1강. GPIO

박 원 엽
010.5451.0113

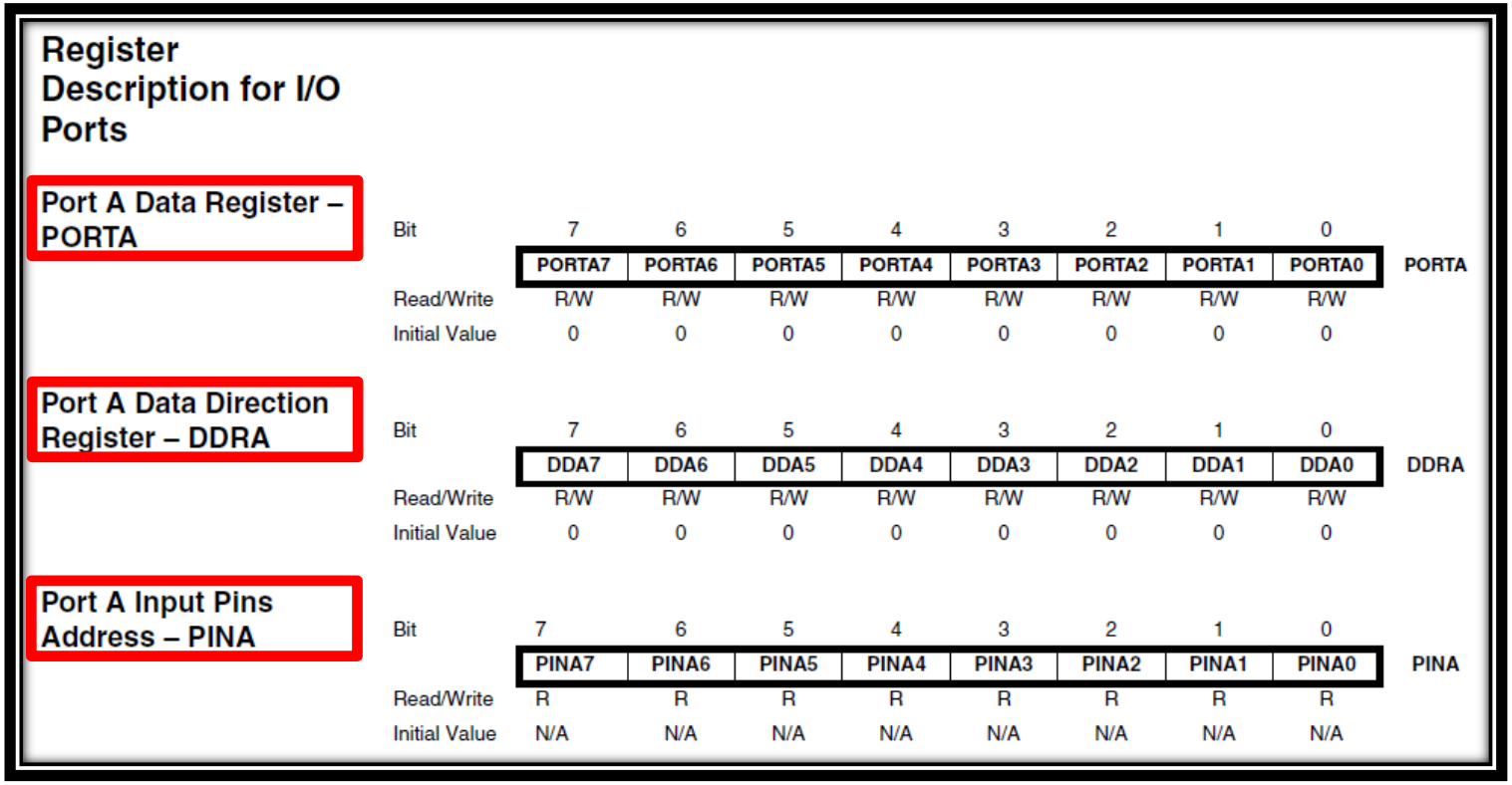
목 차

1강 General Purpose Input/Output (GPIO)

