# 시스템 프로그래밍을 위한 C언어

## MCU 프로그램을 위한 기초 지식들

## - Numbers Representation -

현대자동차 입문교육
박대진 교수

# Lecture Lessoned

- 16진수와 2진수 10진수간 변환을 자유롭게
- 바이트, 비트, 니블
- 2의 승수로 값을 어림짐작 하는 방법
- 2진수 더하기
- Overflow
- 음수표현하기
- 2의 보수
- 2의 보수를 쓰면 좋은 점
- Sign Extension
- Signed와 Unsigned의 차이

# Hexadecimal Numbers

| Hex Digit | Decimal Equivalent | Binary Equivalent |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Hexadecimal to Binary Conversion

- Hexadecimal to binary conversion:
  - Convert $4AF_{16}$ (also written 0x4AF) to binary
  - $0100\ 1010\ 1111_2$

- Hexadecimal to decimal conversion:
  - Convert $4AF_{16}$ to decimal
  - $16^2 \times 4 + 16^1 \times 10 + 16^0 \times 15 = 1199_{10}$

# Bits, Bytes, Nibbles

- Bits

10010110

most significant bit

least significant bit

- Bytes & Nibbles

byte

10010110

nibble

- Bytes

CEBF9AD7

most significant byte

least significant byte

# Large Powers of Two

- $2^{10}$ = 1 kilo ≈ 1000 (1024)
- $2^{20}$ = 1 mega ≈ 1 million (1,048,576)
- $2^{30}$ = 1 giga ≈ 1 billion (1,073,741,824)
- $2^{40}$ = 1 tera ≈ 1 trillion
- $2^{50}$ = 1 peta ≈ 1 quadrillion
- $2^{60}$ = 1 exa ≈ 1 quintillion

# Estimating Powers of Two

- What is the value of $2^{24}$?

  $2^4 \times 2^{20} \approx 16$ **million**

- How many values can a 32-bit variable represent?

  $2^2 \times 2^{30} \approx 4$ **billion**

# Addition

- Decimal

$$
\begin{array}{rr}
 & \color{blue}{11\ \leftarrow \text{carries}} \\
 & 3734 \\
+ & 5168 \\
\hline
 & 8902
\end{array}
$$

- Binary

$$
\begin{array}{rr}
 & \color{blue}{11\ \leftarrow \text{carries}} \\
 & 1011 \\
+ & 0011 \\
\hline
 & 1110
\end{array}
$$

# Addition Example

- Add the following 4-bit binary numbers

```
     1
  1001
+ 0101
------
  1110
```

- Add the following 4-bit binary numbers

```
  111
  1011
+ 0110
------
 10001
```

# Overflow

- Digital systems operate on a fixed number of bits
- Overflow: when result is too big to fit in the available number of bits
- See previous example of 11 + 6

```
   111
   1011
+  0110
-------
  10001
```

# Signed Binary Numbers

- Sign/Magnitude Numbers
- Two's Complement Numbers

# Sign/Magnitude Numbers

- 1 sign bit, $N$-1 magnitude bits
- Sign bit is the most significant (left-most) bit
  - Positive number: sign bit = 0
  - Negative number: sign bit = 1

- Example, 4-bit sign/mag representations of ± 6:

  +6 = **0110**

  - 6 = **1110**

- Range of an $N$-bit sign/magnitude number:

  $[-(2^{N-1}-1), 2^{N-1}-1]$

# Sign/Magnitude Numbers

## Problems:

- Addition doesn't work, for example -6 + 6:

$$\begin{array}{r} 1110 \\ +\ 0110 \\ \hline 10100 \end{array}$$ (wrong!)

- Two representations of 0 ($\pm$ 0):

1000

0000

# Two's Complement Numbers

- Don't have same problems as sign/magnitude numbers:
  - **Addition works**
  - **Single representation for 0**

# "Taking the Two's Complement"

- **Method:**
  1. Invert the bits
  2. Add 1
- **Example:** Flip the sign of $3_{10} = 0011_2$
  1. **1100**
  2. **+ 1**

  **$1101 = -3_{10}$**

# Two's Complement Examples

- Take the two's complement of $6_{10} = 0110_2$
  1. 1001
  2. $+ \quad 1$

     $1010_2 = -6_{10}$

- What is the decimal value of the two's complement number $1001_2$?
  1. 0110
  2. $+ \quad 1$

     $0111_2 = 7_{10}$, so $1001_2 = -7_{10}$

# Two's Complement Addition

- Add 6 + (-6) using two's complement numbers

```
   111
    0110
+   1010
--------
   10000
```

- Add -2 + 3 using two's complement numbers

```
   111
    1110
+   0011
--------
   10001
```

# Increasing Bit Width

- **Extend number from *N* to *M* bits (*M* > *N*) :**
  - Sign-extension
  - Zero-extension

