Chapter 3

Driving LEDs and Keypad

Objectives

- Understanding how to set a port pin for digital Input or Output
- Understanding how to design an interface circuit for LEDs with Microcontroller and how to drive them
- Understanding Mask ON/OFF techniques
- Understanding software time delay technique and utilizing it for blinking LEDs
- Understanding how to manipulate Switches in the Keypad

❑ Configuring Port Pins for Digital Input/Output

- Each Port x has 3 Registers for configuration
  - DDRx : Data Direction Register of Port x
  - PORTx : Controls Output or Internal Pull up Resistor of Port x
  - PINx : Input Register of Port x
    - Here, x can be replaced with A or B or C or D or E or F or G

- DDRx(Data Direction Register)
  - Determine bitwise Data Direction of Port x
    - DDRxn ← 1 : Bit n of Port x(Pxn pin) will be used for Digital Output
    - DDRxn ← 0 : Bit n of Port x(Pxn pin) will be used for Digital Input
  - Examples

    DDRC = 0x0F; // 00001111_2

    → Lower 4 Bits of PORTC(PC3 ~ PC0) : Digital Output

    Upper 4 Bits of PORTC(PC7 ~ PC4) : Digital Input

    DDRA = 0x10; // 000100000_2

    → Bit 4(PA4 Pin) of Port A : Digital Output

    Remaining Bits(Pins) of Port A : Digital Input

    DDRB = 0xFF; // 11111111_2, All Pins of Port B : Digital Output

    DDRF = 0x00; // 00000000_2, All Pins of Port F : Digital Input

❑ Configuring Port Pins for Digital Input/Output

- PORTx Register
  - When Pxn Pin is set for Digital Output
    - PORTxn bit is used as Data Latch for Output
    - Logical Value written in PORTxn bit will be output through Pxn Pin and latched
    - Examples

      DDRA = 0xFF; // Set all the Port A Pins for digital output

      PORTA = 0xAA;// Binary 10101010 will be output to PA7 ~ PA0 Pins
  - When Pxn Pin is set for Digital Input
    - PORTxn bit is used to enable/disable the internal Pull-up Resistor for the Pxn pin
    - PORTxn ← '1' : the internal Pull-up Resistor for Pxn pin will be enabled
    - PORTxn ← '0' : the internal Pull-up Resistor for Pxn pin will be disabled
    - Examples

      DDRC = 0x0F; // Binary 00001111

      → Set Upper Nibble of Port C for Input and Lower Nibble for Output

      PORTC = 0xC6; // Binary 11000110

      → Binary data '0110' will be output to the Lower Nibble Pins of Port C

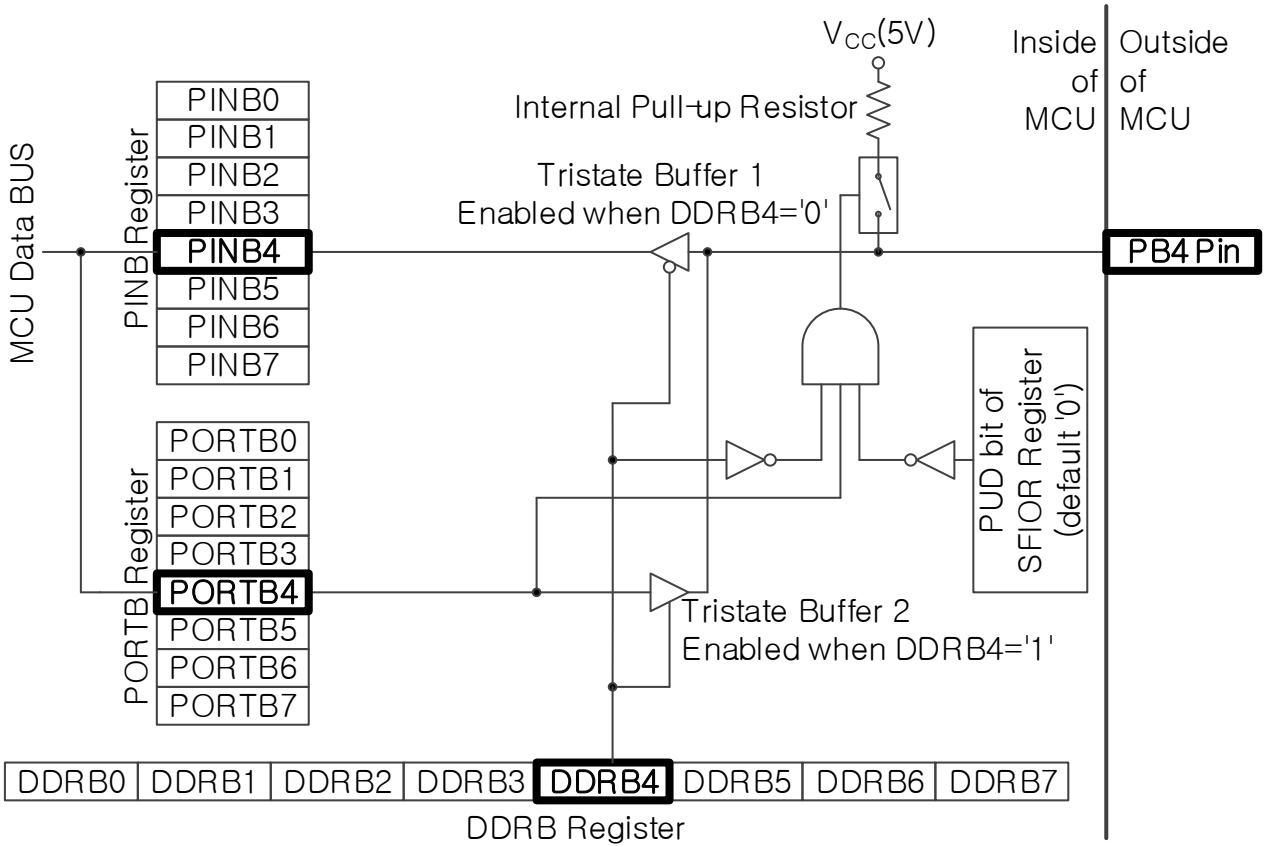
      → Internal Resistors for PC7 and PC6 Pins are activated(enabled) and Internal resistors for PC5 and PC4 Pins are deactivated(disabled)

❑ Configuring Port Pins for Digital Input/Output

- PINx Register
  - PINx Register contains logical status of Input Pins of Port x
    - We can take input values from outside of MCU by reading PINx Register
  - Examples

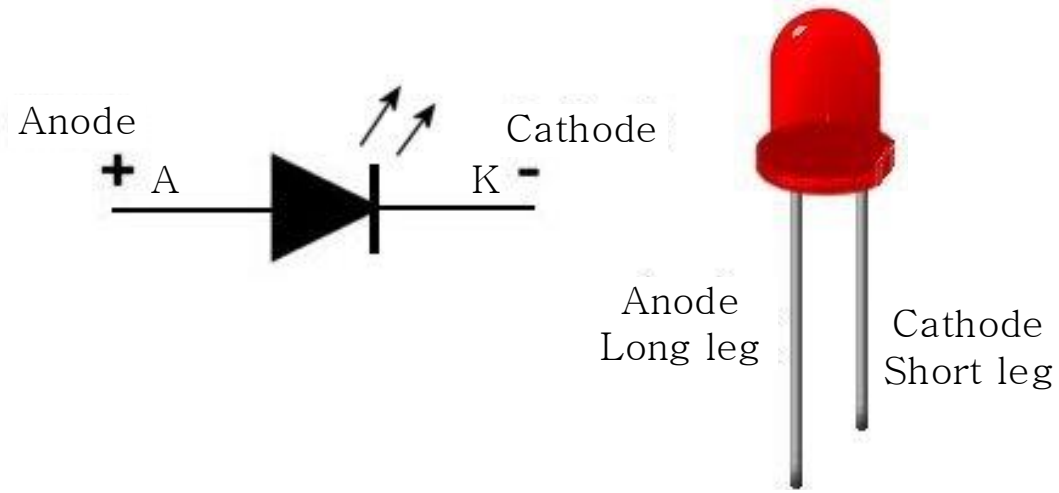
    x = PINE; // takes Input data from Port E and assigns it to the variable x