- **GPIO란?**
  - **범용적으로 사용되는 핀 입/출력을 뜻함**
  - **ATmega128은 모든 포트(A~G까지)를 GPIO 핀으로 사용 가능**
    단, G포트는 특수목적 용도로 사용하므로 GPIO로는 잘 사용하지 않음
  - **출력으로 설정 시, 해당 핀에서 Low(전기적 0V) 또는 High(전기적 +5V) 중 하나의 신호를 출력할 수 있음**
  - **입력으로 설정 시, 해당 핀에서 Low(전기적 0V) 또는 High(전기적 +5V) 중 하나의 신호를 입력받을 수 있음**

1강 General Purpose Input/Output (GPIO)

- 데이터시트 87페이지



Register Description for I/O Ports

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

- **DDRx: 입출력 방향 설정 레지스터, x=A~F**

  - x포트의 8개 핀의 입출력 방향을 설정
  - 레지스터의 각각의 비트(0~7번)는 각각의 핀(0~7번)의 입출력 방향을 결정
  - 해당 비트가 **0이면 입력**, **1이면 출력**으로 설정됨
  - 입력으로 설정 시, PINx 레지스터를 이용하여 해당 핀의 입력 신호(H 또는 L) 읽을 수 있음
  - 출력으로 설정 시, PORTx 레지스터를 이용하여 해당 핀의 출력 신호(H 또는 L)를 설정할 수 있음

▪코드 예

```
DDRA = 0xff; //A포트의 모든 핀(0~7번)을 출력으로 설정

DDRB = 0x3e; //0x3e는 0011 1110 이므로 B포트의 0, 6, 7번핀
             을 입력, 그 외(1, 2, 3, 4, 5번)의 핀들은 출력으로
             설정
```

- **PINx: 입력신호 데이터 레지스터, x=A~F**
  - x포트의 8개 핀의 입력 데이터가 저장됨
  - 레지스터의 각각의 비트(0~7번)는 각각의 핀(0~7번)의 입력 신호를 나타냄
  - 해당 비트가 0이면 L(0V), 1이면 H(5V) 신호가 입력 된 것
  - 레지스터의 값은 외부의 입력 신호에 의해 자동으로 변함
  - 따라서 PINx 레지스터에 특정 값을 입력하는 것이 불가!
  (즉, 읽기만 가능한 레지스터!!! 하지만 코드상의 에러는 없다)

▪코드 예

```
DDRA = 0x00; //A포트의 모든 핀(0~7번)을 입력으로 설정

if(PINA&0x01 == 0x01) //A포트 0번 핀에 스위치가 연결되어있다고 가정
{
    //스위치를 누르면 H신호가 입력된다 했을 때, 이곳에 수행할 코드 작성
}
```

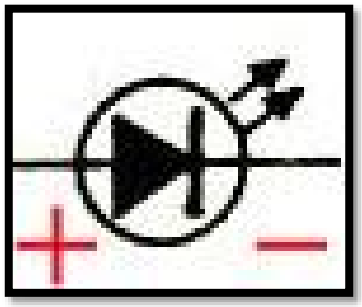
- **PORTx: 출력신호 데이터 레지스터, x=A~F**
  - x포트의 8개 핀의 출력 신호를 설정
  - 레지스터의 각각의 비트(0~7번)는 각각의 핀(0~7번)의 출력 신호를 결정
  - 해당 비트가 0이면 L(0V), 1이면 H(5V) 신호를 출력