# Sign-Extension

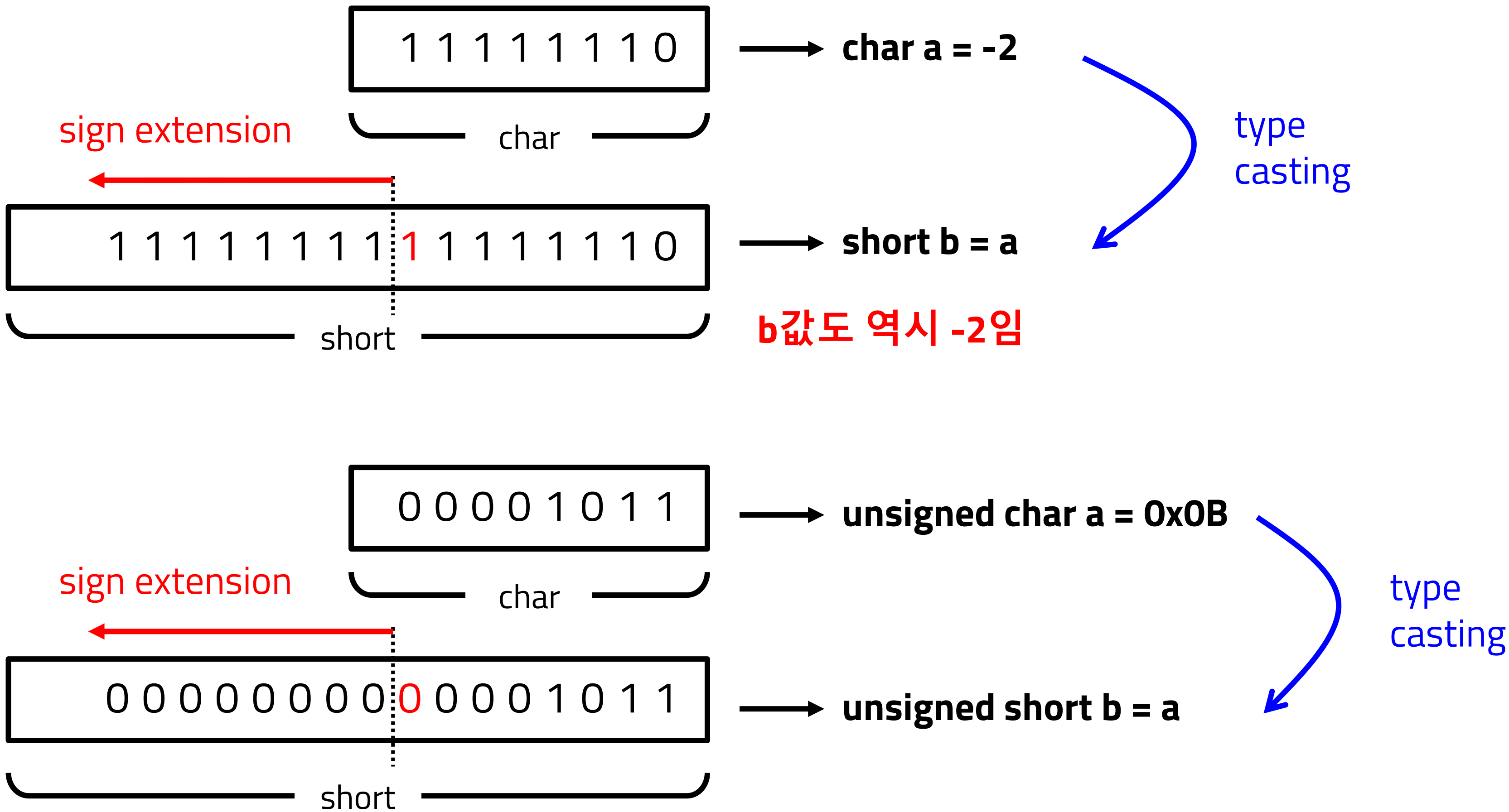
- Sign bit copied to msb's
- Number value is same
- **Example 1:**
  - 4-bit representation of 3 = **0**011
  - 8-bit sign-extended value: **0000**0011
- **Example 2:**
  - 4-bit representation of -5 = **1**011
  - 8-bit sign-extended value: **1111**1011

# Zero-Extension
# (ex. unsigned char → unsigned int)

- Zeros copied to msb's
- Value changes for negative numbers

- **Example 1:**
  - 4-bit value = 0011 = $3_{10}$
  - 8-bit zero-extended value: **0000**0011 = $3_{10}$
- **Example 2:**
  - 4-bit value = 1011 = $11_{10}$
  - 8-bit zero-extended value: **0000**1011 = $11_{10}$

# Sign Extension, Zero Extension in C



# Number System Comparison

| Number System | Range |
|---|---|
| **Unsigned** | $[0, 2^N-1]$ |
| **Sign/Magnitude** | $[-(2^{N-1}-1), 2^{N-1}-1]$ |
| **Two's Complement** | $[-2^{N-1}, 2^{N-1}-1]$ |

For example, 4-bit representation:

| | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Unsigned | | | | | | | | | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| Two's Complement | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | | | | | | | | |
| Sign/Magnitude | | 1111 | 1110 | 1101 | 1100 | 1011 | 1010 | 1001 | 0000<br>1000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | | | | | | | | |