very important to come up with the course.
- Today, Lecture 2 will be provided.

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

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

- <u>Computer</u>: 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:

Example: 793_{10}

  - $= 7 \times 100 + 9 \times 10 + 3$
  - $= 7 \times 10^2 + 9 \times 10^1 + 3 \times 10^0$

- We can get a general form of this

  - ABC_{radix}

  - $A \times (radix)^2 + B \times (radix)^1 + C \times (radix)^0$

Unsigned Binary

- Numbers are represented using the digits 0 and 1.

- Multi-digit numbers are interpreted as in the following example:

Example: 10111_2 (5-bit binary)

  - $= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
  - $= 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1$

- Bit: Each digit is called a **bit**(**B**inary Dig**it**) in binary

- Important! You must write all bits including leading 0s, when we say *n*-bit binary.
  - Ex: 00010111_2 (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:

Example: $76CA_{16}$

  - $= 7 \times 16^3 + 6 \times 16^2 + C(=12) \times 16^1 + A(=10) \times 16^0$
  - $= 7 \times 4096 + 6 \times 256 + 12 \times 16 + 10$
  - $= 30{,}410_{10}$

Notes on Bases

- Subscript is mandatory *at least for a while.*
  - We use it for 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 $\mathbf{1000_{10}}$**. (so, $3E8_{16}$)

Ranges of Unsigned Number Systems

System	Lowest	Highest	Number of values
4-bit binary (1-digit hex)	0000_2 0_{10} 0_{16}	1111_2 15_{10} F_{16}	16_{10}
8-bit binary (1 byte) (2-digit hex)	$0000\ 0000_2$ 0_{10} 0_{16}	$1111\ 1111_2$ 255_{10} FF_{16}	256_{10}
16-bit binary (2 bytes) (1-digit hex)	$0000\ 0000\ 0000\ 0000_2$ 0_{10} 0_{16}	$1111\ 1111\ 1111\ 1111_2$ 65535_{10} $FFFF_{16}$	65536_{10}
n-bit binary	0_{10}	$2^n\text{-}1_{10}$	2^n

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 $-2^{(n-1)}$.

2’s Complement Binary Numbers

Bin	Signed	Unsigned
0000 0000	0	0
0000 0001	1	1
0000 0010	2	2
…	…	…
0111 1110	126	126
0111 1111	127	127
1000 0000	-128	128
1000 0001	-127	129
…	…	…
1111 1110	-2	254
1111 1111	-1	255

2’s Complement Binary Numbers

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

- Examples:

  - Convert 10011010_{2c} in decimal

  - Convert 11011_{2c} in decimal

  - Convert 01011_{2c} in decimal

2’s Complement Binary Numbers

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

- Examples:

  - Convert 10011010_{2c} in decimal
    - $= -2^{(8-1)} \times 1 + 2^4 + 2^3 + 2^1 = -102_{10}$
  - Convert 11011_{2c} in decimal
    - $= -2^{(5-1)} \times 1 + 2^3 + 2^1 + 2^0 = -5_{10}$
  - Convert 01011_{2c} in decimal
    - $= -2^{(5-1)} \times 0 + 2^3 + 2^1 + 2^0 = 11_{10}$

A Group of Bits are A Group of Bits.

- To microprocessors, a group of bits <u>are simply a group of bits</u>.

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

Ranges of Signed Number Systems

System	Lowest	Highest	Number of values
4-bit binary	1000_2 -8_{10}	0111_2 7_{10}	16_{10}
8-bit binary (1 byte)	$1000\ 0000_2$ -128_{10}	$0111\ 1111_2$ 127_{10}	256_{10}
16-bit binary (2 bytes)	$1000\ 0000\ 0000\ 0000_2$ -32768_{10}	$0111\ 1111\ 1111\ 1111_2$ 32767_{10}	65536_{10}
n-bit binary	$-2^{(n-1)}{}_{10}$	$2^{(n-1)}-1_{10}$	2^n

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, **<u>any number with a 1 in the leftmost column will be negative</u>**.