    if(PINA==0x2A) // if Input data from Port A equals 0x2A(=00101010_2) …

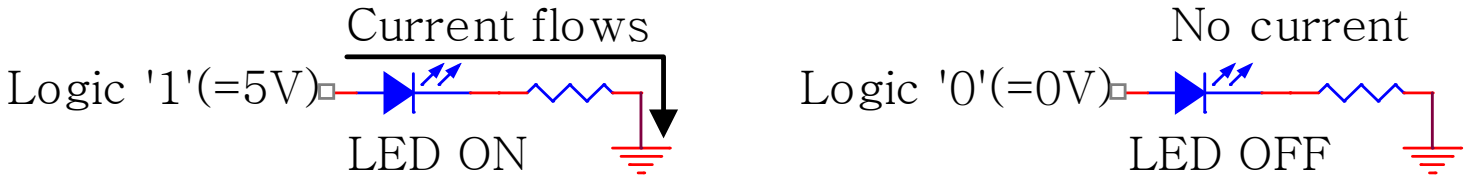


❑ LED Driving Circuit

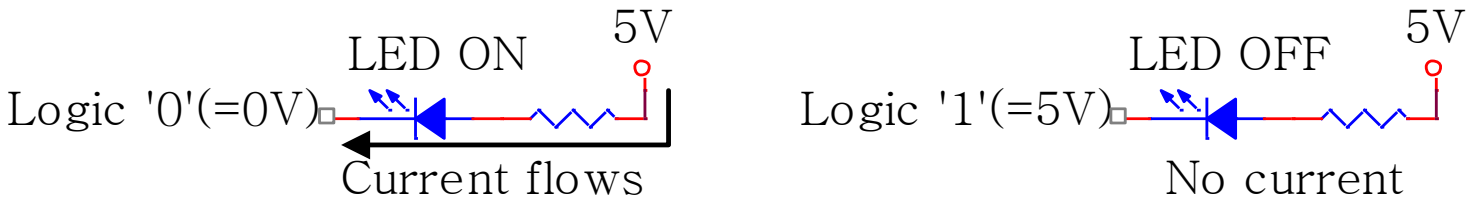
- LED – circuit symbol and typical sample



- LED – driving techniques



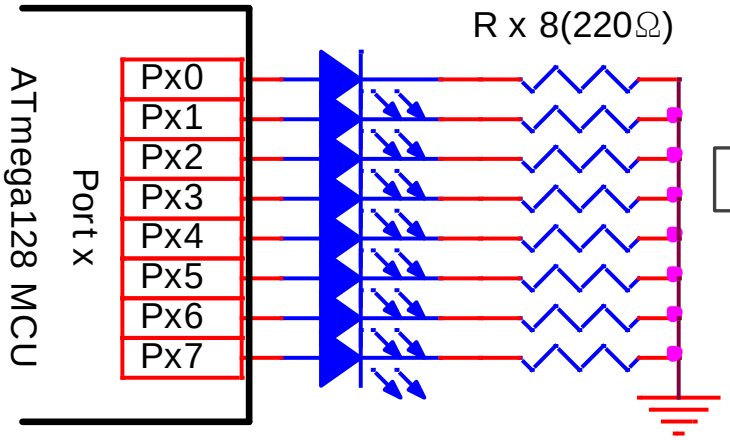
(a) Active High Drive

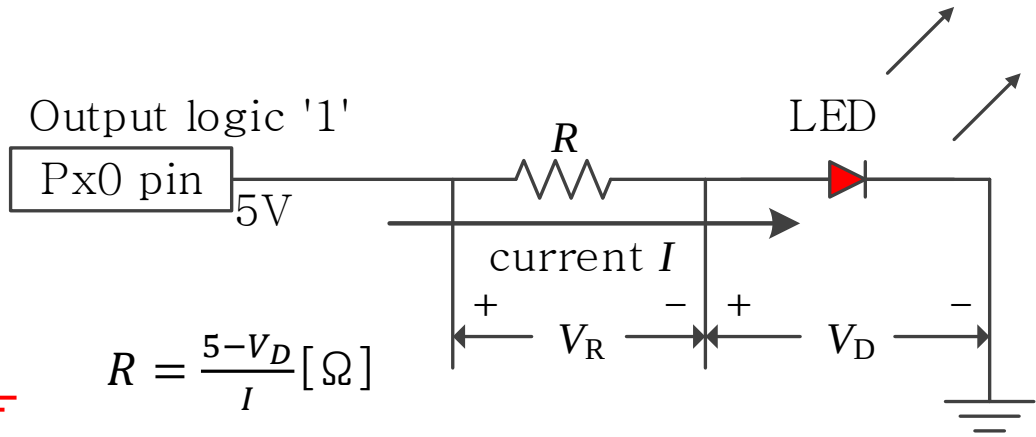


(b) Active Low Drive

❑ LED Driving Circuit

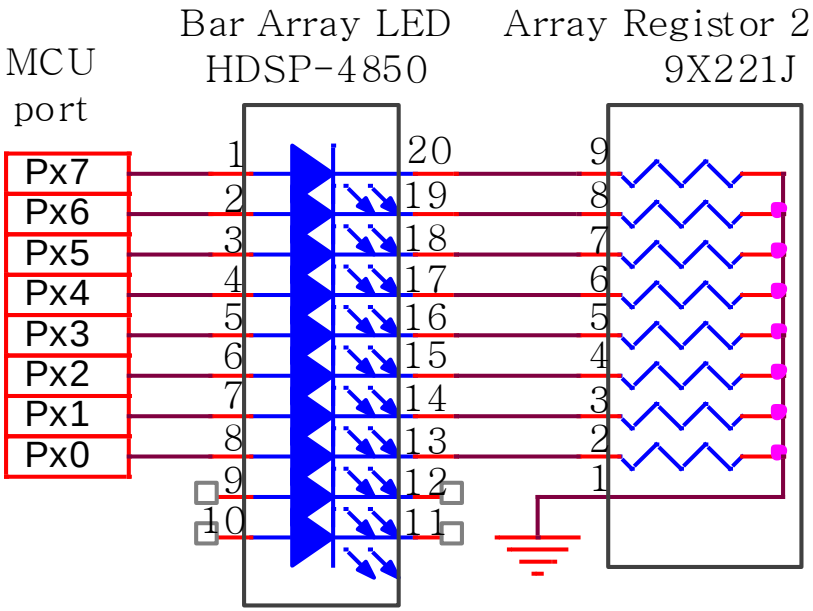
- Circuit for driving 8 LEDs and Design Theory

