C - HW5

임베디드스쿨1기

Lv1과정

2020. 08. 28

박하늘

1. Atmega AVR 특징

1) AVR

-Atmel사에서 제작한 RISC 구조의 저전력 CMOS 8bit microcontroller

2) 특징

① 연산과 데이터 저장을 한번에 할 수 있는 하버드 구조로 되어 있다.
(동시에 프로그램 메모리와 데이터 메모리를 동시에 액세스 가능)
② 대부분의 명령이 단일클럭으로 동작 속도 개선함.
③ 명령어는 16bit 버스 폭의 하드웨어로 처리, 데이터는 8bit길이 기반으로 처리.
④ ISP(In-System Programming)기법이 가능한 플래시 메모리
⑤ 내장 메모리: 128KB 플래쉬 메모, 4KB 데이터(SRAM+EEPROM) + 스택(SRAM)
⑥ 부가 I/O장치
- 입출력(I/O): (A~F port) + G port = (8*6)+5 = 53개
- 디버깅: JTAG 인터페이스 지원
- 타이머/카운터: 8bit 타이머/카운터 2개, 16bit 타이머/카운터 2개 → 총 4개
- ADC: 8채널 10비트 ADC내장
- 통신: USART(시리얼통신) 2채널, TWI(I2C) 1채널, SPI 1채널
- 6개의 PWM 채널
- 별도의 오실레이터로 이루어진 watchdog timer, Analog comparator

⑦ 향상된 RISC구조
- 32개의 8비트 범용 레지스터 + 주변 컨트롤 (상태 또는 제어) 레지스터
- 133개의 명령어: 대부분 1사이클에 처리되는 명령어
- 16MHz의 입력 클록에서 최대 16MIPS로 동작
- 2사이클 내에 동작하는 곱셈기 내장
-완전한 정적 동작 지원

1. Atmega AVR 특징

⑧ 비휘발성 프로그램 메모리와 데이터 메모리
- 128KBytes의 ISP(In System Programming)이 가능한 플래시 메모리 (최대 10,000번 W가능)
  - 프로그램 실행 코드 저장 영역
  - 소프트웨어 보안을 위한 프로그램 메모리 잠금 기능
  - ISP를 위한 SPI 인터페이스 제공
- 4 bytes의 데이터 저장용 SRAM
  - 최대 64KBytes까지 외부 데이터 메모리 확장 기능
- 4 bytes의 EEPROM(최대 100,000번 W가능)
  - 비휘발성 데이터 저장 영역

⑨ 동적전압
- 2.7~5.5V: Atmega128L
- 4.5~5.5V: Atmega128
- 2.7~5.5V: Atmega128A

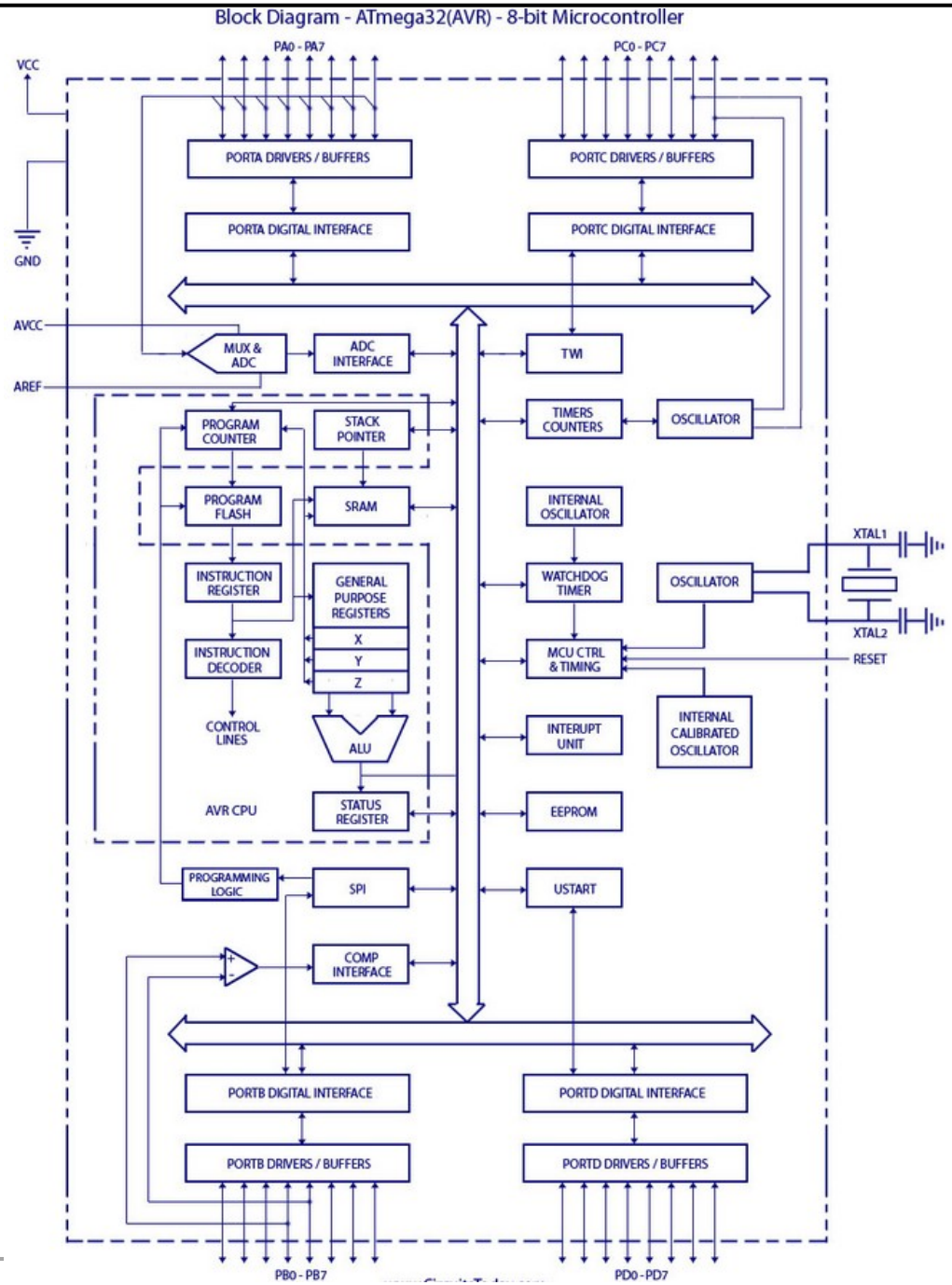
⑩ 동작속도
- 0~8MHz: Atmega128L
- 0~16MHz: Atmega128
- 0~16MHz: Atmega128A

⑪ 모두 64개의 핀으로 구성 : I/O핀 53개, 전원(VCC, AVCC, GND), 클록(XTAL1, XTAL2), 리셋(RESET), 프로그램관련 (PEN)

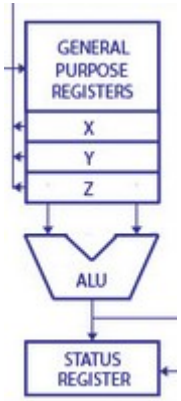
1. Atmega AVR 특징

⑫ 하버드 구조(Harvard architecture)
⑬ 메모리와 버스가 프로그램과 데이터로 분리
⑭ 하나의 명령이 처리되는 동안 다음 명령은 프로그램 메모리로 부터 프리-패치
⑮ 모든 클록 사이클에 명령이 처리되는 것이 가능