Negating a 2's Complement Number

- Negate a number:
  - Generate a number with the same magnitude but with the opposite sign.
  - Ex: 25 ←→ -25

- Two steps in binary systems
  - 1- Perform the 1's complement (flip all the bits)
  - 2- Add 1.

  - Ex: Negate 00101001_{2c} (41_{10})
    - 1. Flip all the bits: 11010110
    - 2. Add 1: 11010110 + 1 → 11010111_{2c} (-41_{10})

Converting Decimal to Binary

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

Converting Decimal to Binary

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

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

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 4bits, padding the number with zeros regardless of the sign of the number.
  - Examples:
    - $1\ 0101\ 1110_{2C} = \mathbf{000}1\ 0101\ 1110_2 = 1\ 5\ E\ _{16}$
    - $1\ 1011_{2C} = \mathbf{000}1\ 1011_2 = 1\ B\ _{16}$

Hexadecimal to Binary

- Hexadecimal is not interpreted as signed or unsigned.

- Converting hexadecimal to binary
  - Examples
    - $BEFA_{16}$ = $1011\ 1110\ 1111\ 1010_2$
    - $73FC_{16}$ = $0111\ 0011\ 1111\ 1100_2$

- We can specify a binary system with any number of bits.
  - Examples
    - $07B_{16}$ to 9-bit signed = $0\ 0111\ 1011_{2C}$
    - $1F_{16}$ to 5-bit unsigned = $1\ 1111_2$

Binary Arithmetic & Overflow

0 + 0 = 0, carry = 0	0 – 0 = 0, borrow = 0
1 + 0 = 1, carry = 0	1 – 0 = 1, borrow = 0
0 + 1 = 1, carry = 0	0 – 1 = 1, borrow = 1
1 + 1 = 0, carry = 1	1 – 1 = 0, borrow = 0

- 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.

Binary Arithmetic & Overflow
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 (**1**110 + **0**011 = 0001 with carry = 1 (carry ignored)
    - -4 – 3 = -7 (**1**100 + **1**101 = **1**001 with carry = -7 (carry ignored)
    - 6 +3 = 9 (overflow), **0**110 + **0**011 = **1**001 (=-7), **incorrect**.
    - -7 – 3 = -10 (underflow), (**1**001 + **1**101 = **0**110 (=6), **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.

Binary Arithmetic & Overflow
Examples

- For signed: examples
  - Addition:
    - 01101011 + 01011010 = **1**1000101.
    - Unsigned (**no overflow**), signed (**overflow**, because the sign of the result is different from numbers being added)

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.

Extending Binary Numbers
Examples

- Extend the binary numbers below to 16 bits.
  - $0110\ 1111_2$ → **0000 0000** $0110\ 1111_2$
  - $1\ 0010\ 1101_2$ → **0000 000**$1\ 0010\ 1101_2$
  - $0\ 1110_{2C}$ → **0000 0000 000**$0\ 1110_2$
  - $1001\ 1001_{2C}$ → **1111 1111** $1001\ 1001_2$

Truncating Binary Numbers

- It is **<u>not possible</u>** to truncate binary numbers if it yields a shorter number that <u>does not represent the same value</u> 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.

Truncating Binary Numbers
Examples

- Truncate 16-bit values to 8 bits
  - $0000\ 0000\ 1011\ 0111_2$ → $1011\ 0111_2$
  - $1111\ 1111\ 1011\ 0111_2$ → not possible
  - $0000\ 0000\ 1011\ 0111_{2C}$ → not possible
  - $0000\ 0000\ 0011\ 0111_{2C}$ → $0011\ 0111_{2C}$
  - $1111\ 1110\ 1011\ 0111_{2C}$ → not possible
  - $1111\ 1111\ 1011\ 0111_{2C}$ → $1011\ 0111_{2C}$

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 (subtraction is your homework)

What to Come

- Lab sessions start from Tuesday.

- Introduction to HCS12