▪ 코드 예

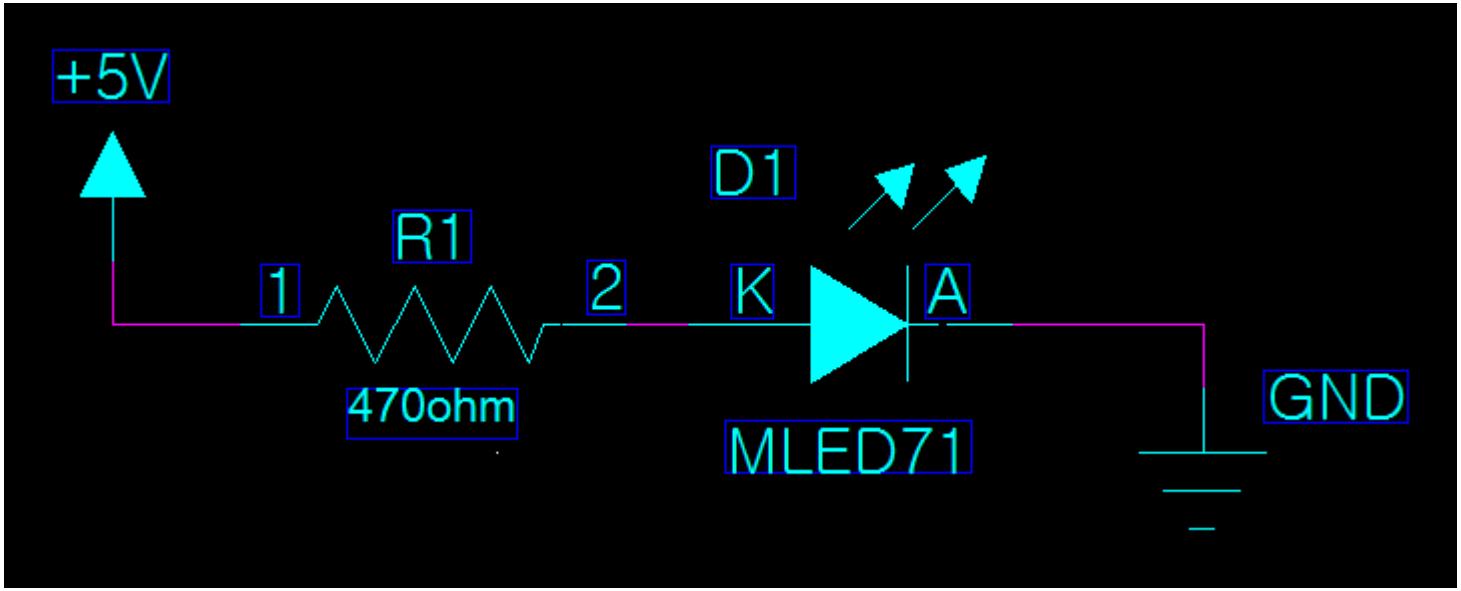
```
DDRA = 0x02; //A포트의 1번 핀을 출력, 나머지 핀을 입력으로 설정

if(PINA&0x01 == 0x01) //A포트 0번 핀에 스위치가 연결되어있다고 가정
{
    PORTA = 0x02; //스위치를 누르면 A포트 1번 핀에 H신호 출력
}
else
{
    PORTA = 0x00; //스위치를 떼면 A포트 1번 핀에 L신호 출력
}
```

- **LED(Light Emitting Diode)란?**

  - 전류가 흐르면 빛을 발광하는 반도체 소자
  - Diode의 한 종류이기 때문에 방향성을 가지고 있음
  - 순방향이면 전류가 흐르고, 역방향이면 흐르지 않음
  - LED 빛을 내기 위해서 순방향의 전류를 흘려주어야 하며 LED 양단에 순방향 전압을 걸어주면 됨
  - 보통 LED는 전류 구동형으로 약 10~20mA 정도의 전류를 흘려줌. 전류의 양에 비례하여 밝기가 밝아지며, 전류가 너무 세면 파괴됨.
  - 5V의 전압에 LED와 470옴 정도의 저항을 직렬로 연결
  - 회로도상의 기호는 다음과 같이 나타냄