2. Atmega328p Block Diagram

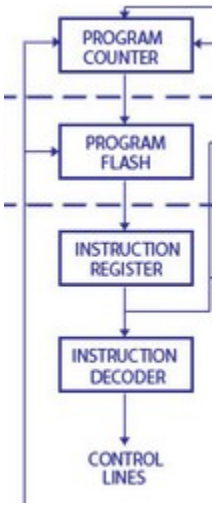


2. Atmega328p Block Diagram



1) General Purpose Registers (범용 레지스터)
   - 16bit 확장이 가능하고 C에서 접근이 불가한 영역
   - 주소가 없음

2) Status Register
   - ALU나 다른 제어회로 상태를 나타냄



3) Program Counter
   - 다음 수행될 명령어가 들어있는 주소를 기억

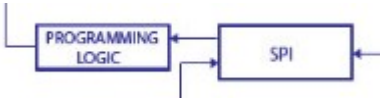
4) Program Flash
   - Atmega328p의 경우 128KB이고 여기에 실행코드가 들어 있음

5) Instruction Register
   - 프로그램 카운터가 가리키는 주소의 명령이 선택되어 명령레지스터에 들어감

6) Instruction Decoder
   - 명령레지스터에 있는 code를 해독하여 여러 장치로 제어 신호를 보냄

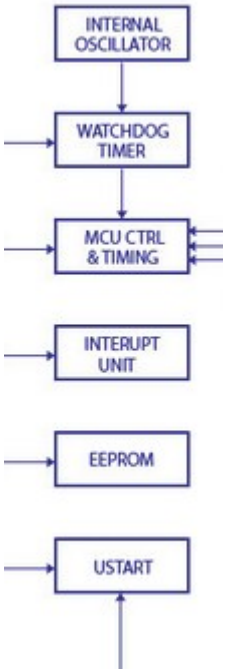
7) Control Lines
   - 제어선이 어디에 연결되어 있는지 확실히 알 수 없으나 거의 모든 장치에 연결 되어 있어, 프로그래머의 의도대로 동작함

2. Atmega328p Block Diagram



8) Programming Logic
- Flash에 w할 준비와 EEPROM에 데이터를 넣거나, 통상모드로 내장된 프로그램을 실행

9) SPI (Serial Peripheral Interface)
-소형 주변 장치간에 별도의 클록 및 데이터 라인으로 통신할 데이터 전송



10) Watchdog Timer
- 프로그램 흐름이 바뀌거나, 하드웨어적인 오류가 발생하였을때 CPU에 리셋시켜주는 기능

11) MCU CTRL
- 외부RAM을 연결할 것인지 느린 RAM과 연결시 Wait를 얼마나 줄 것인지 제어

12) Interupt Unit
- 실행할 명령어 실행에 대한 인터럽트가 있는지 확인

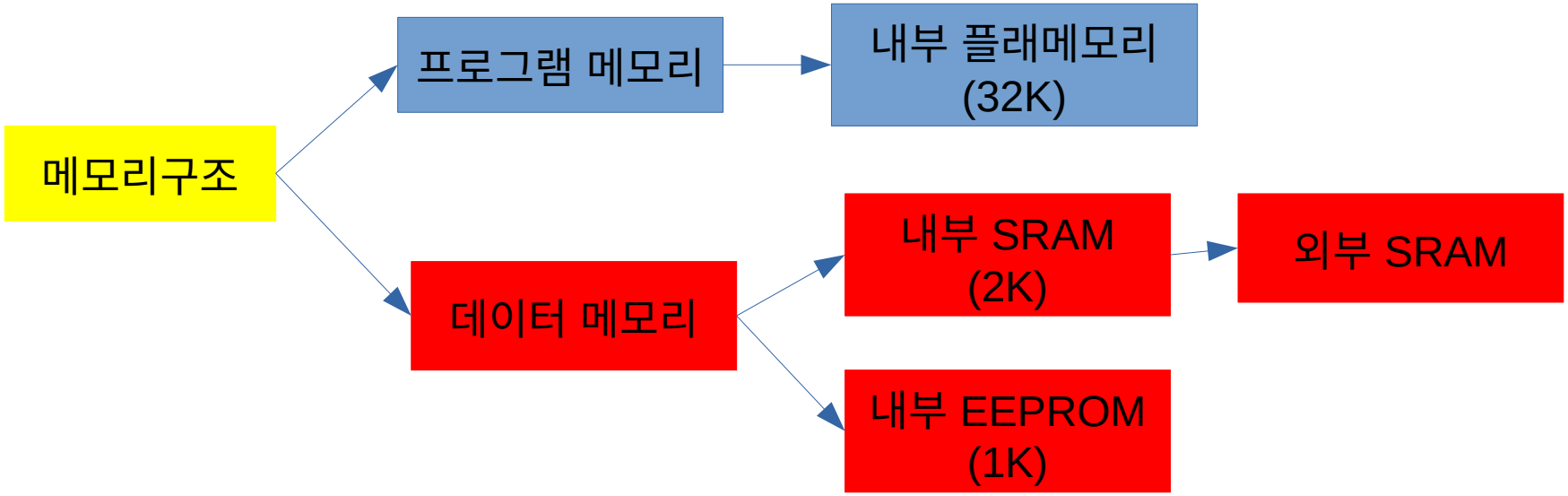
13) EEPROM
- 전원이 차단 되어도 데이터를 영구 저장
- EEPROM 내부 프로그래밍이 느림

14) USTART

3. Atmega 메모리 구조

1) Atmega 메모리 구조

- 기능적 - 프로그램 메모리, 데이터메모리
- 물리적 - 내부 플래시 메모리, 내부 SRAM, 내부 EEPROM, 외부SRAM



(1) 내부 플래시 메모리: Read-only, 비휘발성, 동적 데이터

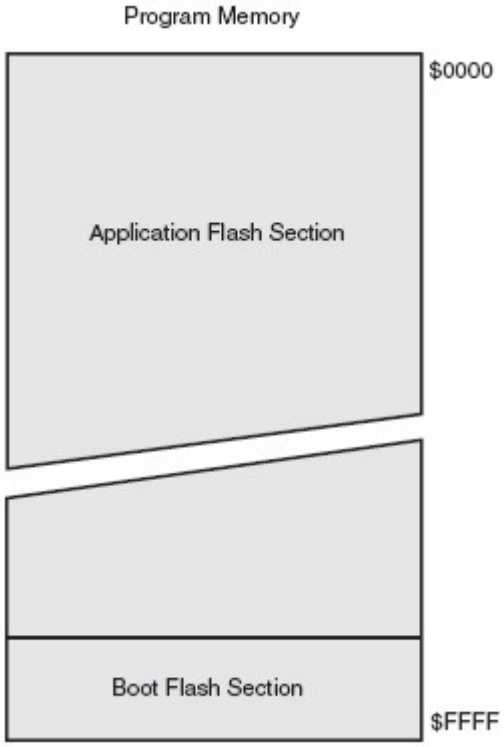
(2) SRAM: 프로그램에 필요한 여러가지 정보가 동적으로 생성됨, 휘발성

(3) EEPROM: 영구적으로 저장할 데이터 저장,

3. Atmega 메모리 구조

1) 플래시 프로그램 메모리 (Flash Program Memory)