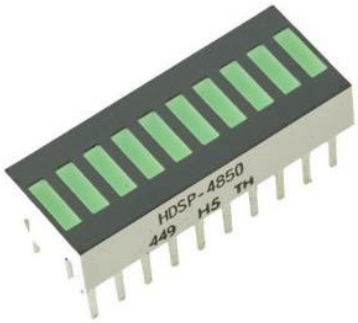


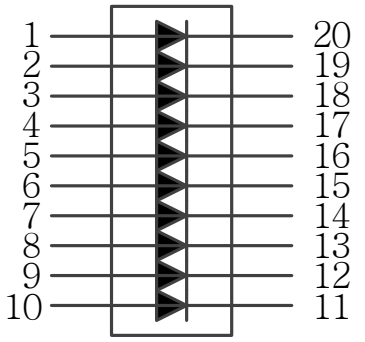


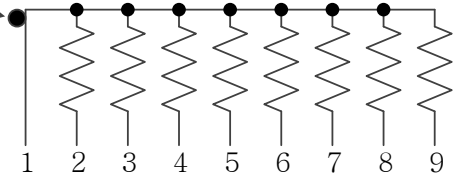
$$R = \frac{5 - V_D}{I} [\Omega]$$

- Modified Circuit for easy Construction



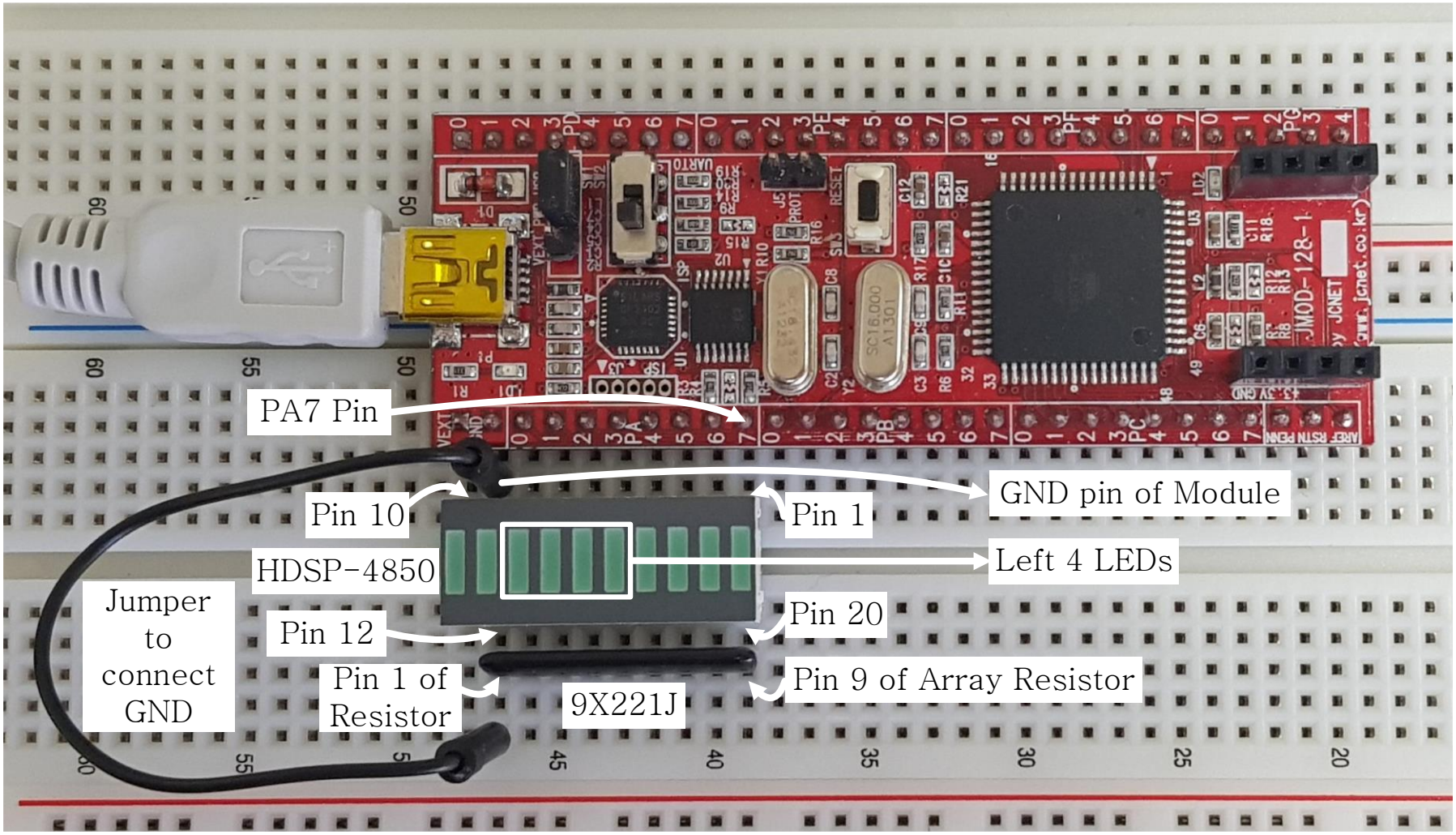






❑ LED Driving Circuit

- Hardware Construction
  - LEDs are connected to Port A of the MCU



❑ Practice 3-1: LED Lighting(ex3-1)

- Goal of the Practice
  - Turn on left 4 LEDs which are connected to the lower 4 bits of Port A

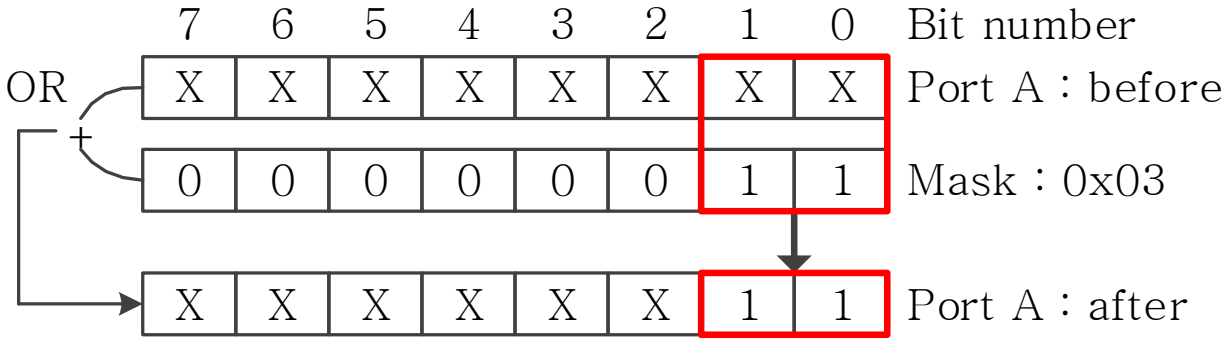
- Necessary Techniques and Programming Algorithms
  - Necessary Techniques
    - Problem Statement
      - For Practice 3-1 we need to set lower 4 bits of Port A for digital Output

        DDRA = 0x0F;
      - It has serious Problem to forcedly set the upper 4 bits of Port A for digital Input while setting lower 4 bits of Port A for digital Output
    - To avoid this problem we need Mask ON/OFF techniques to set/reset selected bits in a byte while other bits are unaffected(unchanged)
    - Mask ON Technique
      - Technique for Setting selected bits while other bits are unaffected
      - Utilizing the characteristic of OR Operation
      - Examples

        DDRA |= 0x0F; // PA3 ~ PA0 pins for Output, other pins unchanged

        PORTA |= 0x03; // Output '1' to PA1, PA0 Pins, other pins unchanged

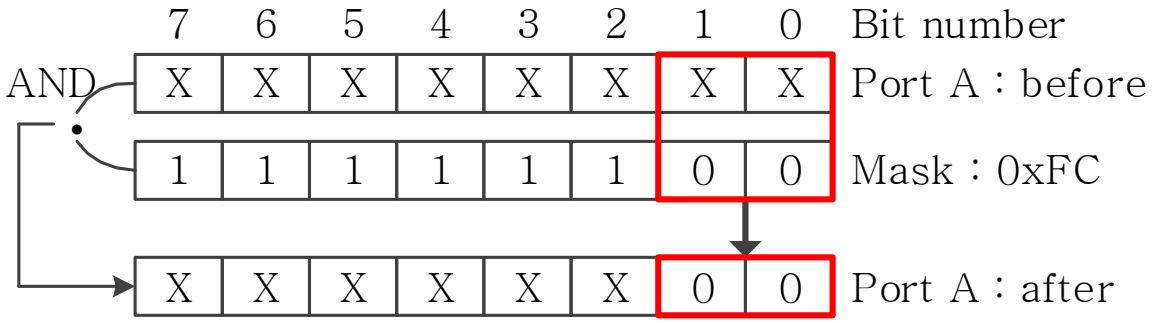
❑ Practice 3-1: LED Lighting(ex3-1)



- Mask OFF Technique
  - Technique for Resetting selected bits while other bits are unaffected
  - Utilizing the characteristic of AND Operation
  - Examples

    DDRA &= 0xF0;

    PORTA &= 0xFC;



- Programming Algorithm
  - Set all pins of Port A for digital output : DDRA = 0xFF;
  - Turn On lower 4 LEDs and turn off upper 4 LEDs : PORTA = 0x0F;
  - Program Termination : while(1)

❑ Practice 3-1: LED Lighting(ex3-1)

- Program Source Code: ex3-1.c

```
#include          <avr/io.h> // 각종 AVR MCU 관련 상수들을 정의한 헤더 파일
int main(void) {
    // DDRA의 모든 비트를 '1'로 설정하여 포트 A의 모든 핀을 출력으로 설정
    DDRA = 0xFF;
    // PA3 ~ PA0 핀에 '1'을 출력시켜 왼쪽 4개의 LED 점등
    PORTA = 0x0F;
    while(1);  // 무한 루프를 돌며 프로그램 종료
}
```

❑ Practice 3-2: LED Blinking(ex3-2)

- Goal of the Practice
  - Turn ON and OFF the left 4 LEDs periodically

- Necessary Techniques and Programming Algorithms
  - Necessary Techniques
    - Time Delay function
      - A function doing nothing but consuming time for a specified interval
      - Software time delay vs. Hardware time delay
      - Library function
        - _delay_ms() : maximally getting [262.14/F_CPU(MHz)]ms
        - _delay_us() : maximally getting [768/F_CPU(MHz)]us
        - You should include the following two compiler directives before using time delay library fuctions(normally beginning of the program)

          ```
          #define    F_CPU    16000000UL
          #include   <util/delay.h>
          ```

        - Arguments for calling delay functions should be constants(variables are not allowed)

❑ Practice 3-2: LED Blinking(ex3-2)

- Custom made function
  - Delay_ms() : 1 ~ 65535ms
  - Delay_us() : 1 ~ 65535us
  - Variables can be used for calling Arguments
- ➢ To get more accurate delays, Compiler Optimization should be activated

- Programming Algorithm
  - Straightforward!
  - Just turn ON the LEDs and wait a little while by calling time delay function(Delay_ms()) and after that turn OFF the LEDs and again wait a little while by calling time delay function and repeat the whole process endlessly.