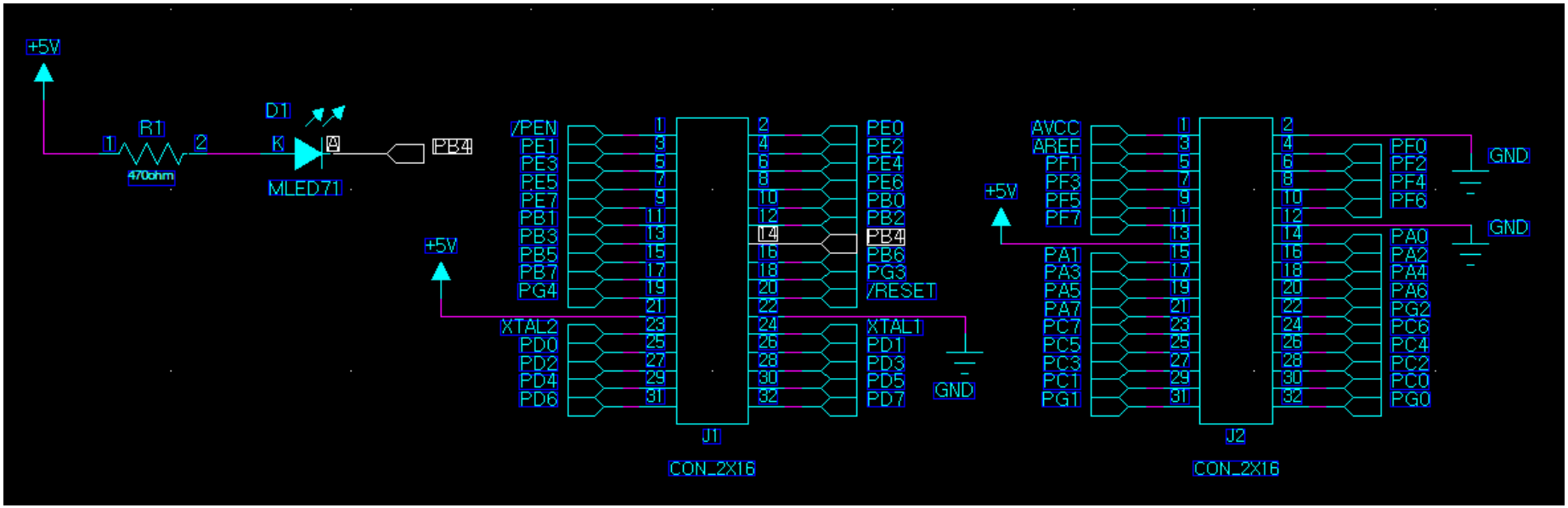


- **LED 연결 회로**



LED에 전류를 흐르게 하기 위해 다음과 같이 연결해 줌.
LED의 방향에 유의!
저항과 LED의 위치가 바뀌는 것은 상관없다.

- **LED 연결 회로 (Atmega128의 PB4에 연결)**



LED에 전류를 흐르게 하기 위해 다음과 같이 연결해 줌.
LED의 방향에 유의!
저항과 LED의 위치가 바뀌는 것은 상관없다.
PB4를 출력으로 설정한 후, L신호를 출력하면 LED 켜짐
H신호를 출력하면 LED 꺼짐

1강 General Purpose Input/Output (GPIO)

다음과 같은 코드를 작성하여 LED의 변화를 확인

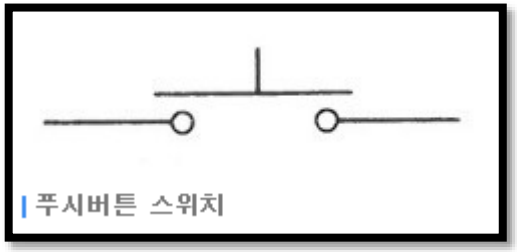
```
1  #include <mega128.h>
2  #include <delay.h>
3  main()
4  {
5      DDRB = 0x10;      //B포트 4번핀을 출력으로 설정, 나머지는 don't care
6      PORTB = 0x00;     //모든 핀 0으로 초기화.
7
8      while(1)
9      {
10         PORTB = 0x00;       //B포트 모든 핀 L신호 출력
11         delay_ms(1000);     //딜레이 1초
12         PORTB = 0x10;       //B포트 모든 핀 H신호 출력
13         delay_ms(1000);     //딜레이 1초
14     }
15 }
```

앞의 코드를 다음과 같이 바꾸면 어떤 차이가 생기는지 생각해보자.

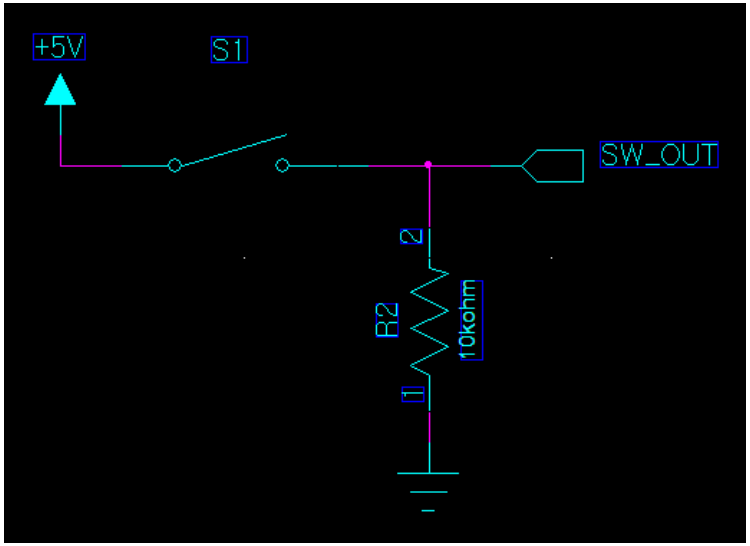
```
#include <mega128.h>
#include <delay.h>
main()
{
    DDRB = 0x10;      //B포트 4번핀을 출력으로 설정, 나머지는 don't care
    PORTB = 0x00;     //모든 핀 0으로 초기화.

    while(1)
    {
        PORTB = PORTB & ~(0x10);     //B포트 4번 핀 L신호 출력
        delay_ms(1000);              //딜레이 1초
        PORTB = PORTB | 0x10;        //B포트 4번 핀 H신호 출력
        delay_ms(1000);              //딜레이 1초
    }
}
```