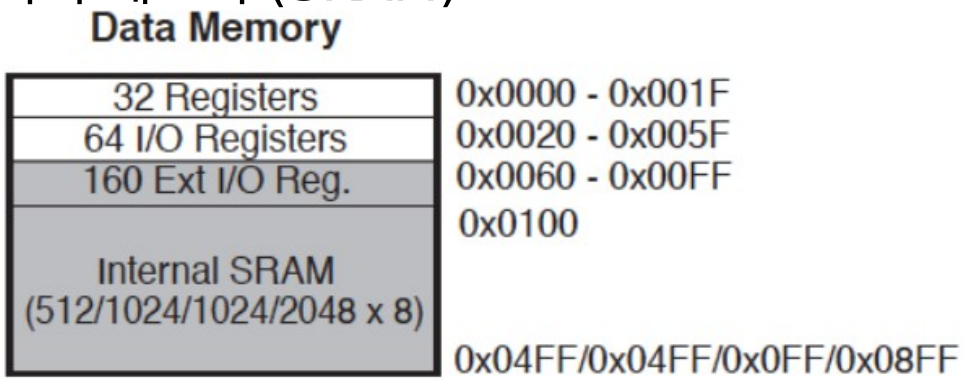
- Program Memory Map으로 PC(Program Counter) 레지스터의 크기가 16-bit로 접근 가능한 주소 공간은 64K의 크기인0x0000~0xFFFF로 됨. 플래쉬 메모리는 10,000번의 쓰기/삭제가 보장
- 2영역 boot flash, application flash로 나뉨
(boot flash 세션: 부트로더, application flash 세션: 프로그램 저장)
- 8비트로 구성되지만, 16비트로 명령어들이 저장



64KB X 16bit(2Byte) = 128 Byte

3. Atmega 메모리 구조

2) 데이터 메모리 (SRAM)



3) EEPROM 데이터 메모리(EEPROM Data Memory)

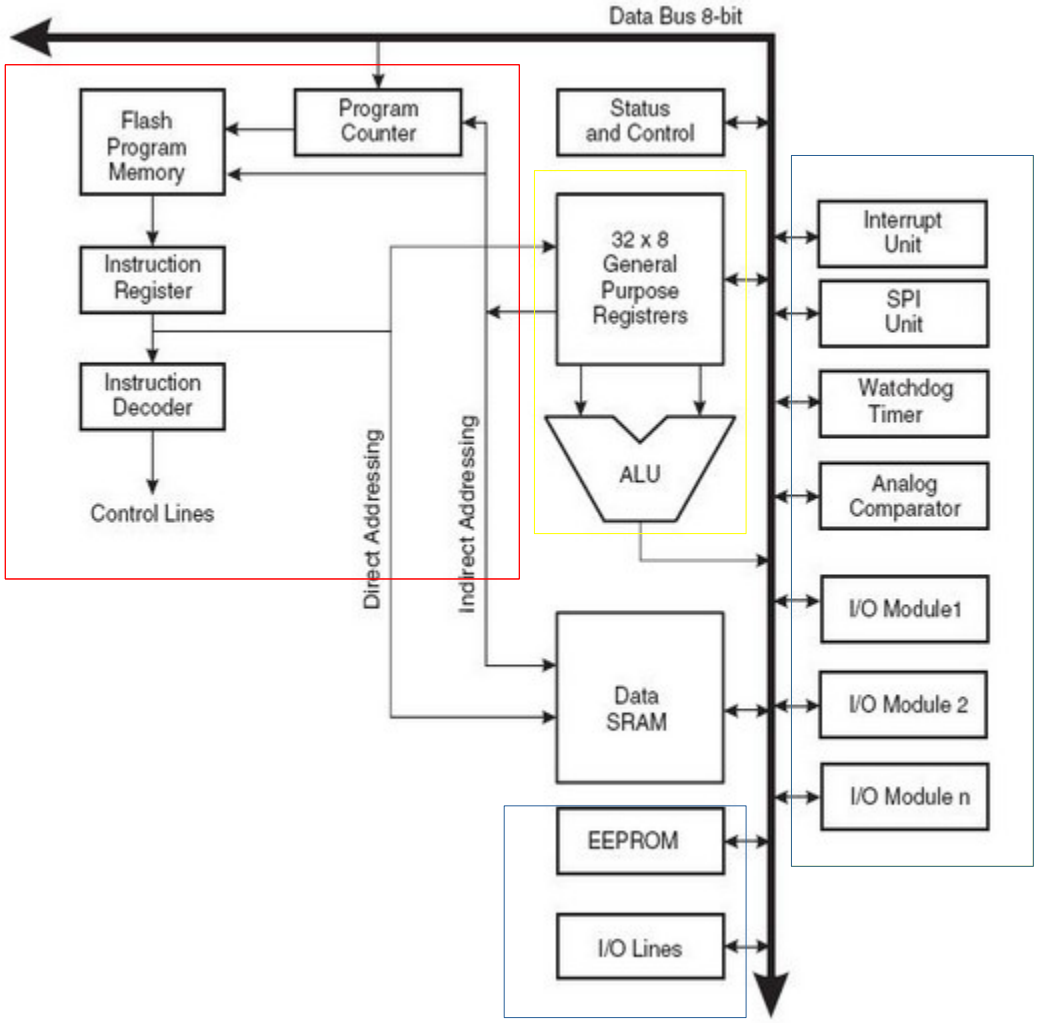
- 1k 공간을 작은 하드디스크처럼 사용할 수 있음.
- EEPROM의 수명이 최대 10만 write/erase 이므로 너무 자주 쓰지말 것.
- 다른 데이터 메모리 어드레스 영역과 별개의 장소에 위치됨
- EEARH(EEARL), EEDR, EECR 레지스터에 의해 1byte씩 엑세스 가능

4) 입출력 메모리(I/O Memory)

- 각종 I/O 모듈(타이머, 카운터, SPI, USART등) 제어 레지스터
- 기본 I/O 레지스터: 실제 존재하는 주소(0x0020~0x005F) 접근시에는 (0X0000~0X003F)로 접근하여 컴파일러가 주소에 맞게 처리
- 확장I/O 레지스터: 추가된I/O 제어레지스터(0X0060~0X00FF), 외부 데이터 메모리 LD/LDS/LDD or SD/STS/STD 명령어 사용

이름	용도	용량	설명	휘발여부
Flash Memory	프로그램 공간	32k	아두이노 스케치가 저장되는 곳	X
SRAM	정적 RAM	2k	아두이노 실행시 생성되는 변수 등이 저장되는 곳	O
EEPROM	비휘발성 기억	1k	오래 보관할 비휘발성 데이터를 저장하는 공간	X