➢ Refer to the program in the textbook

▪ Preparation for easy Programming

- Make a file named ex.h which contains essential definitions, declarations and functions for all the practices in this textbook.
- Store the ex.h file in the “inc” folder

❑ Practice 3-2: LED Blinking(ex3-2)

- Contents of the ex.h file up to chapter 3

```
// definitions and Declarations added in chapter 3
#define    F_CPU    16000000UL // MCU Clock Frequency
#include   <avr/io.h>
#include   <util/delay.h>

// Time delay functions for obtaining millisecond or microsecond order
void Delay_ms(unsigned int MilliSeconds) {
    unsigned int i;
    for(i=0;i<MilliSeconds;i++) _delay_ms(1);
}

void Delay_us(unsigned int MicroSeconds) {
    unsigned int i;
    for(i=0;i<MicroSeconds;i++) _delay_us(1);
}
```

❑ Practice 3-2: LED Blinking(ex3-2)

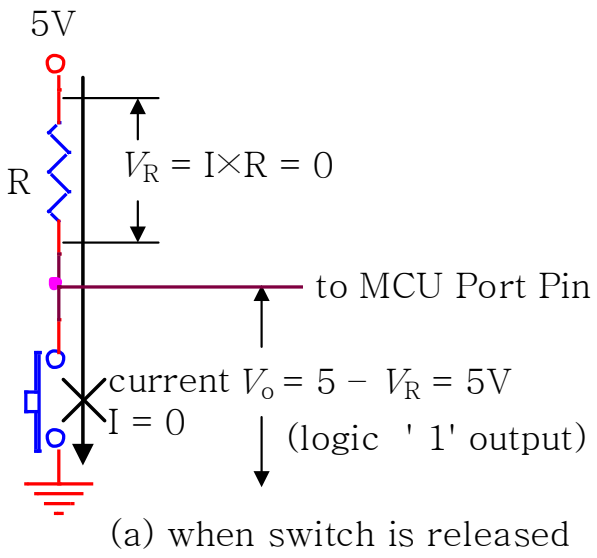
- Program Source Code: ex3-2.c

```
#include      "../../../inc/ex.h“
int main(void) {
    // DDRA의 모든 비트를 ‘1’로 설정하여 포트 A의 모든 비트를 출력으로 설정
    DDRA = 0xFF;
    PORTA = 0x00; // 초기 상태 모든 LED 소등
    while(1) { // 다음의 과정을 무한히 반복하게 한다
        // PA3 ~ PA0 핀에 “1111”을 출력시켜 LED 4개 점등
        PORTA |= 0x0F; // 비트 3 ~ 0 mask on, 나머지 비트들은 이전 값 유지
        // 딜레이 함수 호출
        Delay_ms(500);  // 약 0.5초(500mS) 딜레이
        // PA3 ~ PA0 핀에 “0000”을 출력시켜 LED 소등
        // PORTA 레지스터의 비트 3 ~ 0 mask off, 나머지 비트들은 이전 값 유지
        PORTA &= 0xF0;
        // 딜레이 함수 호출
        Delay_ms(500);  // 약 0.5초 지연
    } // while 루프의 끝
} // 주 프로그램 끝
```

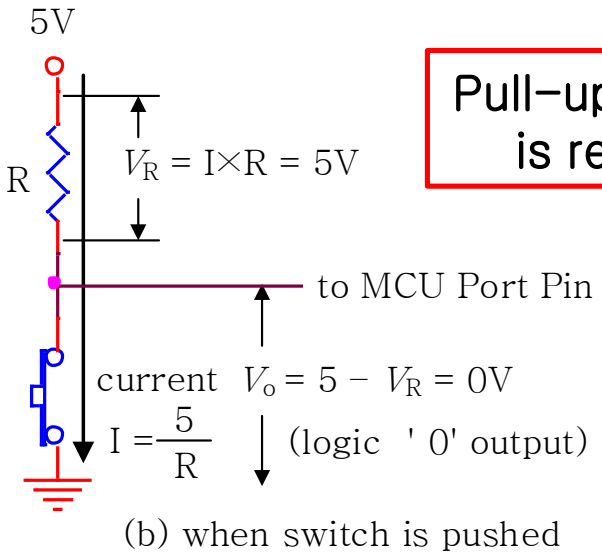
❑ Practice 3-3: LED control 1 by Switch Input(ex3-3)

- Goal of the Practice
  - Add a Pushbutton switch as an Input to control LED lighting
  - If a switch is pushed then the 8 LEDs will be turned ON and if the switch is released then all LEDs will be turned OFF
  - Here we will use 4 x 4 Keypad instead of using a single Switch

- Necessary Techniques and Programming Algorithms
  - Necessary Techniques
    - Designing a Pushbutton Input circuit



(a) when switch is released

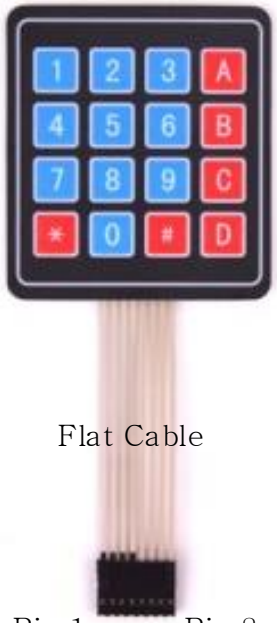


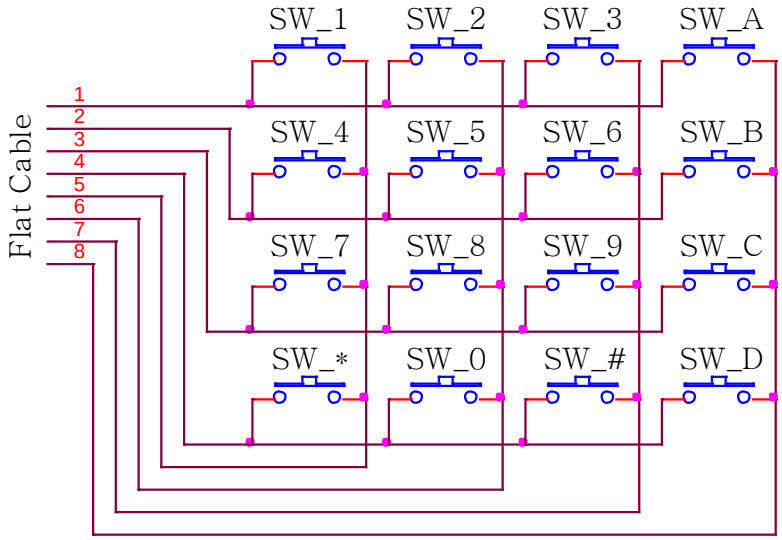
(b) when switch is pushed

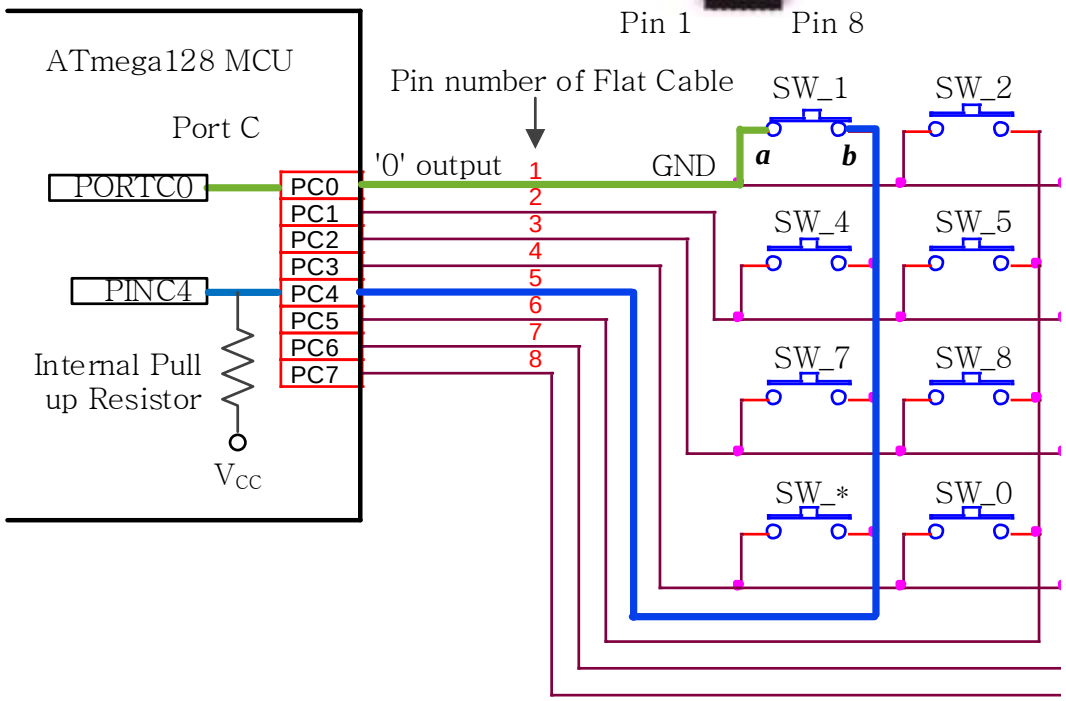
Pull-up Resistor is required

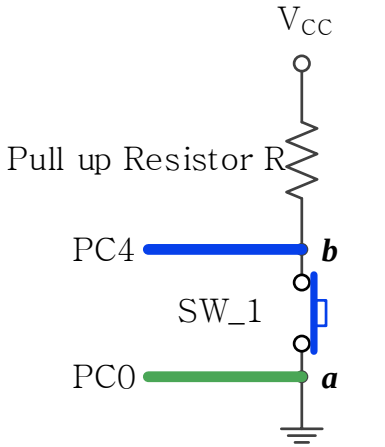
❑ Practice 3-3: LED control 1 by Switch Input(ex3-3)