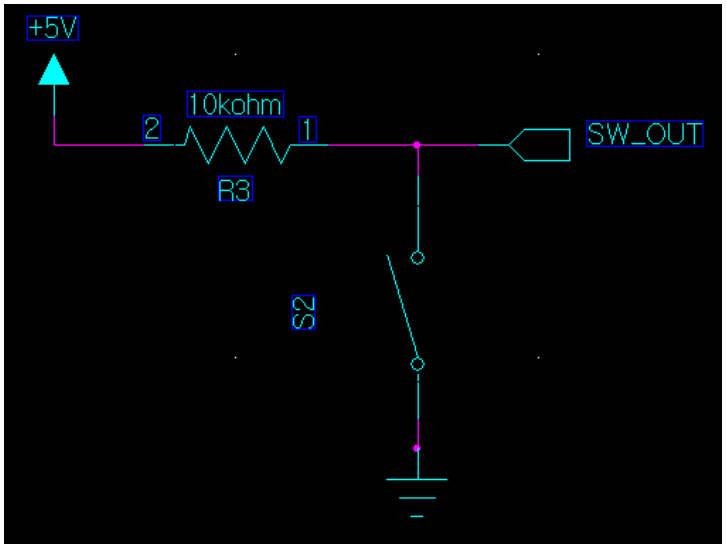
- **Switch란?**

  - 일련의 동작으로 전류의 흐름을 개폐하는 것
  - 스위치의 종류는 동작방식에 따라 기계식, 전자식 스위치가 있음
  - 기계식 스위치의 종류는 푸쉬버튼 스위치, 딥 스위치, 온오프 스위치 등 매우 많으며 본 강의에서는 푸쉬버튼 스위치를 사용
  - 푸쉬버튼 스위치는 2핀, 4핀 방식이 가장 많이 쓰이며, 버튼을 눌렀을 때 Close되고 뗐을 때 Open.
  - 회로도상의 기호는 다음과 같음



- **Switch 연결 회로**

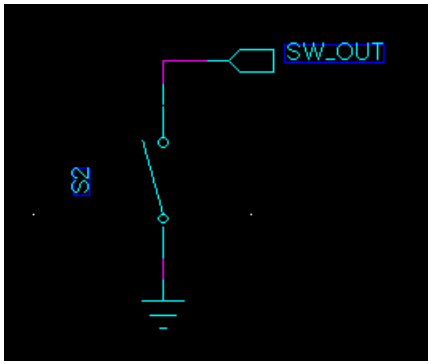




스위치를 연결해주기 위한 회로는 위의 그림과 같이 2가지 방법이 있음. 왼쪽의 그림은 Pull-down 이라고 하고, 오른쪽 그림은 Pull-up 이라고 한다.