4. AVR CPU CORE



: Instruction Fetch & Decode

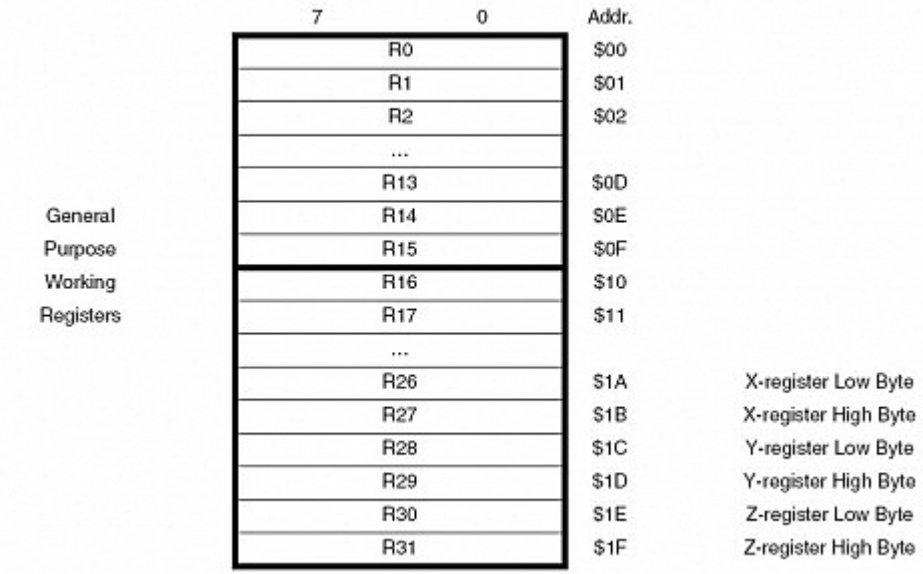
: ALU Instructions

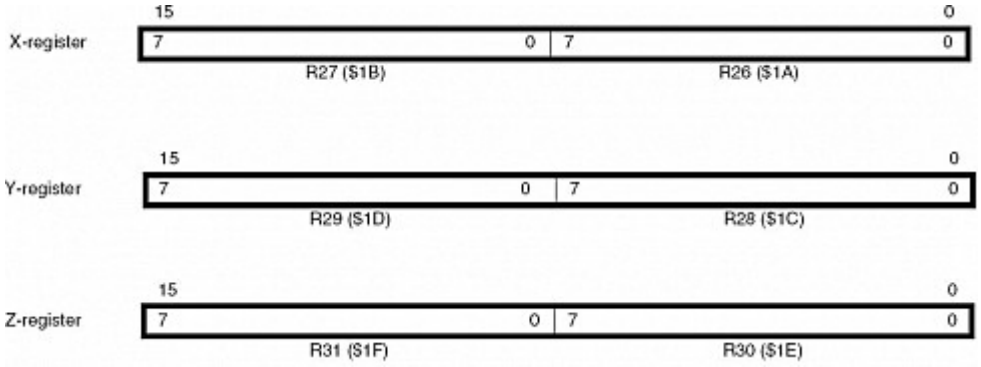
: I/O and special fuctions

4. AVR CPU Core 구조

1)범용 레지스터(General Purpose Register)

- 프로그램 수행 중에 중간 결과나 데이터를 일시적으로 저장하는 레지스터
- R0~R31까지의 32개의 8비트 범용 레지스터를 가짐





- R26 ~ R31 : 각각 2개가 결합하여 3개의 16 비트 레지스터 X,Y,Z레지스터

- X,Y,Z레지스터는 16비트 어드레스를 간접 지정하는 어드레스 포인터로 사용됨

- Z 레지스터는 LPM/ELPM/SPM 메모리 영역의 상수를 엑세스 함

4. AVR CPU Core 구조

2) 상태 레지스터 (Status Register: SREG)

- ALU가 가장 최근에 실행한 연산 명령의 결과 및 상태를 표시하는 레지스터

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
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	

I(7) : interrupt enable : 1= 전체 interrupt enable, 0 = 전체 인터럽트 disable

T(6) : 비트 복사 저장, T bit를 통하여 bit 전송

H(5): half carry flag : 0000 1000 + 0000 1000 → half carry 발생

S(4): sign bit : V 배타적 OR N (XOR을 의미)

V(3): overflow bit : 1000 0000 + 1000 0000 → overflow bit set

(오버플로는 음수+음수 = 양수 일때 발생)

N(2): negative bit : 연산결과가 음수임

Z(1): zero bit : 연산결과가 0임을 나타냄

C(0): carry bit : 연산결과 자리 수 올림(더하기), 혹은 빌림(빼기)

4. AVR CPU Core 구조

3) 스택 포인터 (Stack pointer)

- 서브루틴 호출이나 인터럽트 발생시 돌아올 복귀 주소 저장
- C언어의 지역 변수 저장, 어셈블리어에서 임시 데이터 저장
- 메모리의 상위 → 하위 번지순으로 저장

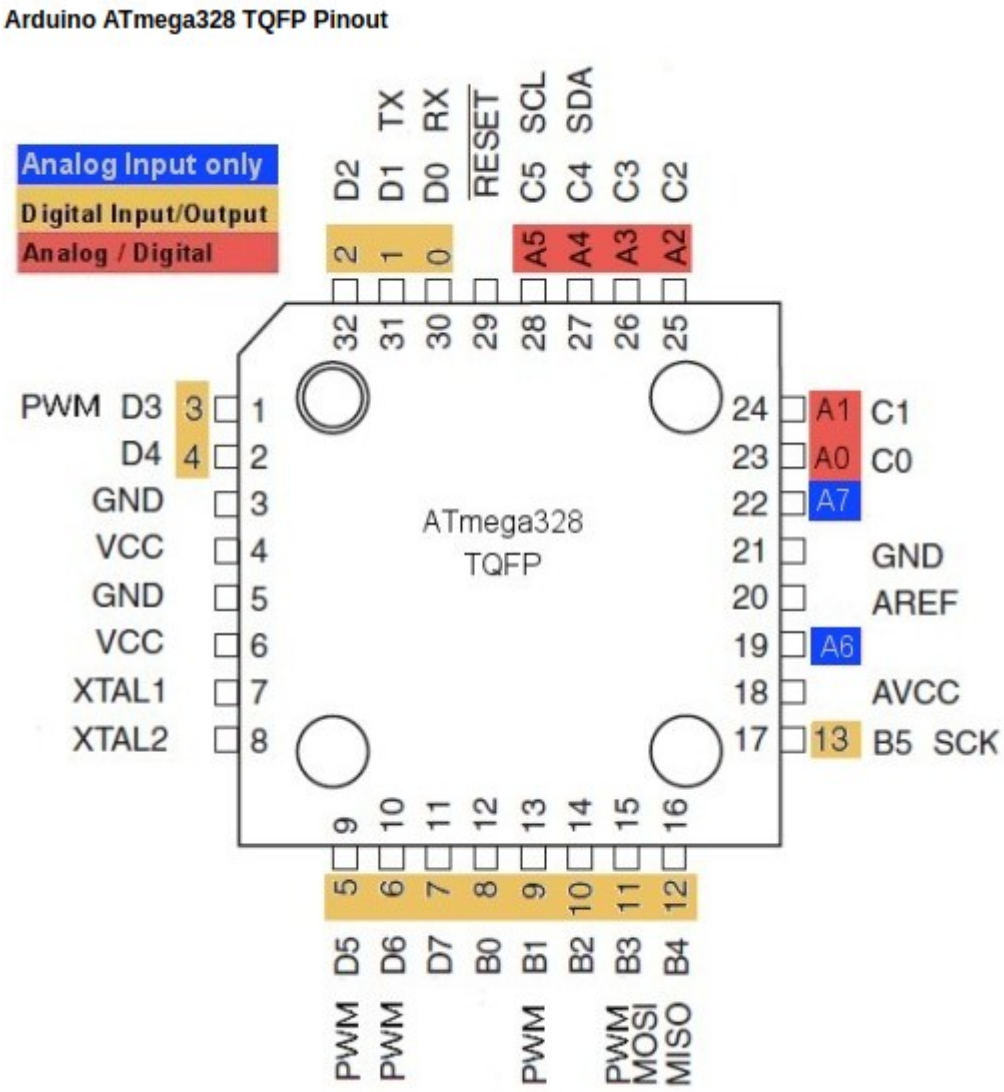
Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