- 4 x 4 Keypad Interface
  - Use Port C









Basic input circuit of Pushbutton switch

❑ Practice 3-3: LED control 1 by Switch Input(ex3-3)

- Testing the digital logic value of a specified bit in a byte data
  - Applying mask ON or mask OFF Operation
  - For example, in order to identify the status of SW_1 in this practice we should test the value of bit 4 of PINC register to which the Input side of the SW_1 switch is connected
  - Applying mask ON Operation to the PINC register using OR operation

7	6	5	4	3	2	1	0	← Bit number
X	X	X	SW_1	X	X	X	X	← Value of PINC register
1	1	1	0	1	1	1	1	← Mask(0xEF)
1	1	1	SW_1	1	1	1	1	← Result after OR Operation

```
if(PINC | 0xEF)==0xEF)  -----;
```

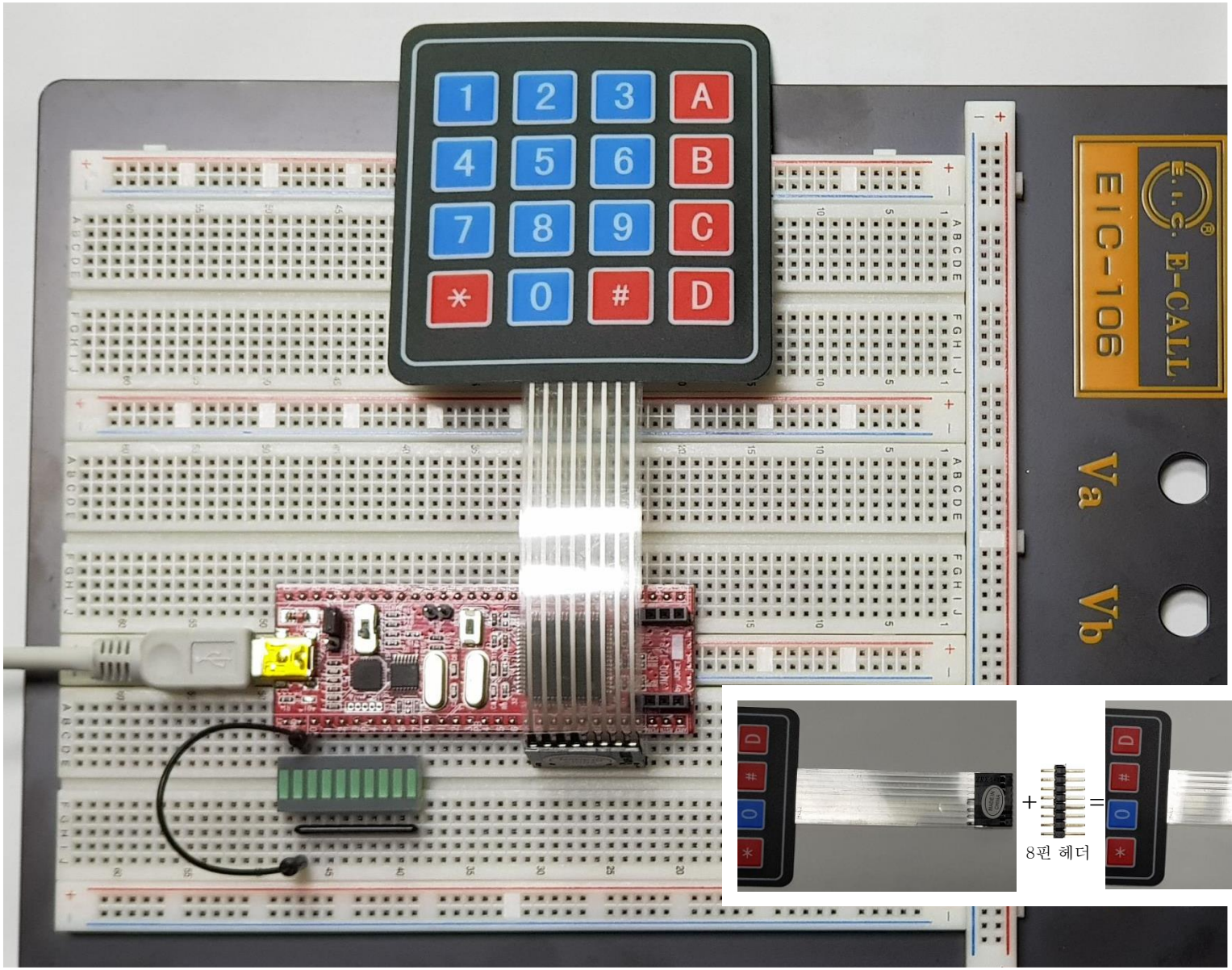
  - Applying mask OFF Operation to the PINC register using AND operation

7	6	5	4	3	2	1	0	← Bit number
X	X	X	SW_1	X	X	X	X	← Value of PINC register
0	0	0	1	0	0	0	0	← Mask(0x10)
0	0	0	SW_1	0	0	0	0	← Result after AND Operation

```
if(PINC & 0x10)==0x00)  -----;
if(!(PINC & 0x10)  -----;
```

❑ Practice 3-3: LED control 1 by Switch Input(ex3-3)

- Hardware construction – adding Keypad to PORTC



❑ Practice 3-3: LED control 1 by Switch Input(ex3-3)

- Programming Algorithm
  - Straightforward!
- A new technique used in this program
  - Using shift operator to create Masks for Mask ON or OFF
  - Examples
    - To set bit 5 of DDRB register(Mask ON), use
      DDRB |= 1<<5; instead of DDRB |= 0x20;
    - To clear bit 5 of DDRB register(Mask OFF), use
      PORTB &= ~(1<<5); instead of PORTB &= 0xDF; // more easier

❑ Practice 3-3: LED control 1 by Switch Input(ex3-3)

- Program Source Code: ex3-3.c

```
#include      <avr/io.h>
int main(void) {

    // LED 출력 포트 설정
    // LED가 연결된 포트 A 전체를 출력으로 설정
    DDRA = 0xFF;

    // SW_1 스위치의 접지 쪽 단자 설정
    // 포트 C의 비트 0 핀(PC0)를 출력으로 설정
    DDRC |= 1<<0;

    // PC0 핀에 '0'을 출력시켜 SW_1 스위치의 한쪽을 접지 시킨다.
    PORTC &= ~(1<<0);

    // SW_1 스위치의 입력 쪽 단자 설정
    // 포트 C의 비트 4 핀(PC4)을 입력으로 설정
    DDRC &= ~(1<<4); // PC4 핀은 SW_1 스위치의 다른 쪽에 연결되어 있음
```

❑ Practice 3-3: LED control 1 by Switch Input(ex3-3)

```
    // 포트 C의 비트 4 핀(PC4)에 대한 내부 풀업 저항 활성화
    PORTC |= 1<<4;
    while(1) { // 다음의 과정을 무한히 반복하게 한다
        // SW_1 스위치가 눌러졌으면 LED 8개를 모두 점등한다
        if(!(PINC & (1<<4))) PORTA = 0xFF;
        // SW_1이 눌러져 있지 않으면 LED를 모두 소등한다
        else PORTA = 0x00;
    } // while 루프의 끝
}       // 프로그램 끝
```

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-1)