- **풀업(Pull-up) 이란? (풀다운(Pull-down)은 반대 개념)**

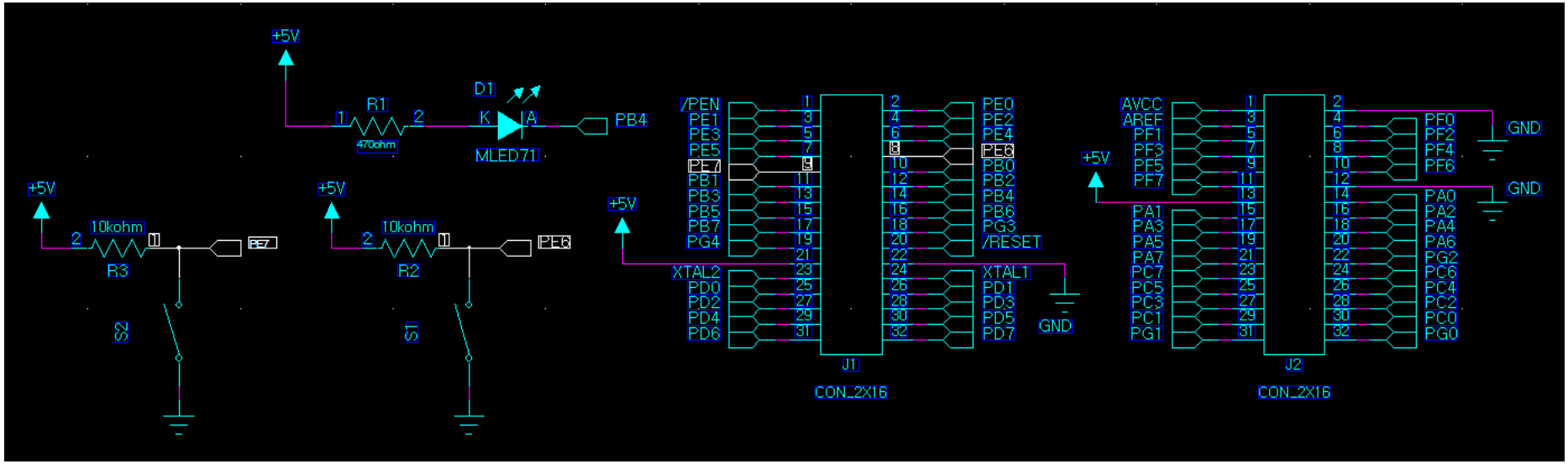
  - 위의 오른쪽 회로에서 풀업 저항이 없다면 어떤 일이 발생하는지 생각해보자



잘못된 연결의 예

- 오른쪽과 같은 회로에서 스위치가 눌려있다면 SW_OUT으로 L 신호가 출력될 것이다.
- 스위치가 눌려있지 않다면? 이 상태를 플로팅(Floating)되어 있다고 하며, 이 상태는 회로가 완전히 open된 상태이며 SW_OUT으로 어떤 신호가 나올지 모른다. 따라서 이를 방지하기 위해 +5V와 SW_OUT 사이에 1~10k옴 정도의 저항을 달아준다. 이렇게 하면 스위치를 누르지 않았을 때, H(+5V)신호가 출력된다. 이를 풀업 시킨다고 하며 이때 사용하는 저항을 풀업 저항이라고 한다.

- **스위치 연결 회로 (Atmega128의 PE6, PE7, PF6, PF7에 연결)**



위의 회로와 같이 PE6, PE7번에 풀업 방식으로 스위치를 연결함. 풀업 방식으로 회로를 구성하면, 해당 핀에 스위치를 누르지 않으면 H, 누르면 L의 신호가 입력.

1강 General Purpose Input/Output (GPIO)

다음과 같은 코드를 작성하여 스위치를 눌렀을 때, LED가 켜지는 프로그램을 작성

```
#include <mega128.h>
#include <delay.h>
main()
{
    DDRB = 0x10;      //B포트 4번 핀을 출력으로 설정, 나머지는 don't care
    PORTB = 0x00;     //모든 핀 0으로 초기화.
    DDRE = 0x00;      //E포트 6, 7번 핀에 스위치 연결. 풀업 되어있음.

    while(1)
    {
        if(PINE & 0x80) //E포트 7번 핀이 H일때. 즉, 스위치가 눌리지 않았을 때.
        {
            PORTB = PORTB & ~(0x10);    //B포트 4번 핀 L신호 출력. 즉, LED 켜짐
        }
        else //E포트 7번 핀이 L일때. 즉, 스위치를 눌렀을 때.
        {
            PORTB = PORTB | 0x10;       //B포트 4번 핀 H신호 출력. 즉, LED 꺼짐
        }
    }
}
```

1강 General Purpose Input/Output (GPIO)

문1) 1번 스위치(PE6)를 누르면 LED가 켜지고,
2번 스위치(PE7)를 누르면 LED가 꺼지는 프로그램 작성.

문2) 1번 스위치를 누르면 LED가 0.5초에 한번씩 깜빡
거리고, 2번 스위치를 누르면 LED가 꺼지는 프로그램 작성.