- SPH: 상위 바이트 / SPL: 하위바이트
(스택 포인터는 항상 상단을 가르키는 16비트 레지스터, 데이터 저장이 가능한 스택의 번지)

- PUSH명령어: 1byte의 데이터를 sp가 가리키는 스텍 메모리 번지에 저장하고 sp값을 1만큼 감소

- POP명령어: 먼저 SP값을 1만큼 증가시킨 후 SP가 가리키는 스택 메모리 번지에서 1byte를 꺼냄

5. PIN Configuration



5. PIN Configuration

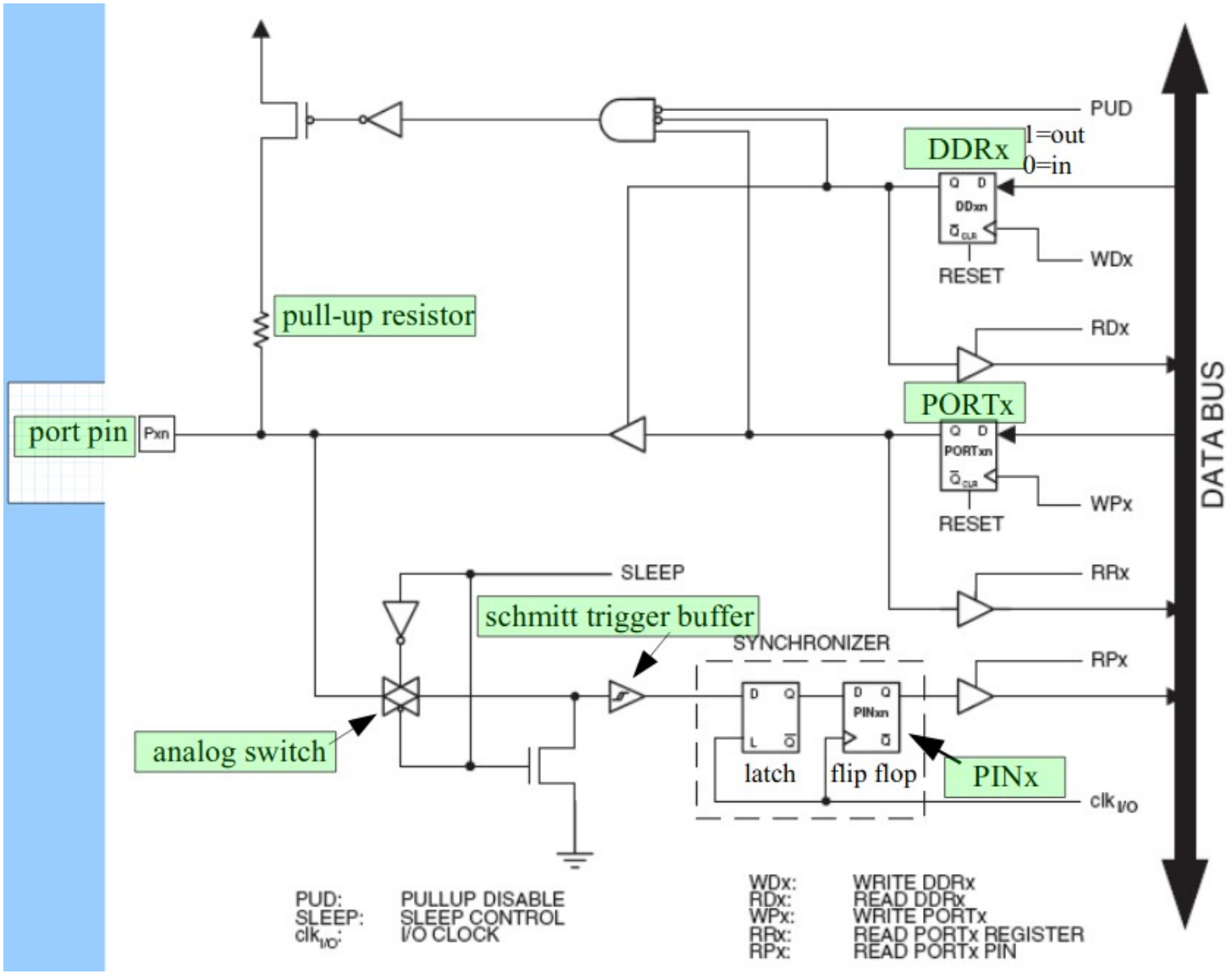
Pin No.	Pin name	Description	Secondary Function
1	PC6 (RESET)	Pin6 of PORTC	Pin by default is used as RESET pin. PC6 can only be used as I/O p when RSTDISBL Fuse is programmed.
2	PD0 (RXD)	Pin0 of PORTD	RXD (Data Input Pin for USART) USART Serial Communication Interface [Can be used for programming]
3	PD1 (TXD)	Pin1 of PORTD	TXD (Data Output Pin for USART) USART Serial Communication Interface [Can be used for programming] INT2(External Interrupt 2 Input)
4	PD2 (INT0)	Pin2 of PORTD	External Interrupt source 0
5	PD3 (INT1/OC2B)	Pin3 of PORTD	External Interrupt source1 OC2B(PWM - Timer/Counter2 Output Compare Match B Output)
6	PD4 (XCK/T0)	Pin4 of PORTD	T0(Timer0 External Counter Input) XCK (USART External Clock I/O)
7	VCC		Connected to positive voltage
8	GND		Connected to ground
9	PB6 (XTAL1/TOSC1)	Pin6 of PORTB	XTAL1 (Chip Clock Oscillator pin 1 or External clock input) TOSC1 (Timer Oscillator pin 1)
10	PB7 (XTAL2/TOSC2)	Pin7 of PORTB	XTAL2 (Chip Clock Oscillator pin 2) TOSC2 (Timer Oscillator pin 2)
11	PD5 (T1/OC0B)	Pin5 of PORTD	T1(Timer1 External Counter Input) OC0B(PWM - Timer/Counter0 Output Compare Match B Output)
12	PD6 (AIN0/OC0A)	Pin6 of PORTD	AIN0(Analog Comparator Positive I/P) OC0A(PWM - Timer/Counter0 Output Compare Match A Output)
13	PD7 (AIN1)	Pin7 of PORTD	AIN1(Analog Comparator Negative I/P)
14	PB0 (ICP1/CLKO)	Pin0 of PORTB	ICP1(Timer/Counter1 Input Capture Pin) CLKO (Divided System Clock. The divided system clock can be output on the PB0 pin)
15	PB1 (OC1A)	Pin1 of PORTB	OC1A (Timer/Counter1 Output Compare Match A Output)
16	PB2 (SS/OC1B)	Pin2 of PORTB	SS (SPI Slave Select Input). This pin is low when controller acts as slave. [Serial Peripheral Interface (SPI) for programming] OC1B (Timer/Counter1 Output Compare Match B Output)