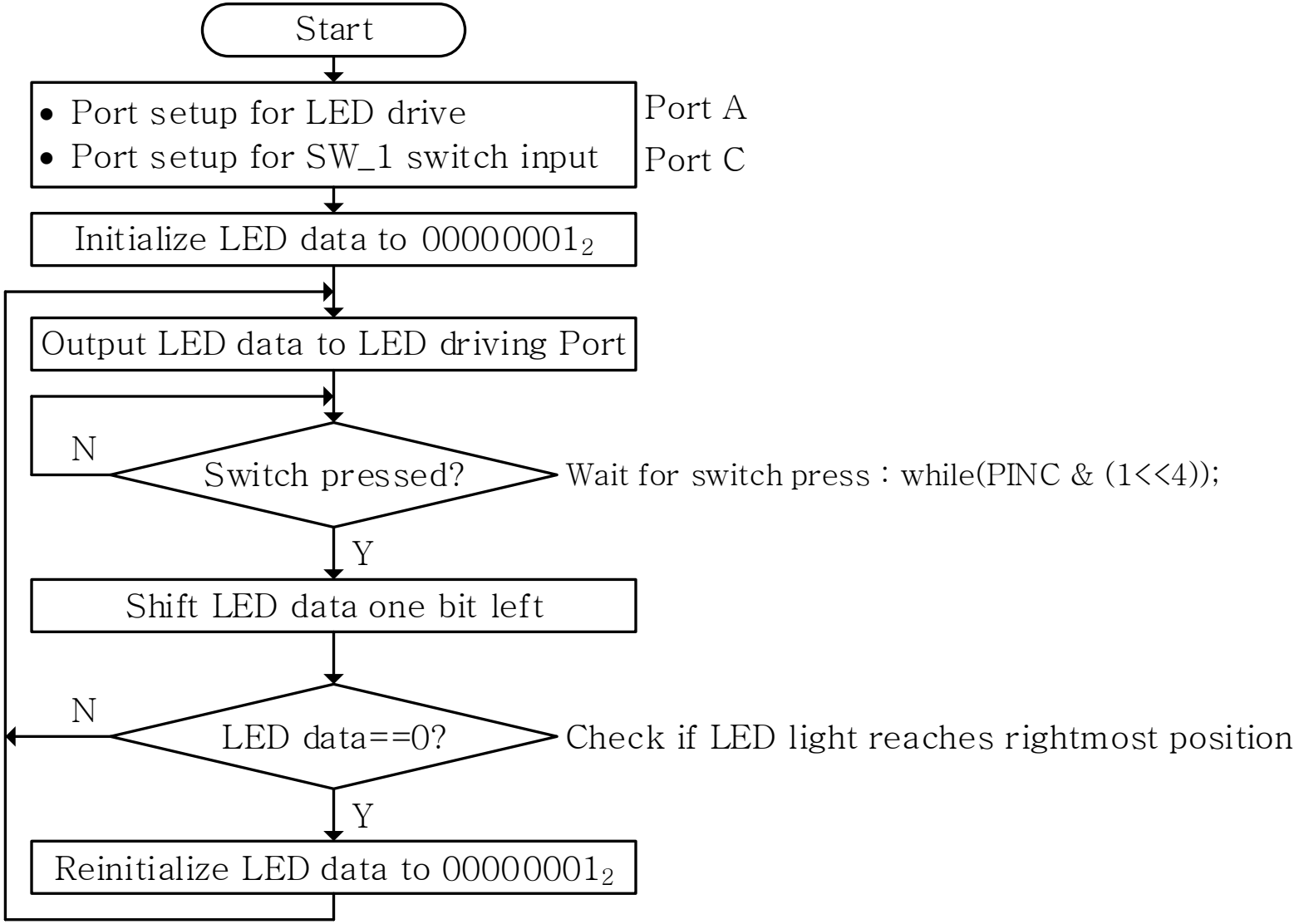
- ## Goal of the Practice
  - When Power is turned on, leftmost LED will be lit
  - Afterwards, whenever a specified switch is pressed LED light moves one position to the right
  - If LED light reaches to the rightmost place then LED light will move to the leftmost position again by the next switch press and repeat above process infinitely

- ## Necessary Techniques and Programming Algorithms
  - Necessary Techniques
    - In order to move LED light one position to the right with every switch press you should output a sequence of binary data to Port A as follows;

      00000001, 00000010, 00000100, 00001000, 00010000, 00100000, 01000000, 10000000
    - To do this we can use “shift left” operator ‘<<‘
    - ➢ To understand the relationship between the binary sequence and LED light movement direction, examine the arrangement of LEDs in the assembled circuit on the breadboard

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-1)

- Programming Algorithm 1 – Problem Introduction



❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-1)

- Program Source Code: ex3-4-1.c

```
#include <avr/io.h>
int main(void) {
    unsigned char LED_Data;

    // LED 출력 포트 설정
    // LED가 연결된 포트 A 전체를 출력으로 설정
    DDRA = 0xFF;

    // SW_1 스위치의 접지쪽 단자 설정
    // 포트 C의 비트 0 핀(PC0)를 출력으로 설정
    DDRC |= 1<<0;

    // PC0 핀에 '0'을 출력시켜 SW_1 스위치의 한쪽을 접지 시킨다.
    PORTC &= ~(1<<0);
```

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-1)

```
// SW_1 스위치의 입력쪽 단자 설정
// 포트 C의 비트 4 핀(PC4)을 입력으로 설정
DDRC &= ~(1<<4); // PC4 핀은 SW_1 스위치의 다른 쪽에 연결되어 있음

// 포트 C의 비트 4 핀(PC4)에 대한 내부 풀업 저항 활성화
PORTC |= 1<<4;

// LED 데이터 초기화
LED_Data = 0x01; // = 00000001₂, 초기 맨 왼쪽 LED부터 시작

while(1) { // 다음의 과정을 무한히 반복하게 한다
    PORTA = LED_Data;// LED 구동 포트로 데이터를 보내 LED 점등

    // SW_1 스위치가 눌러질 때까지 기다린다.
    while(PINC & (1<<4));  // SW_1이 눌러지지 않으면 계속 루핑

    // SW_1이 눌러졌으면 LED_Data 값을 한 비트 왼쪽으로 시프트
    LED_Data <<= 1;
```

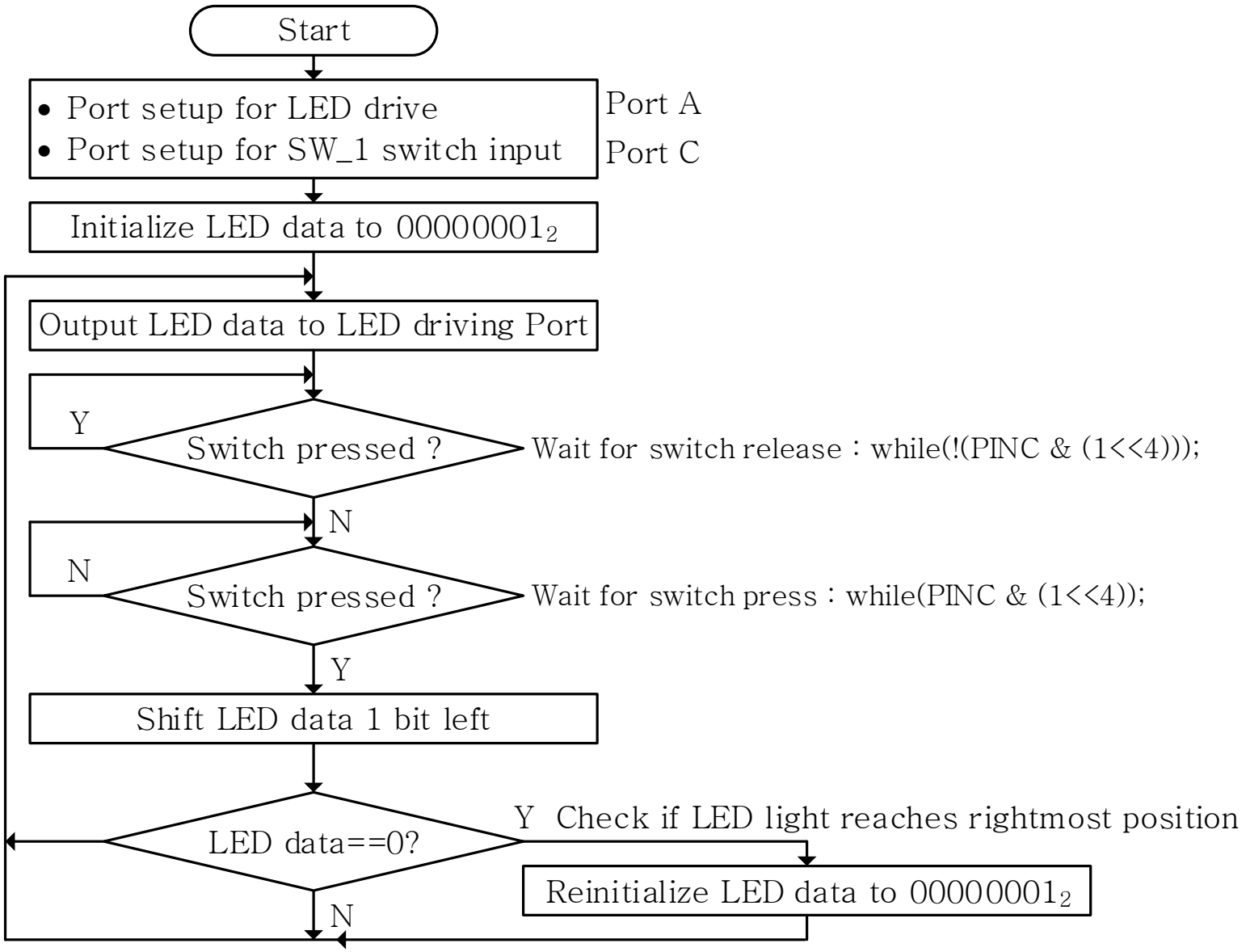
❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-1)

```
        // 왼쪽으로 완전히 시프트가 되었는지(LED_Data 값이 00000000₂ 인지)
        // 검사하여 그렇다면 LED_Data 값을 다시 00000001₂로 초기화 한다.
        if(!LED_Data) LED_Data = 0x01;

    } // while 루프의 끝
}         // 프로그램 끝
```