5. PIN Configuration

17	PB3 (MOSI/OC2A)	Pin3 of PORTB	MOSI (Master Output Slave Input). When controller acts as slave, the data is received by this pin. [Serial Peripheral Interface (SPI) for programming] OC2 (Timer/Counter2 Output Compare Match Output)
18	PB4 (MISO)	Pin4 of PORTB	MISO (Master Input Slave Output). When controller acts as slave, the data is sent to master by this controller through this pin. [Serial Peripheral Interface (SPI) for programming]
19	PB5 (SCK)	Pin5 of PORTB	SCK (SPI Bus Serial Clock). This is the clock shared between this controller and other system for accurate data transfer. [Serial Peripheral Interface (SPI) for programming]
20	AVCC		Power for Internal ADC Converter
21	AREF		Analog Reference Pin for ADC
22	GND		GROUND
23	PC0 (ADC0)	Pin0 of PORTC	ADC0 (ADC Input Channel 0)
24	PC1 (ADC1)	Pin1 of PORTC	ADC1 (ADC Input Channel 1)
25	PC2 (ADC2)	Pin2 of PORTC	ADC2 (ADC Input Channel 2)
26	PC3 (ADC3)	Pin3 of PORTC	ADC3 (ADC Input Channel 3)
27	PC4 (ADC4/SDA)	Pin4 of PORTC	ADC4 (ADC Input Channel 4) SDA (Two-wire Serial Bus Data Input/output Line)
28	PC5 (ADC5/SCL)	Pin5 of PORTC	ADC5 (ADC Input Channel 5) SCL (Two-wire Serial Bus Clock Line)

6. I/O Port

1) AVR Port architecture



6. I/O Port

2-1). DDRx – PORT X Data Direction Register
- 8비트를 조정하여 1이면 출력, 0이면 입력 (기본값 0 셋팅)

7	6	5	4	3	2	1	0	
DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

2-2). PORTx – PORT X Data Register
- DDRx로 출력을 사용하면, 이 레지스터에서 포트 값을 출력할 수 있다.
- 비트 별로 수정하려면 PORTx.n = 1을 사용하여 수정함

7	6	5	4	3	2	1	0	
PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

2-3). PINx – PORT X Input Pins Address
- DDRx로 입력을 사용하면, 이 레지스터에서 포트 값을 받을 수 있다.
- PORTx와 같이 비트별로 받을 경우 PINx.n을 사용하여 수정

7	6	5	4	3	2	1	0	
PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
R	R	R	R	R	R	R	R	
N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

2-4). SFIOR – Special Function IO Register
- PUD: Pull-up disable
-

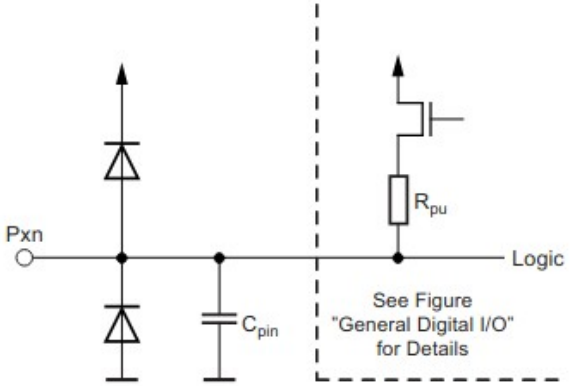
7	6	5	4	3	2	1	0	
TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
R/W	R	R	R	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

6. I/O Port

3) I/O Port input 구조

-protection diodes

-programmable pull-up resistor



4)핀의 기능 설정

: DDRx레지스터는 PORTx의 핀의 방향을 결정

(DDxn: 0 → Pxn 입력 / Ddxn: 1 → Pxn출력

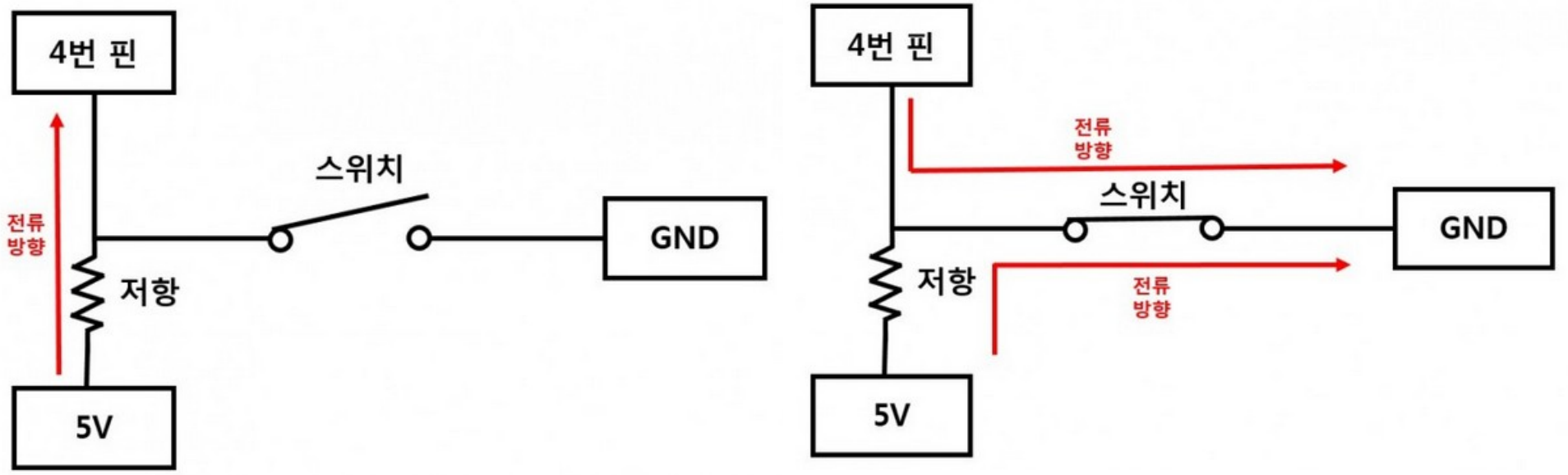
DDxn	PORTxn	I/O	풀업 저항	설 명
0	0	입력	×	hi-Z
0	1	입력	¡	풀업 저항에 의해 입력 '1'
1	0	출력	×	출력 '0'
1	1	출력	×	출력 '1'

7. Pull-up & Pull-down

1) 풀업(PULL-UP)

-플로팅 상태일때의 값을 끌어올린다는 의미
- 저항을 VCC단자에 달아줌(풀다운일경우 GND)

(스위치 OFF/ON)
스위치 OFF: 전류는 GND가 아닌 입출력핀으로 흐름, 입출력핀 1(HIGH)의 값 읽음
스위치 ON: GND는 모든 전류가 도착하는 전압이 가장 낮은 지점이기 때문에 모든 전류는 GND 방향으로 흐름 (입출력핀에 흐르는 전류가 없기 때문에 0(LOW)가 출력)