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-2)

- Programming Algorithm 2 – First Solution



❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-2)

- Program Source Code: ex3-4-2.c

```
#include <avr/io.h>
int main(void) {

    unsigned char LED_Data;

    // LED 출력 포트 설정
    // LED가 연결된 포트 A 전체를 출력으로 설정
    DDRA = 0xFF;

    // SW_1 스위치의 접지쪽 단자 설정
    // 포트 C의 비트 0 핀(PC0)를 출력으로 설정
    DDRC |= 1<<0;

    // PC0 핀에 '0'을 출력시켜 SW_1 스위치의 한쪽을 접지 시킨다.
    PORTC &= ~(1<<0);
```

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-2)

```
// SW_1 스위치의 입력쪽 단자 설정
// 포트 C의 비트 4 핀(PC4)을 입력으로 설정
DDRC &= ~(1<<4); // PC4 핀은 SW_1 스위치의 다른 쪽에 연결되어 있음

// 포트 C의 비트 4 핀(PC4)에 대한 내부 풀업 저항 활성화
PORTC |= 1<<4;

// LED 데이터 초기화
LED_Data = 0x01; // = 00000001₂, 초기 맨 왼쪽 LED부터 시작

while(1) { // 다음의 과정을 무한히 반복하게 한다
    PORTA = LED_Data;     // LED 구동 포트로 데이터를 보내 LED 점등

    while(!(PINC & (1<<4))); // SW_1 스위치가 떨어질 때까지 기다린다.

    while(PINC & (1<<4));    // SW_1 스위치가 다시 눌러질 때까지 기다린다.

    LED_Data <<= 1;     // LED_Data 값을 한 비트 왼쪽으로 시프트
```

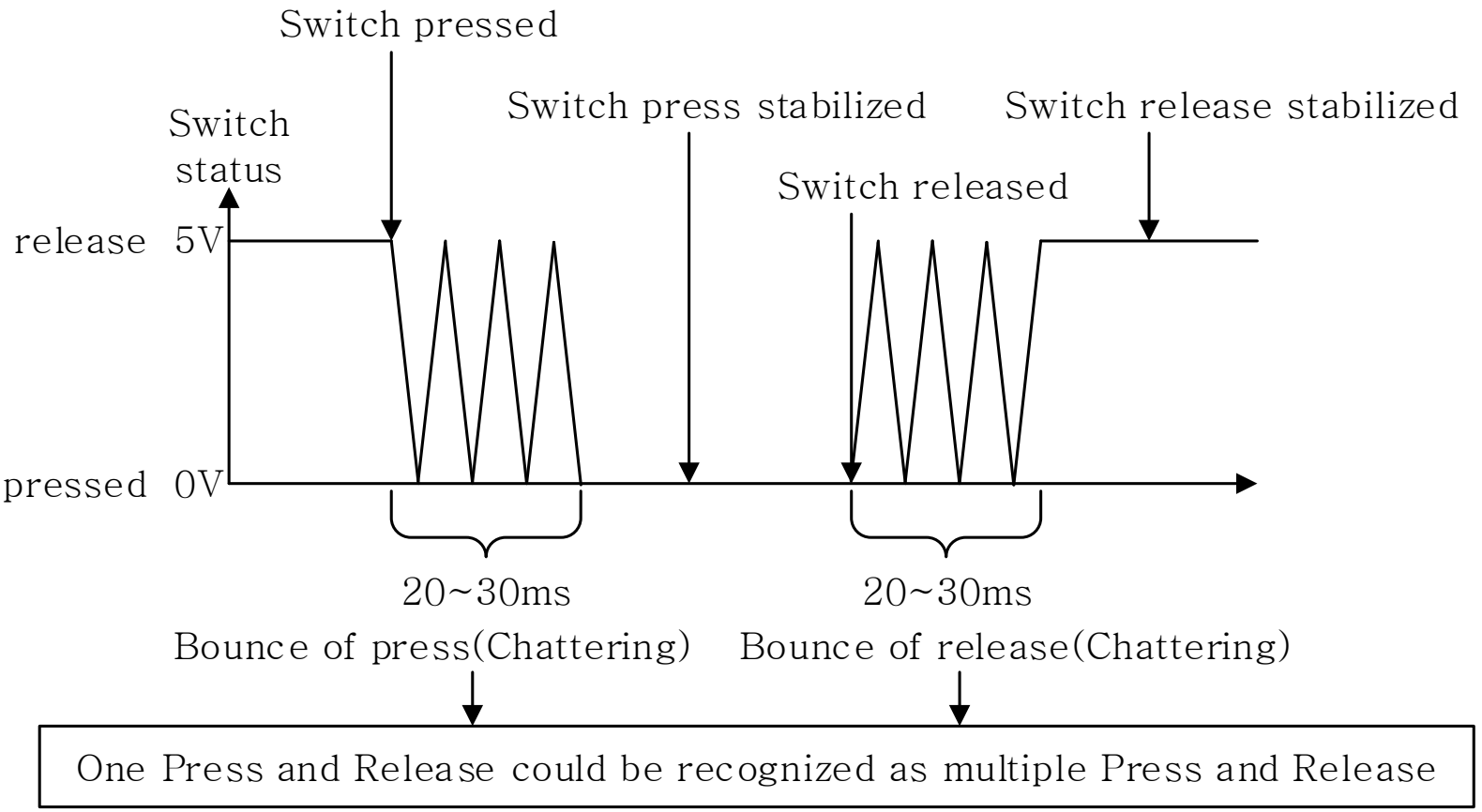
❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-2)

```
        // 왼쪽으로 완전히 시프트가 되었는지(LED_Data 값이 00000000₂ 인지)
        // 검사하여 그렇다면 LED_Data 값을 다시 00000001₂로 초기화 한다.
        if(!LED_Data) LED_Data = 0x01;

    } // while 루프의 끝
} // 프로그램 끝
```

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-3)

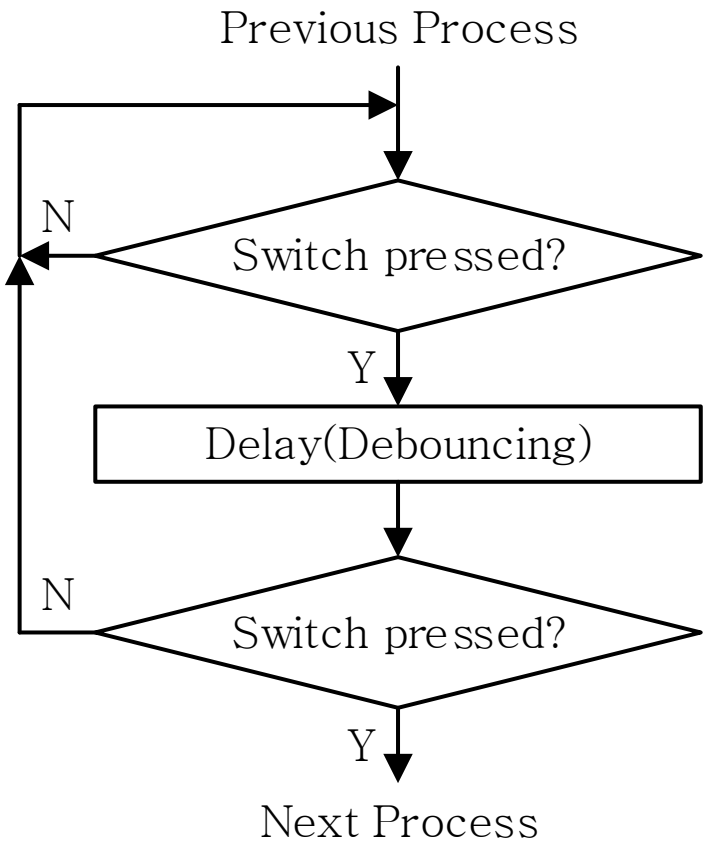
- Programming Algorithm 3 – Second Solution : Complete Solution
  - Bouncing



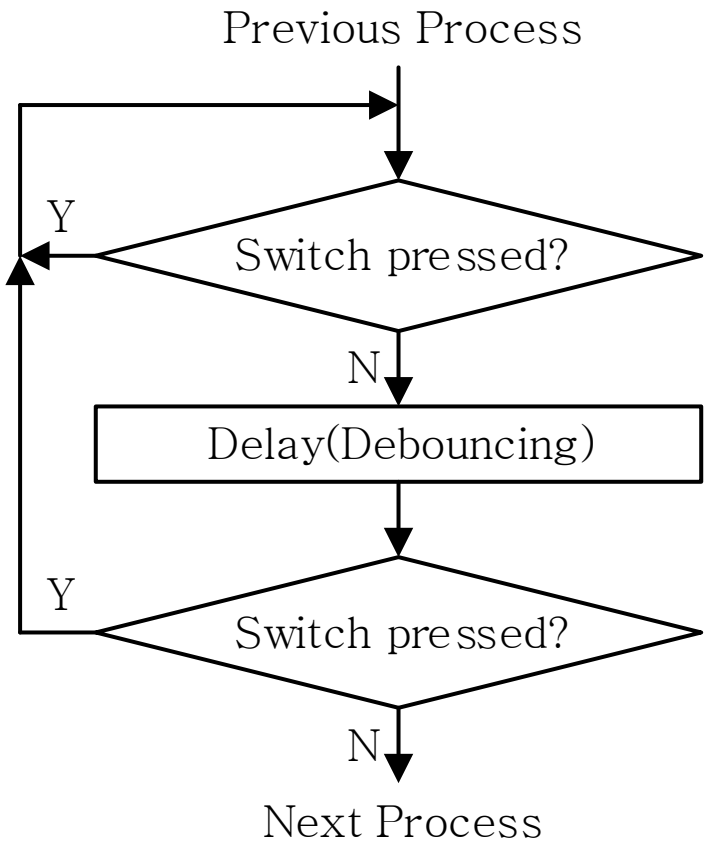
  - Debouncing
    - Techniques for eliminate bounces and generate only one pulse for one press of a switch → can be done by Hardware or Software

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-3)

- Software Debouncing
  - Debouncing should be done at both of press and release



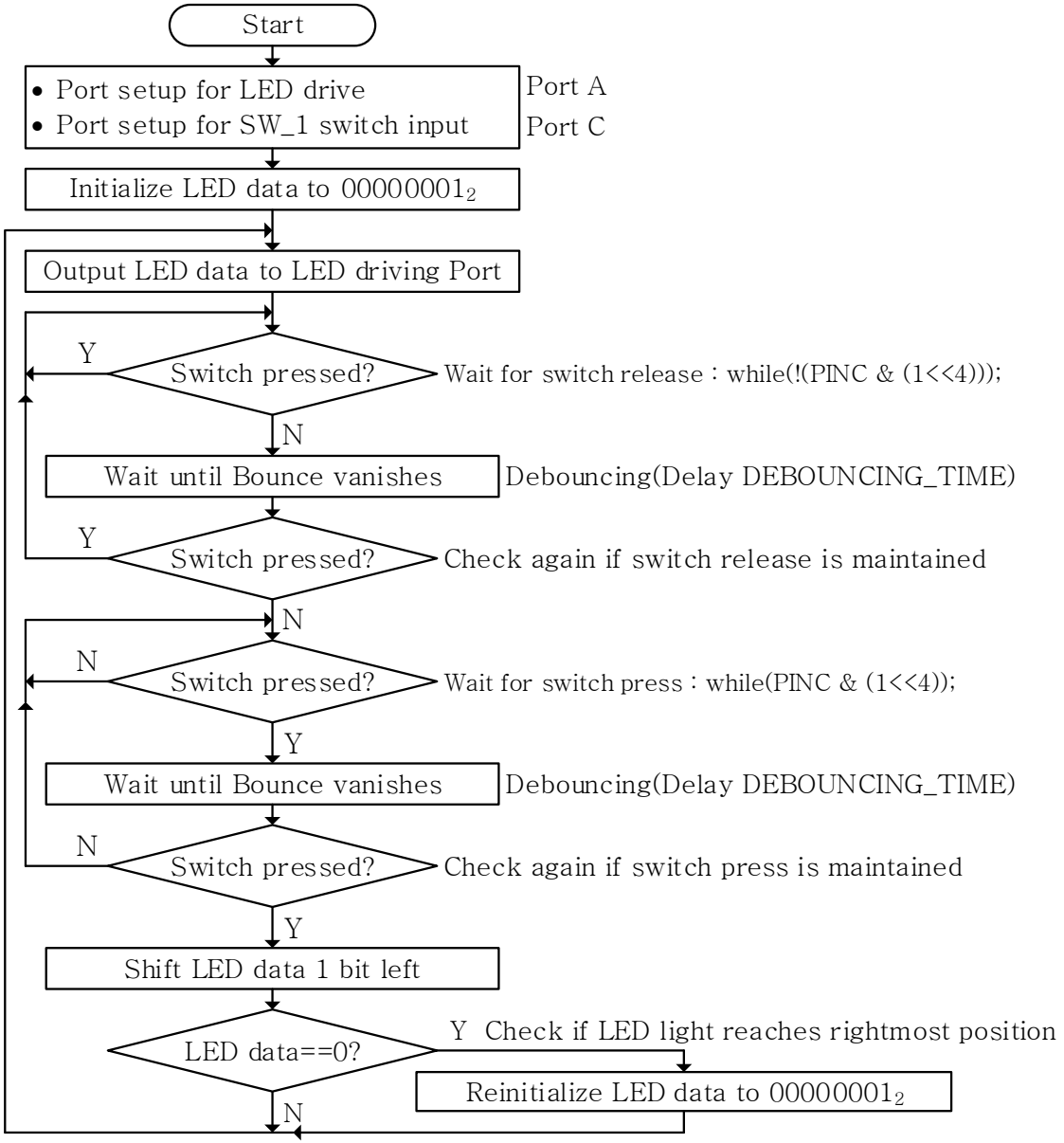
(a) Debouncing for Switch Press



(b) Debouncing for Switch Release

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-3)

- Final Algorithm 3 with Switch Debouncing



❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-3)

- Program Source Code: ex3-4-3.c

```
#include      "../../../inc/ex.h"
#define       DEBOUNCING_TIME     20 // 디바운싱 시간 20ms
int main(void) {

    unsigned char LED_Data;

    // LED 출력 포트 설정
    // LED가 연결된 포트 A 전체를 출력으로 설정
    DDRA = 0xFF;

    // SW_1 스위치의 접지쪽 단자 설정
    // 포트 C의 비트 0 핀(PC0)를 출력으로 설정
    DDRC |= 1<<0;

    // PC0 핀에 '0'을 출력시켜 SW_1 스위치의 한쪽을 접지 시킨다.
    PORTC &= ~(1<<0);
```

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-3)

```
// SW_1 스위치의 입력쪽 단자 설정
// 포트 A의 비트 4 핀(PC4)을 입력으로 설정
DDRC &= ~(1<<4); // PC4 핀은 SW_1 스위치의 다른 쪽에 연결되어 있음

// 포트 C의 비트 4 핀(PC4)에 대한 내부 풀업 저항 활성화
PORTC |= 1<<4;

// LED 데이터 초기화
LED_Data = 0x01; // = 00000001₂, 초기 맨 왼쪽 LED부터 시작

while(1) { // 다음의 과정을 무한히 반복하게 한다
    PORTA = LED_Data;    // LED 구동 포트로 데이터를 보내 LED 점등

    while(1) { // SW_1 스위치 누름이 해제될 때까지 루핑
        if(!(PINC & (1<<4))) continue; // 눌러져 있으면 떨어질 때까지 기다림
        // 버튼스위치가 떨어지고 난 후 바운싱이 종료될 때까지 기다린다.
        _delay_ms(DEBOUNCING_TIME);
        // 바운싱 종료 후 다시 스위치의 상태 확인
```

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-3)

```
    if(!(PINC & (1<<4))) continue;     // 아직 스위치가 눌러져 있는 상태면
                                       // noise 입력으로 간주하고 다시 while
                                       // 루프를 반복하며 버튼 해제를 기다림
    else break;   // 디바운싱 시간 경과 후 버튼 해제가 안정적으로 유지되고
                  // 있으면 while 루프를 빠져 나가 다음 프로세스로 진행
}   // 내부 while 루프 끝

while(1) { // SW_1 스위치가 눌러질 때까지 루핑
    if(PINC & (1<<4)) continue; // SW_1 스위치가 눌러질 때까지 기다림
                  // 스위치가 눌러지고 난 후 바운싱이 종료될 때까지 기다린다.
    _delay_ms(DEBOUNCING_TIME);
    // 바운싱 종료 후 다시 스위치의 상태 확인
    if(PINC & (1<<4)) continue;    // 이 때 스위치가 해제된 상태이면 noise
                                   // noise 입력으로 간주하고 다시 while
                                   // 루프를 반복하며 버튼 누름을 기다린다.
    else break;   // 디바운싱 시간 경과 후 버튼 누름이 안정적으로 유지되고
                  // 있으면 while 루프를 빠져 나가 다음 프로세스로 진행
}   // 내부 while 루프 끝
```

❑ Practice 3-4: LED control 2 by Switch Input(ex3-4-3)

```
        LED_Data <<= 1;    // LED_Data 값을 한 비트 왼쪽으로 시프트

        // 왼쪽으로 완전히 시프트가 되었는지(LED_Data 값이 00000000₂ 인지)
        // 검사하여 그렇다면 LED_Data 값을 다시 00000001₂로 초기화 한다.
        if(!LED_Data) LED_Data = 0x01;
    } // while 루프의 끝

} // 프로그램 끝
```