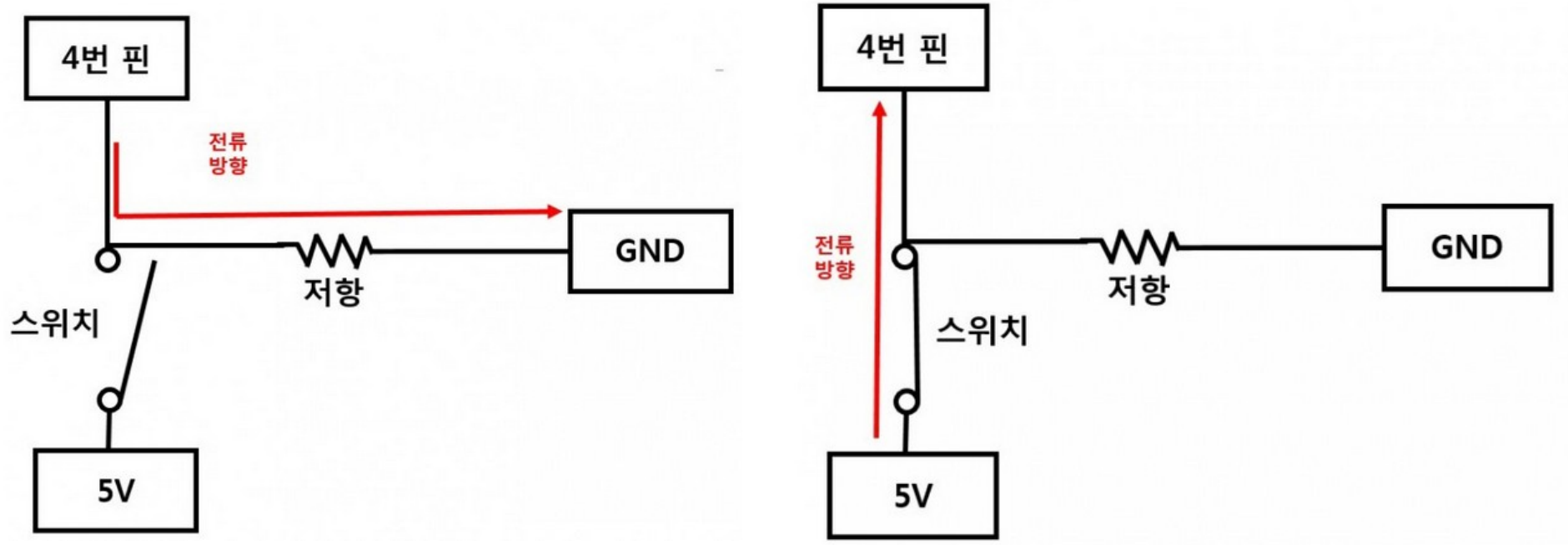


7. Pull-up & Pull-down

2) 풀다운(PULL-DOWN)

-플로팅 상태의 값을 다운시켜버림
-저항이 VCC가 아닌 GND쪽에 달려있음 (풀업 반대)

(스위치 OFF/ON)
스위치 OFF: VCC와 회로는 단절, 입출력핀에서 흐르는 전류는 GND로 향해 0(LOW) 출력
스위치 ON: VCC에서 흐르는 전류로 입출력핀 1(HIGH)이 출력



8. Arduino 실습1

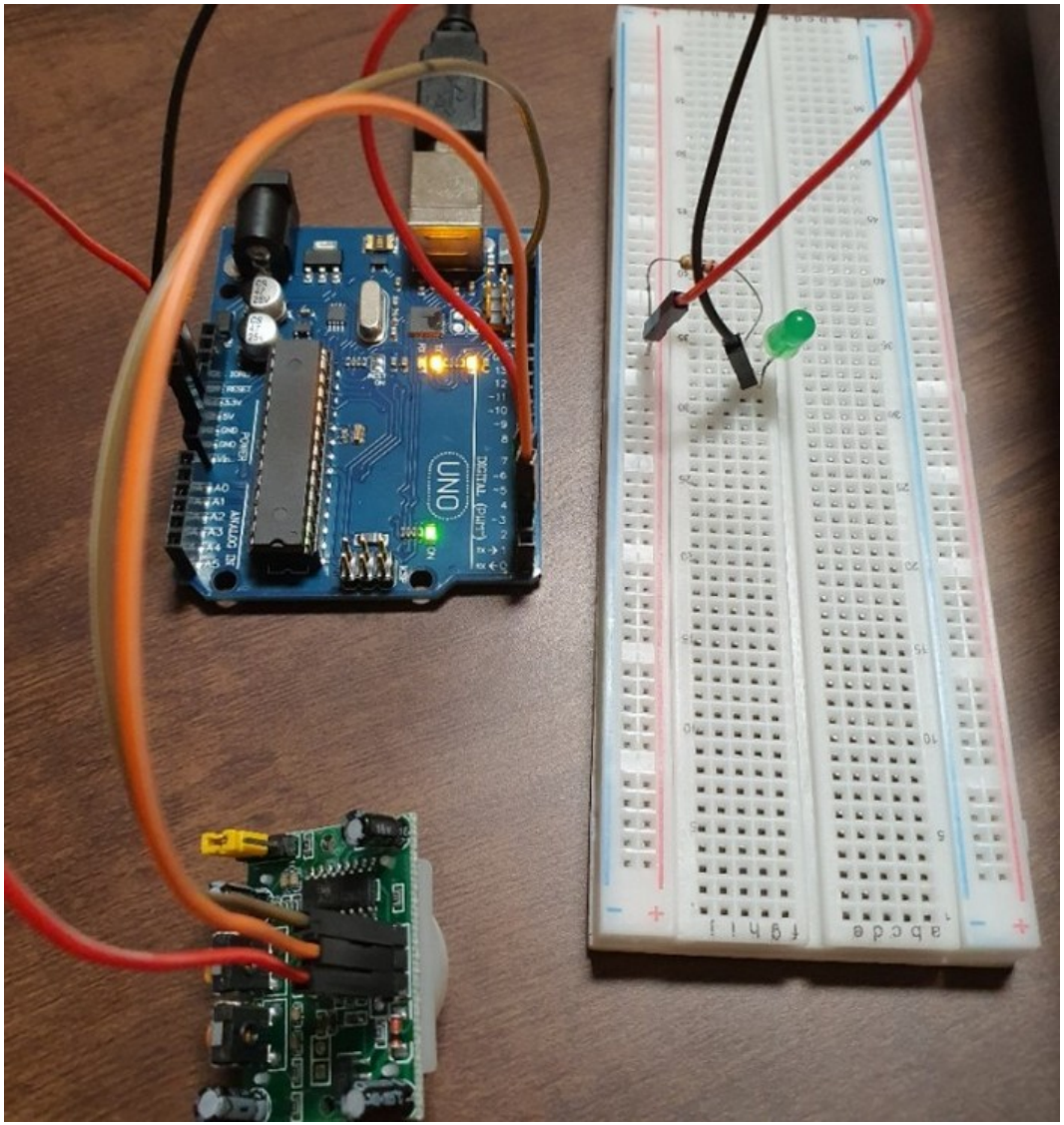
1) PIR Sensor ON/OFF

PIR_SENSOR_ON_OFF

```
int val = 0;

void setup() {
  // put your setup code here, to run once:
  pinMode(2,INPUT);  //2번 핀 입력
  pinMode(3,OUTPUT);  //3번 핀 출력
  Serial.begin(9600); //시리얼 통신 시작, 통신속도 9600
}

void loop() {
  // put your main code here, to run repeatedly:
  val = digitalRead(2); //변수 val에 digital 2번핀 입력
  if (val == HIGH)  //val이 HIGH면,
  {
    digitalWrite(3,HIGH); //3번핀 HIGH 출력
  }
  else
  {
    digitalWrite(3,LOW); //3번핀 LOW 출력
  }
  Serial.println(val);
}
```



9. Arduino 실습2

2) HC_Distance_Sensor_with PIR

HC_Distance_sensor_with_PIR

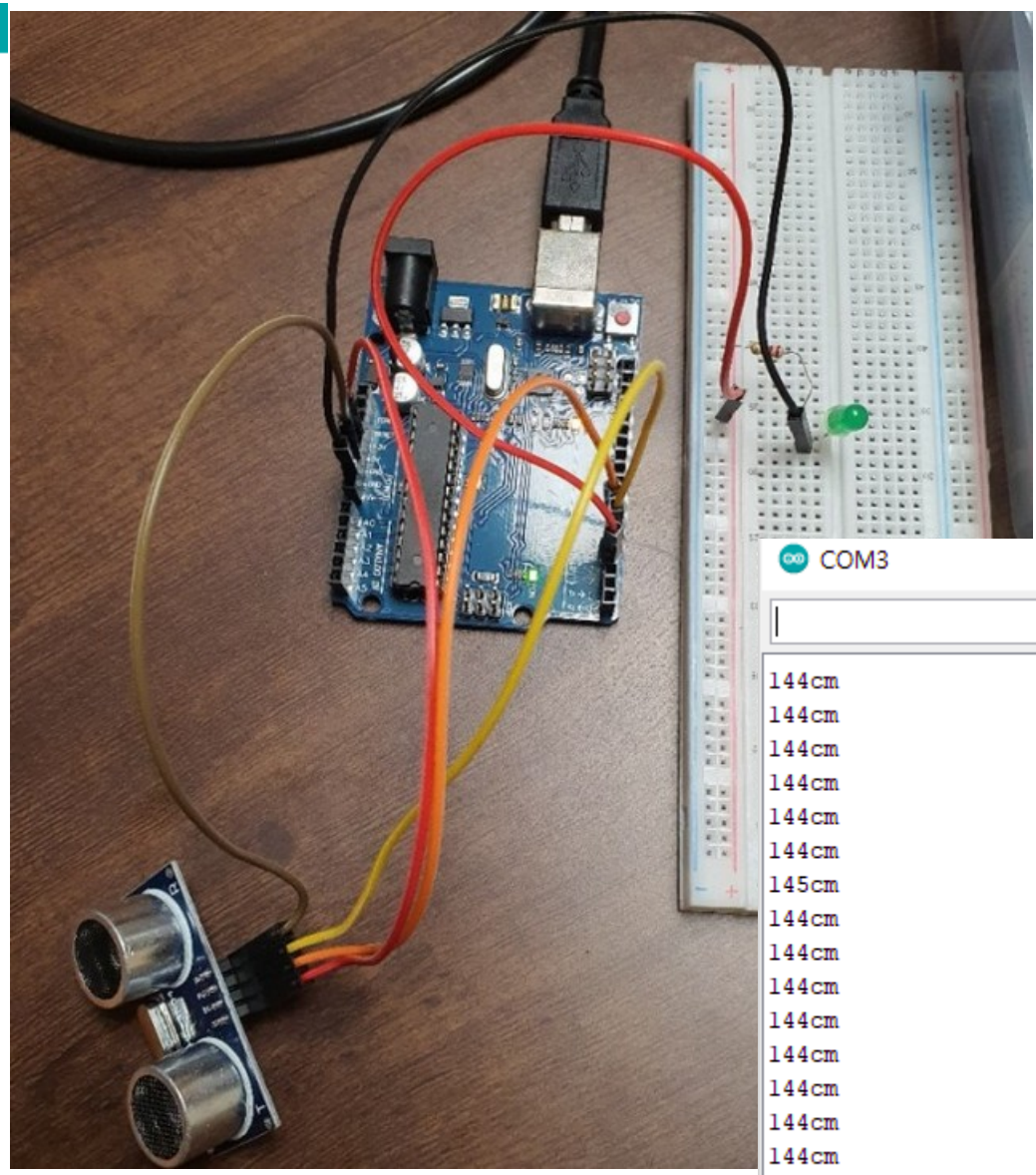
```
int trig = 6;
int echo = 5;

void setup() {
  // put your setup code here, to run once:
  pinMode(3,INPUT);  //3번 핀 출력
  Serial.begin(9600); //시리얼 통신 시작, 통신속도 9600
  pinMode(trig,OUTPUT);  //6번 핀 출력
  pinMode(echo,INPUT);  //5번 핀 입력
  }

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);  //10마이크로초 동안 대기
  digitalWrite(trig, LOW);

  //20Mhz = 340m/s, m -> cm (
  int distance = pulseIn(echo, HIGH) * 340 / 3 / 10000;
  Serial.print(distance);
  Serial.println("cm");
  delay(100);

  if (distance < 20)
  {
    digitalWrite(3,HIGH); //3번핀 HIGH 출력
  }
  else
  {
    digitalWrite(3,LOW); //3번핀 LOW 출력
  }
}
```



COM3

```
144cm
144cm
144cm
144cm
144cm
144cm
145cm
144cm
144cm
144cm
144cm
144cm
144cm
144cm
144cm
144cm
```

☑ 자동 스크롤 ☐ 타임스탬프 표시

C source. Quicksort

```c
#include <stdio.h>
#include <stdlib.h>   //srand
#include <time.h>    //time

#define SIZE 10

void output(int* ary, int len);
void swap(int*, int*);
void quicksort(int*, int, int);

int nLandNum[SIZE] = { 0, };     //생성된 번호를 저장할 변수

void rand_num(void)
{
        srand((unsigned)time(NULL));    //srand로 초기화

        for (int i = 0; i < SIZE; i++)    //번호가 10개 생성될 때까지 돈다
        {
                int nTemp = rand() % 100;
                nLandNum[i] = nTemp + 1;
        }
}
int main(void)
{
        int start = 0;
        rand_num();         //랜덤값 생성하는 함수

        output(nLandNum, SIZE);      //sorting전
        quicksort(nLandNum, start, SIZE - 1);  //quicksorting 함수
        output(nLandNum, SIZE);      //sorting후

        return 0;
}
```

```
43 44 66 55 96 90 50 27 67 41
27 41 43 44 50 55 66 67 90 96
```

```c
void quicksort(int* ary, int start, int end)
{
        int pivot = ary[end];      //pivot 기준값 설정, 시작&끝 인덱스 정의
        int left = start;
        int right = end;

        if (left >= right)       //예외처리
                return;

        while (left < right)      //왼쪽같이 오른쪽값보다 작을때만 loop순환
        {
                while (ary[left] < pivot && left < right)
                {
                        left++;        //기준값보다 크면 오른쪽으로 이동
                }
                while (ary[right] >= pivot && left < right)
                {
                        right--;       //기준값보다 작으면 오른쪽으로 이동
                }
                swap(&ary[left], &ary[right]);   //왼쪽값과 오른쪽값이 역전되면 swap진행
        }

        swap(&ary[right], &ary[end]);    //pivot값과 왼쪽or오른쪽 값 swap
        quicksort(ary, start, right - 1);   //왼쪽 재귀순환 진행
        quicksort(ary, right + 1, end);    //오른쪽 재귀순환 진행

}
void output(int* arr, int len)     //printf해줌
{
        int i = 0;
        for (i = 0; i < len; i++)
        {
             printf("%d ", arr[i]);
        }
         printf("\n");
}
void swap(int* num1, int* num2)     //swap함수
{
        int temp;

        temp = *num1;
        *num1 = *num2;
        *num2 = temp;
}
```